machine learning(732A99) lab1 Block 2

Anubhav Dikshit(anudi287)

4 December 2018

Contents

Assignment 1

Loading The Libraries

1.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.

Loading Input files

```
spam_data <- read.csv(file = "spambase.data", header = FALSE)
colnames(spam_data)[58] <- "Spam"
spam_data$Spam <- factor(spam_data$Spam, levels = c(0,1), labels = c("0", "1"))
```

Splitting into Train and Test with 66% and 33% ratio.

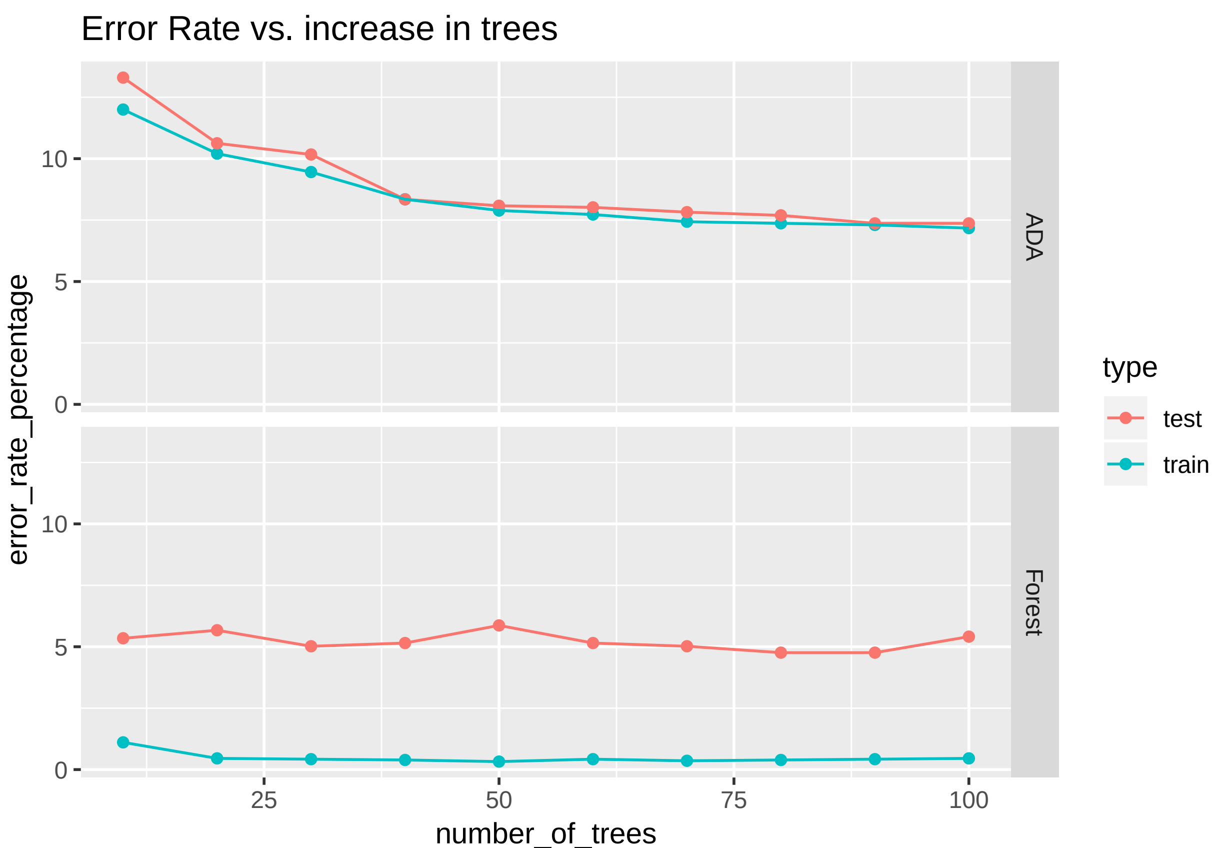
```
set.seed(12345)
n =  NROW(spam_data)
id = sample(1:n, floor(n*(2/3)))
train = spam_data[id,]
test = spam_data[-id,]
```

Trainning the Model

Adaboost with varying depth

```
final_result <- NULL
for(i in seq(from = 10, to = 100, by = 10)){

ada_model <- mboost::blackboost(Spam~.,
                                 data = train,
                                 family = AdaExp(),
                               control=boost_control(mstop=i))

forest_model <- randomForest(Spam~., data = train, ntree = i)


prediction_function <- function(model, data){
  predicted <- predict(model, newdata = data, type = c("class"))
  predict_correct <- ifelse(data$Spam == predicted, 1, 0)
  score <- sum(predict_correct)/NROW(data)
  return(score)
}


train_ada_model_predict <- predict(ada_model, newdata = train, type = c("class"))
test_ada_model_predict <- predict(ada_model, newdata = test, type = c("class"))
train_forest_model_predict <- predict(forest_model, newdata = train, type = c("class"))
test_forest_model_predict <- predict(forest_model, newdata = test, type = c("class"))
```

```
test_predict_correct <- ifelse(test$Spam == test_forest_model_predict, 1, 0)
train_predict_correct <- ifelse(train$Spam == train_forest_model_predict, 1, 0)


train_ada_score <-  prediction_function(ada_model, train)
test_ada_score <-  prediction_function(ada_model, test)
train_forest_score <-  prediction_function(forest_model, train)
test_forest_score <-  prediction_function(forest_model, test)

iteration_result <- data.frame(number_of_trees = i,
                               accuracy = c(train_ada_score,
                                            test_ada_score,
                                            train_forest_score,
                                            test_forest_score),
                               type  = c("train", "test", "train", "test"),
                               model = c("ADA", "ADA",  "Forest", "Forest"))


final_result <- rbind(iteration_result, final_result)
}

final_result$error_rate_percentage <- 100*(1 - final_result$accuracy)
ggplot(data = final_result, aes(x = number_of_trees,
                                y = error_rate_percentage,
                                group = type, color = type)) +
  geom_point() +
  geom_line() +
  ggtitle("Error Rate vs. increase in trees") + facet_grid(rows = vars(model))
```



Analysis:

From the plots we can clearly see that ADA boosted methods uses more trees(~50) to reduce the test error, while randomforest achieves saturation in short number of trees(~10). We also see that random forest achieves less error than ADA tree for both tree and test cases.

2 Your task is to implement the EM algorithm for mixtures of multivariate Bernoulli distributions. Please use the template in the next page to solve the assignment. Then, use your implementation 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.

Function for EM Algorithm

```
myem <- 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
```

```r
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = K) # true mixing coefficients
true_mu <- matrix(nrow=K, ncol=D) # true conditional distributions
true_pi=c(rep(1/3, K))

if(K == 2){
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")

  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)
}else if(K == 3){
    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")

  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)
}else {
    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")
    points(true_mu[4,], type="o", col="yellow")

    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)
    true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)}

# Producing the training data
for(n in 1:N) {
k <- sample(1:K,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}

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) {
```

```r
if(K == 2){
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
}else if(K == 3){
  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")
}else{
    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(k in 1:K)
prod <- exp(x %*% log(t(mu))) * exp((1-x) %*% t(1-mu))

num = matrix(rep(pi,N), ncol = K, byrow = TRUE) * prod
dem = rowSums(num)
poster = num/dem

#Log likelihood computation.
llik[it] = sum(log(dem))
# Your code here
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
# Your code here
num_pi = colSums(poster)
pi = num_pi/N
mu = (t(poster) %*% x)/num_pi
}

a <- pi
b <- mu
c <- plot(llik[1:it], type="o")
result <- list(c(a,b,c))
return(result)
}
```

Analysis:

EM is an iterative expectation maximumation technique. The way this works is for a given mixed distribution we guess the components of the data. This is done by first guessing the number of components and then randomly initializing the parameters of the said distribution (Mean, Varience).

Sometimes the data do not follow any known probability distribution but a mixture of known distributions such as:

$$p(x) = \sum_{k=1}^{K} p(k).p(x|k)$$

where p(x|k) are called mixture components and p(k) are called mixing coefficients: where p(k) is denoted by

$$\pi_k$$

With the following conditions

$$0 \leq \pi_k \leq 1$$

and

$$\sum_k \pi_k = 1$$

$$\pi_k^{ML} = \frac{\sum_N p(z_{nk}|x_n, \mu, \pi)}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p(z_{nk}|x_n, \mu, \pi)}{\sum_n p(z_{nk}|x_n, \mu, \pi)}$$

Where

$$p(z_{nk}|x_n, \mu, \pi) = Z = \frac{\pi_k p(x_n|\mu_k)}{\sum_k p(x_n|\mu_k)}$$

EM function with loops

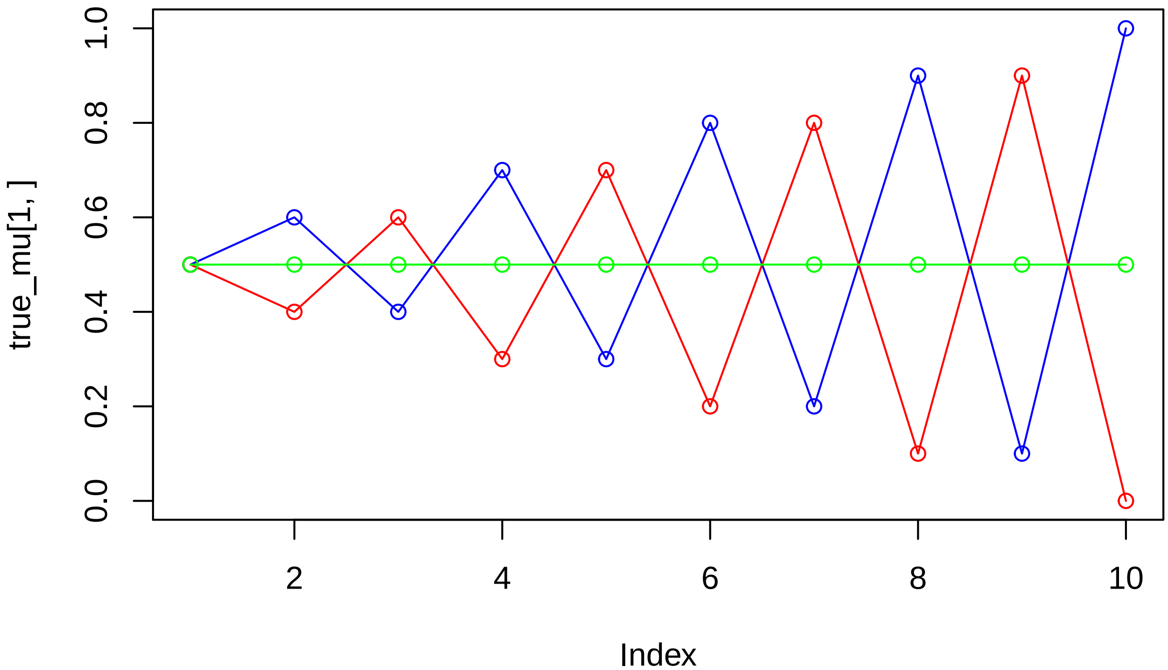
```
myem_loop <- function(K){
# 2 - Mixture Models ####
set.seed(1234567890)

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 = K) # true mixing coefficients
true_mu <- matrix(nrow=K, ncol=D) # true conditional distributions
true_pi=c(rep(1/3, K))


if (K == 2){
  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)
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")
}else if (K == 3){
  true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
```

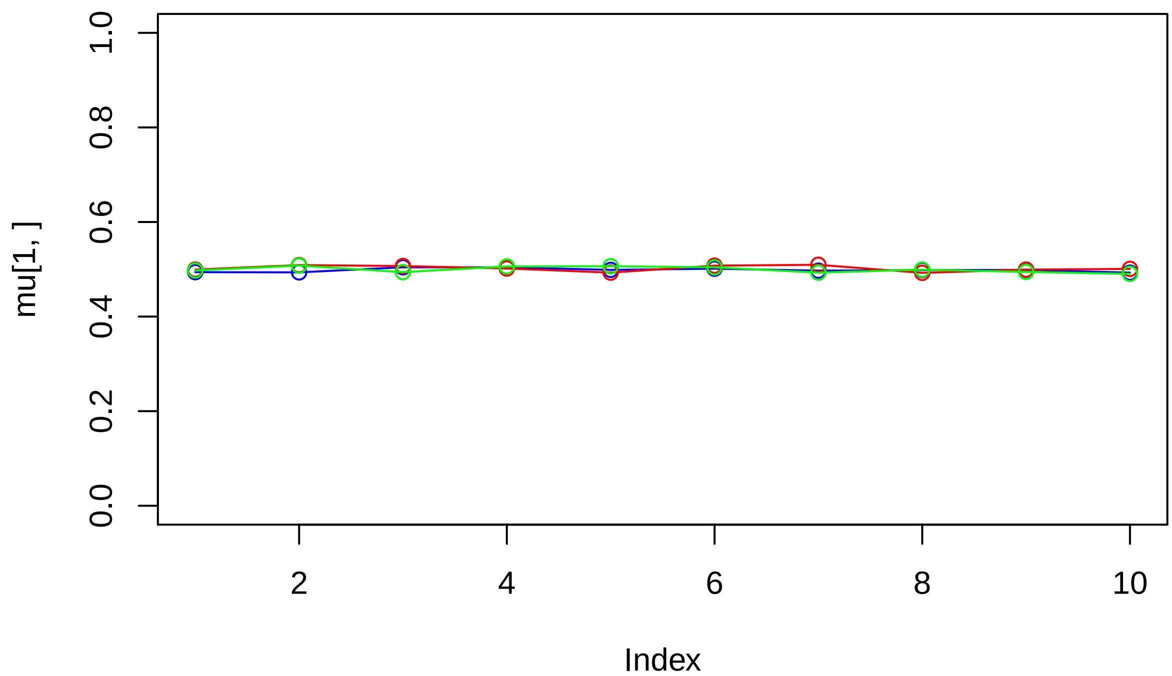
```r
  true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
  true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")
  points(true_mu[3,], type="o", col="green")
}else{
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
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)
true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)
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")
points(true_mu[4,], type="o", col="yellow")
}


# Producing the training data
for(n in 1:N) {
  k <- sample(1:K,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

 # number of guessed components
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)
}
pi
mu
for(it in 1:max_it) {
  if (K == 2){
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
  }else if (K == 3){
    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")
  }else{
    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)
```

```r
# E-step: Computation of the fractional component assignments
m <- matrix(NA, nrow = 1000, ncol = k)

#Here I create the Bernouilli probabilities, lecture 1b, slide 7. I use 3 loops to do it for the thre
# not very efficient, but it works.
for (j in 1:k){
  for(each in 1:nrow(x)){
    row <- x[each,]
    vec <- c()
    for (i in 1:10) {
      a <- mu[j,i]^row[i]
      b <- a * ((1-mu[j,i])^(1-row[i]))
      vec[i] <- b
      c <- prod(vec)
    }
    m[each, j] <- c
  }
}

# Here I create a empty matrix, to store all values for the numerator of the formula on the bottom of
# slide 9, lecture 1b.
m2 <- matrix(NA, ncol = k, nrow = 1000)

# m2 stores all the values for the numerator of the formula on the bottom of slide 9, lecture 1b.
for (i in 1:1000){
  a <- pi * m[i,]
  m2[i,] <- a
}

# Sum m2 to get the denominator of the formula on the bottom of slide 9, lecture 1b.
m2_sum <- rowSums(m2)
m_final <- m2 / m2_sum

#Log likelihood computation.
ll <- matrix(nrow = 1000, ncol = K)
for (j in 1:K){
  for (i in 1:1000){
    ll[i, j] <- sum(((x[i,] * log(mu[j,])) + (1 - x[i,])*log(1-mu[j,])))
  }
}

ll <- ll + pi
llnew <- m_final * ll
llik[it] <- sum(rowSums(llnew))

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
```

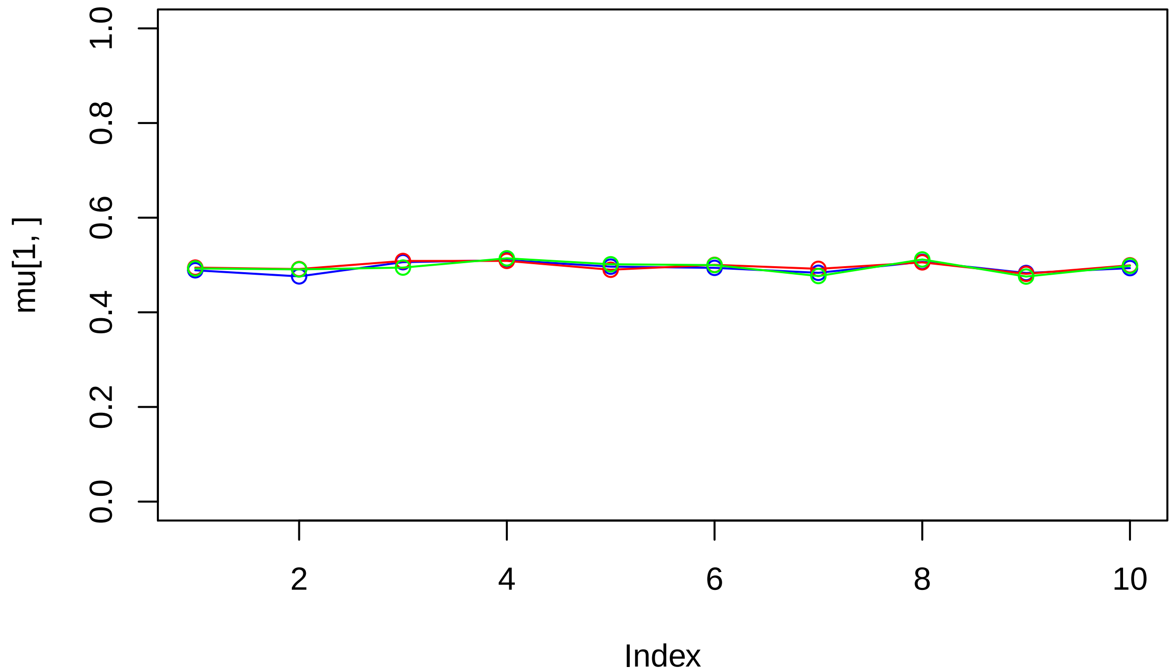
```r
  # Create the numerator for pi, slide 9, lecture 1b.
  numerator_pi <- colSums(m_final)

  # Create new values for pi, stored in the vector pi_new
  pi_new <- numerator_pi / N
  pi_new
  mnew <- matrix(NA, nrow = 1000, ncol = 10)
  mu_new <- matrix(NA, nrow = K, ncol = 10)

  for (j in 1:k){
    for (i in 1:1000){
      row <- x[i,] * m_final[i,j]
      mnew[i,] <- row
    }
    mnewsum <- colSums(mnew)/numerator_pi[j]
    mu_new[j,] <- mnewsum
  }

  # Now, to create the iterations, I have to run the code again and again, and specifying mu as new the
  # created for mu. Same goes for the other variables.
  mu <- mu_new
  pi <- pi_new
}
z <- m_final
output1 <- pi
output2 <- mu
output3 <- plot(llik[1:it], type="o")

result <- list(c(output1, output2, output3))
return(result)
}
```

```r
myem_loop(K=3)
```

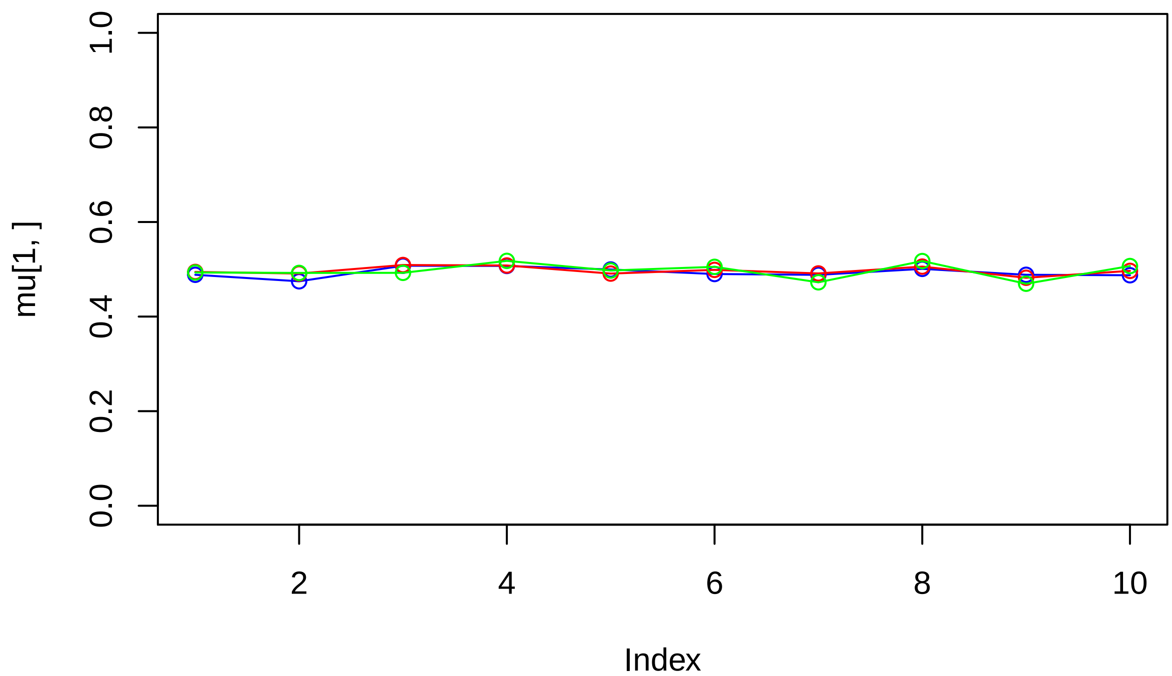

true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



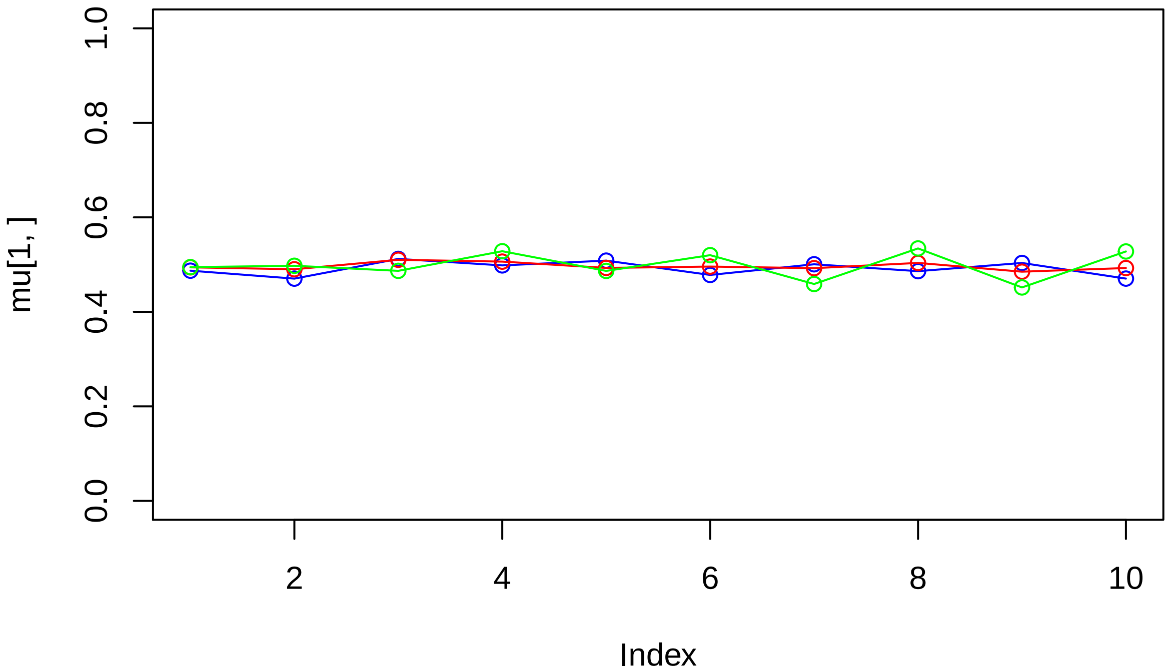

```
## iteration:  1 log likelihood:  -6597.778
```



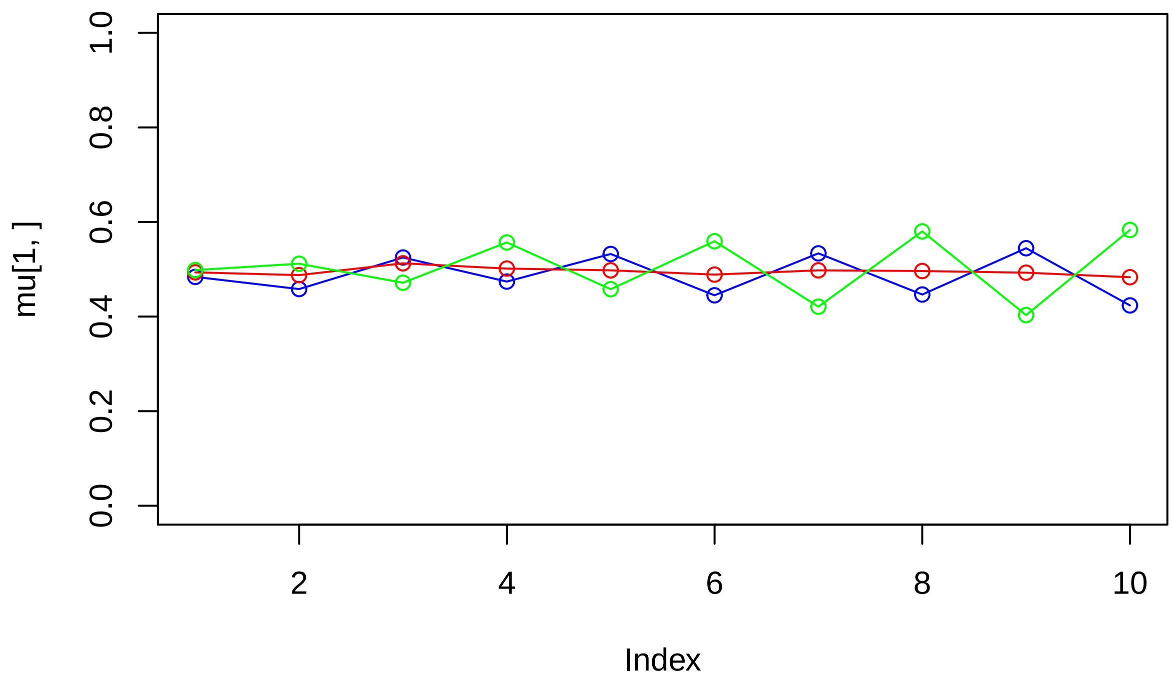

```
## iteration:  2 log likelihood:  -6595.239
```



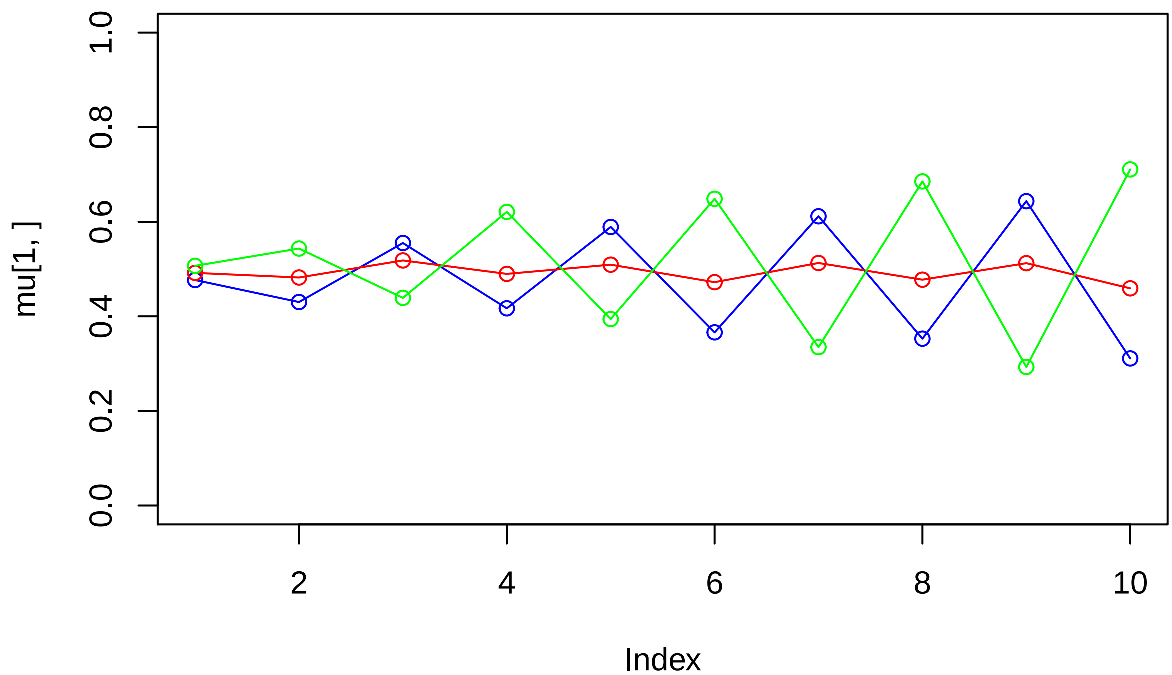

```
## iteration:  3 log likelihood:  -6592.753
```



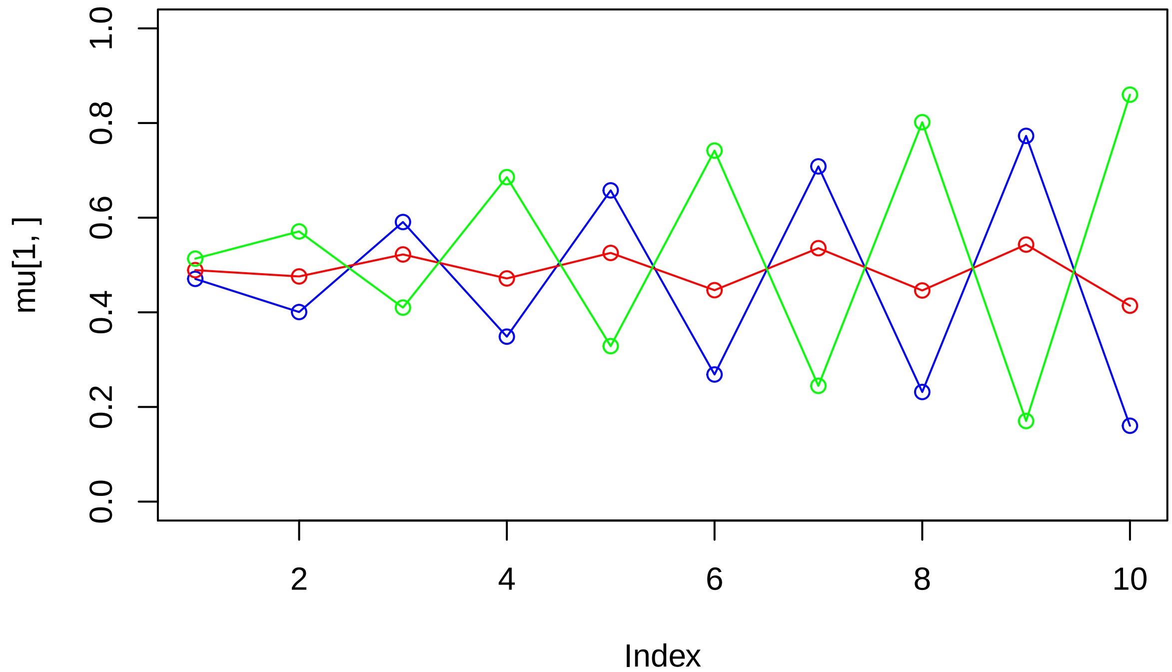

```
## iteration:  4 log likelihood:  -6573.7
```



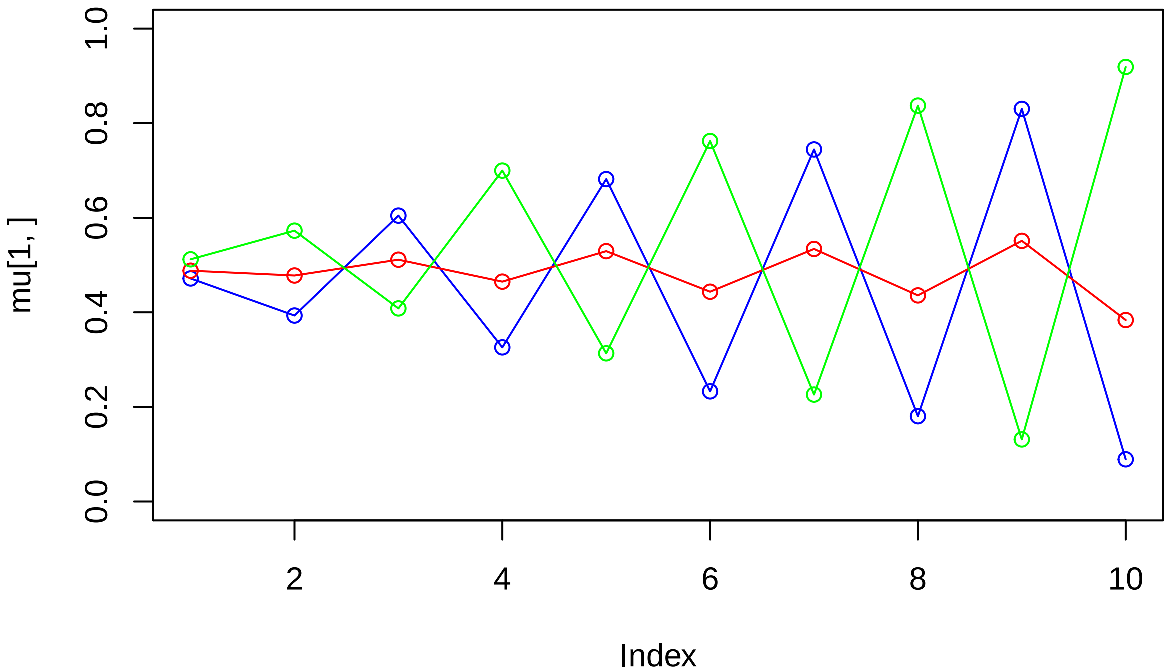

```
## iteration:  5 log likelihood:  -6446.022
```



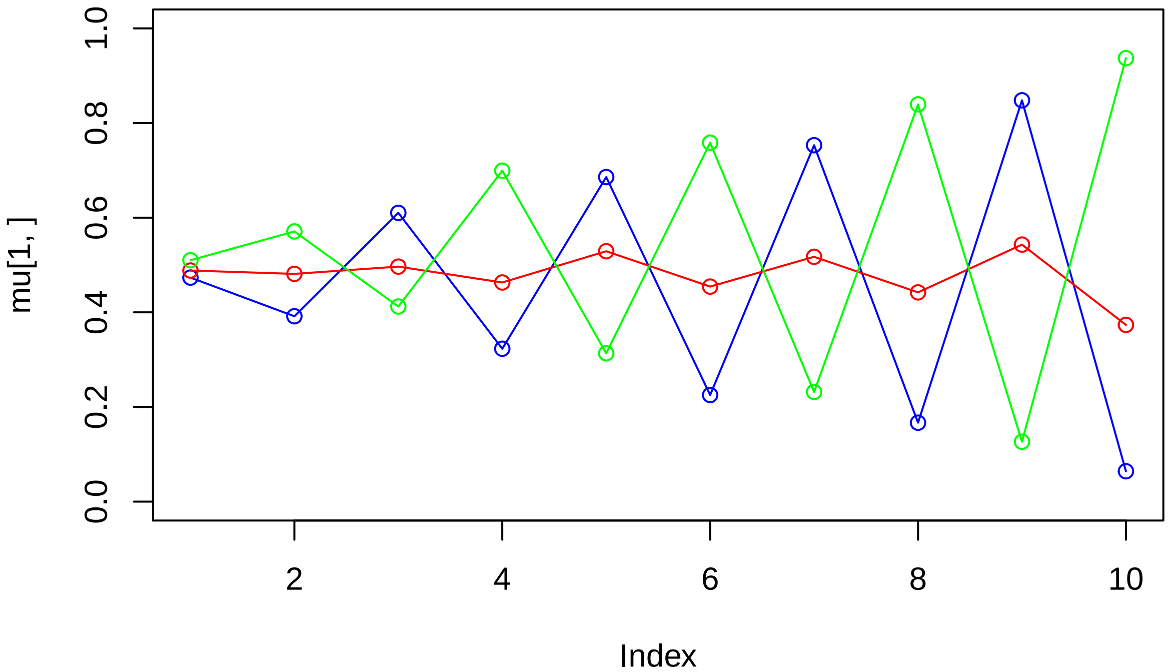

```
## iteration:  6 log likelihood:  -5978.865
```



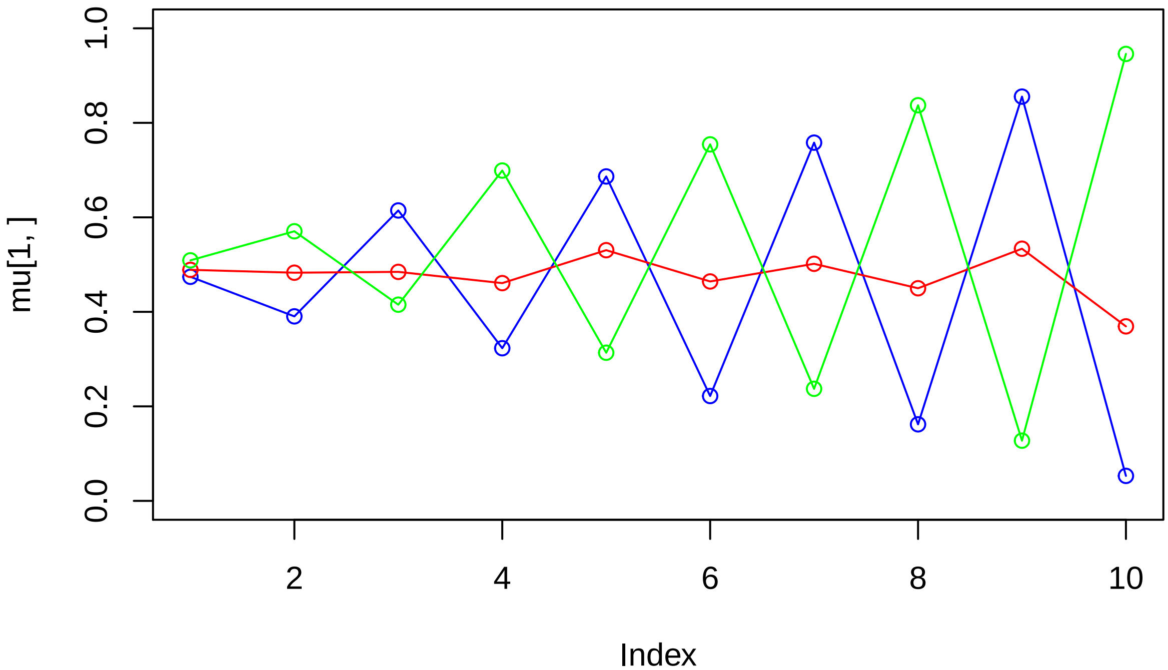

```
## iteration:  7 log likelihood:  -5537.074
```



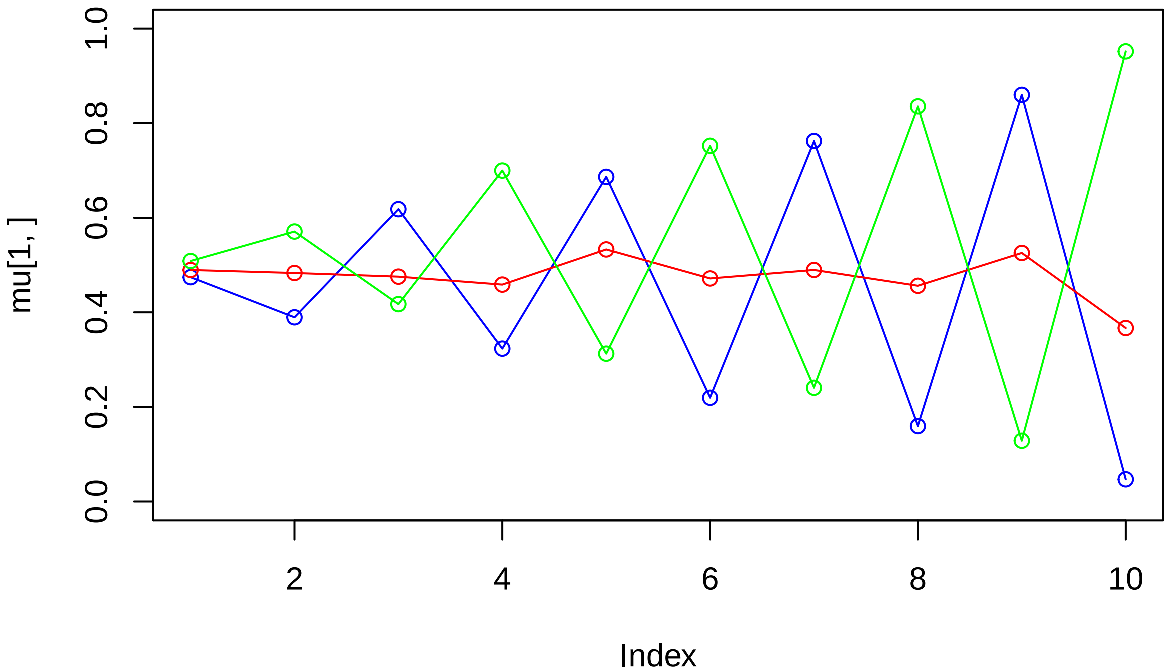

```
## iteration:  8 log likelihood:  -5429.225
```



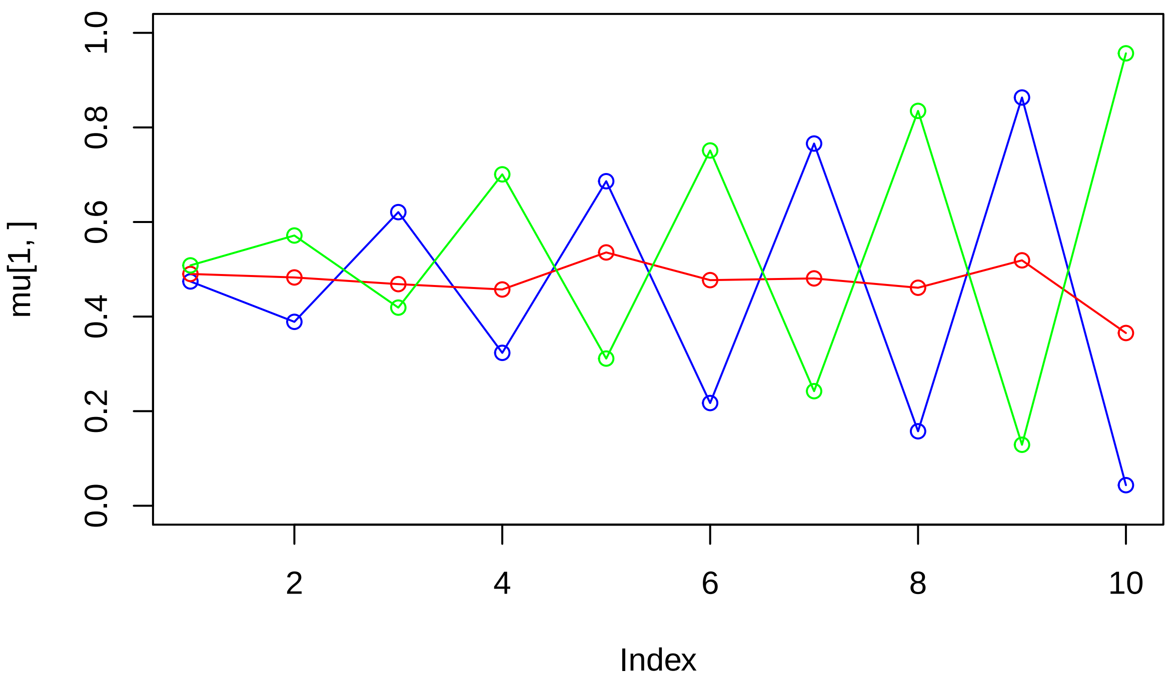

```
## iteration:  9 log likelihood:  -5401.95
```



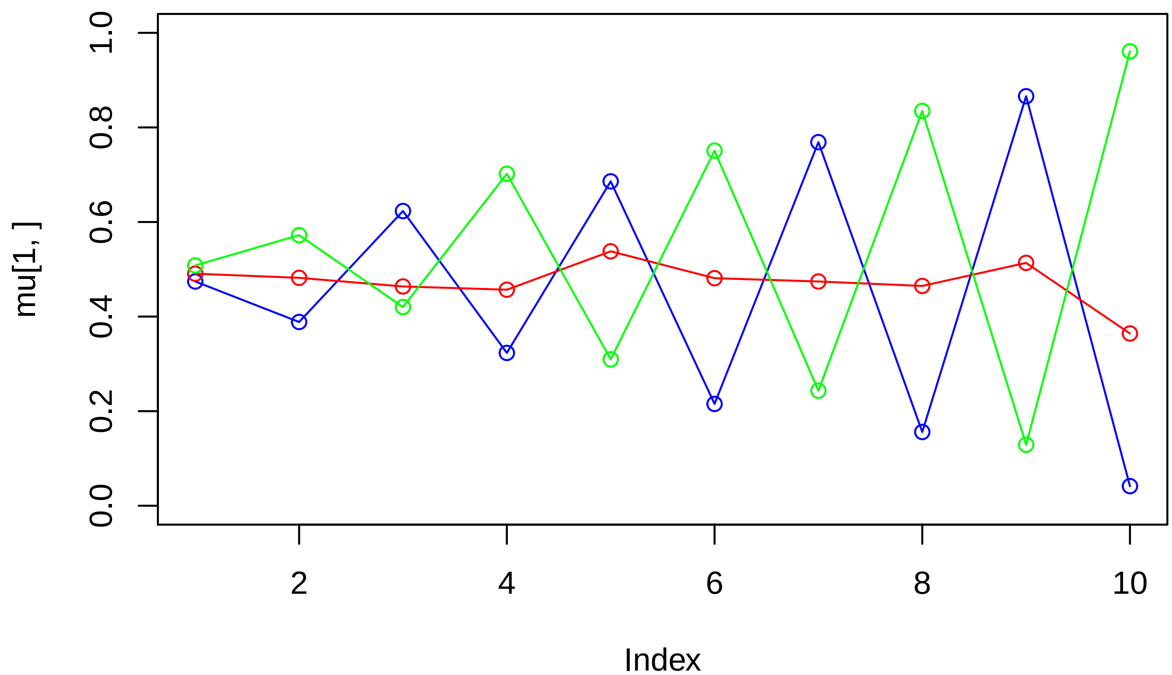

```
## iteration:  10 log likelihood:  -5389.023
```



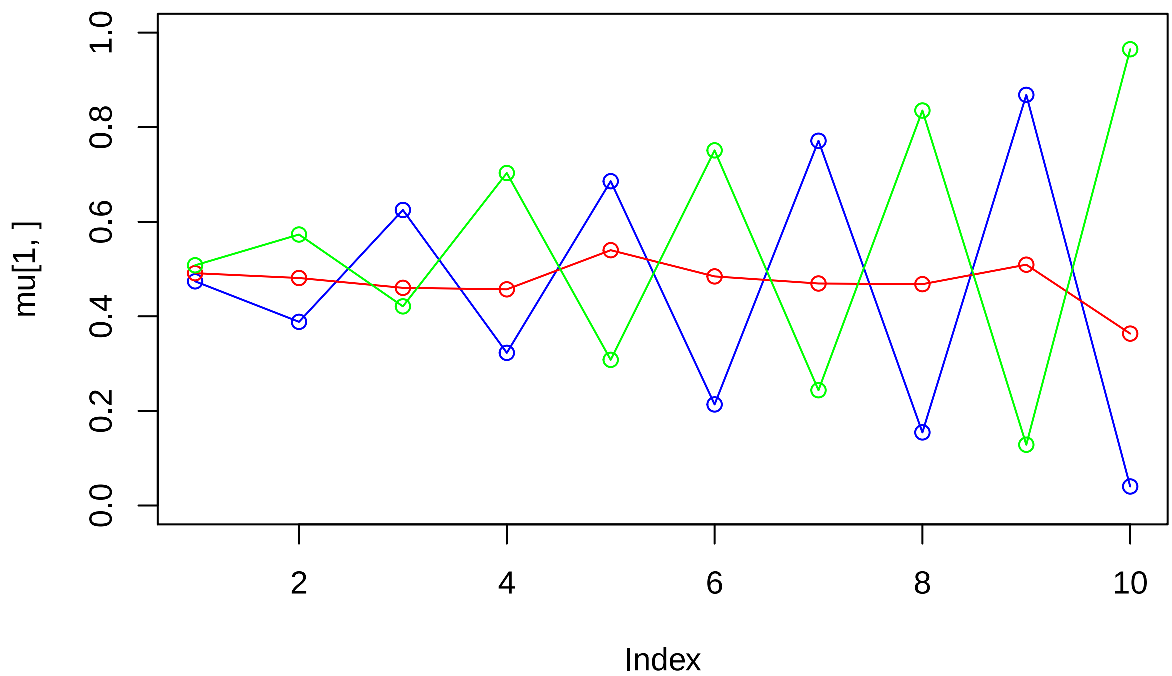

```
## iteration:  11 log likelihood:  -5380.443
```



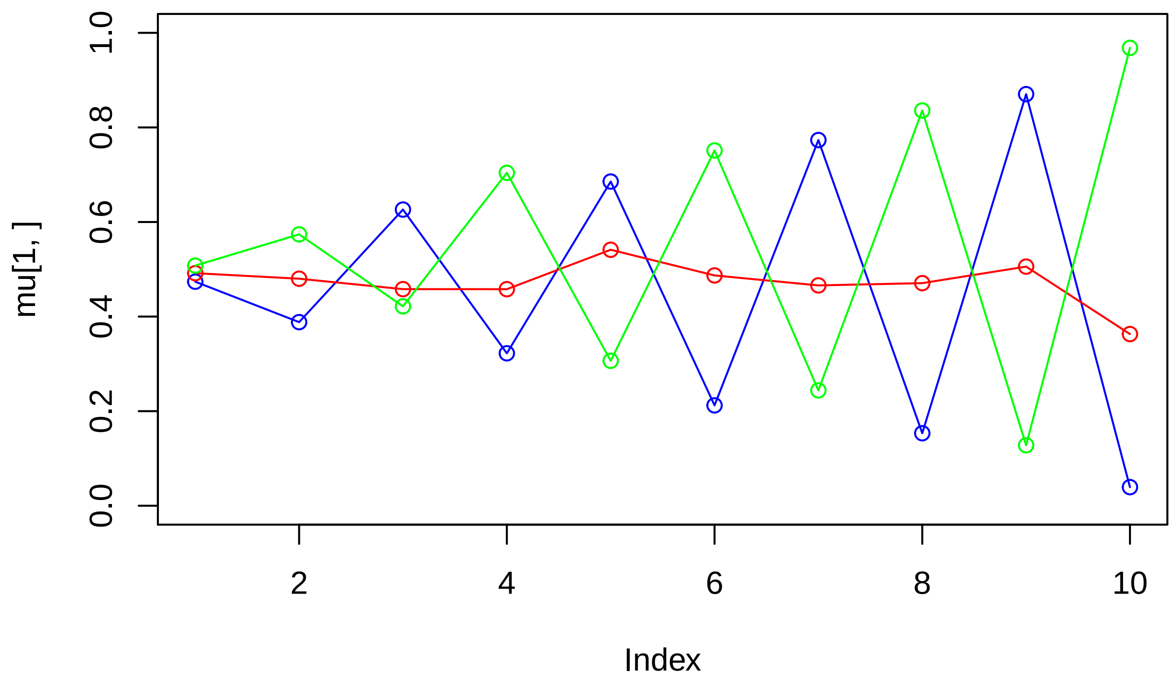

```
## iteration:  12 log likelihood:  -5373.845
```



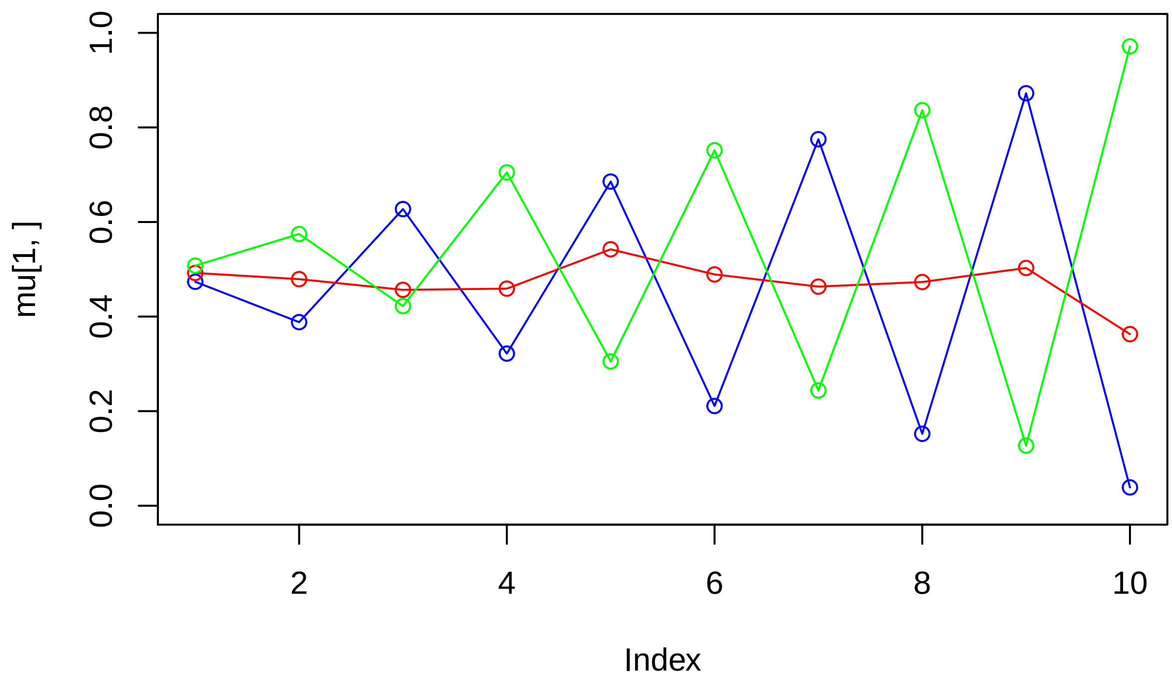

```
## iteration:  13 log likelihood:  -5368.41
```



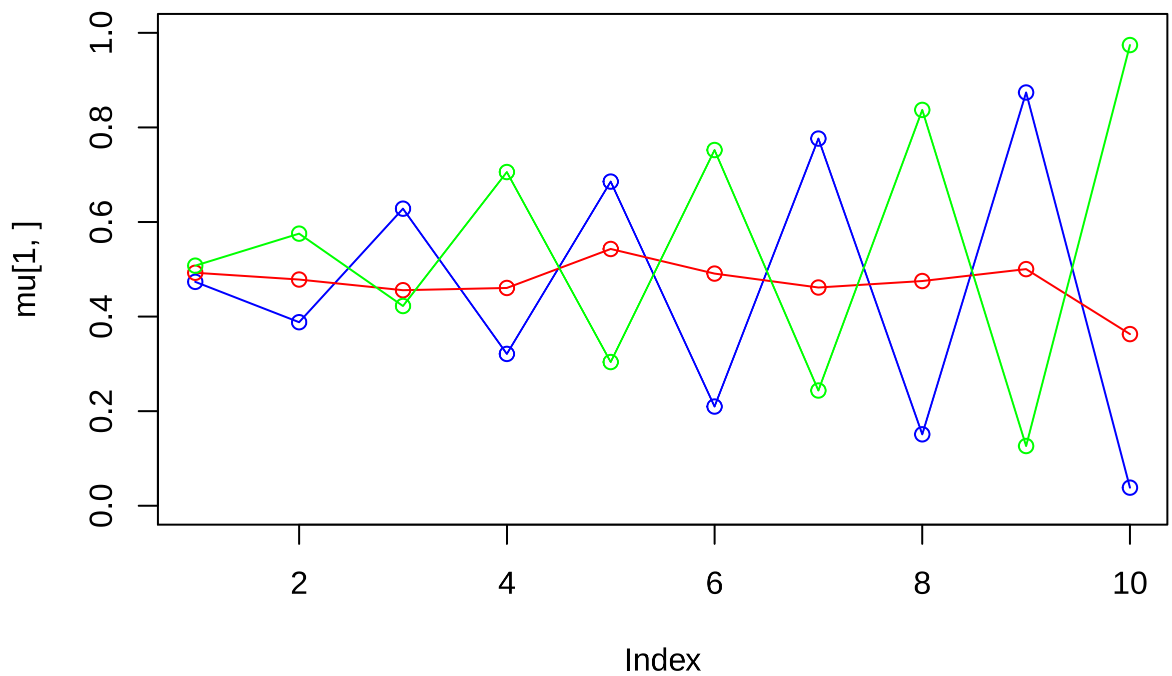

```
## iteration:  14 log likelihood:  -5363.759
```



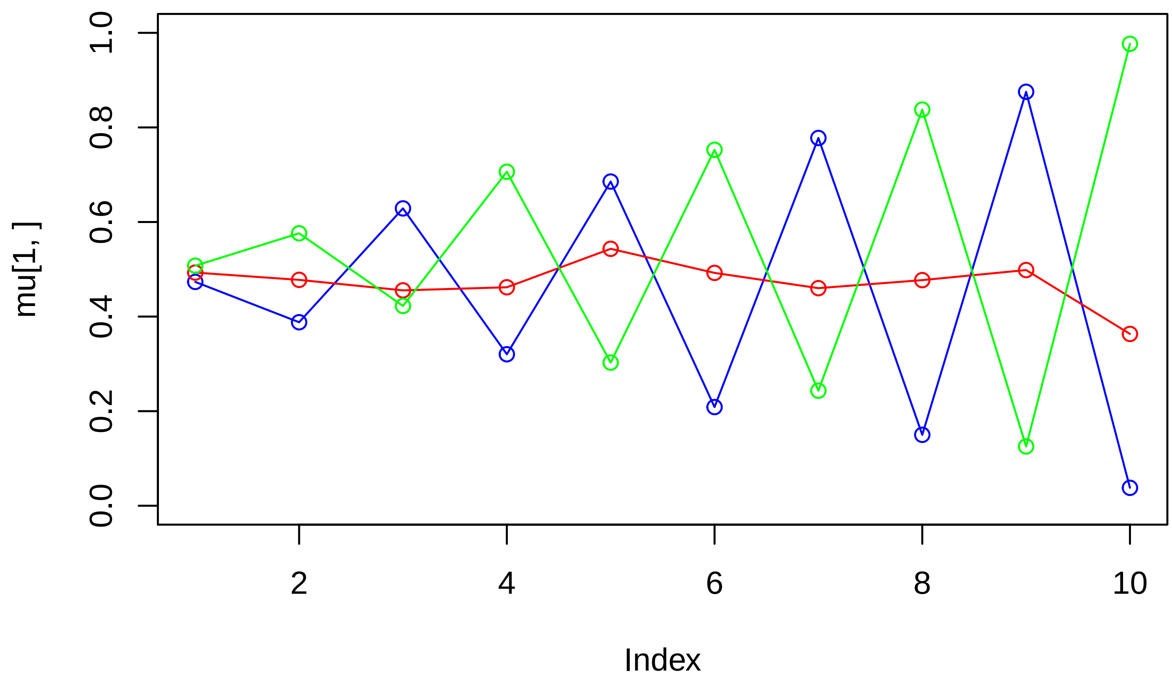

```
## iteration:  15 log likelihood:  -5359.682
```



