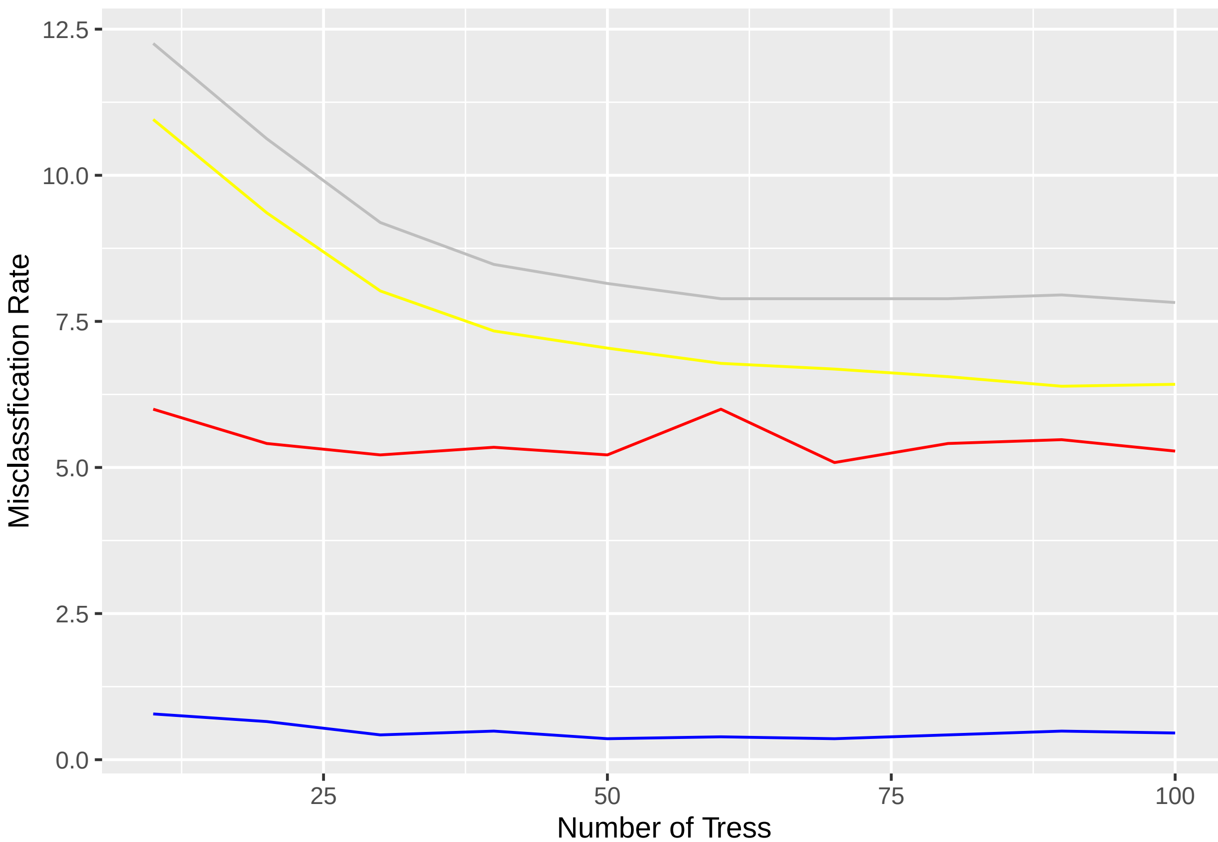
Block 2 Lab 1 Report

Zuxiang Li(zuxli371)

12/3/2019

1. ENSEMBLE METHODS

The file spambase.csv contains information about the frequency of various words, char- acters, etc. for a total of 4601 e-mails. Furthermore, these e-mails have been classified as spams (spam = 1) or regular e-mails (spam = 0). Your task is to evaluate the performance of Adaboost classification trees and random forests on the spam data. Specifically, provide a plot showing the error rates when the number of trees considered are 10, 20, . . . , 100. To estimate the error rates, use 2/3 of the data for training and 1/3 as hold-out test data. To learn Adaboost classification trees, use the function blackboost() of the R package mboost. Specify the loss function corresponding to Adaboost with the parameter family. To learn random forests, use the function randomForest of the R package randomForest.



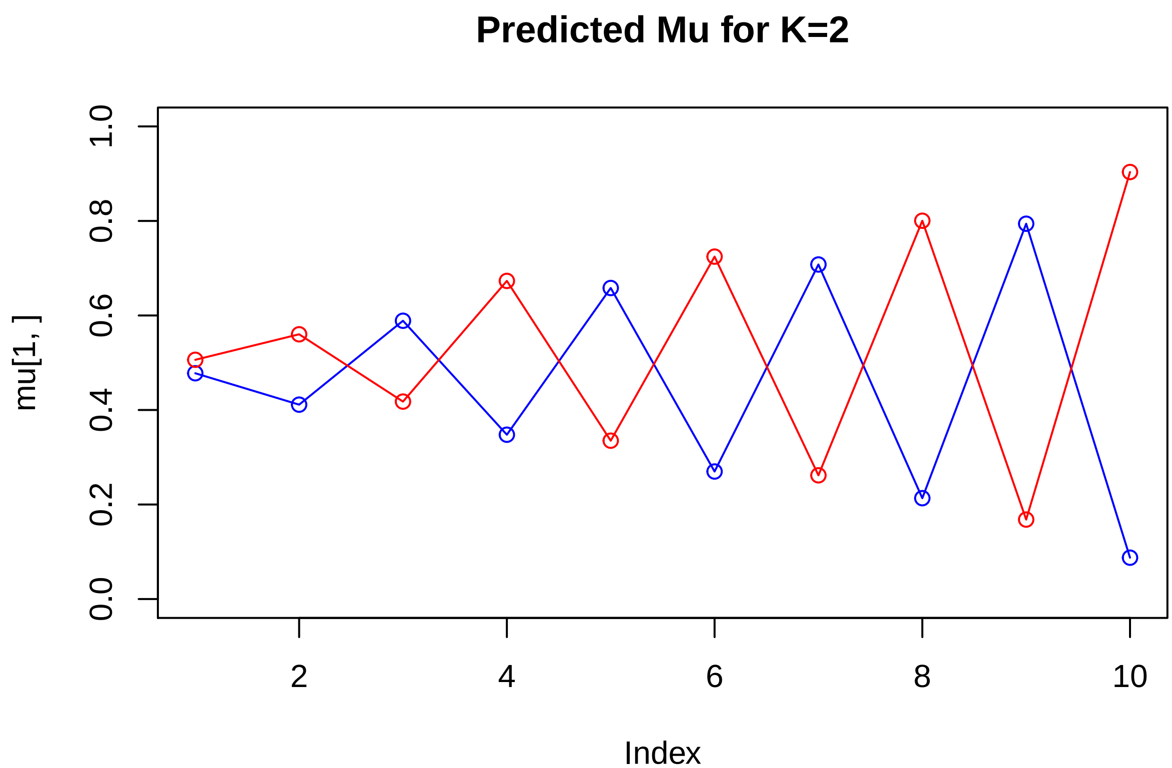
The plot above showed the Misclassfication rate using different models and number of trees. Grey and yellow represent Adaboost model for train and test data. Red and blue lines are Randomforest model for train and test data.

The misclassfication values of Adaboost with incresaing number of trees are constantly decreasing for both train and test data. In every iteration a classfier is produced and given different weight. Then use the classifer generated in previous iteration to produced.It returns a weighted average of the classfiers. So the rate keeps decrease.

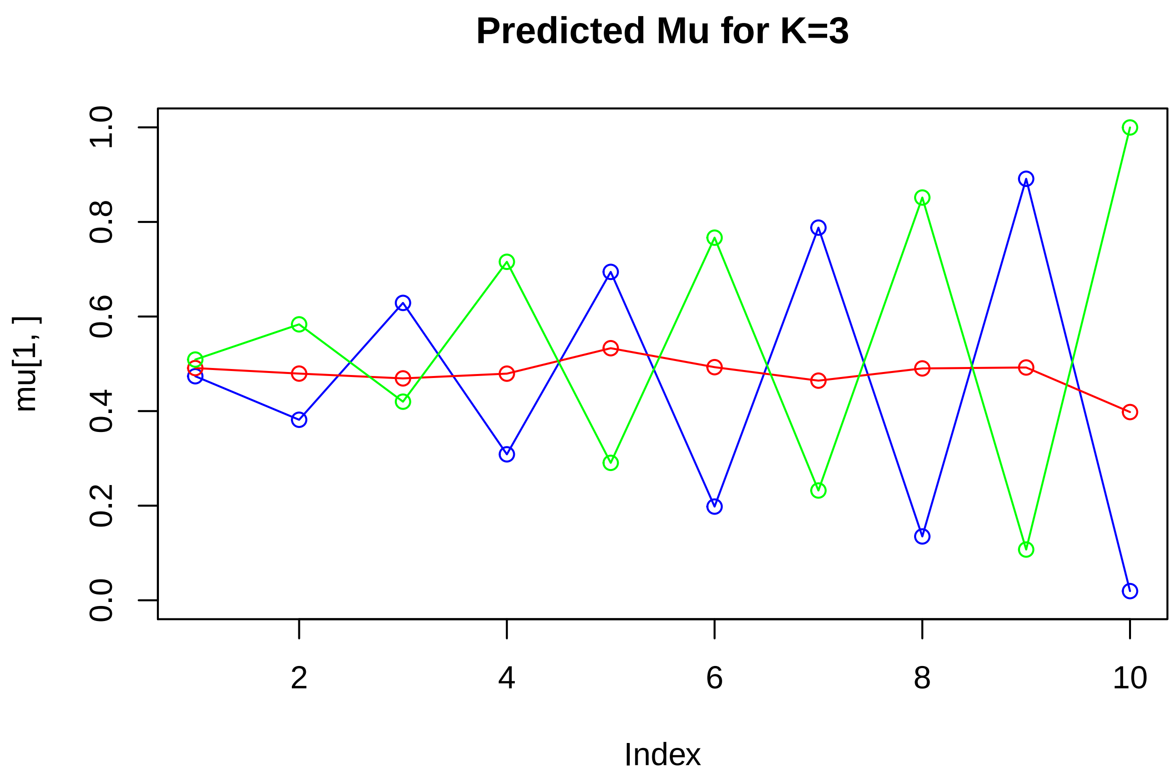
As for Random forest, number of tress seems have no effect on the misclassfication rate for both train and test data. Since Random forest used bagging, individual classifiers are then combined by taking a simple majority vote of their decisions. Random forest model have a better performance than Adaboost.

2. MIXTURE MODELS

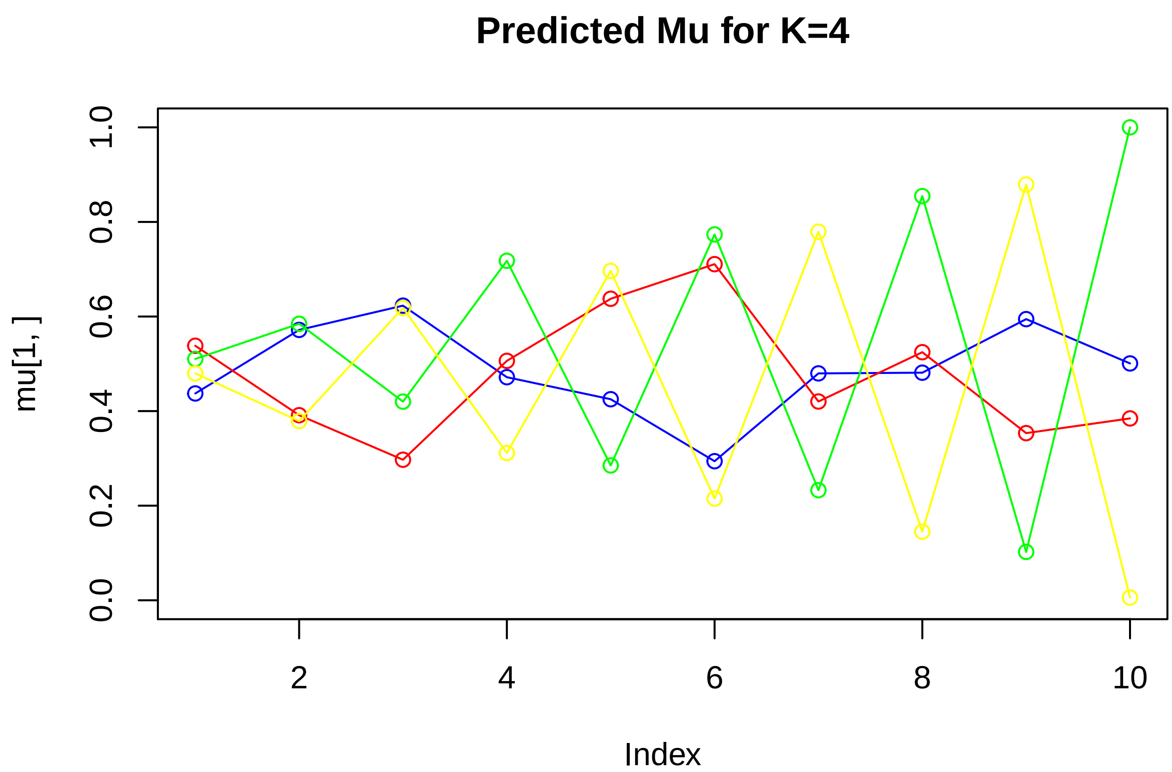
Your task is to implement the EM algorithm for mixtures of multivariate Benoulli distributions. Please use the template in the next page to solve the assignment. Then, use your implementa- tion to show what happens when your mixture models has too few and too many components, i.e. set K = 2, 3, 4 and compare results. Please provide a short explanation as well.



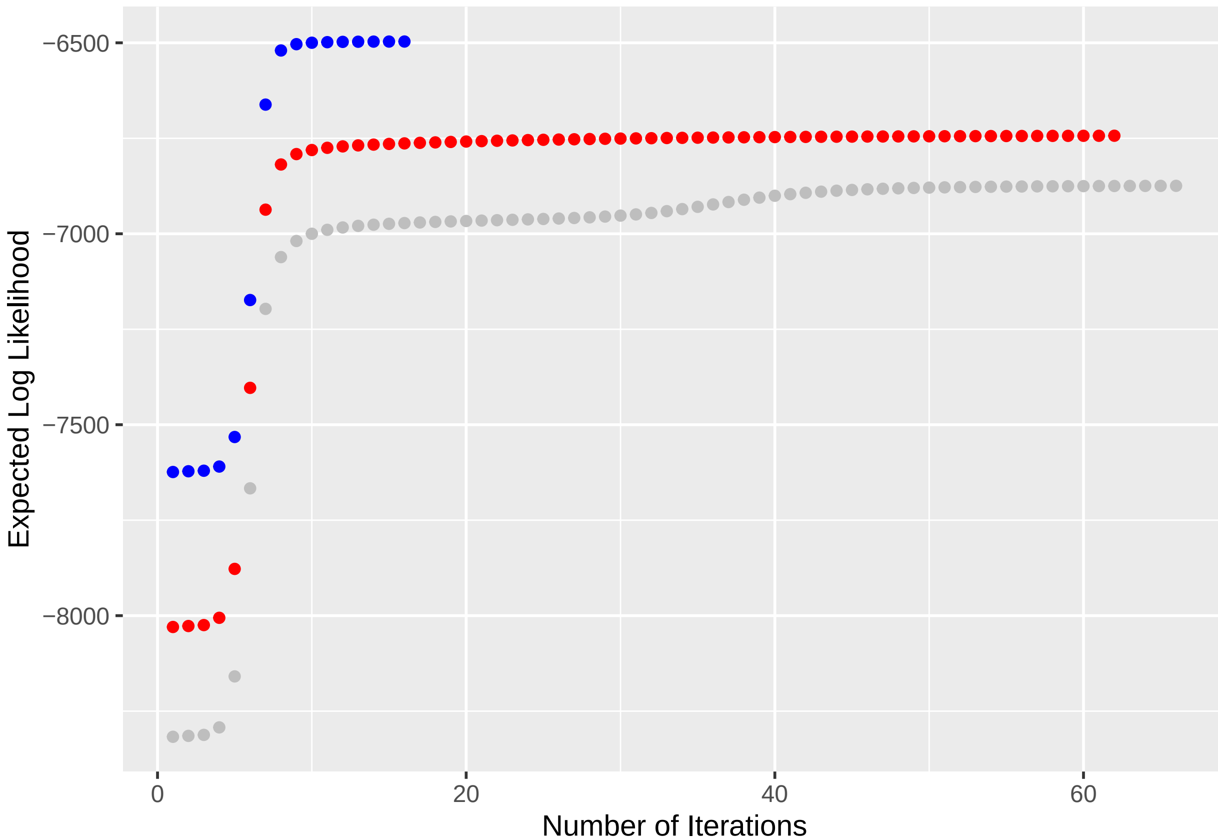

```
## [1] 0.4981919 0.5018081
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4777136 0.4113065 0.5888317 0.3477062 0.6580979 0.2698303 0.7078467
## [2,] 0.5061835 0.5601552 0.4177868 0.6731171 0.3350702 0.7245255 0.2617664
##           [,8]      [,9]     [,10]
## [1,] 0.2134061 0.7941168 0.0875152
## [2,] 0.8004711 0.1681467 0.9035340
```




```
## [1] 0.3259592 0.3044579 0.3695828
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737193 0.3817120 0.6288021 0.3086143 0.6943731 0.1980896 0.7879447
## [2,] 0.4909874 0.4793213 0.4691560 0.4791793 0.5329895 0.4928830 0.4643990
## [3,] 0.5089571 0.5834802 0.4199272 0.7157107 0.2905703 0.7667258 0.2320784
##           [,8]      [,9]      [,10]
## [1,] 0.1349651 0.8912534 0.01937869
## [2,] 0.4902682 0.4922194 0.39798407
## [3,] 0.8516111 0.1072226 0.99981353
```




```
## [1] 0.1614155 0.1383613 0.3609912 0.3392319
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4372908 0.5716691 0.6230114 0.4717152 0.4251232 0.2940734 0.4797605
## [2,] 0.5381955 0.3913346 0.2971686 0.5062848 0.6375272 0.7107583 0.4202372
## [3,] 0.5102441 0.5846281 0.4200464 0.7178717 0.2850900 0.7735833 0.2327656
## [4,] 0.4797762 0.3788928 0.6181216 0.3114748 0.6964392 0.2149967 0.7793732
##           [,8]      [,9]       [,10]
## [1,] 0.4812185 0.5945364 0.500815206
## [2,] 0.5246082 0.3534161 0.384513179
## [3,] 0.8546627 0.1022323 0.999999734
## [4,] 0.1450708 0.8791286 0.005800712
```



In this case, we generate data using true_mu and true_pi. For K=2, the blue dots, which means too few components, the blue points in the plot, only 16 iterations were made, this model is underfitted. For K=4, the grey dots, too many components, grey dots in the plot, 66 iterations were made, then this model is overfitted. Only when K=3, the red dots in plot, we got similar mu and pi compare to the true values.

Appendix

1. ENSEMBLE METHODS

```
library(mboost)
library(randomForest)
# Import data
sp <- read.csv2("material/spambase.csv")
sp$Spam <- as.factor(sp$Spam)
n <- dim(sp)[1]
set.seed(1234567890)
id <- sample(1:n, floor(n*2/3))
train <- sp[id,]
test <- sp[-id,]

# Generate confusion matrix and calculate misclassfication rate
cft<-function(Pred,Actual){
  cft<-table(Pred,Actual)
  # True positive
  tp <- cft[2, 2]
  # True negetive
  tn <- cft[1, 1]
  # False positive
  fp <- cft[2, 1]
  # False negetive
  fn <- cft[1, 2]

  misclassfication <- (1-(tp + tn)/(tp + tn + fp + fn))*100
  return(misclassfication)
}


ensemble_methods<-function(method,data){
  err_rate<-c()
  # select methods mboost
  if(substitute(method)=="mboost"){
    # Loop for 10 times each time the number of trees increase 10
    for(i in seq(10,100,10)){
      # Generate mboost model using train data,specify loss function as Binomial()
      bb<-blackboost(Spam~.,data=train,control = boost_control(mstop = i),family = AdaExp())
      # Get the predict value based on model we generated
      pred<-predict.mboost(bb,data,type="class")
      # Get the missclassfication rate for current number of trees
      res<-cft(Pred=pred,Actual=data$Spam)
      err_rate<-c(err_rate,res)
    }
  }
  # select methods randomforest
  else if(substitute(method)=="randomforest"){
    # Loop for 10 times each time the number of trees increase 10
    for(i in seq(10,100,10)){
      # Generate randomforest model
```

```r
      rf<-randomForest(Spam~.,data=train,ntree=i)
      # Get predict value
      pred<-predict(rf,data)
      # Get the missclassfication rate for current number of trees
      res<-cft(Pred=pred,Actual=data$Spam)
      err_rate<-c(err_rate,res)
    }
  }
      return(err_rate)
}
err_rate1<-ensemble_methods("mboost",train)
err_rate2<-ensemble_methods("mboost",test)
err_rate3<-ensemble_methods("randomforest",train)
err_rate4<-ensemble_methods("randomforest",test)

df<-data.frame(step=seq(10,100,10),err1=err_rate1,err2=err_rate2,err3=err_rate3,err4=err_rate4)
ggplot()+xlab("")+geom_line(data = df, aes(x = step, y = err3), color = "blue") +
  geom_line(data = df, aes(x = step, y = err4), color = "red") +geom_line(data = df, aes(x = step, y = e
  geom_line(data = df, aes(x = step, y = err2), color = "grey")+xlab("Number of Tress")+ylab("Misclassfi
```

