Block2 Group A2

liume102@student.liu.se
hanxi898@student.liu.se
xiali125@student.liu.se

2024-11-26

Contents

0.1 Introduction

This is the lab for bock2 in the Machine Learning. In this lab, contains the following tasks:1. ENSEMBLE METHODS.2.MIXTURE MODELS.3. Theory.

0.2 Assignment 1:ENSEMBLE METHODS

#build test data set

First of all, we fixed the data of the test set to avoid the result error caused by the difference of the test set in the subsequent calculation process.

```
set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
testdata <- cbind(x1, x2)
y1 <- as.numeric(x1 < x2)
testlabels <- as.factor(y1)
y2 <- as.numeric(x1 < 0.5)
testlabels2 <- as.factor(y2)
y3 <- as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
testlabels3 <- as.factor(y3)
```

#build the train dataset for 1000 times with the size of 100 Create 1000 training data sets of size 100, learn a random forest from each data set, and compute the classification error in the same test data set of size 1000. Report results for when the random forest has 1, 10 and 100 trees.

We fixed the data of the train set.

```r
set.seed(123)
train_data_list <- lapply(1:1000, function(i) {
  x3 <- runif(100)
  x4 <- runif(100)
  trdata <- cbind(x3, x4)
  colnames(trdata) <- c("x1", "x2")
  list(trdata = trdata)
})
```

#Conditon1 Compute the misclassification error in the same test dataset of size 1000. Report results for when the random forest has 1, 10 and 100 trees.

```r
error_rate_1<- list(
  number1 = rep(0,1000),
  number2 = rep(0,1000),
  number3 = rep(0,1000)
)
mean_error_1<- c()
var_error_1<- c()
for (i in 1:1000) {
  trdata <- train_data_list[[i]]$trdata
  y <- as.numeric(trdata[, 1] < trdata[, 2])
  trlabels <- as.factor(y)

  #build the models
  rf_model1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
  rf_model2 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
  rf_model3 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)

  #predictions and error rates
  predictions1<- predict(rf_model1,testdata)
  error_rate_1$number1[i] <- mean(predictions1 != testlabels)
  predictions2<- predict(rf_model2,testdata)
  error_rate_1$number2[i] <- mean(predictions2 != testlabels)
  predictions3<- predict(rf_model3,testdata)
  error_rate_1$number3[i] <- mean(predictions3 != testlabels)

}
```

#compute the mean and variance of error rates

```r
mean_error_1[1]<- mean(error_rate_1$number1)
mean_error_1[2]<- mean(error_rate_1$number2)
mean_error_1[3]<- mean(error_rate_1$number3)

var_error_1[1] <- var(error_rate_1$number1)
var_error_1[2] <- var(error_rate_1$number2)
var_error_1[3] <- var(error_rate_1$number3)
```

Then, we can change the condition,and the calculation is the same as before.

#repeat for the conditions (x1<0.5) for 1,10,100 trees, the results are summarized in mean_error_2 and var_error_2.

#repeat for the conditions ((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5)) and node size 12 for 1,10,100 trees, the results are summarized in mean_error_3 and var_error_3.

#summary the results

```
result<- list(
  mean_error_1 = mean_error_1,
  mean_error_2 = mean_error_2,
  mean_error_3 = mean_error_3,

  var_error_1 = var_error_1,
  var_error_2 = var_error_2,
  var_error_3 = var_error_3
)
print(result)
```

```
## $mean_error_1
## [1] 0.212314 0.134672 0.110031
##
## $mean_error_2
## [1] 0.094541 0.015954 0.006759
##
## $mean_error_3
## [1] 0.244643 0.119890 0.076860
##
## $var_error_1
## [1] 0.0034664178 0.0009542587 0.0008674575
##
## $var_error_2
## [1] 1.895466e-02 7.167787e-04 6.893385e-05
##
## $var_error_3
## [1] 0.013417381 0.002800795 0.001326289
```

0.2.0.1 Questions:

0.2.0.2 What happens with the mean error rate when the number of trees in the random forest grows? Why? As the number of trees in the random forest increases, the mean error rate decreases. This happens because: a random forest combines multiple decision trees. As more trees are added, the predictions are averaged, reducing the variance of the model. With more trees, the random forest effectively samples a larger variety of subsets from the training data. This increases the likelihood that the model captures the true underlying patterns.

0.2.0.3 The third data set represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance. Despite being more complex, the random forest performs better on the third data set with sufficient trees because: the third data set is more complex and may contain multiple non-linear boundaries or classes that are more difficult to distinguish. In this case, a single tree may not be enough to capture all the complex patterns, and increasing the number of trees can provide more decision rules, thereby improving the performance of the model. More trees can improve the generalization ability of the model in an integrated way.

0.3 Assignment 2:MIXTURE MODELS

In this assignment,we will chose different values of M,and compare their fit results.The E and M steps are calculated as follows:

Learn the GMM

Data: Unlabeled training data $\mathcal{T} = \{\mathbf{x}_i\}_{i=1}^n$, number of clusters M.
Result: Gaussian mixture model

1 Initialize $\widehat{\boldsymbol{\theta}} = \{\widehat{\pi}_m, \widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{\Sigma}}_m\}_{m=1}^M$
2 **repeat**
3 For each $\mathbf{x}_i$ in $\{\mathbf{x}_i\}_{i=1}^n$, compute the prediction $p(y \mid \mathbf{x}_i, \widehat{\boldsymbol{\theta}})$ according to (10.5) using the current parameter estimates $\widehat{\boldsymbol{\theta}}$.
4 Update the parameter estimates $\widehat{\boldsymbol{\theta}} \leftarrow \{\widehat{\pi}_m, \widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{\Sigma}}_m\}_{m=1}^M$ according to (10.16)
5 **until** *convergence*

Predict as QDA, Method 10.1

$$p(y = m \mid \mathbf{x}_\star) = \frac{\widehat{\pi}_m \mathcal{N}\left(\mathbf{x}_\star \mid \widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{\Sigma}}_m\right)}{\sum_{j=1}^M \widehat{\pi}_j \mathcal{N}\left(\mathbf{x}_\star \mid \widehat{\boldsymbol{\mu}}_j, \widehat{\boldsymbol{\Sigma}}_j\right)}. \tag{10.5}$$

$$\widehat{\pi}_m = \frac{1}{n} \sum_{i=1}^n w_i(m), \tag{10.16a}$$

$$\widehat{\boldsymbol{\mu}}_m = \frac{1}{\sum_{i=1}^n w_i(m)} \sum_{i=1}^n w_i(m) \mathbf{x}_i, \tag{10.16b}$$

$$\widehat{\boldsymbol{\Sigma}}_m = \frac{1}{\sum_{i=1}^n w_i(m)} \sum_{i=1}^n w_i(m)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_m)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_m)^\intercal. \tag{10.16c}$$

where $w_i(m) = p(y_i = m \mid \mathbf{x}_i, \widehat{\boldsymbol{\theta}})$

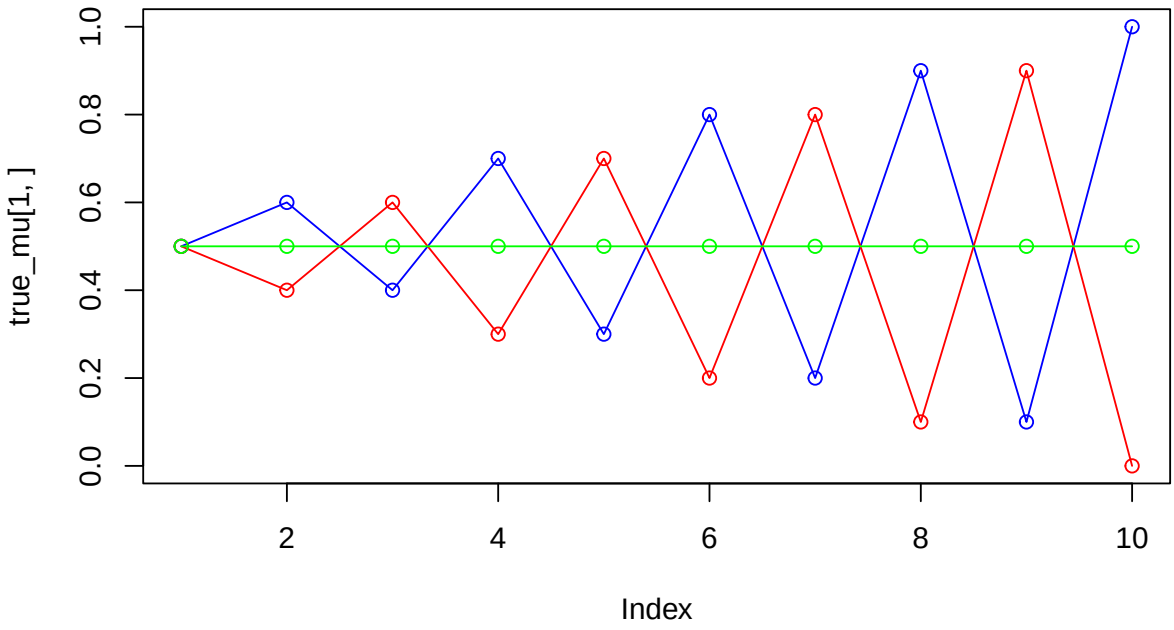
Method 10.3: Unsupervised learning of the GMM

- The algorithm above is also known as EM algorithm: Step 3 is called Expectation, and step 4 is called Maximization.
- It converges to a **local maximum** of the likelihood function of the training data

$$\log L(\Theta \mid X) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{M} \pi_k \cdot p(x$$

and the formula for calculating the log-likelihood value is as follows:

```
# Let's take K=3 for example
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
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
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
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])
  }
}
M=3 # number of clusters,it can be changed to 2 or 4
w <- matrix(nrow=n, ncol=M) # weights
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
```