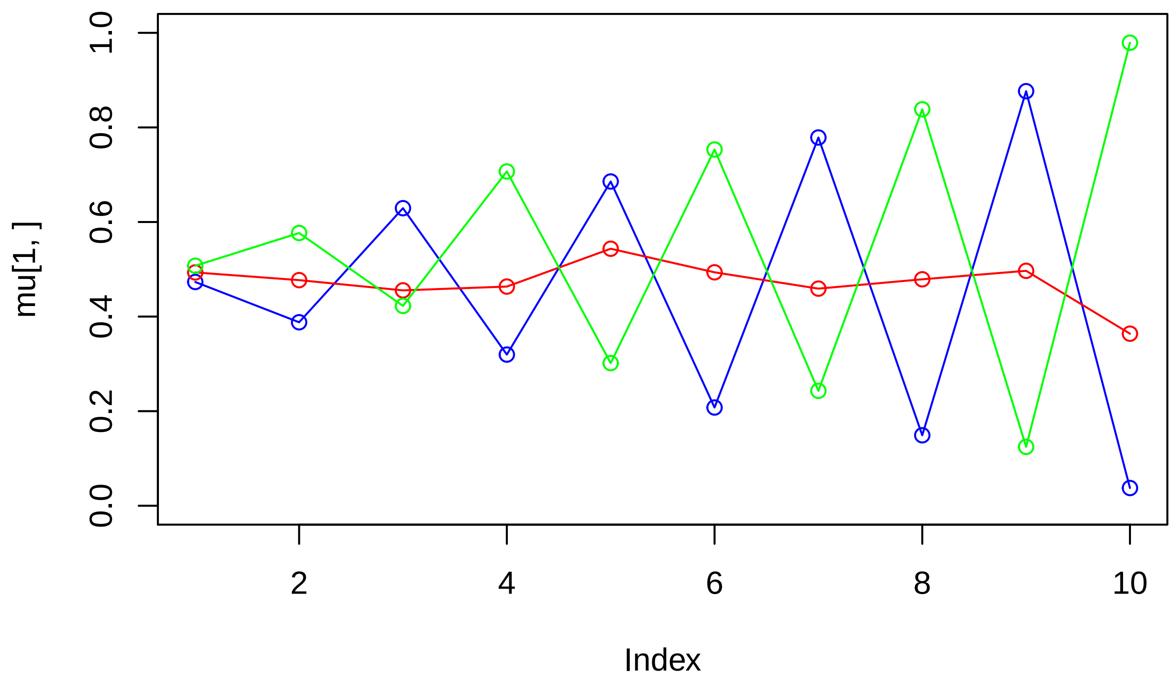

```
## iteration:  16 log likelihood:  -5356.051
```



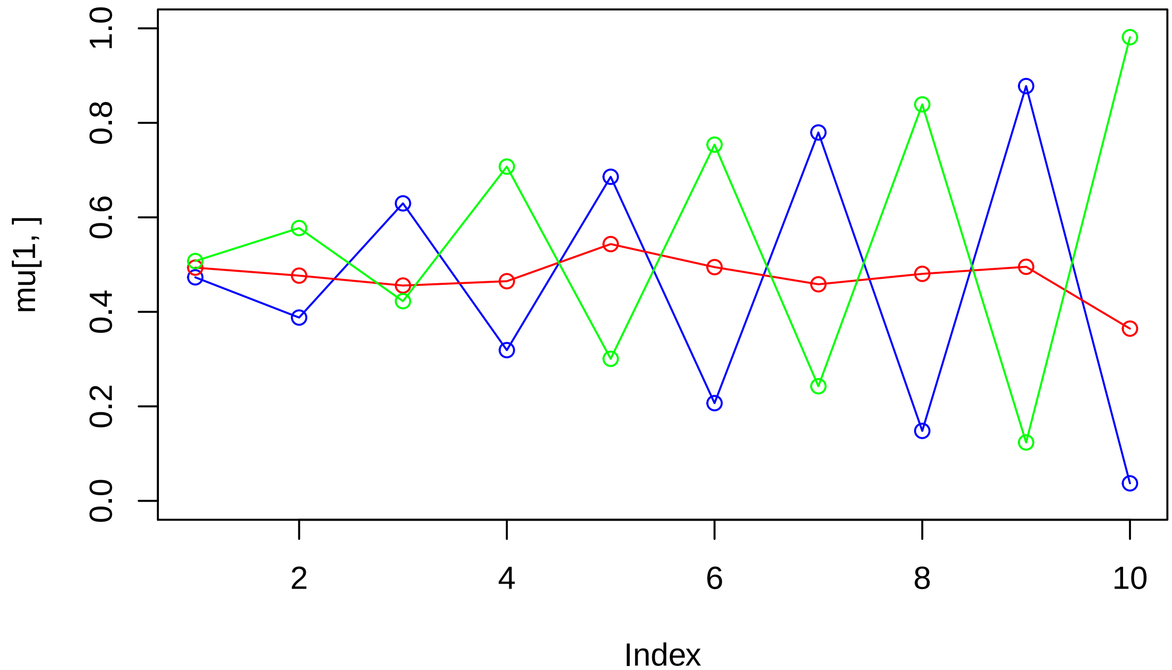

```
## iteration:  17 log likelihood:  -5352.782
```



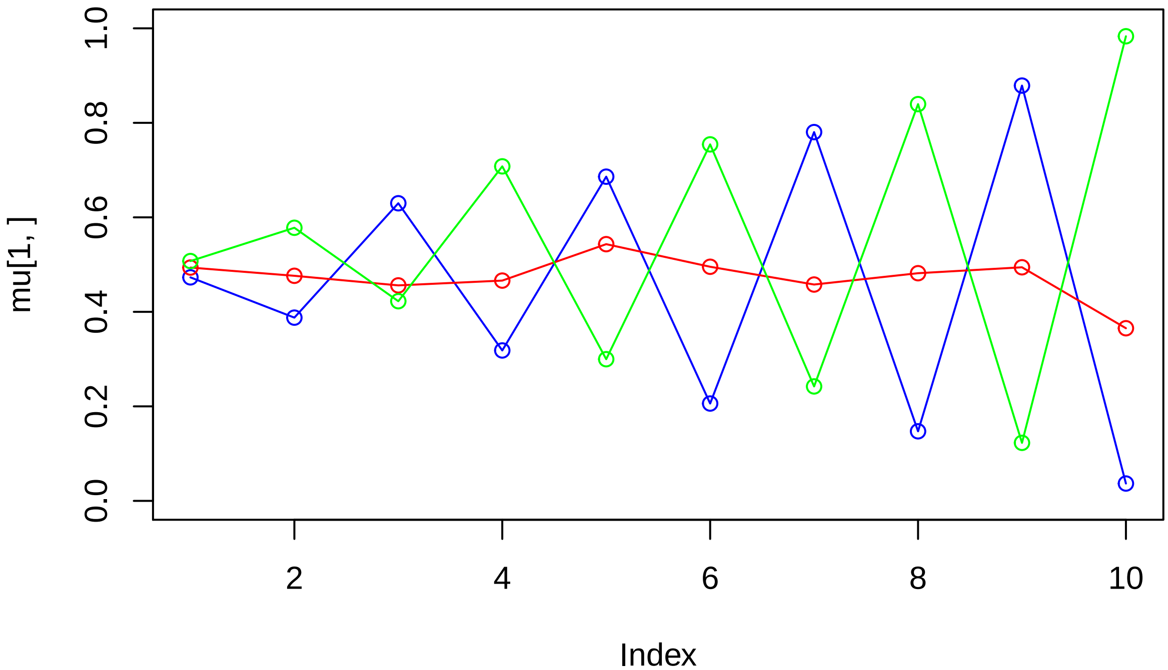

```
## iteration:  18 log likelihood:  -5349.816
```



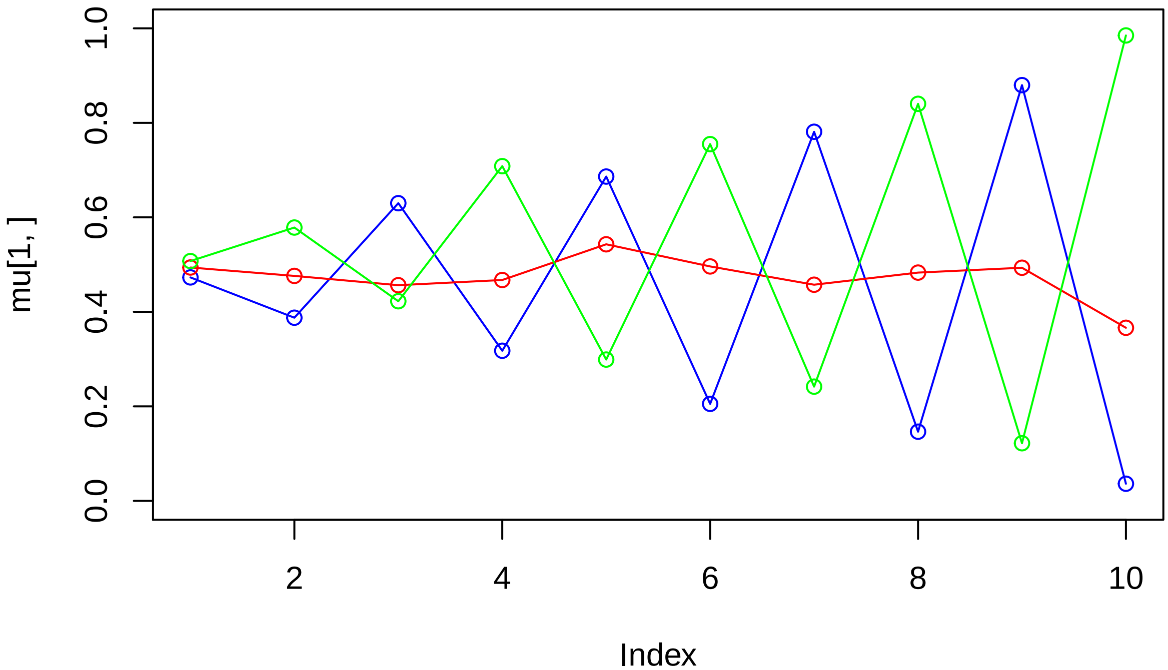

```
## iteration:  19 log likelihood:  -5347.113
```



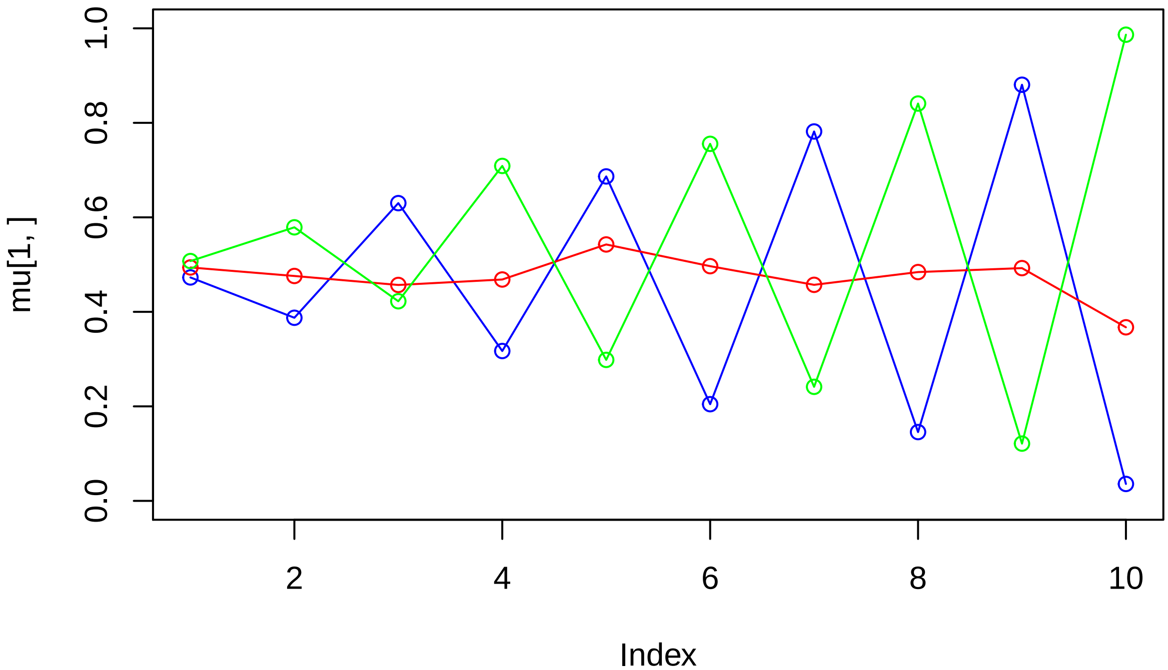

```
## iteration:  20 log likelihood:  -5344.641
```



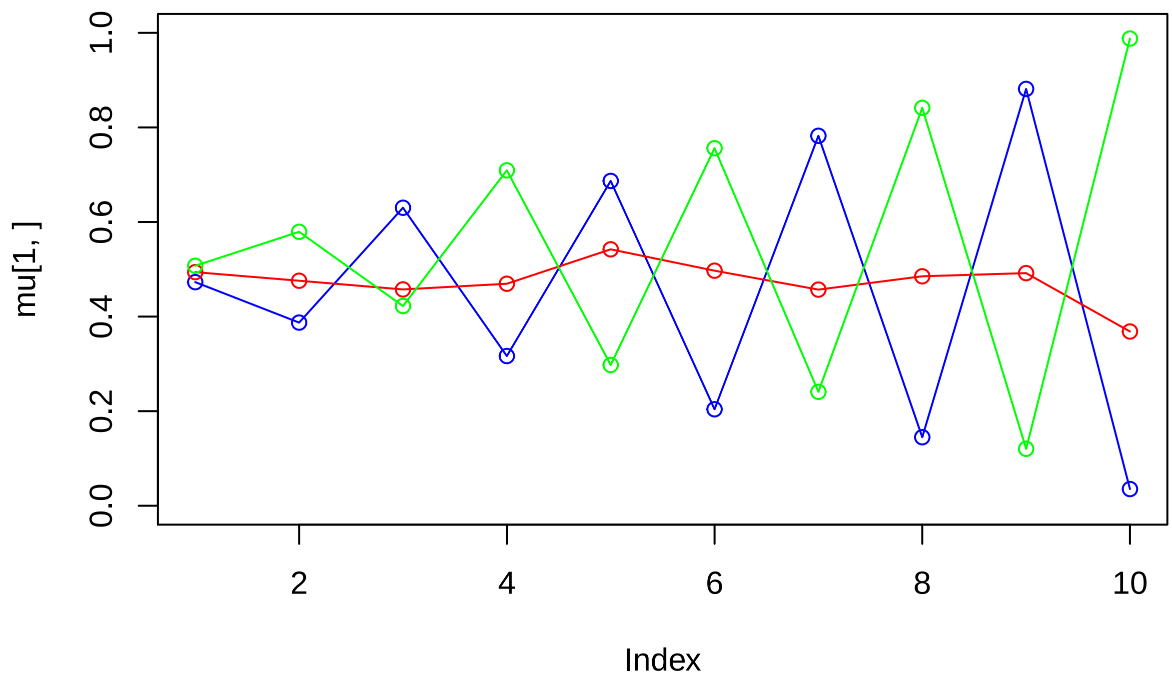

```
## iteration:  21 log likelihood:  -5342.375
```



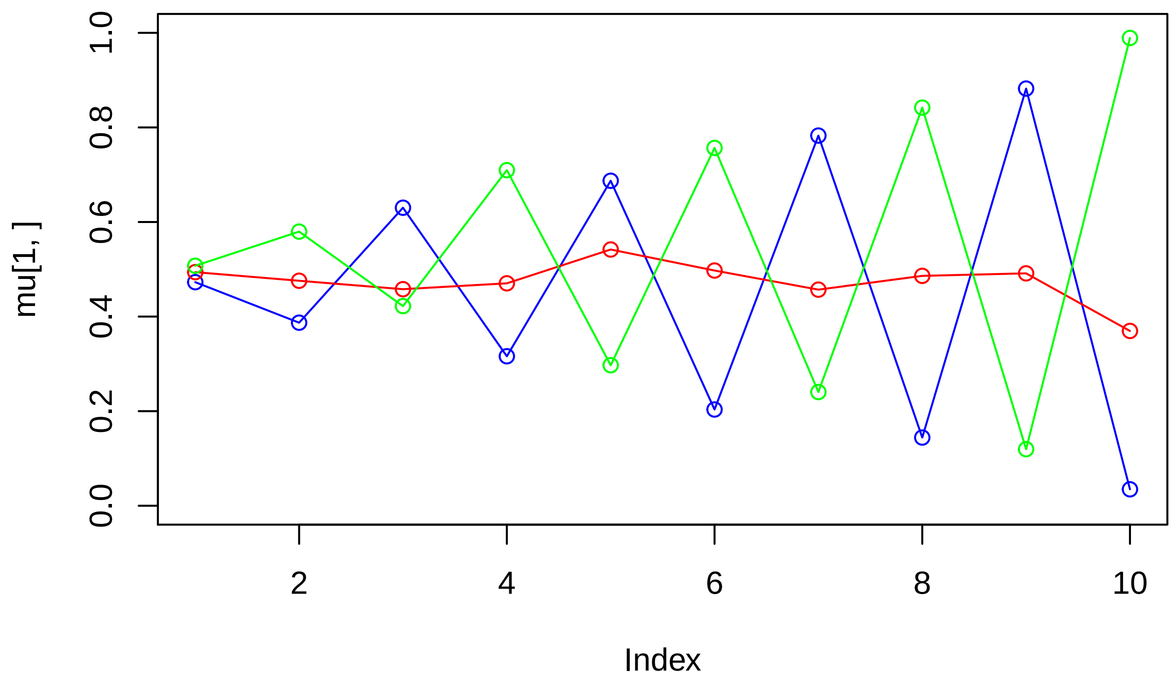

```
## iteration:  22 log likelihood:  -5340.295
```



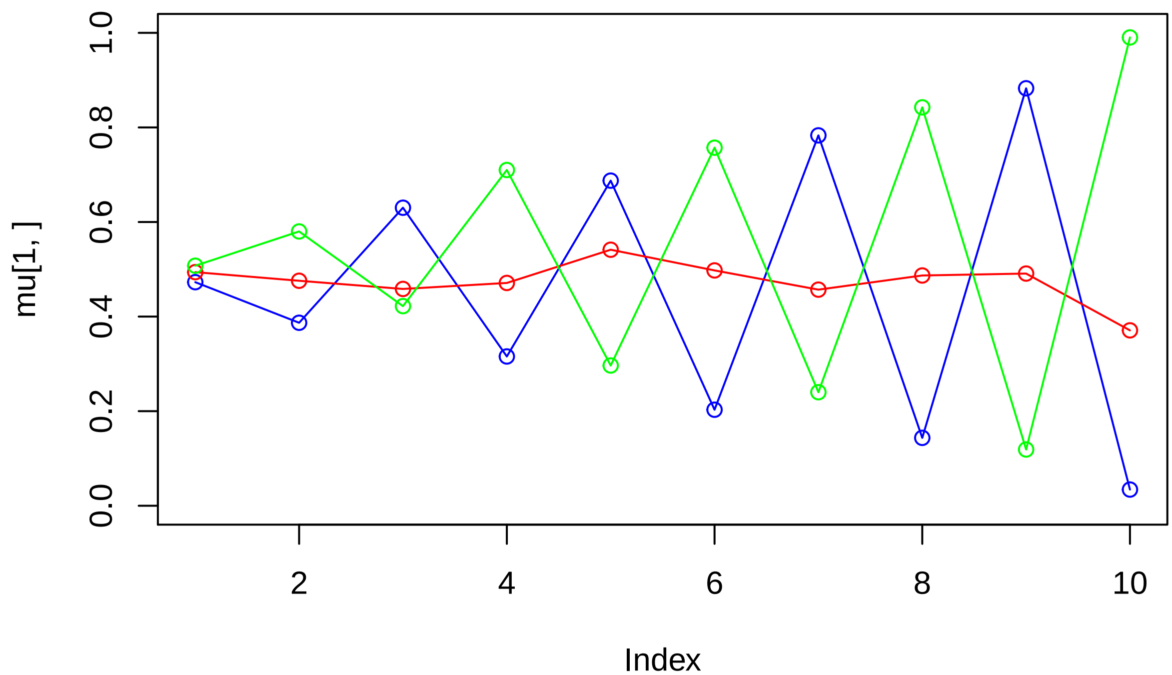

```
## iteration:  23 log likelihood:  -5338.385
```



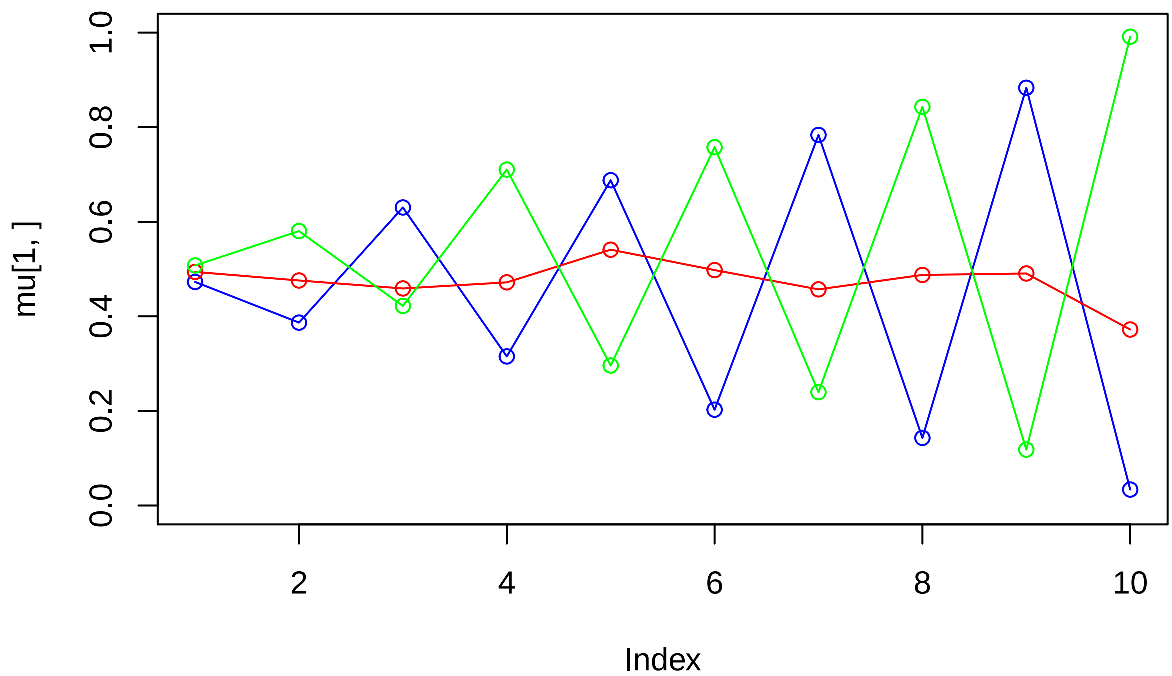

```
## iteration:  24 log likelihood:  -5336.63
```



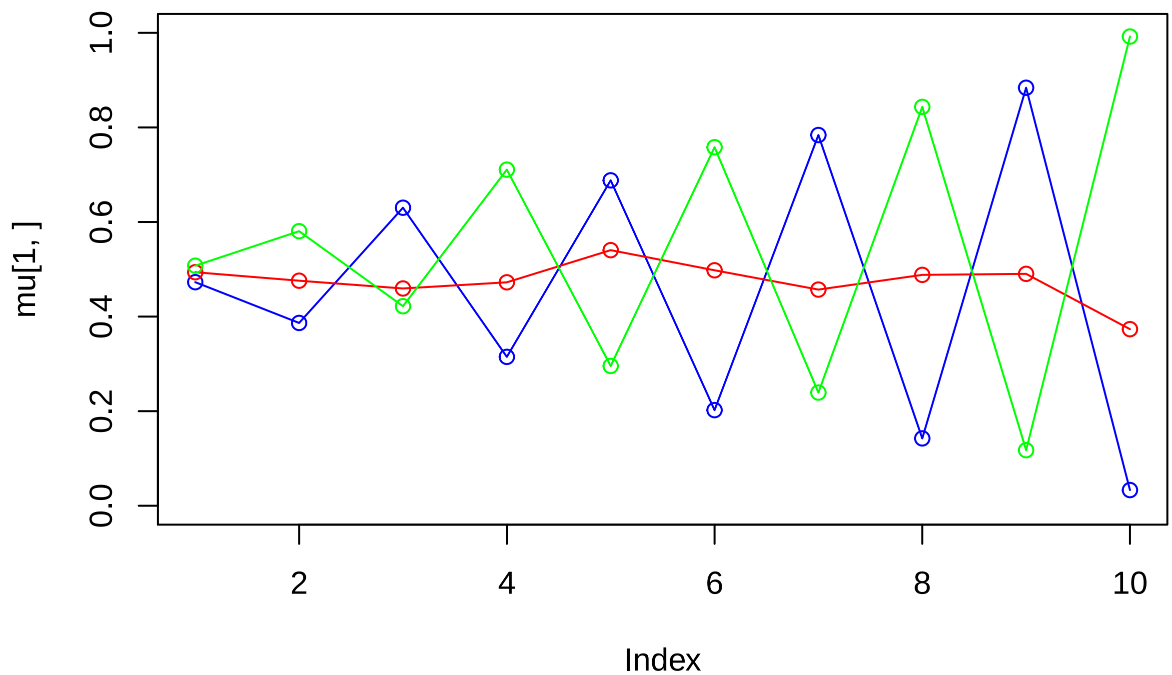

```
## iteration:  25 log likelihood:  -5335.015
```



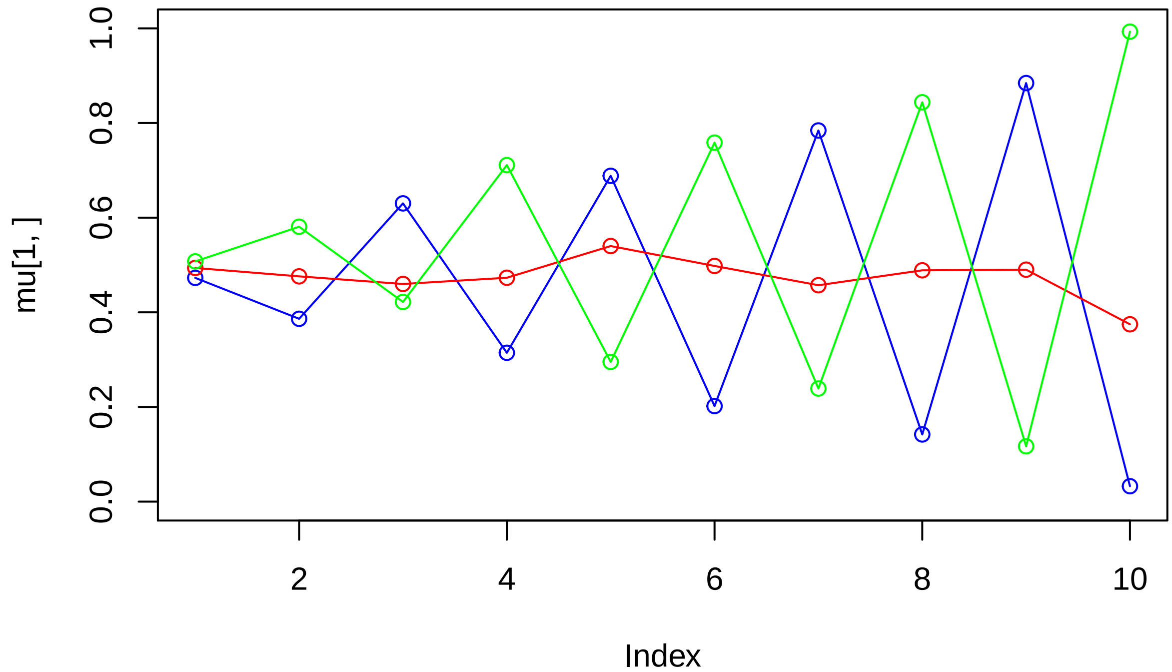

```
## iteration:  26 log likelihood:  -5333.529
```



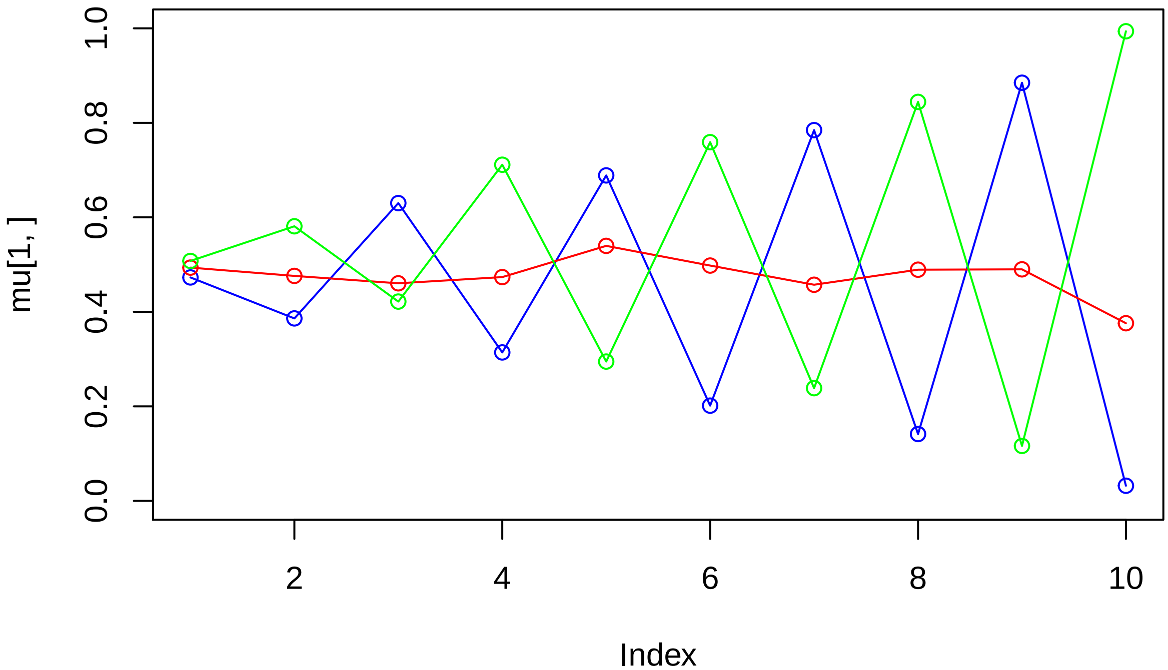

```
## iteration:  27 log likelihood:  -5332.16
```



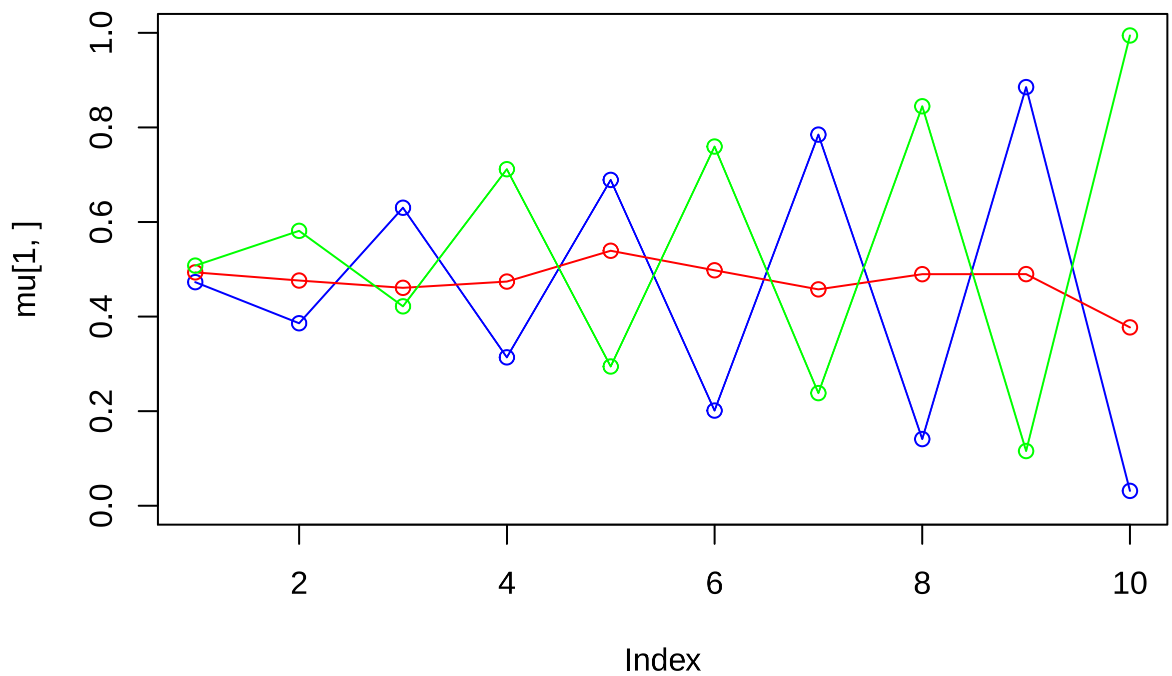

```
## iteration:  28 log likelihood:  -5330.9
```



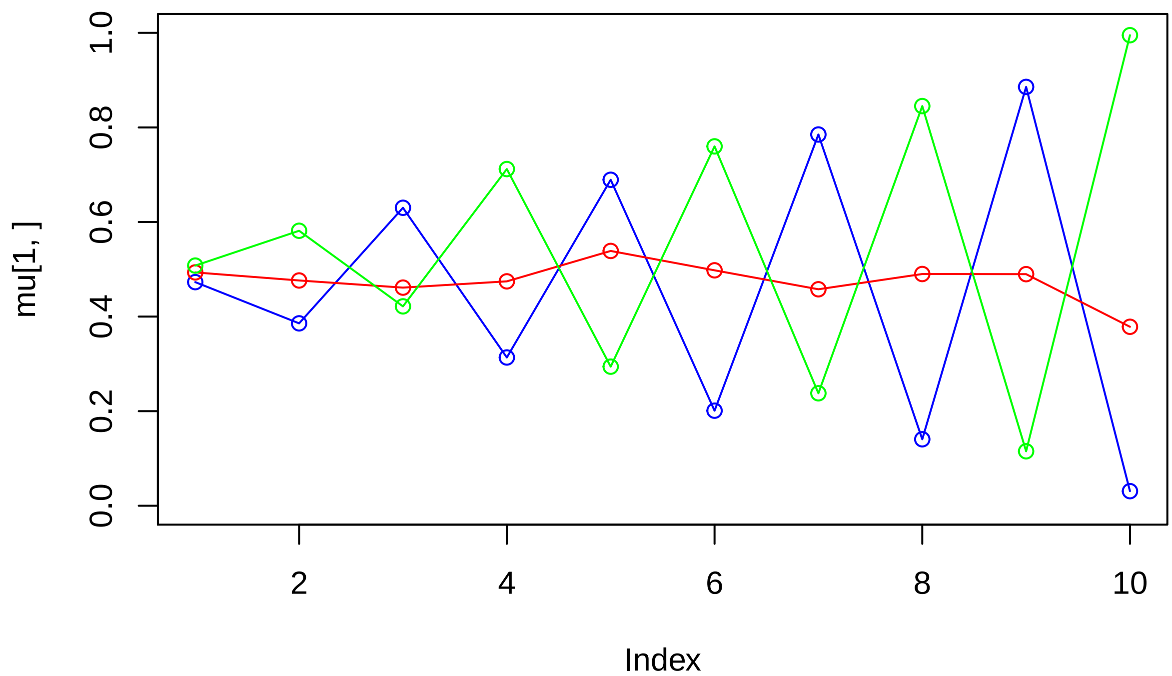

```
## iteration:  29 log likelihood:  -5329.738
```



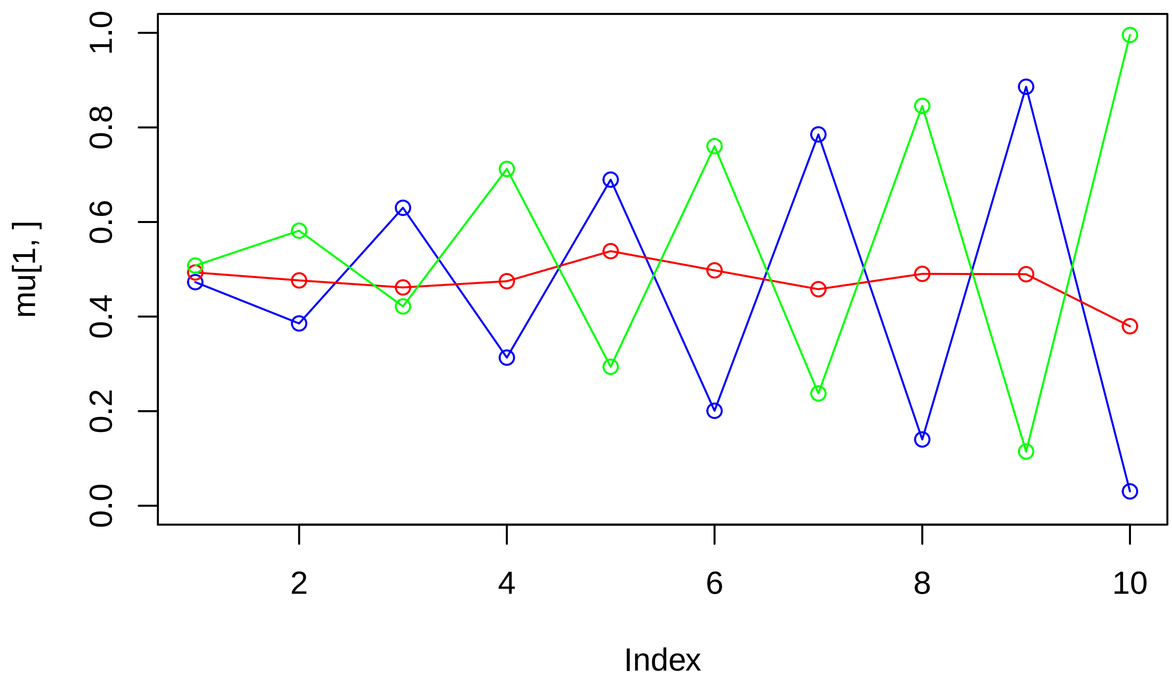

```
## iteration:  30 log likelihood:  -5328.666
```



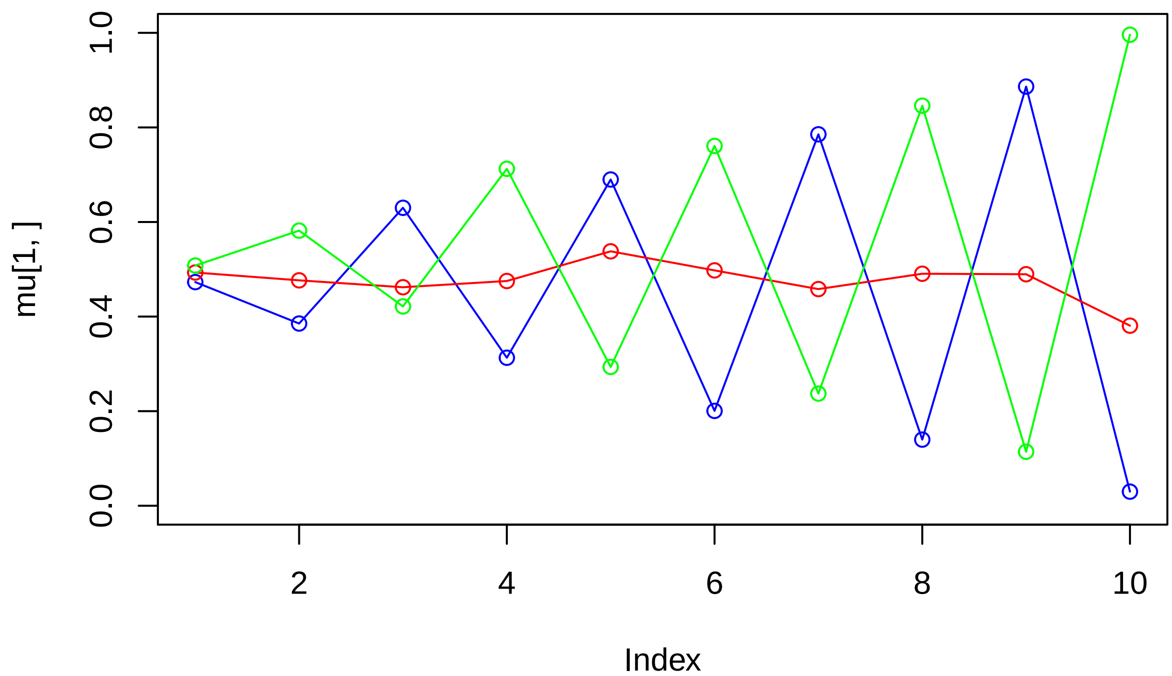

```
## iteration:  31 log likelihood:  -5327.676
```



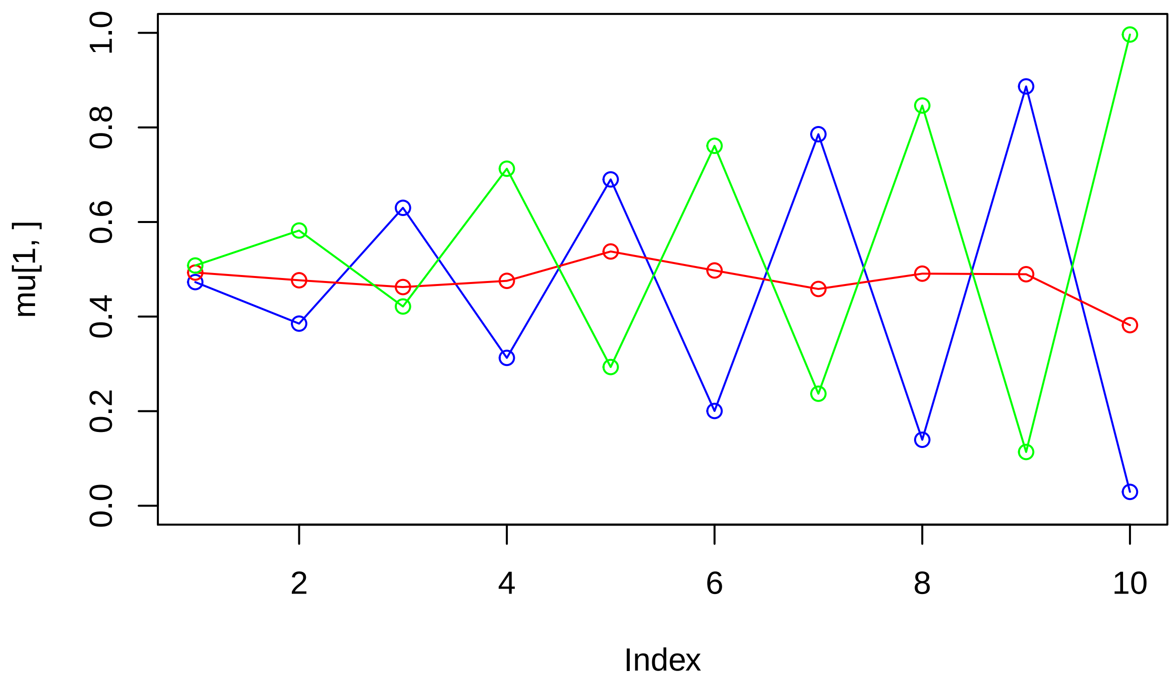

```
## iteration:  32 log likelihood:  -5326.762
```



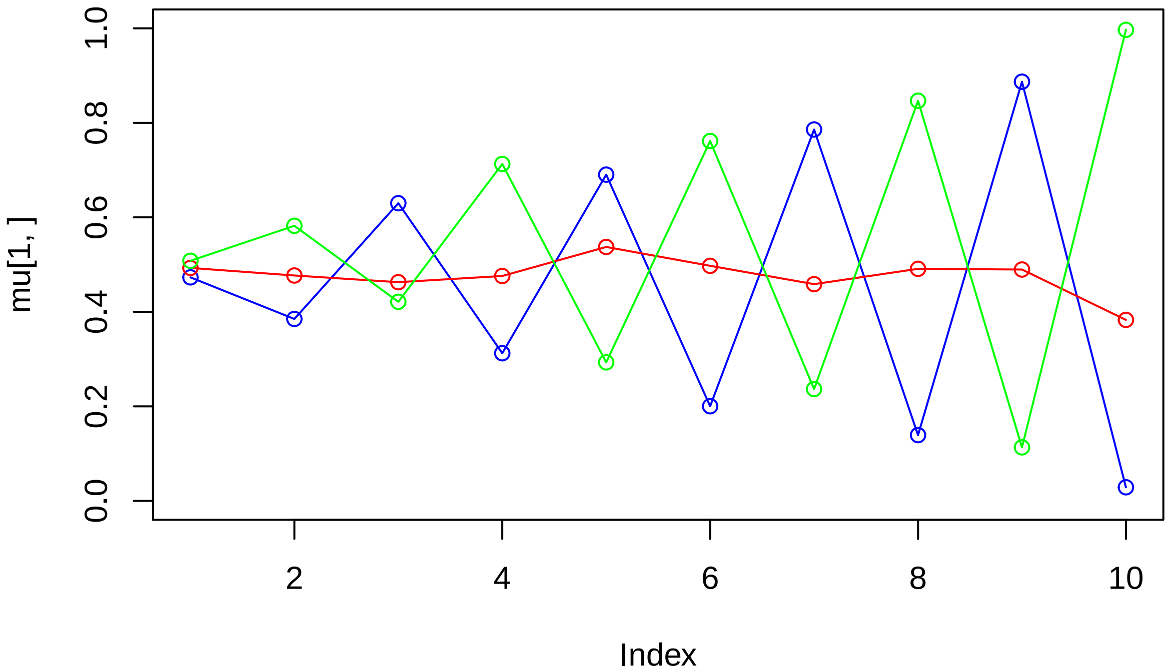

```
## iteration:  33 log likelihood:  -5325.917
```



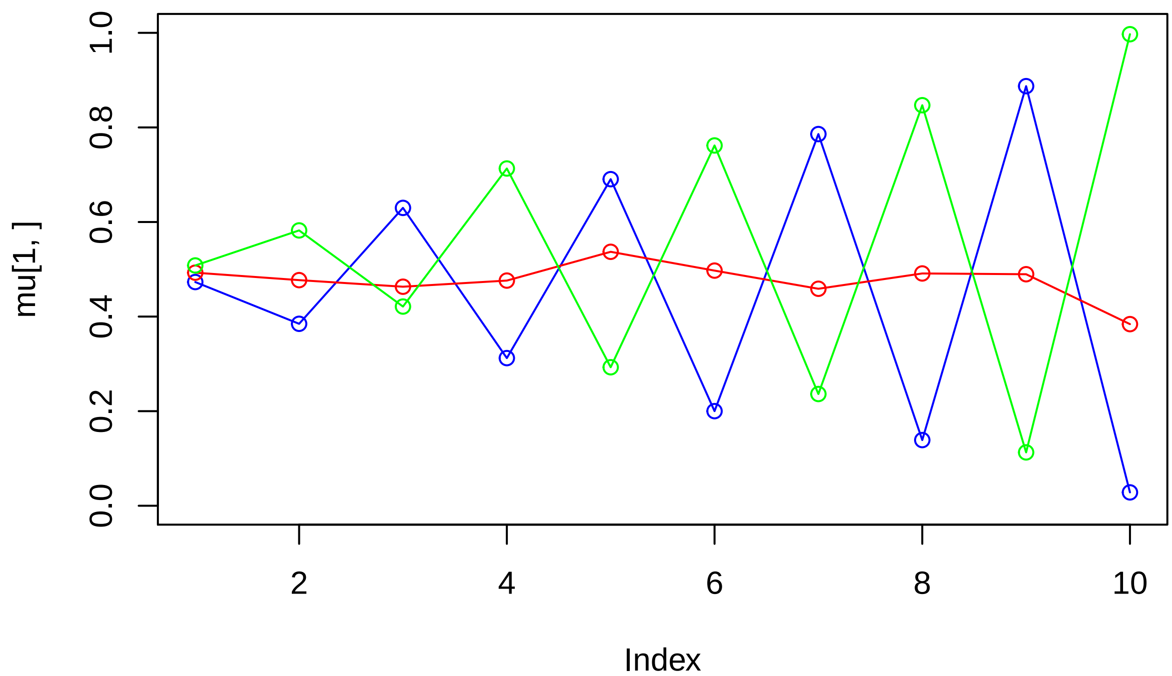

```
## iteration:  34 log likelihood:  -5325.135
```



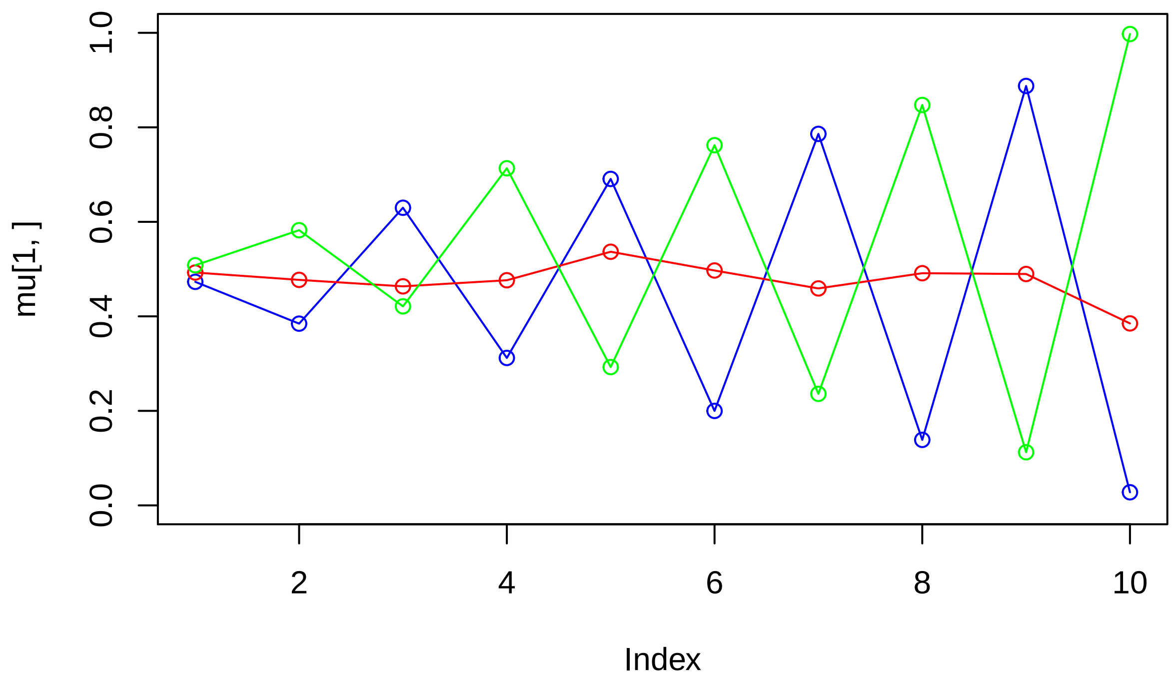

```
## iteration:  35 log likelihood:  -5324.41
```



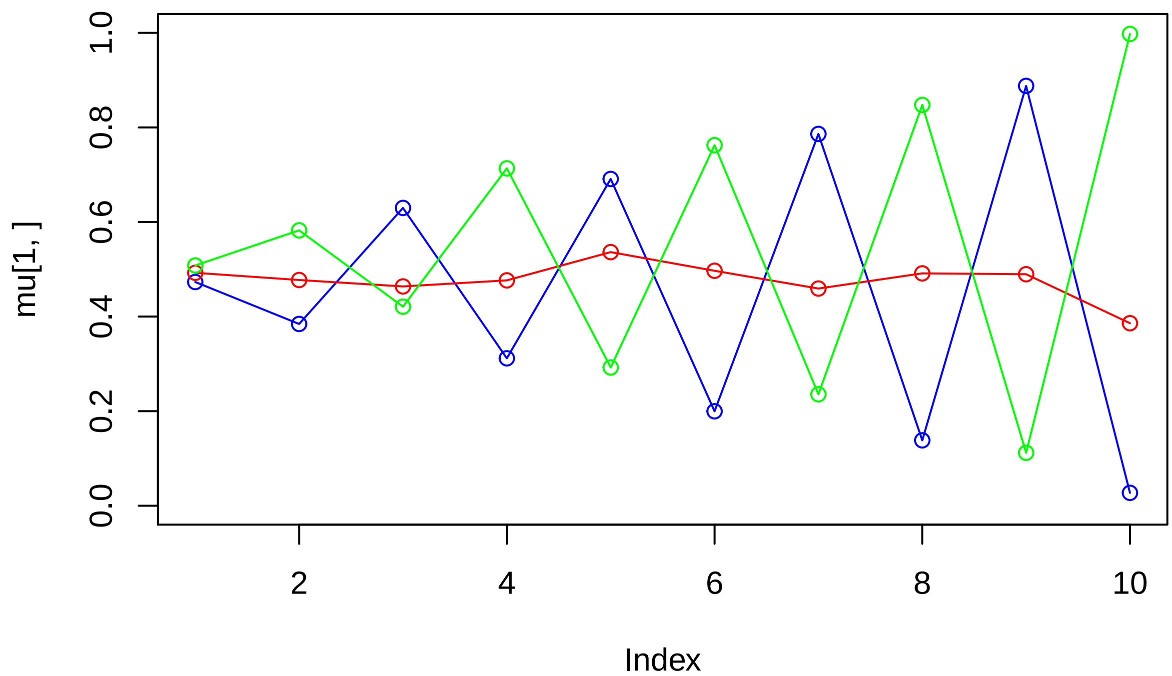

```
## iteration:  36 log likelihood:  -5323.739
```



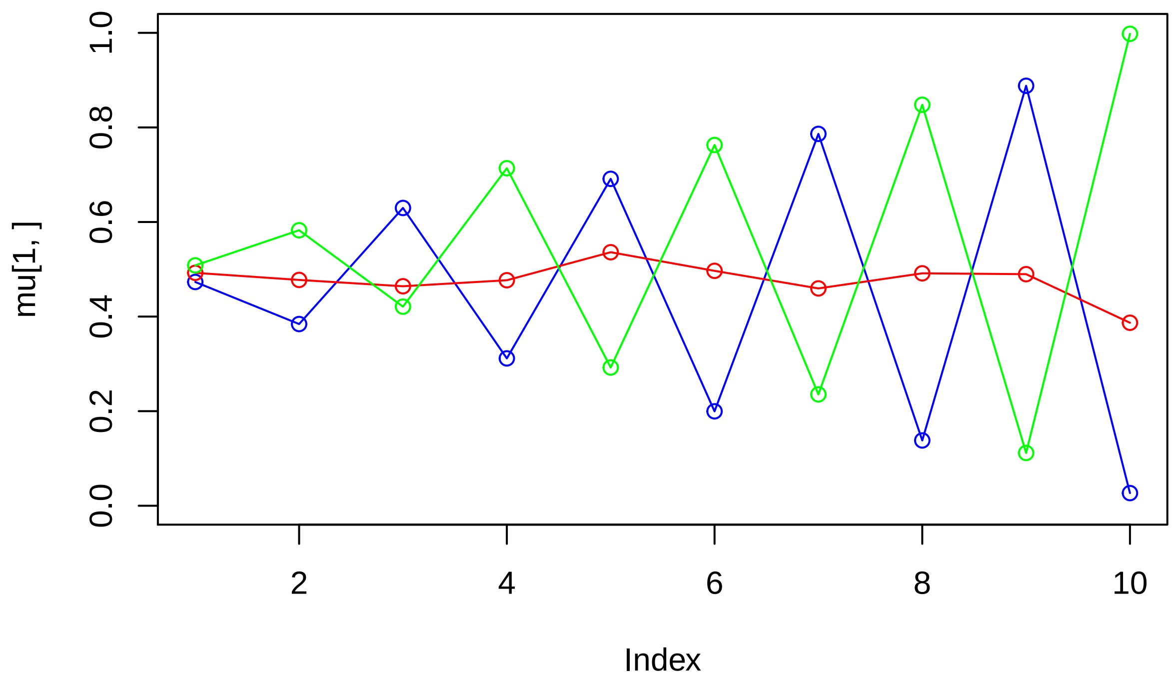

```
## iteration:  37 log likelihood:  -5323.115
```



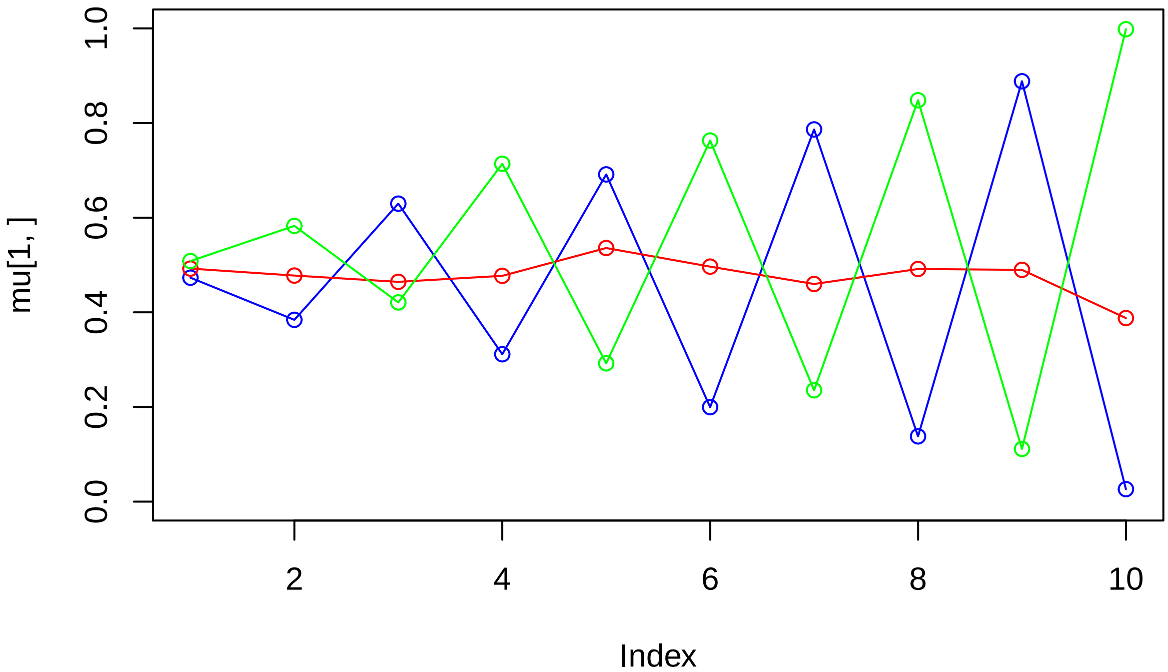

```
## iteration:  38 log likelihood:  -5322.537
```