2. MIXTURE MODELS

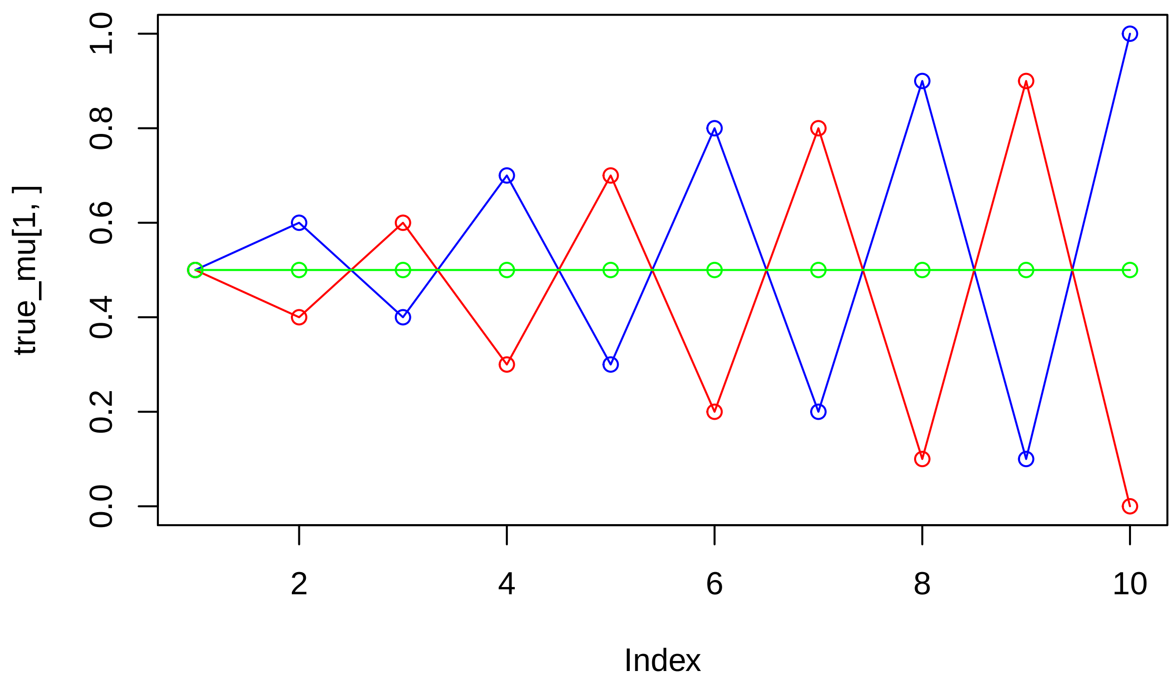
```r
RNGversion('3.5.1')
```

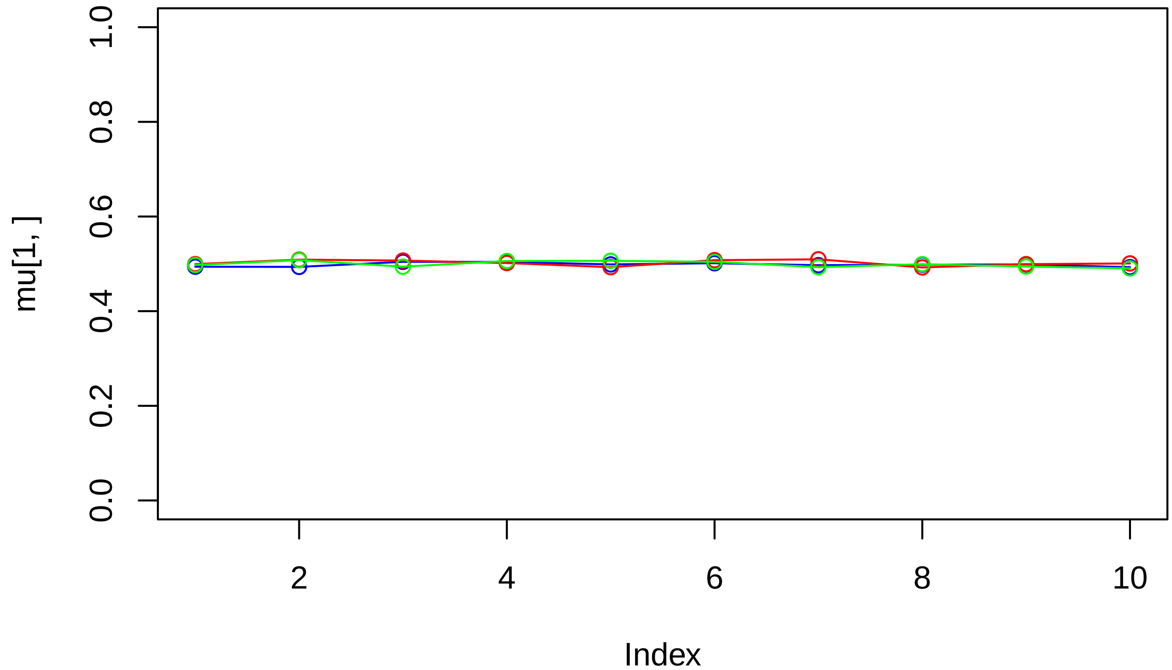
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

```r
EM_algorithm<-function(K){
  set.seed(1234567890)
  max_it <- 100 # max number of EM iterations
  min_change <- 0.1 # min change in log likelihood between two consecutive EM 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(n in 1:N) {
    k <- sample(1:3,1,prob=true_pi)
    for(d in 1:D) {
      x[n,d] <- rbinom(1,1,true_mu[k,d])
    }
  }
  #K=3 # number of guessed components
```

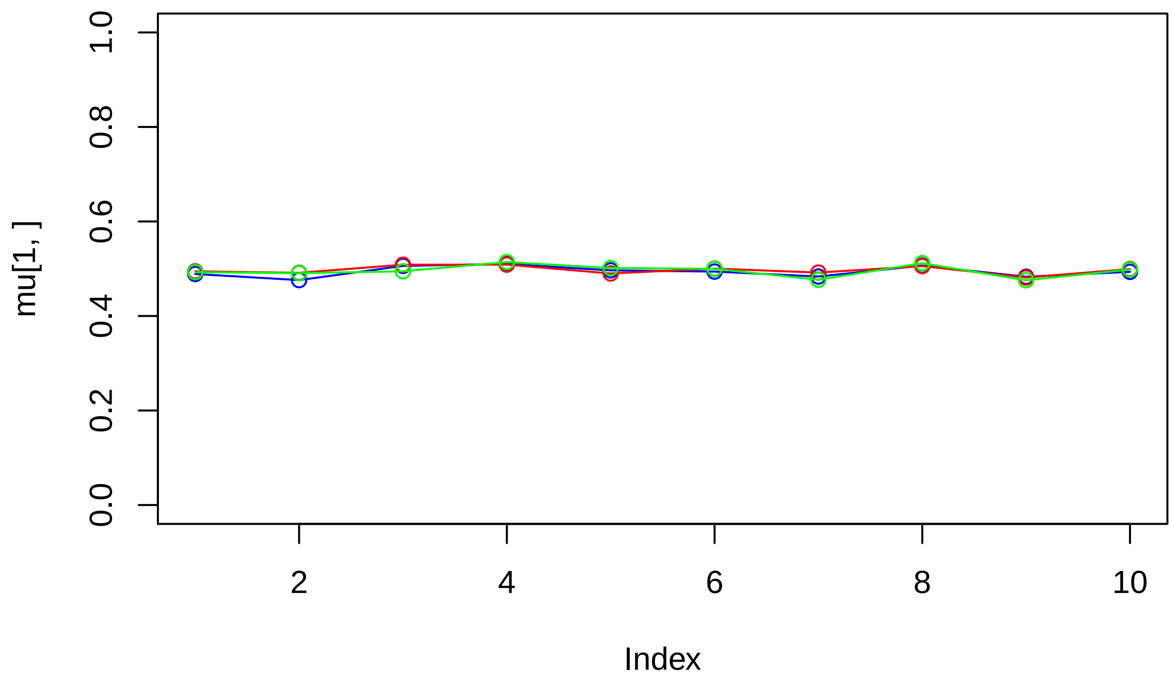
```r
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

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 fractional component assignments

  for(n in 1:N){
    denominator<-c()
    numerator<-c()
    for(k in 1:K){
      new_numerator<-pi[k]*prod(mu[k,]^x[n,]*(1-mu[k,])^(1-x[n,]))
      numerator<-c(numerator,new_numerator)
      denominator<-sum(numerator)
    }
    z[n,]<-numerator/denominator
  }
  #Log likelihood computation.
  sum=0
  for(n in 1:N){
    for(k in 1:K){
      sum=sum+z[n,k]*(log(pi[k])+sum(x[n,]*log(mu[k,])+(1-x[n,])*log(1-mu[k,])))
    }
  }
  llik[it]<-sum

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if(it>1){
    if(abs(llik[it]-llik[it-1])<min_change)
      break
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  # New mixing coefficients for next iteration
  pi=apply(z,2,mean)
  # New conditional distributions
  for(k in 1:K){
    sum=0
    for (n in 1:N) {
```

```
        sum = sum + x[n,] * z[n,k]
      }
      mu[k,] = sum / sum(z[,k])
    }
  }
  pi
  mu
  plot(llik[1:it], type="o")
  return(llik[1:it])
  #return(list(pi=pi,mu=mu,res=,llik=llik[1:it])
}

#em1<-EM_algorithm(K=2)
em2<-EM_algorithm(K=3)
```

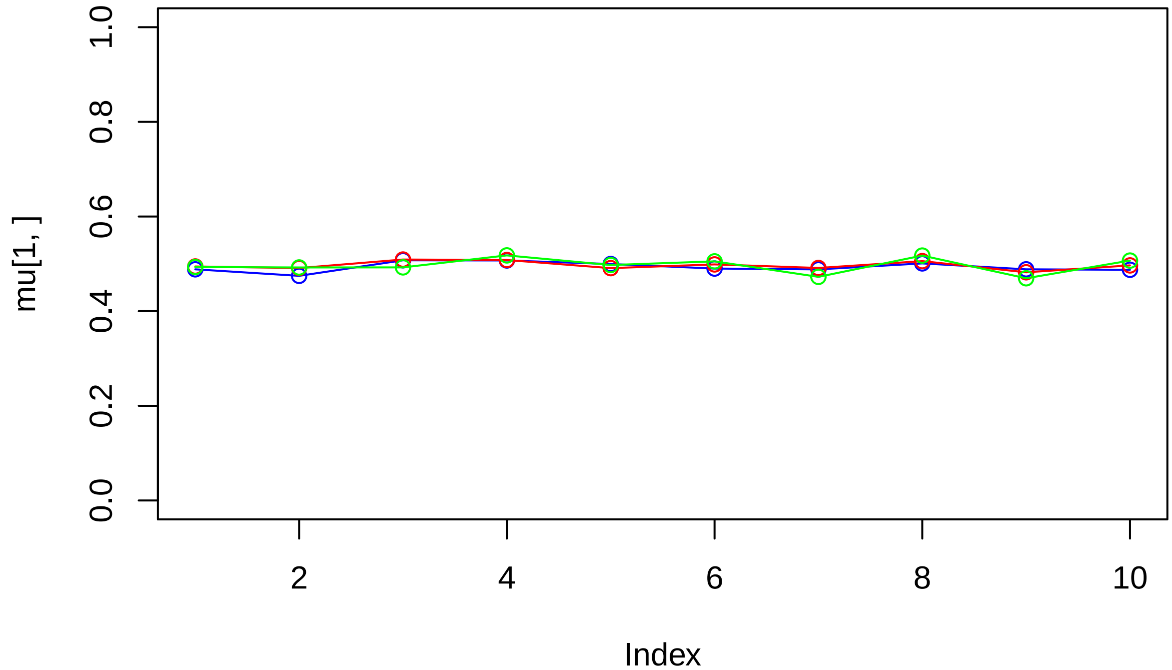




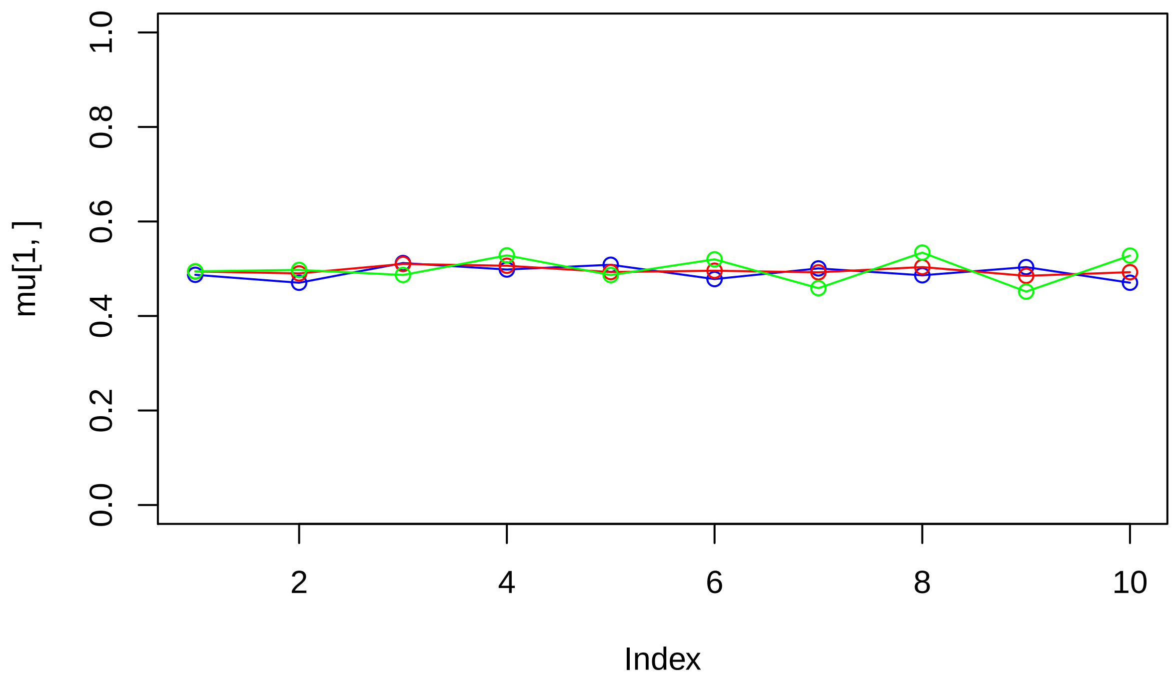

```
## iteration:  1 log likelihood:  -8029.723
```



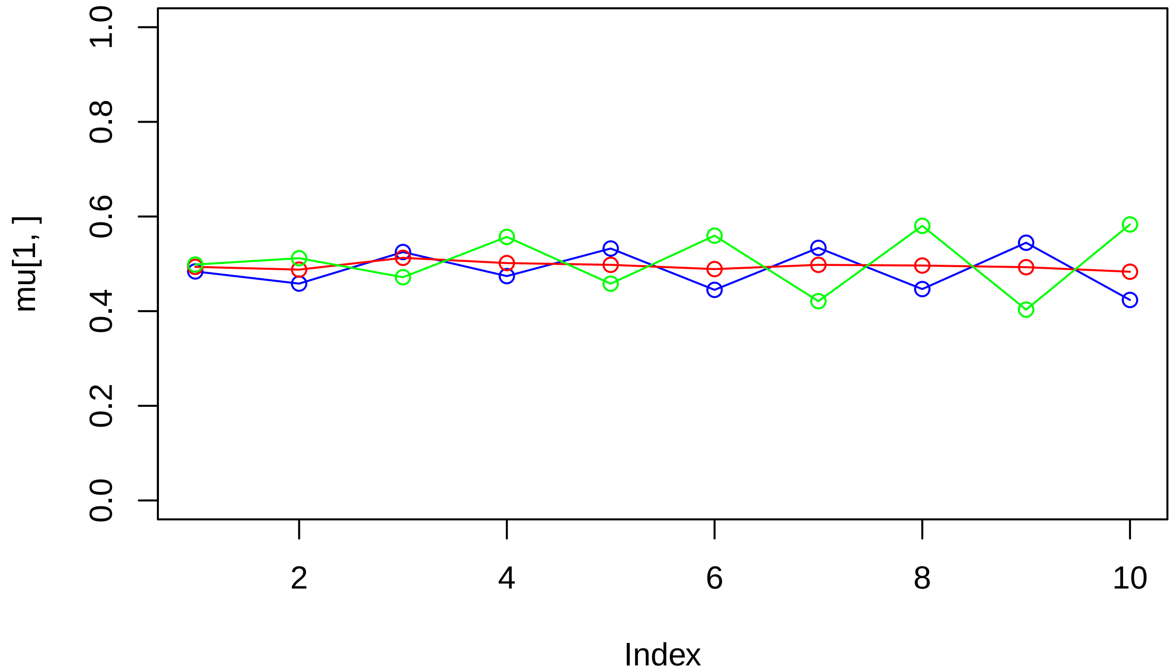

```
## iteration:  2 log likelihood:  -8027.183
```



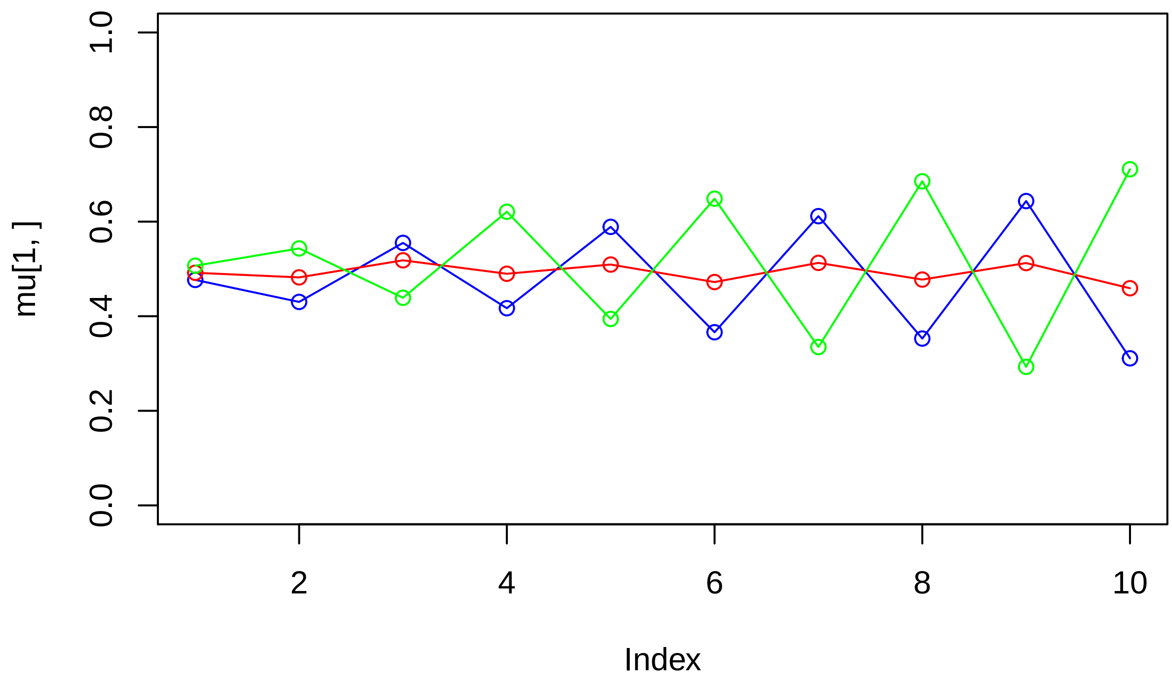

```
## iteration:  3 log likelihood:  -8024.696
```



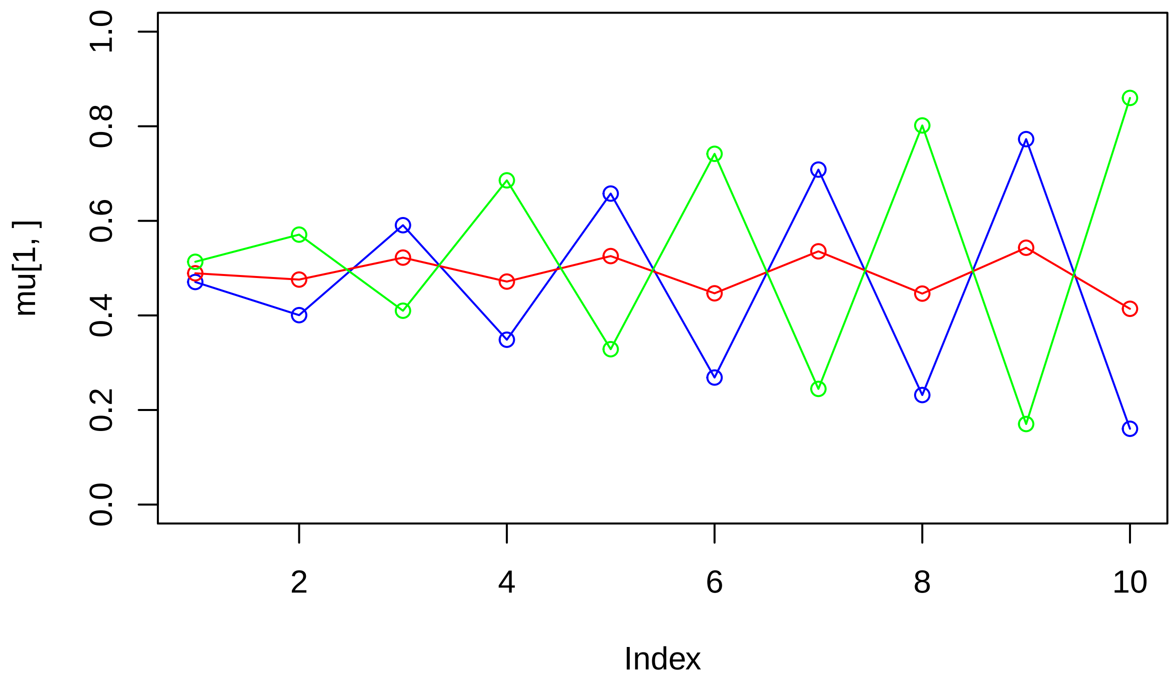

```
## iteration:  4 log likelihood:  -8005.631
```



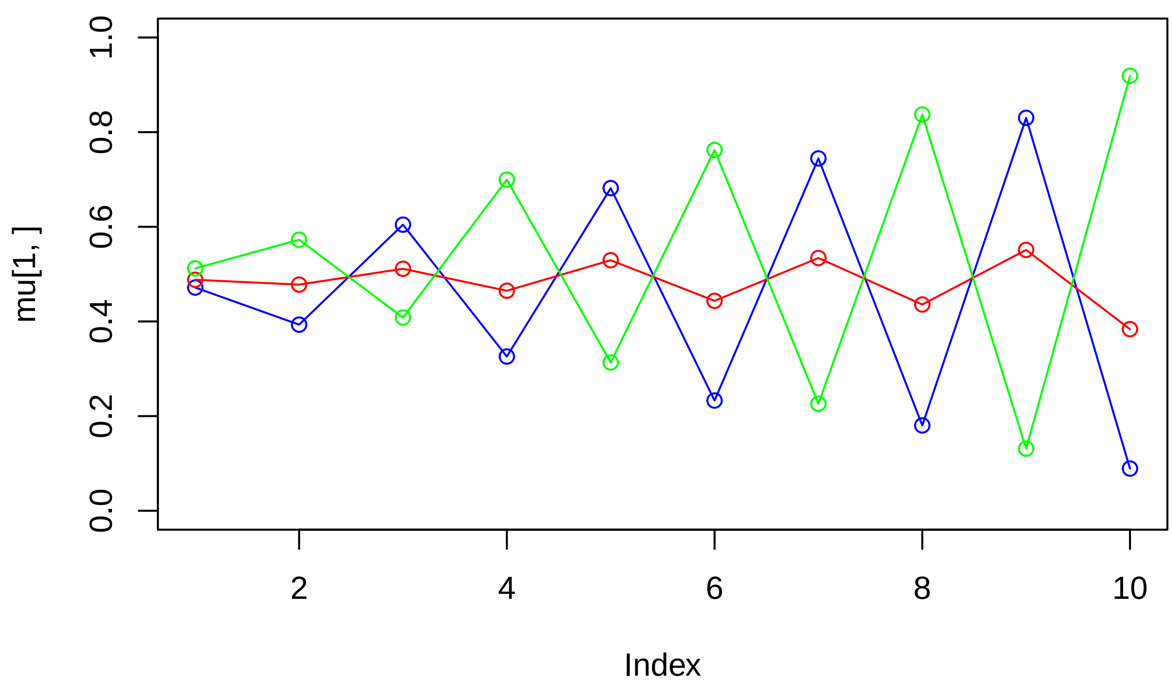

```
## iteration:  5 log likelihood:  -7877.606
```