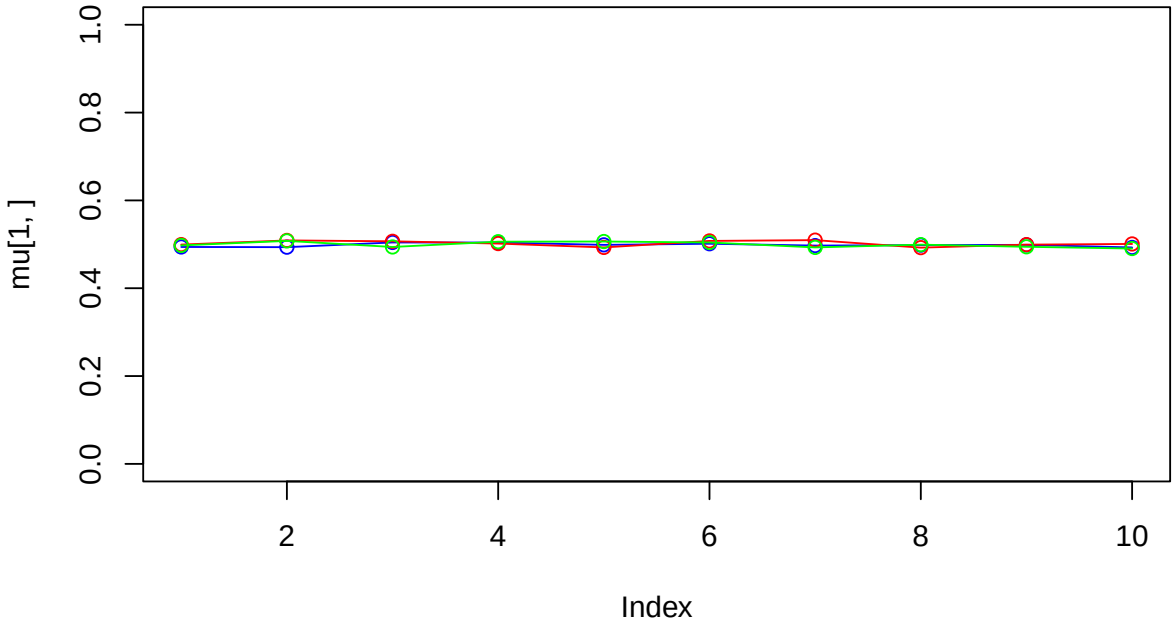
```
## [1] 0.3326090 0.3336558 0.3337352
```

```
mu
```

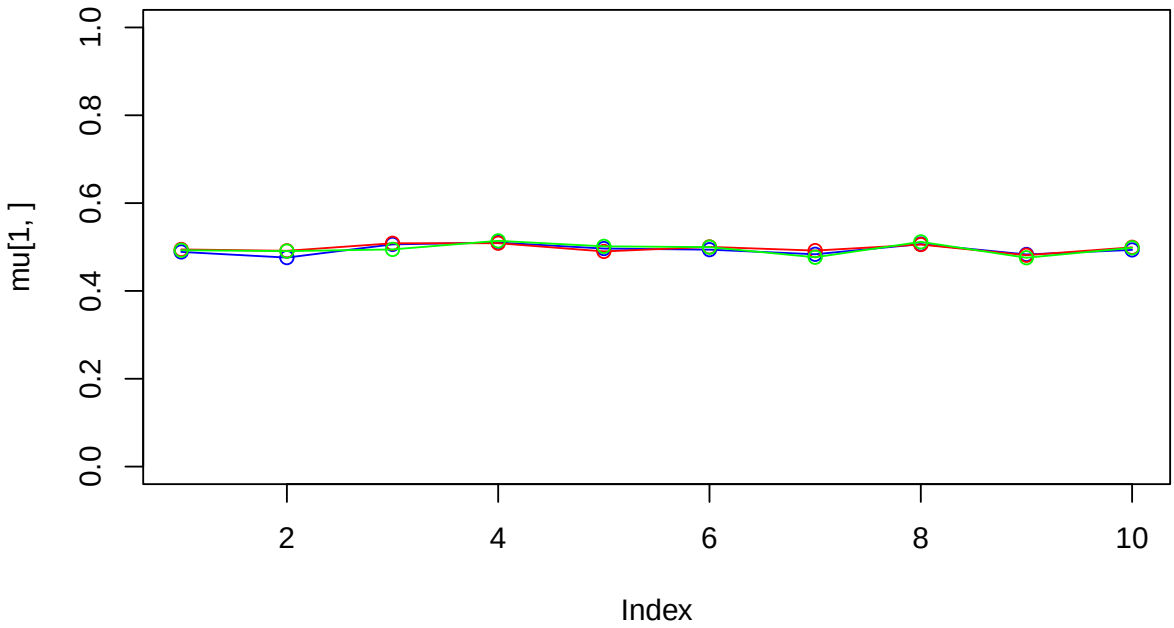
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4939877 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754 0.4971036
## [2,] 0.4993719 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146 0.5095075
```

```
## [3,] 0.4975302 0.5077926 0.4939841 0.5059821 0.5063490 0.5041462 0.4929400
##           [,8]      [,9]     [,10]
## [1,] 0.4982144 0.4987654 0.4929075
## [2,] 0.4924574 0.4992470 0.5008651
## [3,] 0.4992362 0.4943482 0.4903974
```

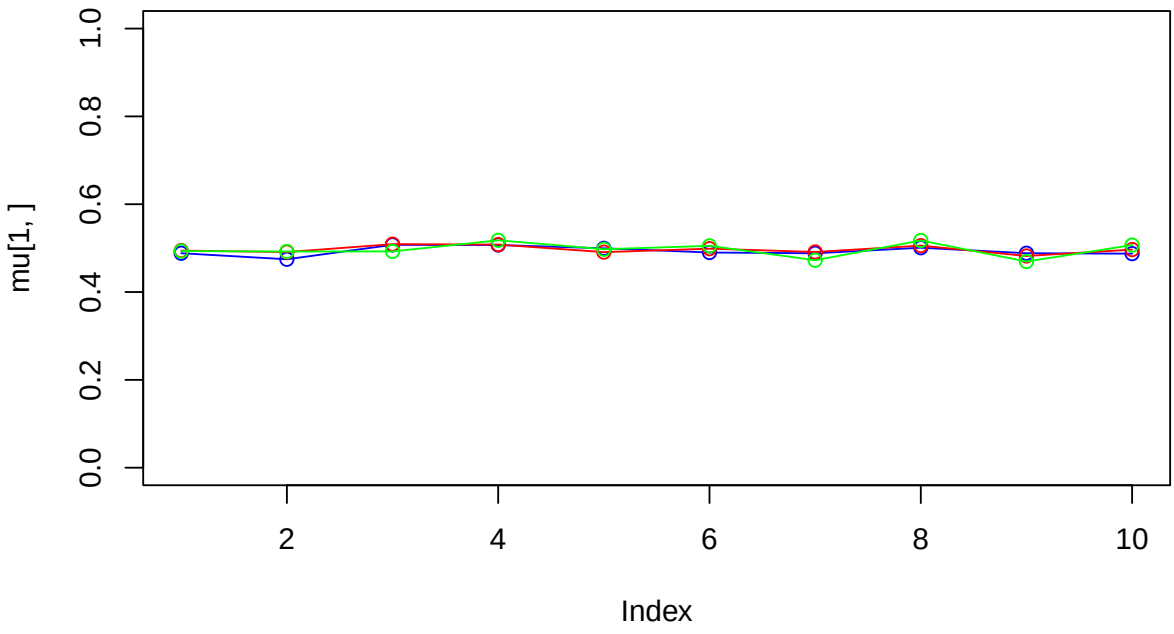
```
for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")#This line should be ignored if K=2
  #points(mu[4,], type="o", col="yellow")#This line should be ignored if K=2 or k=3
  Sys.sleep(0.5)

  # E-step: Computation of the weights
  for (i in 1:n) {
    for (m in 1:M) {
      numerator <- pi[m] * prod(mu[m, ]^x[i, ] * (1 - mu[m, ])^(1 - x[i, ]))
      denominator <- sum(sapply(1:M, function(k) {
        pi[k] * prod(mu[k, ]^x[i, ] * (1 - mu[k, ])^(1 - x[i, ]))
      }))
      w[i, m] <- numerator / denominator
    }
  }
#Log likelihood computation.
  llik[it] <- sum(sapply(1:n, function(i) {
    log(sum(sapply(1:M, function(m) {
      pi[m] * prod(mu[m, ]^x[i, ] * (1 - mu[m, ])^(1 - x[i, ]))
    })))
  }))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()

  # Stop if the log likelihood has not changed significantly
  if (it > 1 && abs(llik[it] - llik[it - 1]) < min_change) {
    cat("Converged at iteration", it, "\n")
    break
  }

  #M-step: ML parameter estimation from the data and weights
  for (m in 1:M) {
    pi[m] <- sum(w[, m]) / n
    for (d in 1:D) {
      mu[m, d] <- sum(w[, m] * x[, d]) / sum(w[, m])
    }
  }
}
```



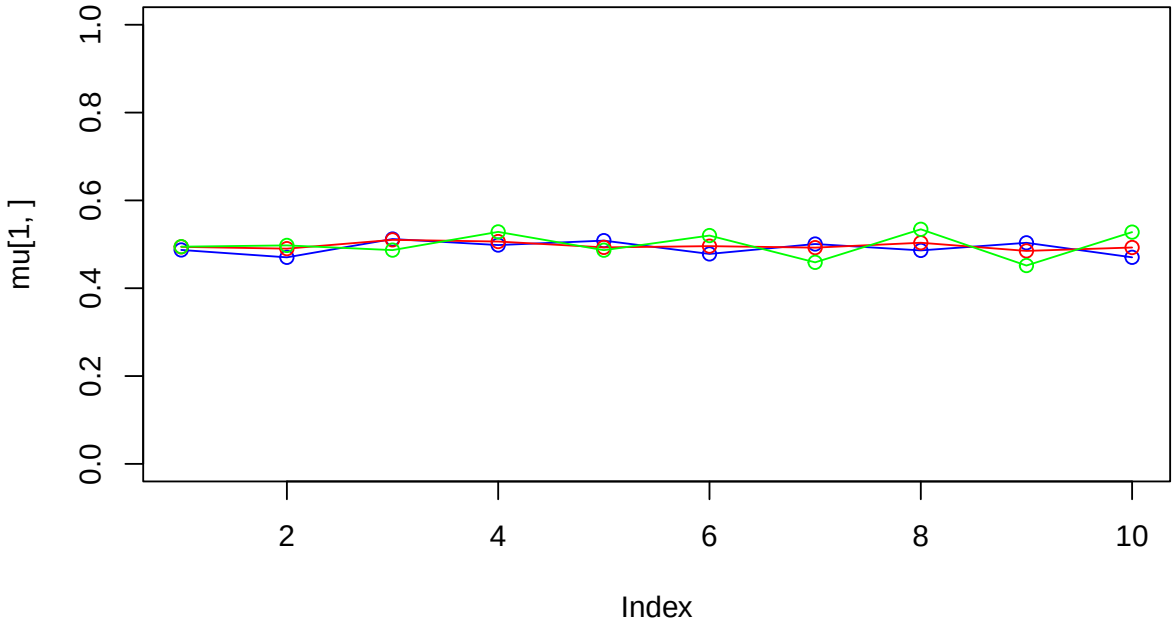