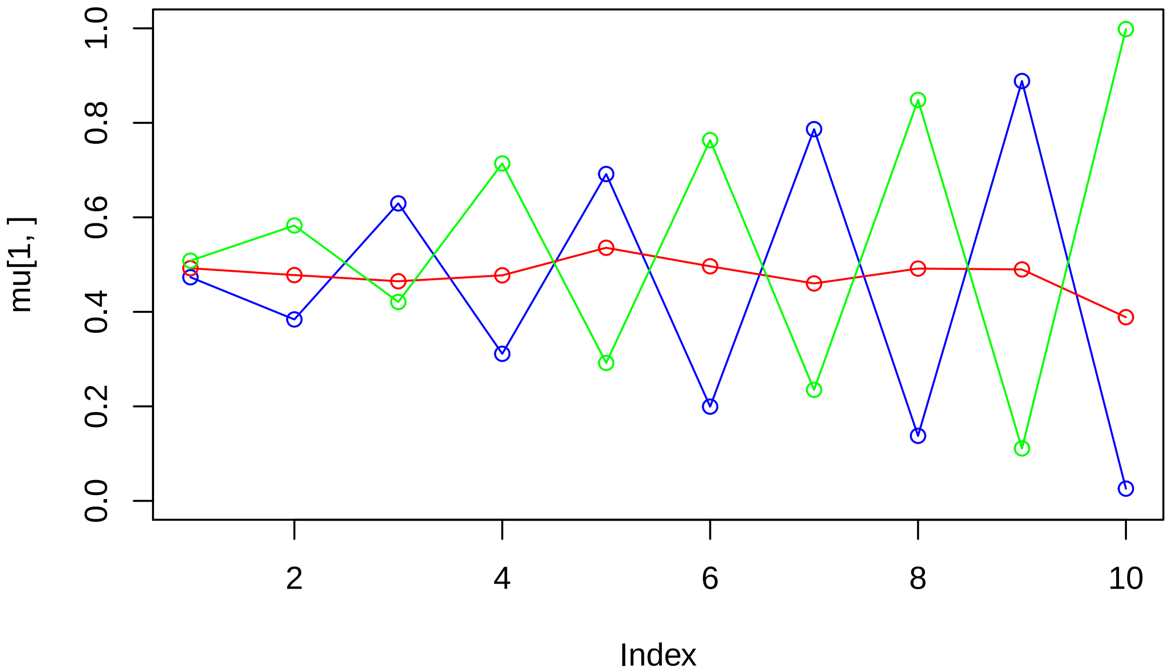

```
## iteration:  39 log likelihood:  -5321.999
```



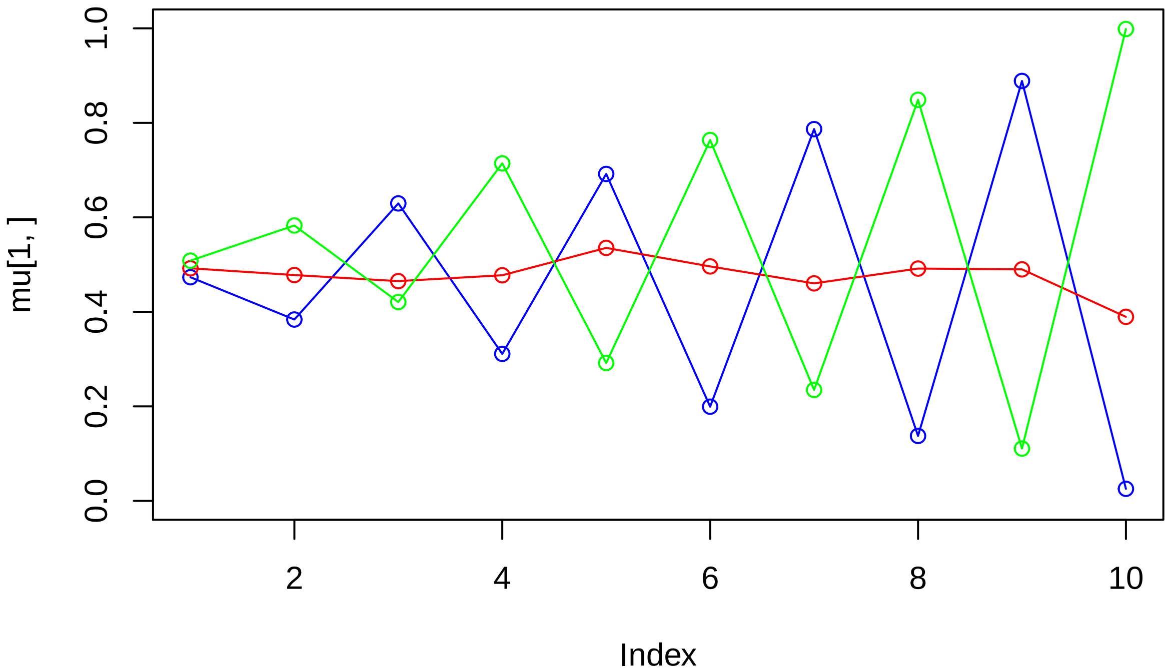

```
## iteration:  40 log likelihood:  -5321.498
```



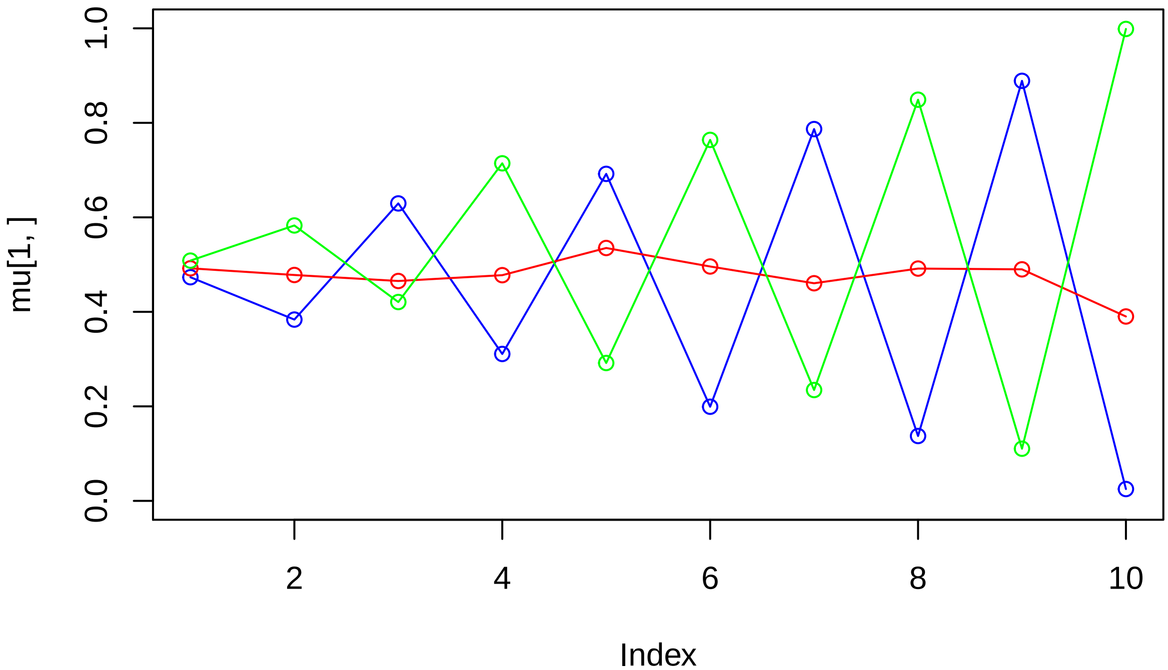

```
## iteration:  41 log likelihood:  -5321.031
```



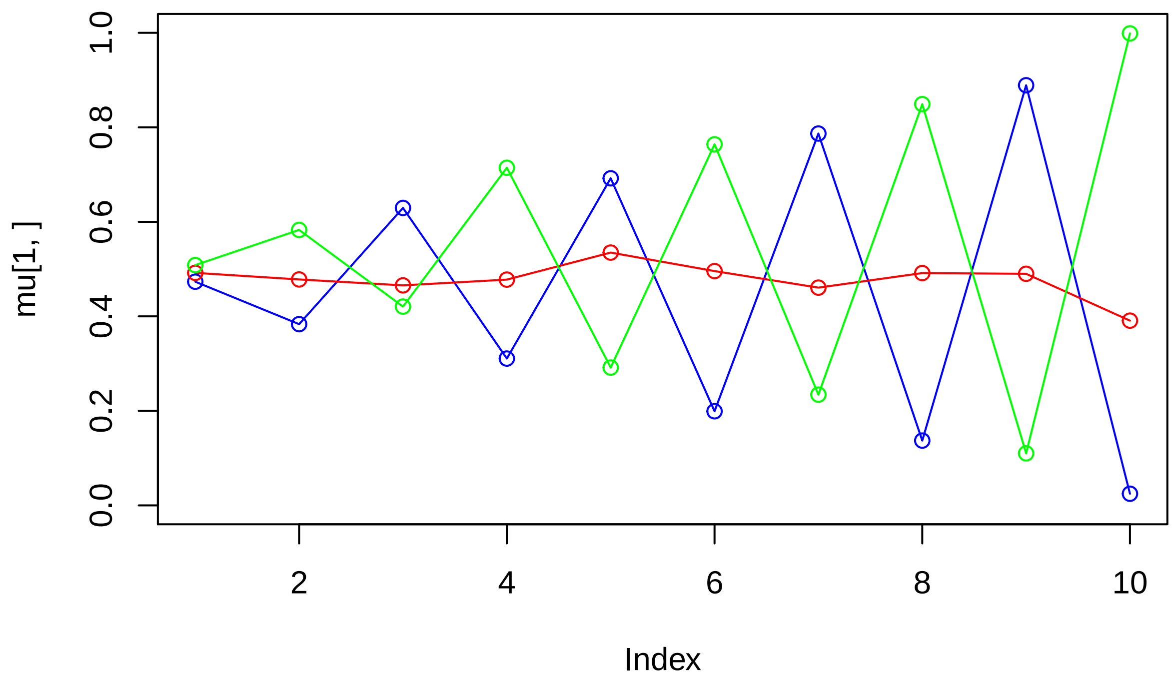

```
## iteration:  42 log likelihood:  -5320.596
```



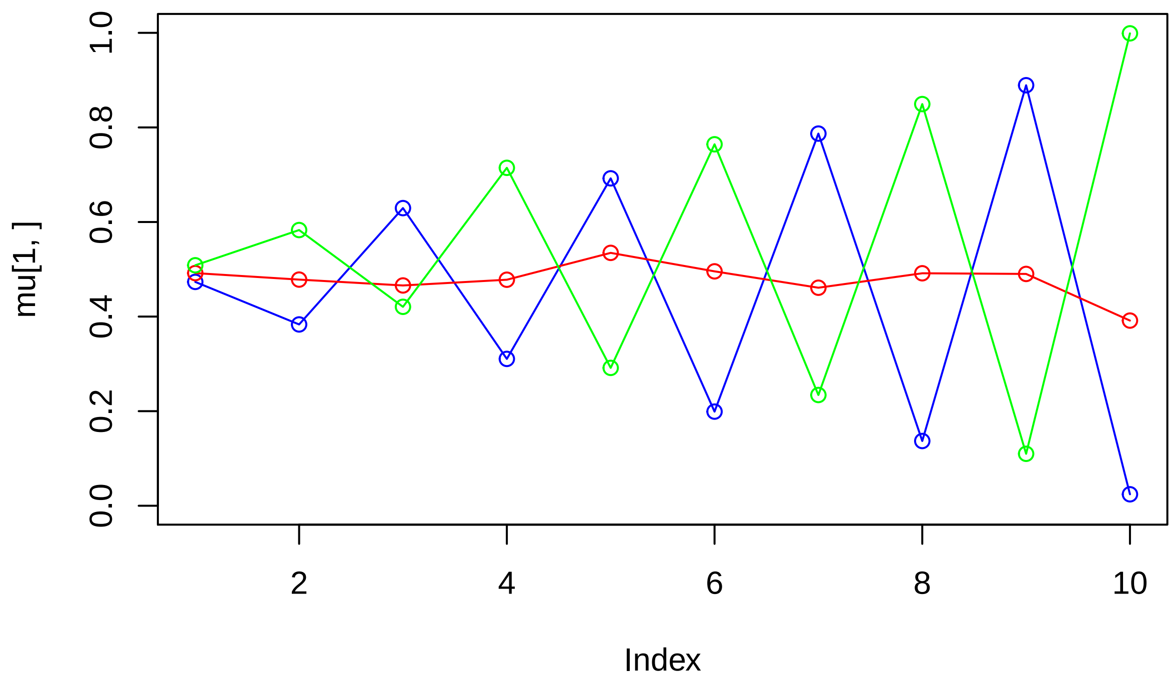

```
## iteration:  43 log likelihood:  -5320.19
```



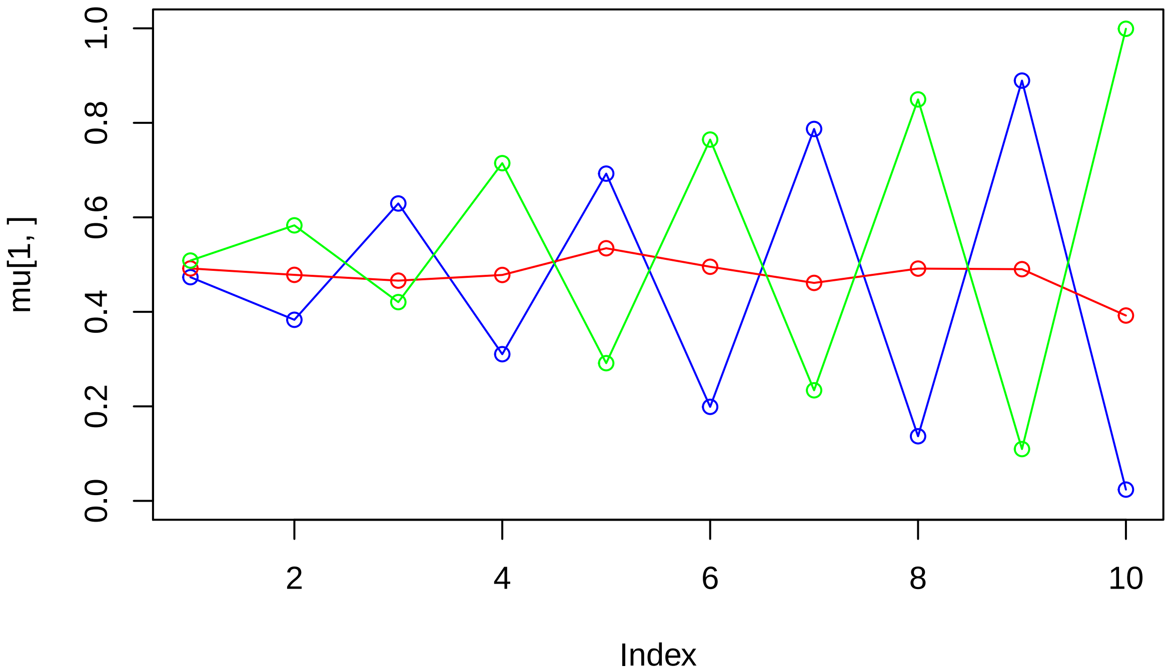

```
## iteration:  44 log likelihood:  -5319.81
```



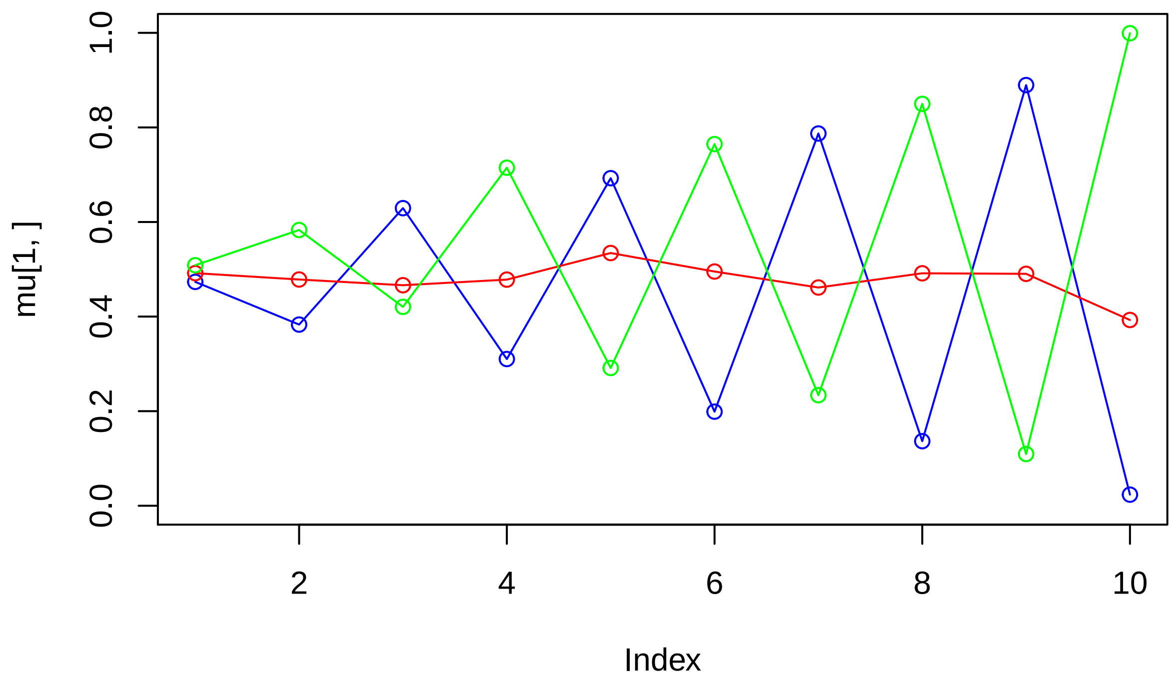

```
## iteration:  45 log likelihood:  -5319.454
```



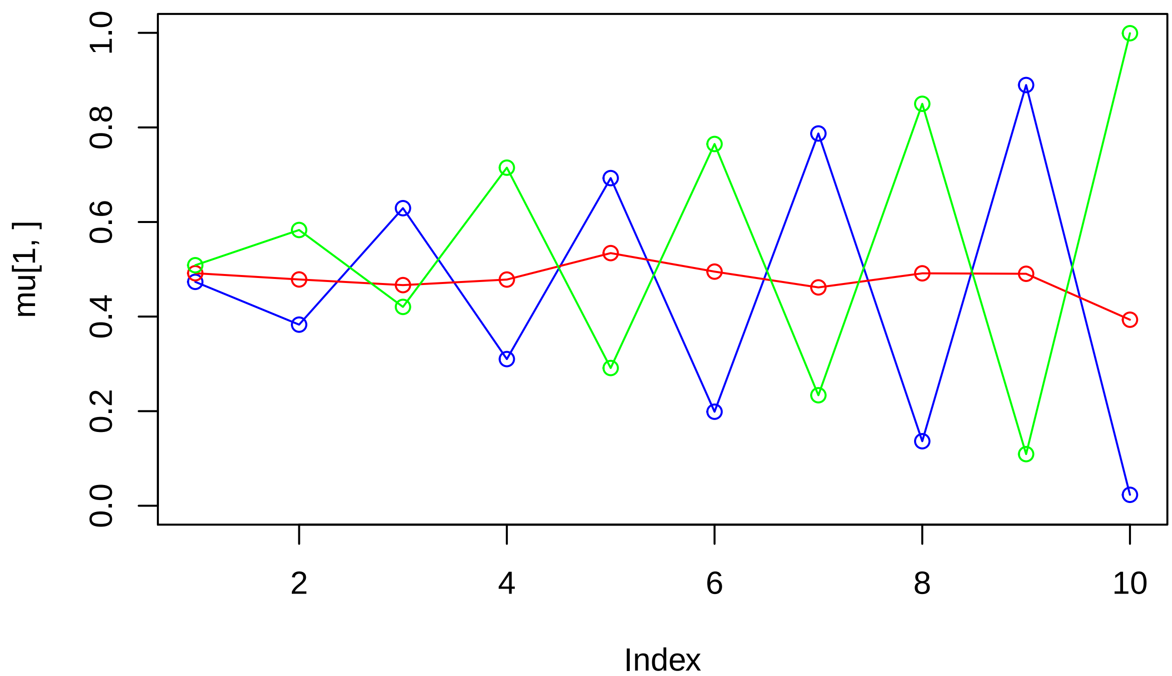

```
## iteration:  46 log likelihood:  -5319.121
```



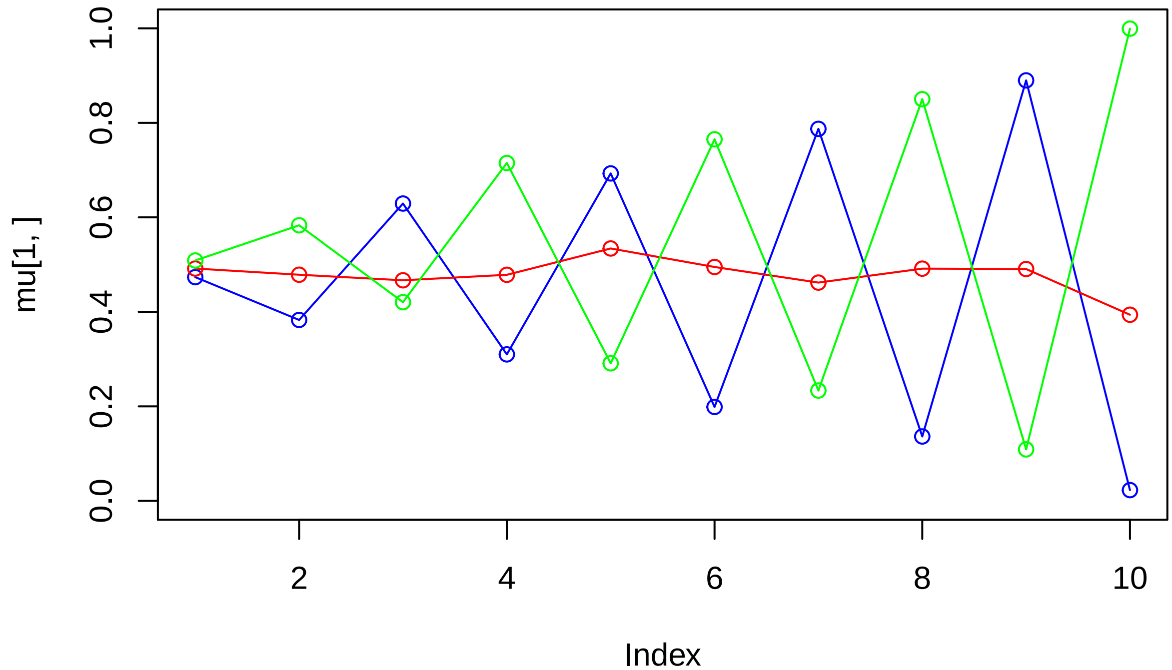

```
## iteration:  47 log likelihood:  -5318.809
```



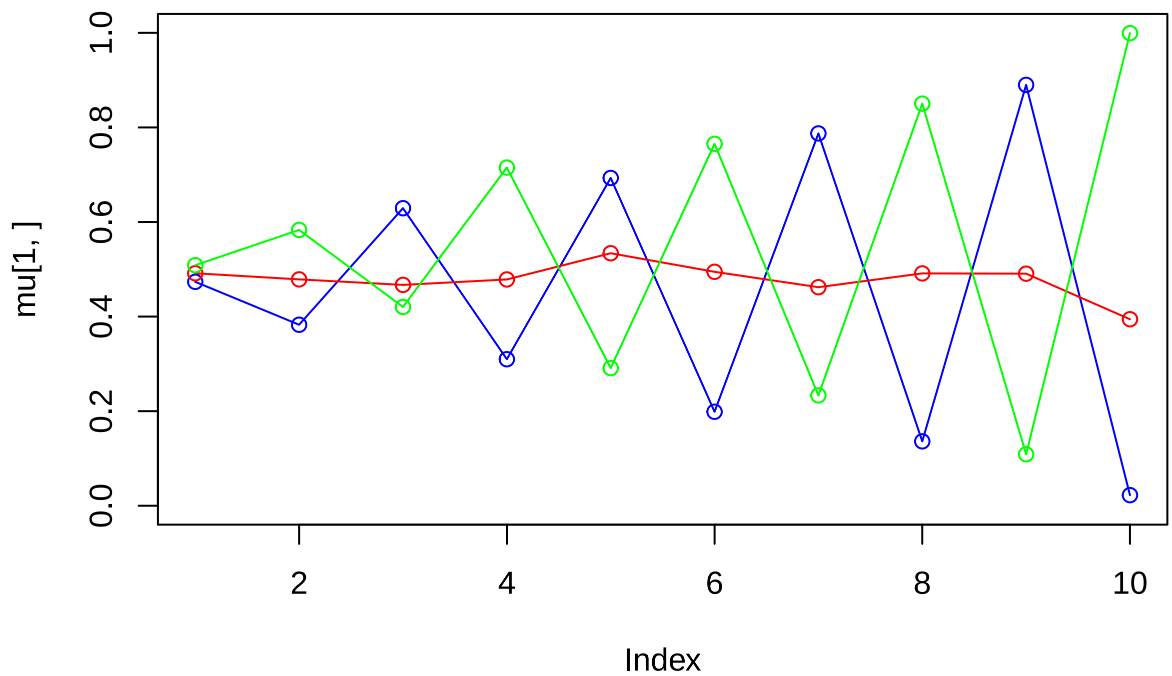

```
## iteration:  48 log likelihood:  -5318.515
```



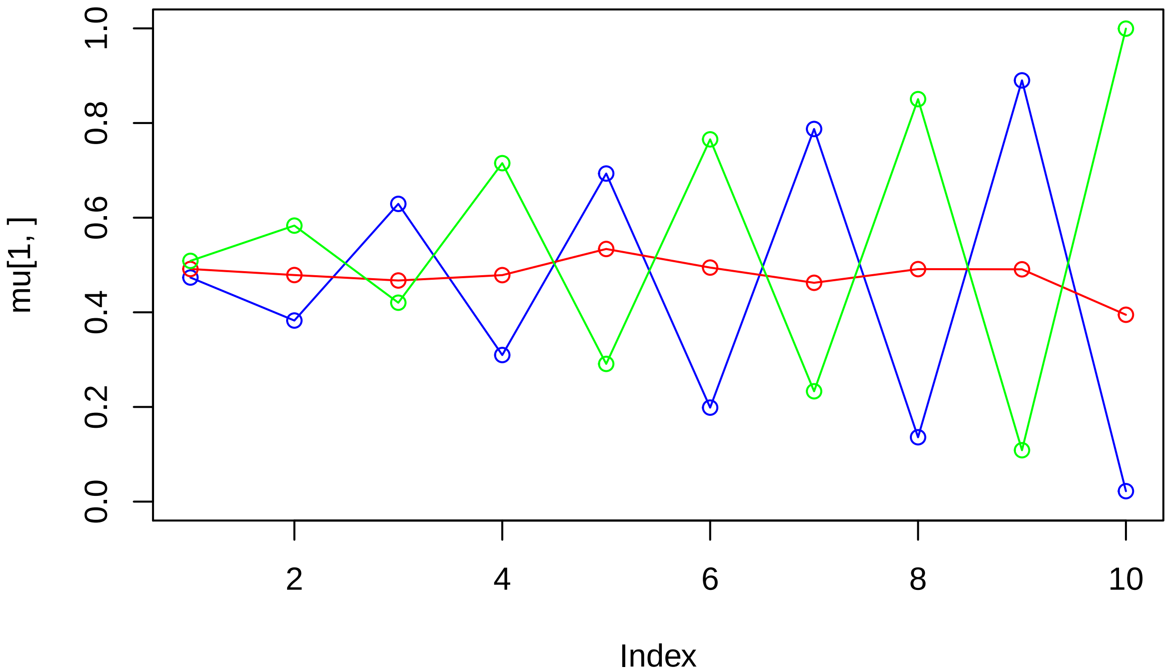

```
## iteration:  49 log likelihood:  -5318.239
```



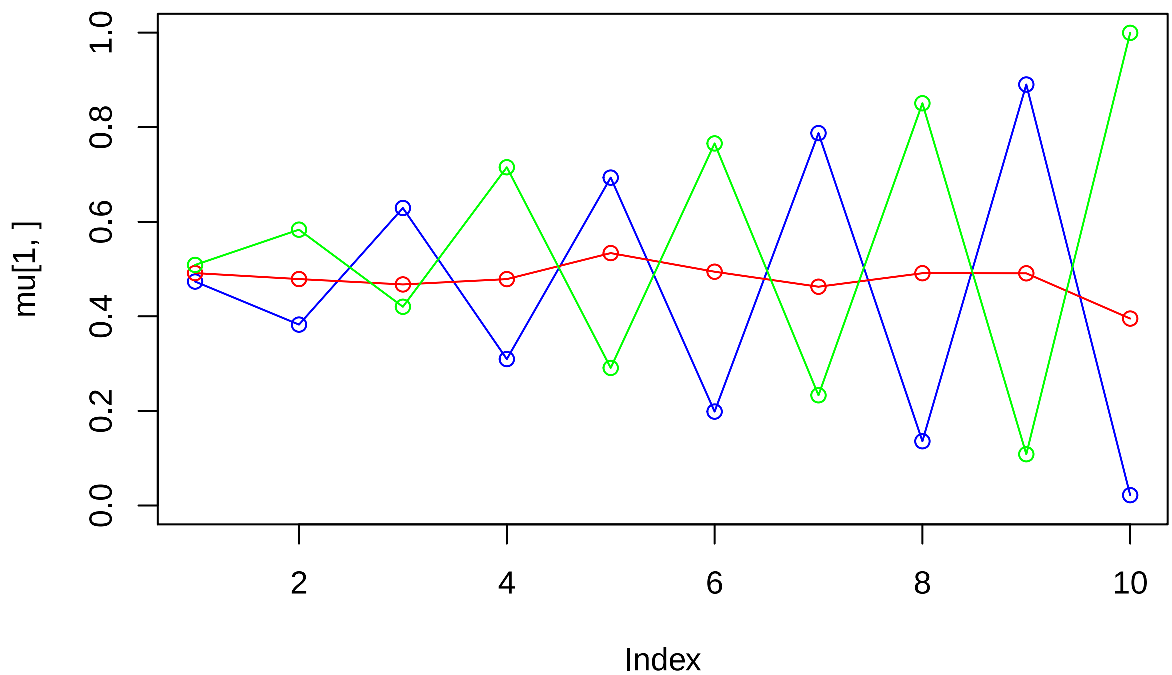

```
## iteration:  50 log likelihood:  -5317.979
```



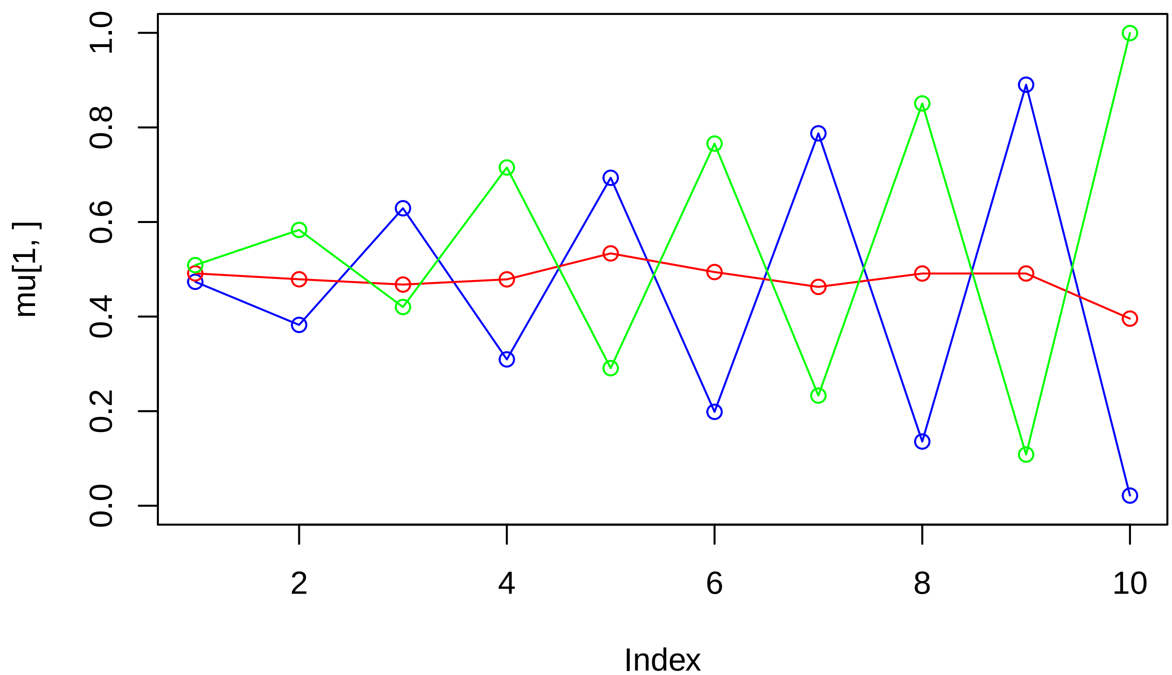

```
## iteration:  51 log likelihood:  -5317.734
```



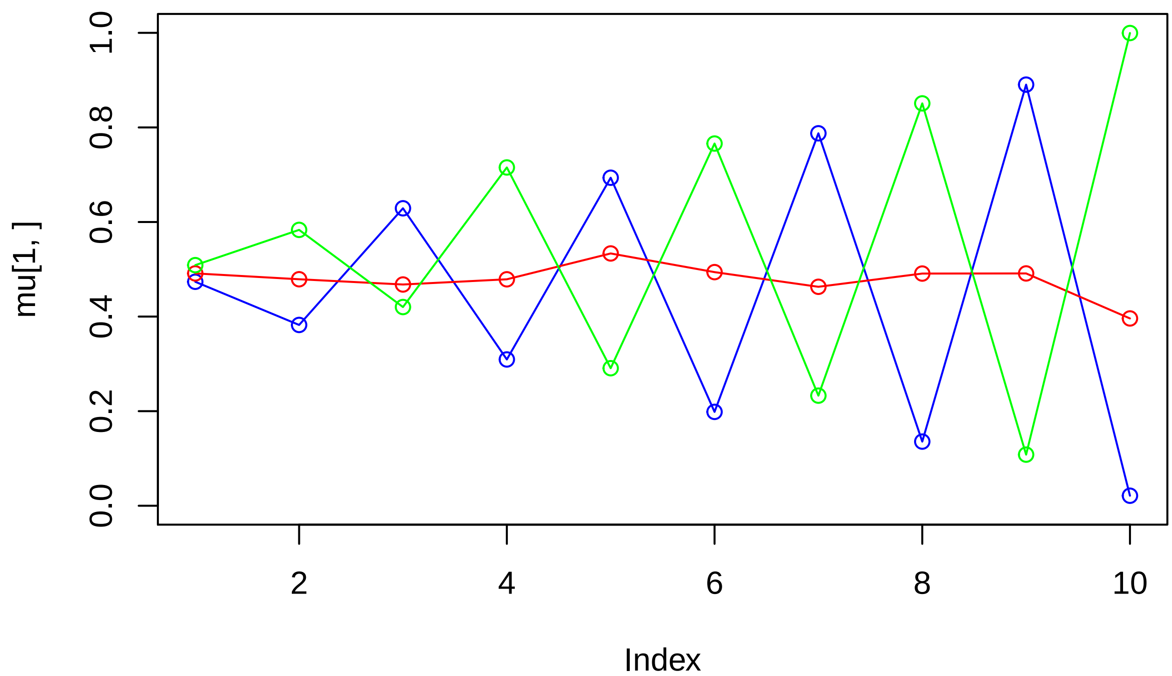

```
## iteration:  52 log likelihood:  -5317.503
```



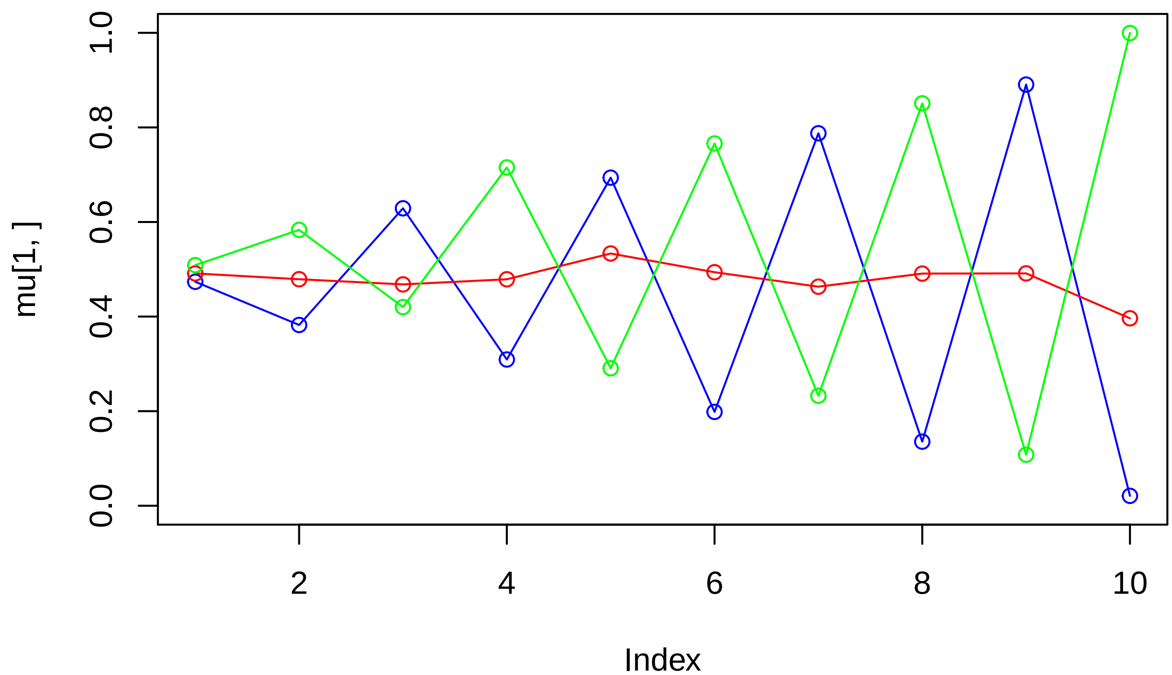

```
## iteration:  53 log likelihood:  -5317.284
```



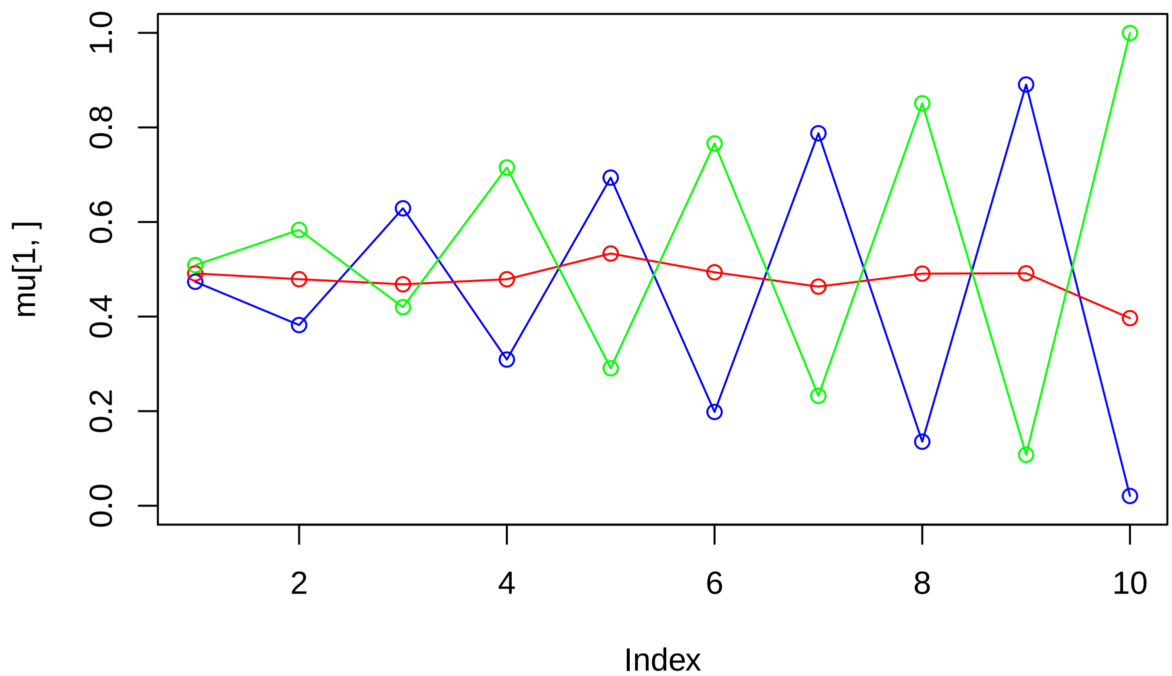

```
## iteration:  54 log likelihood:  -5317.077
```



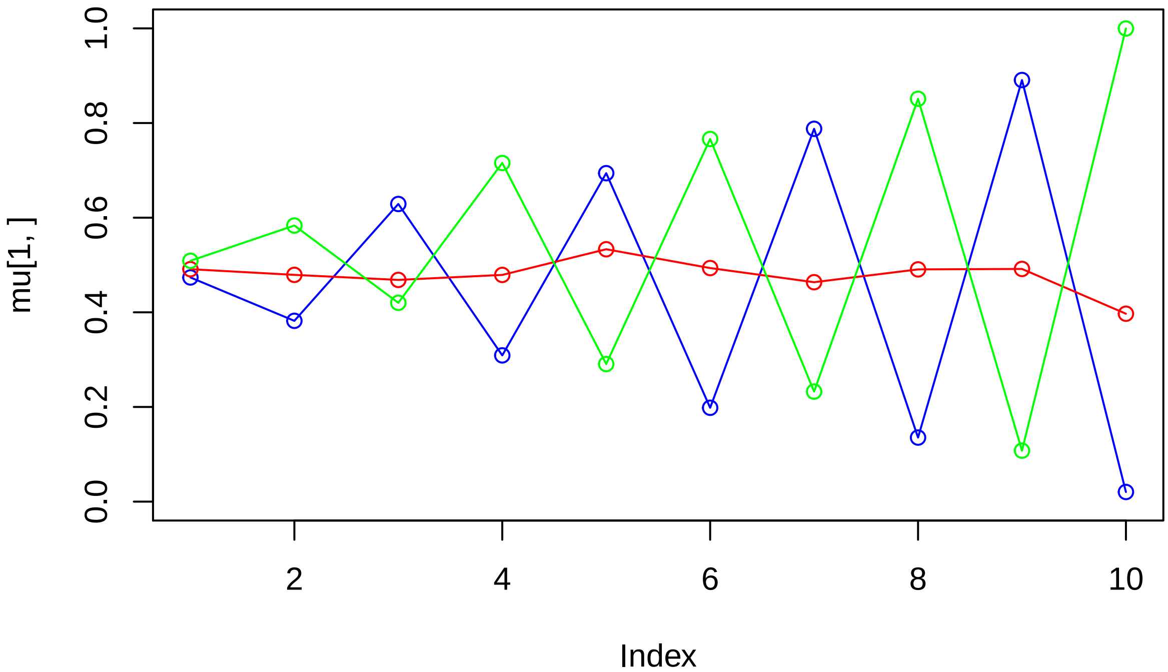

```
## iteration:  55 log likelihood:  -5316.881
```



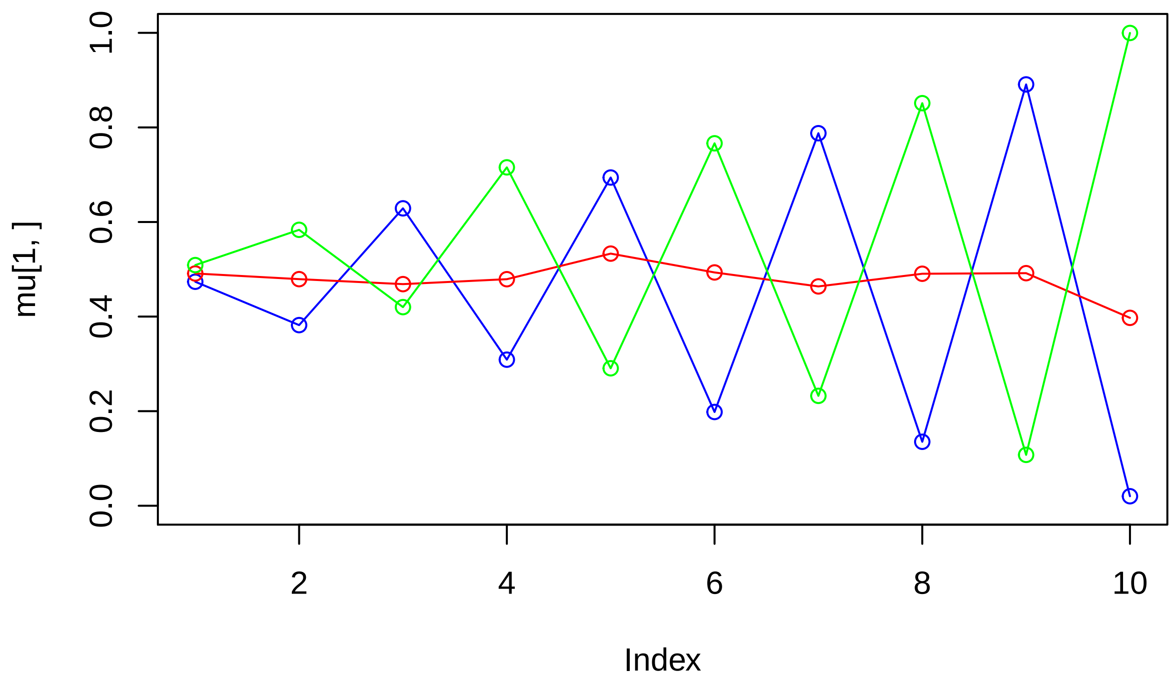

```
## iteration:  56 log likelihood:  -5316.695
```



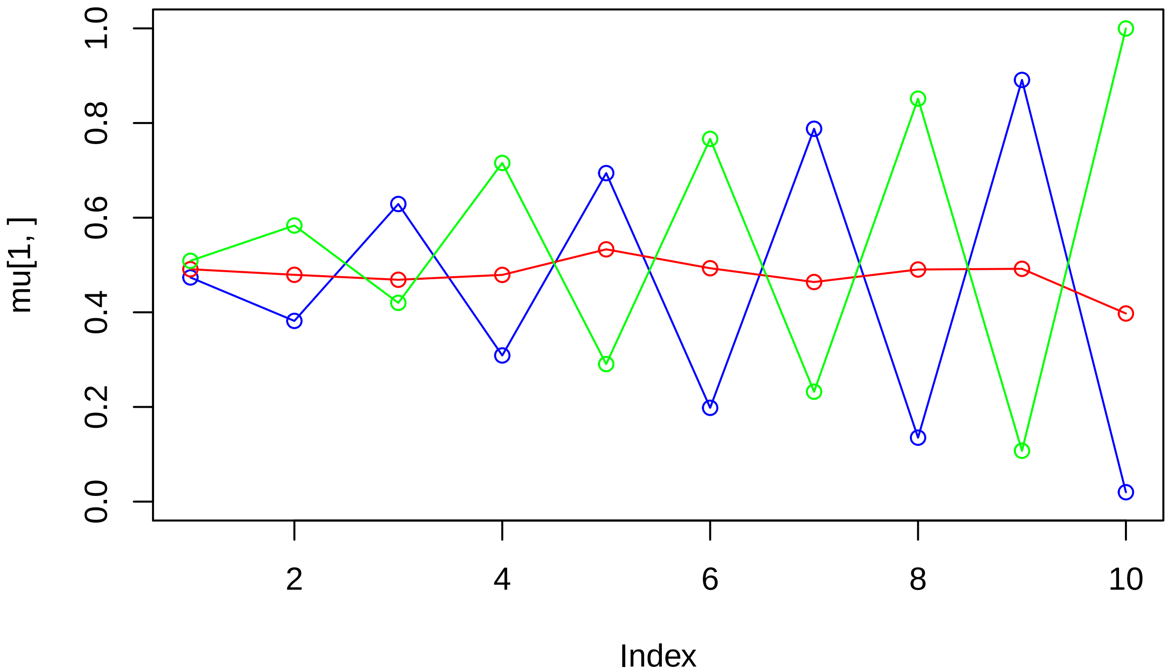

```
## iteration:  57 log likelihood:  -5316.518
```



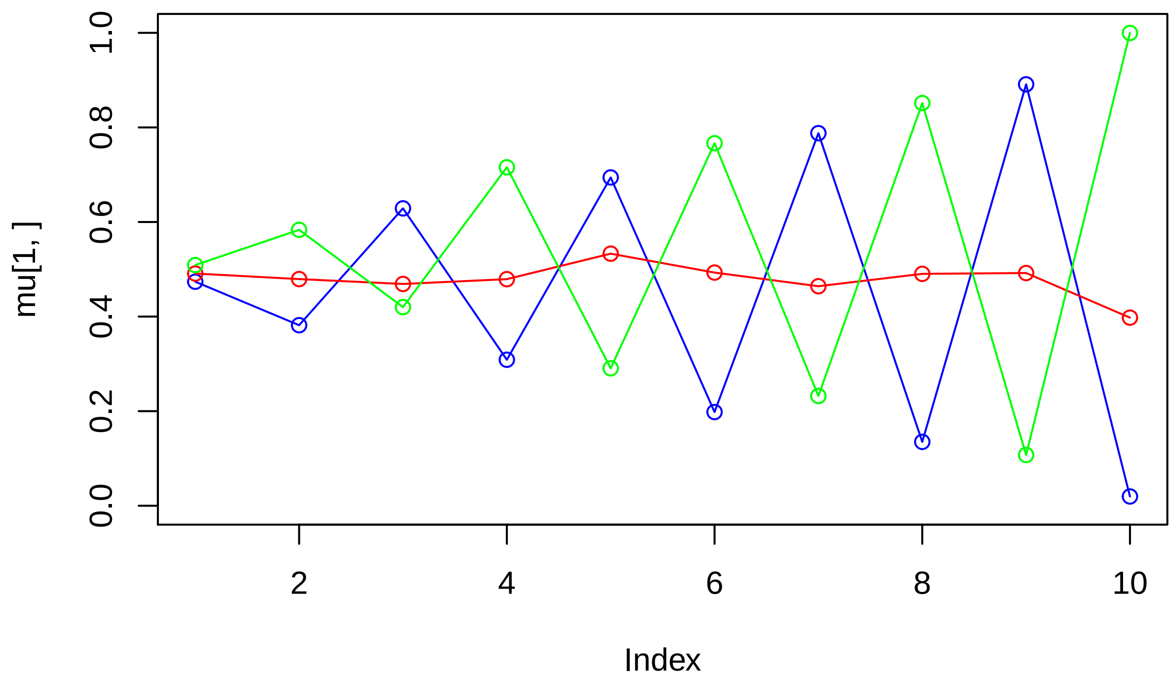

```
## iteration:  58 log likelihood:  -5316.349
```



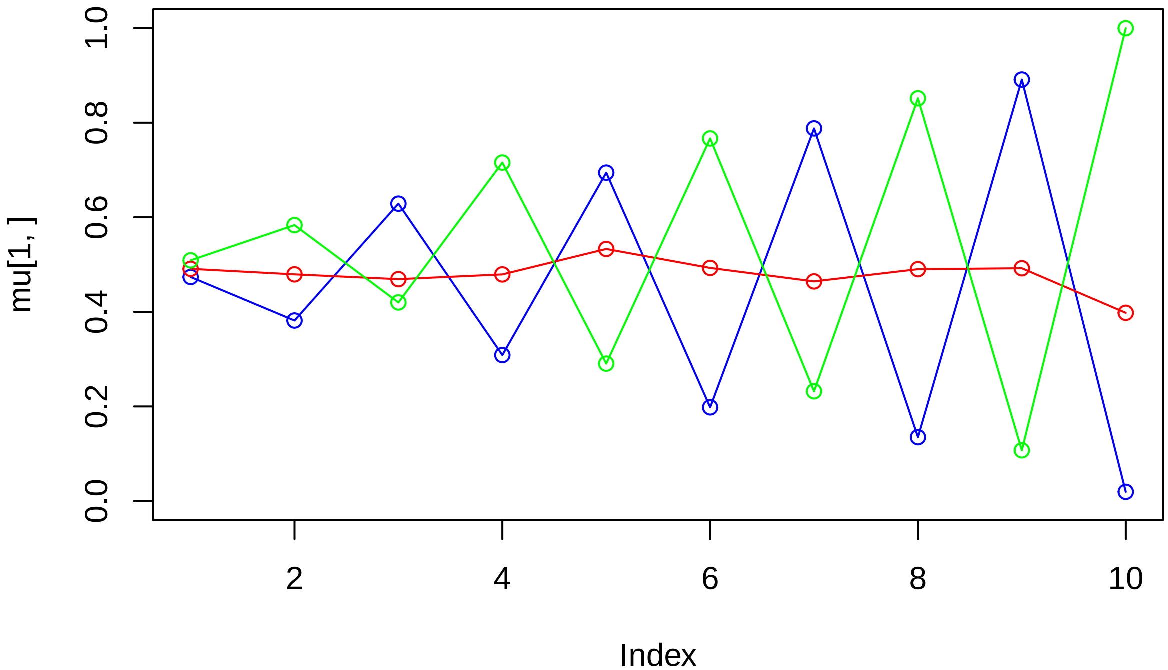

```
## iteration:  59 log likelihood:  -5316.189
```



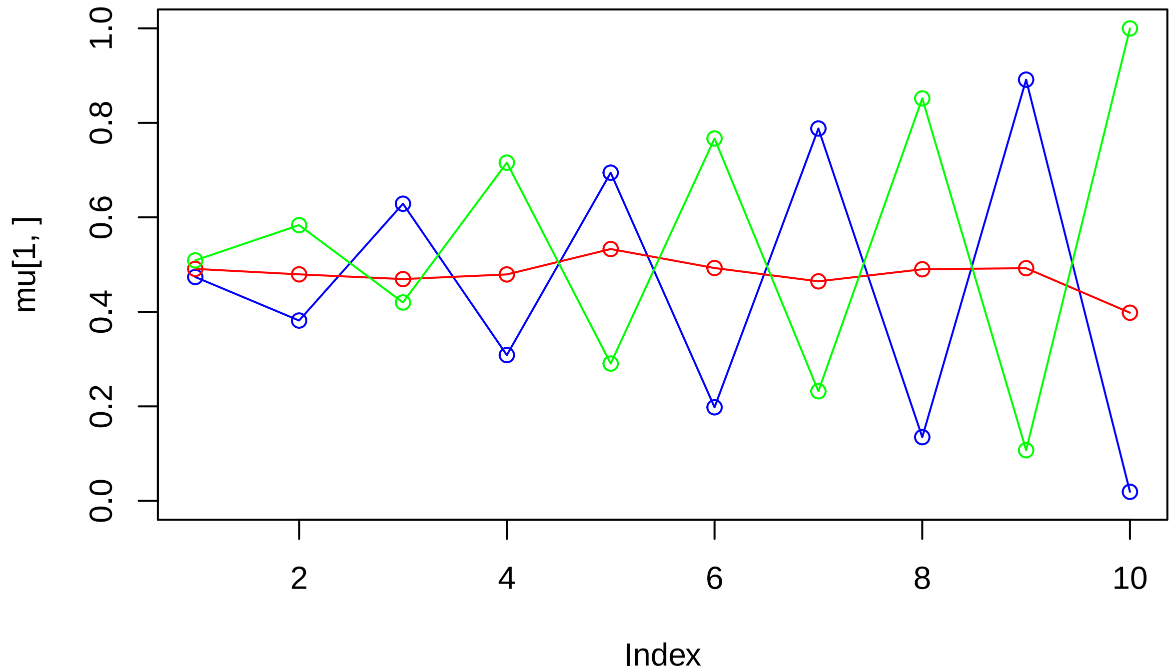

```
## iteration:  60 log likelihood:  -5316.036
```



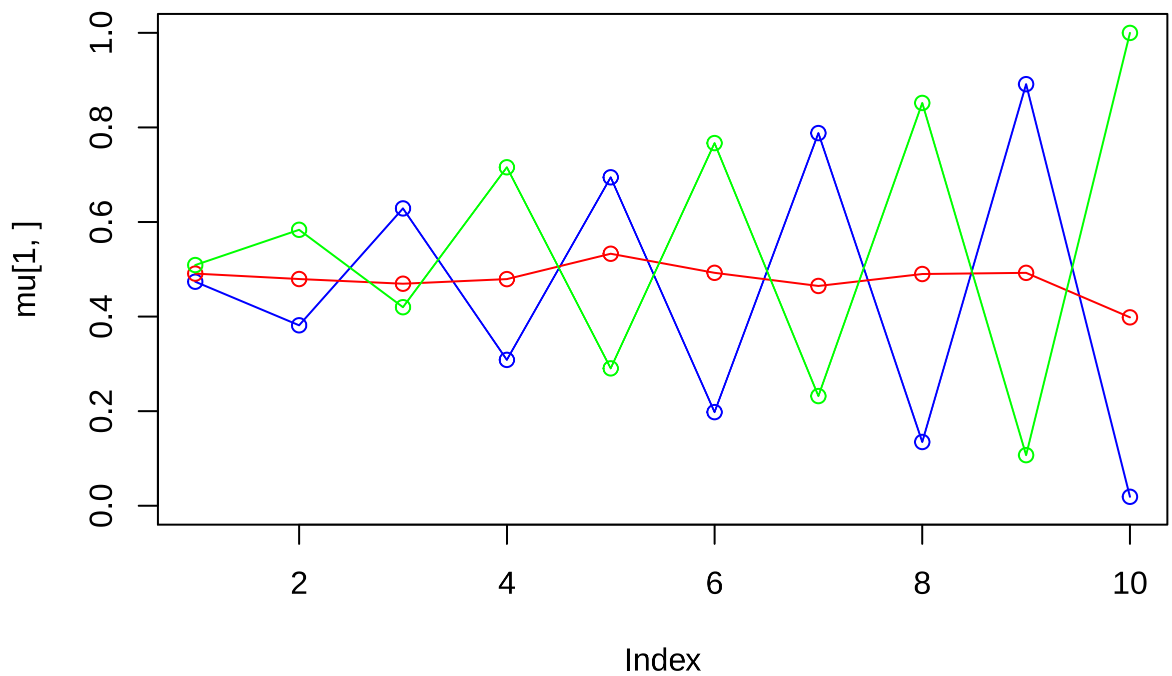

```
## iteration:  61 log likelihood:  -5315.89
```



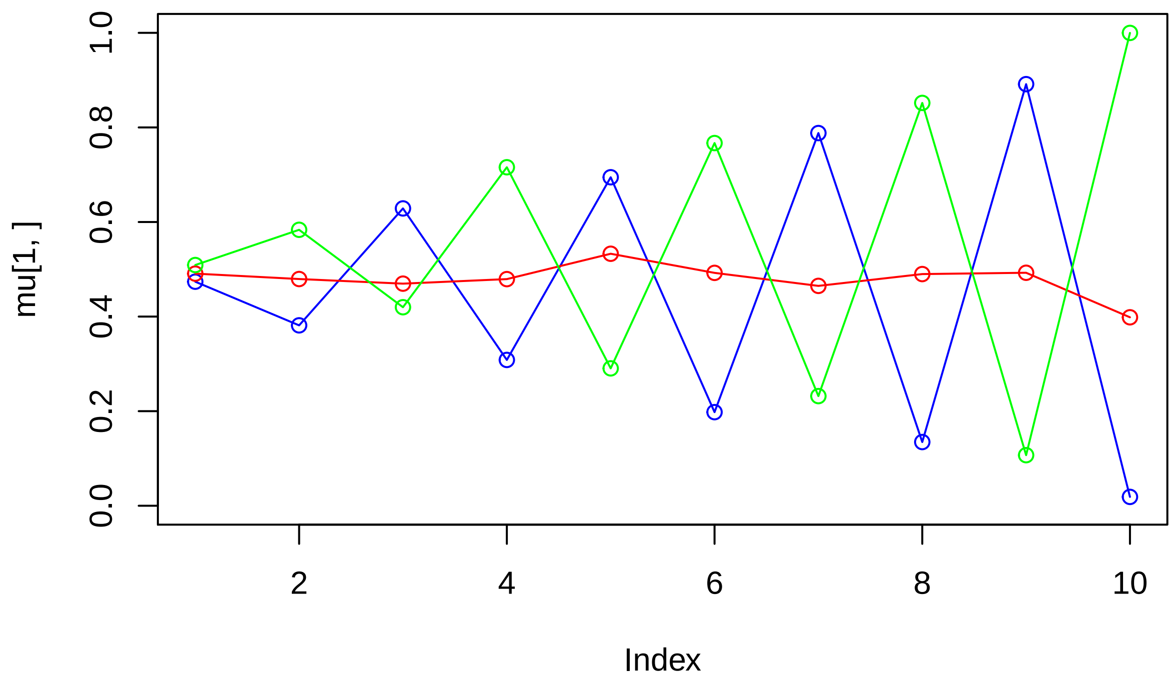

```
## iteration:  62 log likelihood:  -5315.75
```