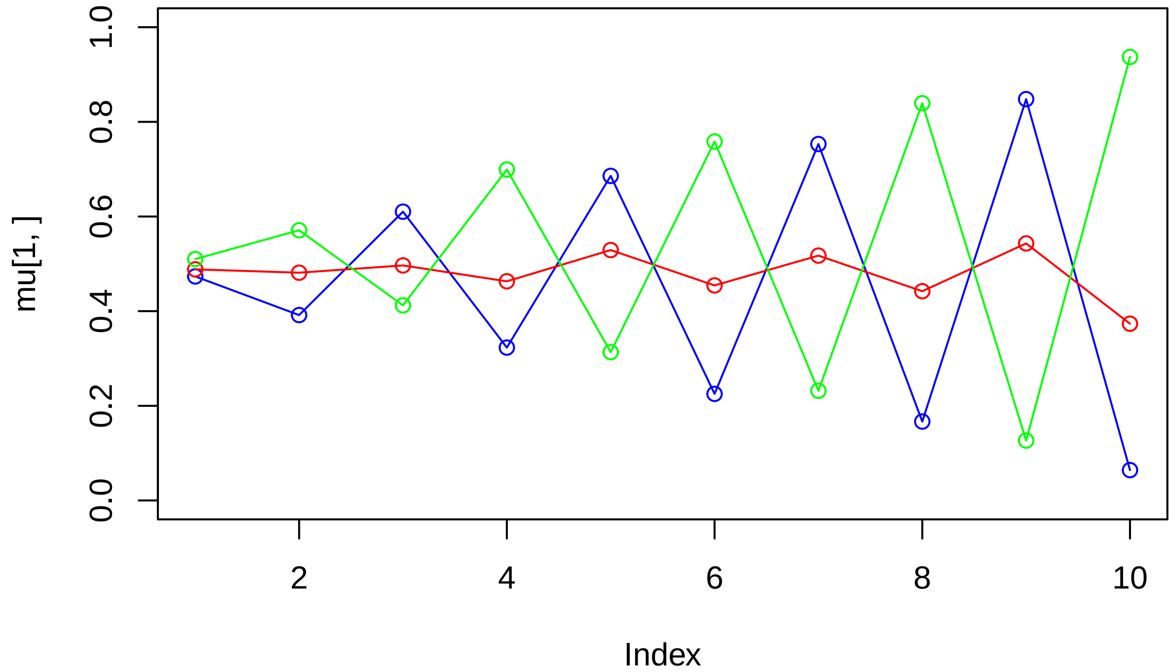

```
## iteration:  6 log likelihood:  -7403.513
```



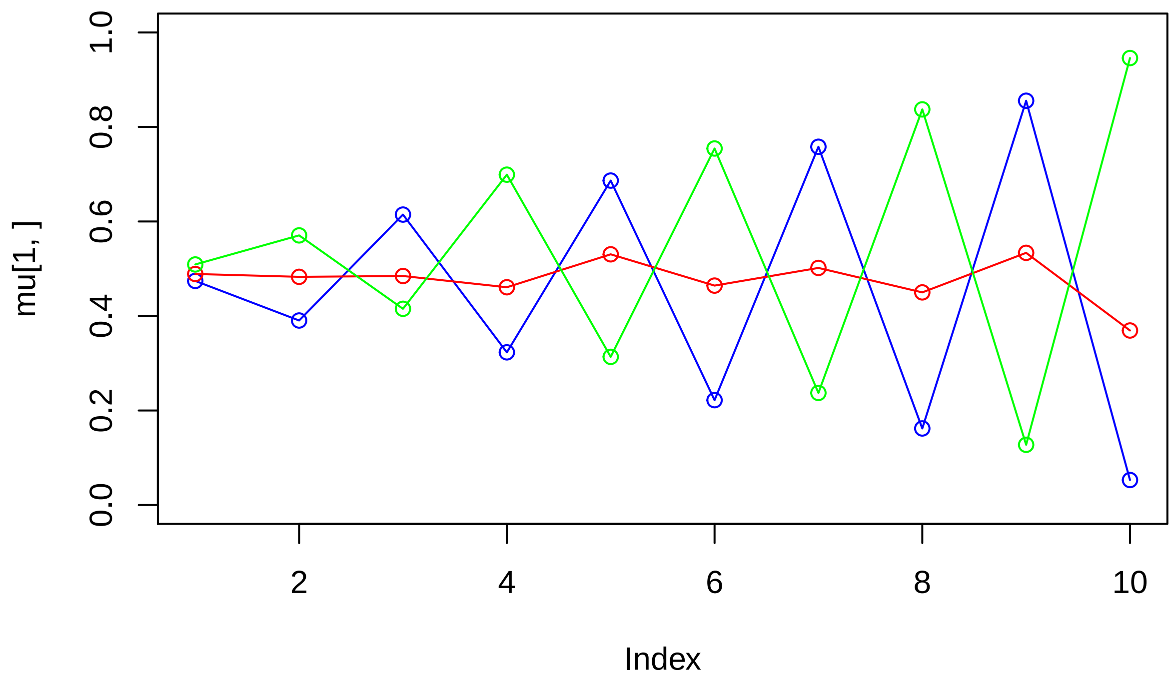

```
## iteration:  7 log likelihood:  -6936.919
```



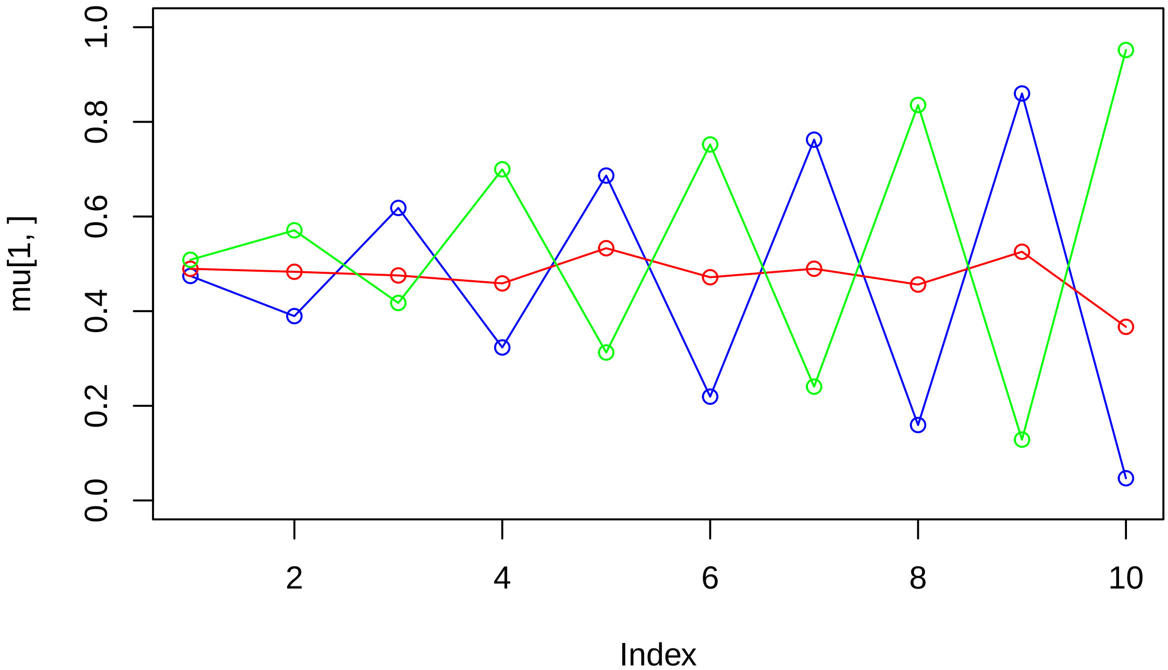

```
## iteration:  8 log likelihood:  -6818.582
```



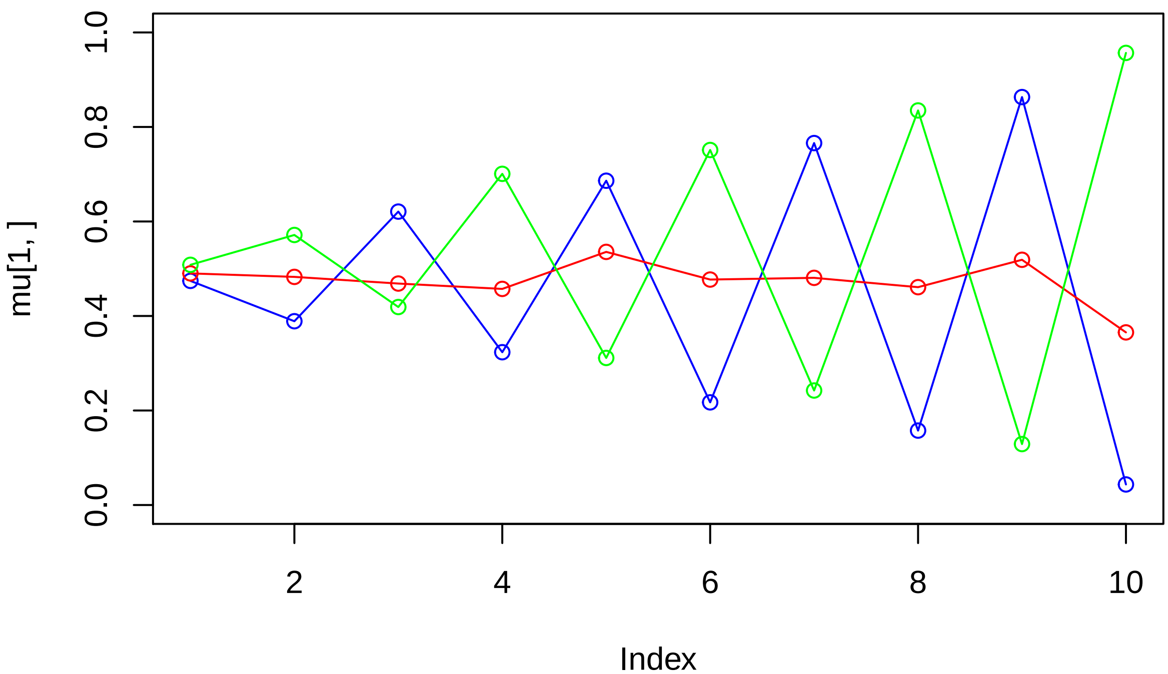

```
## iteration:  9 log likelihood:  -6791.377
```



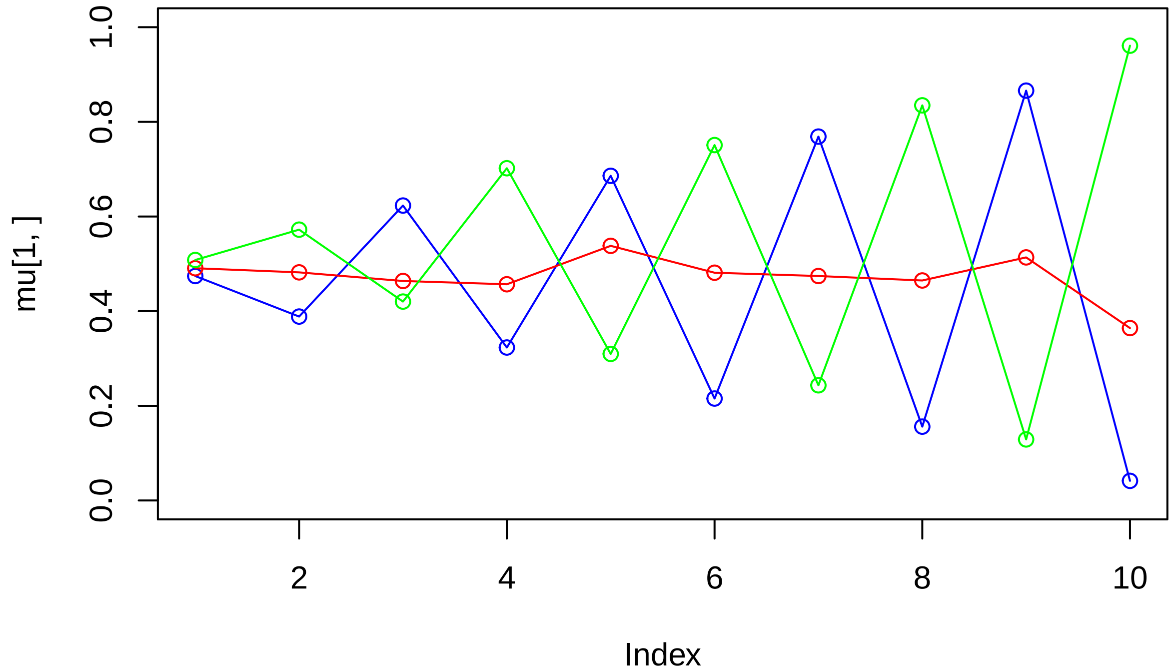

```
## iteration:  10 log likelihood:  -6780.713
```



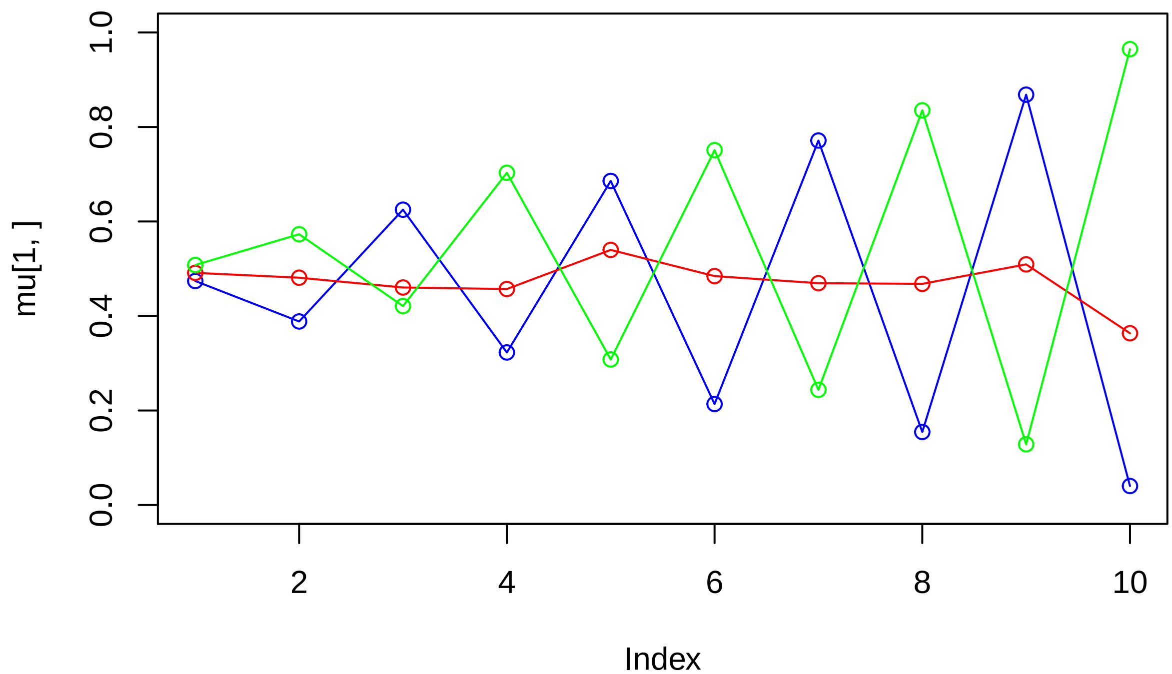

```
## iteration:  11 log likelihood:  -6774.958
```



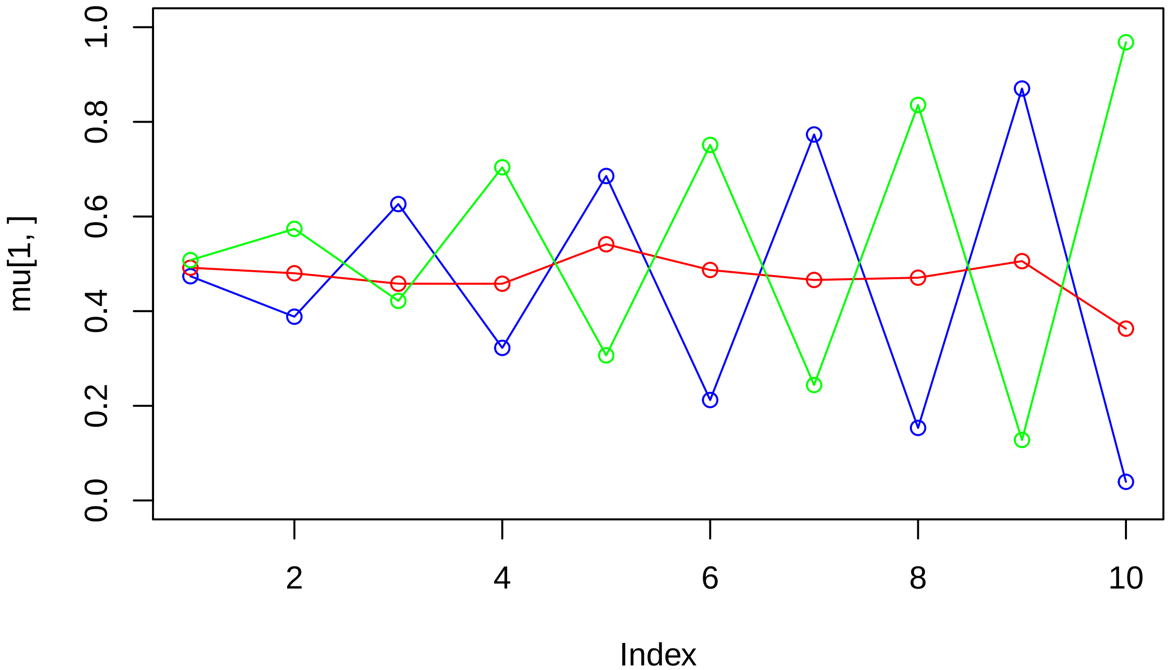

```
## iteration:  12 log likelihood:  -6771.261
```



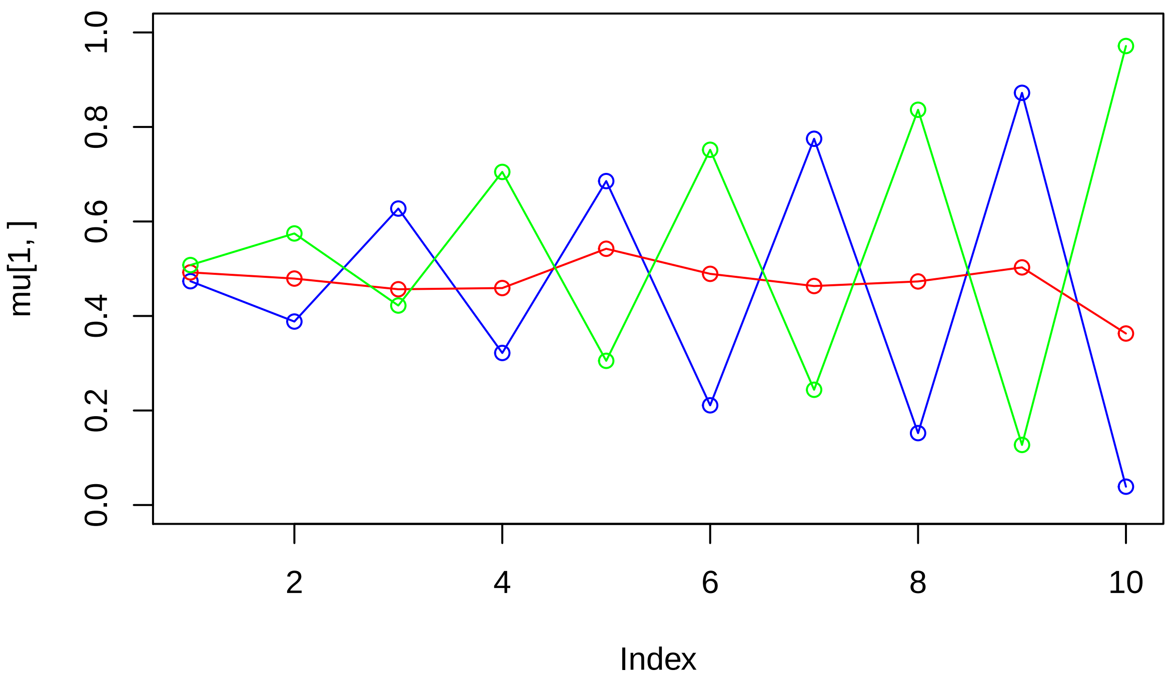

```
## iteration:  13 log likelihood:  -6768.606
```



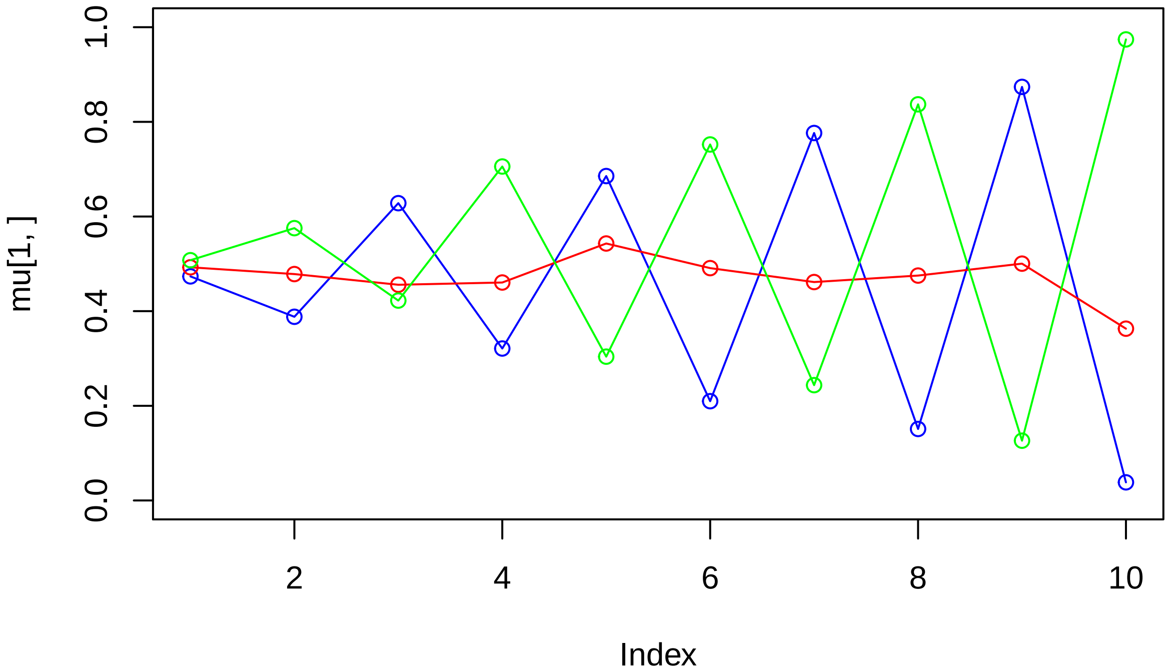

```
## iteration:  14 log likelihood:  -6766.535
```



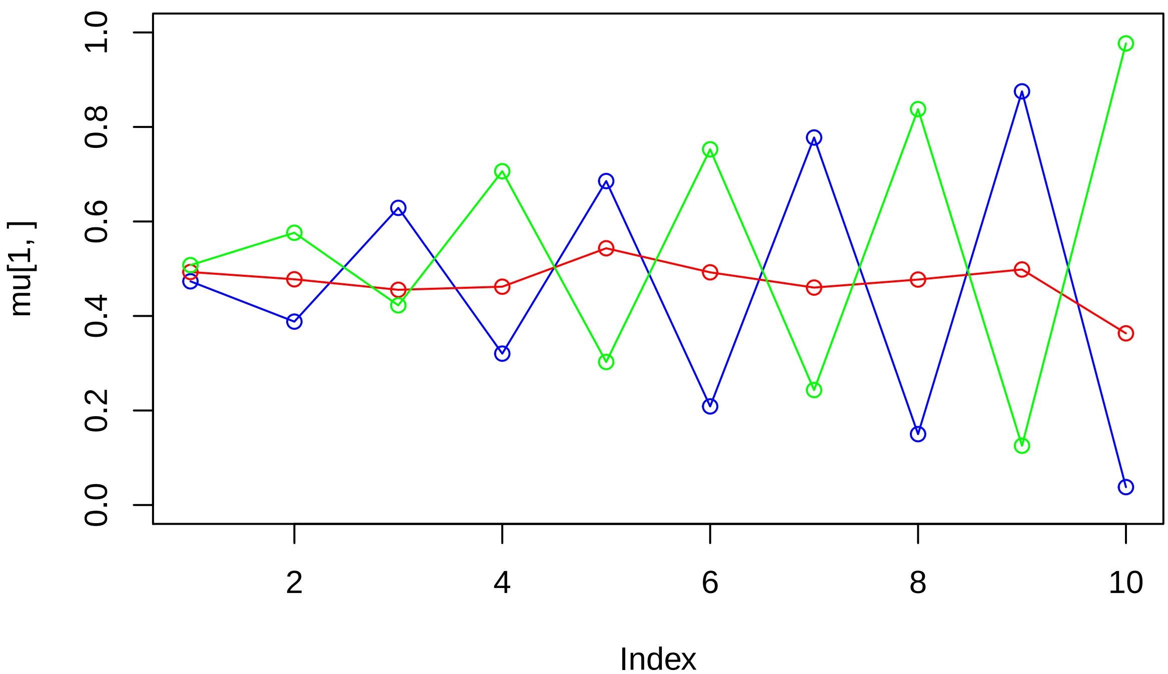

```
## iteration:  15 log likelihood:  -6764.815
```