```
## iteration:  1 log likelihood:  -6931.482
```



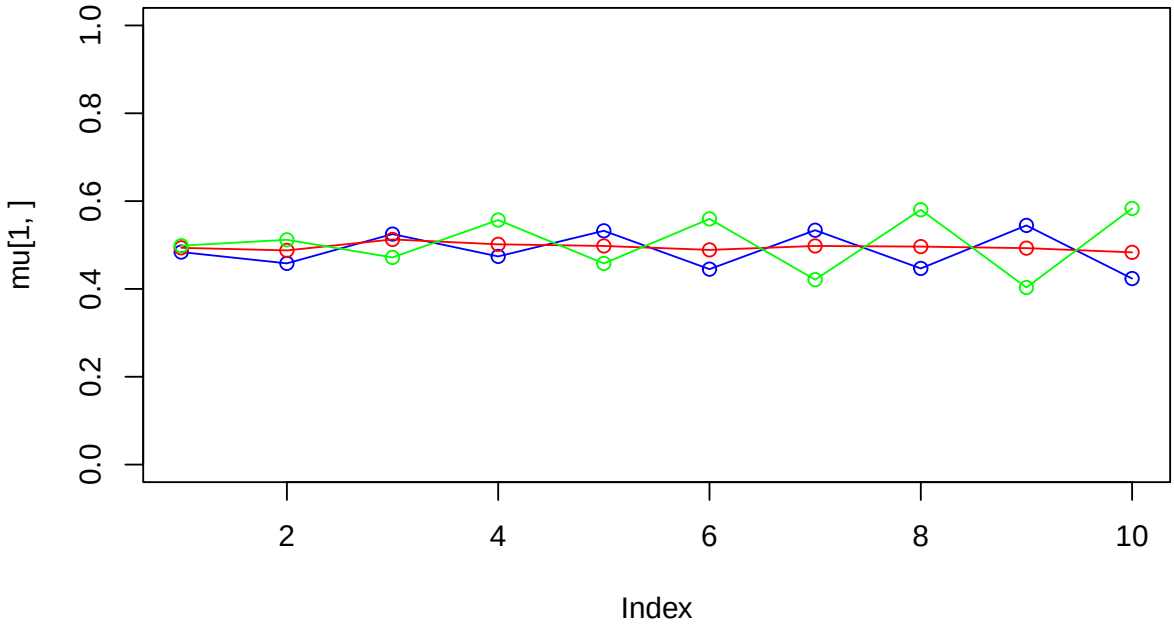
```
## iteration:  2 log likelihood:  -6929.074
```



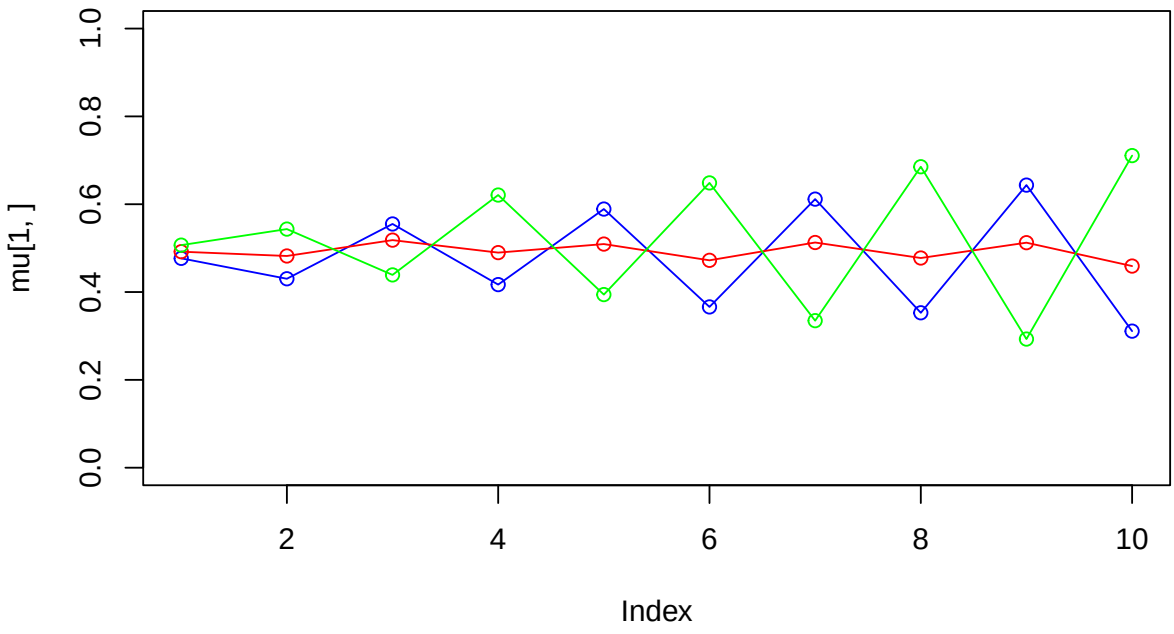

```
## iteration:  3 log likelihood:  -6928.081
```



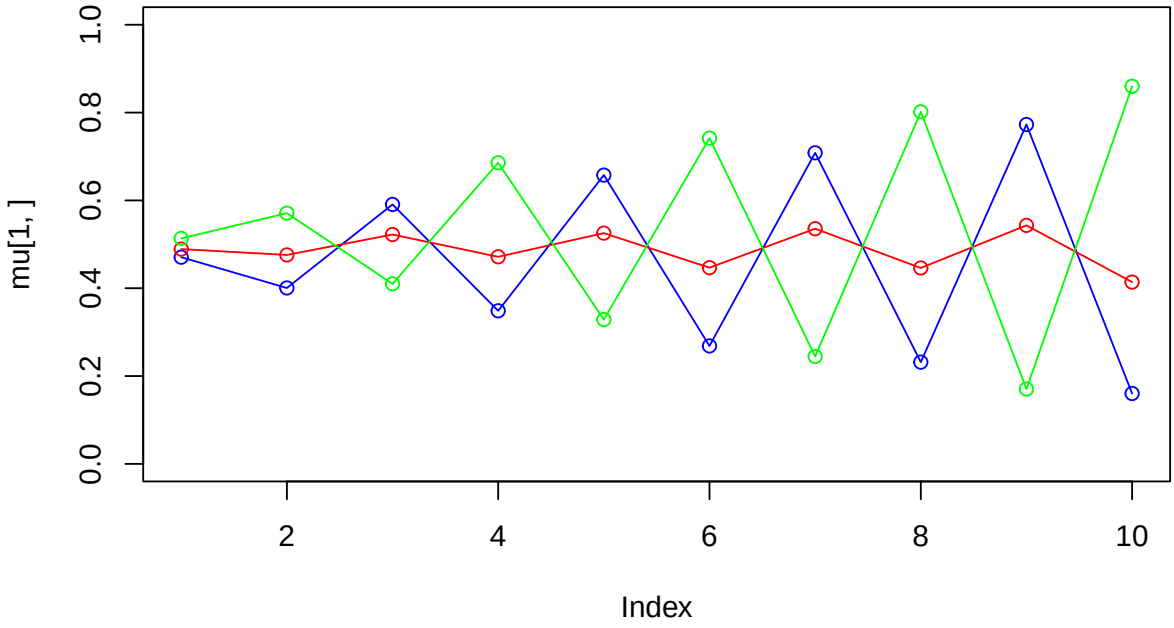

```
## iteration:  4 log likelihood:  -6920.57
```



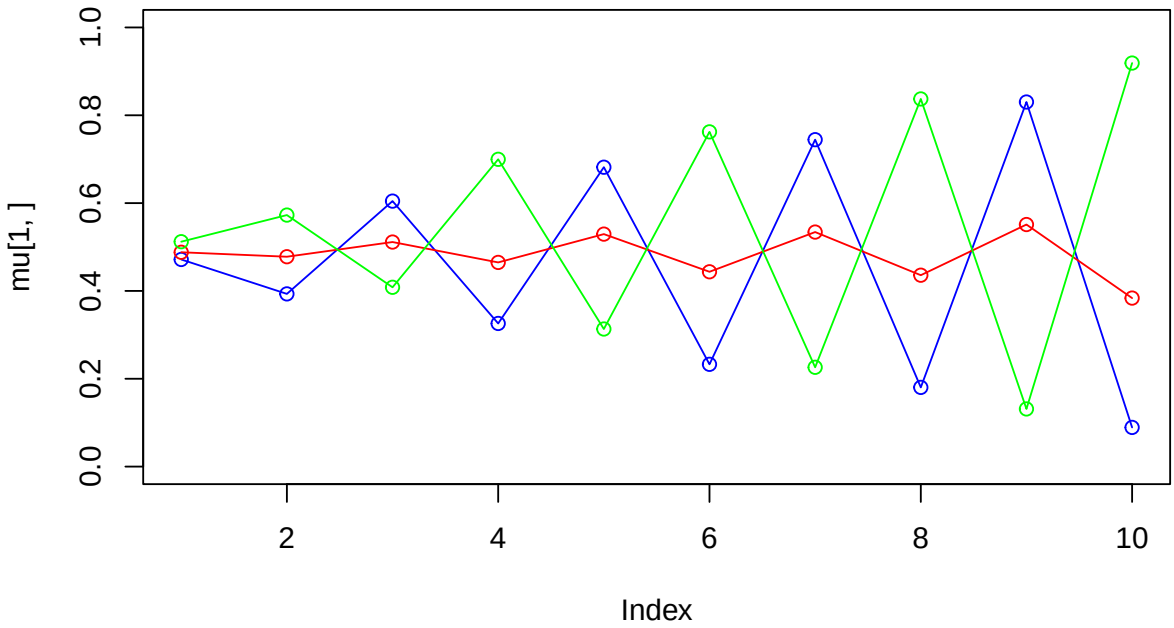
```
## iteration:  5 log likelihood:  -6868.29
```



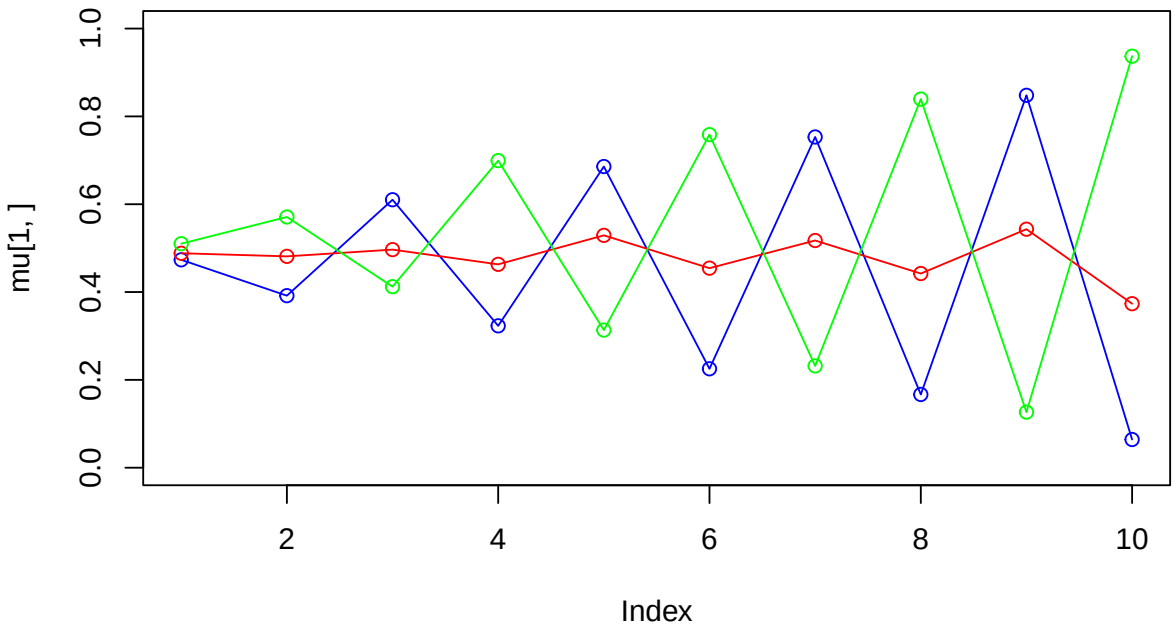