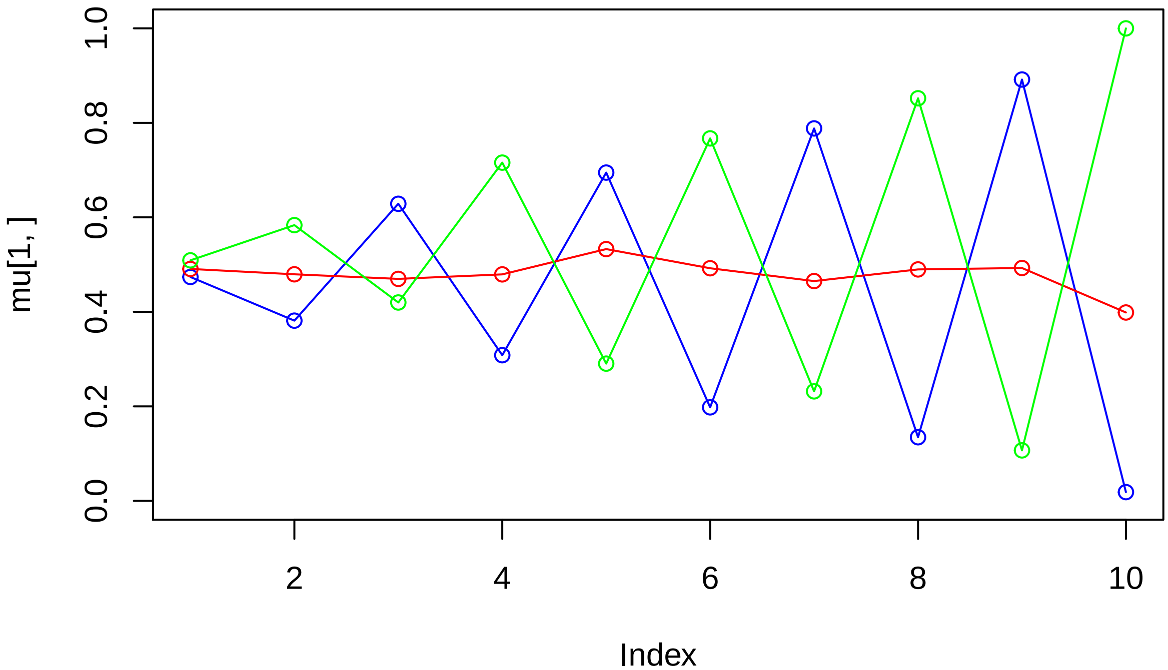

```
## iteration:  63 log likelihood:  -5315.616
```



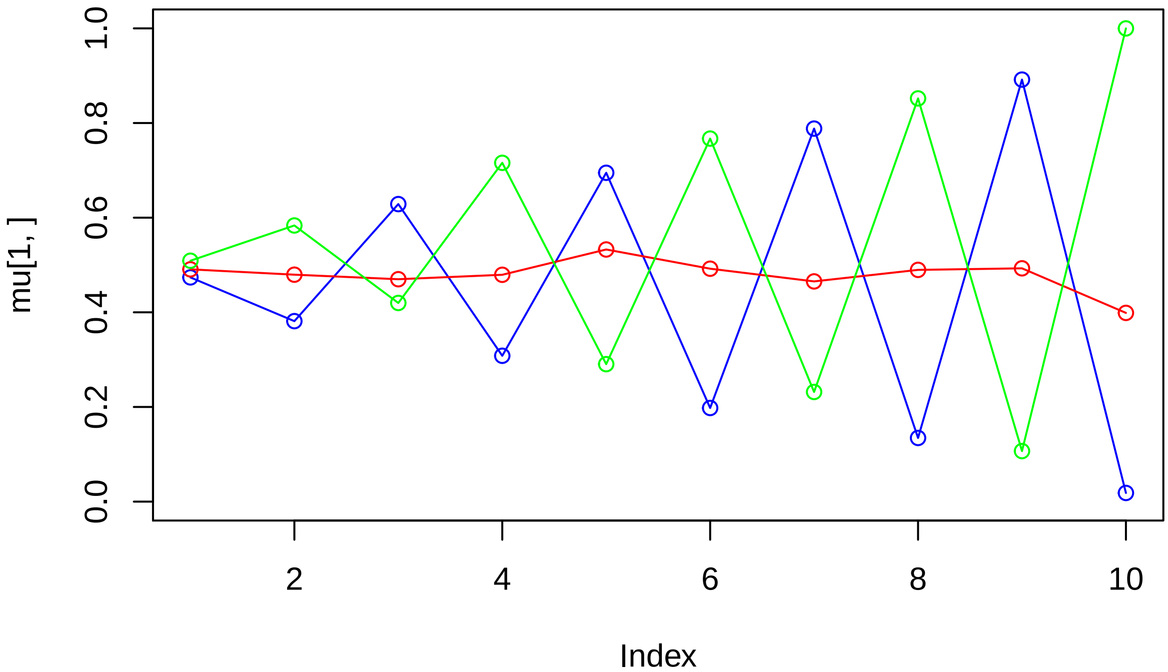

```
## iteration:  64 log likelihood:  -5315.487
```



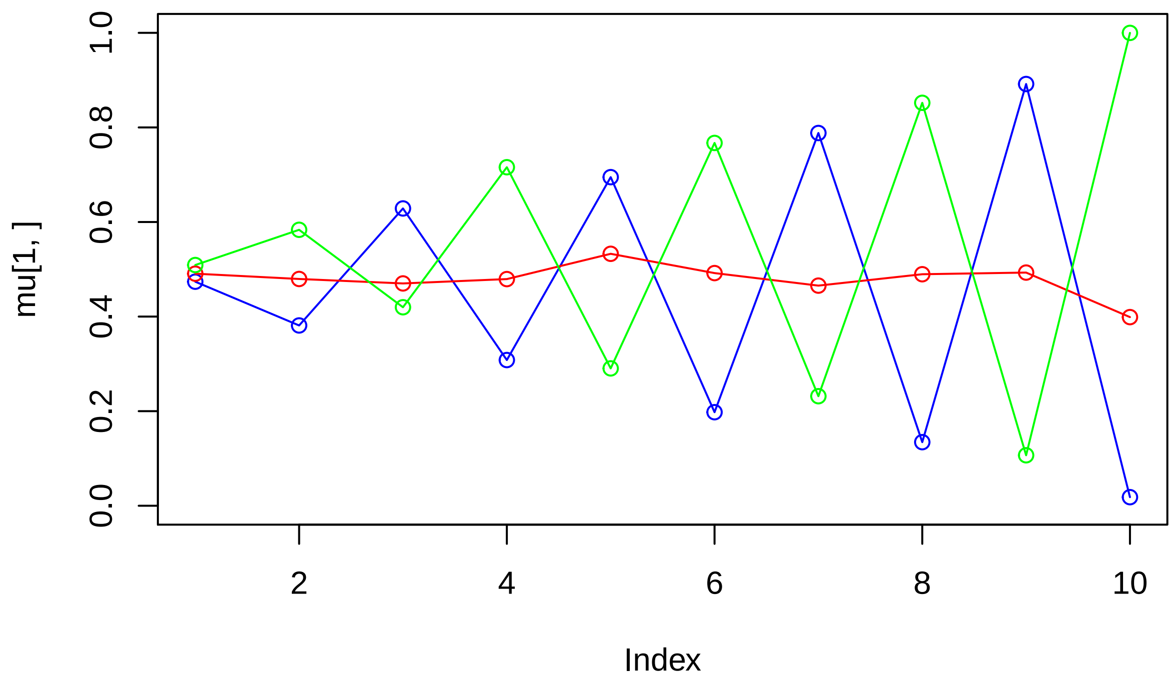

```
## iteration:  65 log likelihood:  -5315.364
```



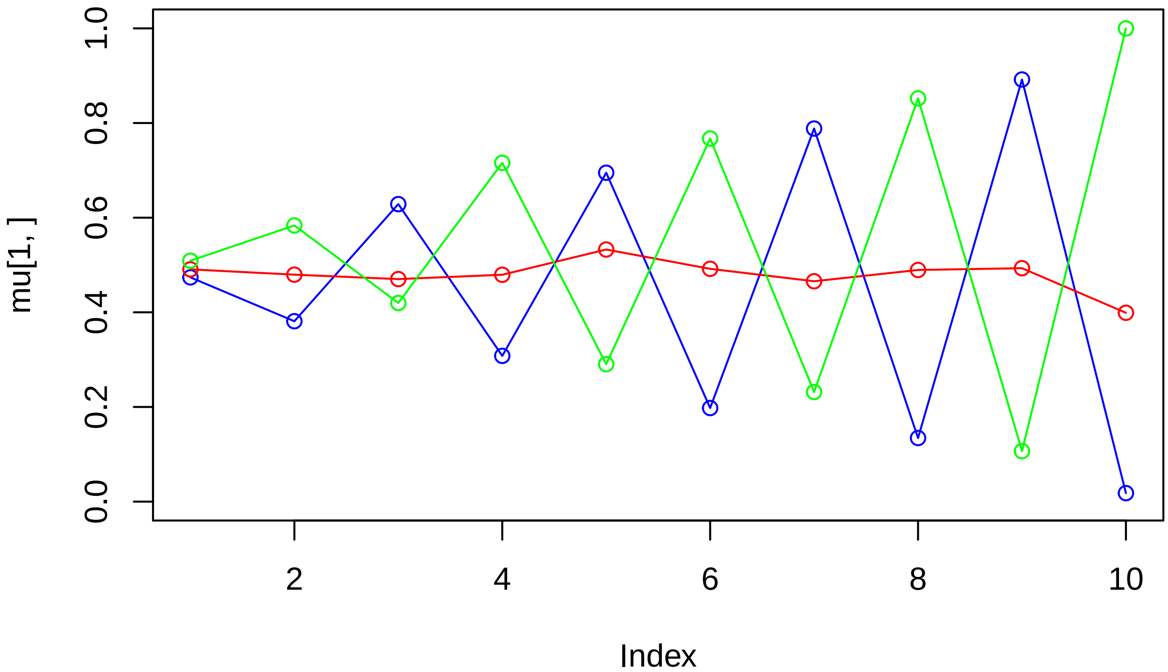

```
## iteration:  66 log likelihood:  -5315.246
```



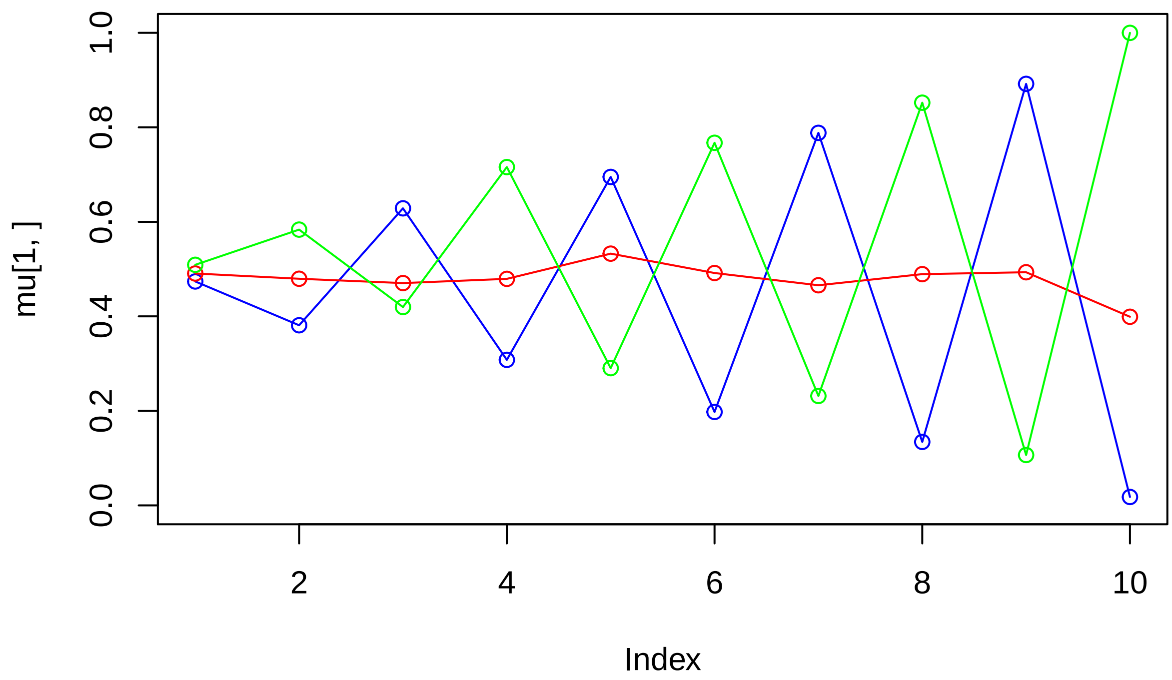

```
## iteration:  67 log likelihood:  -5315.132
```



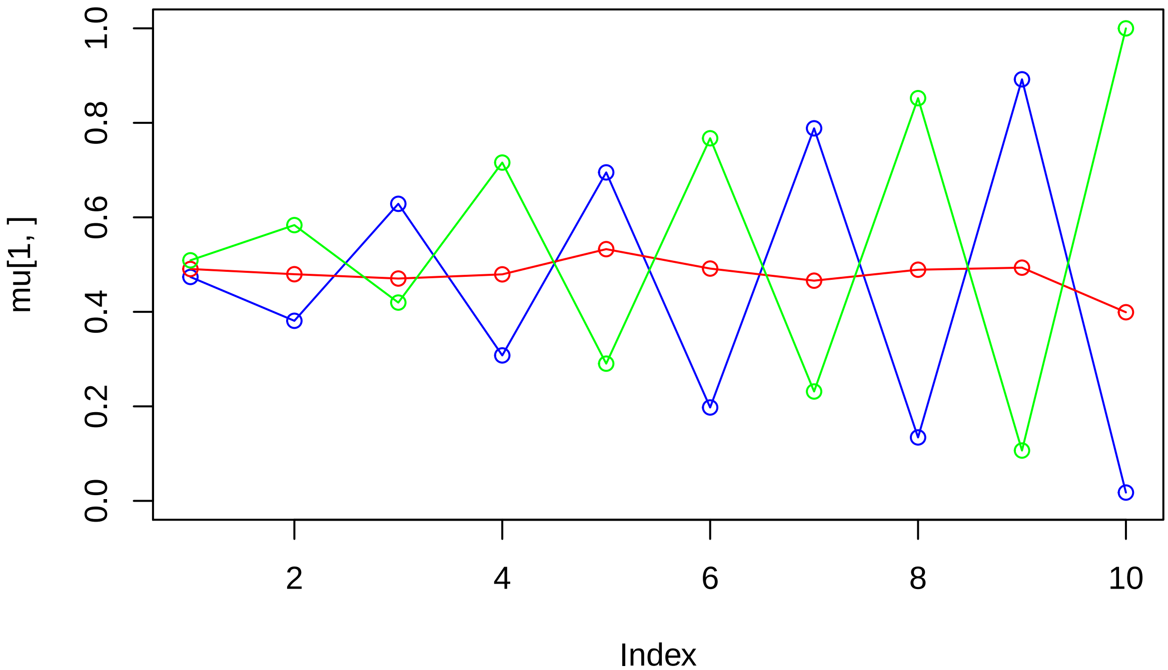

```
## iteration:  68 log likelihood:  -5315.022
```



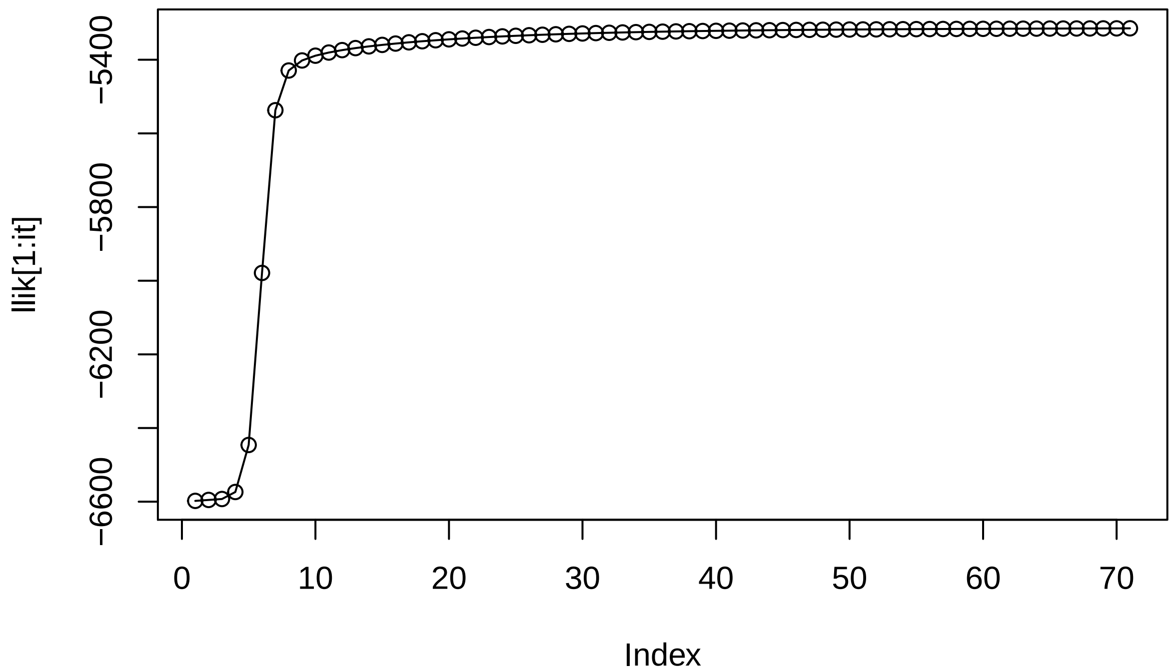

```
## iteration:  69 log likelihood:  -5314.916
```




```
## iteration:  70 log likelihood:  -5314.814
```



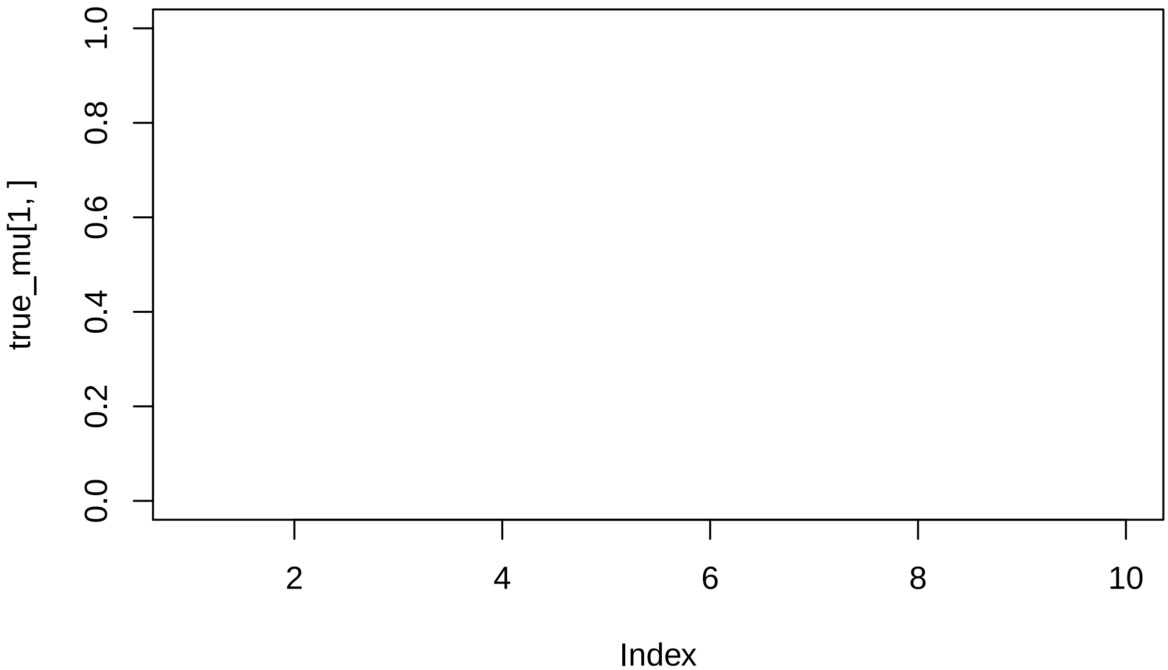

```
## iteration:  71 log likelihood:  -5314.715
```

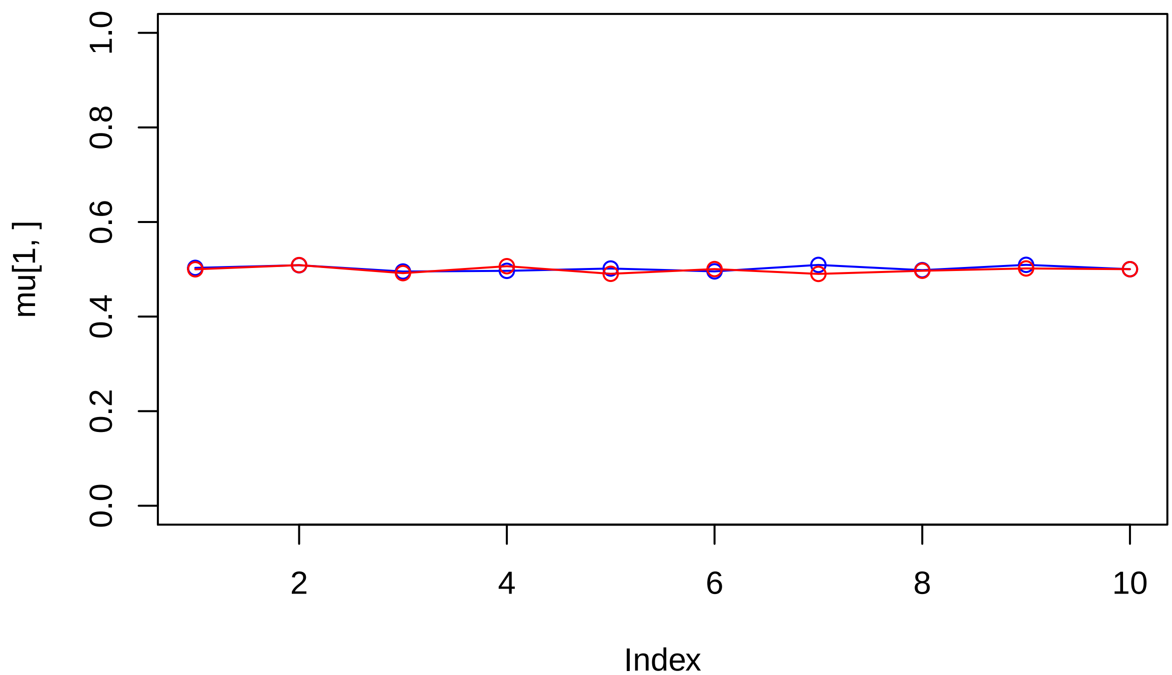



```
## [[1]]
##  [1] 0.32411556 0.30717327 0.36871116 0.47383554 0.49067297 0.50907301
##  [7] 0.38106095 0.47962547 0.58355730 0.62860603 0.47050854 0.41965465
## [13] 0.30784064 0.47936473 0.71594258 0.69512193 0.53269167 0.29039403
## [19] 0.19764738 0.49158731 0.76736739 0.78836950 0.46597001 0.23146477
## [25] 0.13424145 0.48919757 0.85221677 0.89207742 0.49348782 0.10652662
## [31] 0.01757154 0.39918885 0.99992807
```

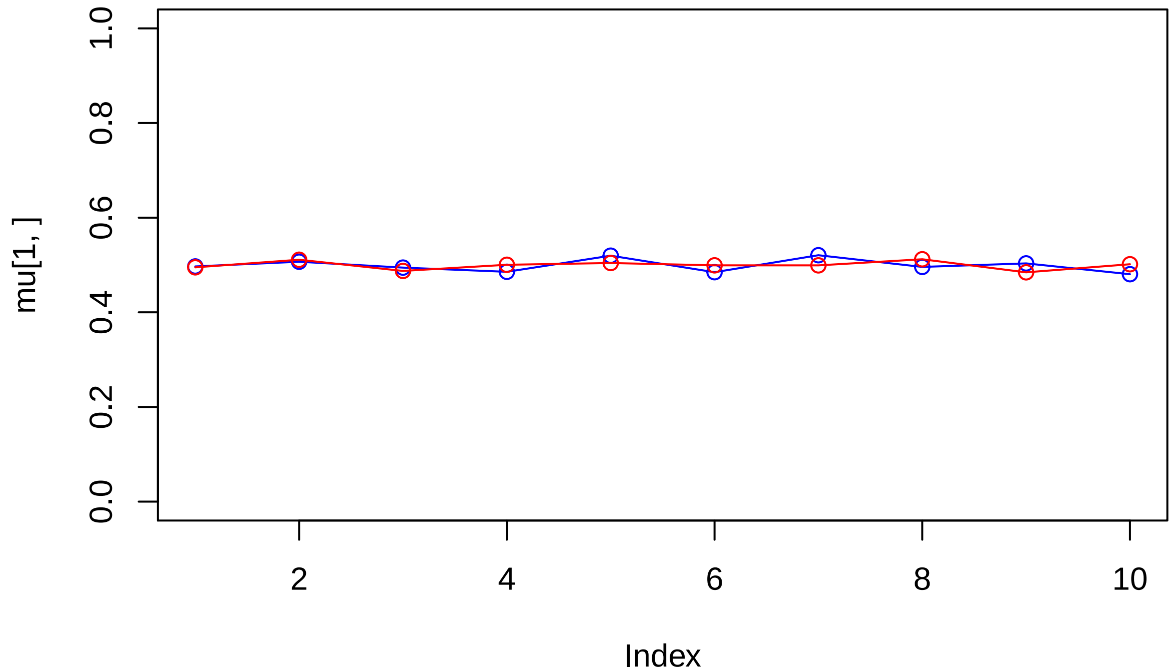
$\mathbf{K = 2}$

```r
myem(K=2)
```

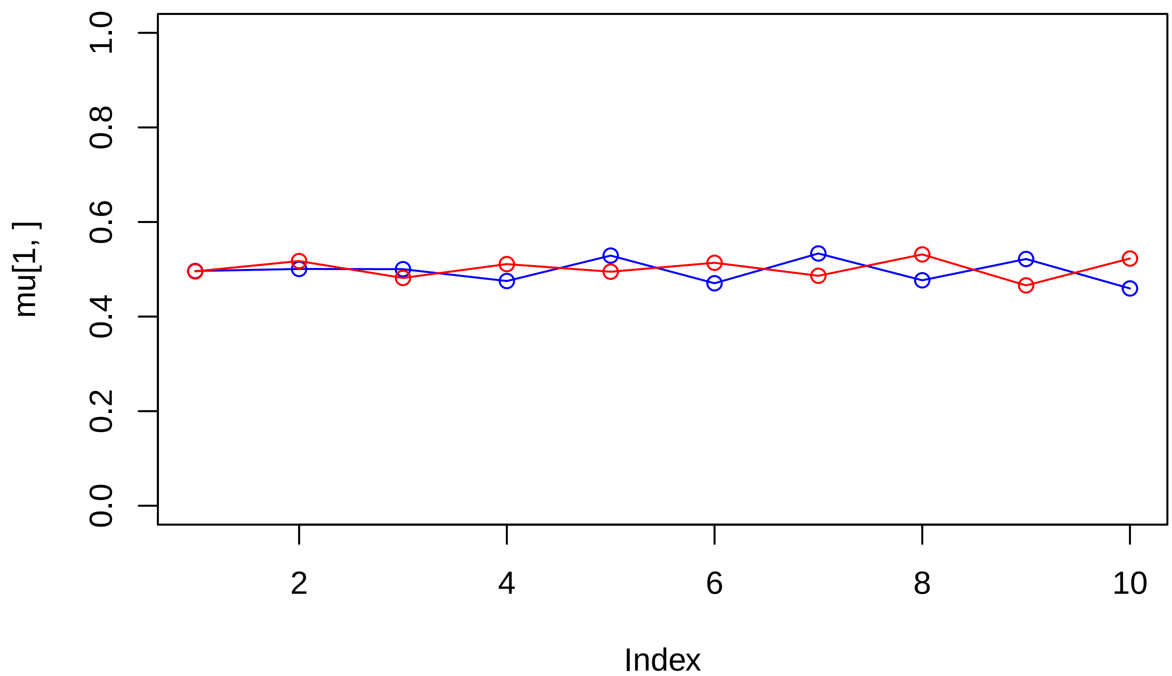

true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



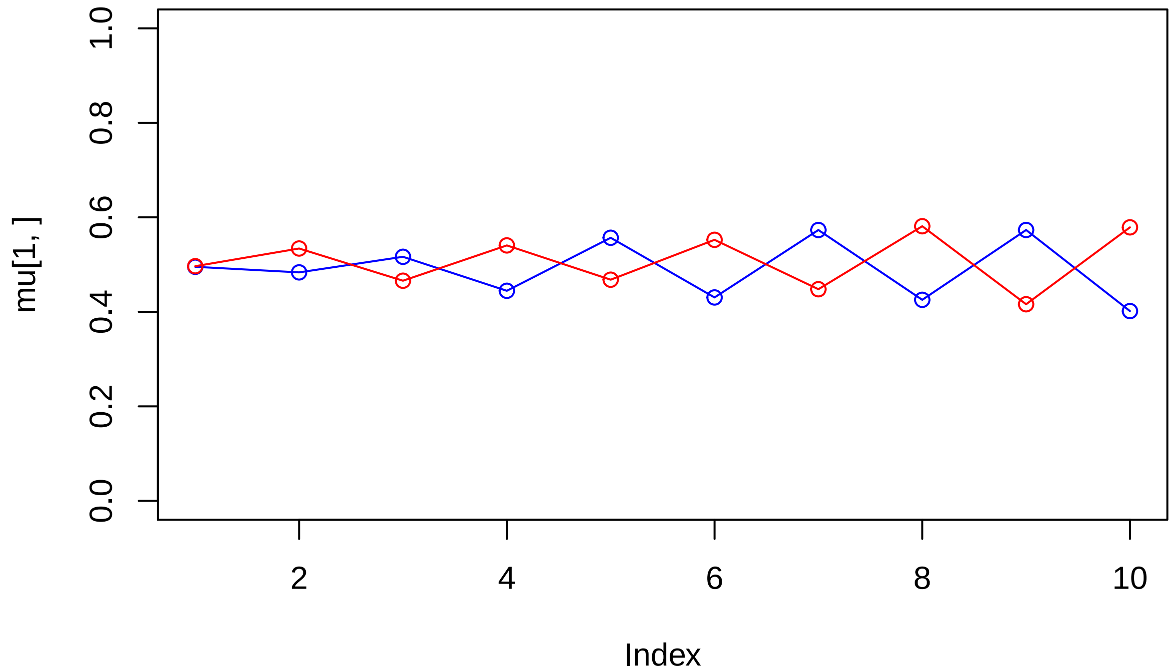

```
## iteration:  1 log likelihood:  -954.7133
```



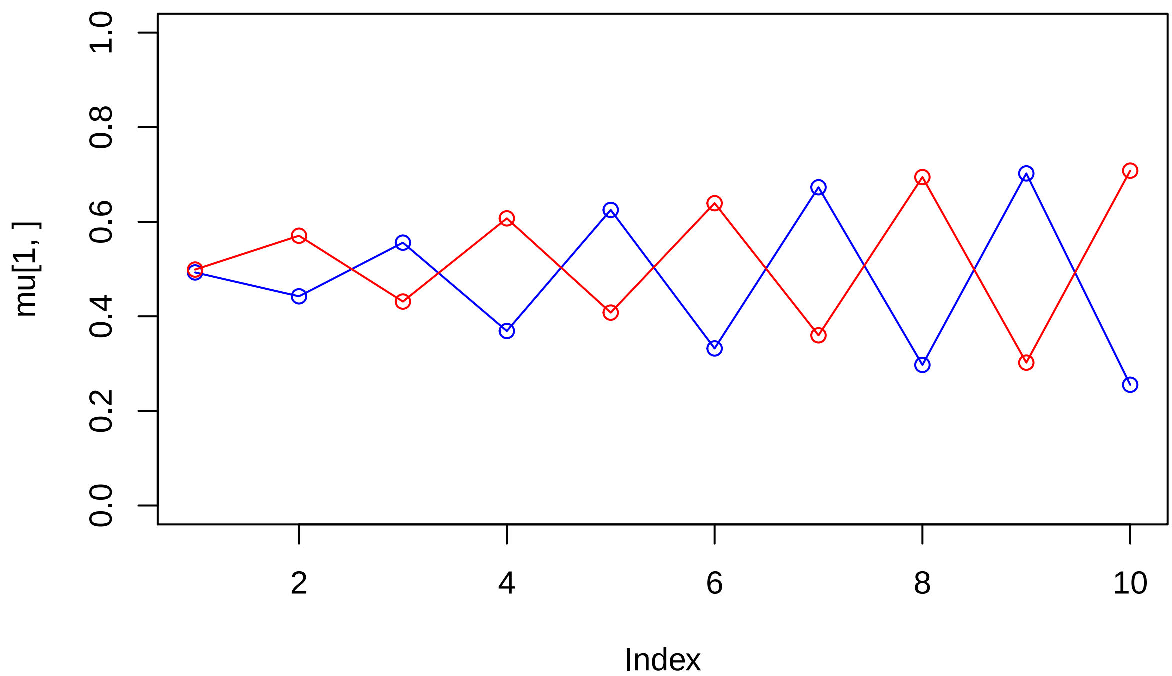

```
## iteration:  2 log likelihood:  -957.1002
```



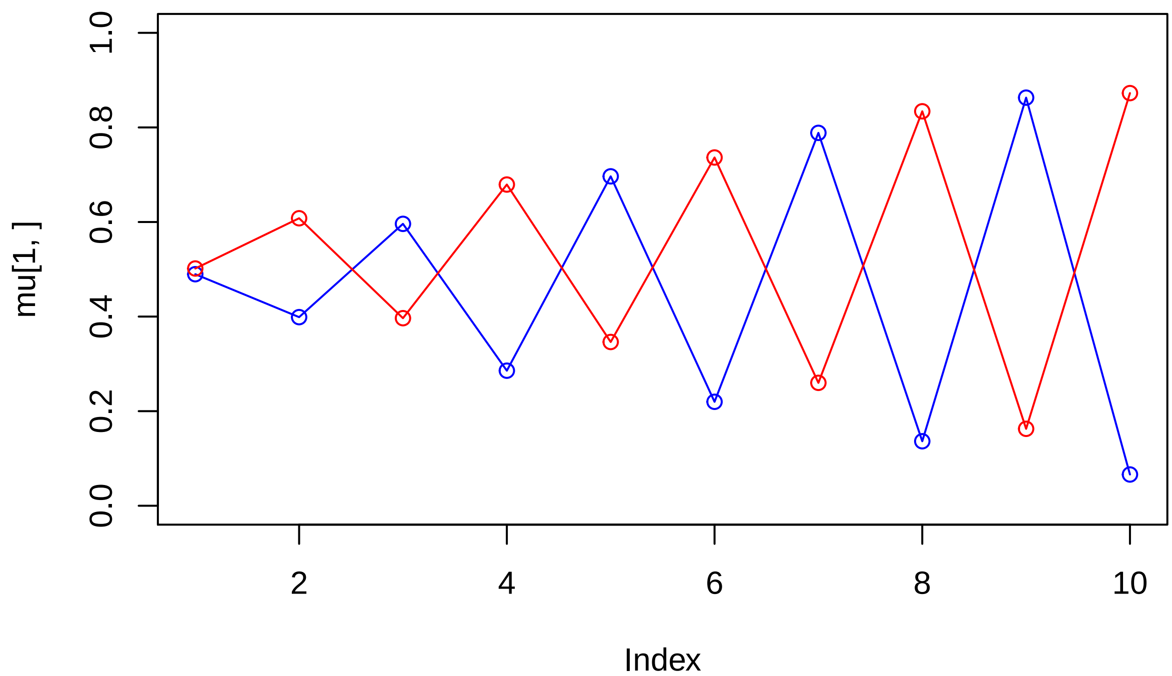

```
## iteration:  3 log likelihood:  -944.9229
```



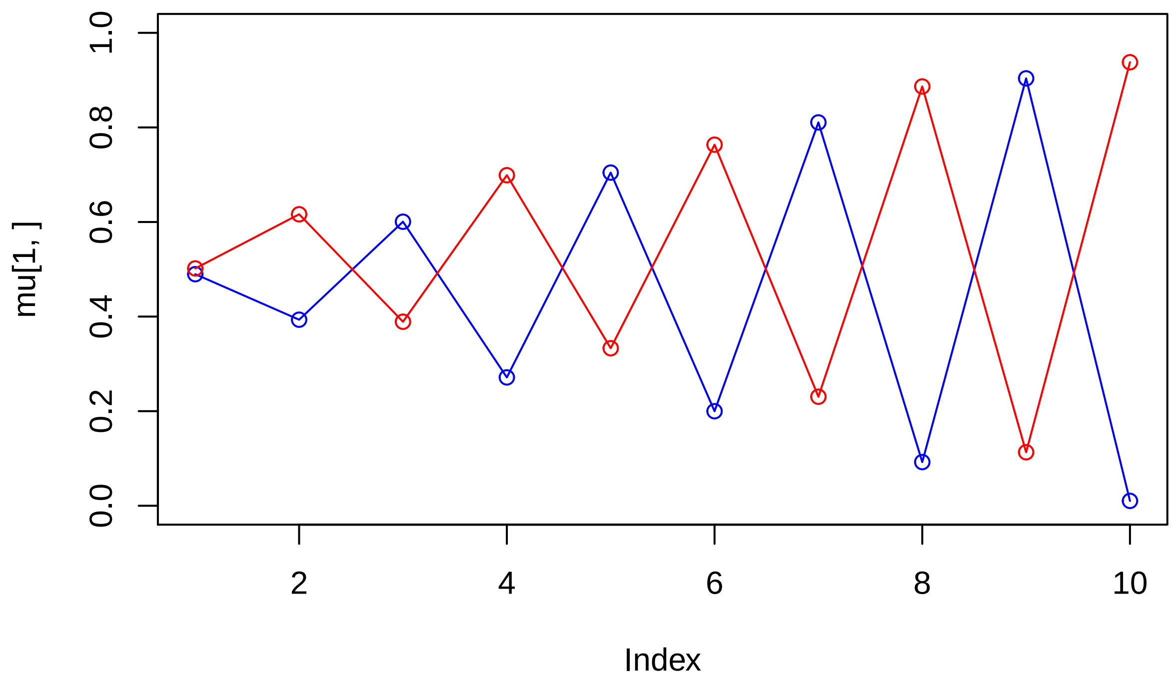

```
## iteration:  4 log likelihood:  -857.3443
```



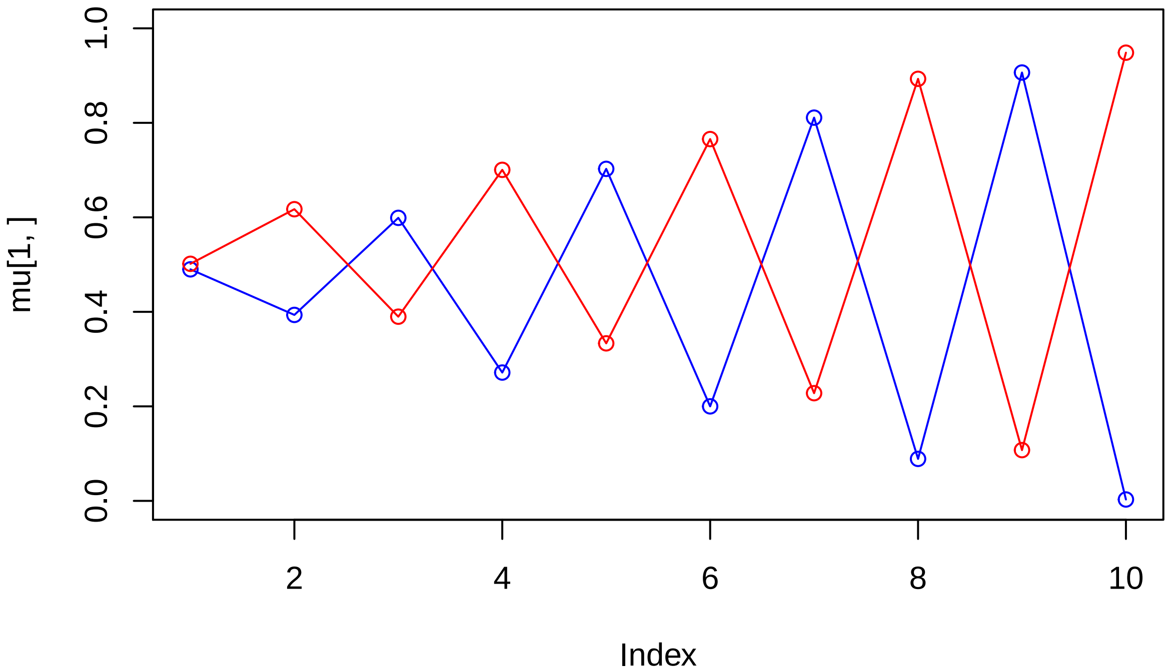

```
## iteration:  5 log likelihood:  -464.0063
```



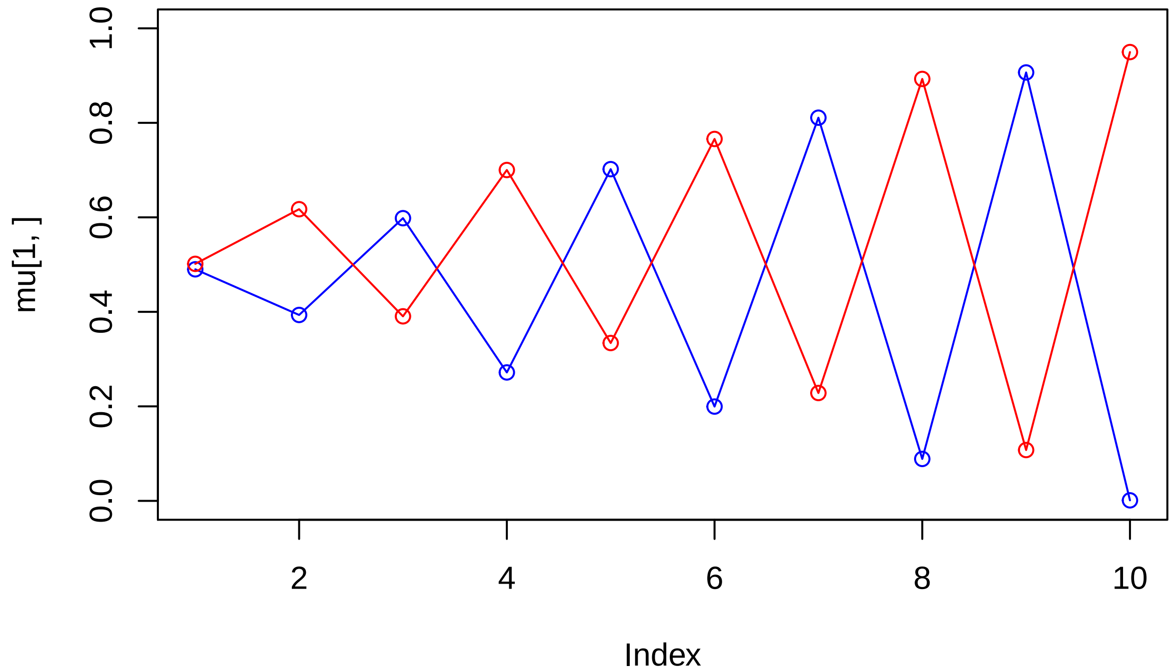

```
## iteration:  6 log likelihood:  50.2616
```



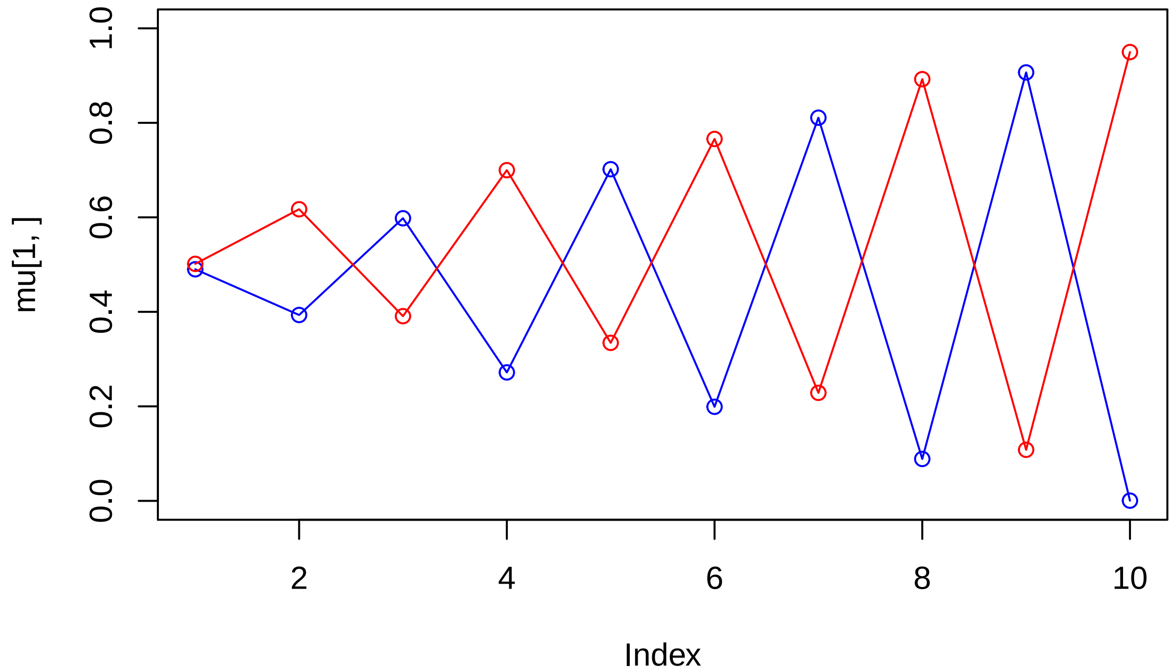

```
## iteration:  7 log likelihood:  177.0235
```



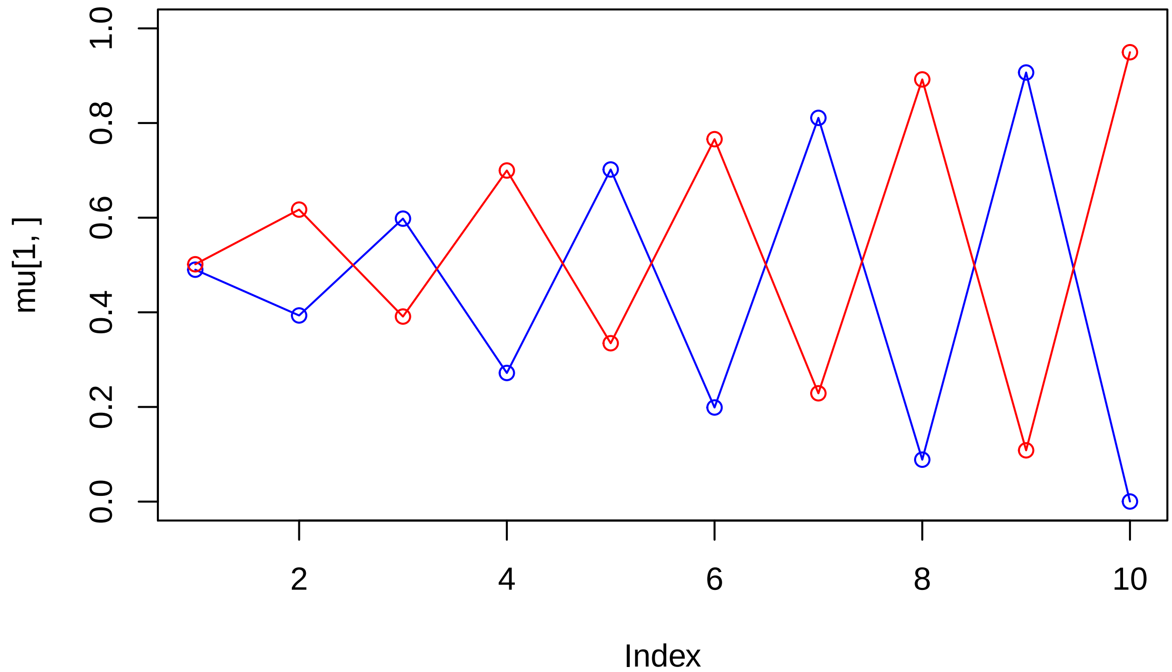

```
## iteration:  8 log likelihood:  189.1059
```



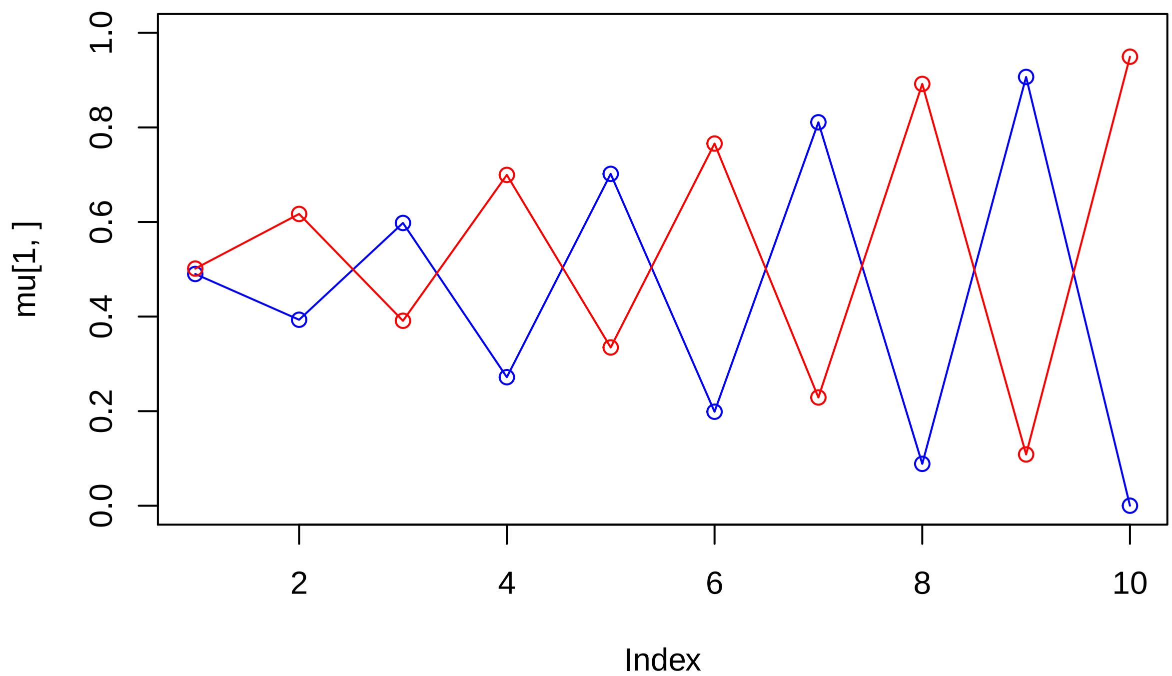

```
## iteration:  9 log likelihood:  190.0362
```



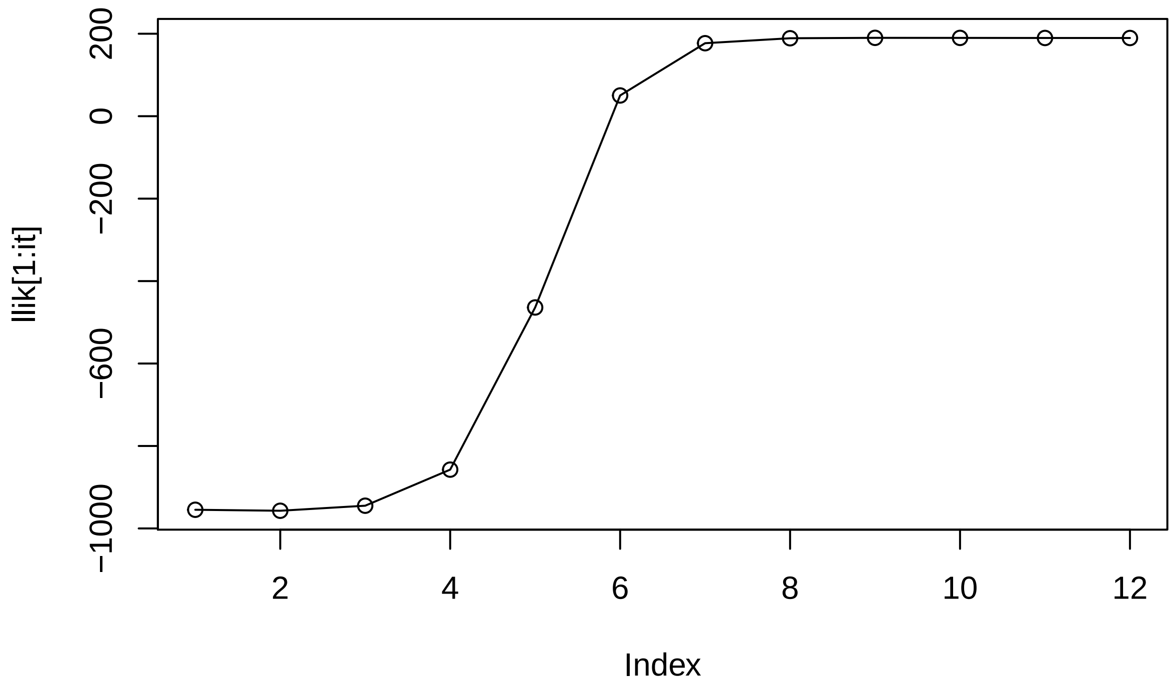

```
## iteration:  10 log likelihood:  189.9033
```




```
## iteration:  11 log likelihood:  189.7476
```



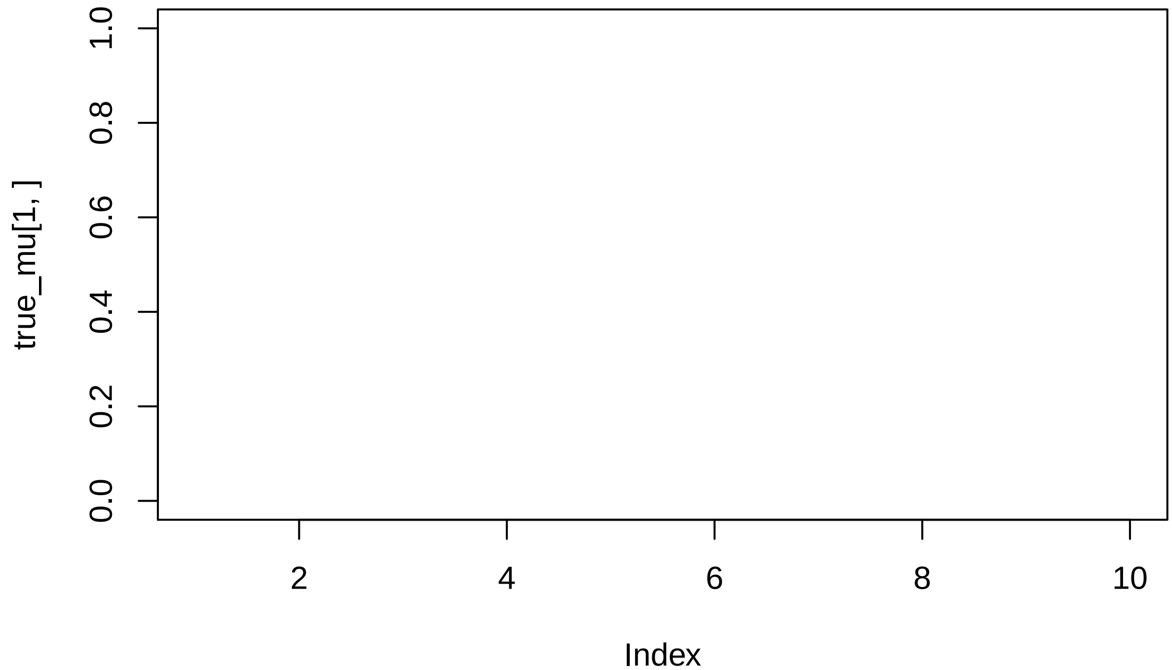

```
## iteration:  12 log likelihood:  189.6572
```

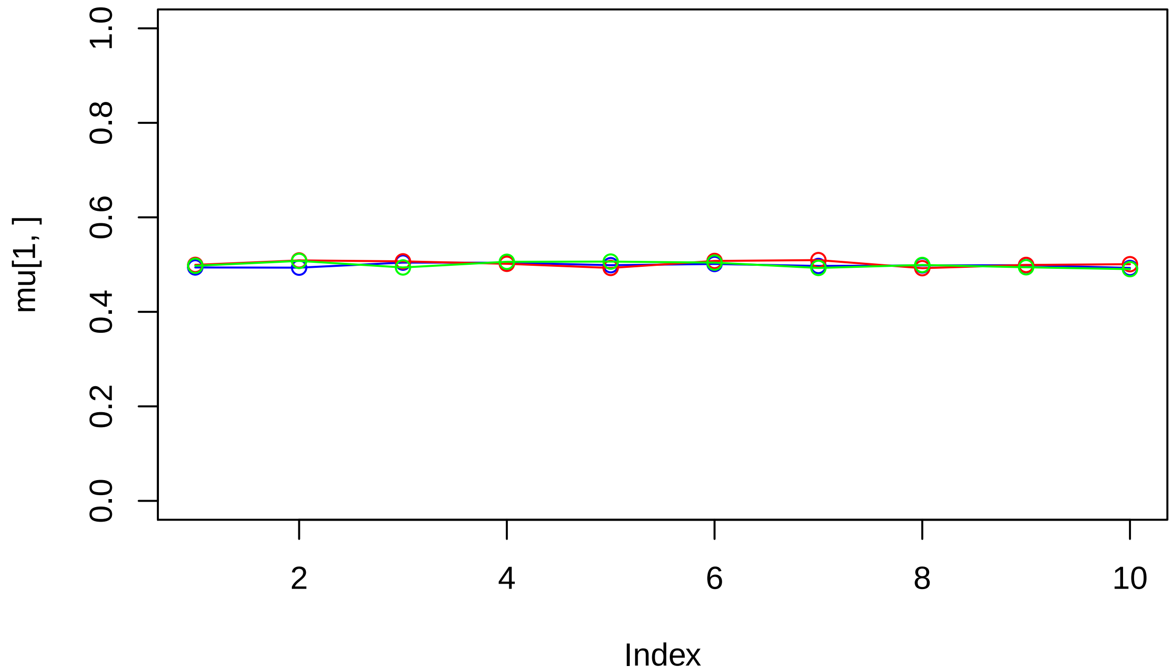



```
## [[1]]
##  [1] 0.48293133107 0.51706866893 0.49007966733 0.50152946699 0.39332820172
##  [6] 0.61703504228 0.59792428535 0.39113495199 0.27182539774 0.69957245640
## [11] 0.70178609664 0.33474375189 0.19877624614 0.76586485831 0.81088543925
## [16] 0.22897928765 0.08857255800 0.89200054923 0.90675481733 0.10849562636
## [21] 0.00008952955 0.94950012035
```