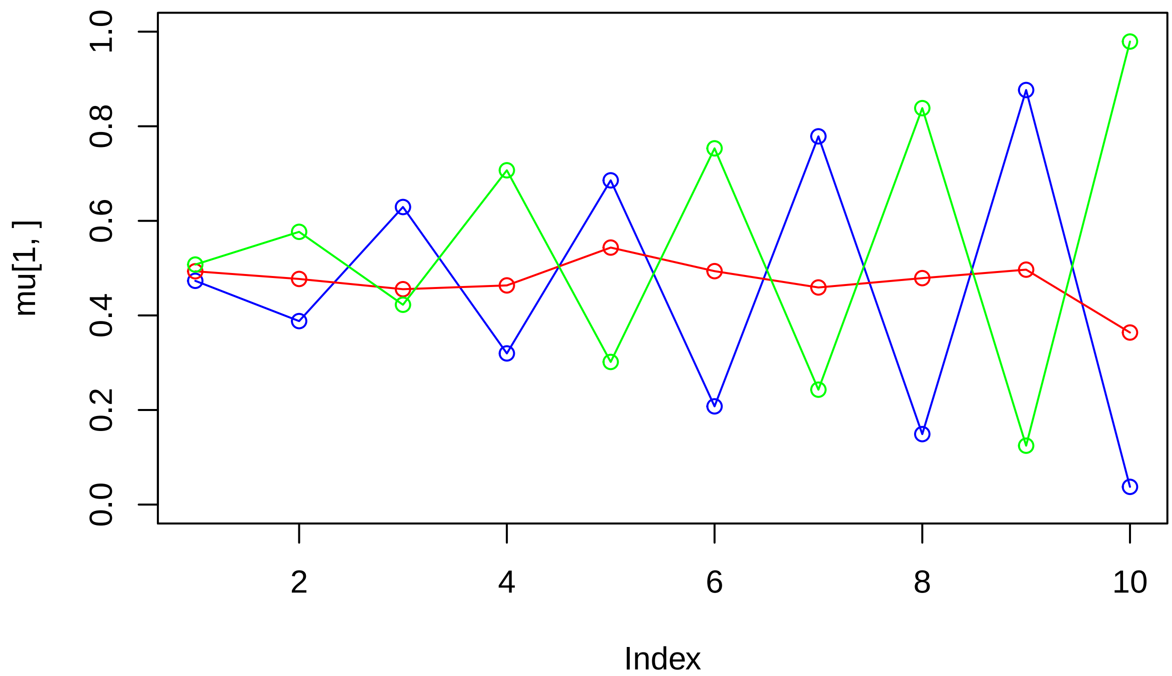

```
## iteration:  16 log likelihood:  -6763.316
```



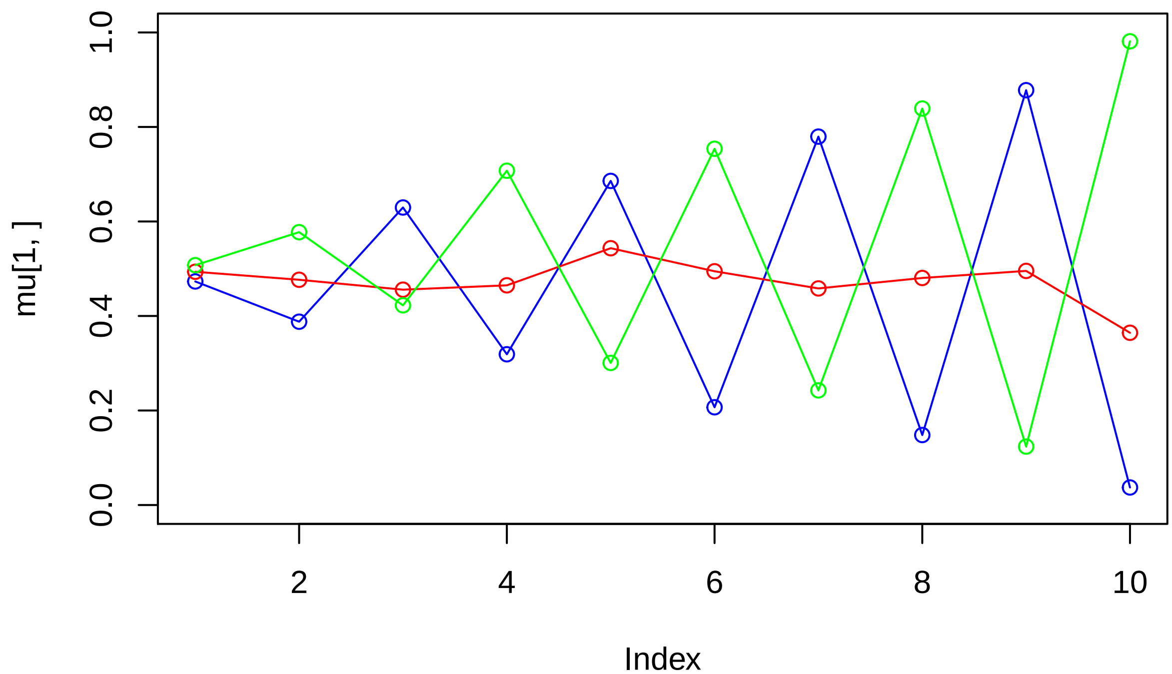

```
## iteration:  17 log likelihood:  -6761.967
```



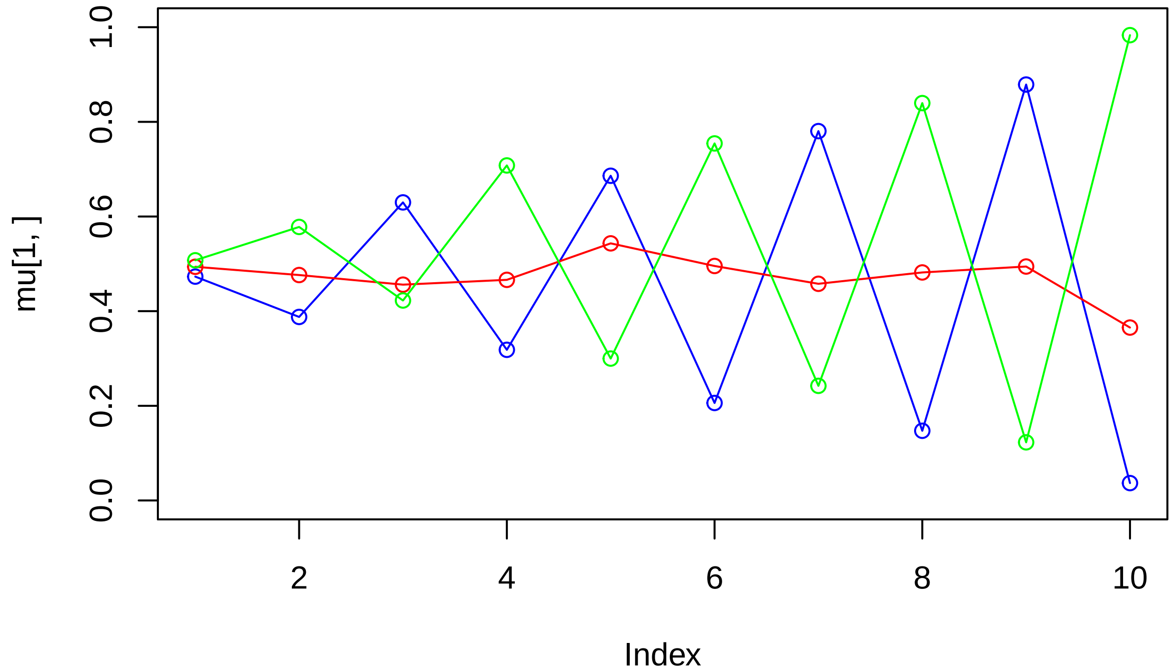

```
## iteration:  18 log likelihood:  -6760.727
```



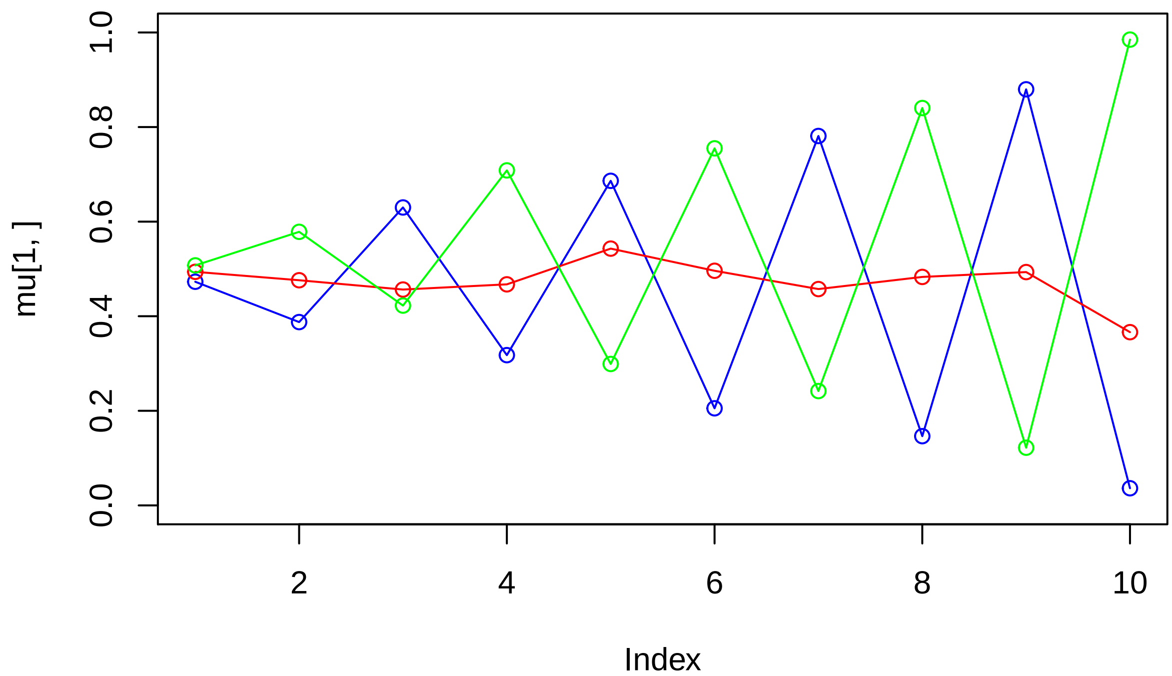

```
## iteration:  19 log likelihood:  -6759.572
```



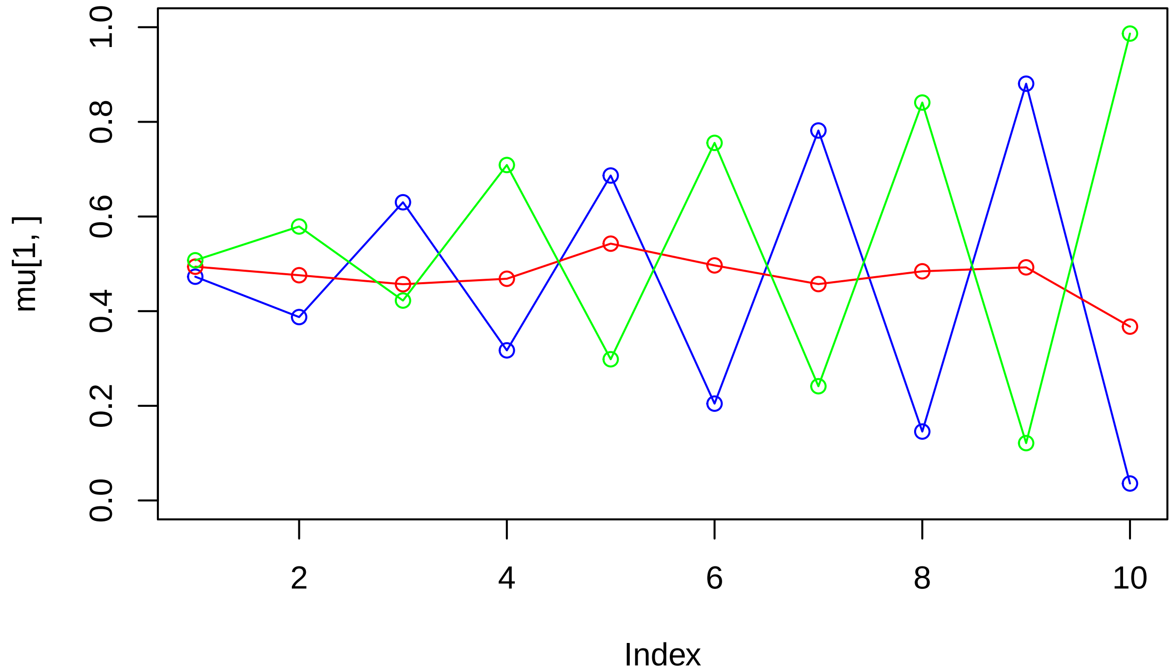

```
## iteration:  20 log likelihood:  -6758.491
```



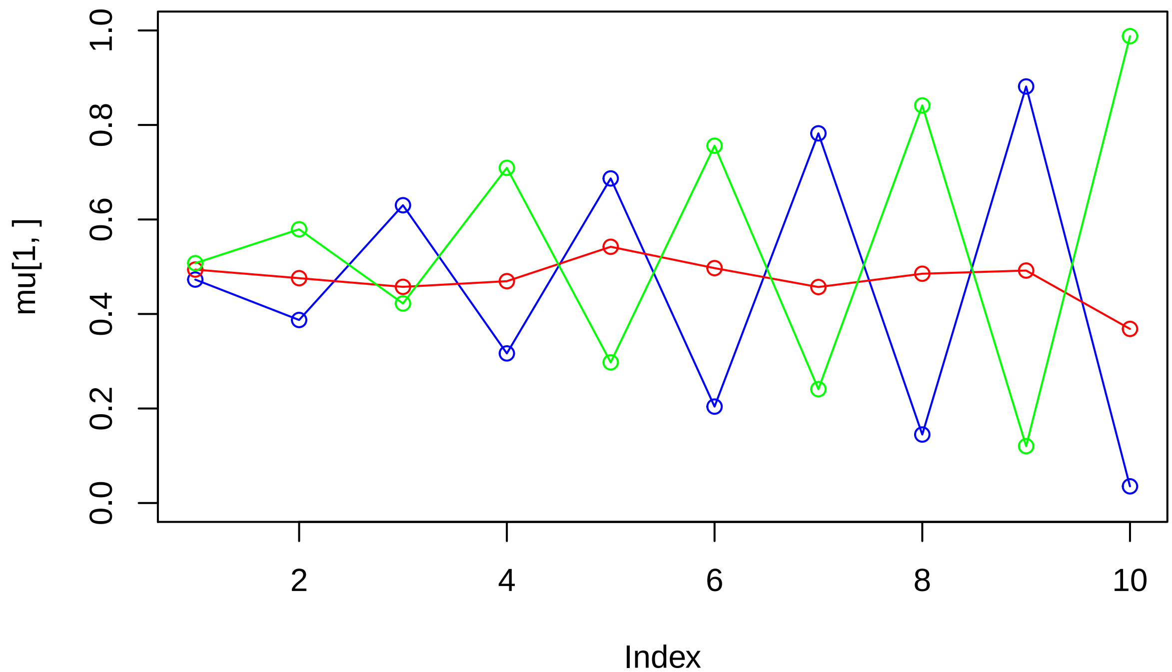

```
## iteration:  21 log likelihood:  -6757.475
```



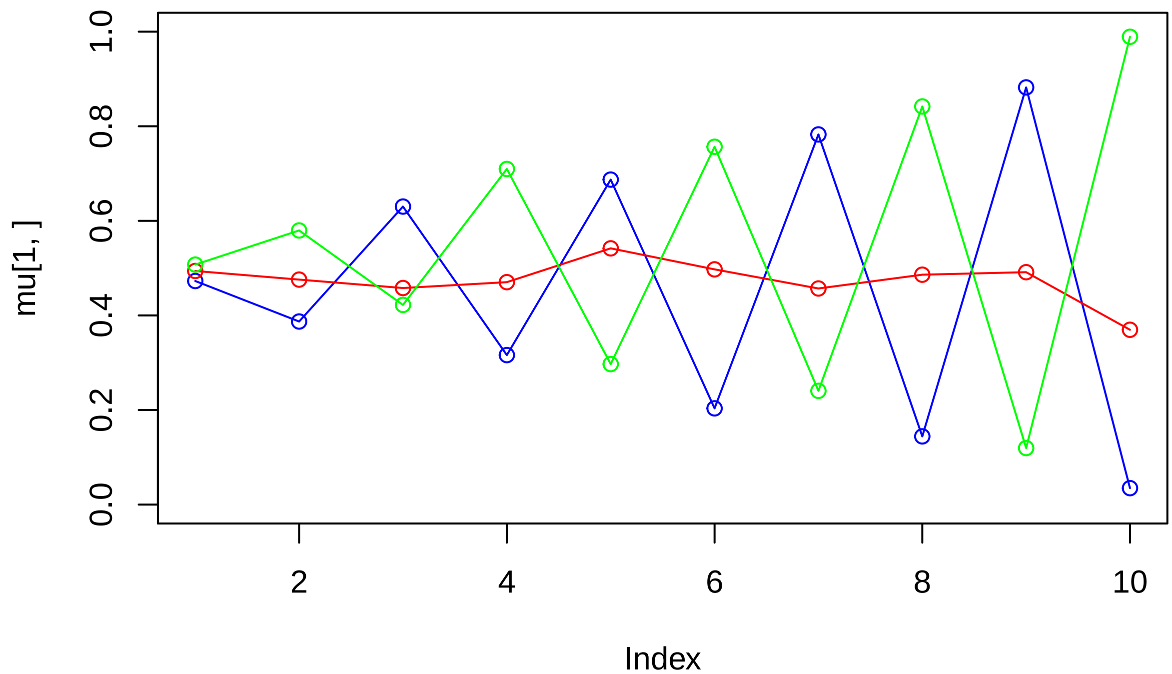

```
## iteration:  22 log likelihood:  -6756.521
```



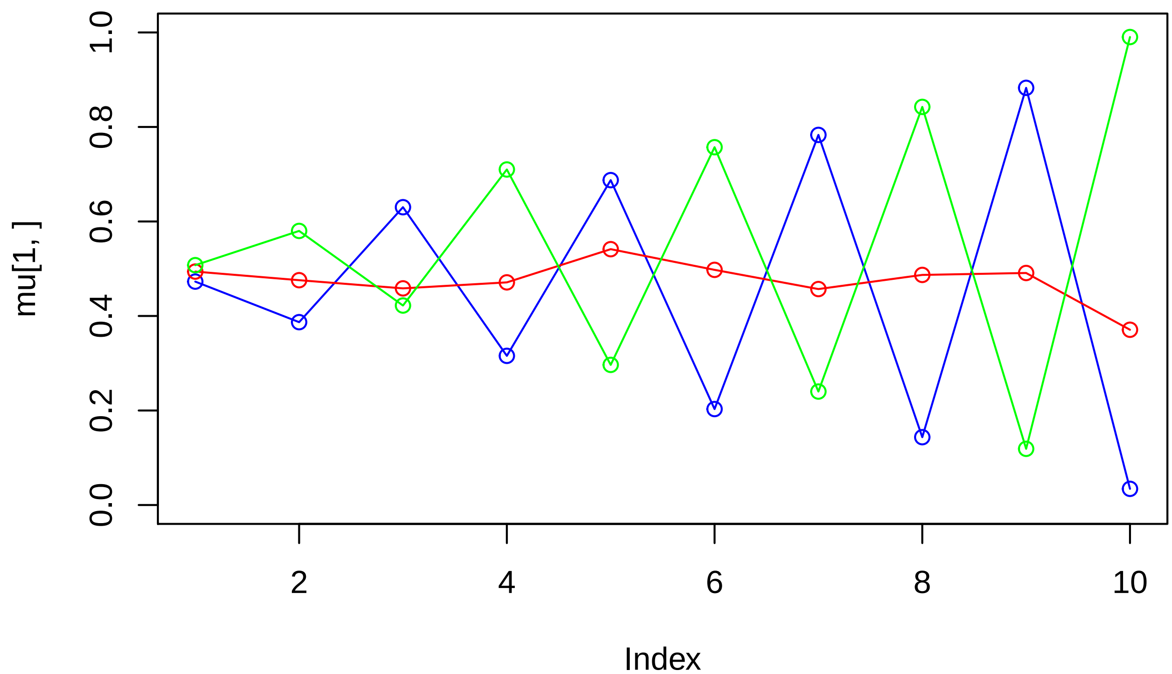

```
## iteration:  23 log likelihood:  -6755.625
```



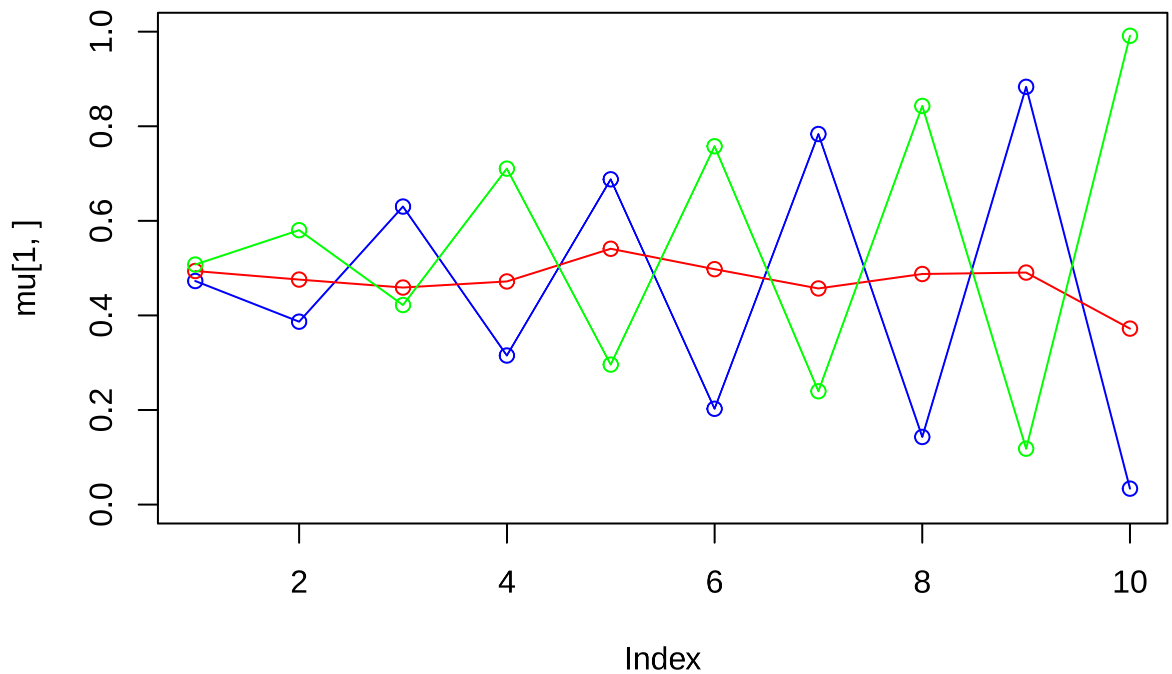

```
## iteration:  24 log likelihood:  -6754.784
```