```
## iteration:  6 log likelihood:  -6646.505
```



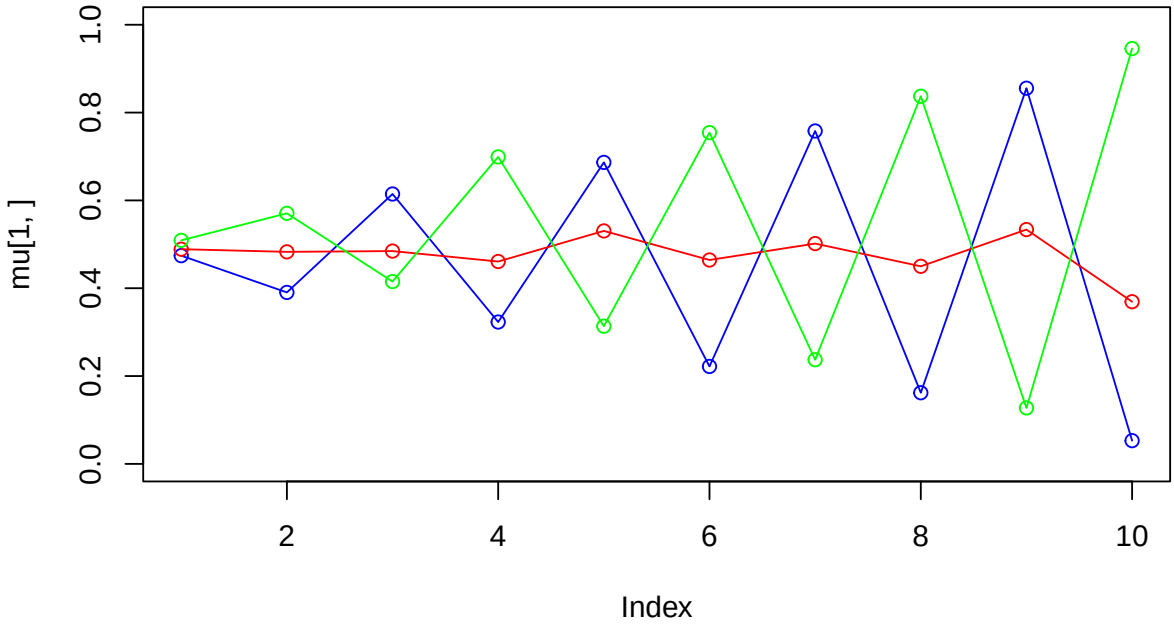

```
## iteration:  7 log likelihood:  -6403.476
```



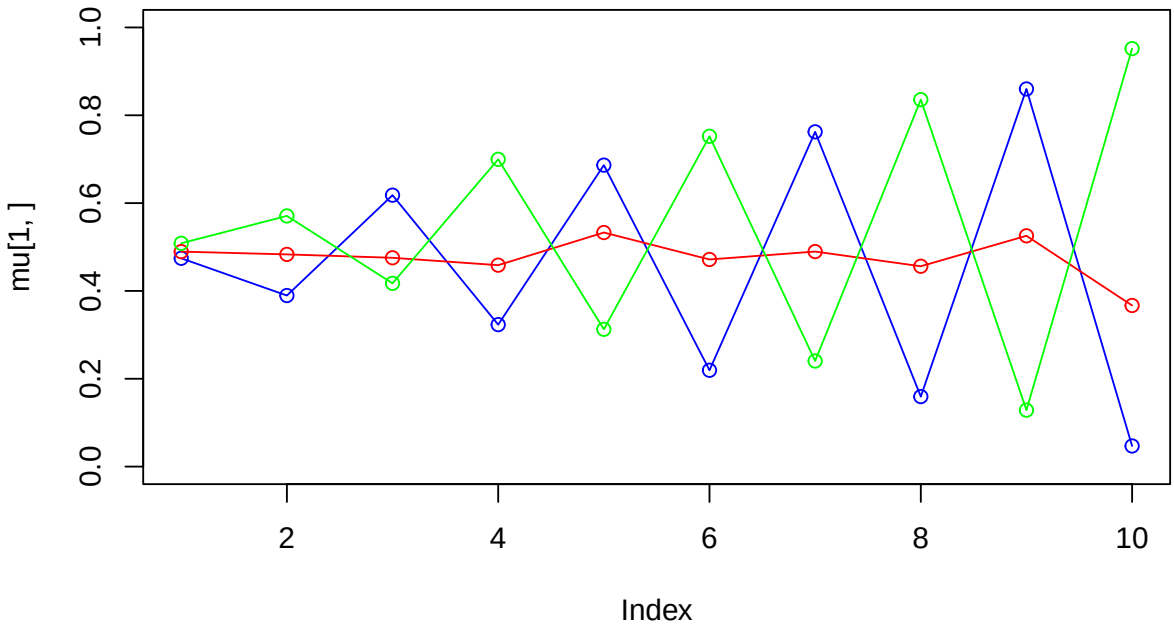
```
## iteration:  8 log likelihood:  -6357.743
```



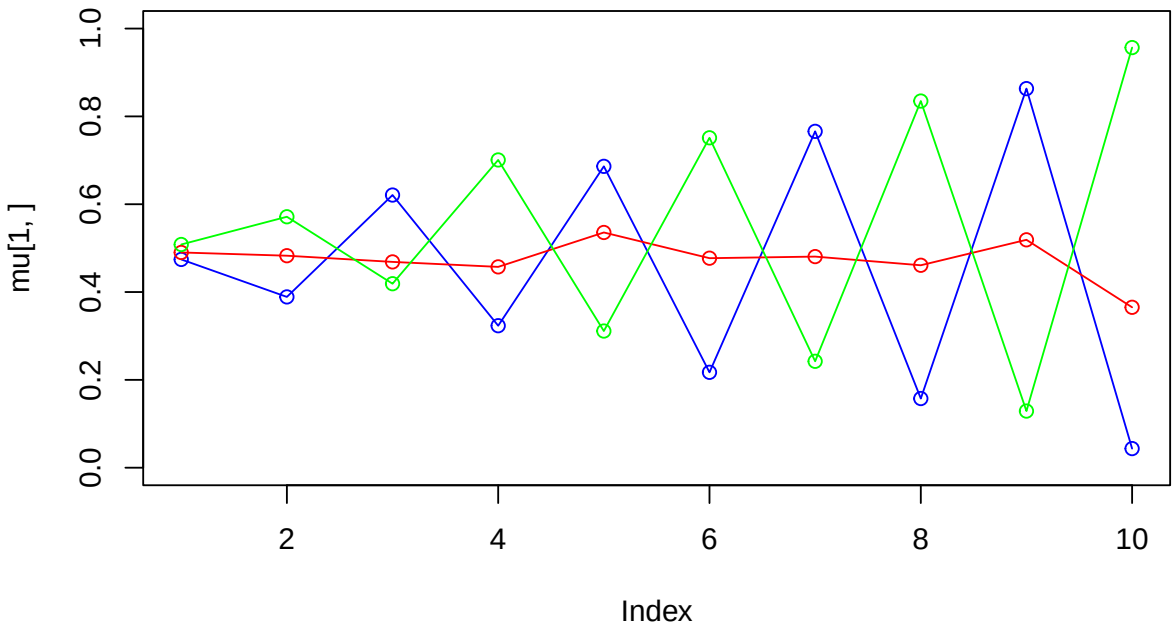