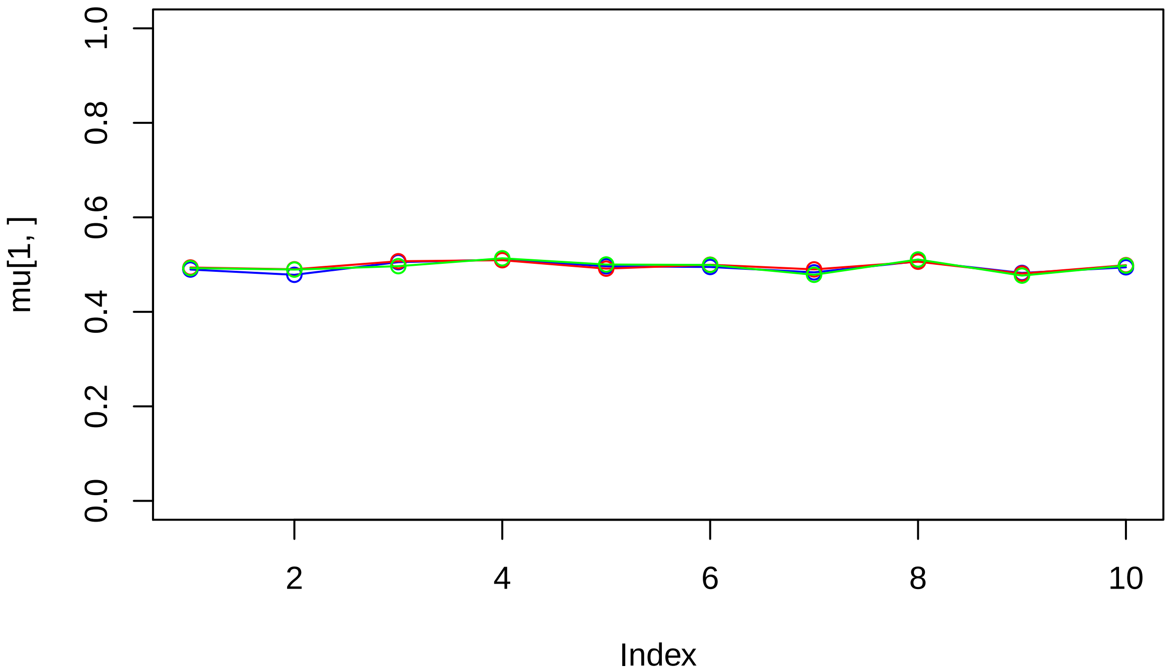
K = 3

```
myem(K=3)
```

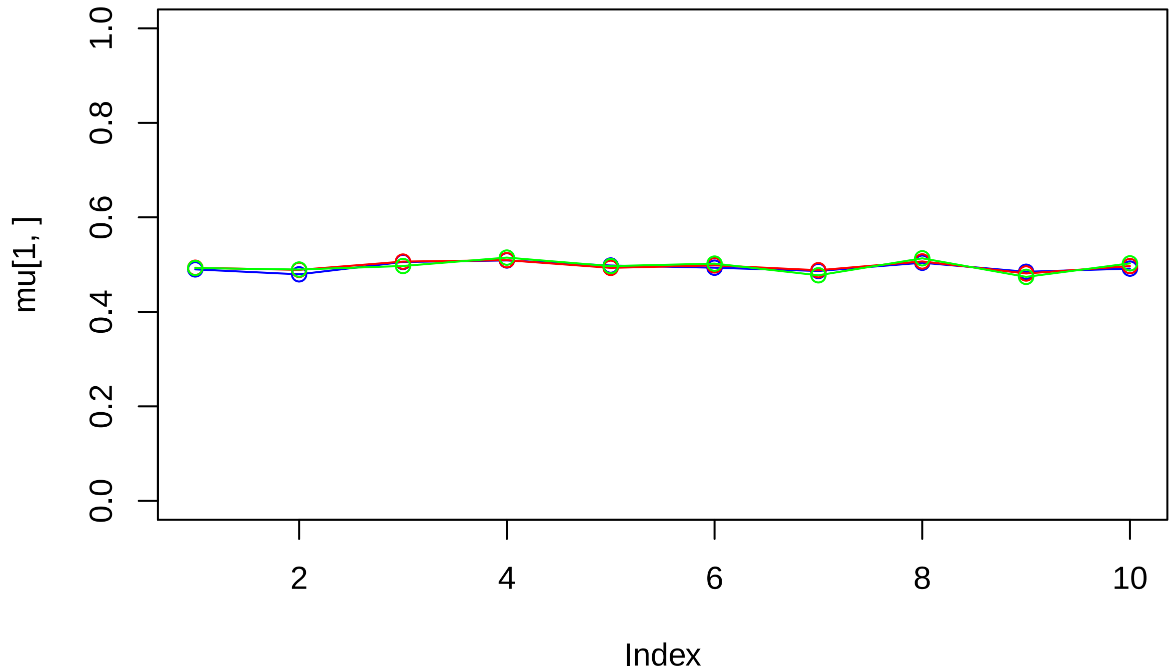

true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



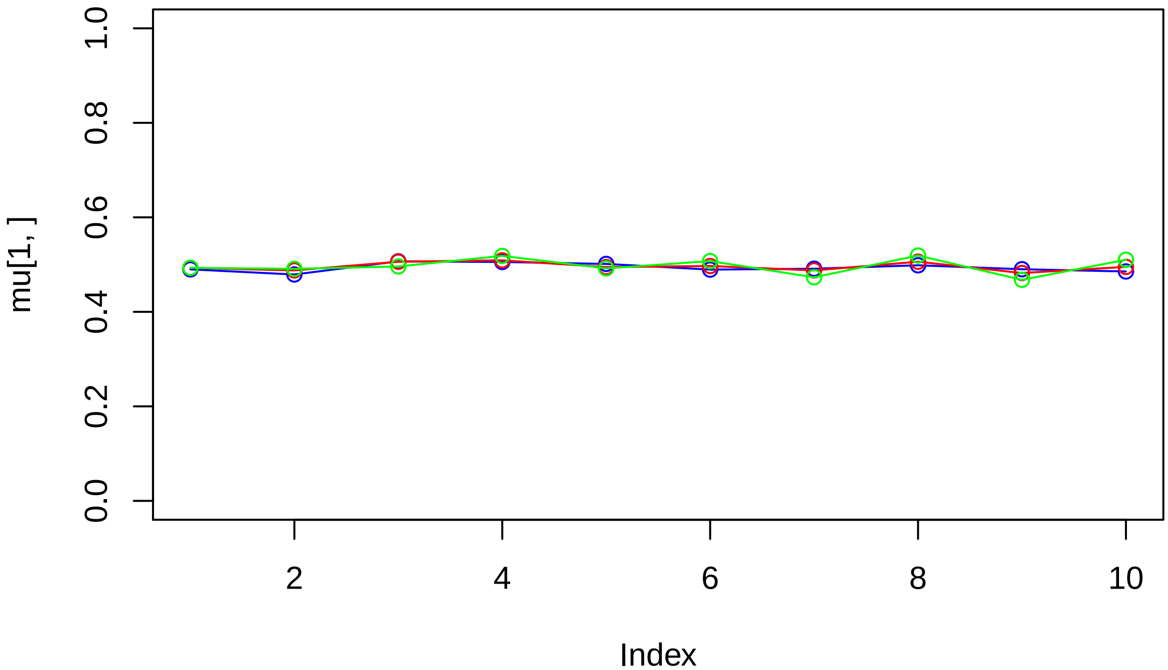

```
## iteration:  1 log likelihood:  -912.7567
```



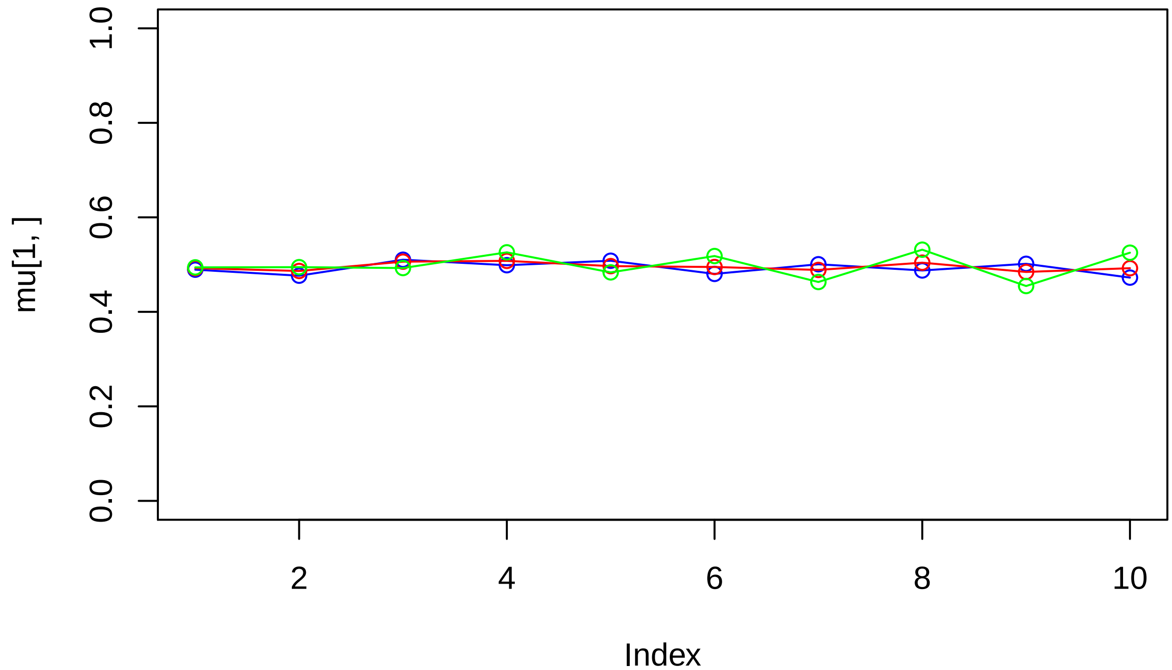

```
## iteration:  2 log likelihood:  -932.1921
```



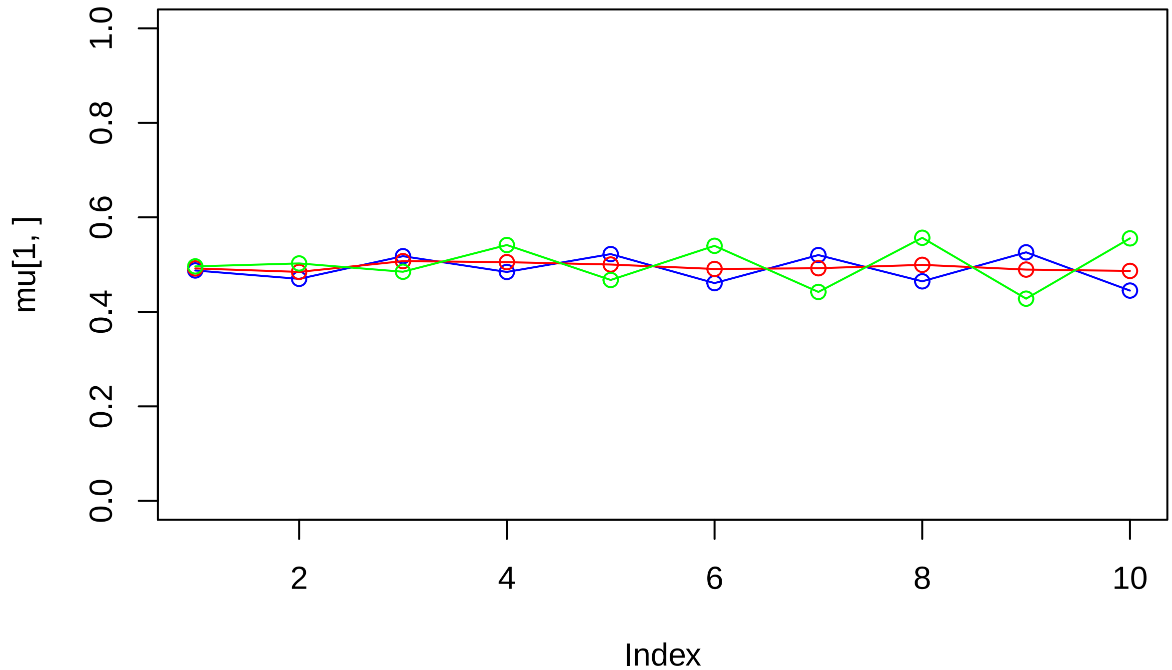

```
## iteration:  3 log likelihood:  -932.0234
```



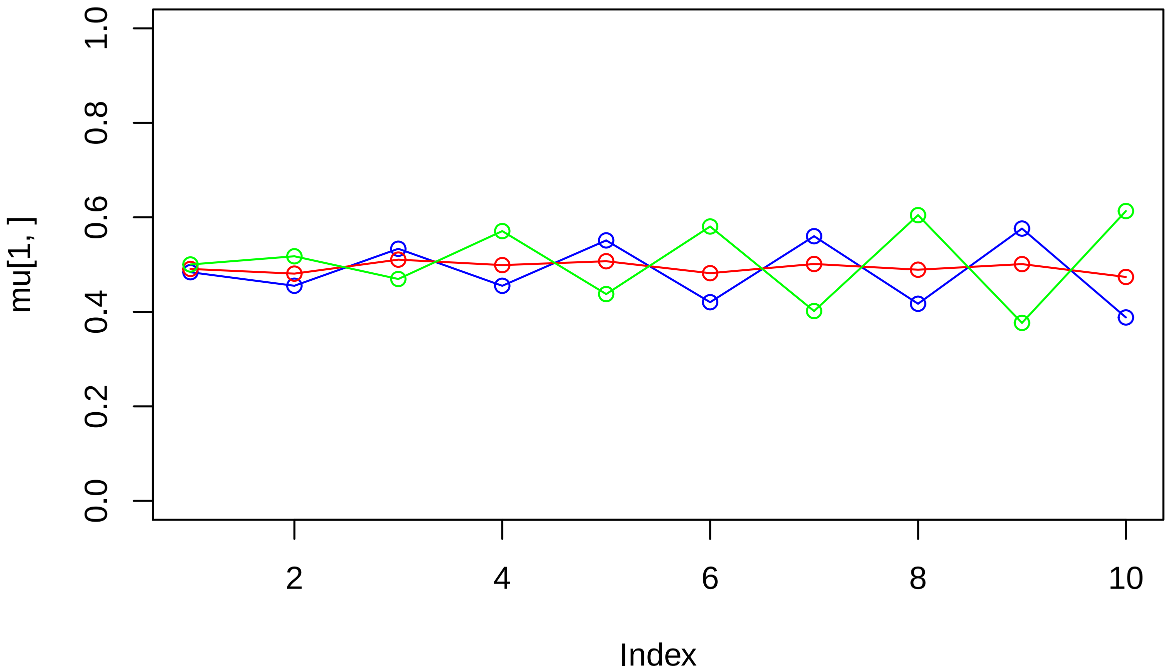

```
## iteration:  4 log likelihood:  -931.2587
```



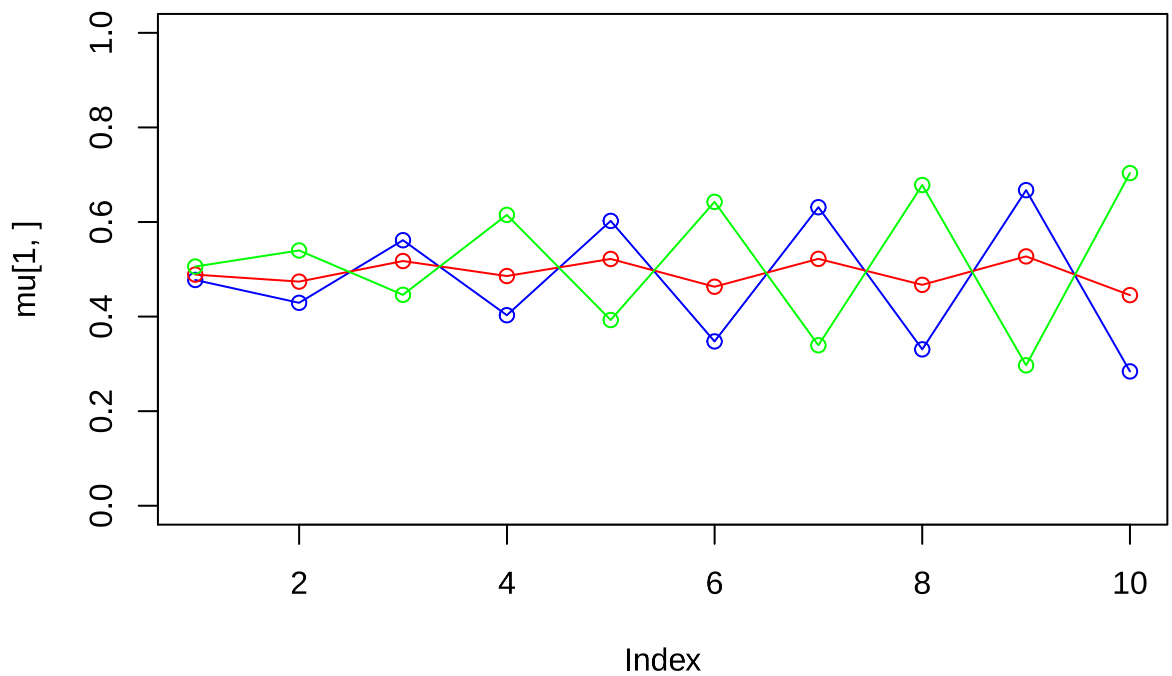

```
## iteration:  5 log likelihood:  -927.8881
```



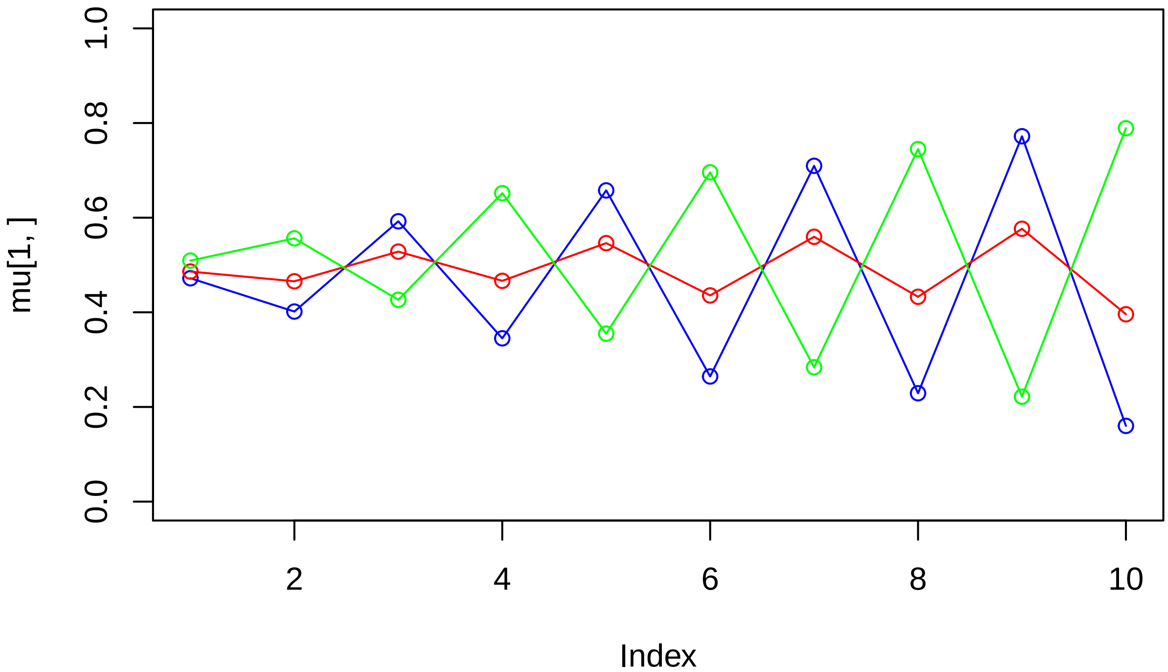

```
## iteration:  6 log likelihood:  -913.454
```



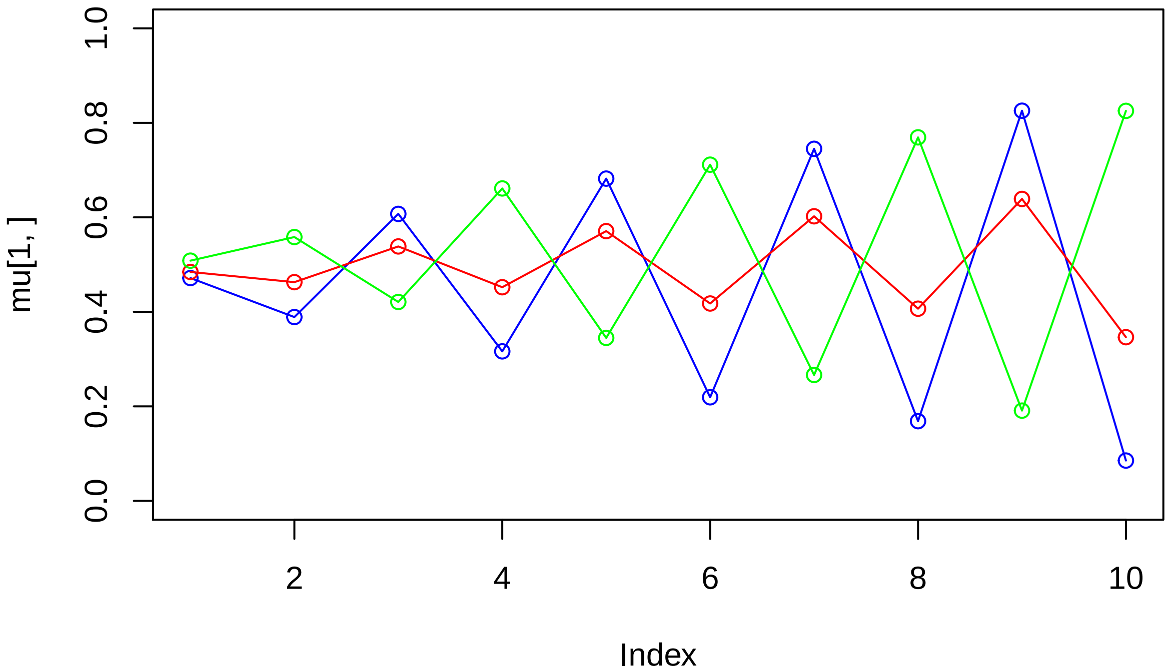

```
## iteration:  7 log likelihood:  -858.0583
```



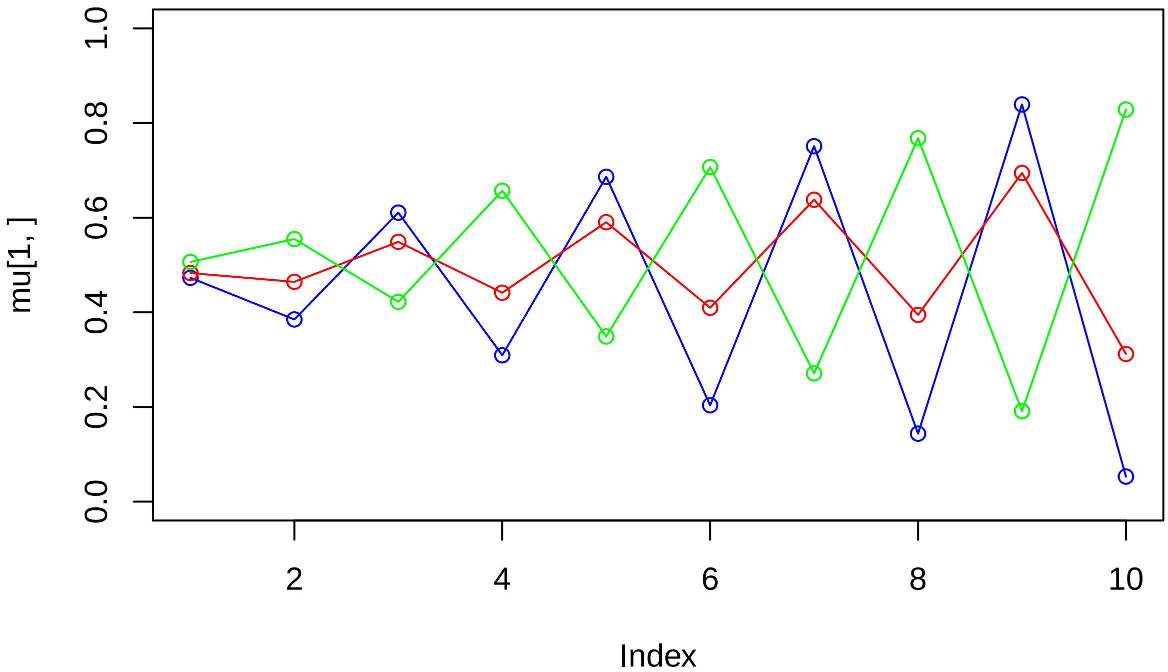

```
## iteration:  8 log likelihood:  -709.6665
```



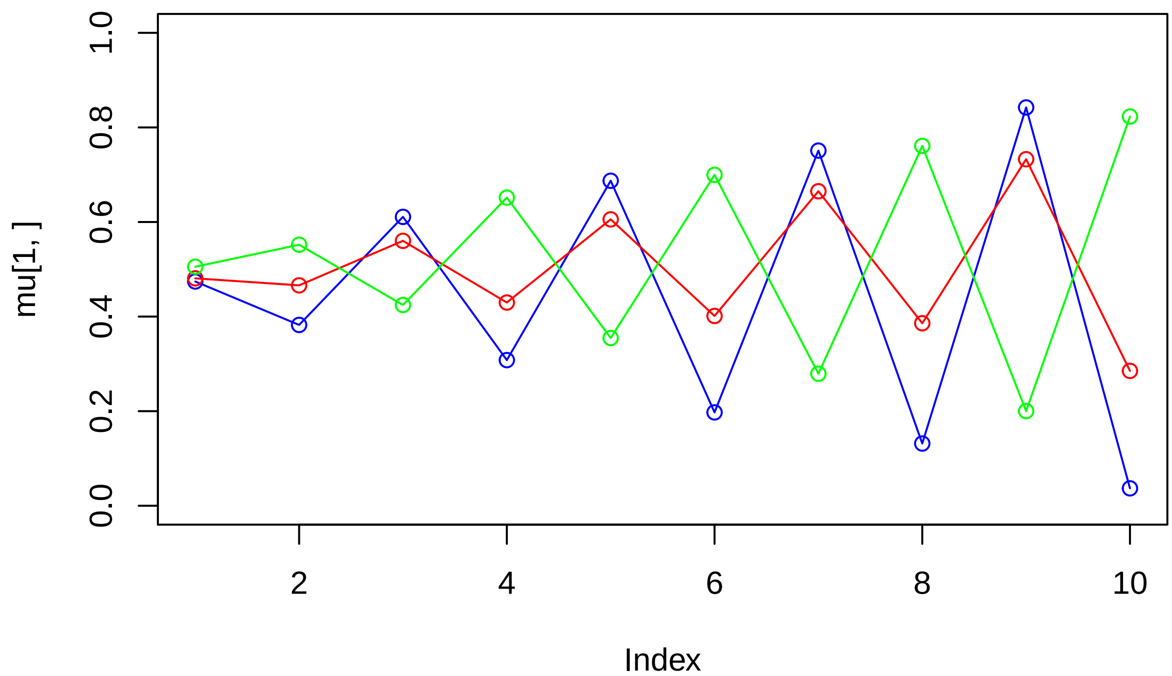

```
## iteration:  9 log likelihood:  -524.1097
```



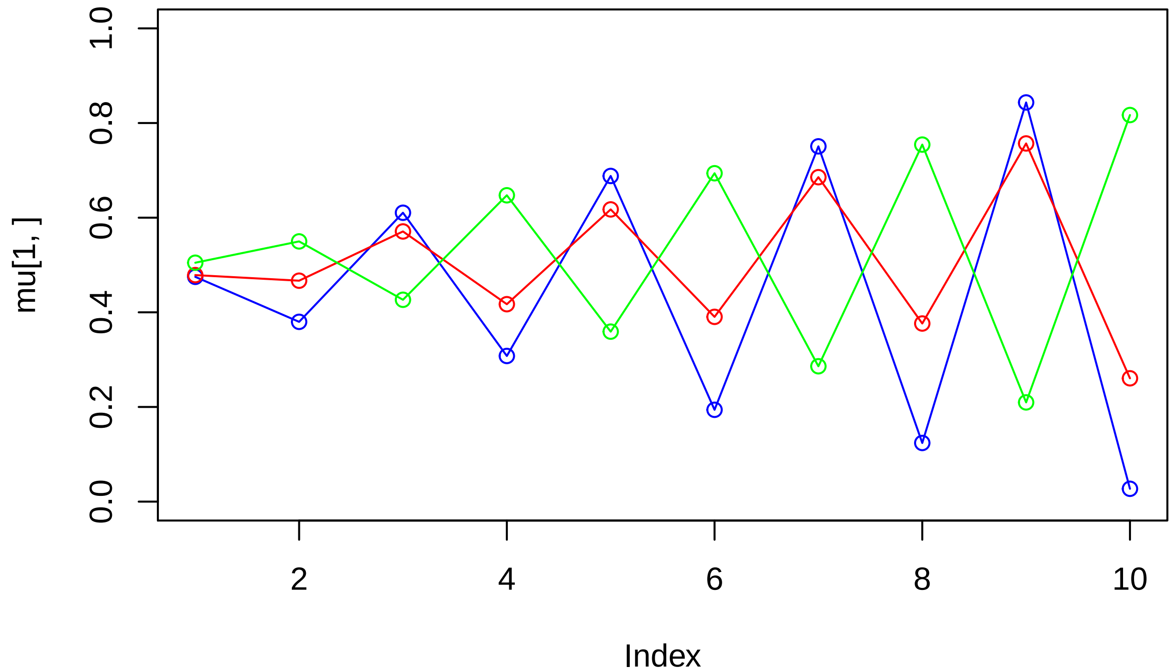

```
## iteration:  10 log likelihood:  -433.1614
```



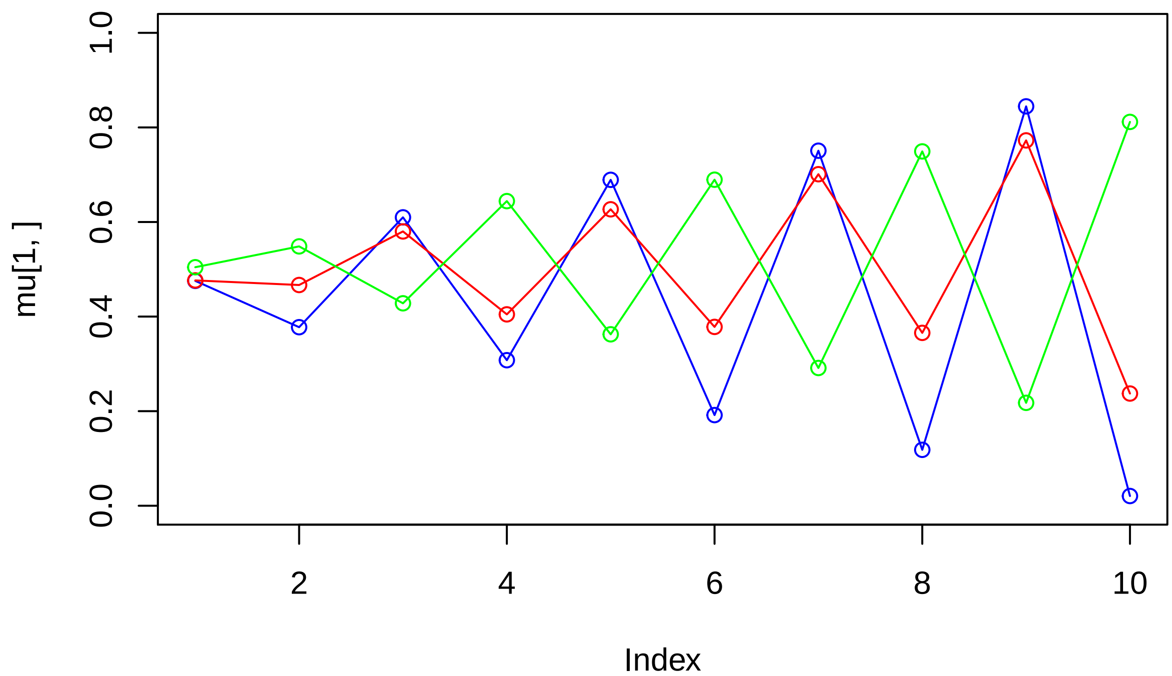

```
## iteration:  11 log likelihood:  -409.3331
```



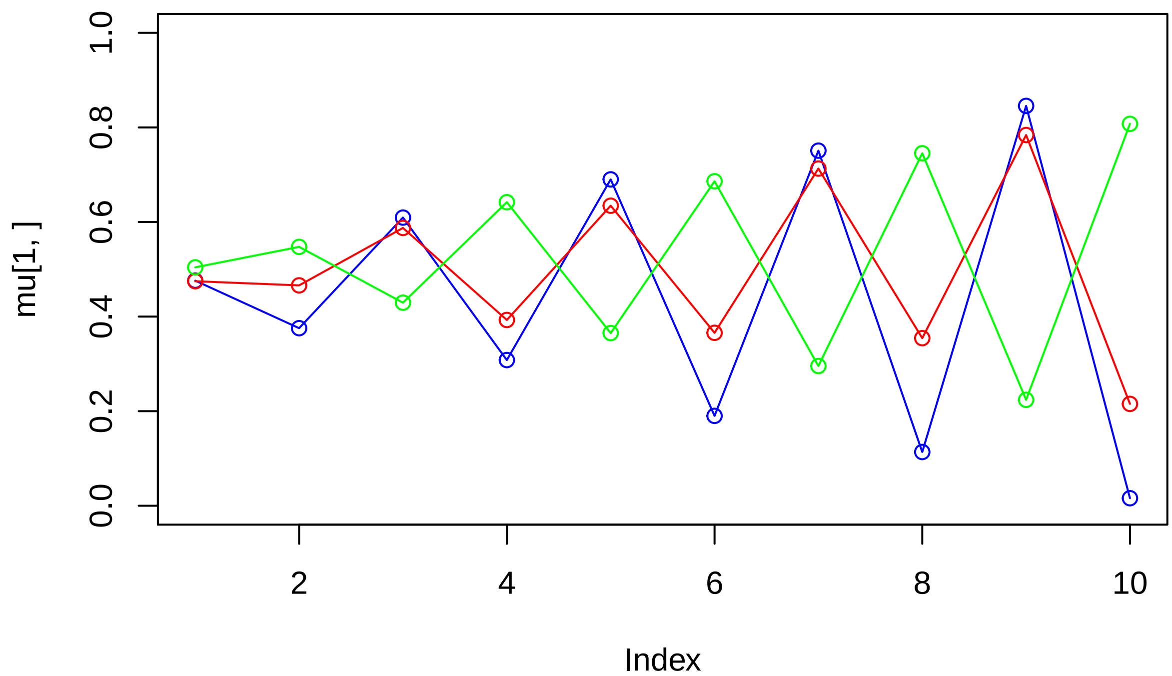

```
## iteration:  12 log likelihood:  -405.2132
```



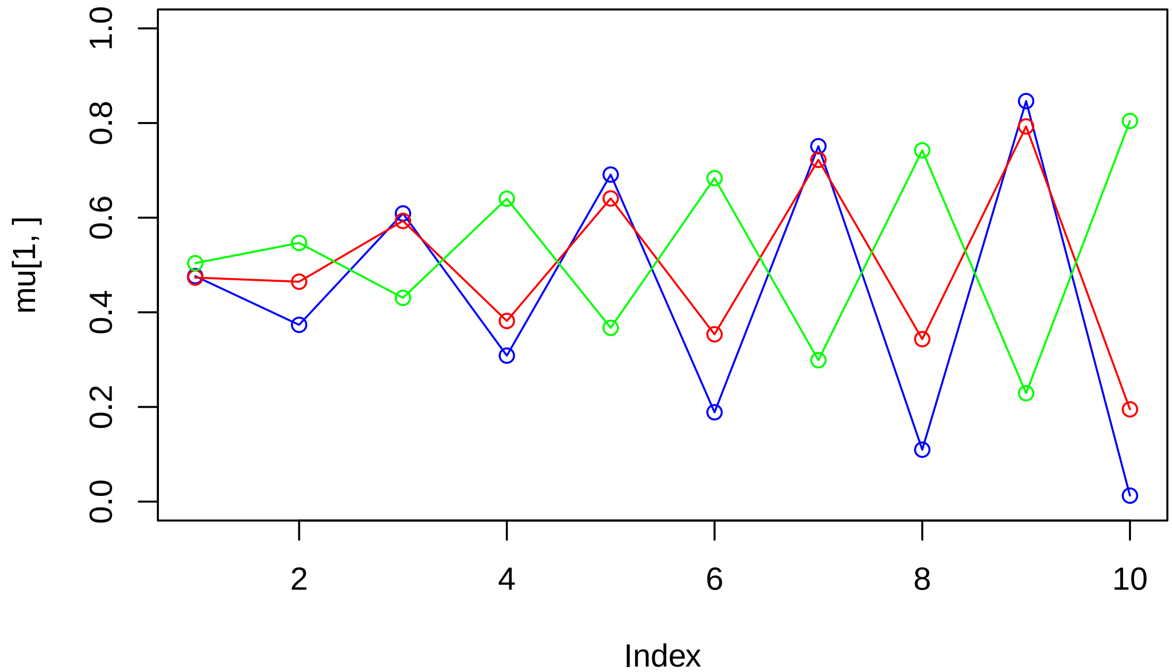

```
## iteration:  13 log likelihood:  -405.7233
```



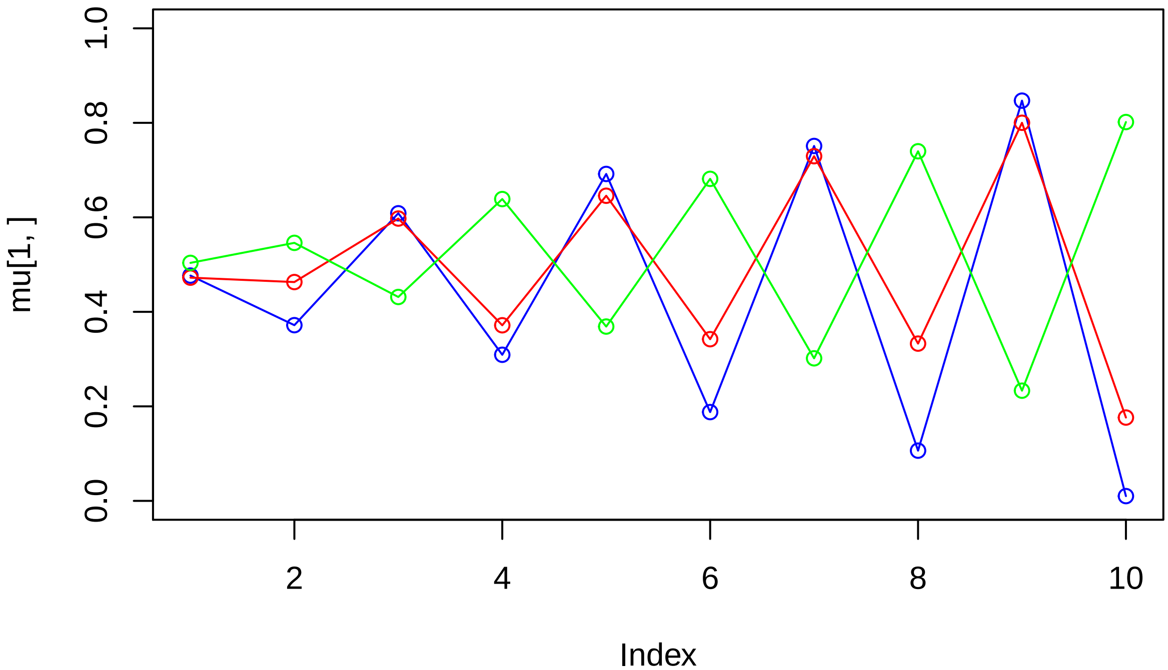

```
## iteration:  14 log likelihood:  -407.1621
```



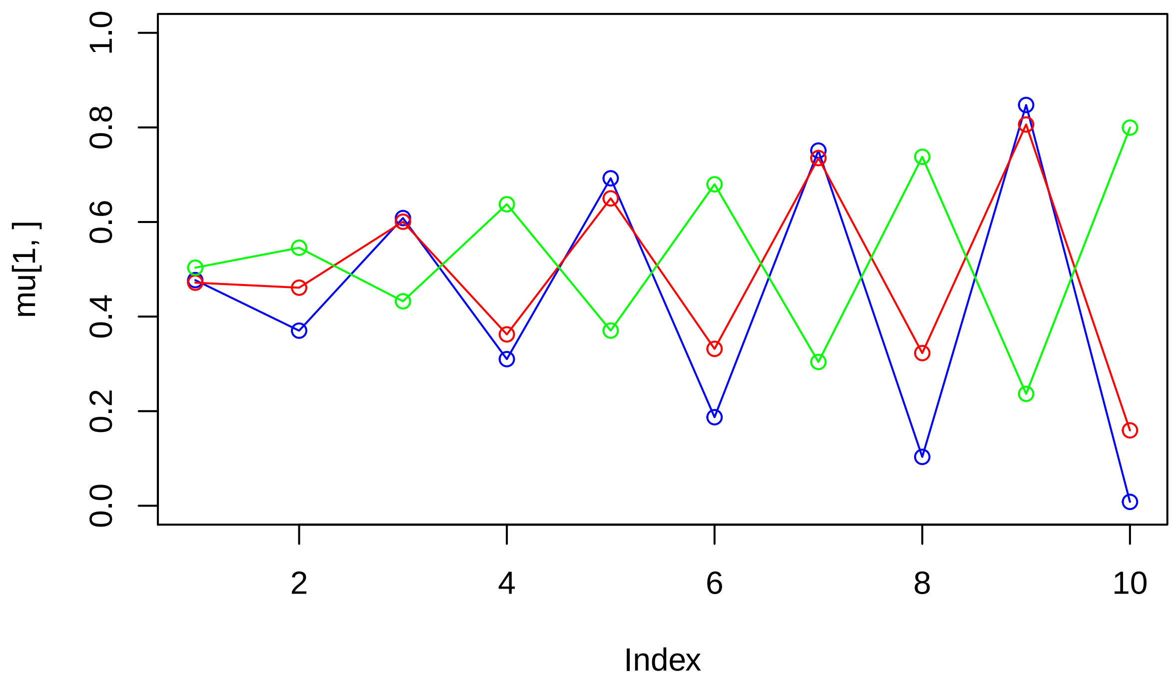

```
## iteration:  15 log likelihood:  -408.6475
```



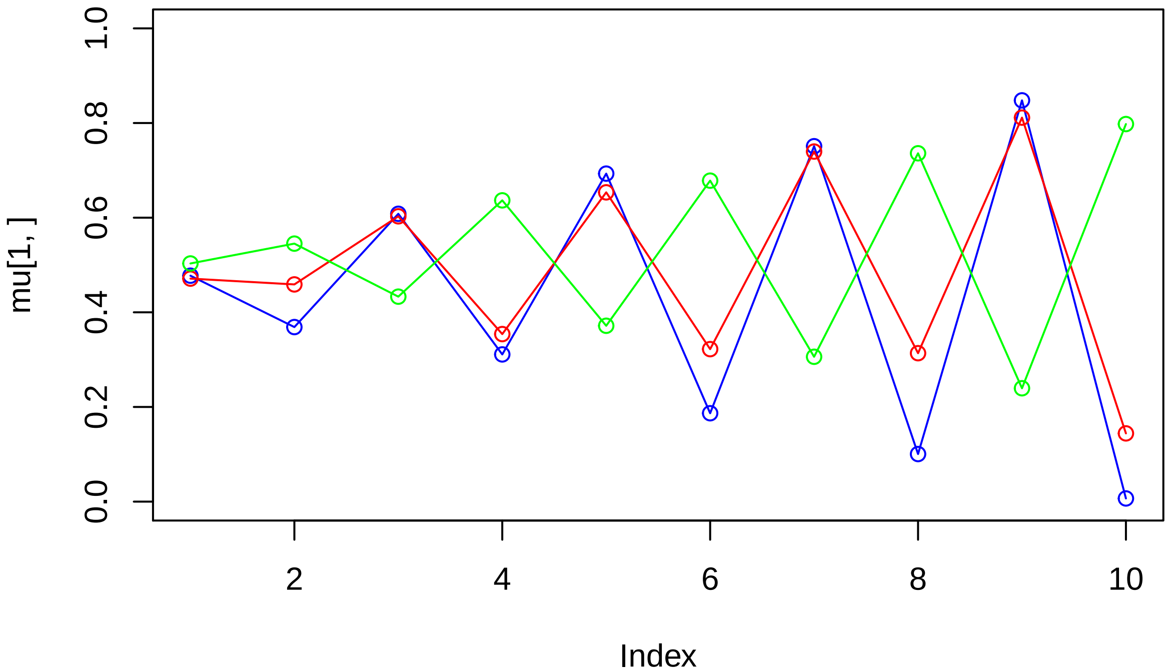

```
## iteration:  16 log likelihood:  -409.9879
```



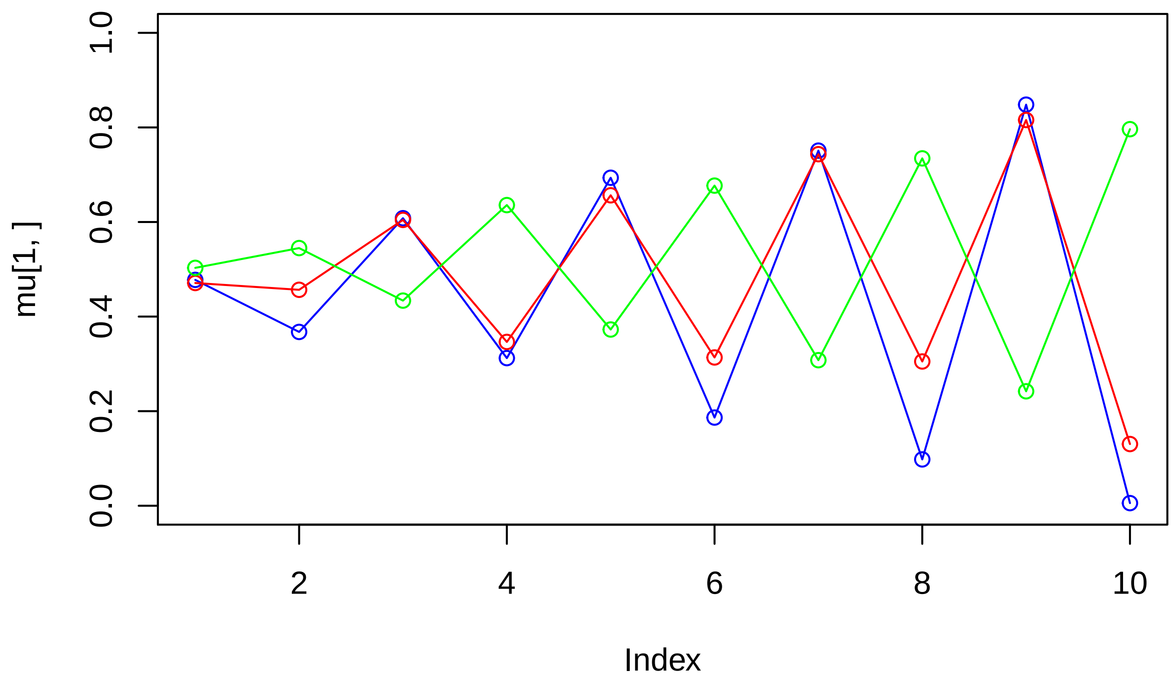

```
## iteration:  17 log likelihood:  -411.1645
```



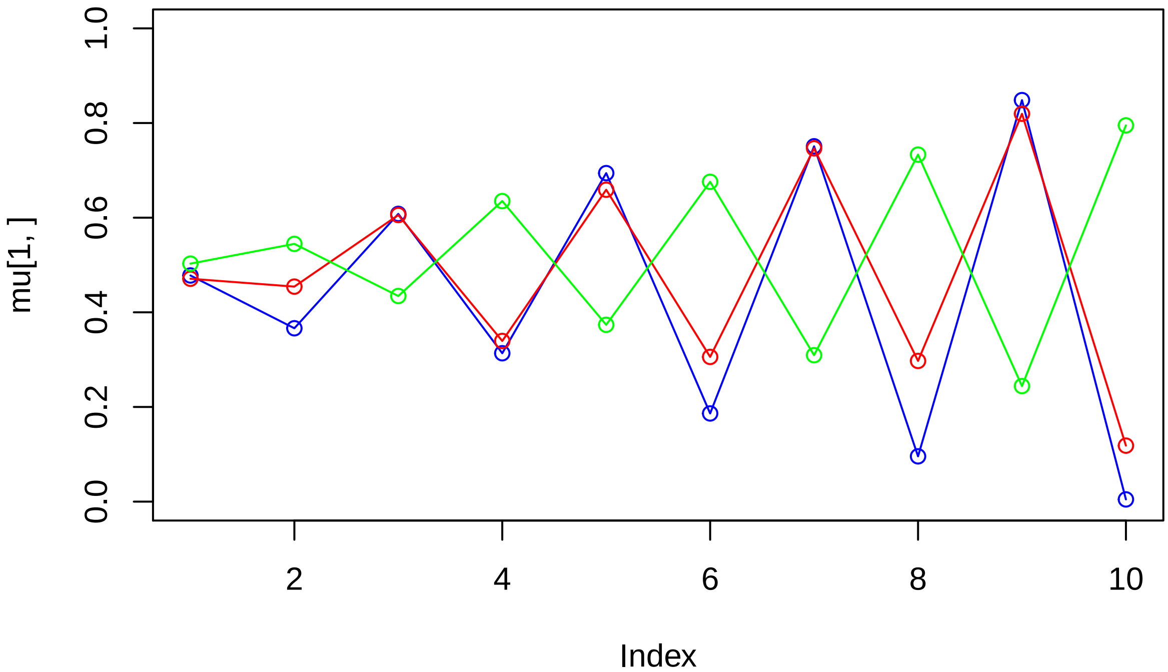

```
## iteration:  18 log likelihood:  -412.1979
```



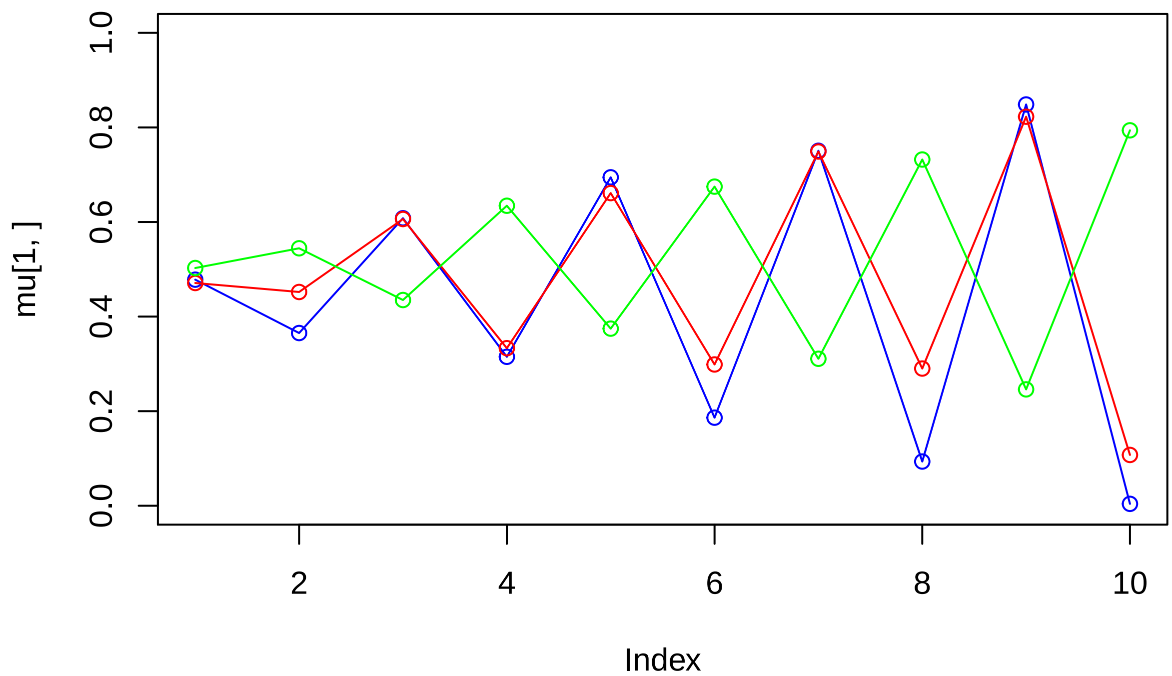

```
## iteration:  19 log likelihood:  -413.1139
```



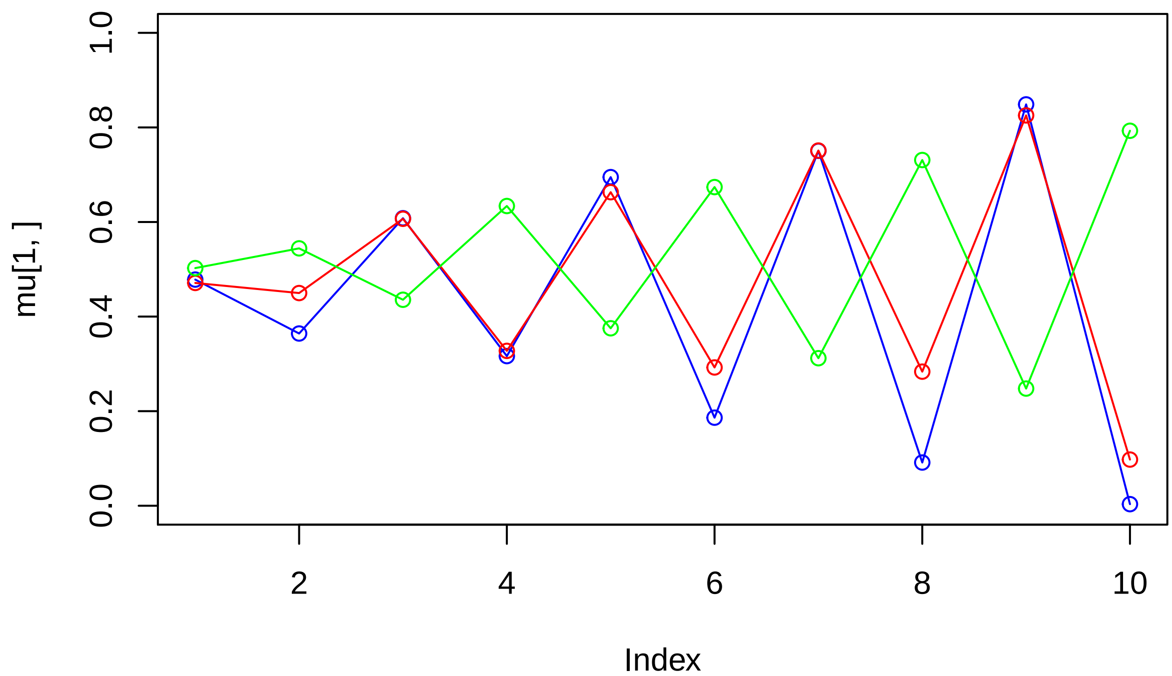

```
## iteration:  20 log likelihood:  -413.934
```



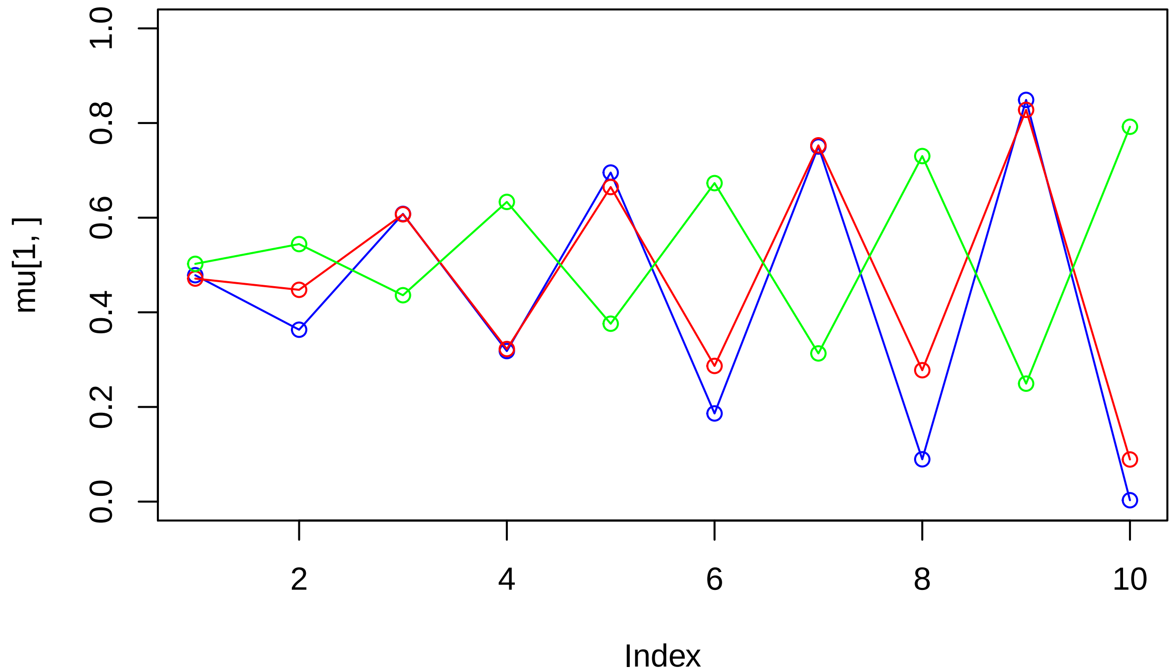

```
## iteration:  21 log likelihood:  -414.675
```