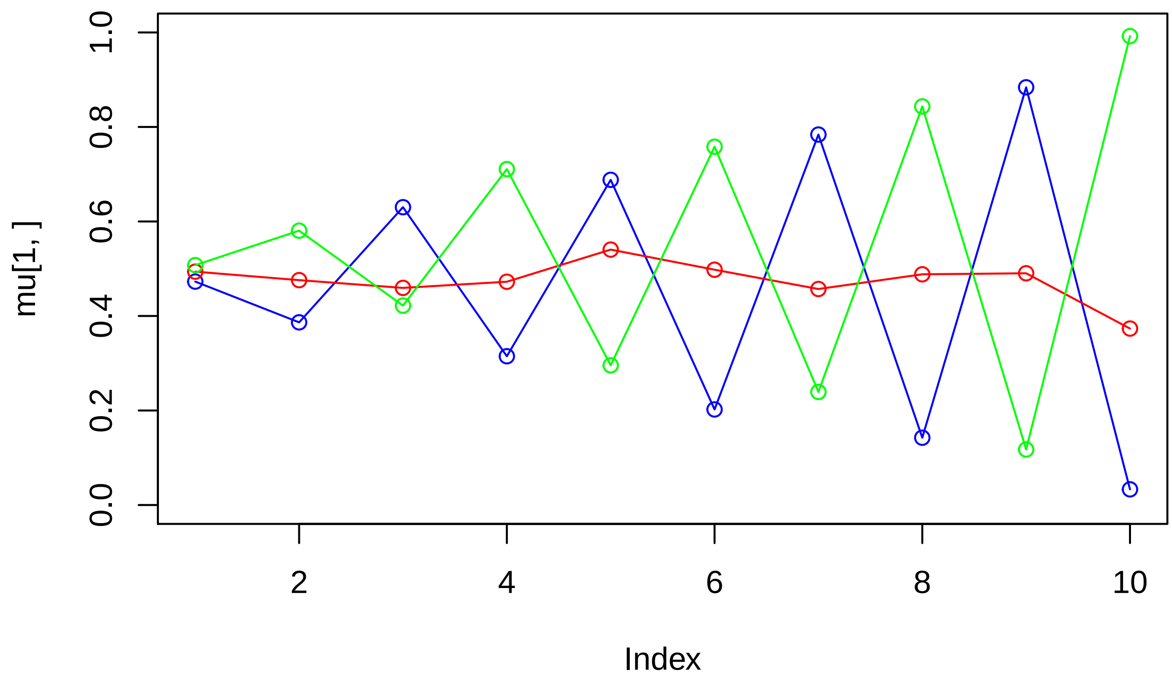

```
## iteration:  25 log likelihood:  -6753.996
```



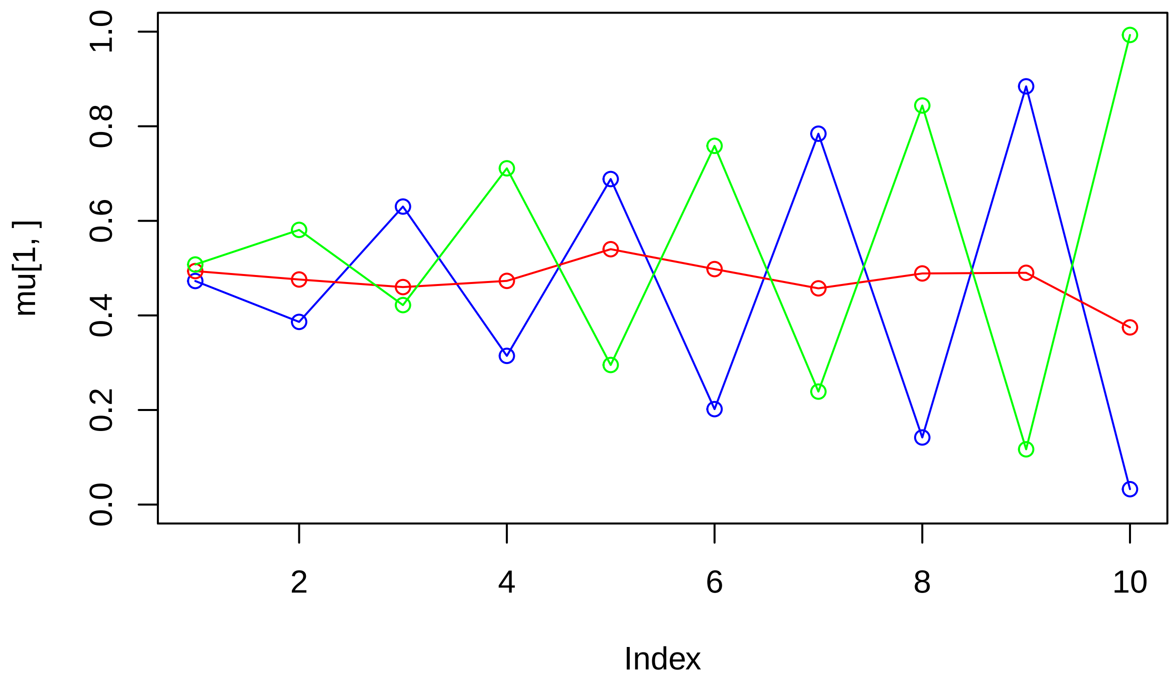

```
## iteration:  26 log likelihood:  -6753.26
```



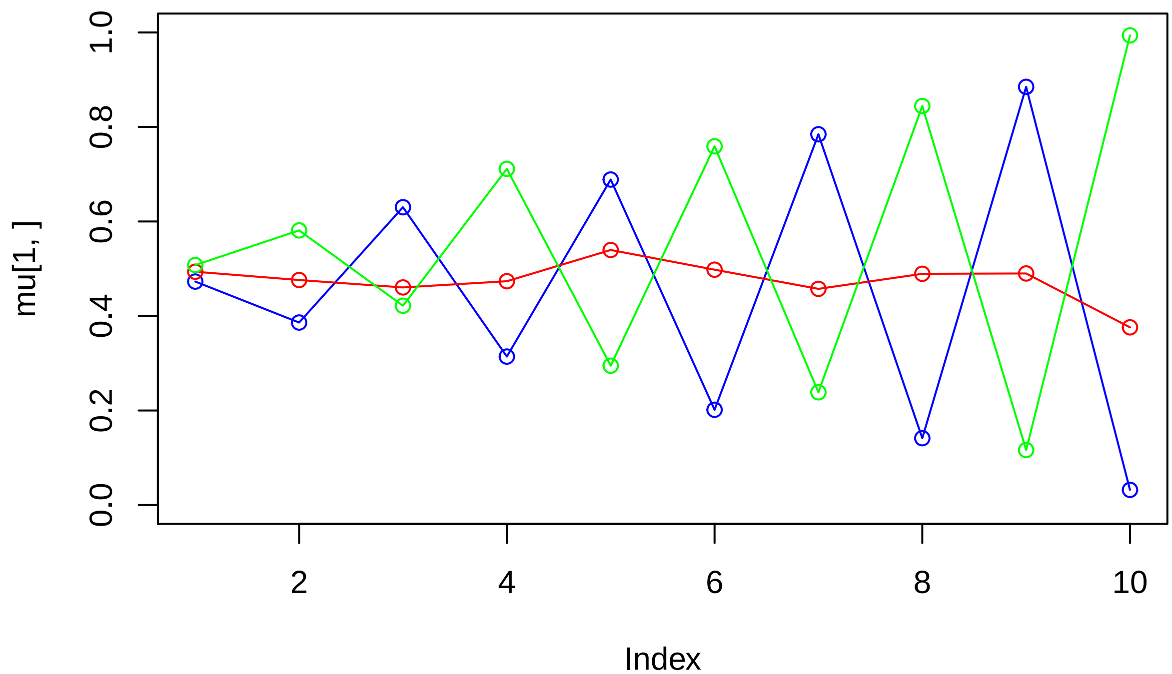

```
## iteration:  27 log likelihood:  -6752.571
```



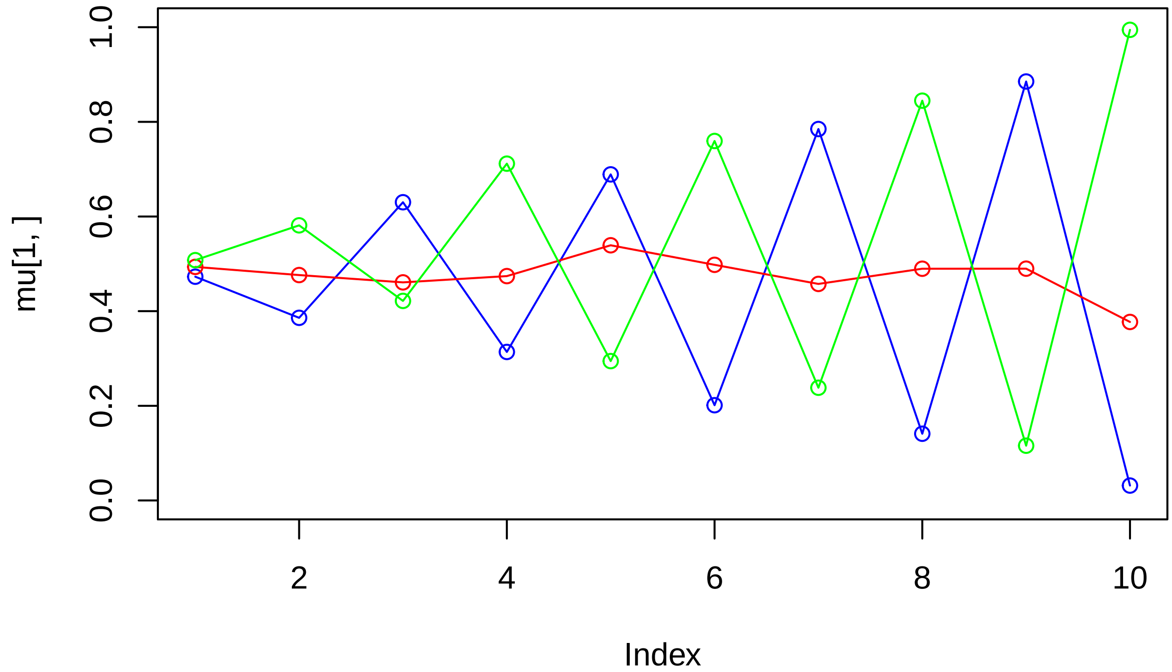

```
## iteration:  28 log likelihood:  -6751.928
```



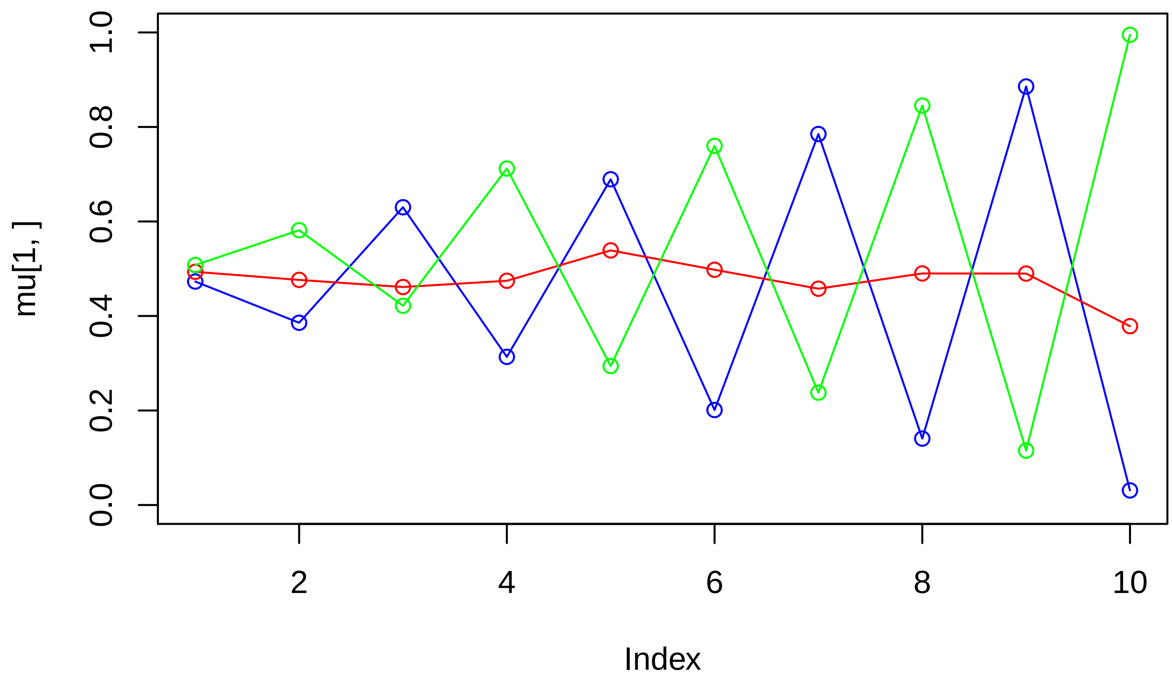

```
## iteration:  29 log likelihood:  -6751.328
```



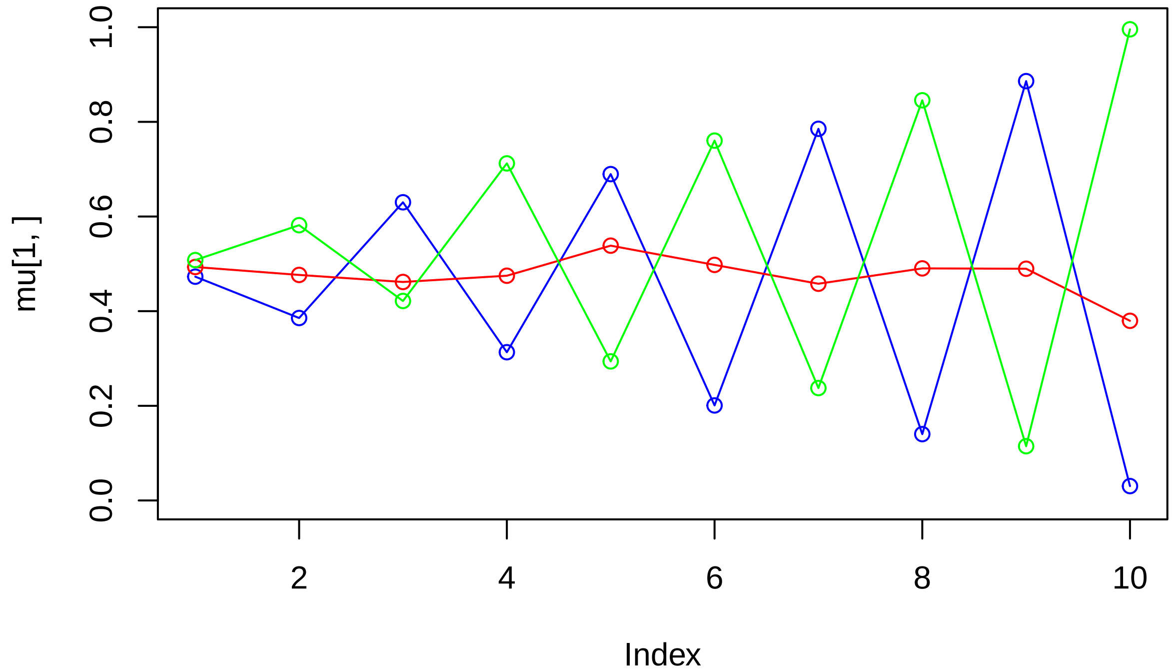

```
## iteration:  30 log likelihood:  -6750.768
```



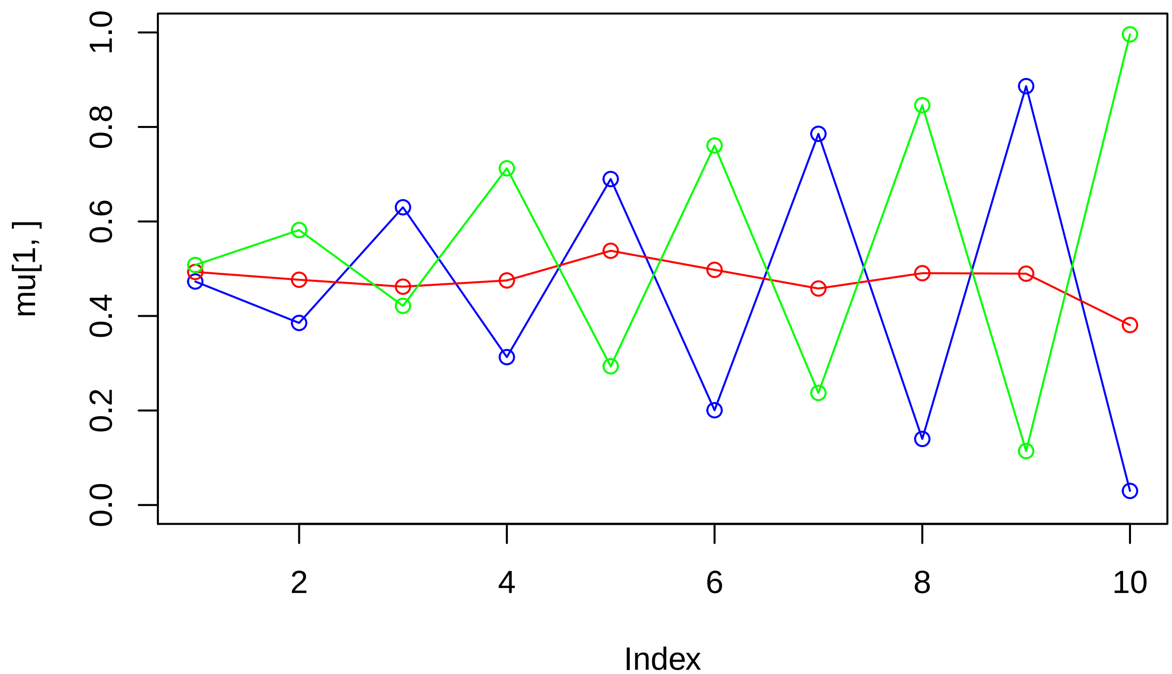

```
## iteration:  31 log likelihood:  -6750.246
```



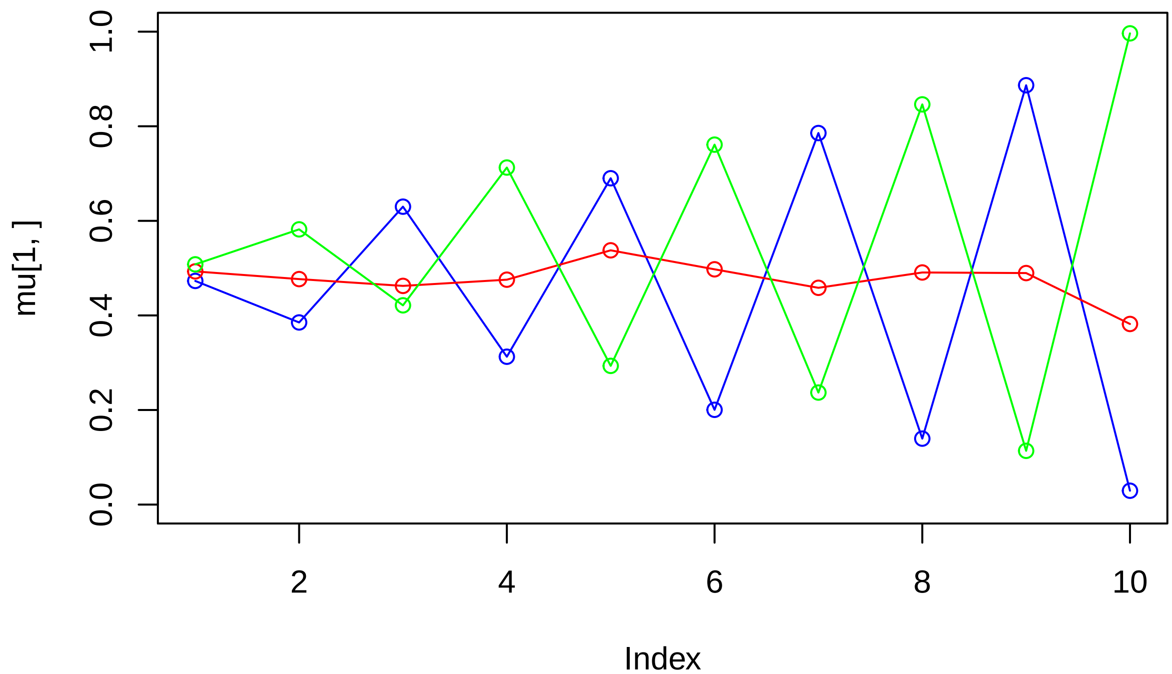

```
## iteration:  32 log likelihood:  -6749.758
```



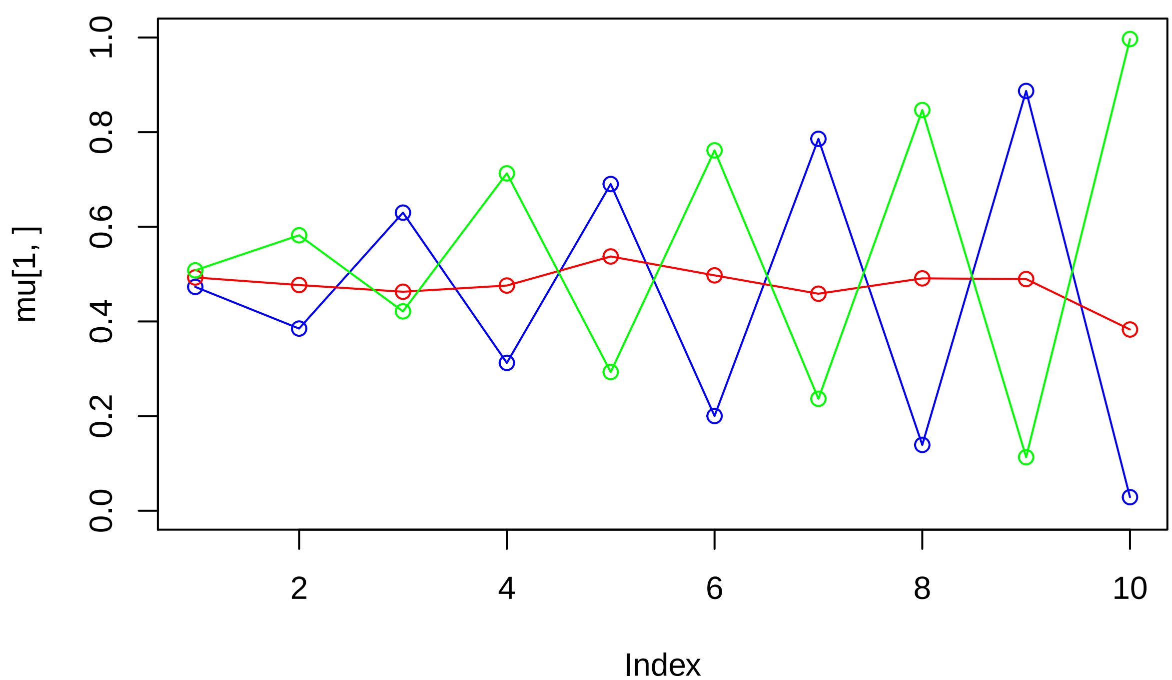

```
## iteration:  33 log likelihood:  -6749.304
```