```
## iteration:  9 log likelihood:  -6351.637
```



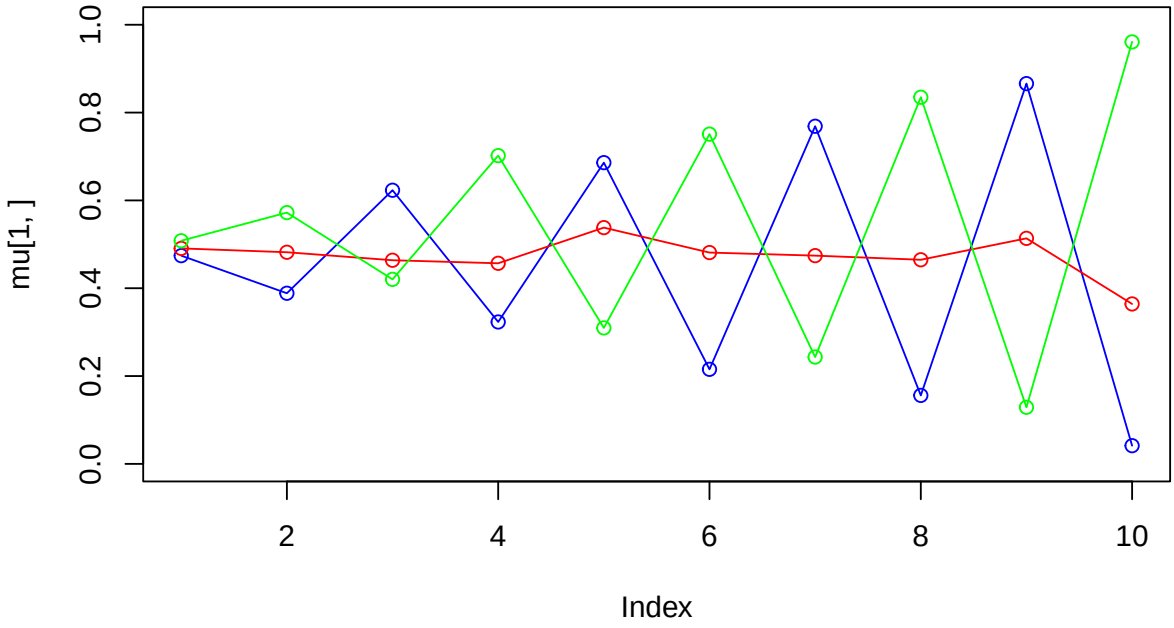

```
## iteration:  10 log likelihood:  -6349.59
```



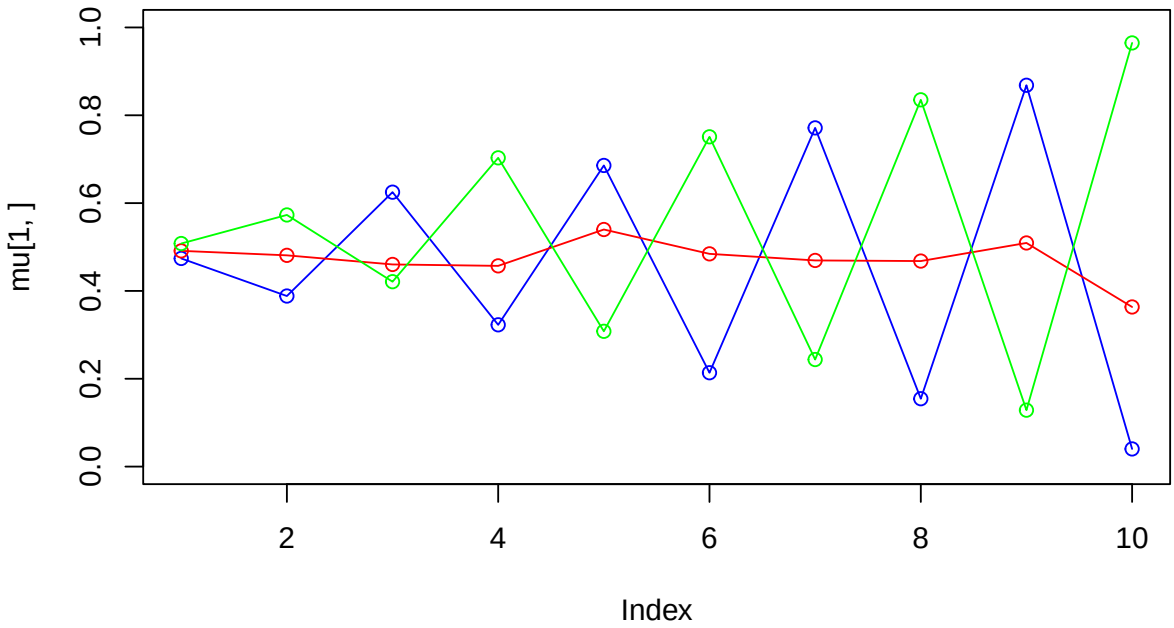
```
## iteration:  11 log likelihood:  -6348.513
```



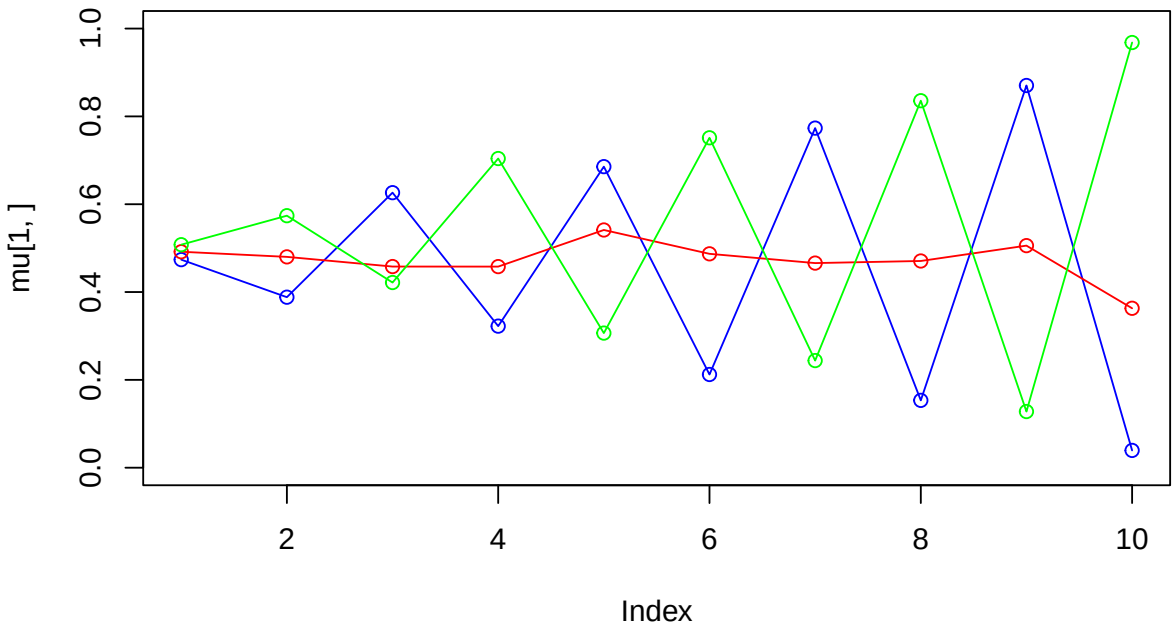

```
## iteration:  12 log likelihood:  -6347.809
```



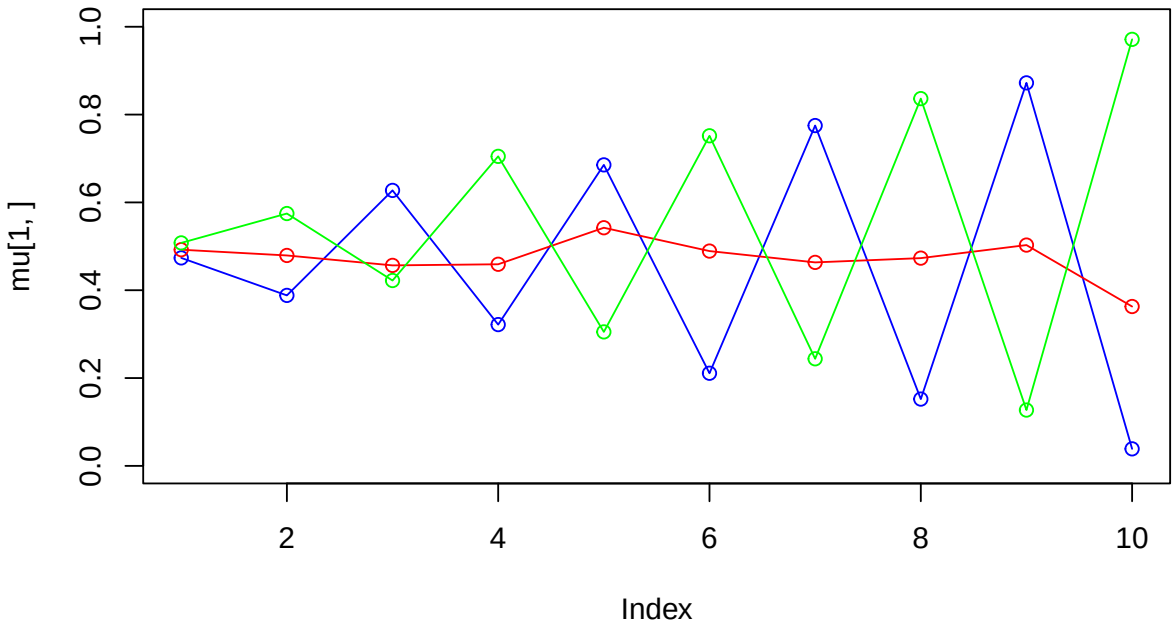