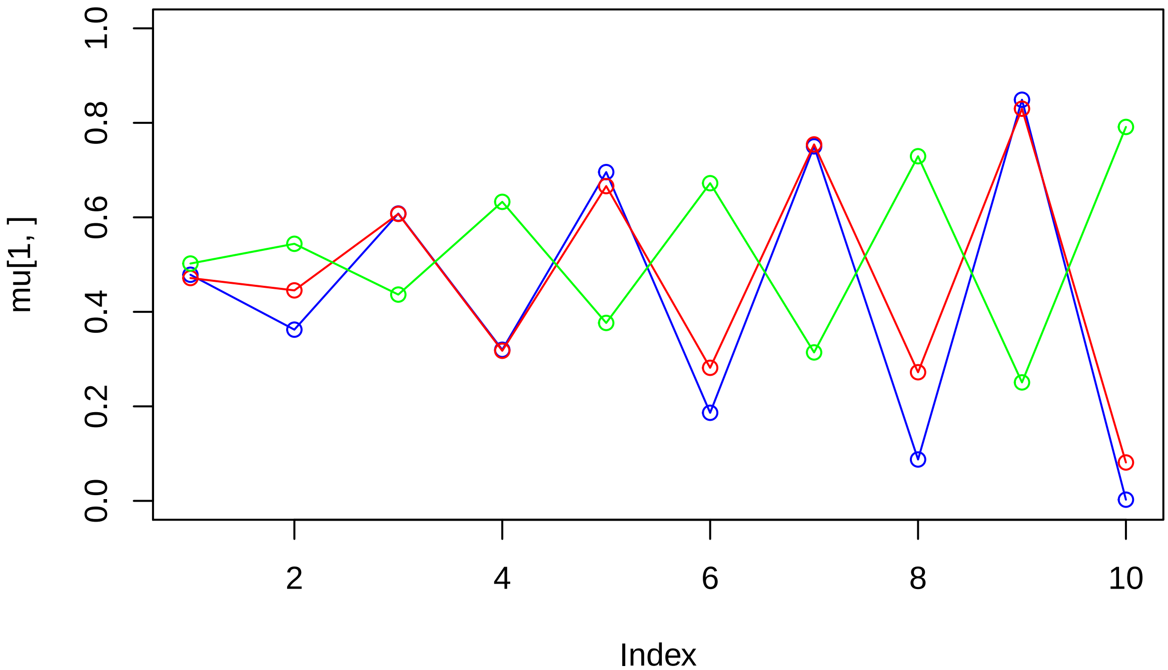

```
## iteration:  22 log likelihood:  -415.3492
```



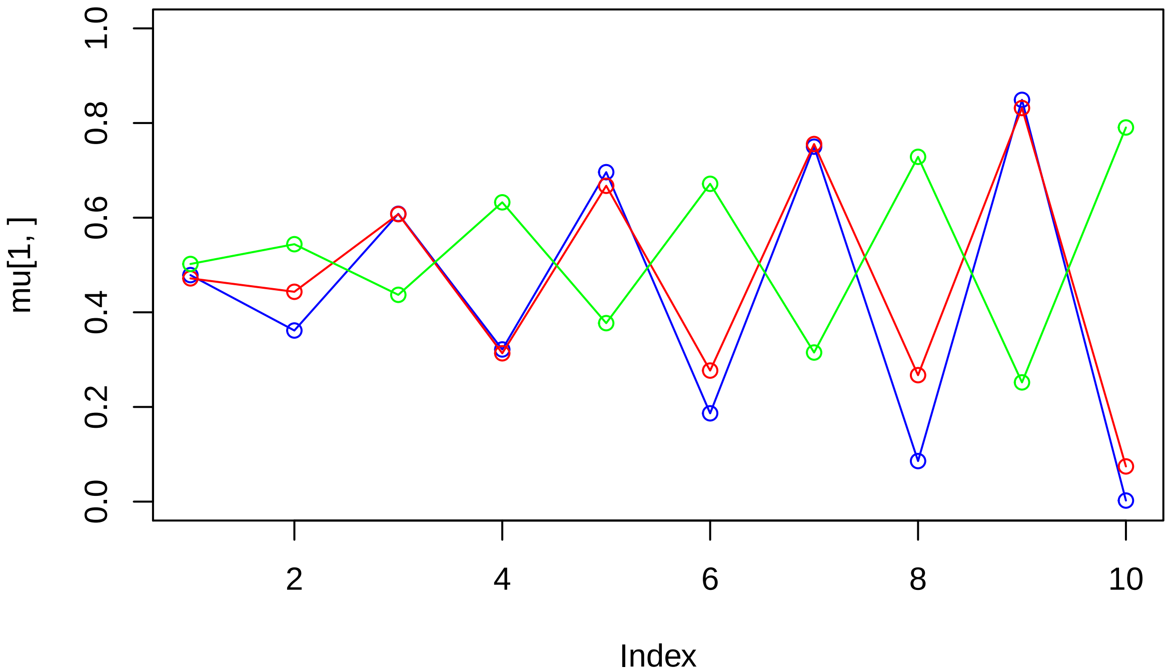

```
## iteration:  23 log likelihood:  -415.9659
```



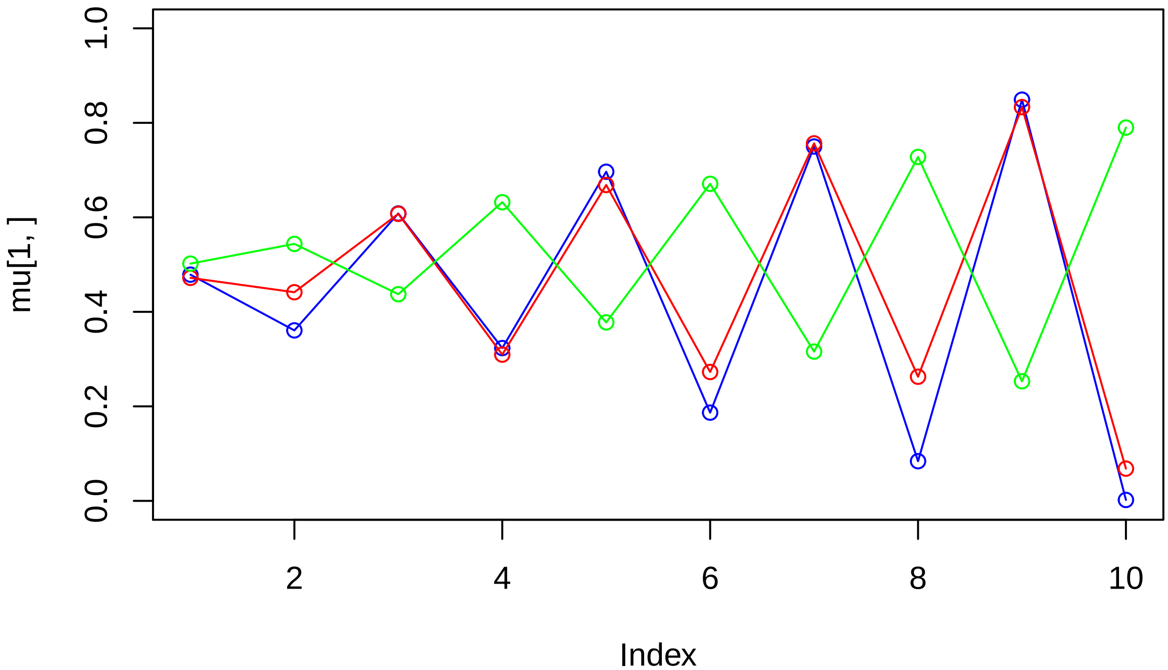

```
## iteration:  24 log likelihood:  -416.532
```



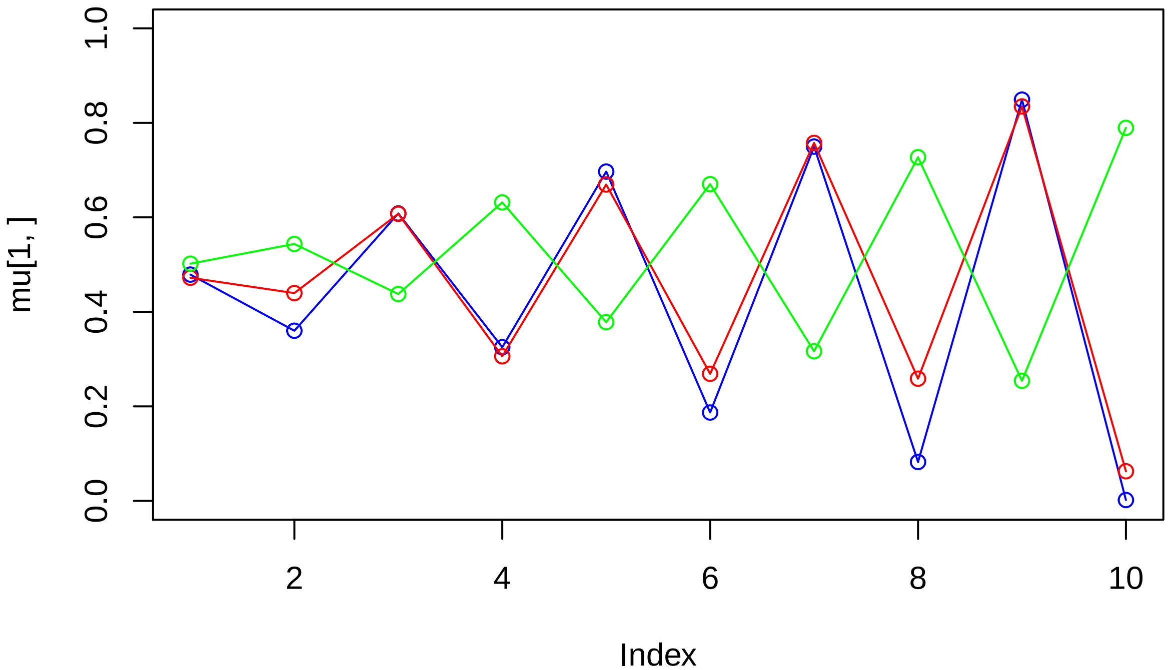

```
## iteration:  25 log likelihood:  -417.0528
```



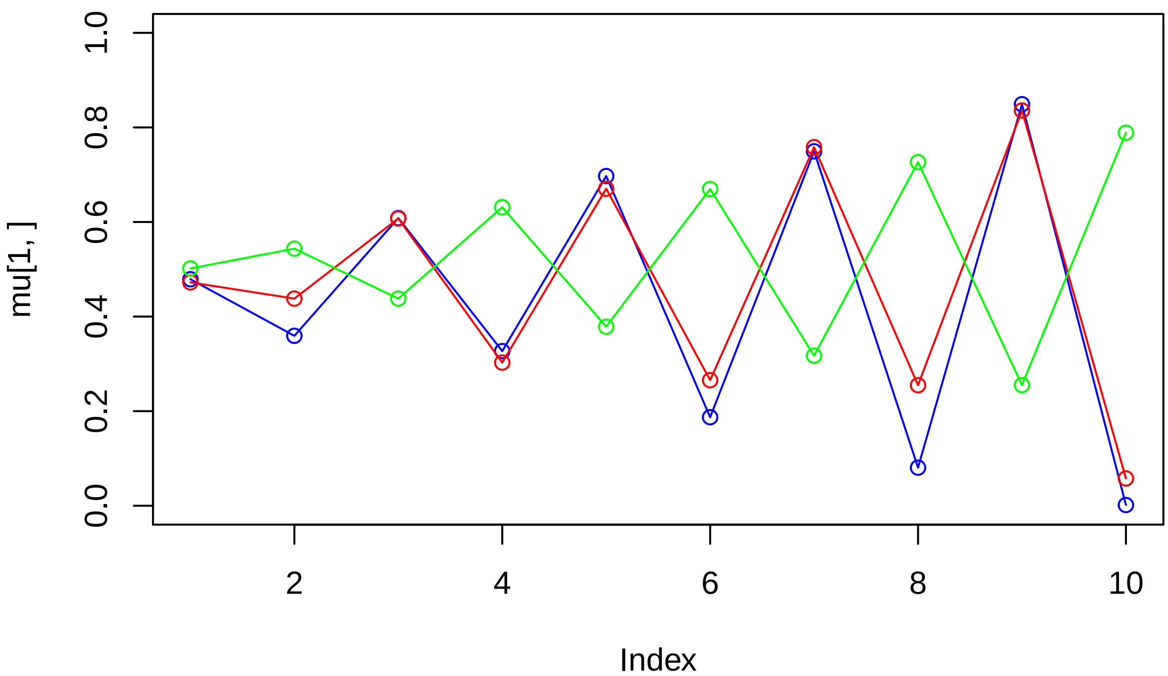

```
## iteration:  26 log likelihood:  -417.5328
```



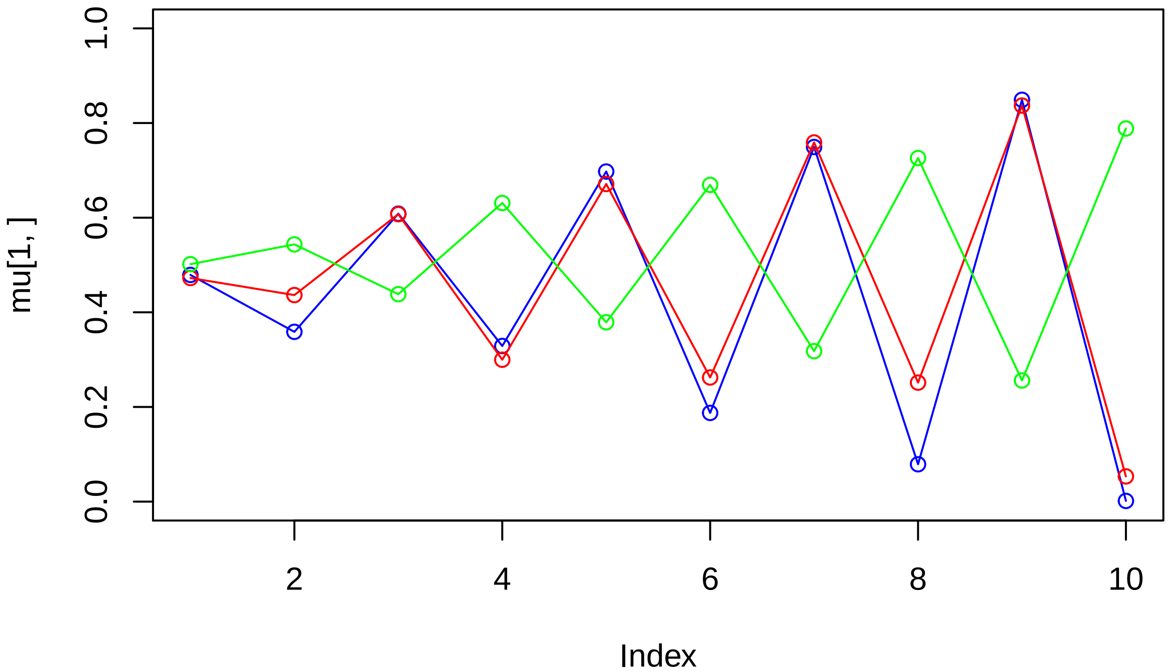

```
## iteration:  27 log likelihood:  -417.9753
```



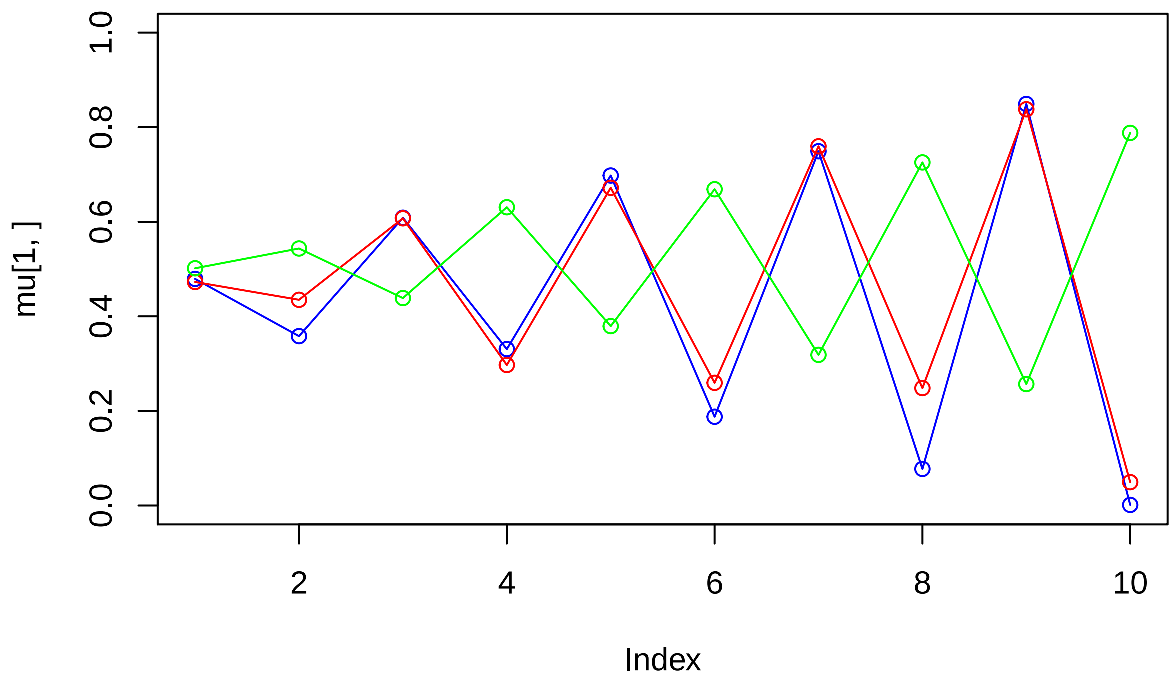

```
## iteration:  28 log likelihood:  -418.3836
```



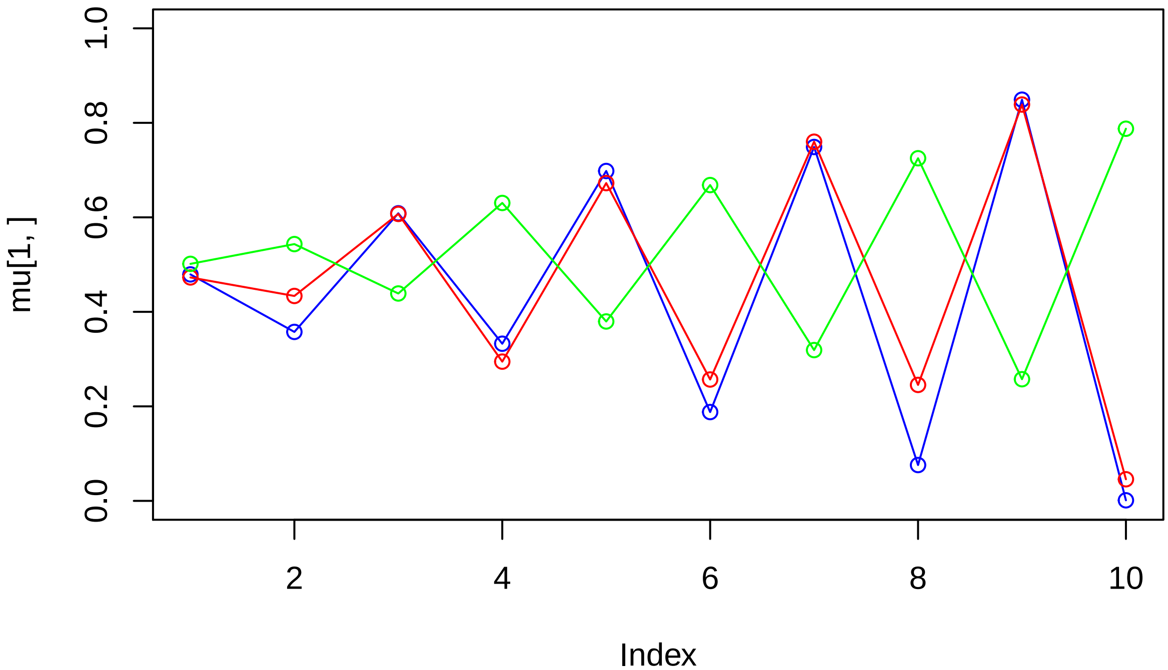

```
## iteration:  29 log likelihood:  -418.7601
```



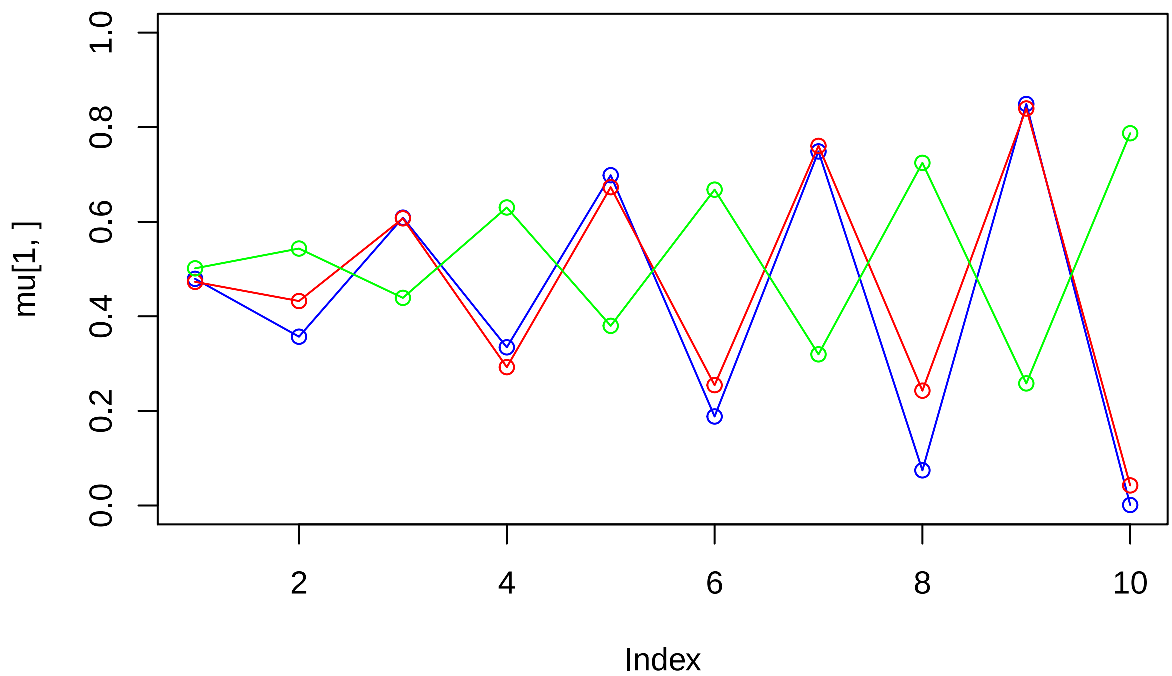

```
## iteration:  30 log likelihood:  -419.1074
```



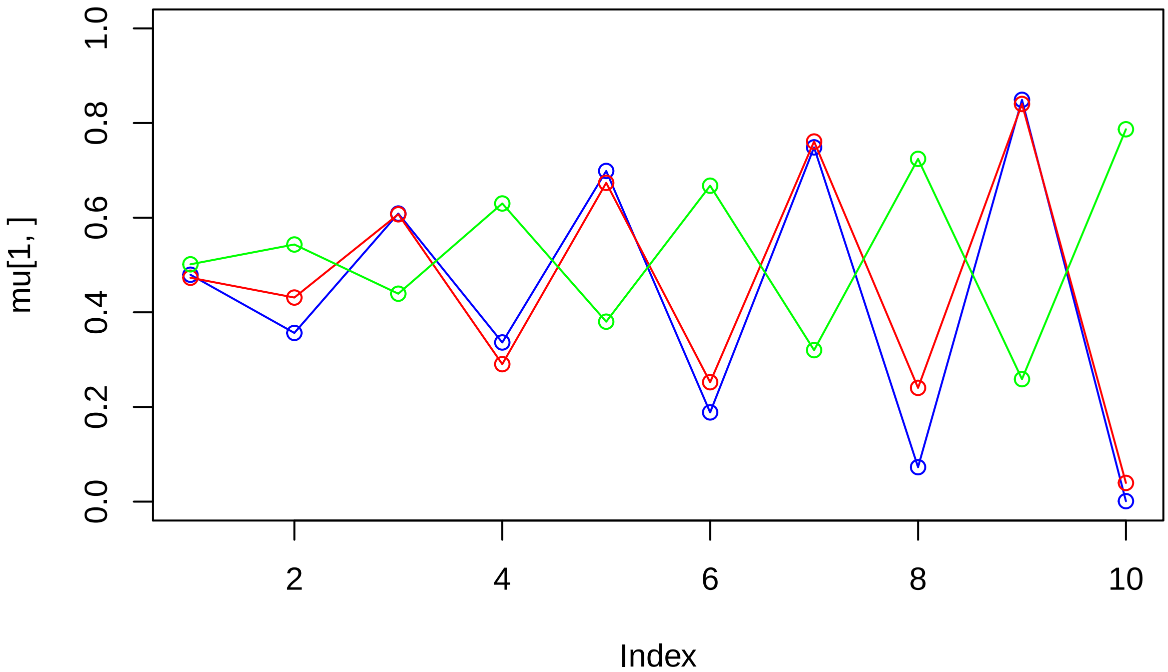

```
## iteration:  31 log likelihood:  -419.4277
```



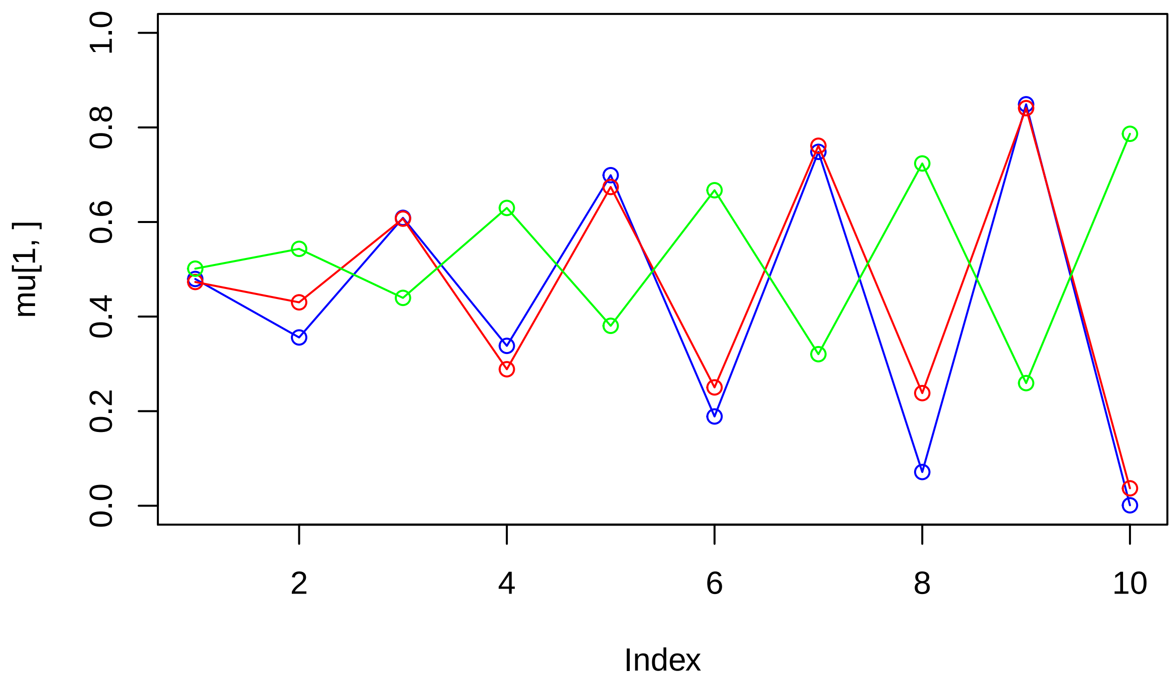

```
## iteration:  32 log likelihood:  -419.7229
```



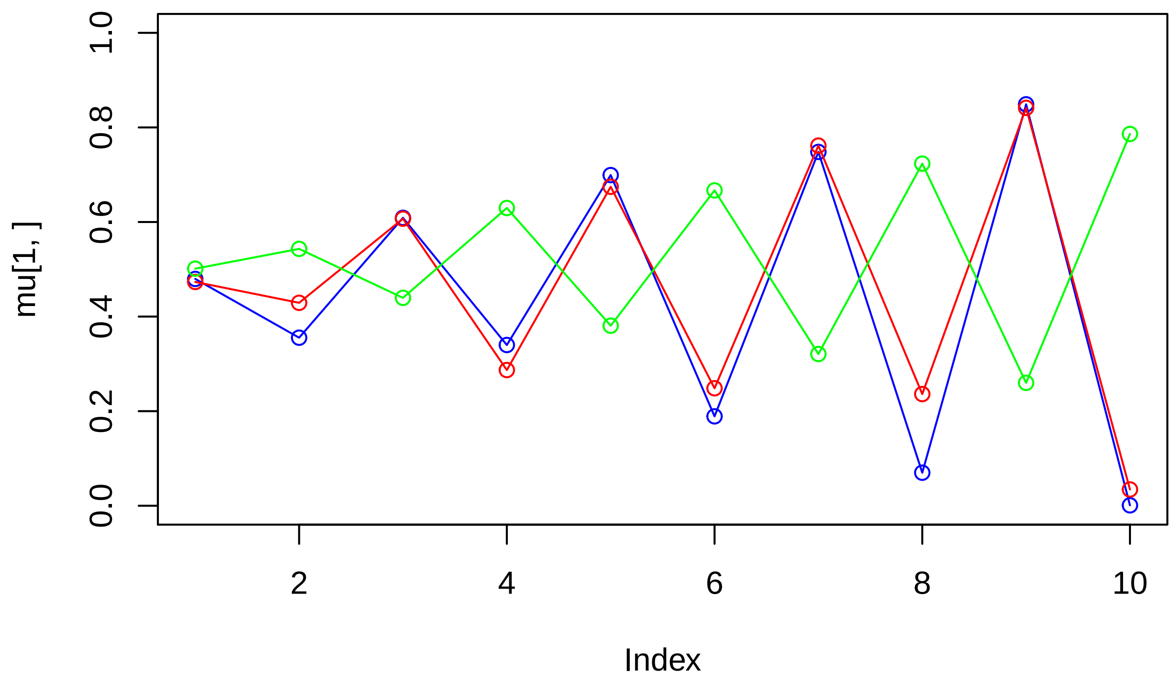

```
## iteration:  33 log likelihood:  -419.995
```



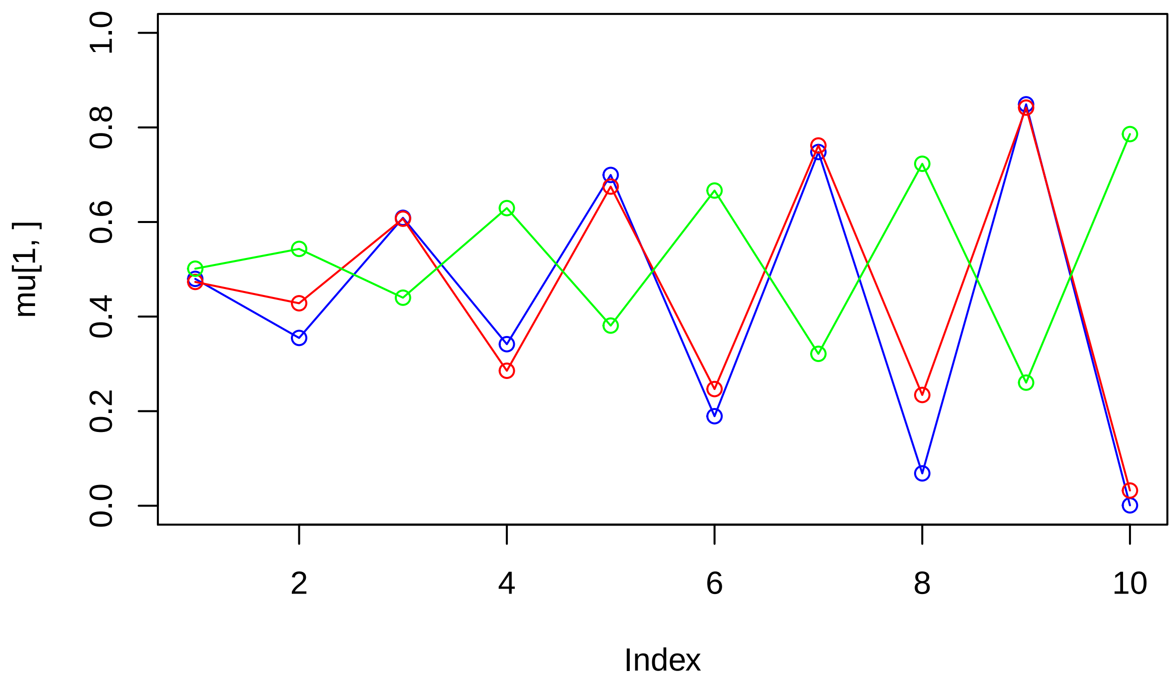

```
## iteration:  34 log likelihood:  -420.2457
```



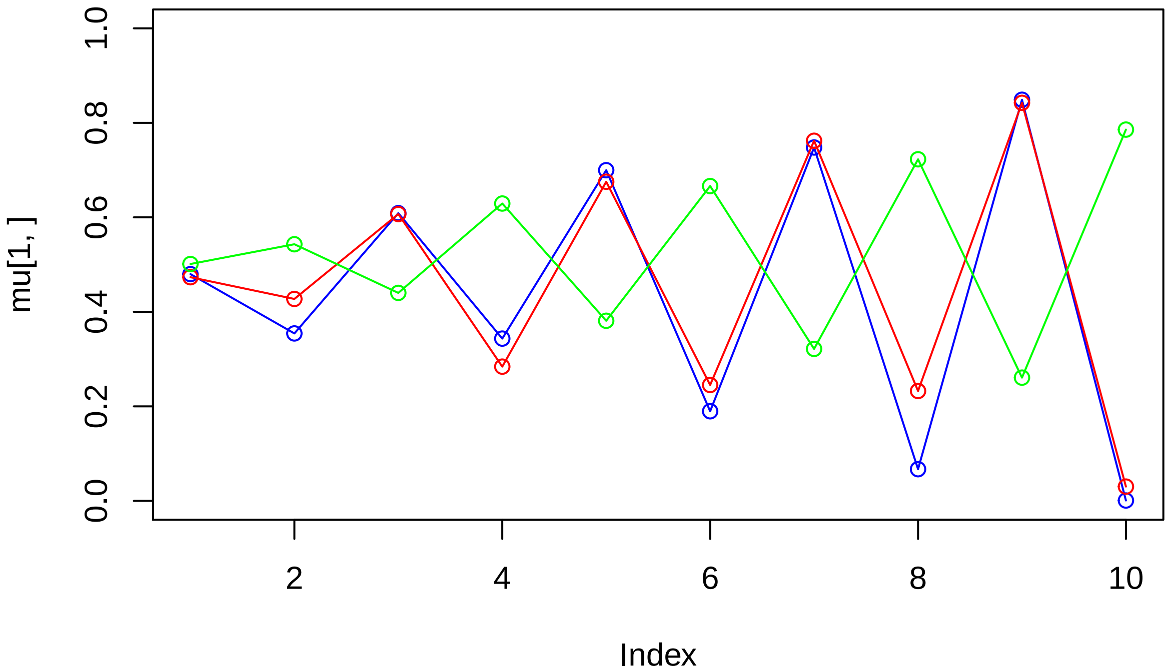

```
## iteration:  35 log likelihood:  -420.4767
```



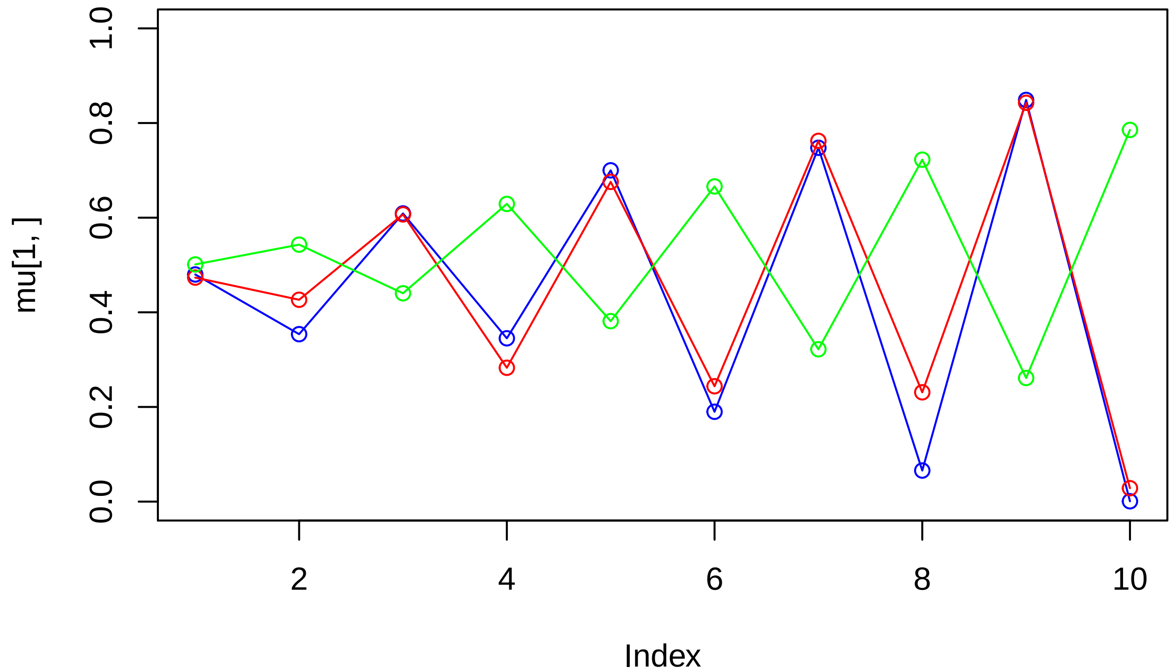

```
## iteration:  36 log likelihood:  -420.6895
```



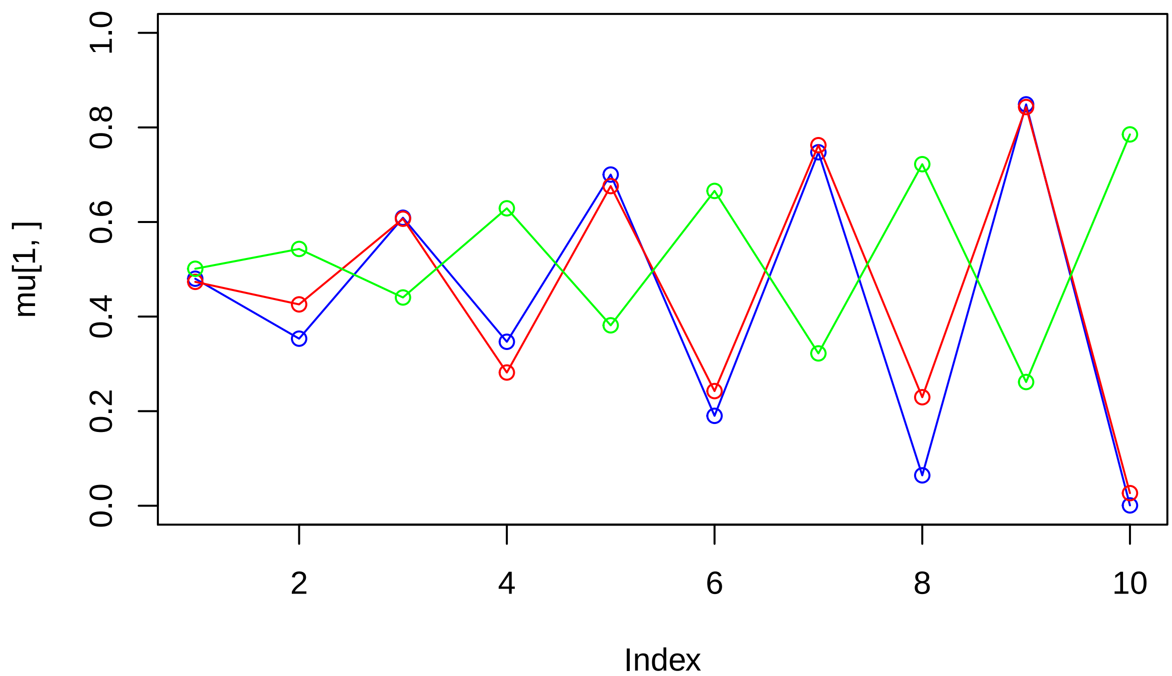

```
## iteration:  37 log likelihood:  -420.8856
```



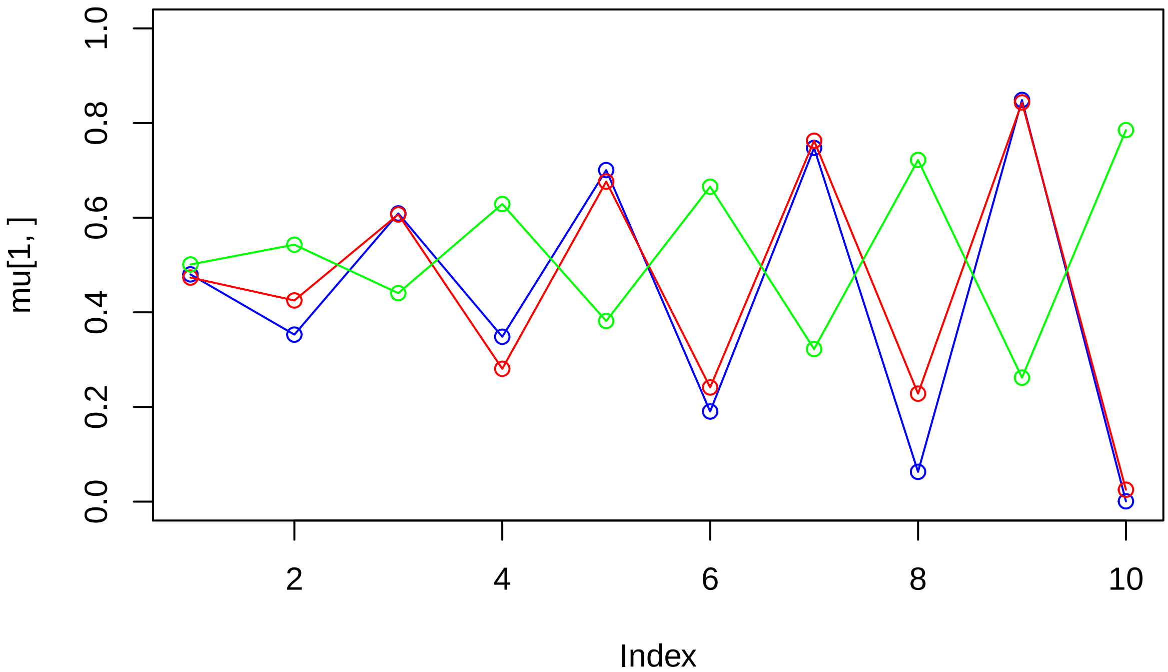

```
## iteration:  38 log likelihood:  -421.0663
```



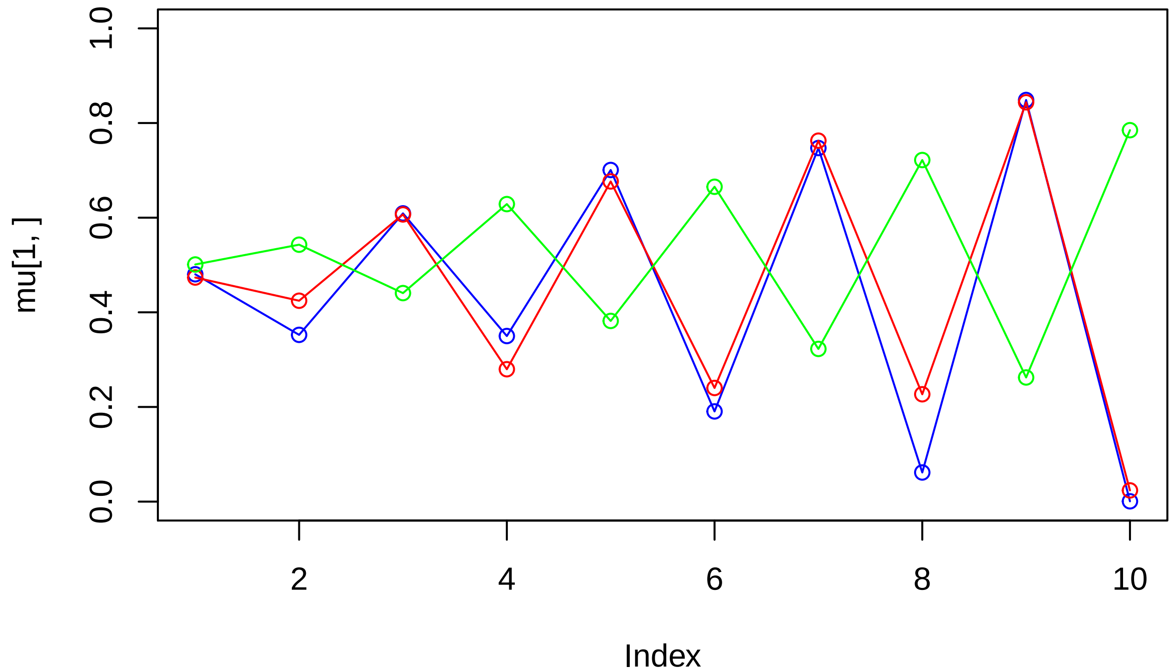

```
## iteration:  39 log likelihood:  -421.2329
```



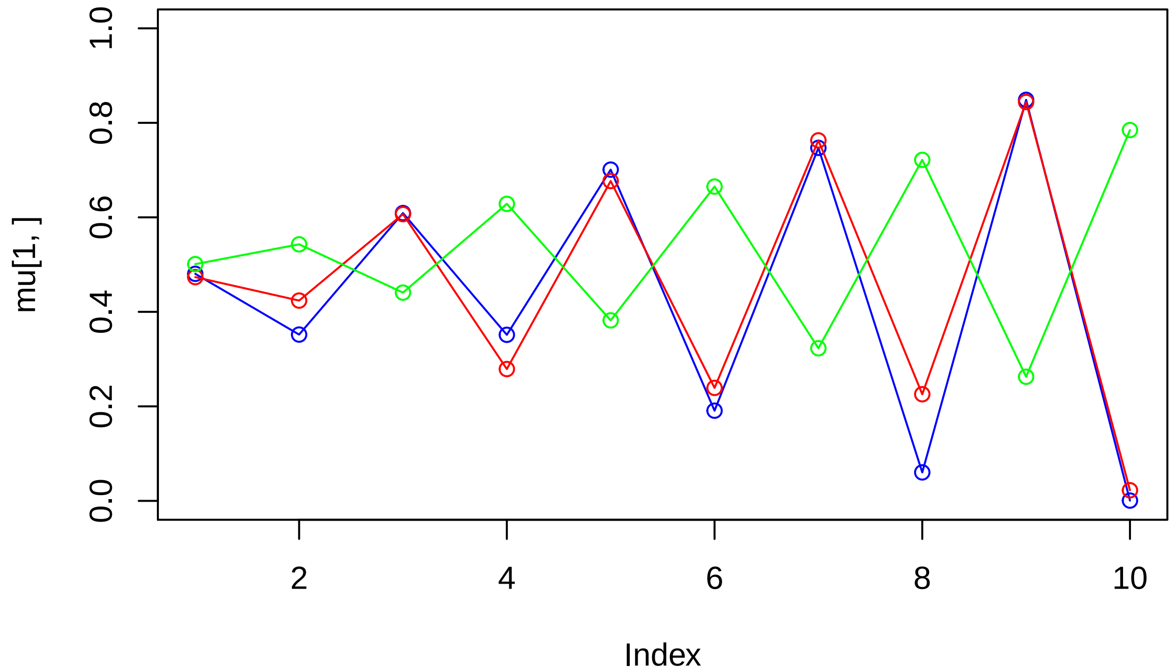

```
## iteration:  40 log likelihood:  -421.3865
```



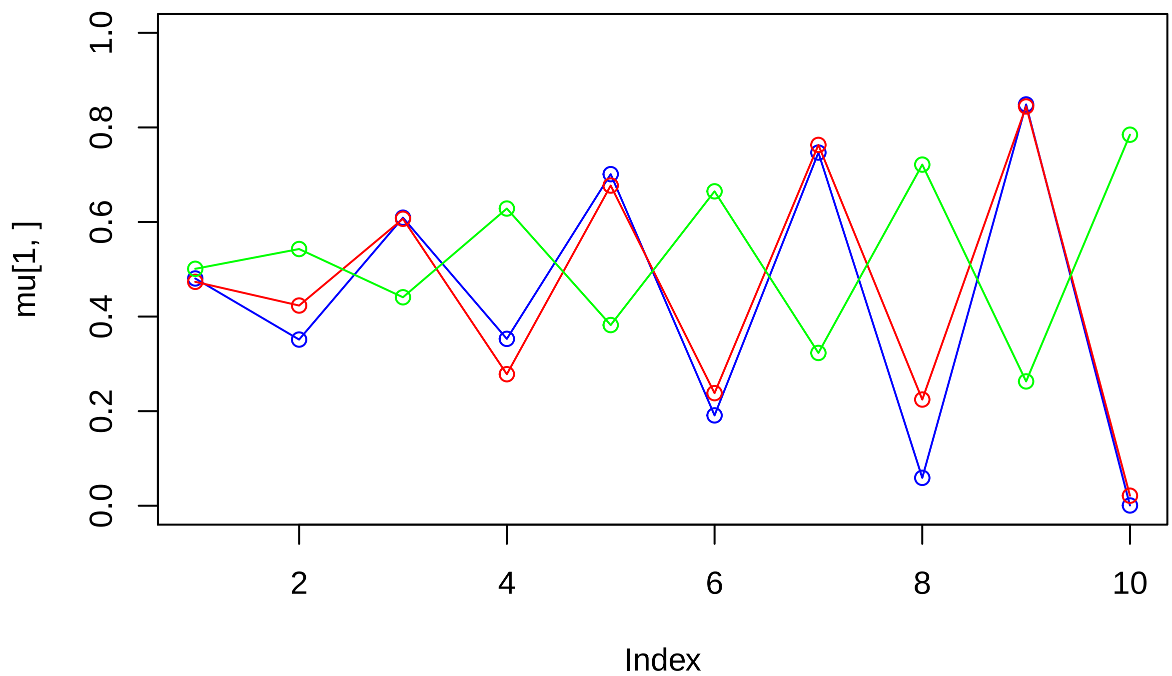

```
## iteration:  41 log likelihood:  -421.5282
```



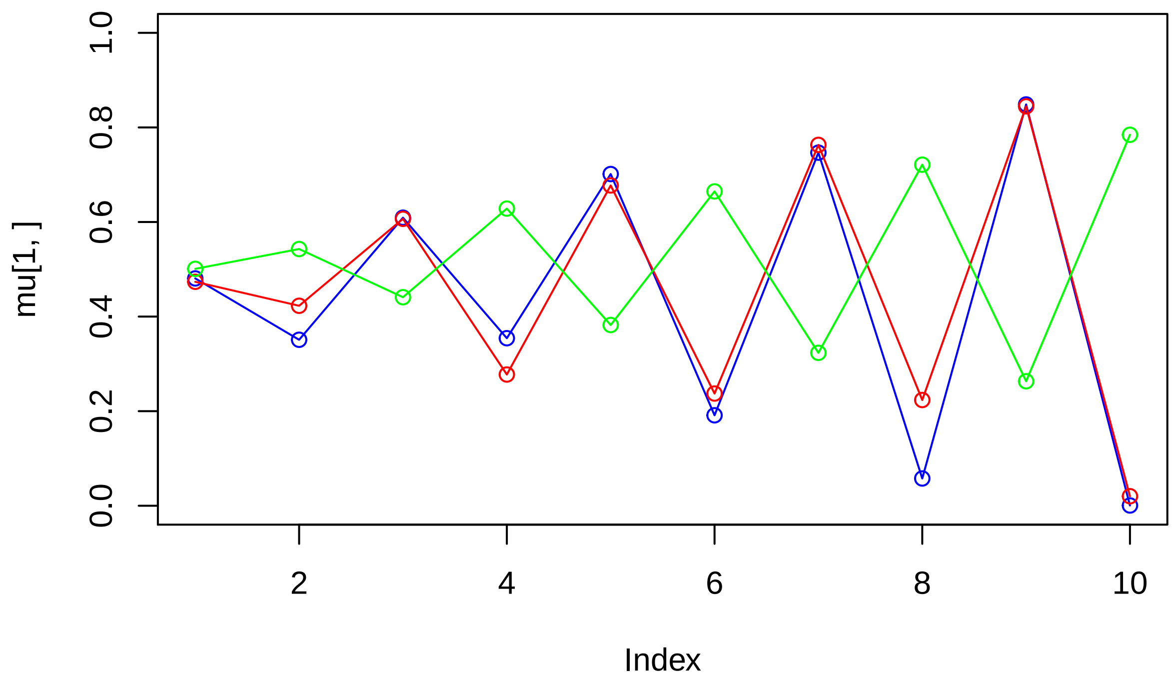

```
## iteration:  42 log likelihood:  -421.659
```



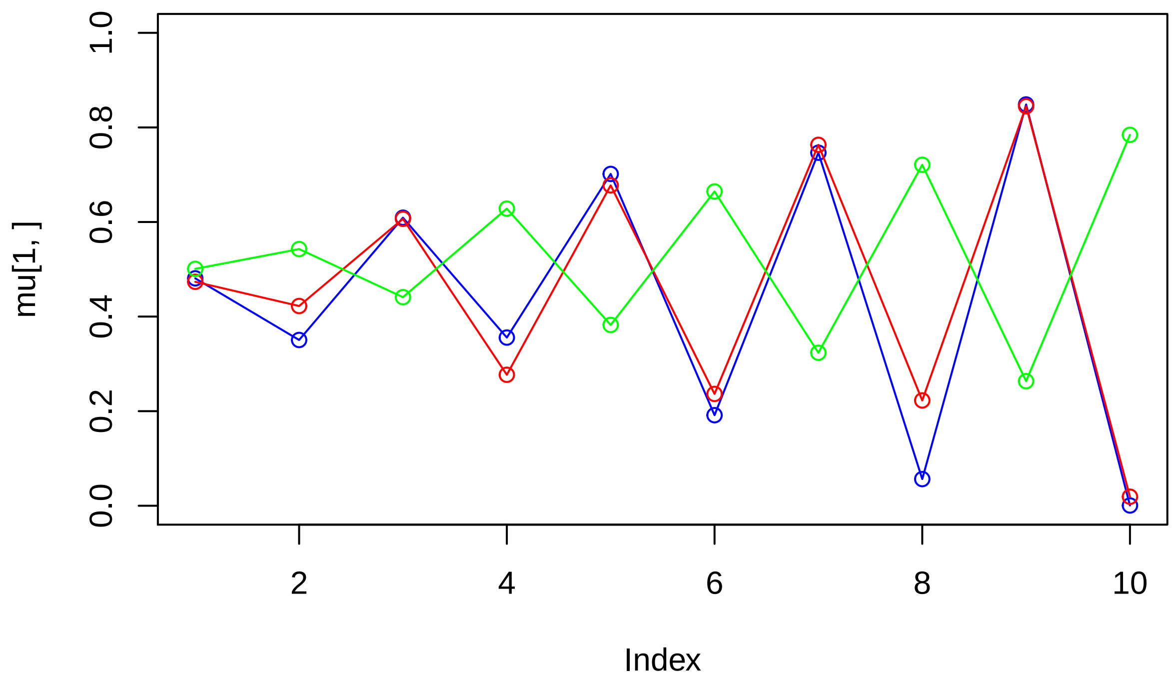

```
## iteration:  43 log likelihood:  -421.7797
```



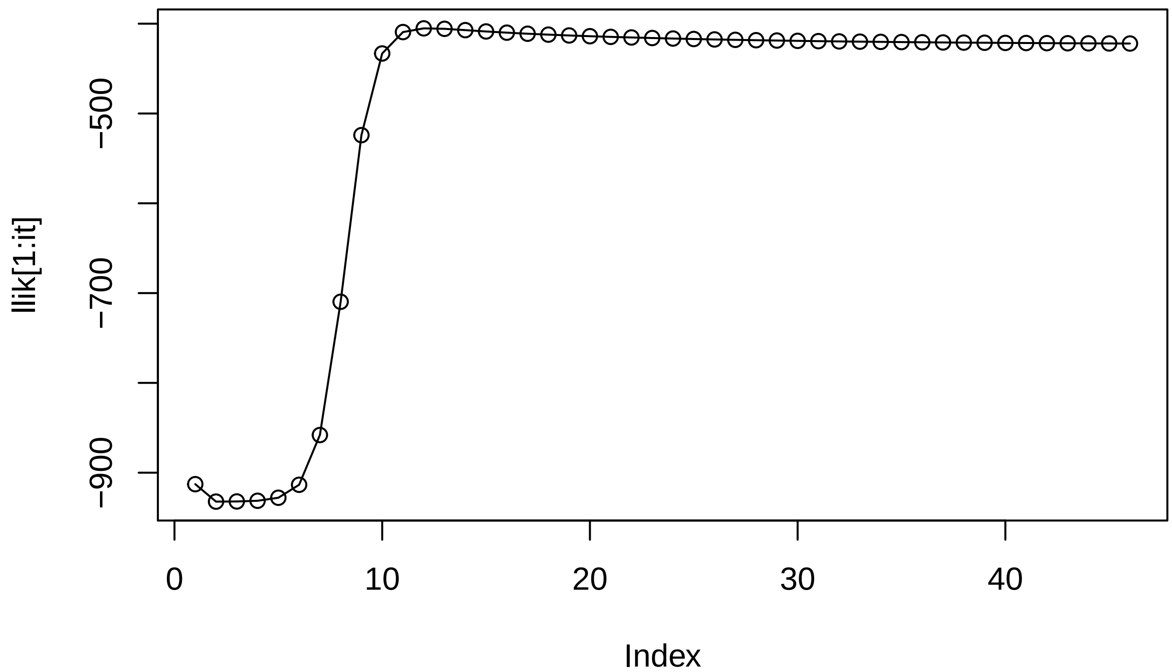

```
## iteration:  44 log likelihood:  -421.8913
```




```
## iteration:  45 log likelihood:  -421.9945
```



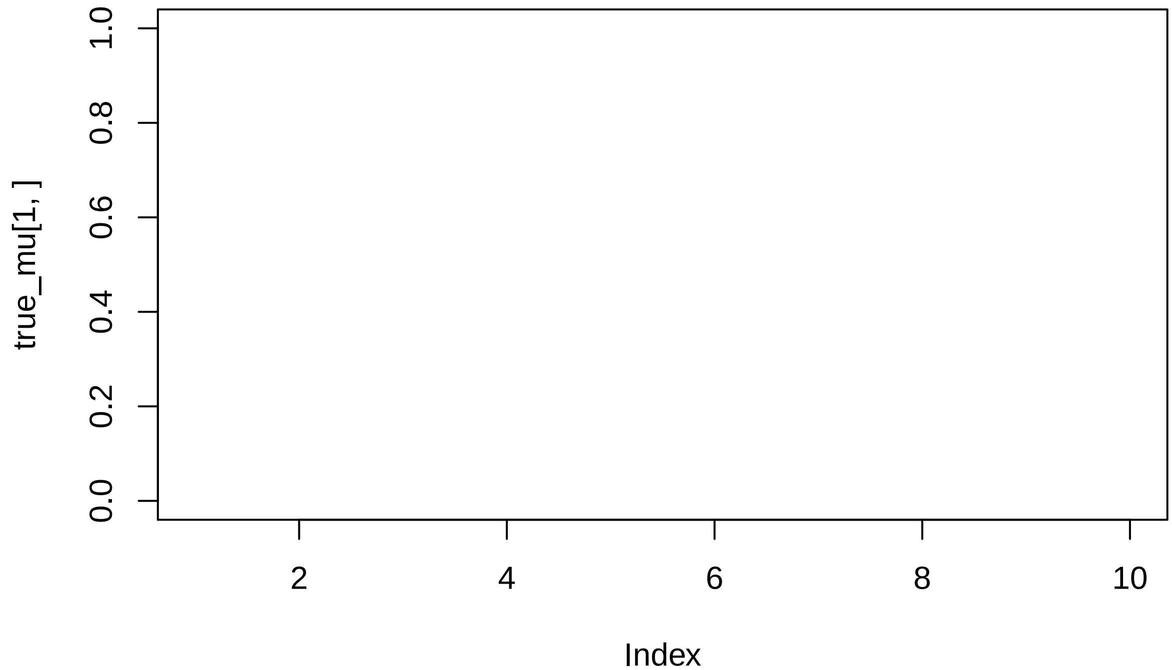

```
## iteration:  46 log likelihood:  -422.09
```