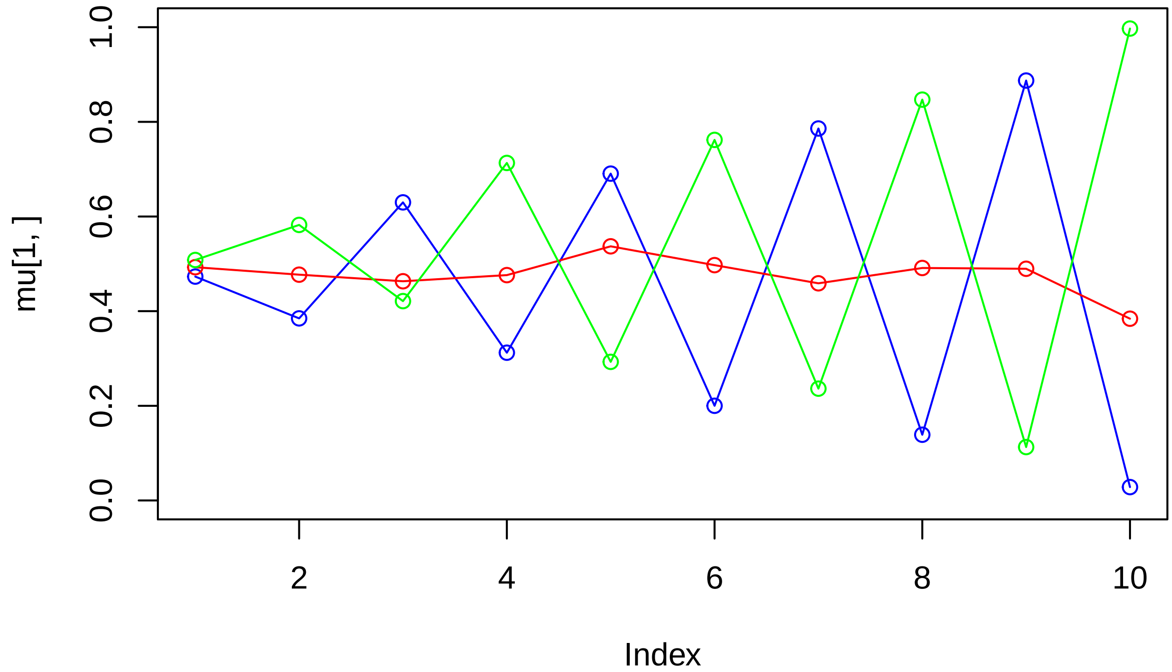

```
## iteration:  34 log likelihood:  -6748.88
```



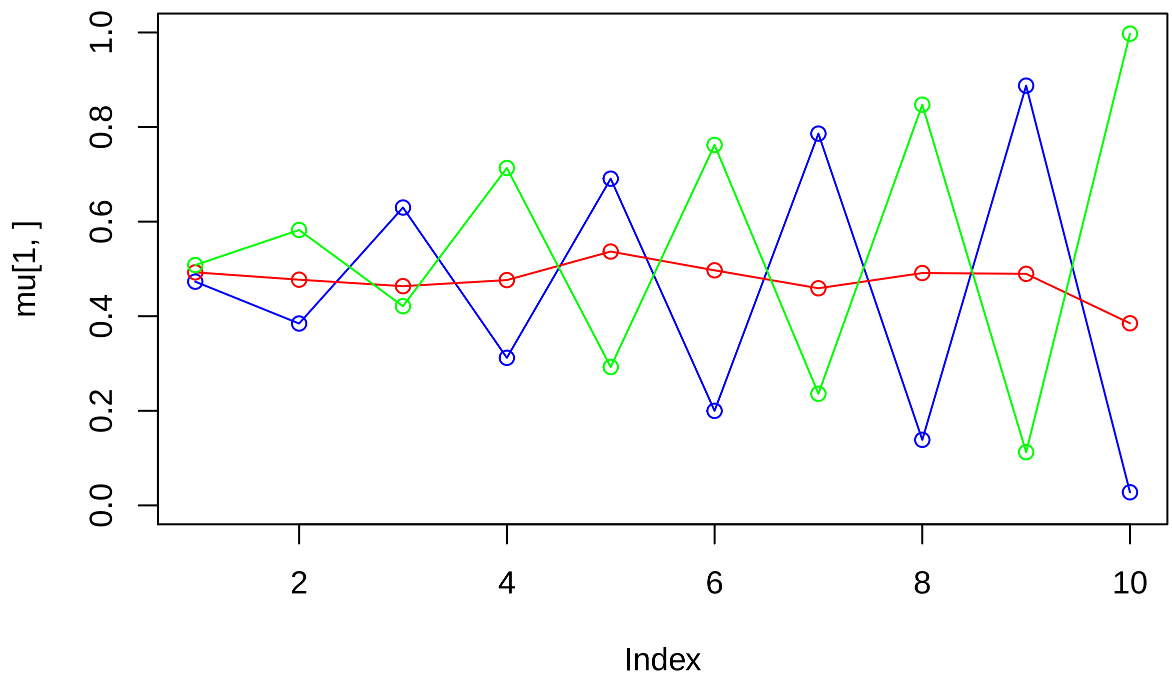

```
## iteration:  35 log likelihood:  -6748.484
```



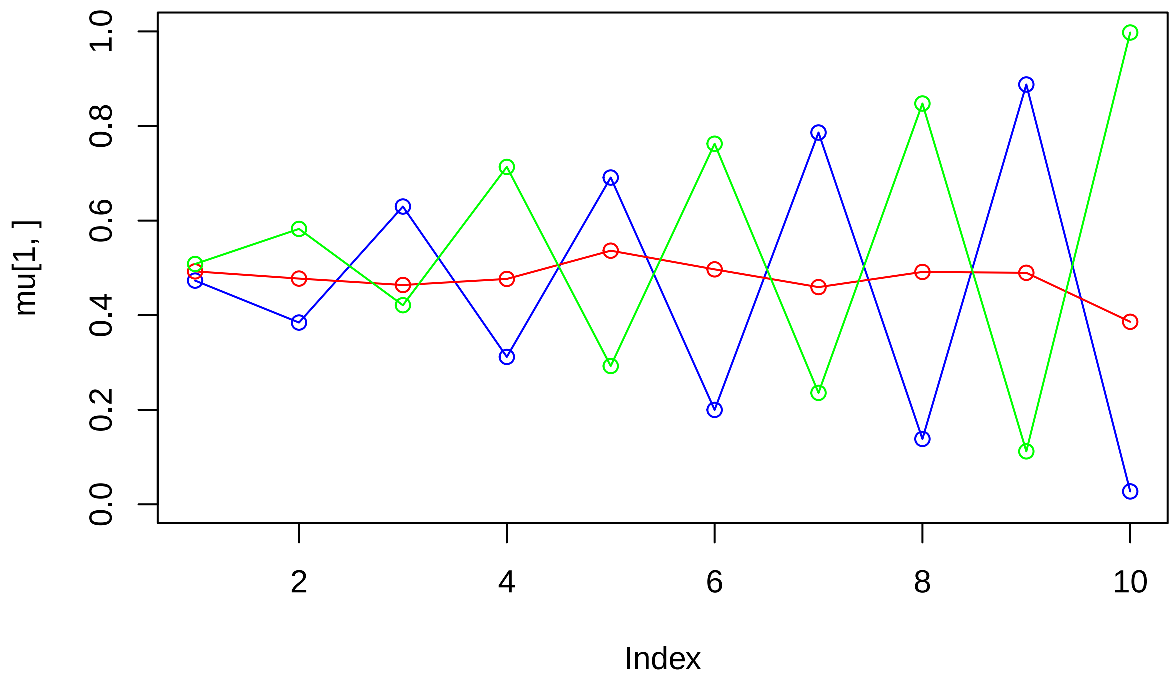

```
## iteration:  36 log likelihood:  -6748.114
```



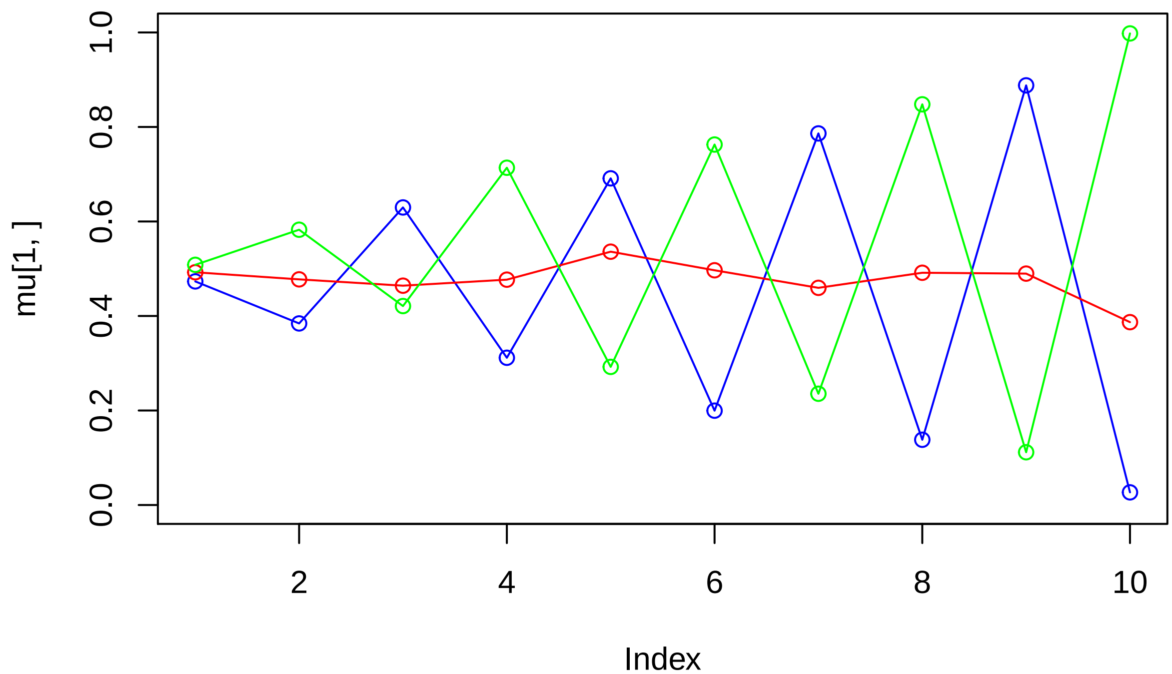

```
## iteration:  37 log likelihood:  -6747.767
```



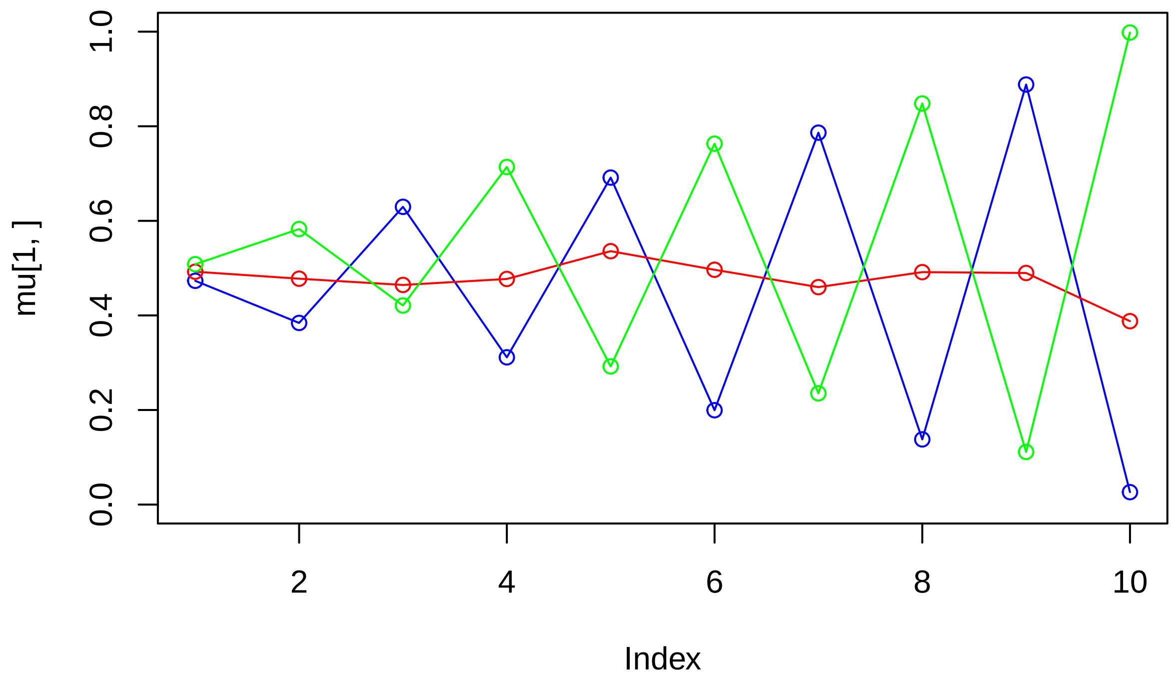

```
## iteration:  38 log likelihood:  -6747.444
```



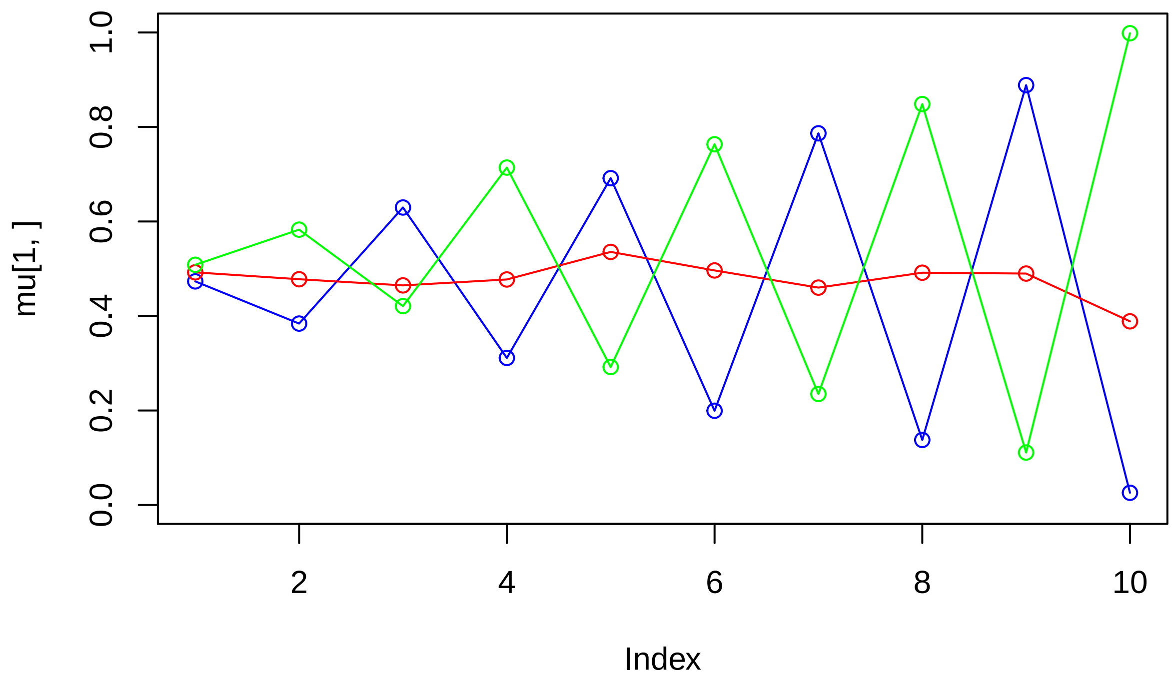

```
## iteration:  39 log likelihood:  -6747.14
```



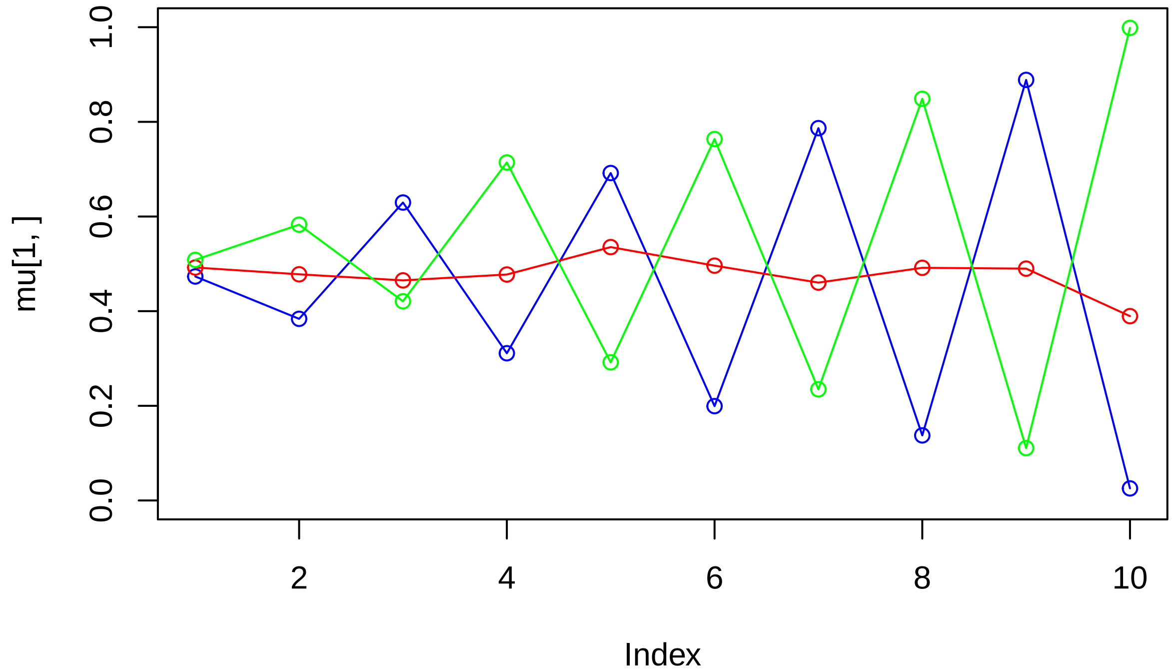

```
## iteration:  40 log likelihood:  -6746.856
```



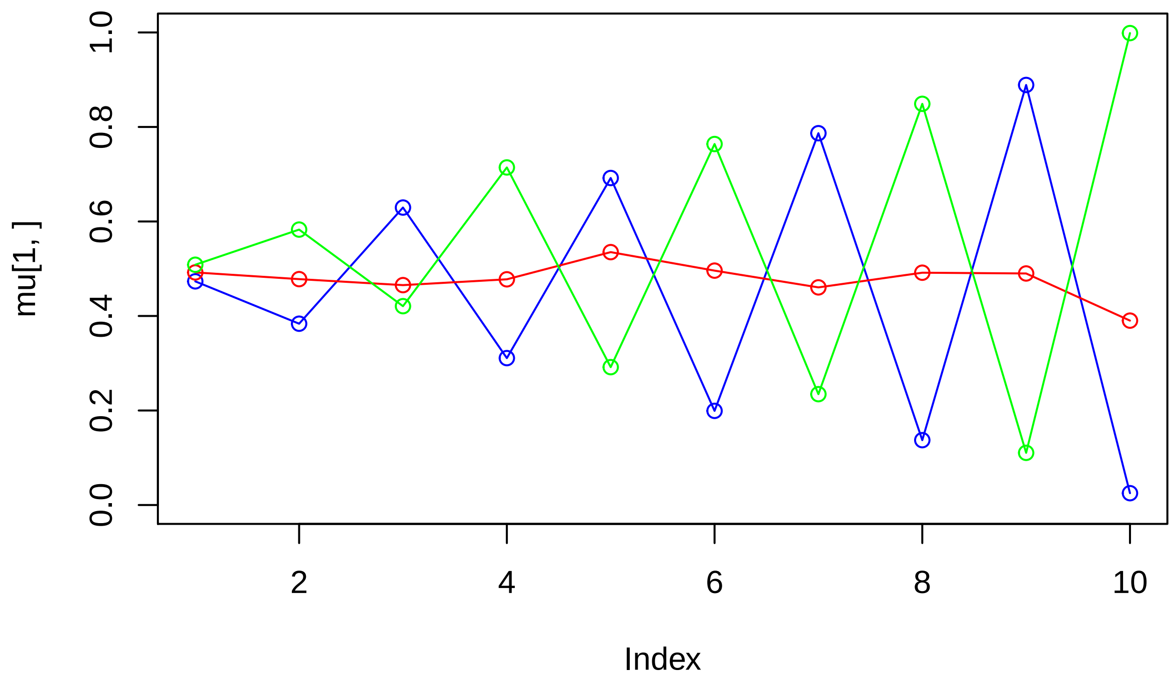

```
## iteration:  41 log likelihood:  -6746.589
```



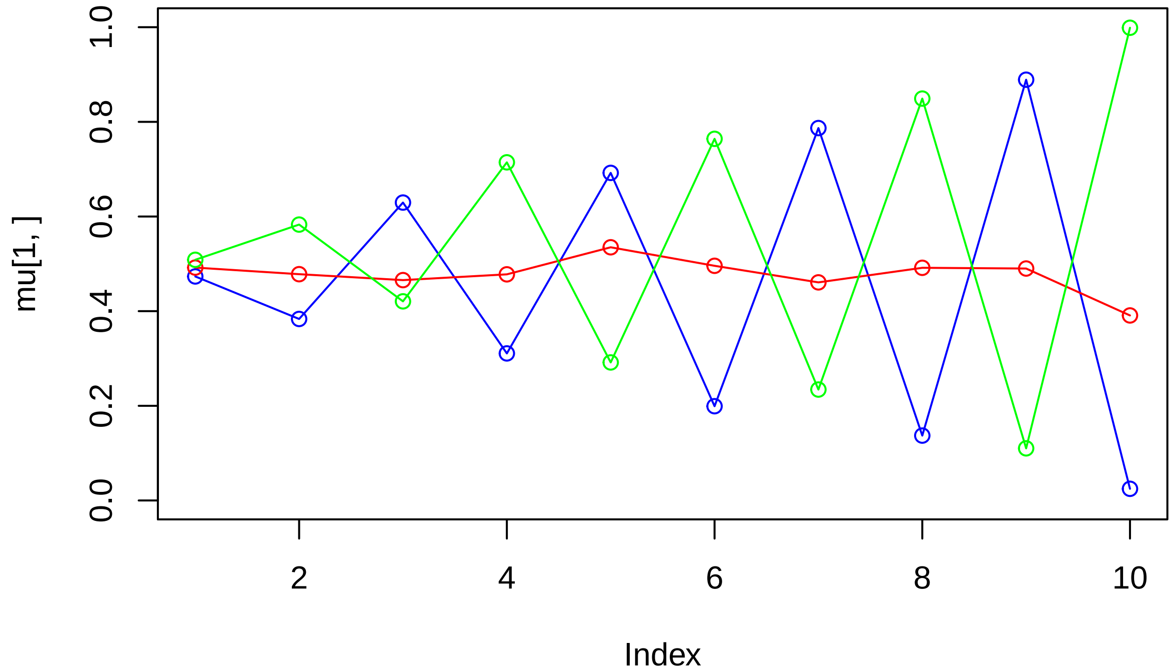

```
## iteration:  42 log likelihood:  -6746.338
```