```
## iteration:  13 log likelihood:  -6347.284
```



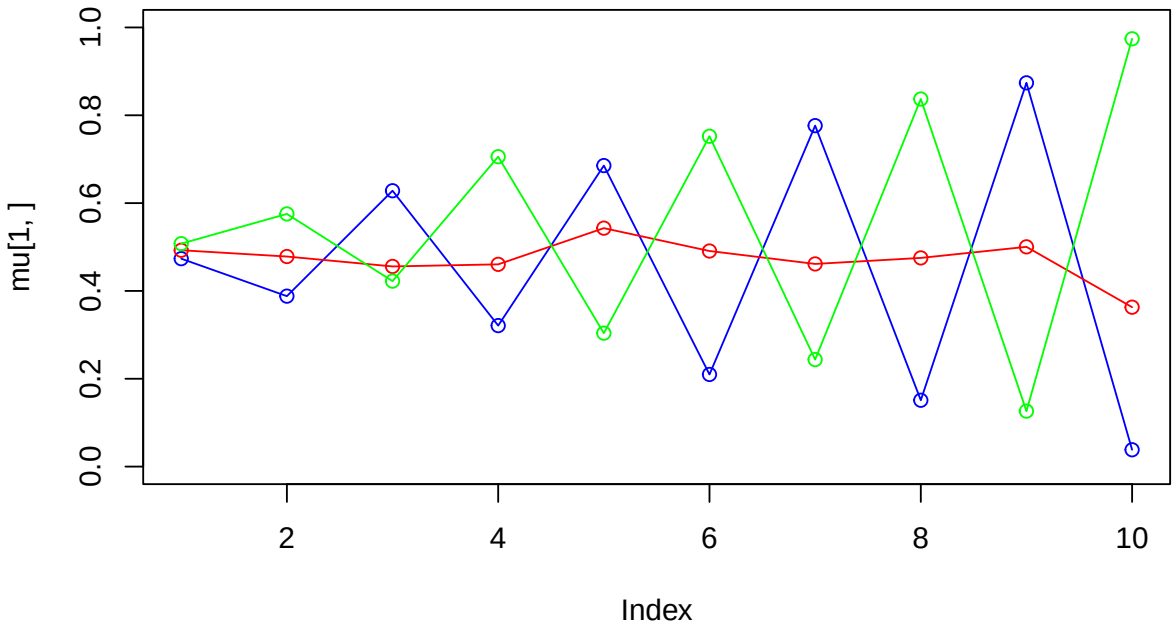
```
## iteration:  14 log likelihood:  -6346.861
```



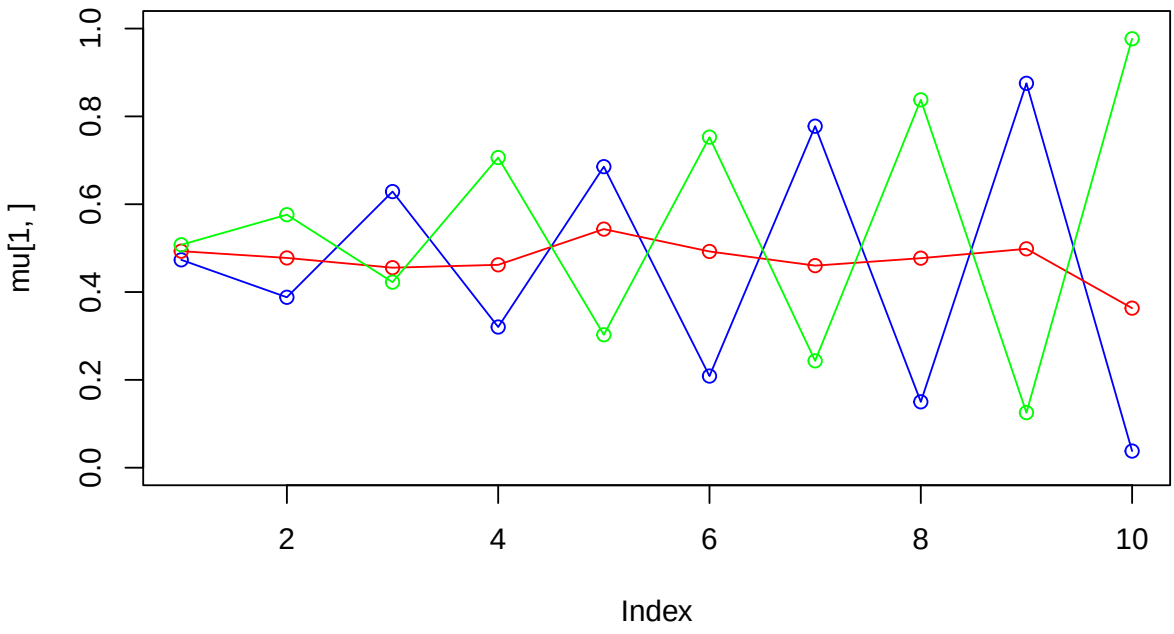

```
## iteration:  15 log likelihood:  -6346.506
```



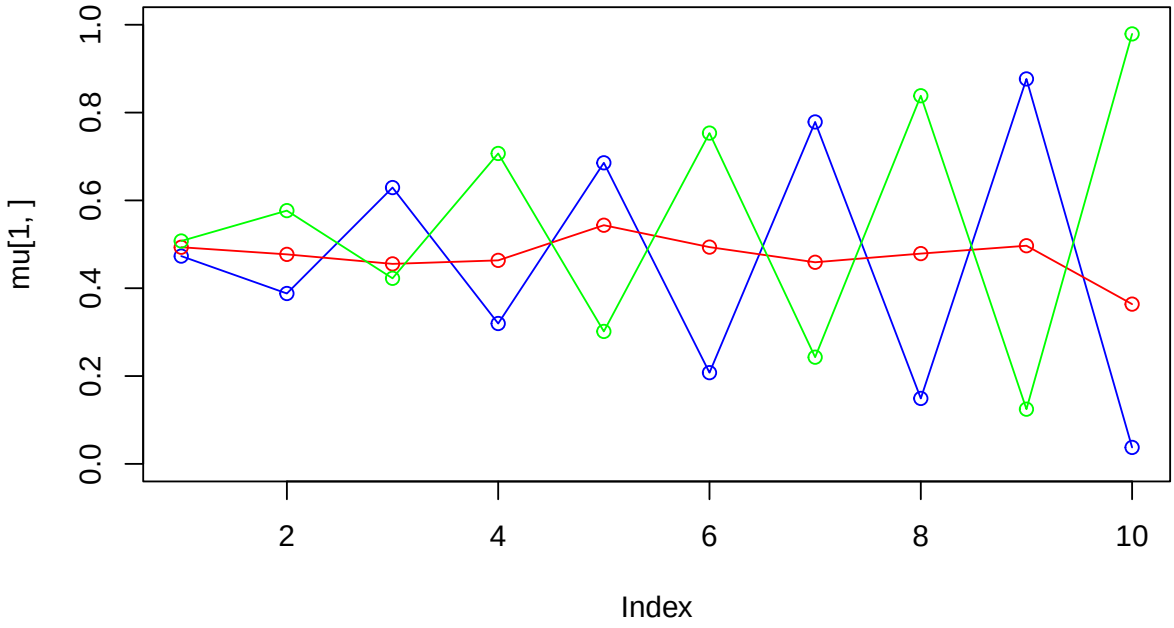

```
## iteration:  16 log likelihood:  -6346.2
```



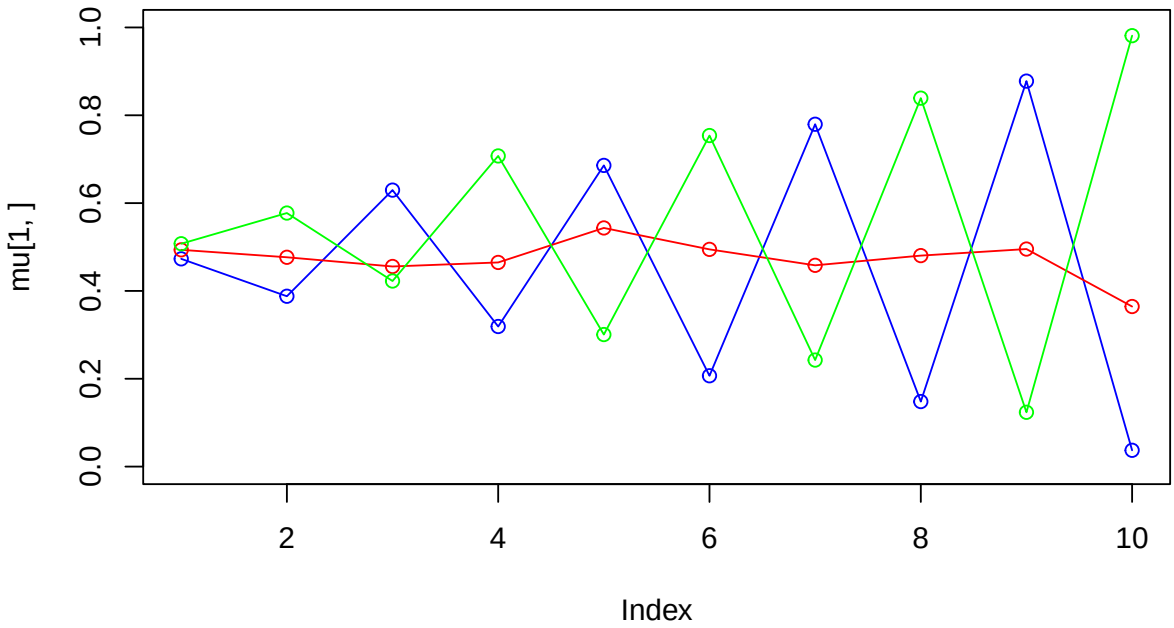
```
## iteration:  17 log likelihood:  -6345.934
```



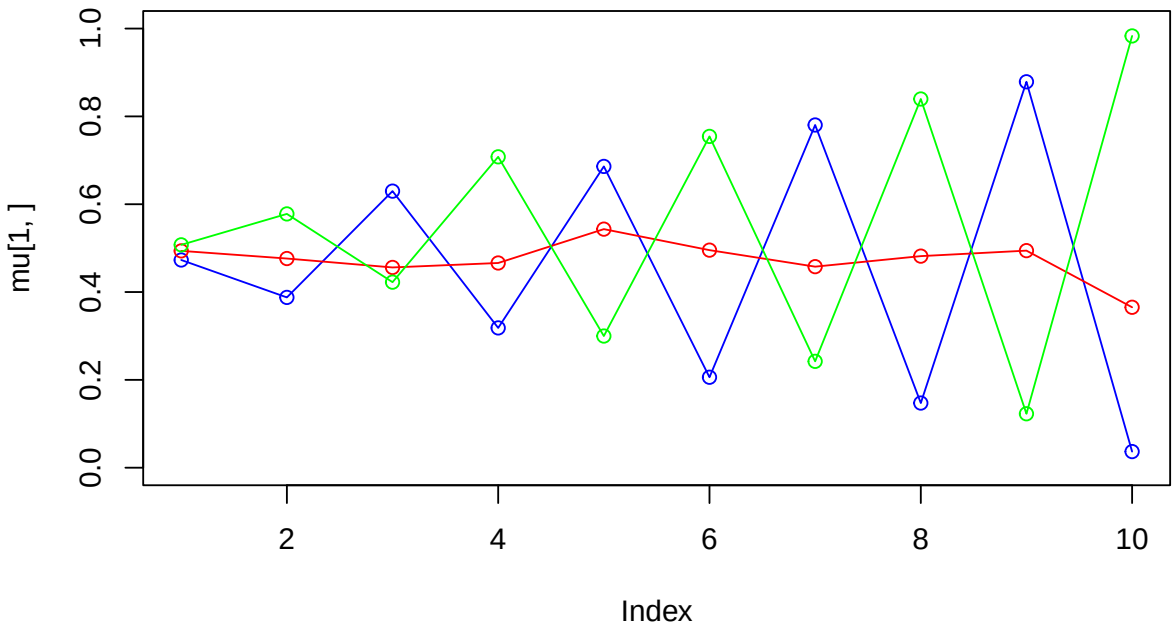