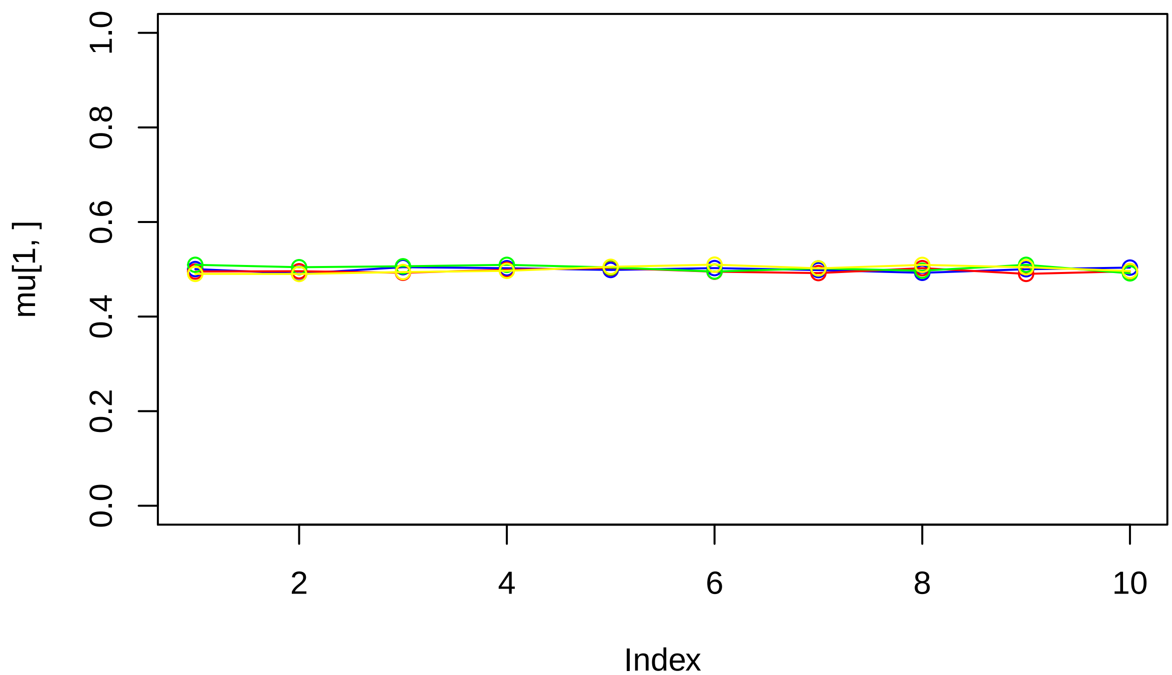



```
## [[1]]
##  [1] 0.1679716522 0.2034249451 0.6286034026 0.4808697447 0.4735292886
##  [6] 0.5009515437 0.3505033441 0.4223594793 0.5428015802 0.6091318393
## [11] 0.6067582335 0.4410624521 0.3556548071 0.2768901990 0.6282716562
## [16] 0.7016525159 0.6775124264 0.3823067747 0.1914725060 0.2364291675
## [21] 0.6645564670 0.7465112147 0.7631736115 0.3235087926 0.0563854887
## [26] 0.2226418271 0.7210236740 0.8485479479 0.8448194739 0.2634580777
## [31] 0.0005534402 0.0190935069 0.7843147843
```

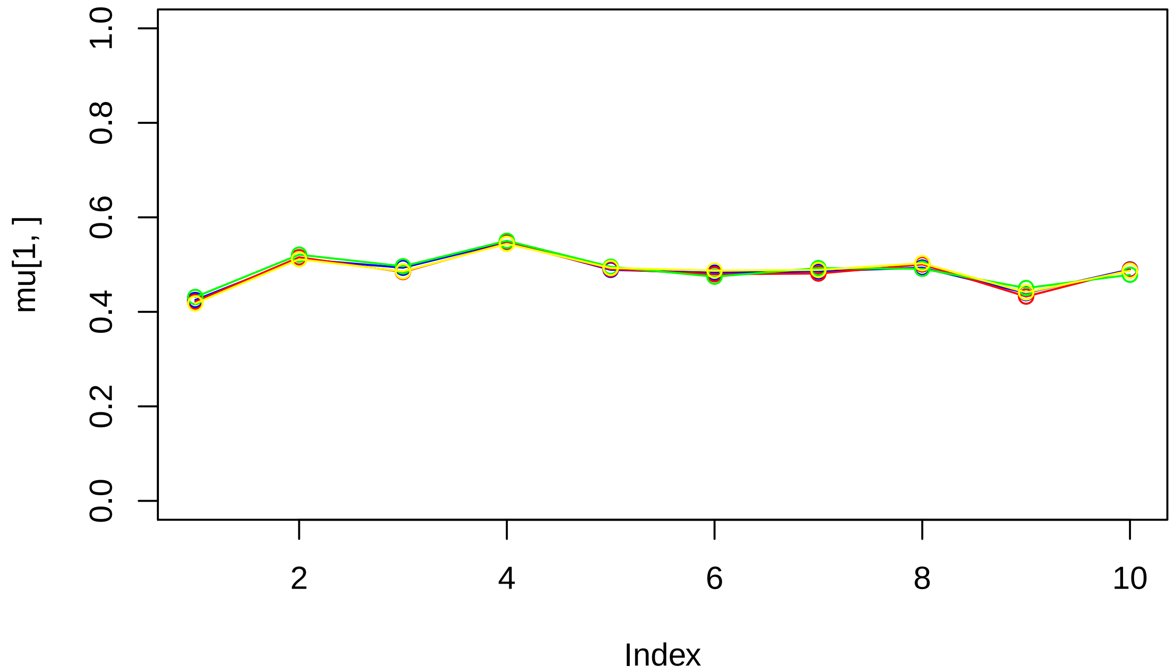
$\mathbf{K = 4}$

```r
myem(K=4)
```

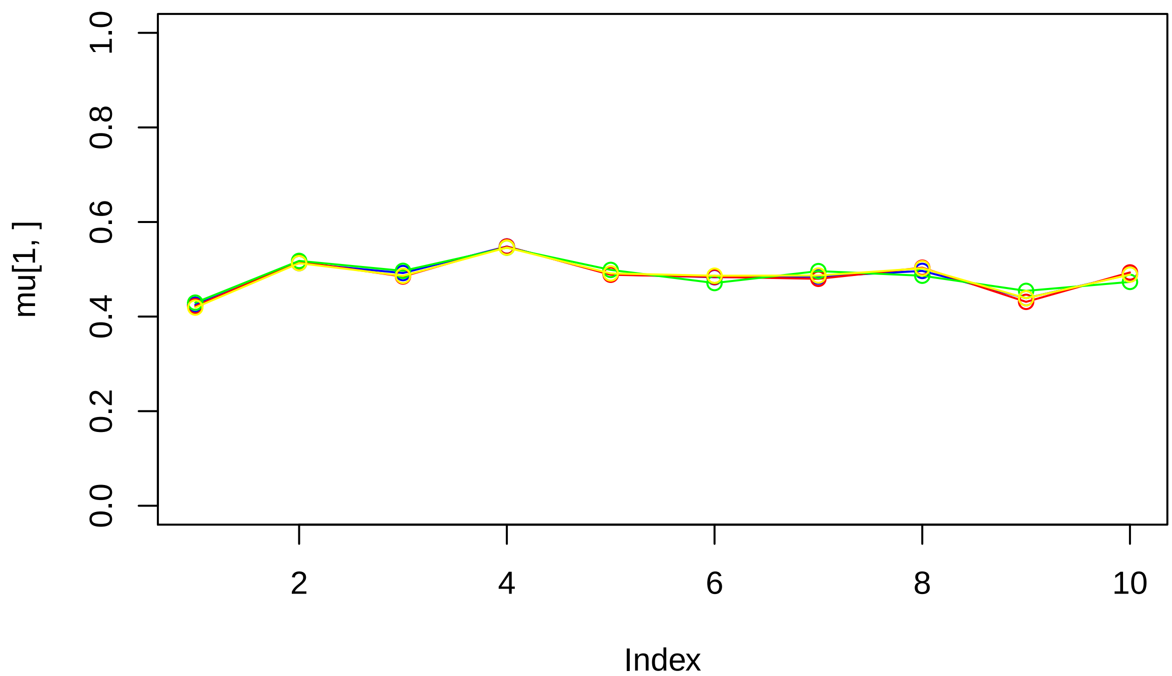

true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



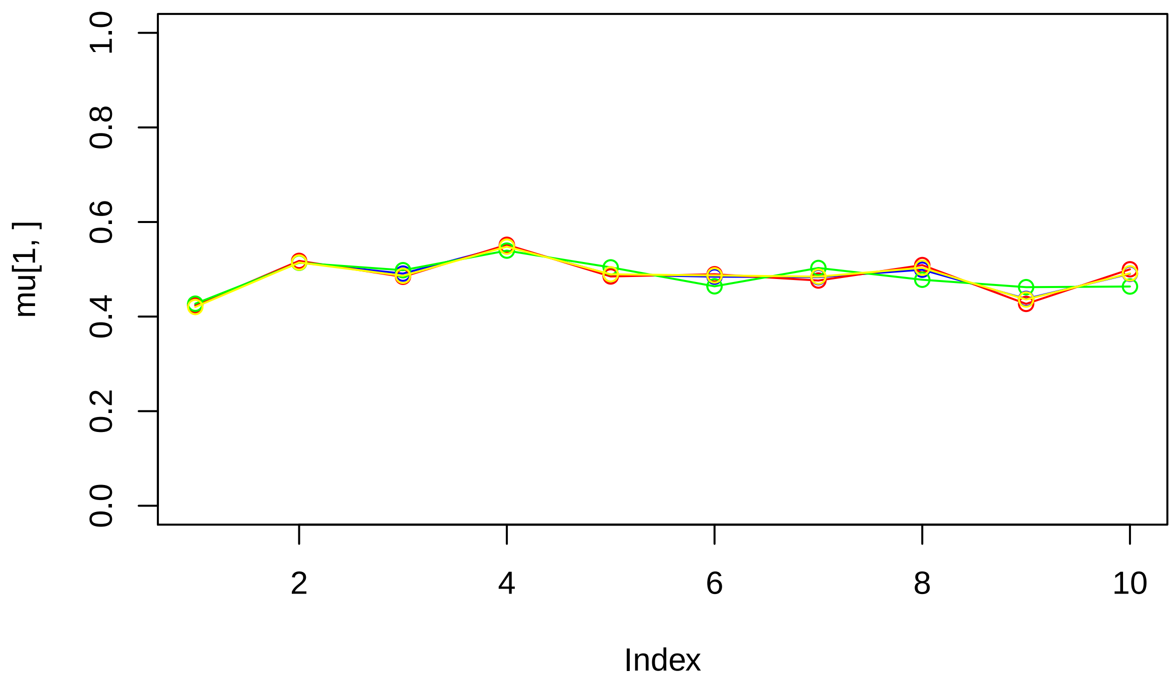

```
## iteration:  1 log likelihood:  -800.5436
```



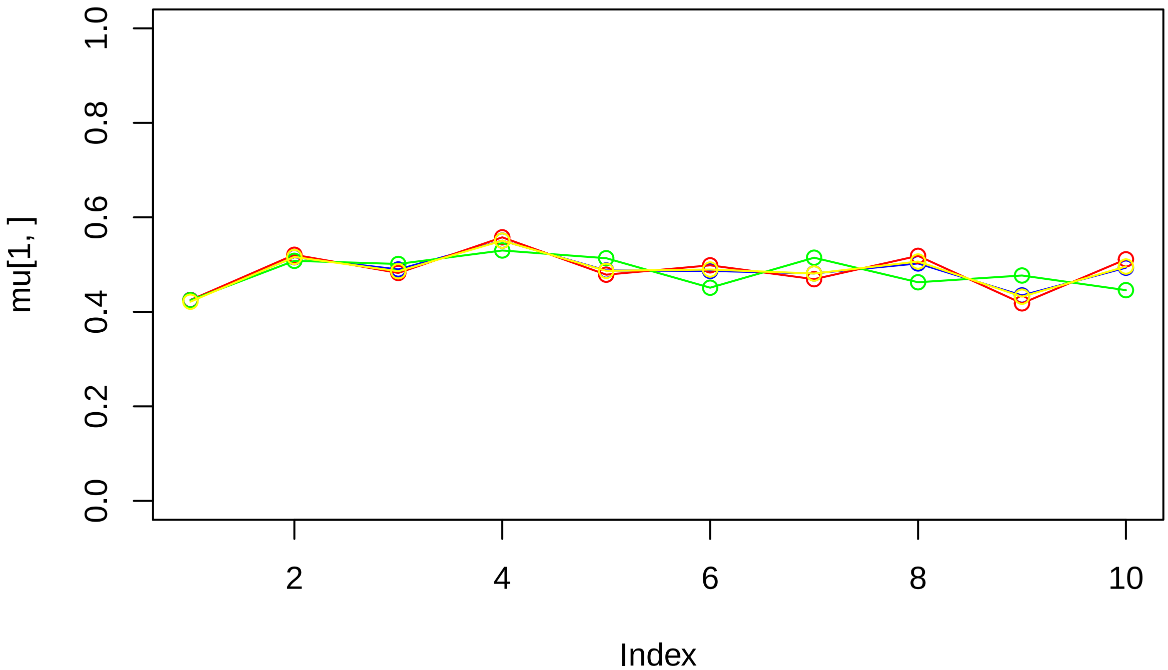

```
## iteration:  2 log likelihood:  -842.949
```



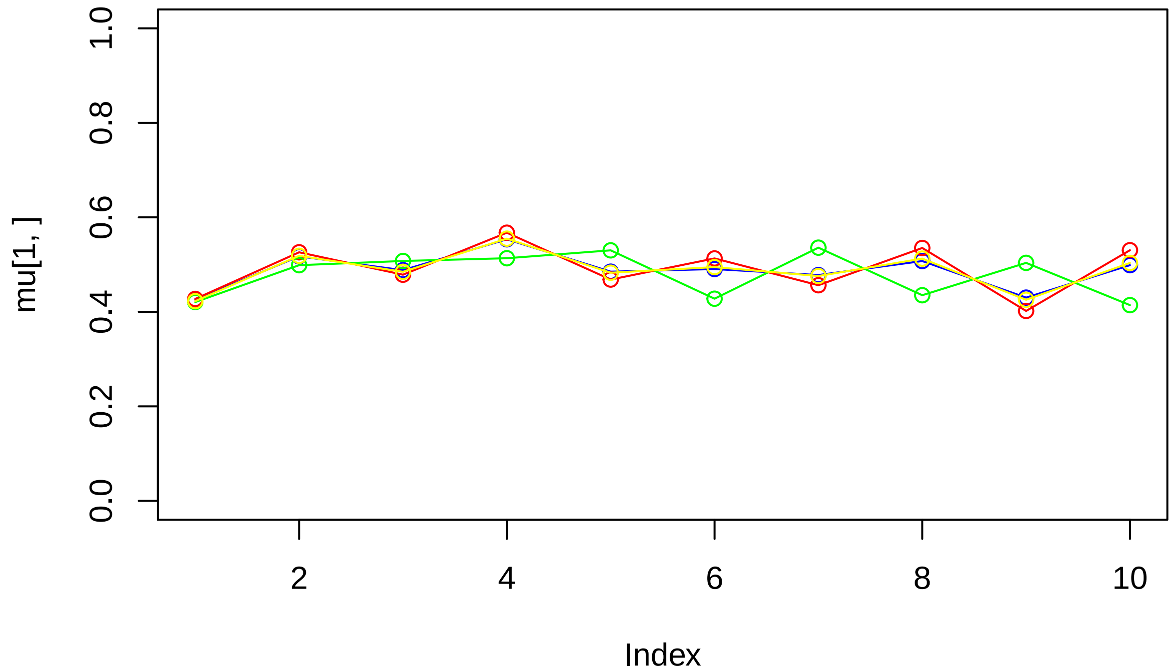

```
## iteration:  3 log likelihood:  -842.6806
```



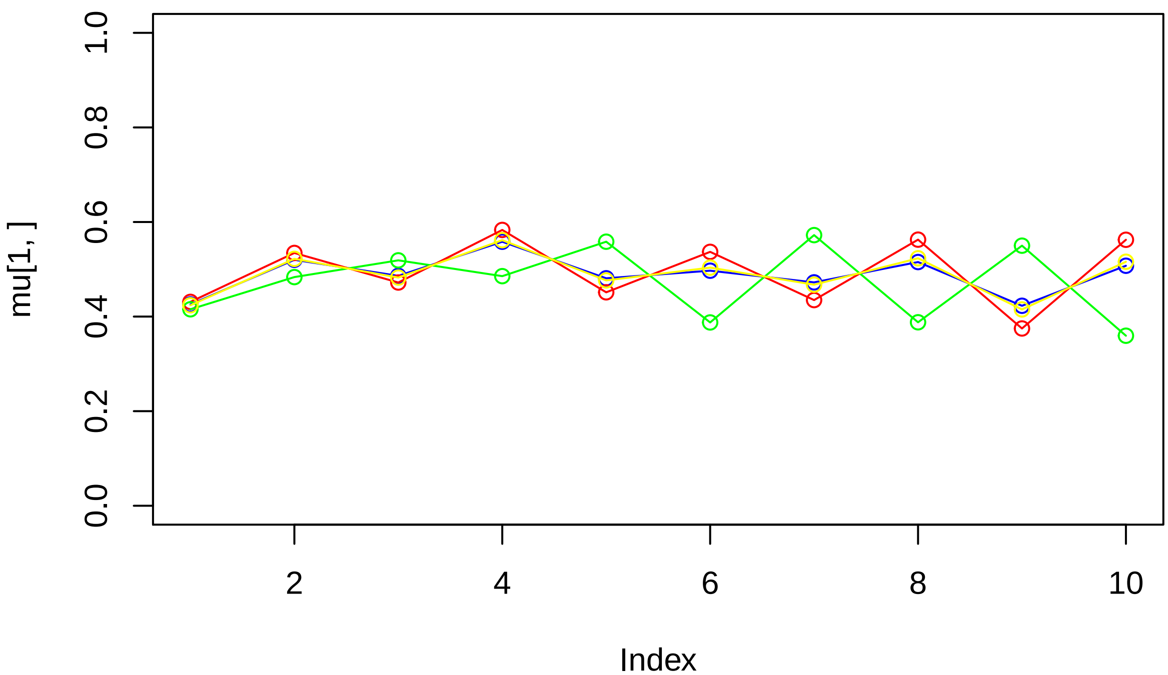

```
## iteration:  4 log likelihood:  -841.7499
```



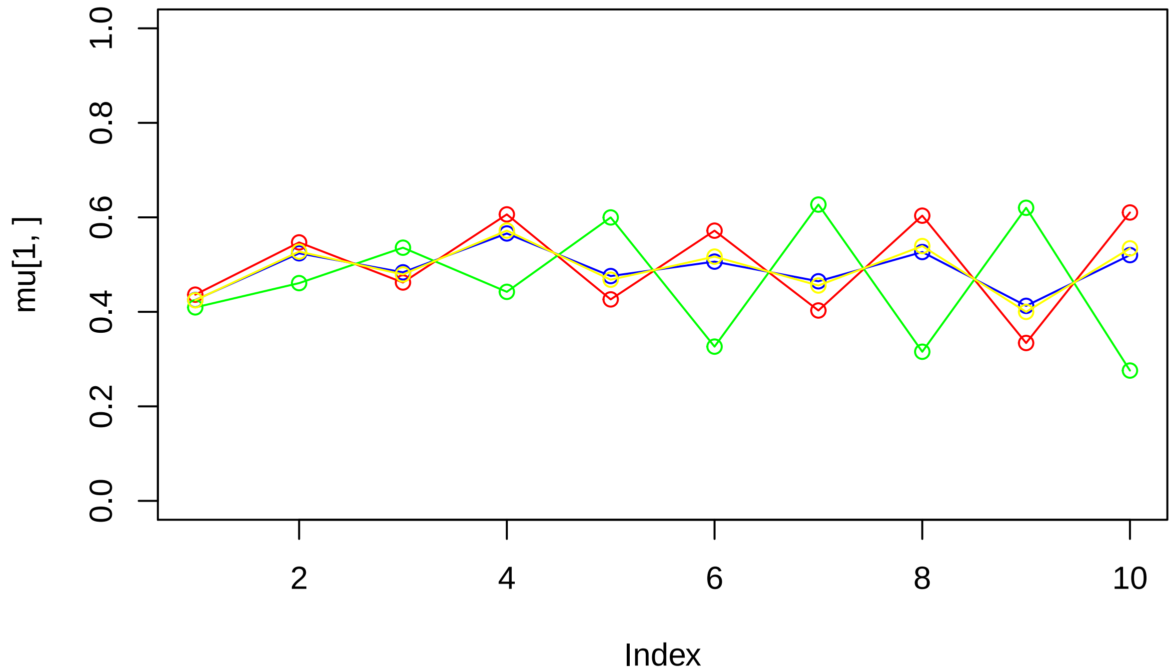

```
## iteration:  5 log likelihood:  -838.7414
```



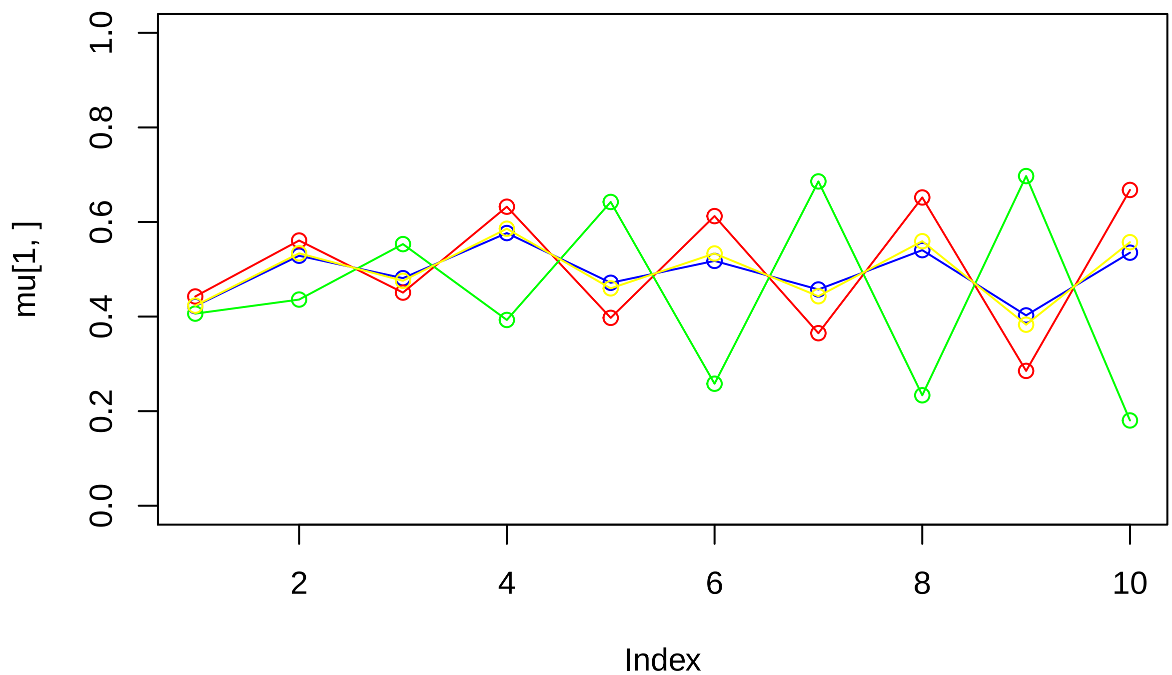

```
## iteration:  6 log likelihood:  -829.4624
```



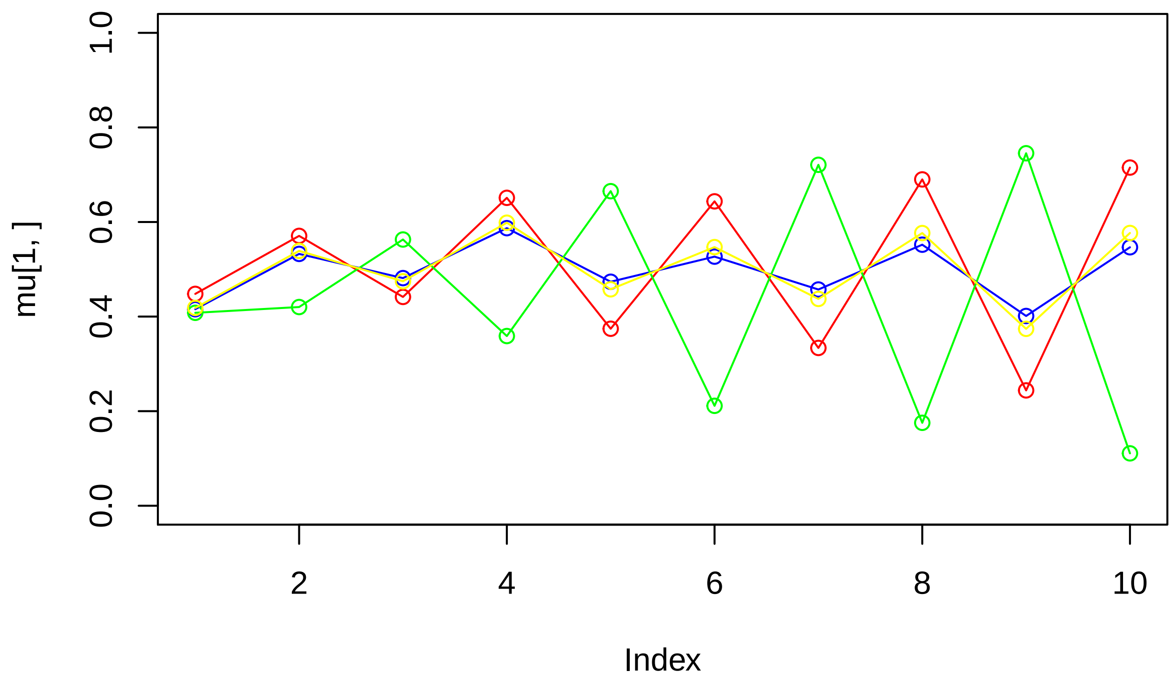

```
## iteration:  7 log likelihood:  -803.3592
```



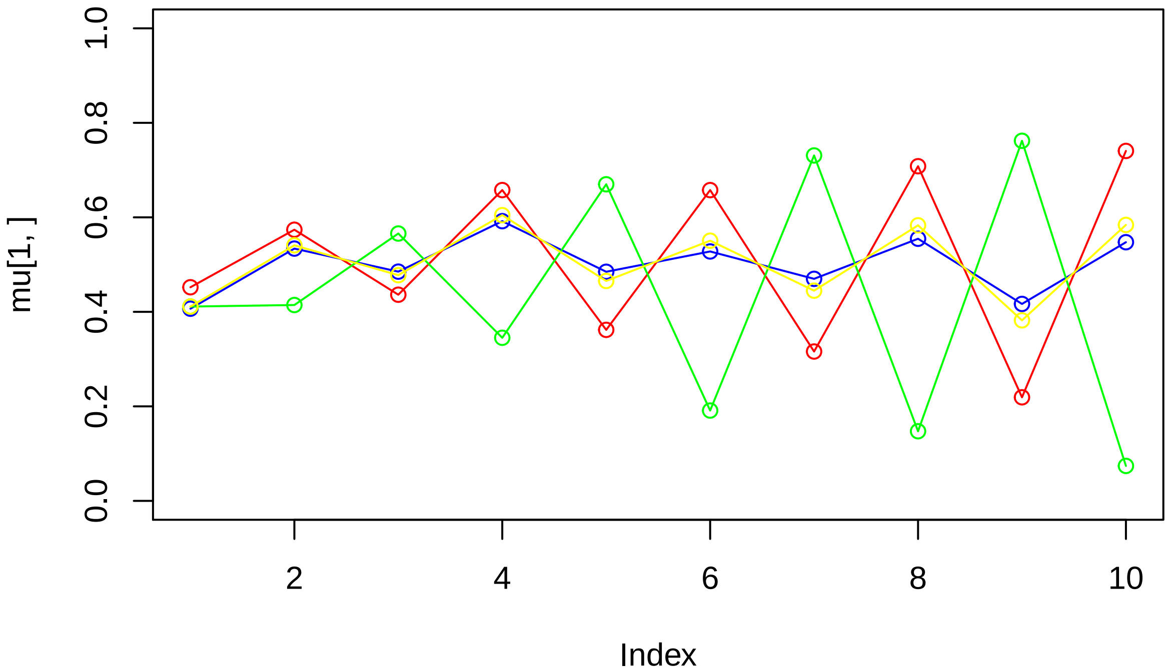

```
## iteration:  8 log likelihood:  -744.3623
```



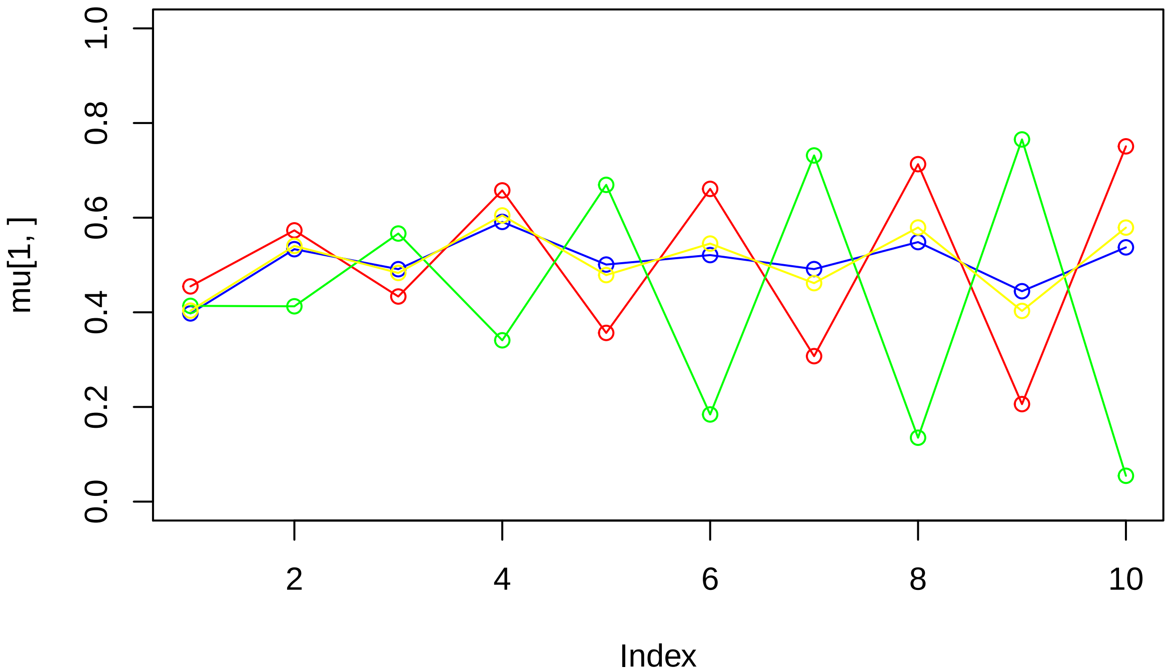

```
## iteration:  9 log likelihood:  -658.0191
```



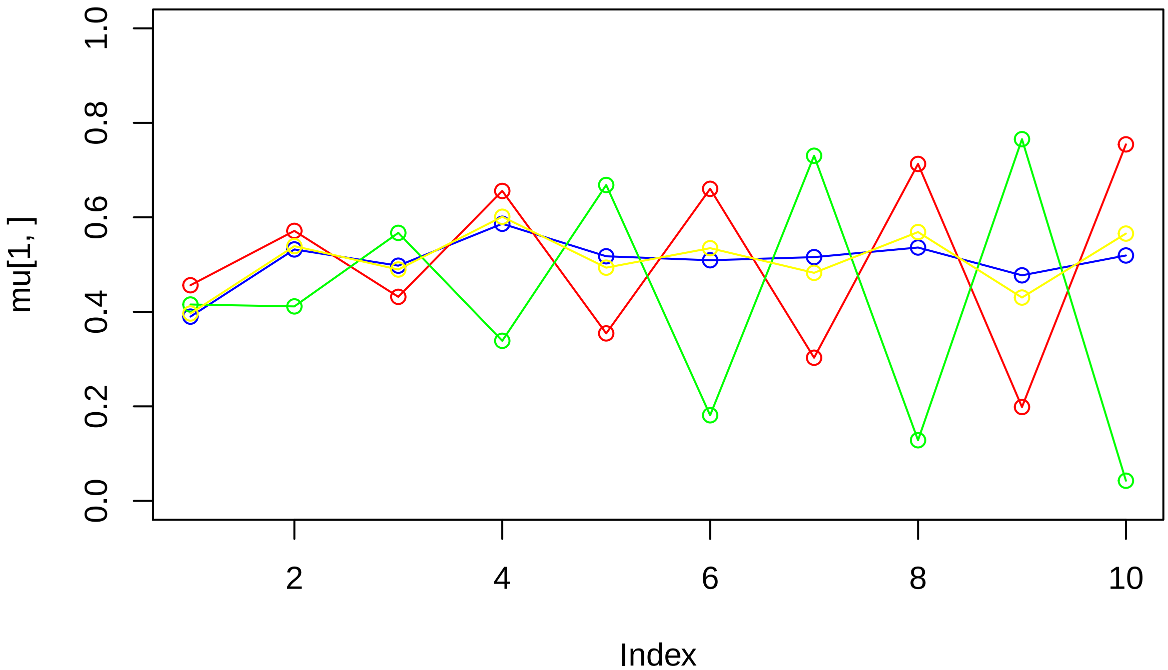

```
## iteration:  10 log likelihood:  -588.2999
```



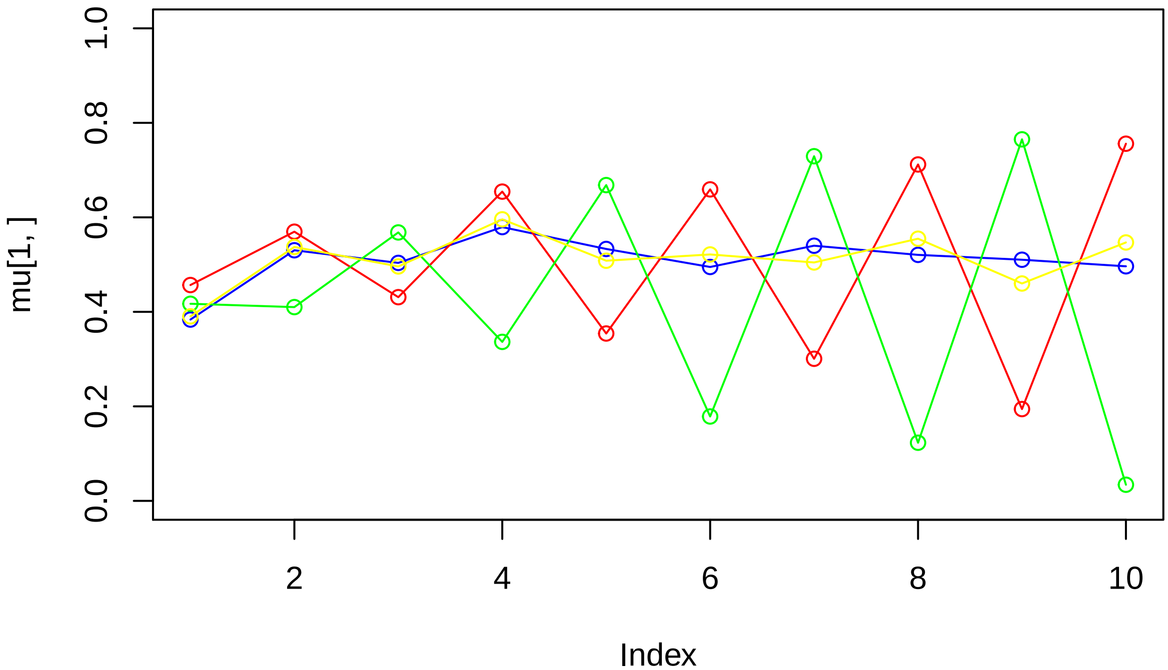

```
## iteration:  11 log likelihood:  -553.5615
```



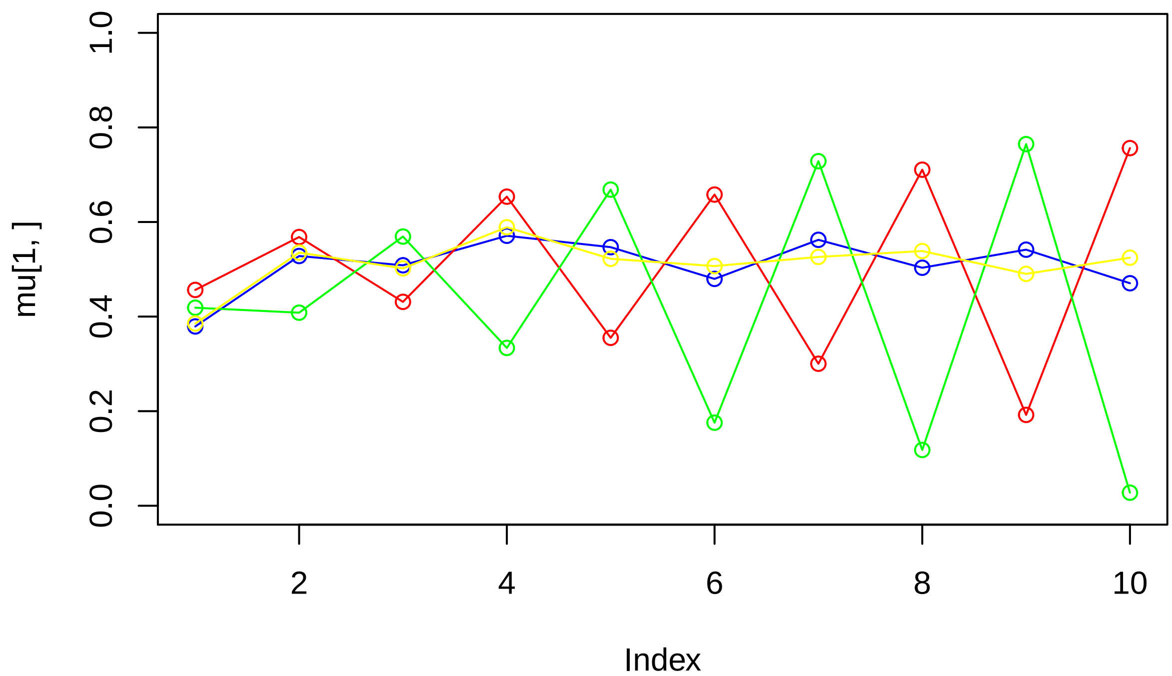

```
## iteration:  12 log likelihood:  -538.8823
```



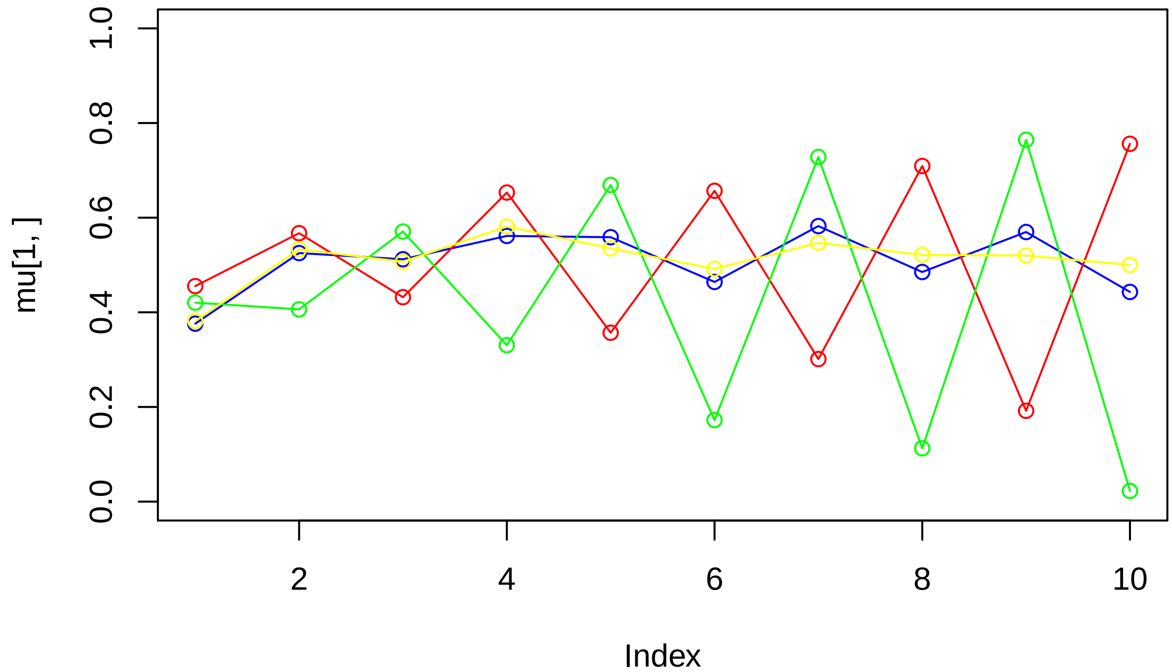

```
## iteration:  13 log likelihood:  -531.9182
```



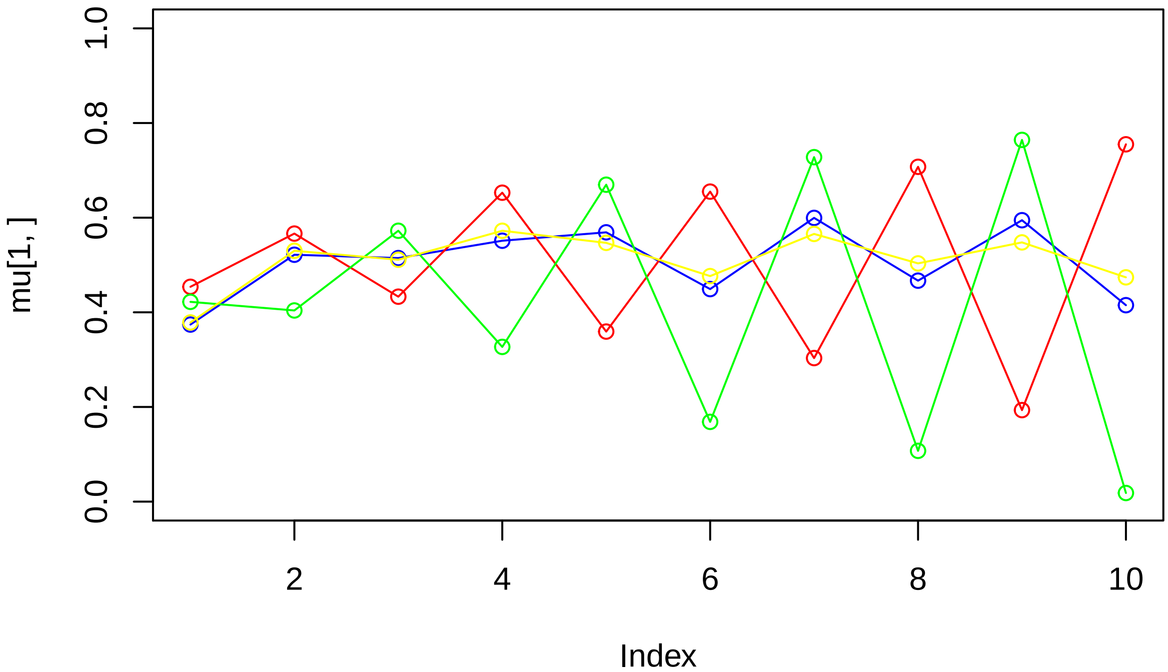

```
## iteration:  14 log likelihood:  -527.7567
```



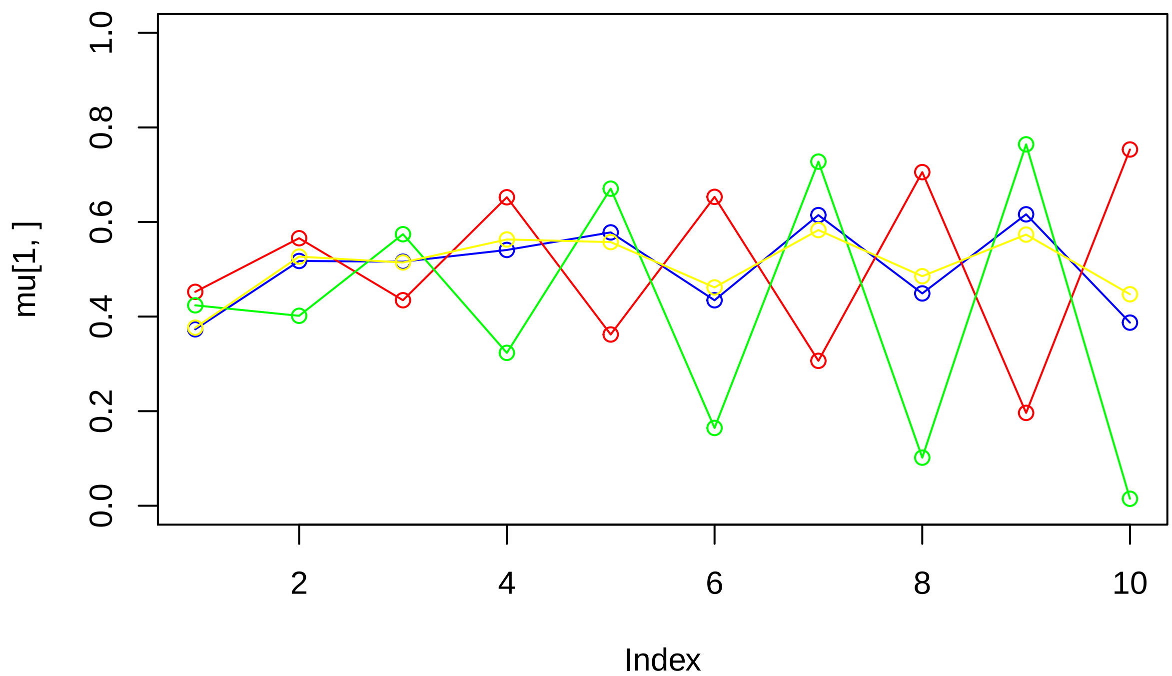

```
## iteration:  15 log likelihood:  -524.8526
```



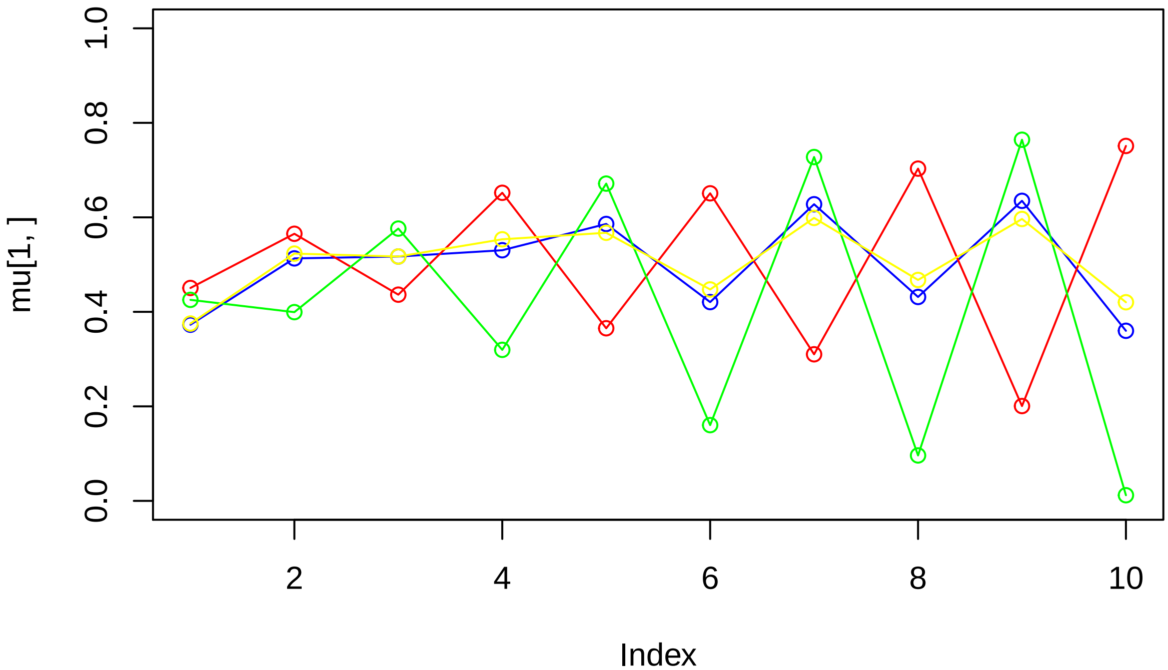

```
## iteration:  16 log likelihood:  -522.7751
```



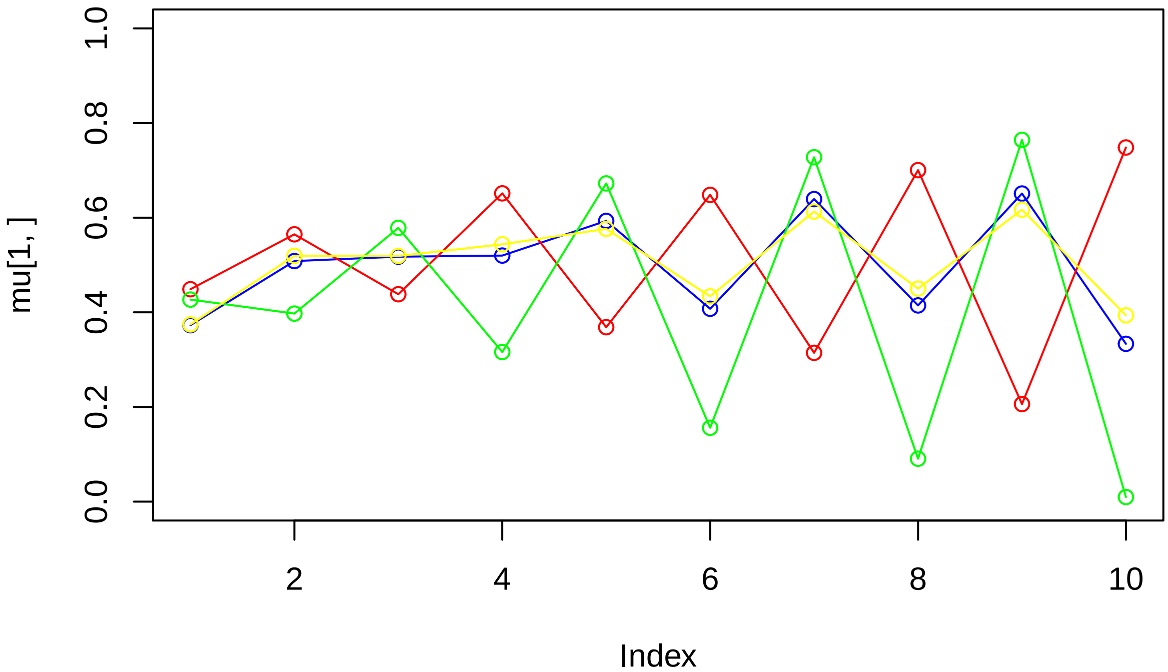

```
## iteration:  17 log likelihood:  -521.3929
```



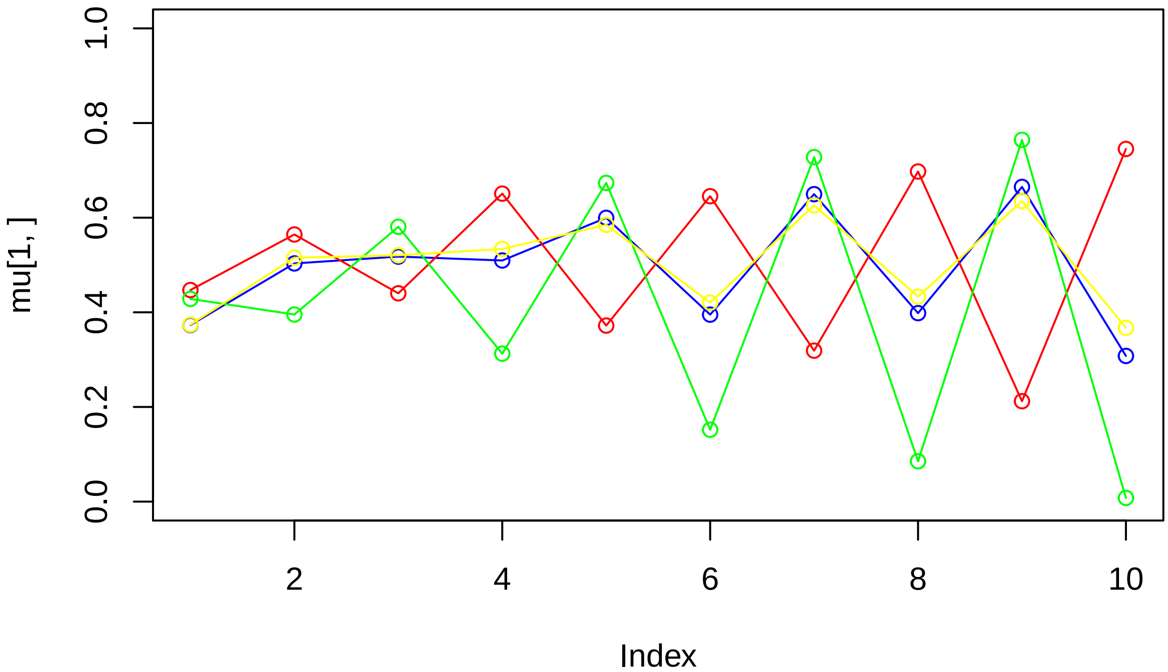

```
## iteration:  18 log likelihood:  -520.6263
```



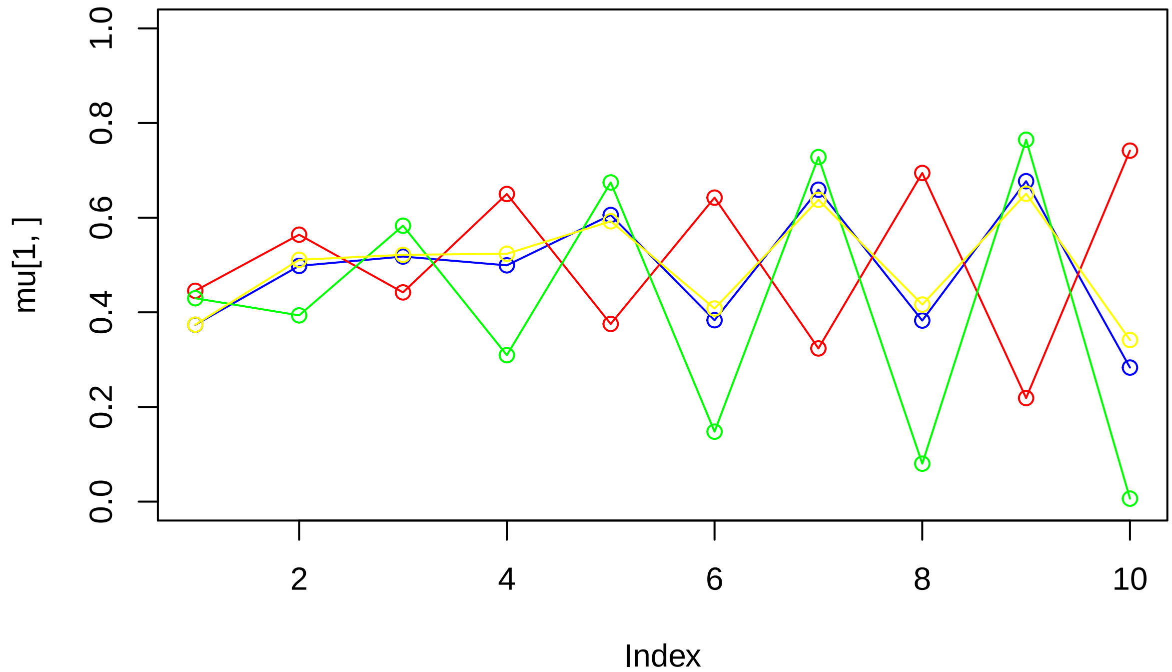

```
## iteration:  19 log likelihood:  -520.391
```



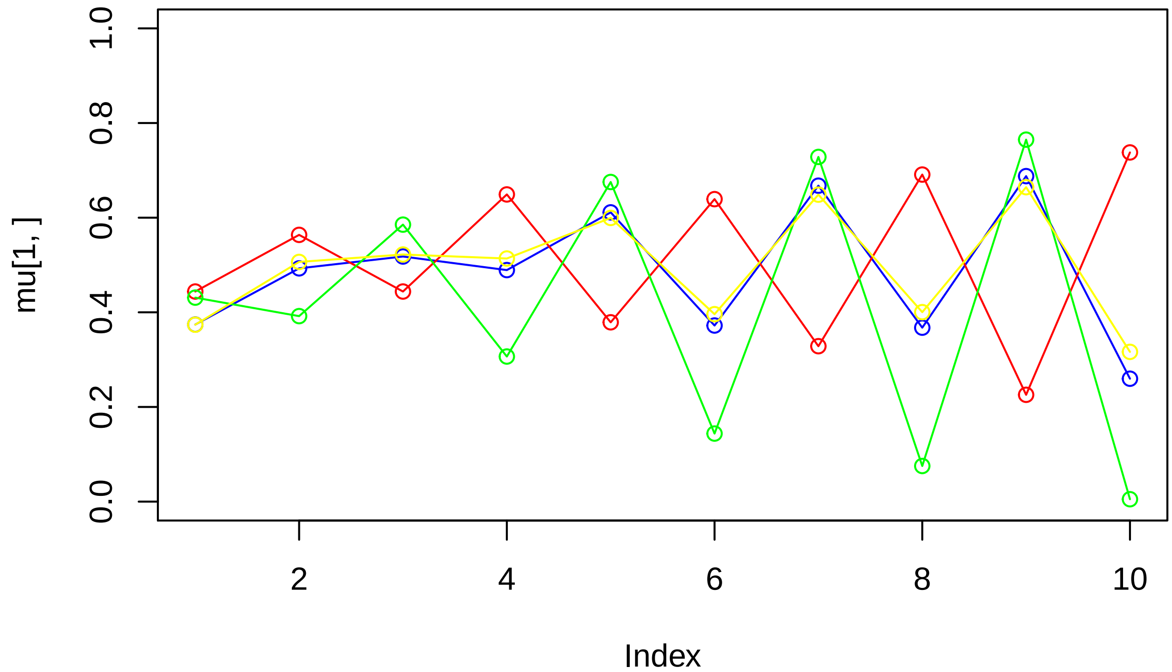

```
## iteration:  20 log likelihood:  -520.5983
```