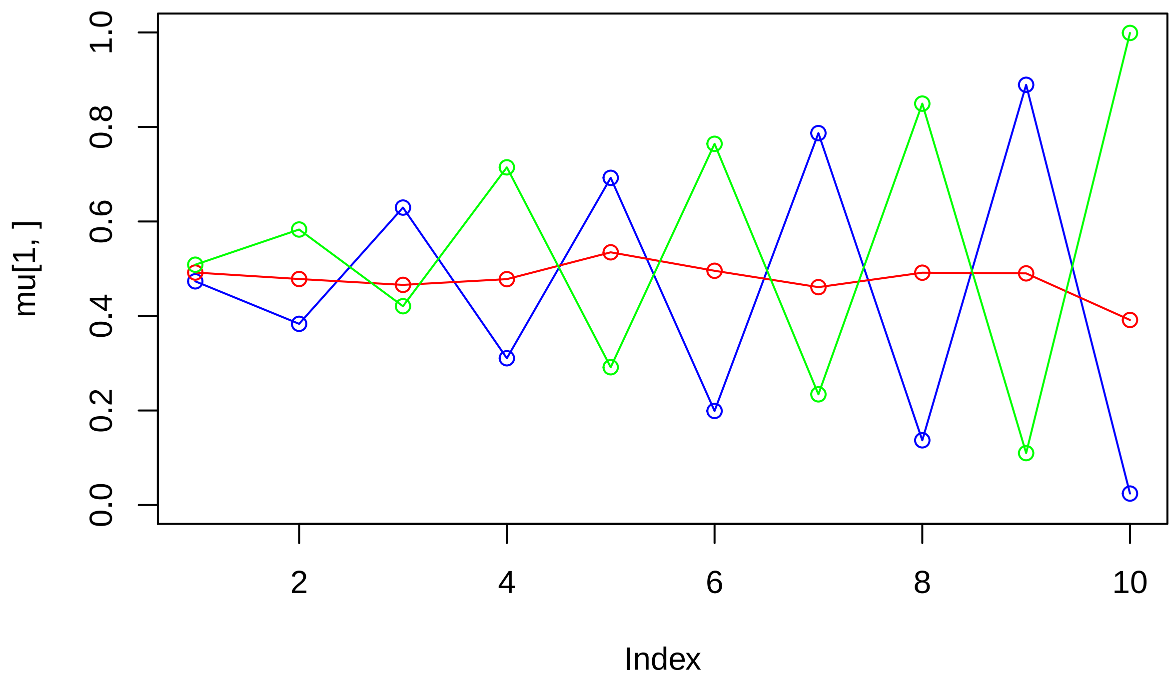

```
## iteration:  43 log likelihood:  -6746.102
```



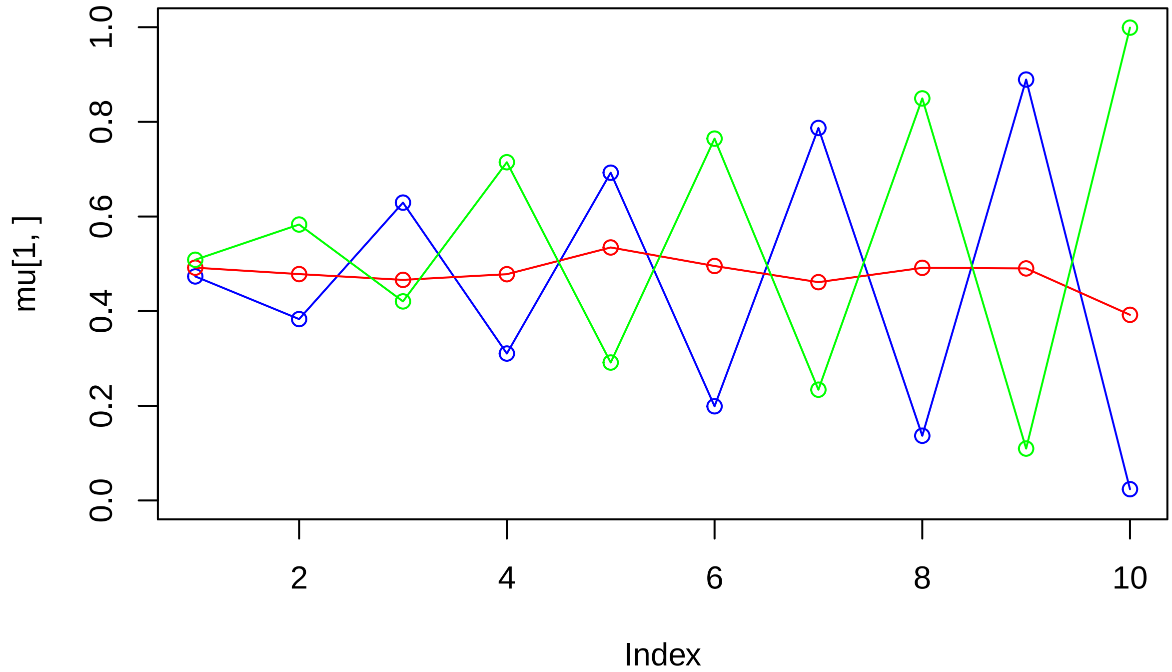

```
## iteration:  44 log likelihood:  -6745.88
```



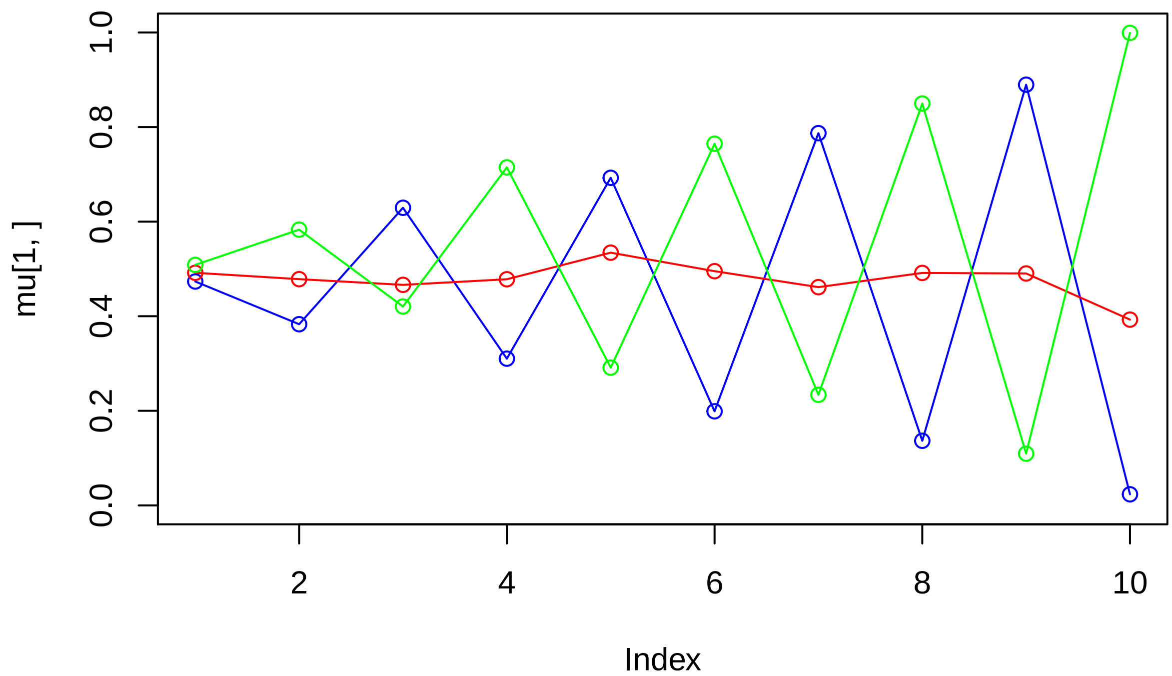
iteration: 45 log likelihood: -6745.67



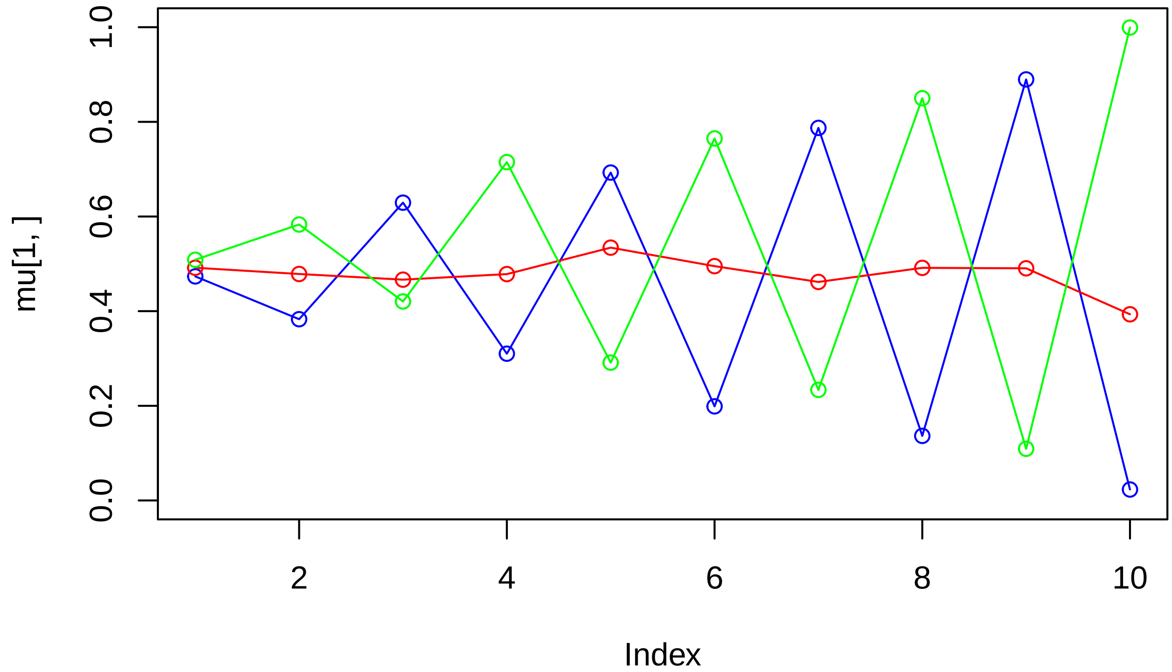
iteration: 46 log likelihood: -6745.472



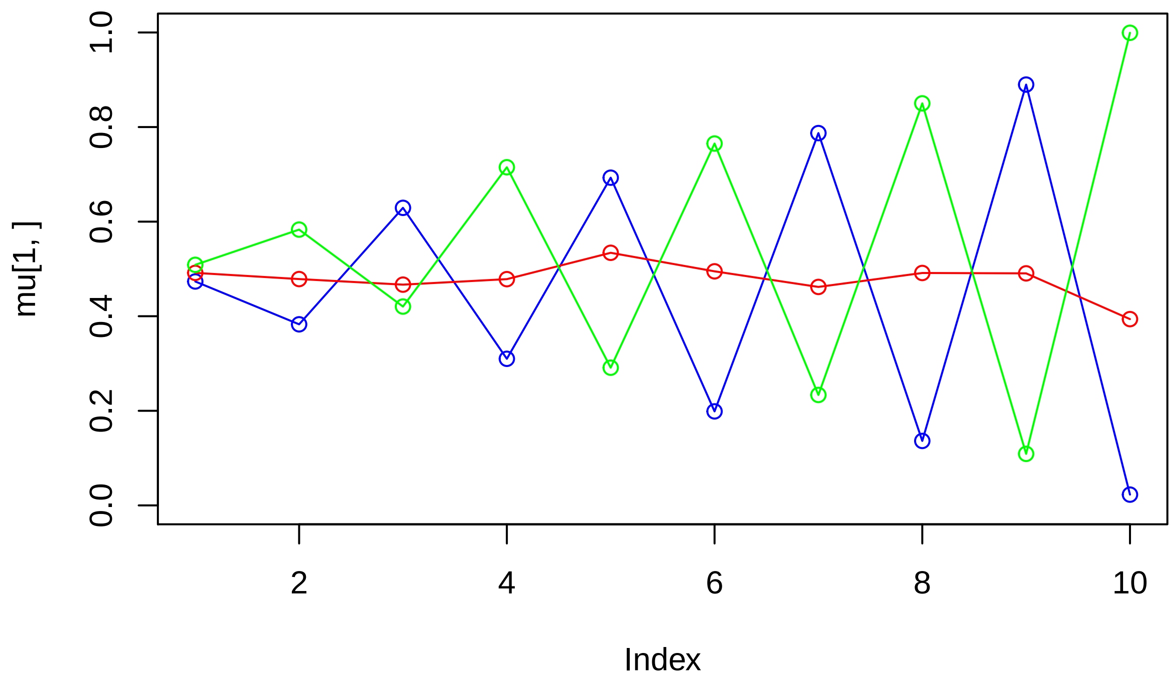

```
## iteration:  47 log likelihood:  -6745.285
```



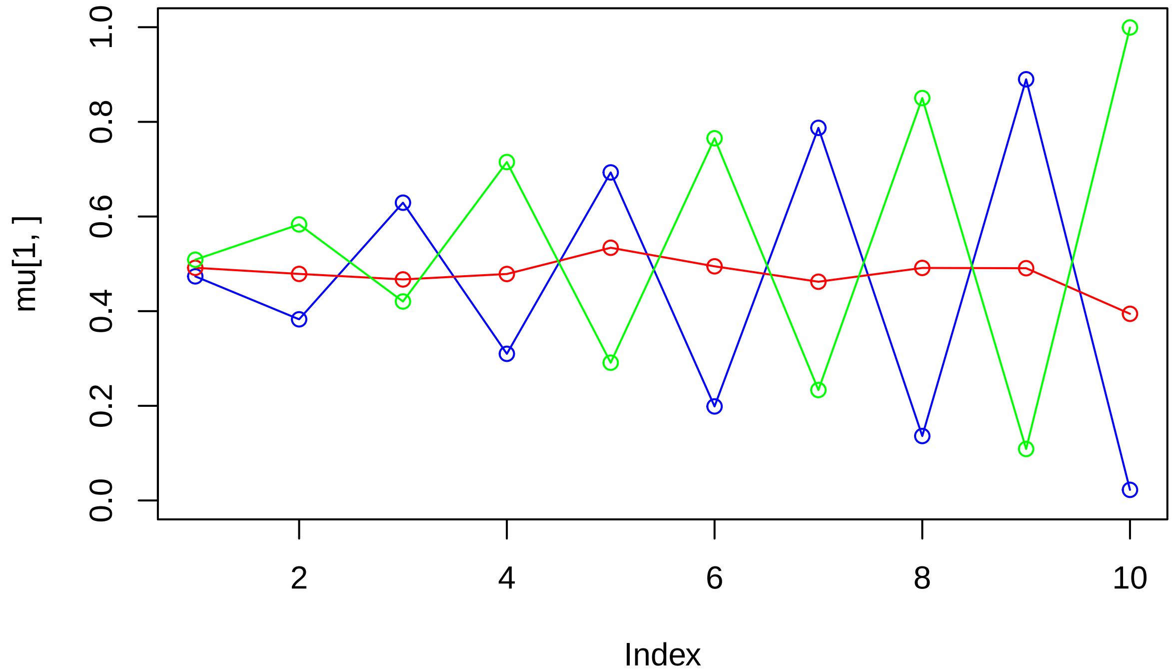

```
## iteration:  48 log likelihood:  -6745.108
```



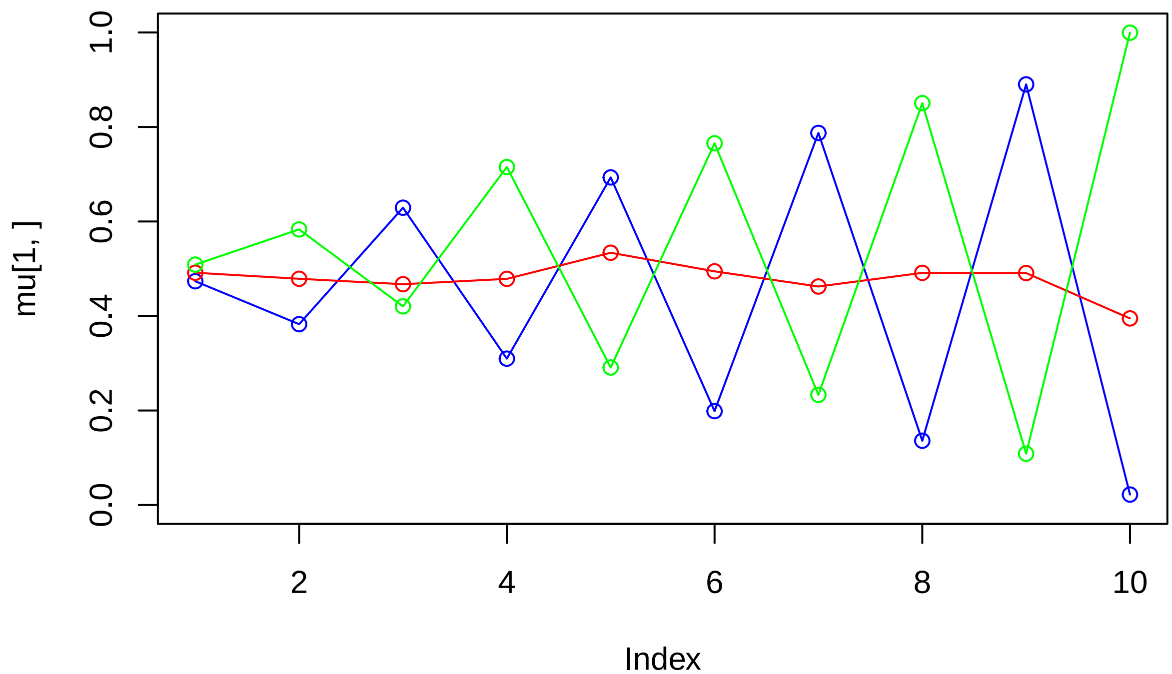

```
## iteration:  49 log likelihood:  -6744.939
```



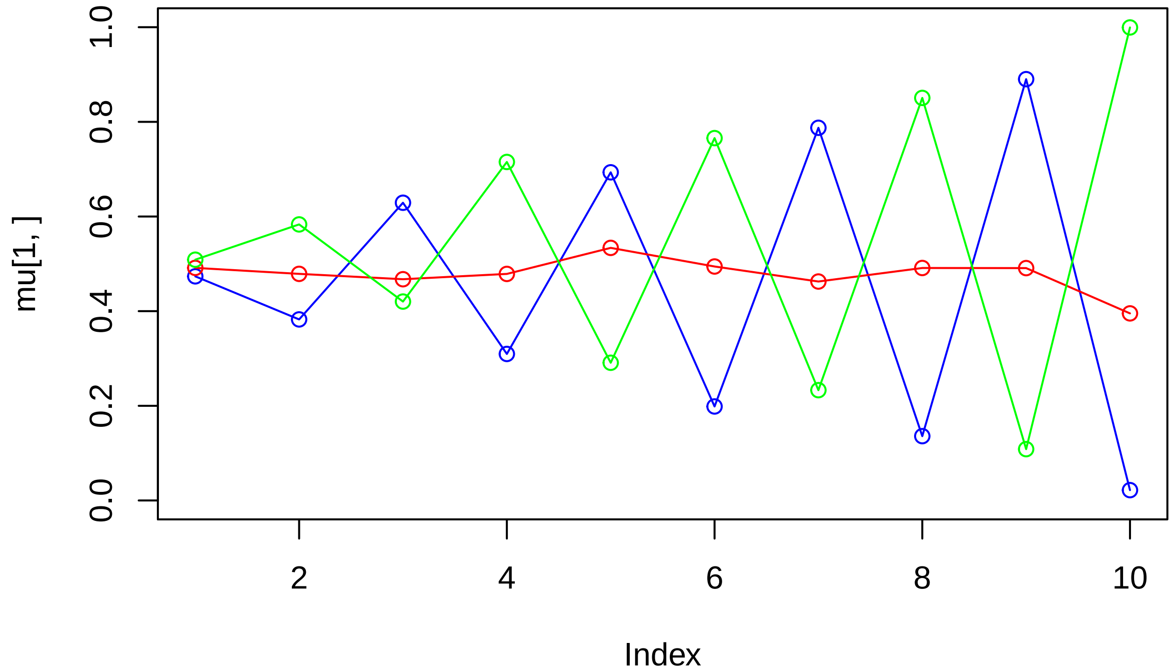

```
## iteration:  50 log likelihood:  -6744.78
```



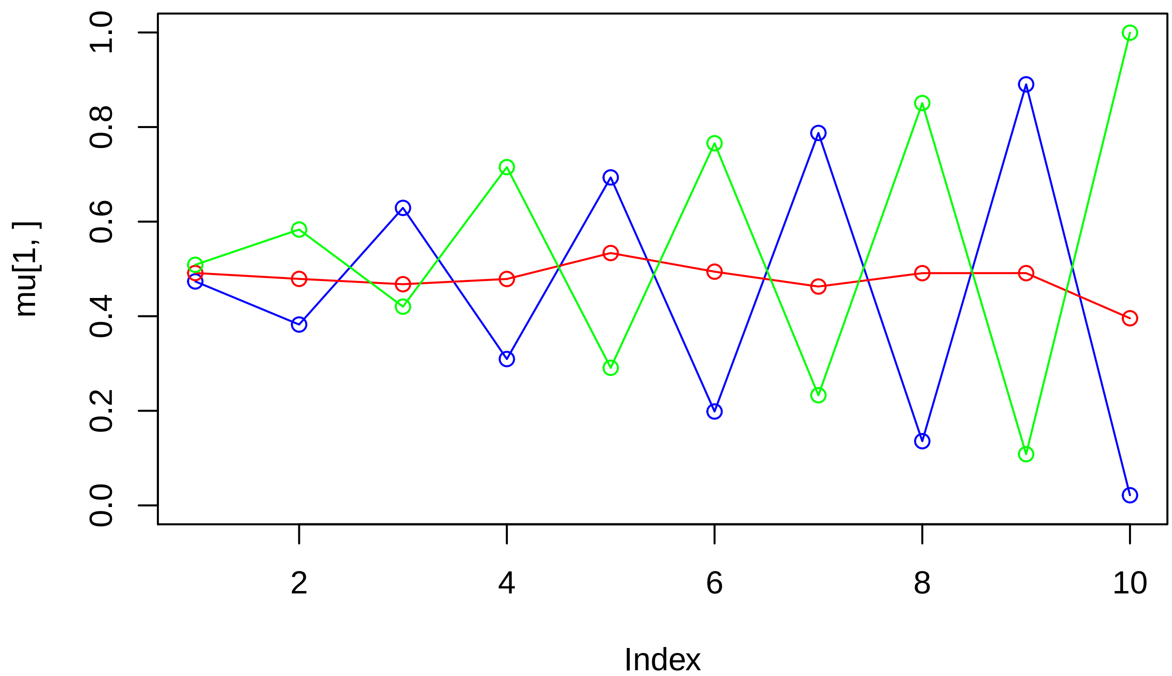

```
## iteration:  51 log likelihood:  -6744.627
```