```
## iteration:  18 log likelihood:  -6345.699
```



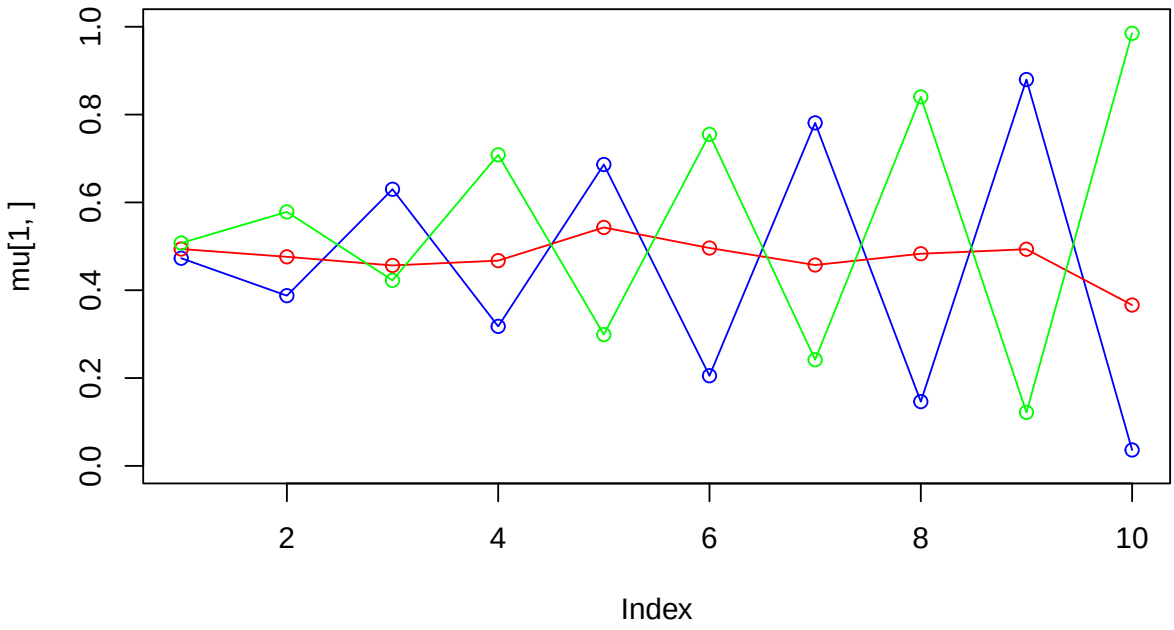

```
## iteration:  19 log likelihood:  -6345.492
```



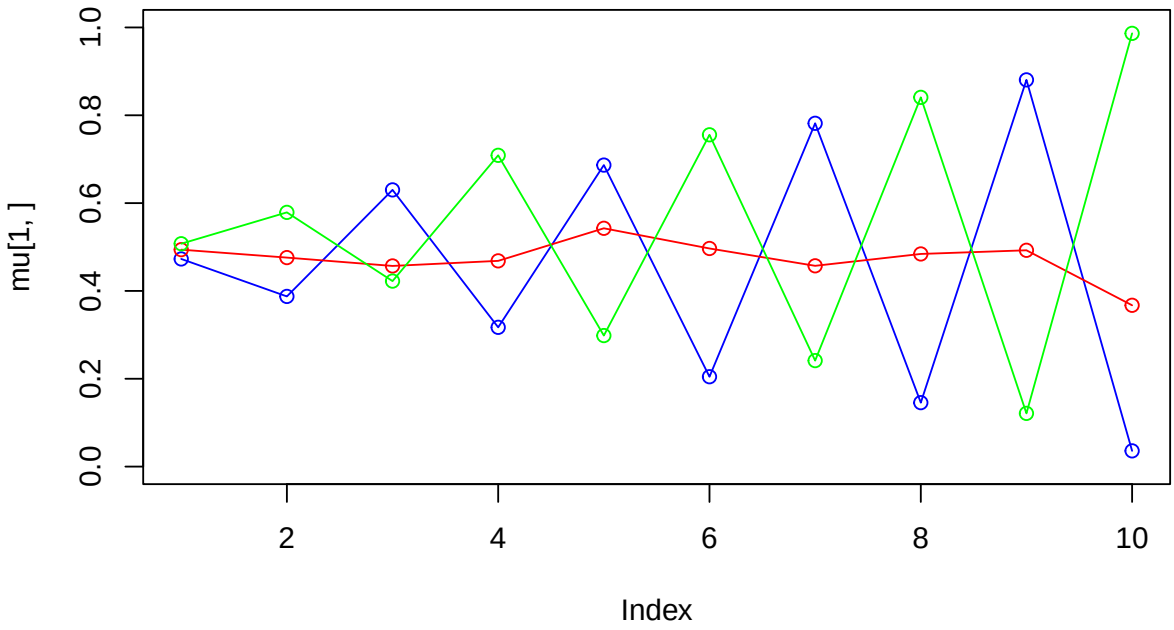
```
## iteration:  20 log likelihood:  -6345.309
```



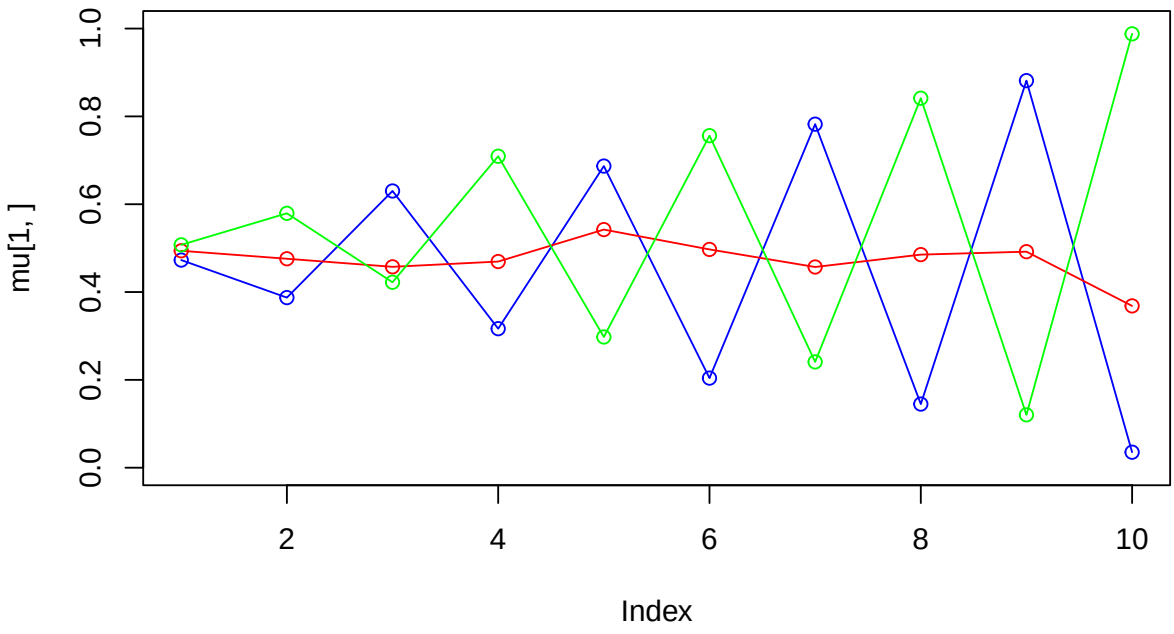

```
## iteration:  21 log likelihood:  -6345.147
```



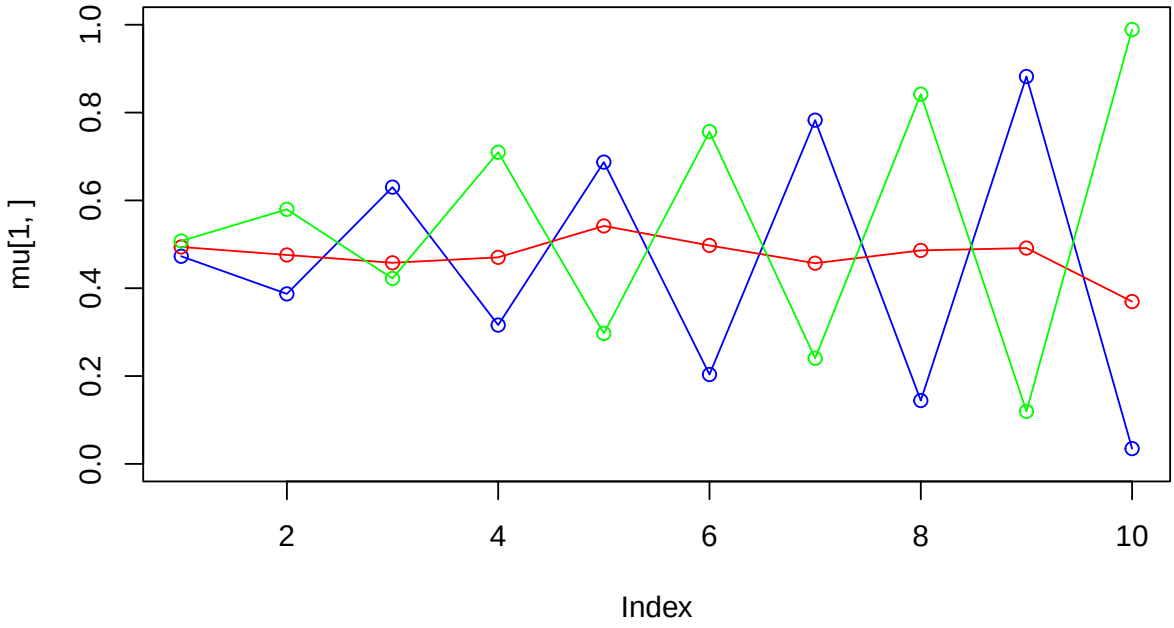