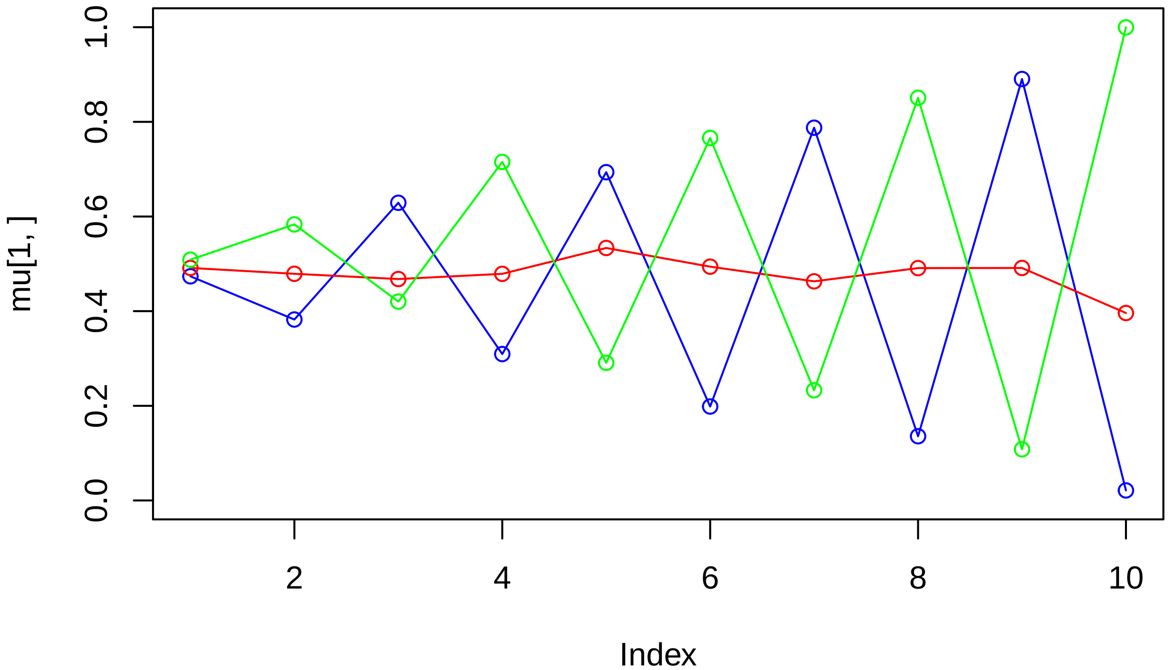

```
## iteration:  52 log likelihood:  -6744.483
```



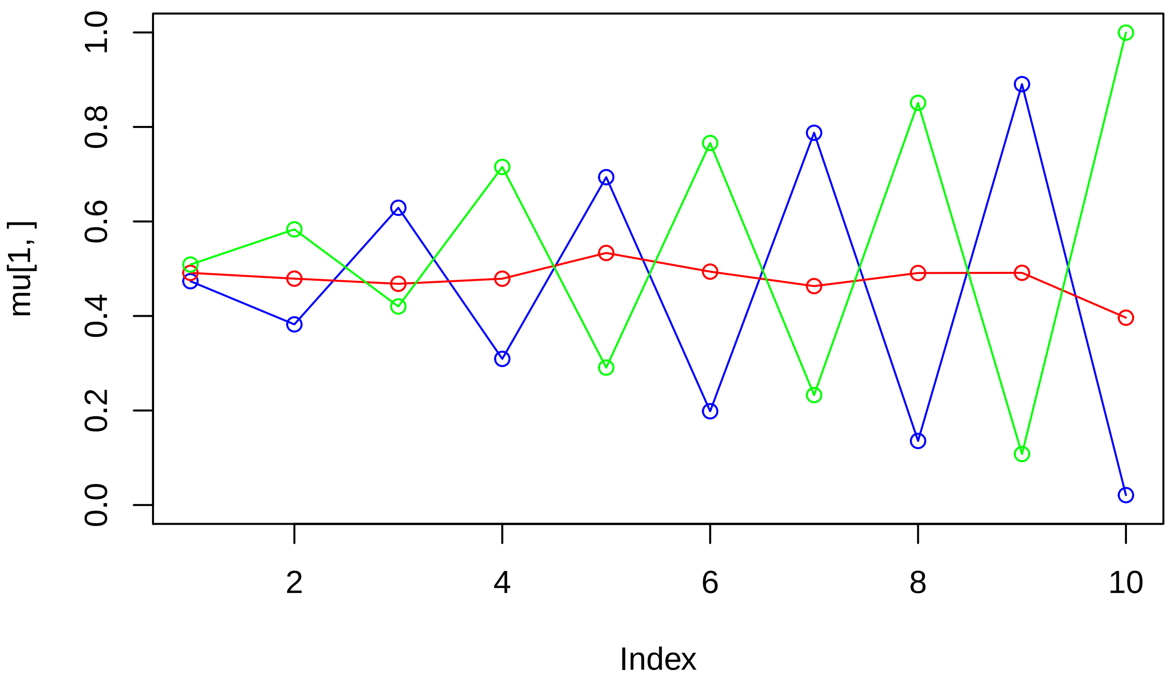

```
## iteration:  53 log likelihood:  -6744.344
```



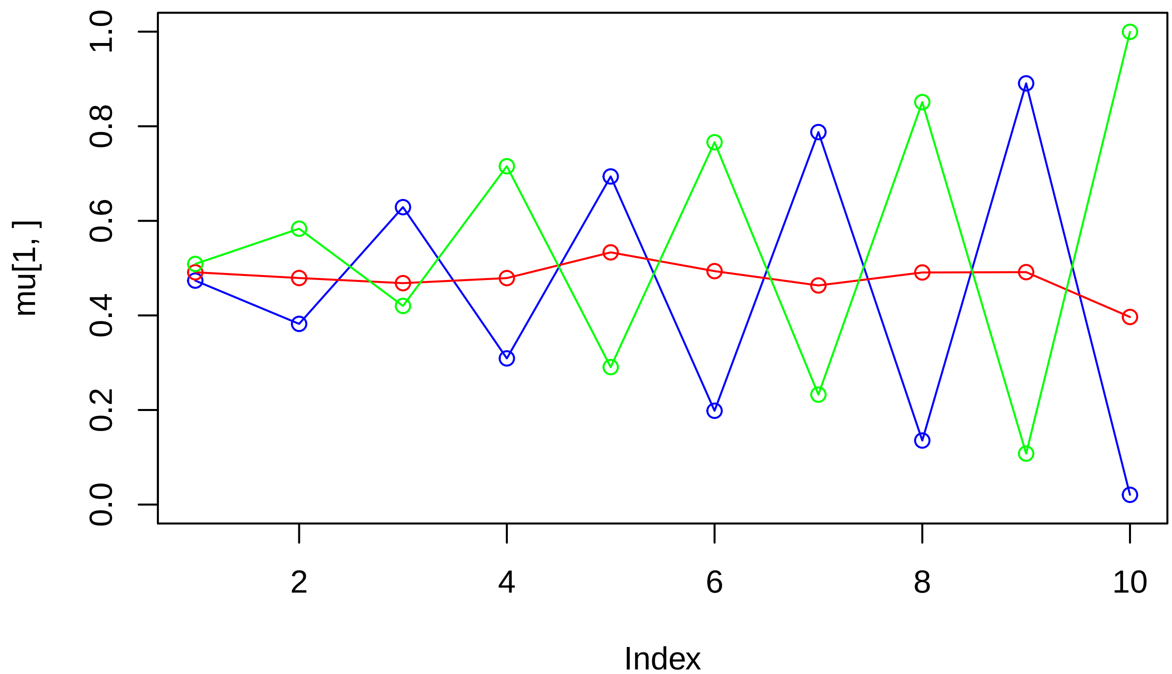

```
## iteration:  54 log likelihood:  -6744.212
```



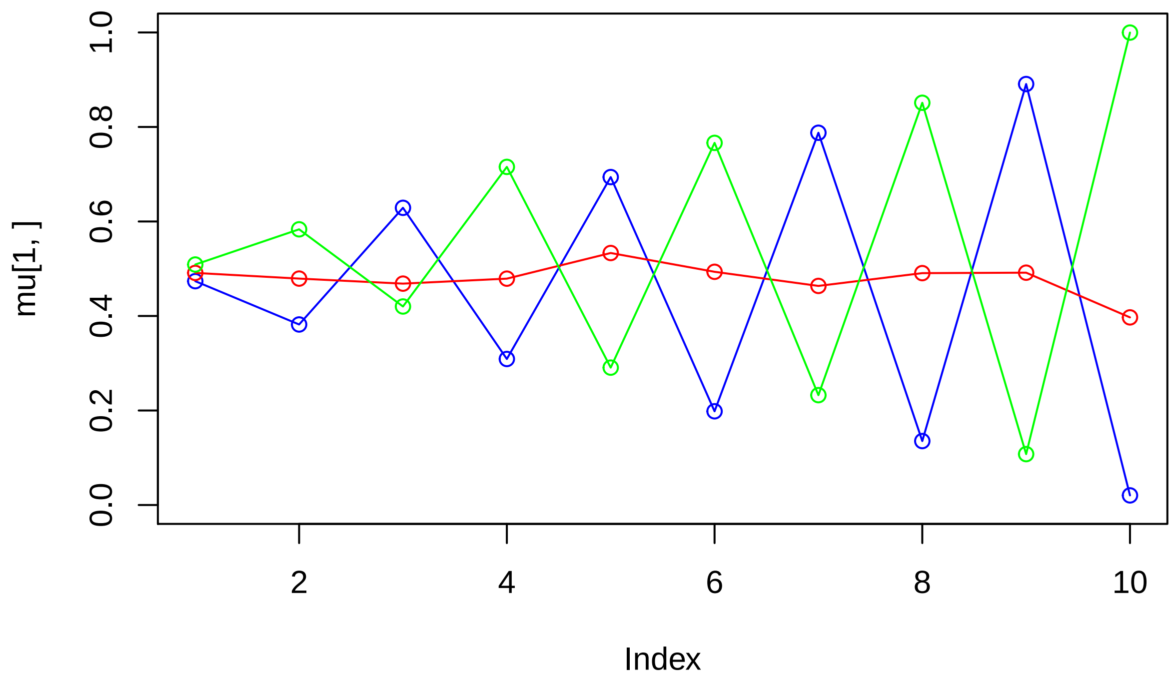

```
## iteration:  55 log likelihood:  -6744.086
```



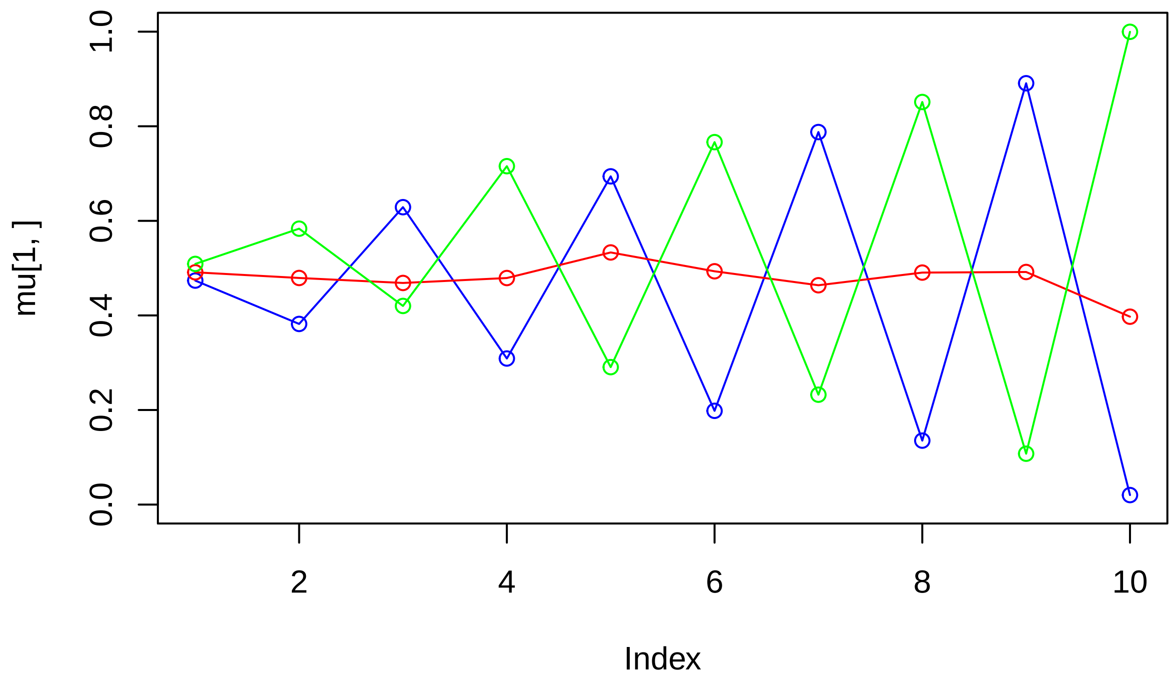

```
## iteration:  56 log likelihood:  -6743.964
```



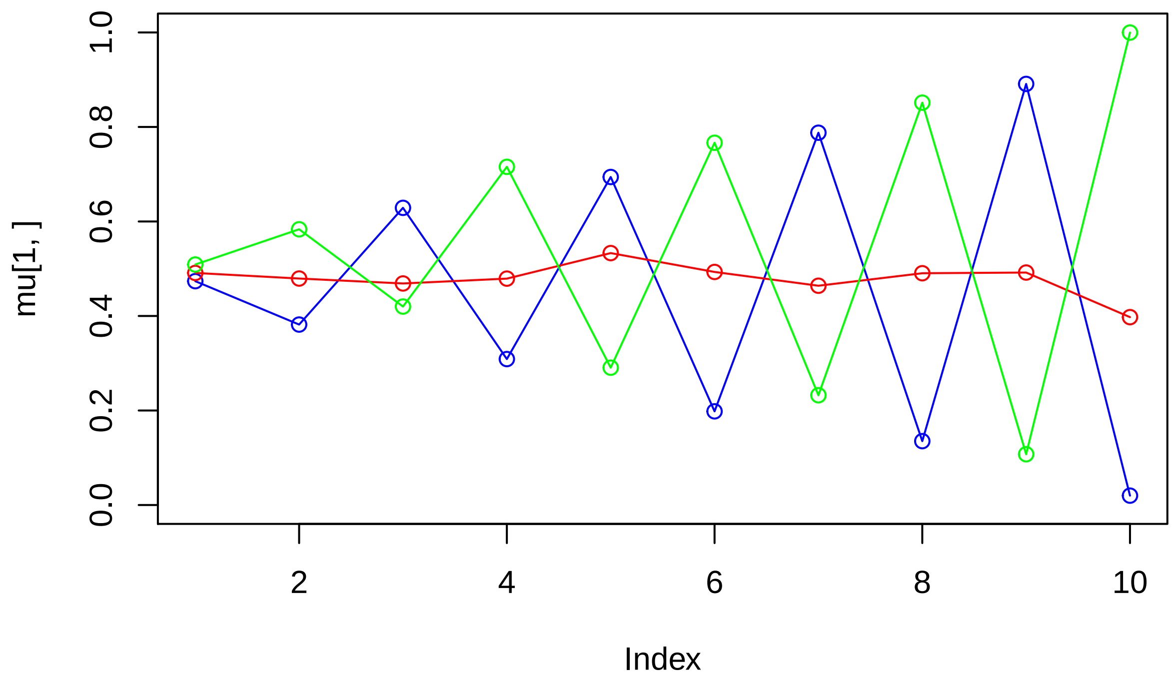

```
## iteration:  57 log likelihood:  -6743.848
```



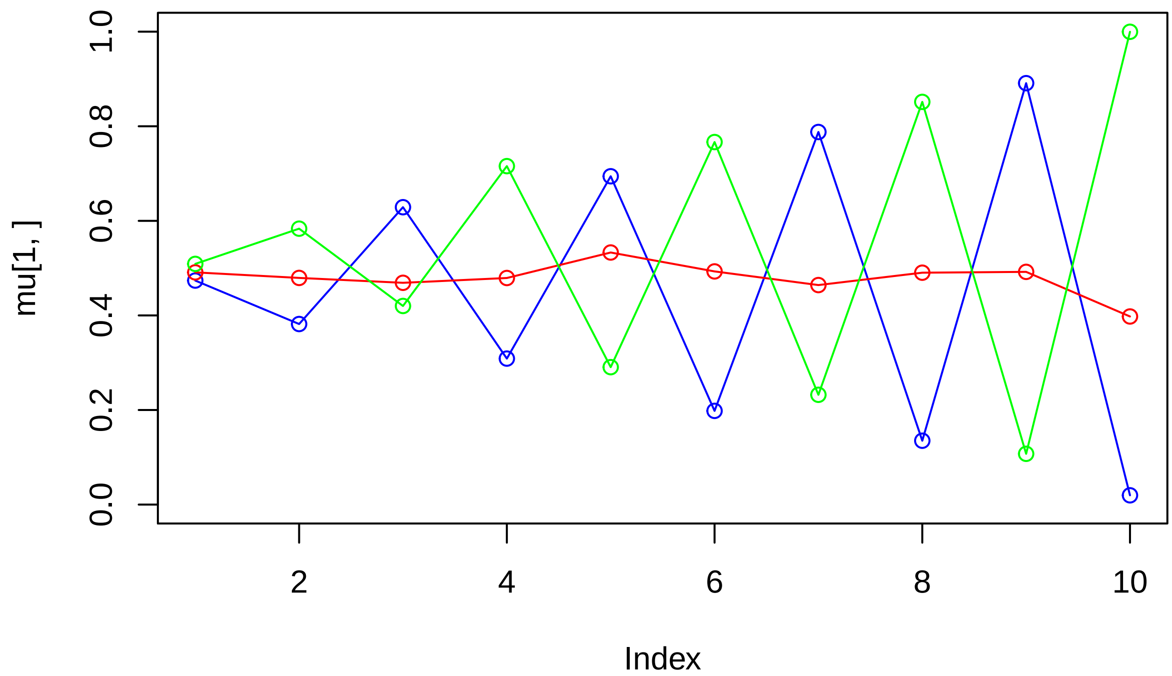

```
## iteration:  58 log likelihood:  -6743.736
```



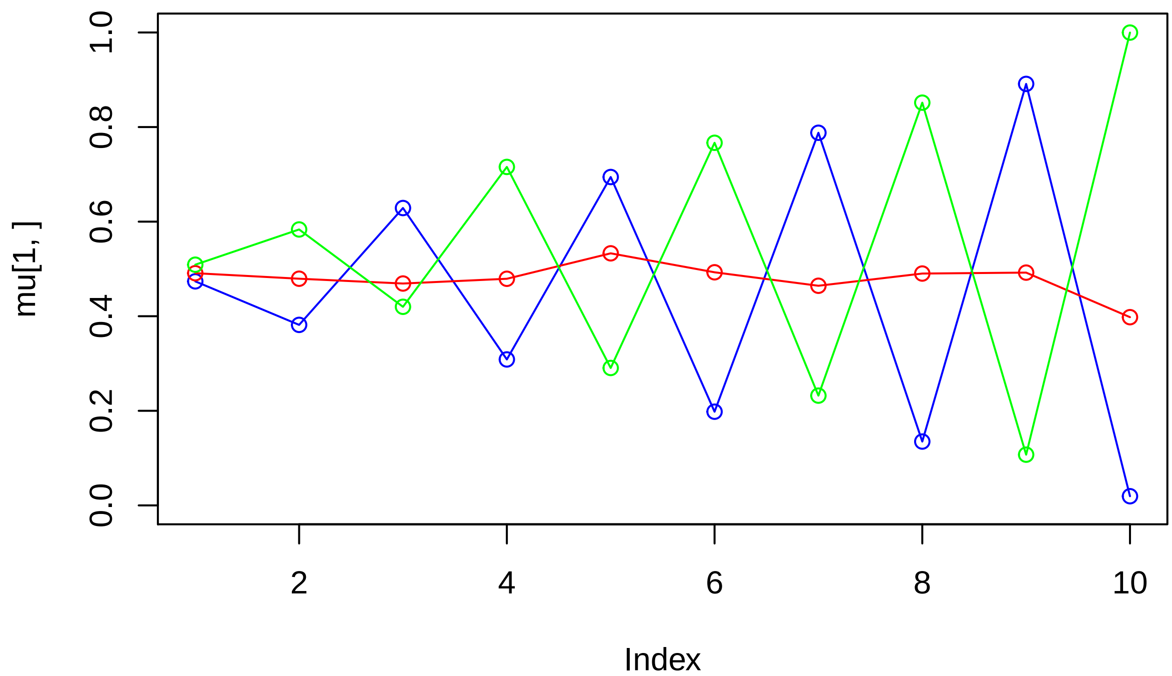

```
## iteration:  59 log likelihood:  -6743.628
```



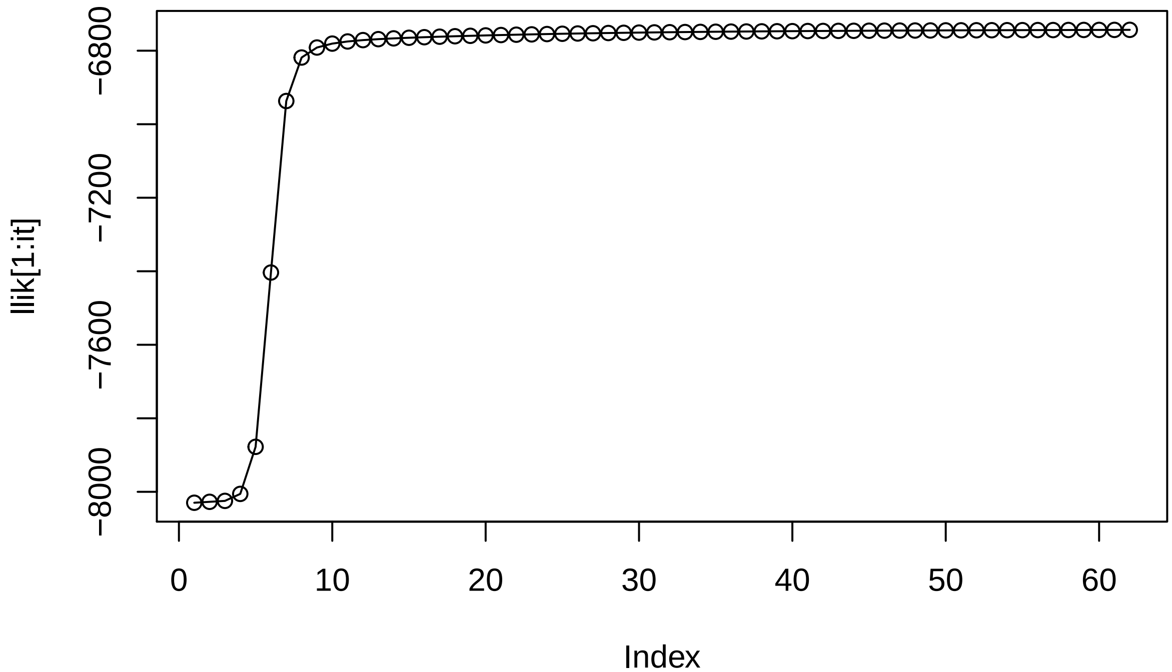

```
## iteration:  60 log likelihood:  -6743.524
```




```
## iteration:  61 log likelihood:  -6743.423
```




```
## iteration:  62 log likelihood:  -6743.326
```




```
#em3<-EM_algorithm(K=4)

# ggplot()+geom_point(aes(x=c(1:length(em1)),y=em1),color="blue")+
#   geom_point(aes(x=c(1:length(em2)),y=em2),color="red")+
#   geom_point(aes(x=c(1:length(em3)),y=em3),color="grey")+
#   xlab("Number of Iterations")+ylab("Expected Log Likelihood")
```