```
## iteration:  22 log likelihood:  -6345.003
```



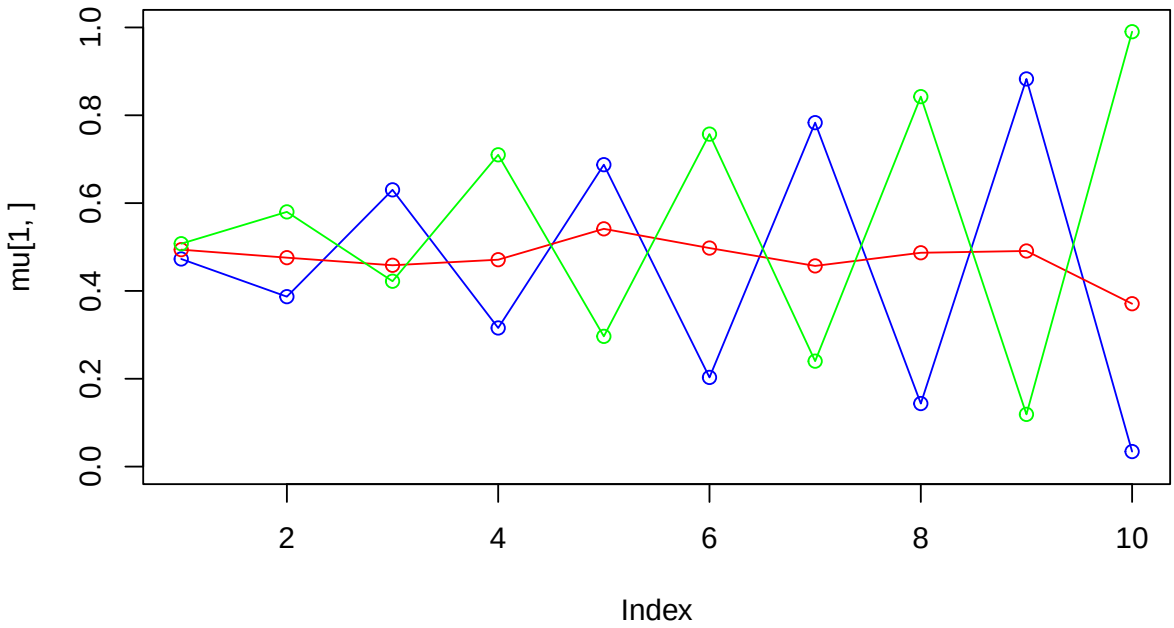
```
## iteration:  23 log likelihood:  -6344.875
```




```
## iteration:  24 log likelihood:  -6344.762
```




```
## iteration:  25 log likelihood:  -6344.66
```



```
## iteration:  26 log likelihood:  -6344.57
## Converged at iteration 26
```

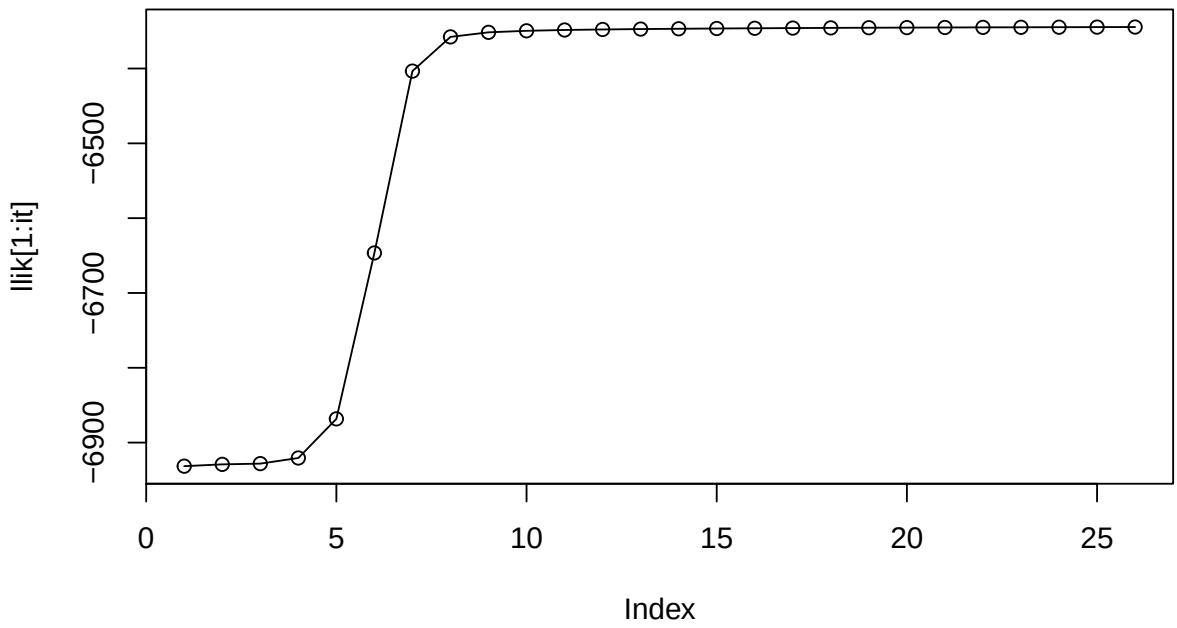
```
pi
```

```
## [1] 0.3416794 0.2690298 0.3892909
```

```
mu
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4727544 0.3869396 0.6302224 0.3156325 0.6875038 0.2030173 0.7832090
## [2,] 0.4939501 0.4757687 0.4584644 0.4711358 0.5413928 0.4976325 0.4569664
## [3,] 0.5075441 0.5800156 0.4221148 0.7100227 0.2965478 0.7571593 0.2400675
##           [,8]      [,9]      [,10]
## [1,] 0.1435650 0.8827796 0.03422816
## [2,] 0.4869015 0.4909904 0.37087402
## [3,] 0.8424441 0.1188864 0.99033611
```

```
plot(llik[1:it], type="o")
```



We can get results for different K values.When M is equal to 2, the coefficient (pi) approaches 0.5, meaning that the two clusters contribute nearly equally to the data. Although there are significant differences in the conditional distribution (mu) on each dimension,there is a big deviation from the true value, resulting in underfitting.When M is equal to 3, the convergence rate is slightly slower than when M is equal to 2, but the mu is more dispersed and closer to the true value, which can accurately capture the true structure of the data without being overly complicated.When M is equal to 4, the log-likelihood converges at the 44th iteration, and the convergence speed is the slowest, which indicates that the complexity of the model is significantly increased, and the weight of some clusters (such as pi1 and pi2) is very small, which indicates that the data may be unnecessarily split and overfitting may occur.

0.4 Assignment 3:MIXTURE MODELS

- In an ensemble model, is it true that the larger the number B of ensemble members the more flexible the ensemble model?
  - No, more ensemble members does not make the resulting model more flexible but only reduces the variance.(page 169)
- In AdaBoost, what is the loss function used to train the boosted classifier at each iteration?
  - The loss fucntion is the exponential loss function below (page 177):

$$L(y \cdot f(x)) = \exp(-y \cdot f(x))$$

- Sketch how you would use cross-validation to select the number of components (or clusters) in unsupervised learning of GMMs.
  - Cluster number M is chosen when the benefit from choosing M+1 clusters is insignificant. By looking into the case provided by course book, the objective drops significantly when the number of cluster increases from 1 to 4. But when it increases from 4 to 5, the influence on objective is negligible. The 'bending point' is usually a good candidate of the cluster number. (Page 267

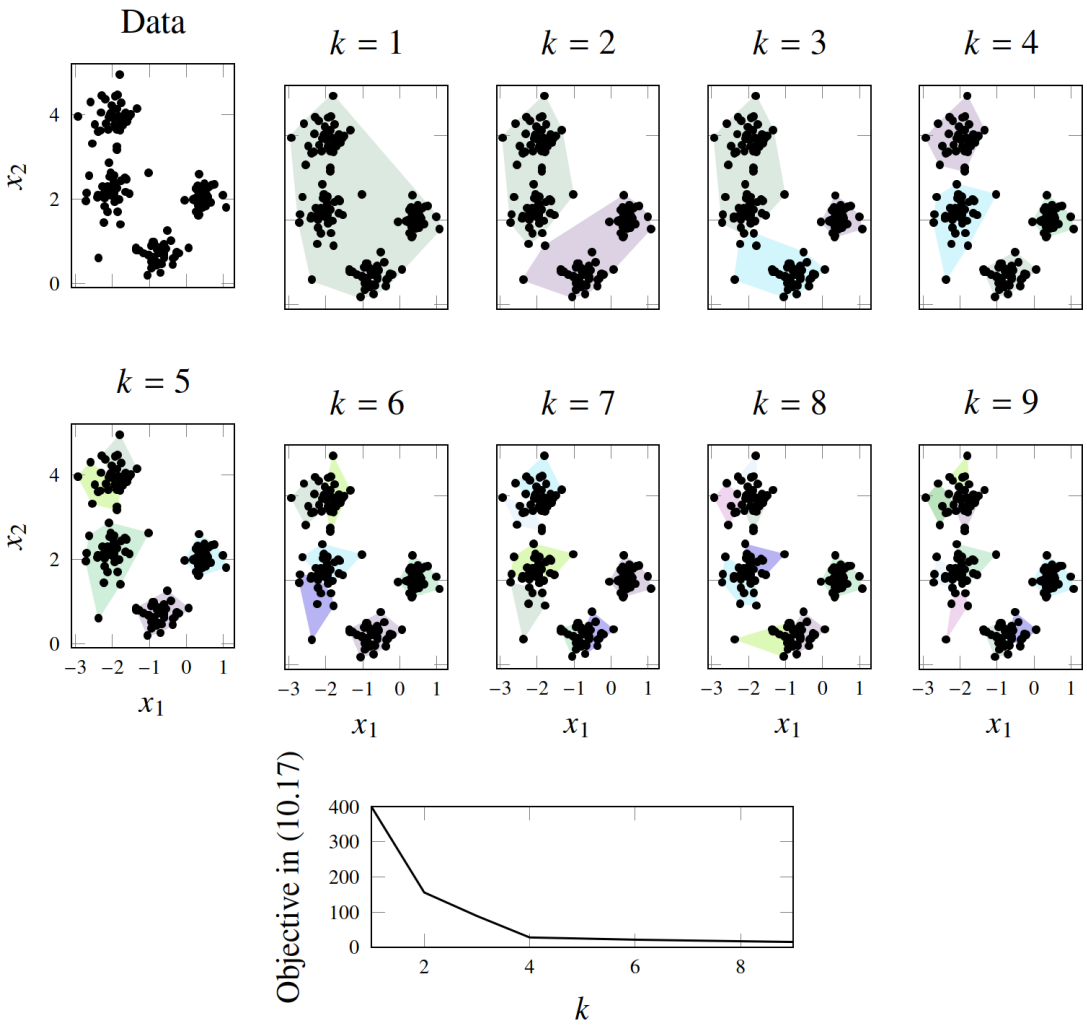


Figure 1: Selection of number of components for unsupervised learning of GMMs