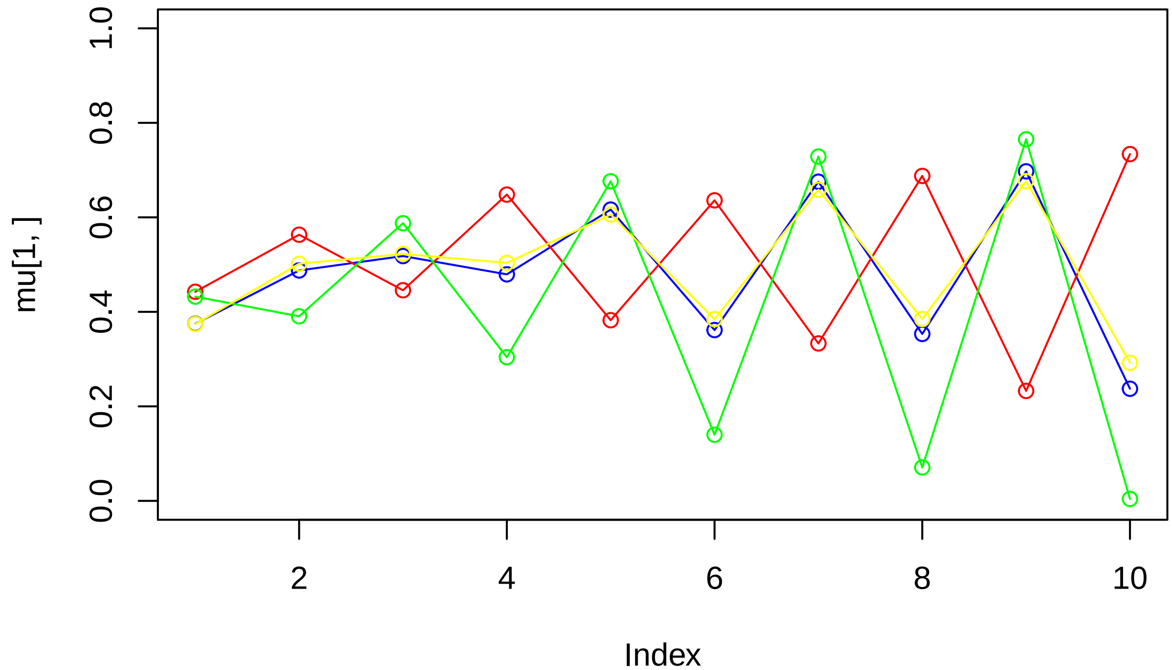

```
## iteration:  21 log likelihood:  -521.1652
```



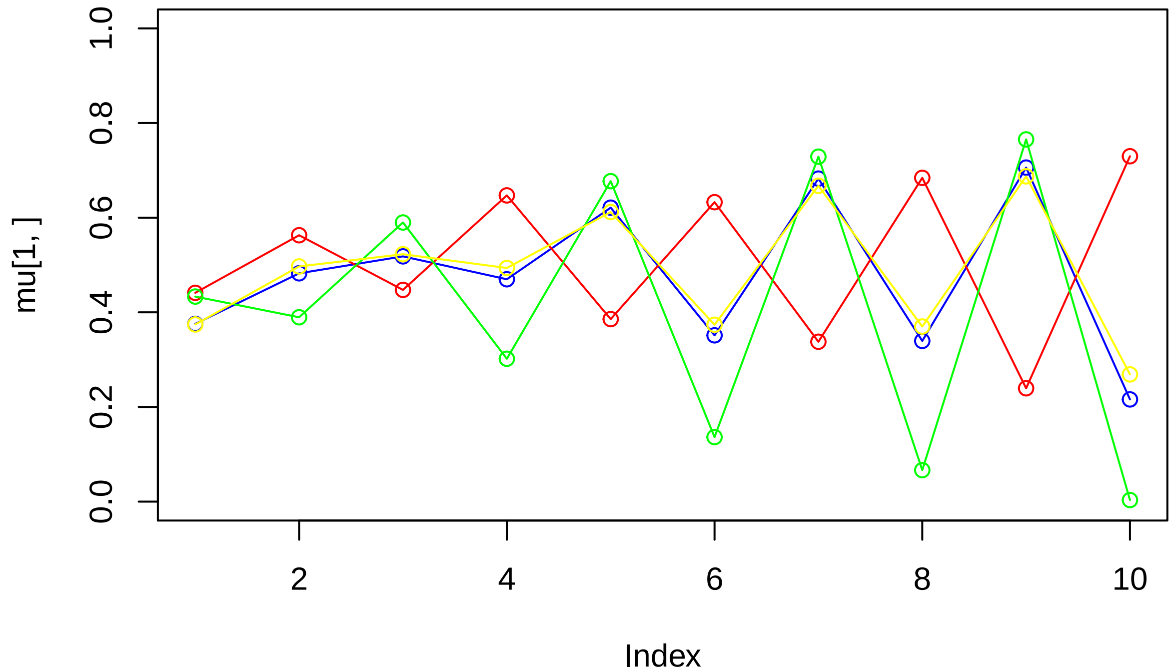

```
## iteration:  22 log likelihood:  -522.0204
```



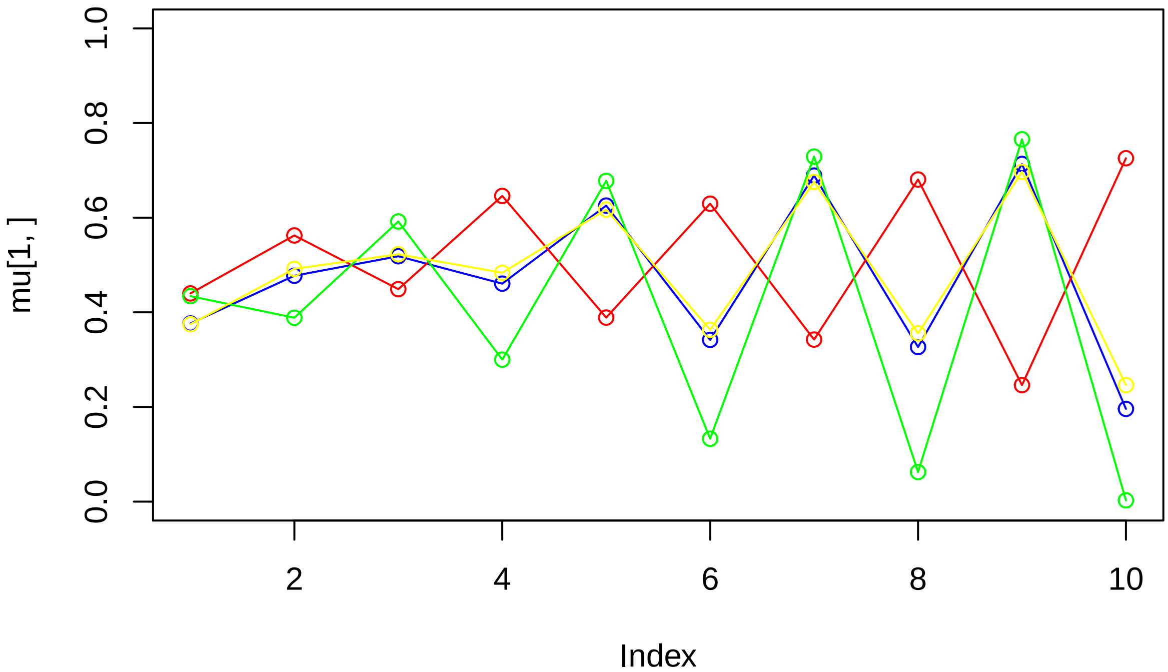

```
## iteration:  23 log likelihood:  -523.1059
```



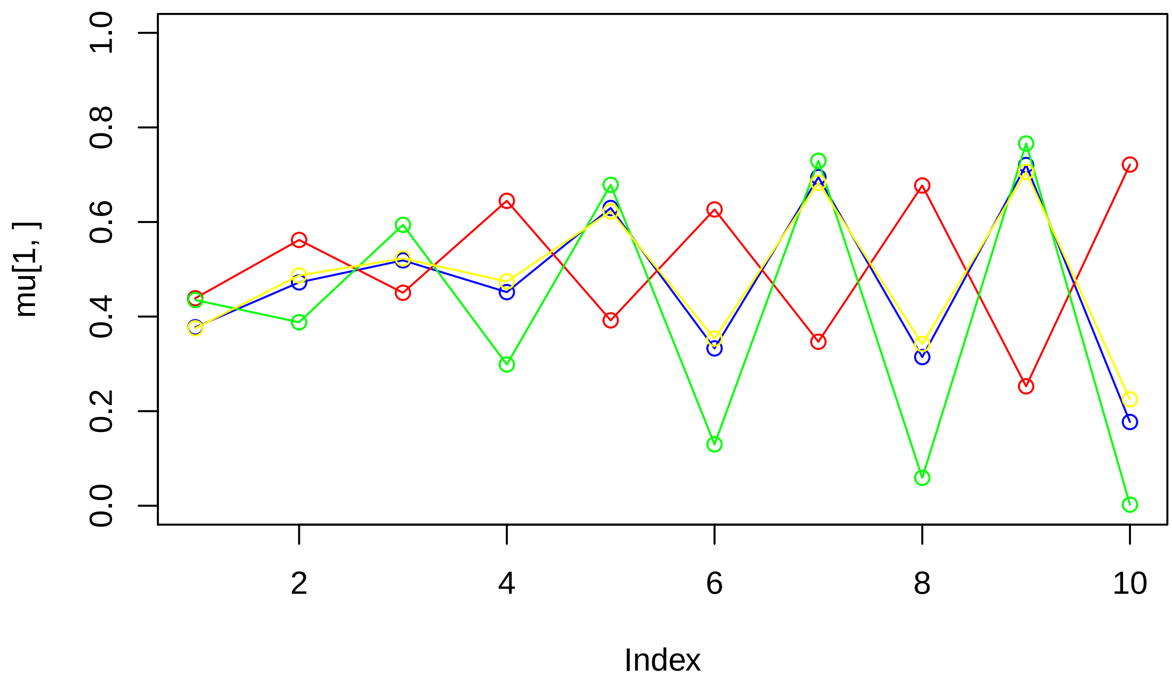

```
## iteration:  24 log likelihood:  -524.3754
```



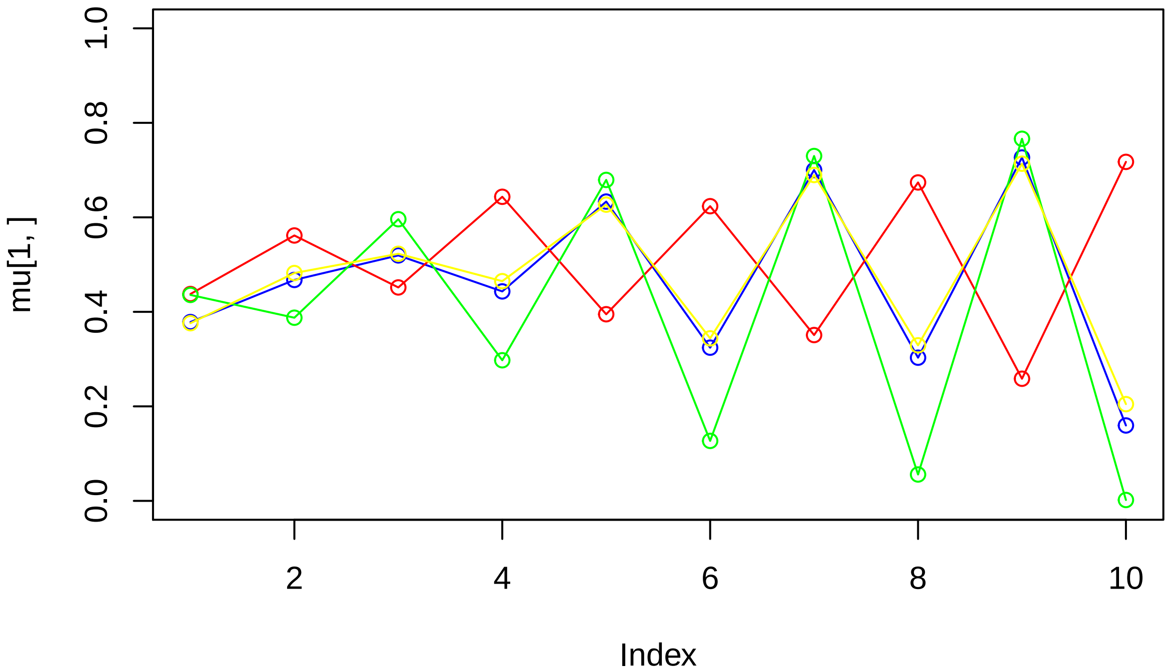

```
## iteration:  25 log likelihood:  -525.7912
```



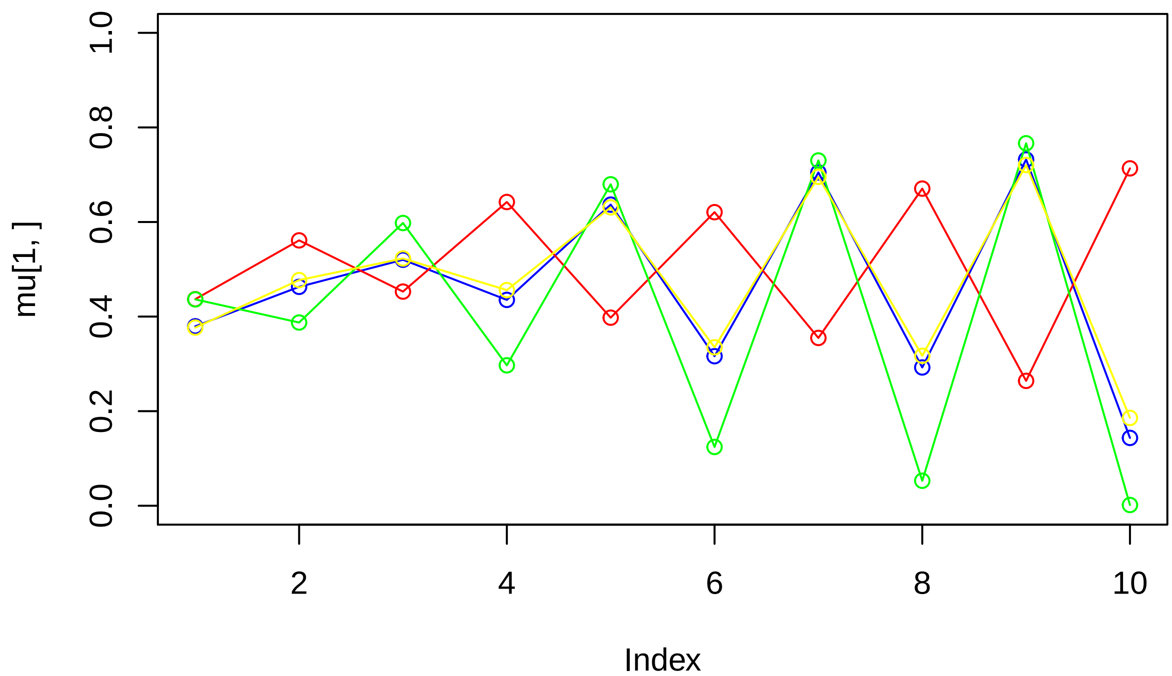

```
## iteration:  26 log likelihood:  -527.3207
```



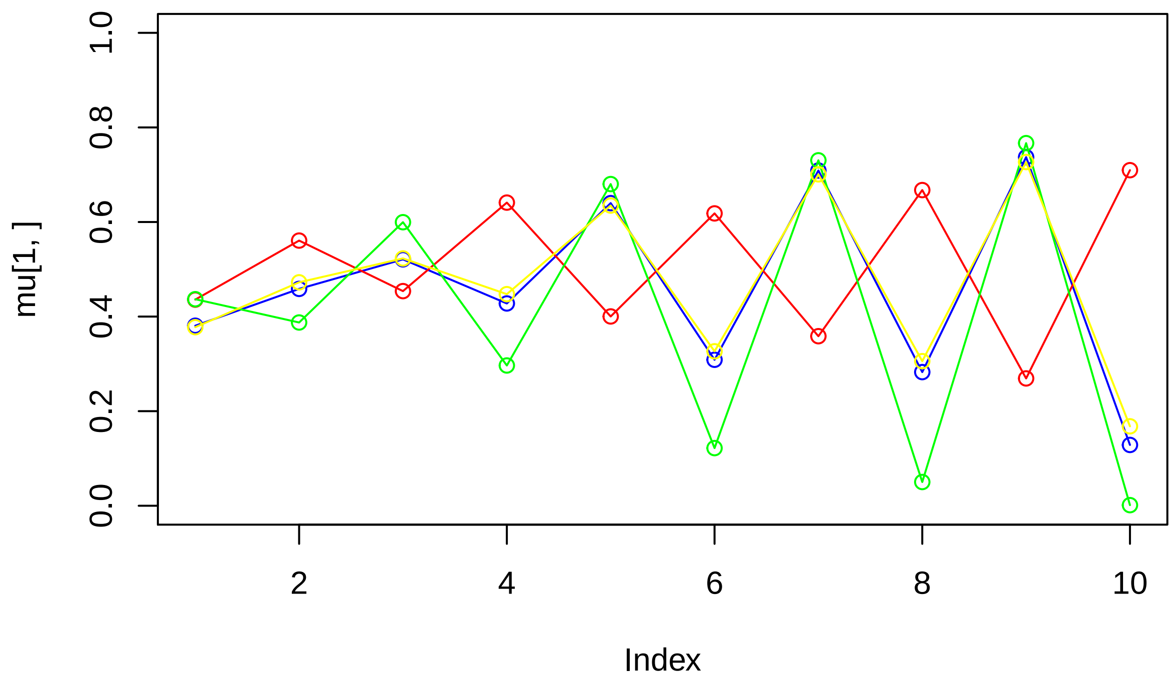

```
## iteration:  27 log likelihood:  -528.9346
```



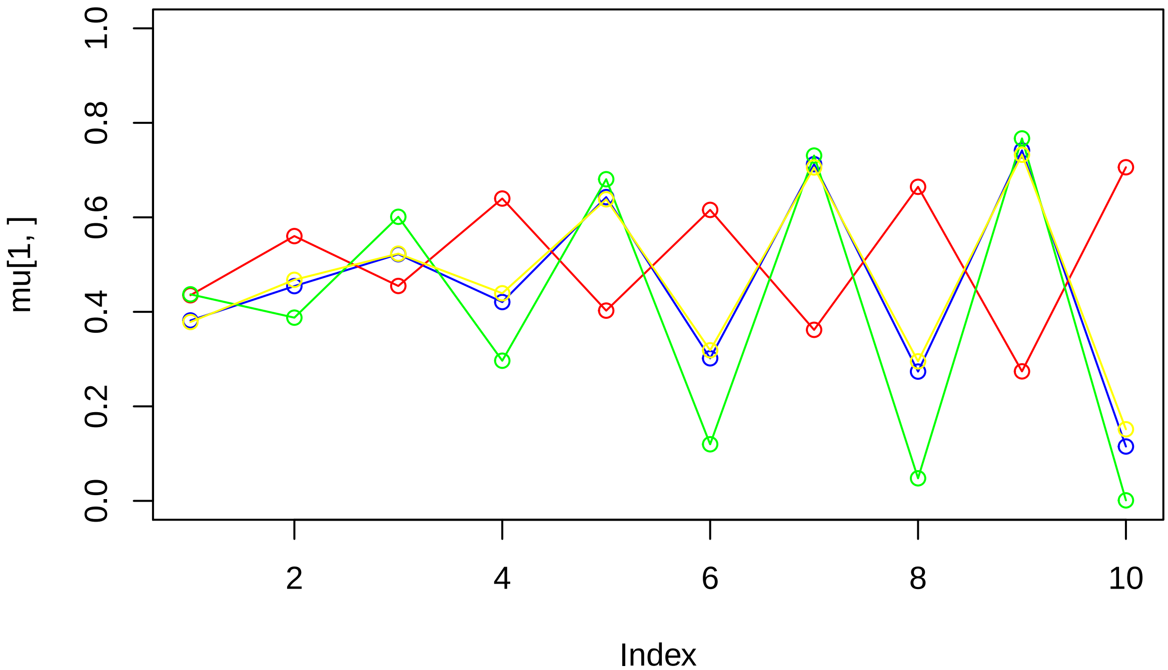

```
## iteration:  28 log likelihood:  -530.6046
```



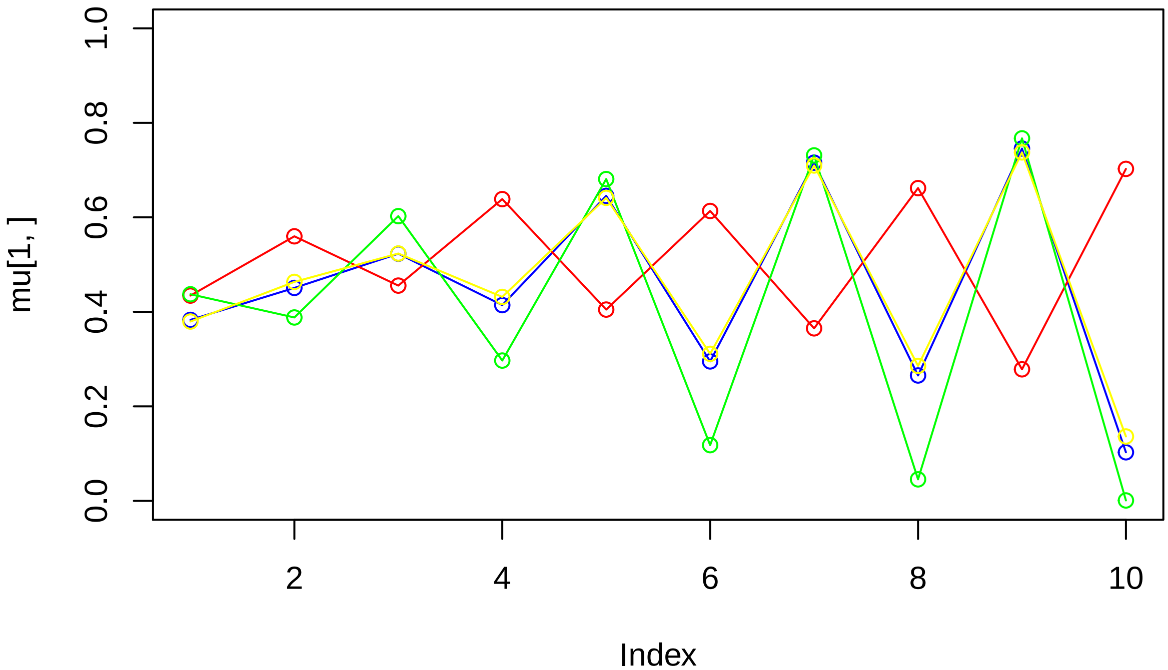

```
## iteration:  29 log likelihood:  -532.304
```



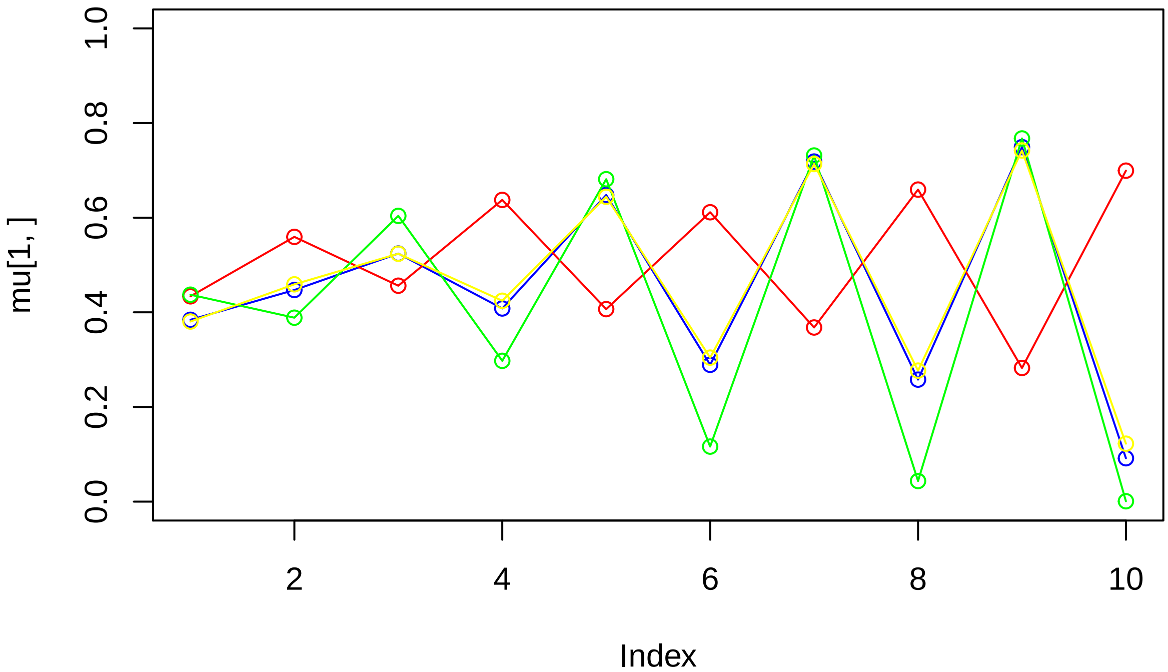

```
## iteration:  30 log likelihood:  -534.0069
```



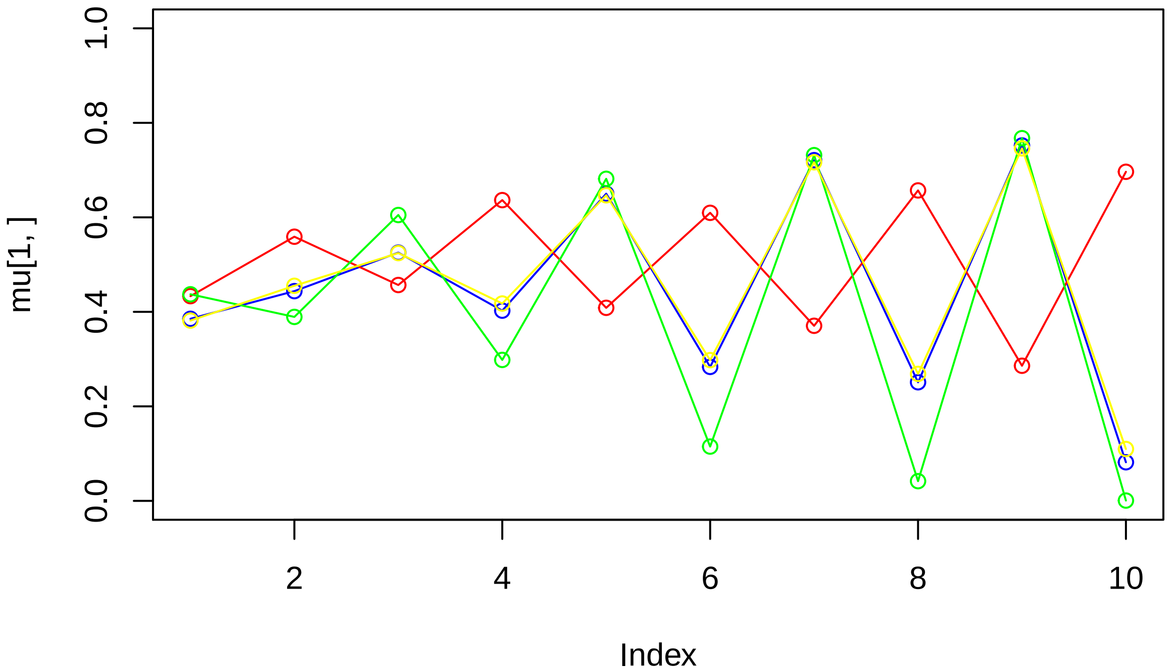

```
## iteration:  31 log likelihood:  -535.6895
```



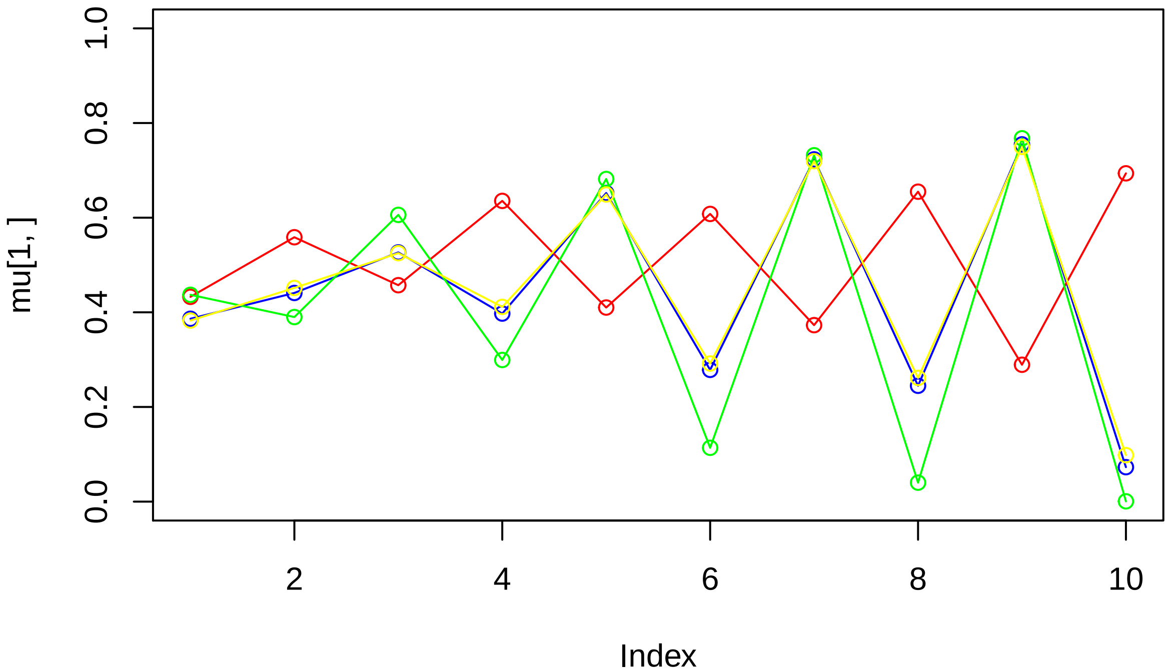

```
## iteration:  32 log likelihood:  -537.3305
```



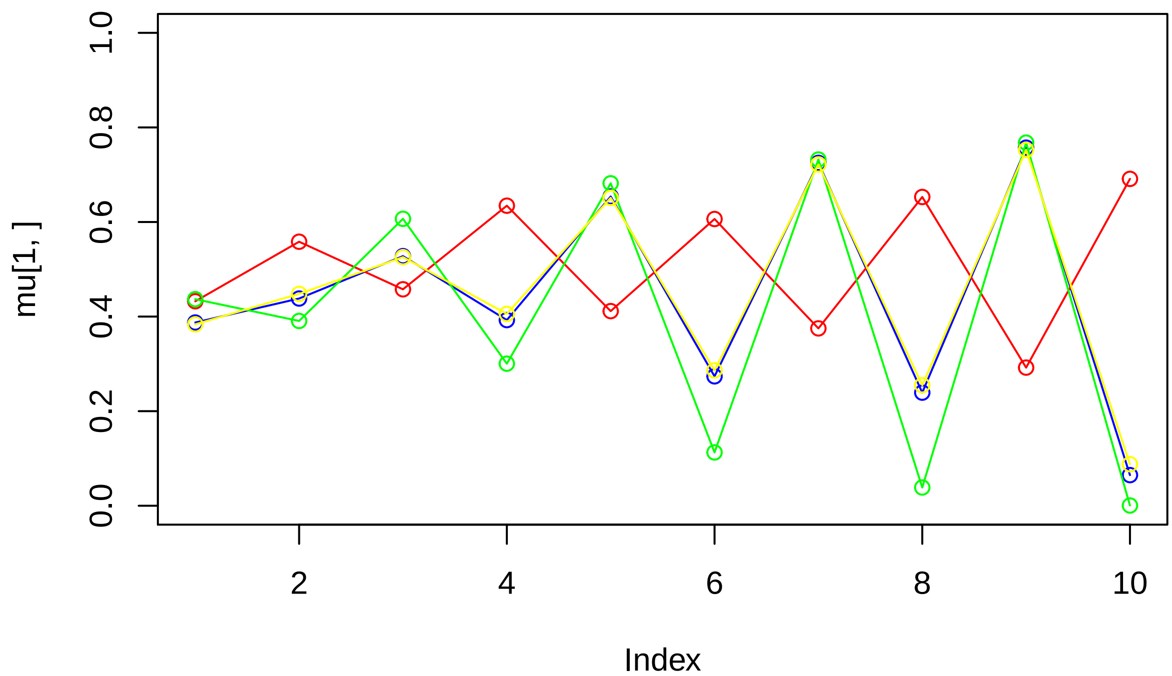

```
## iteration:  33 log likelihood:  -538.912
```



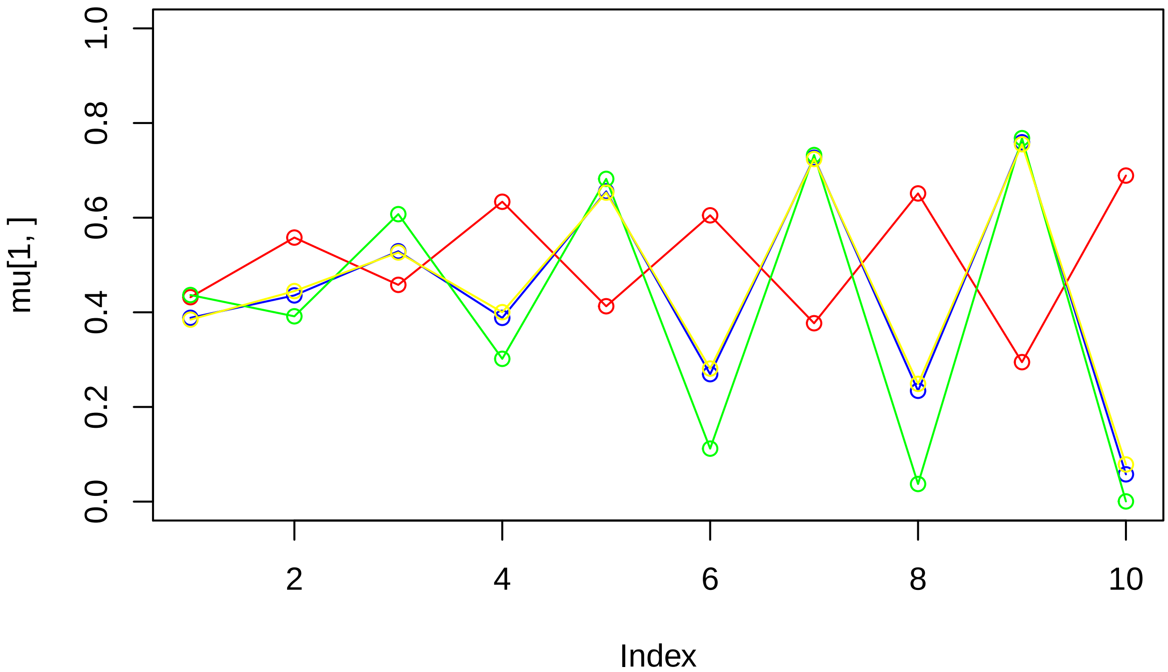

```
## iteration:  34 log likelihood:  -540.4198
```



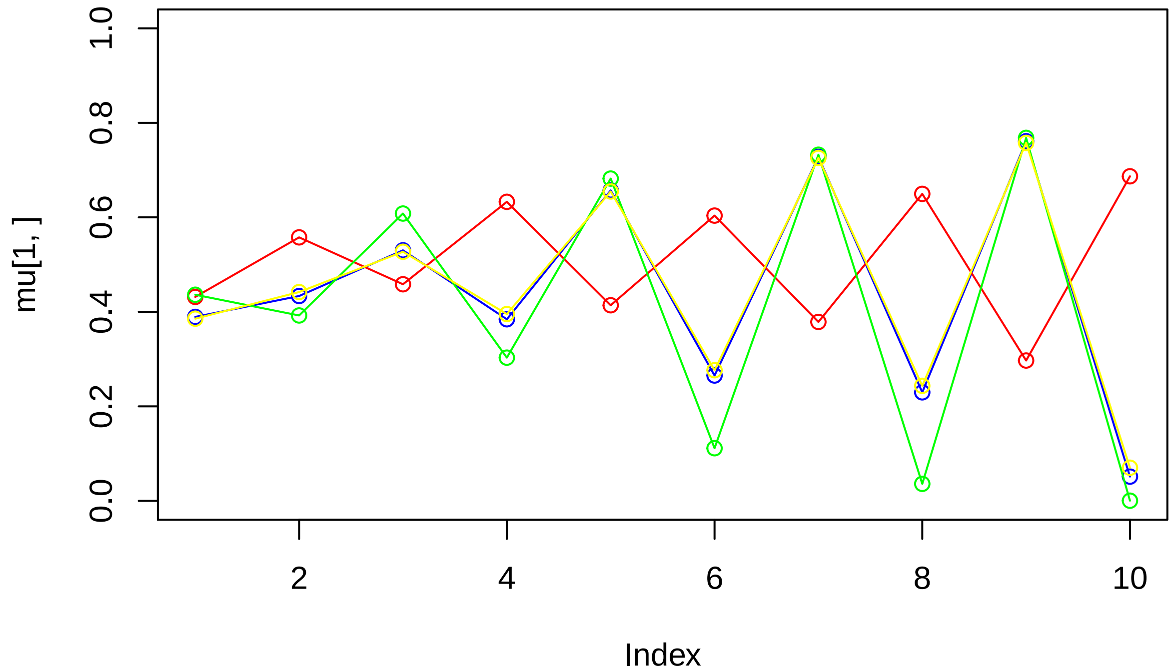

```
## iteration:  35 log likelihood:  -541.8433
```



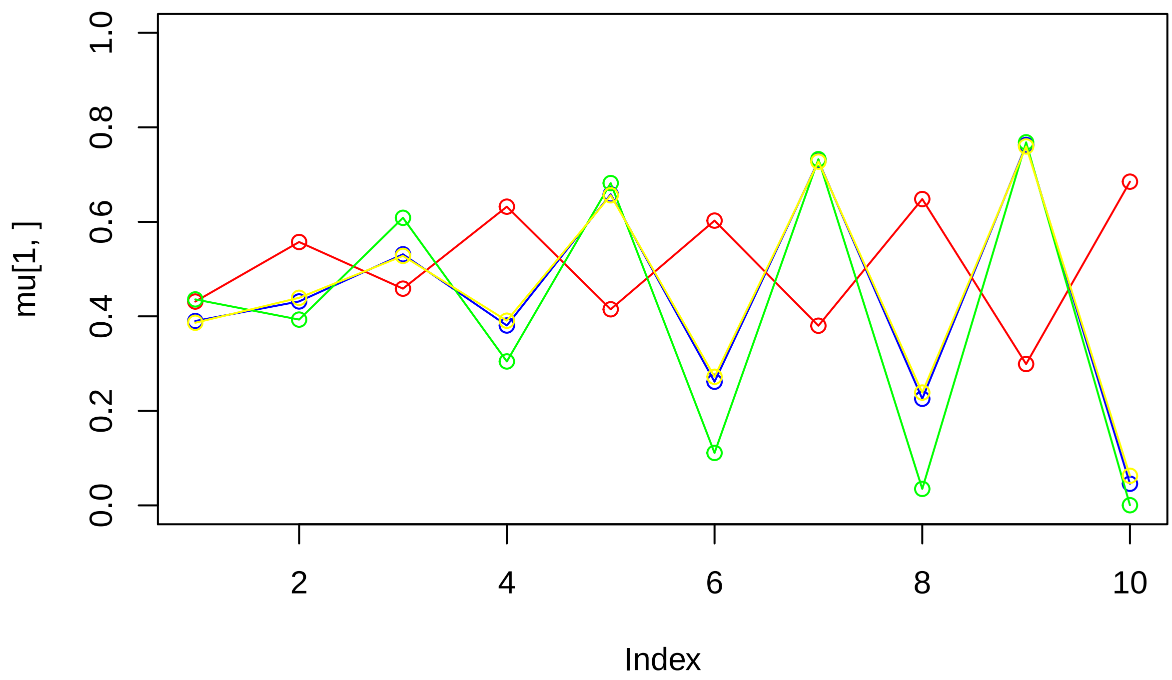

```
## iteration:  36 log likelihood:  -543.1756
```



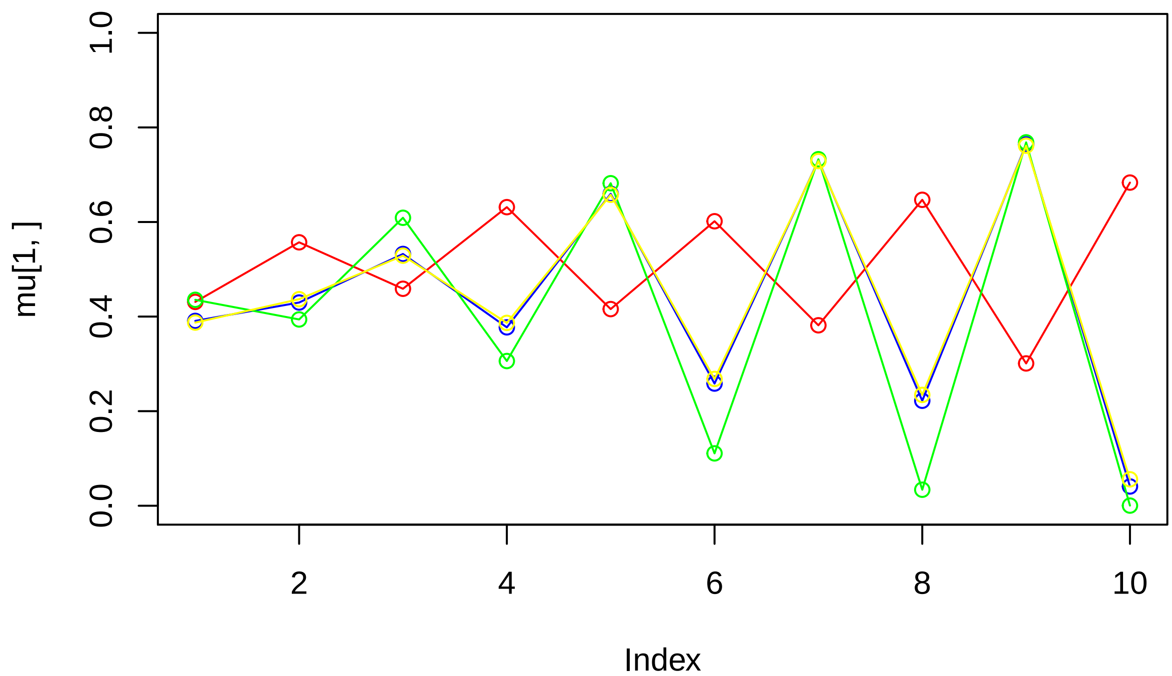

```
## iteration:  37 log likelihood:  -544.4133
```



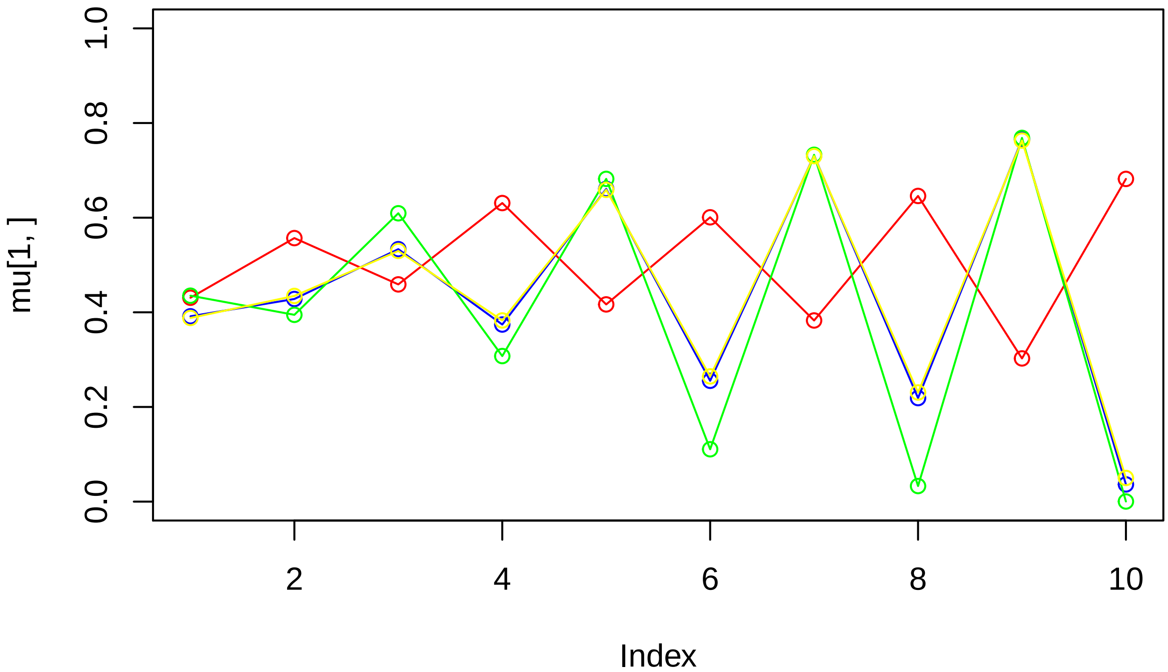

```
## iteration:  38 log likelihood:  -545.5555
```



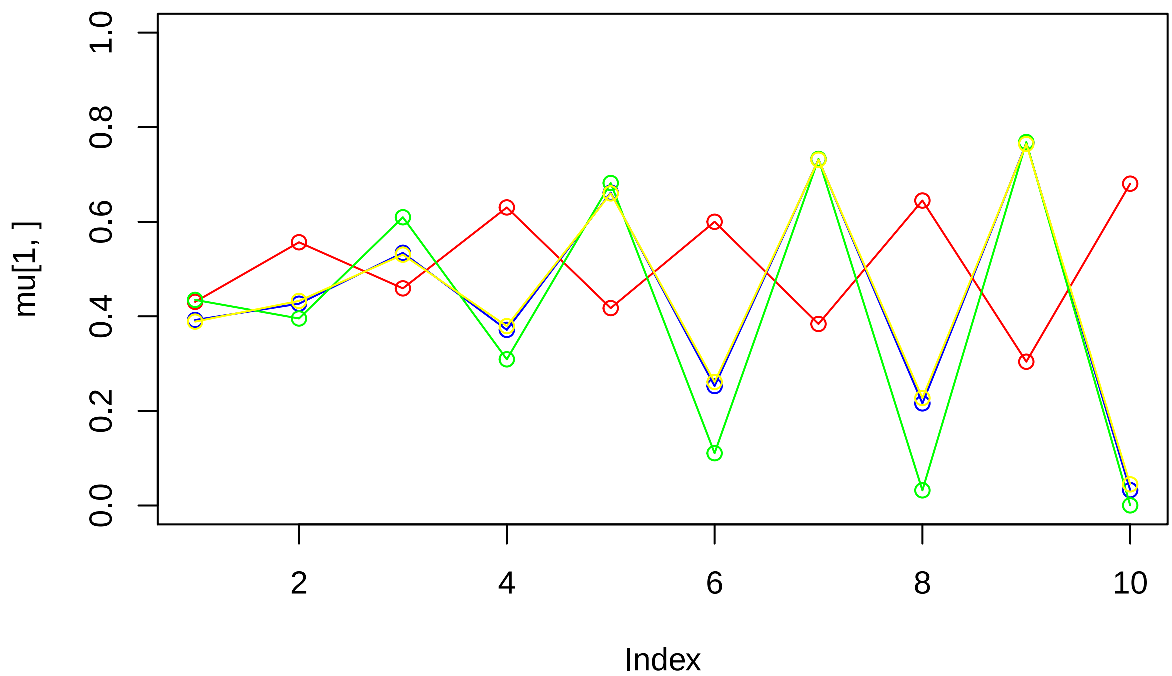

```
## iteration:  39 log likelihood:  -546.6036
```



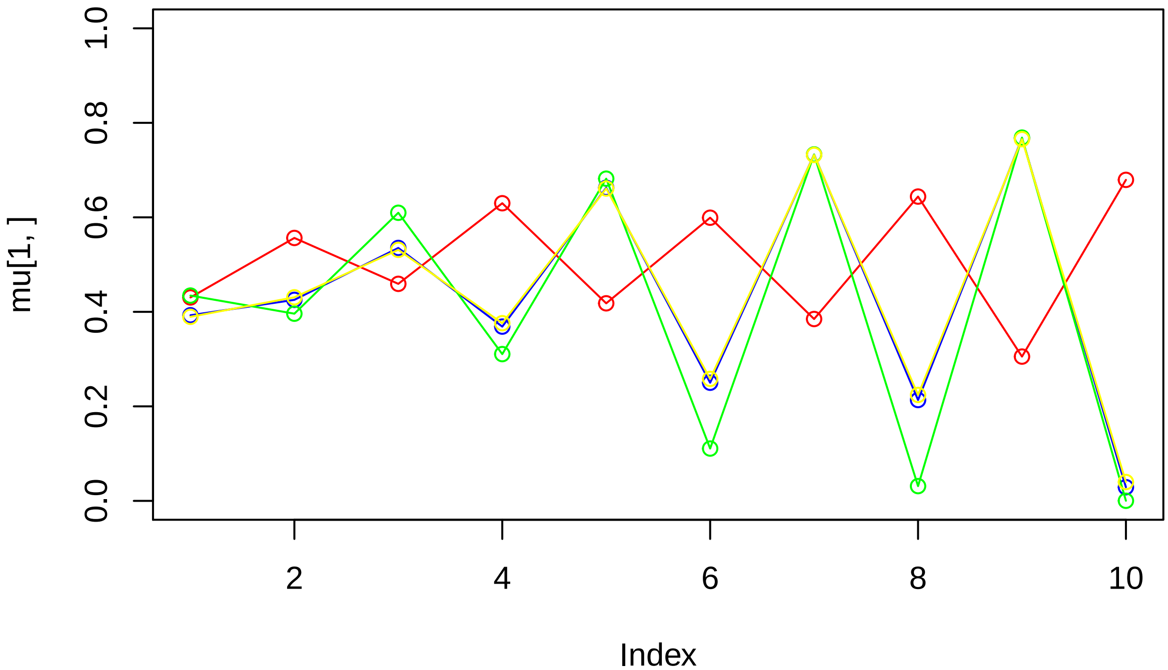

```
## iteration:  40 log likelihood:  -547.5609
```



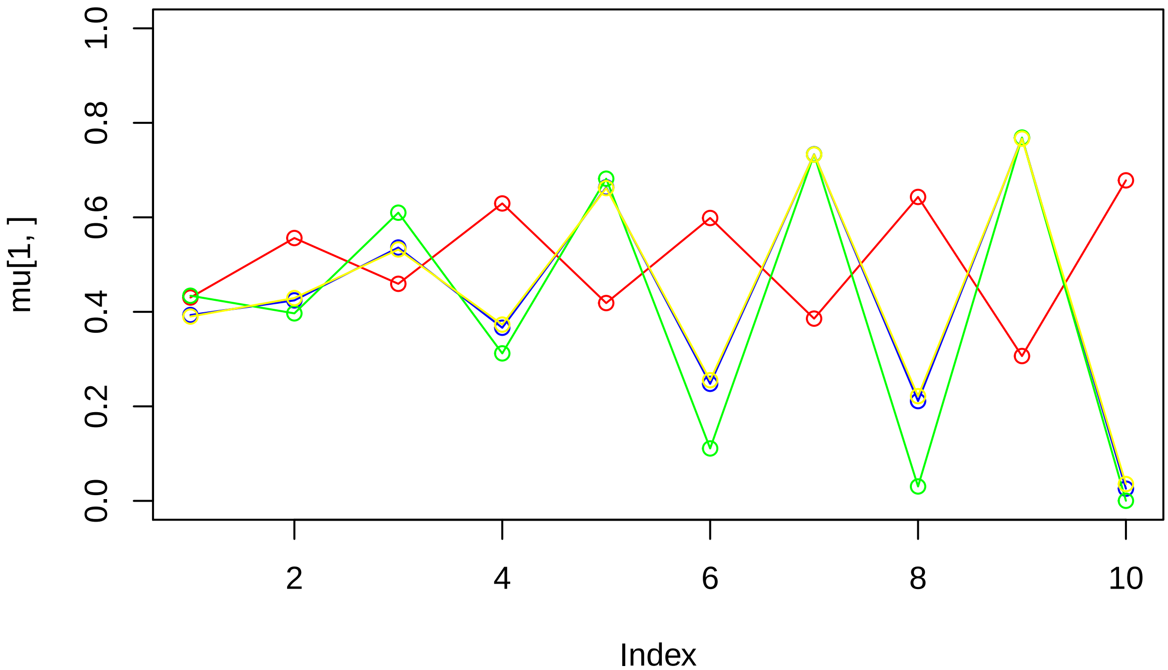

```
## iteration:  41 log likelihood:  -548.4315
```



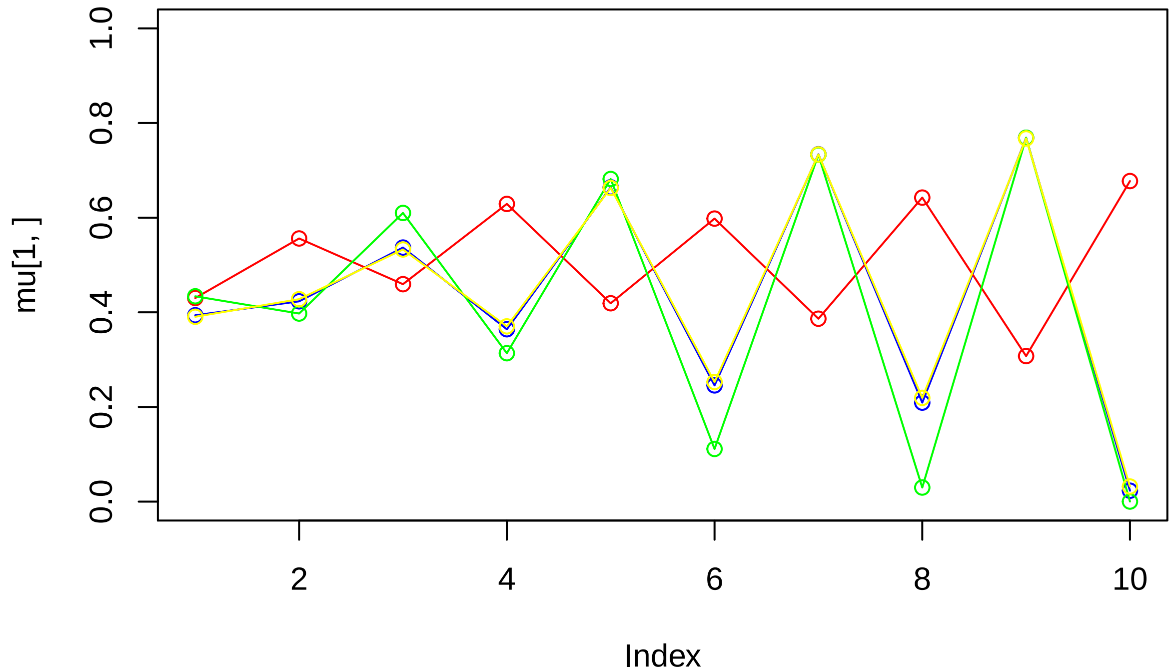

```
## iteration:  42 log likelihood:  -549.2208
```



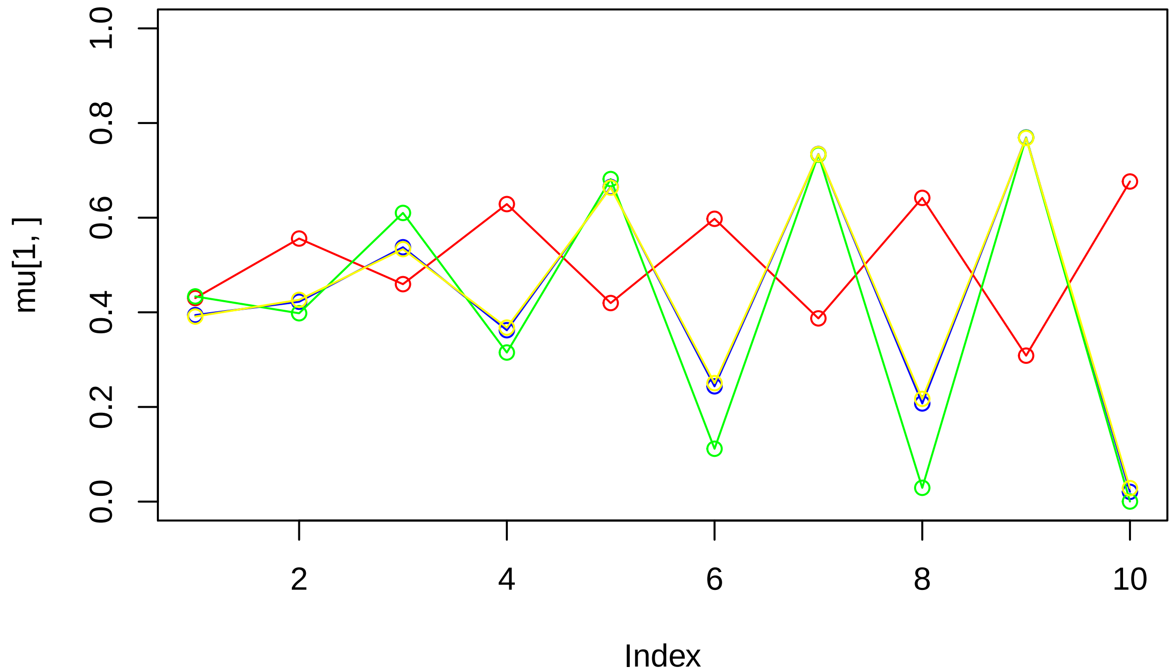

```
## iteration:  43 log likelihood:  -549.9344
```



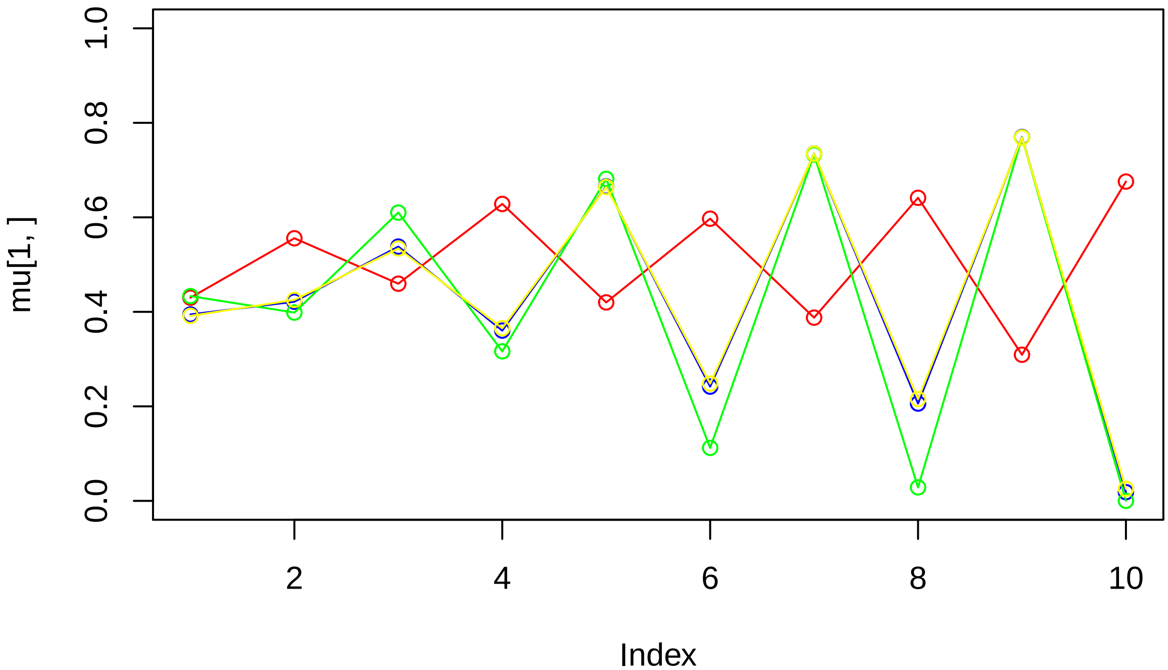

```
## iteration:  44 log likelihood:  -550.5781
```



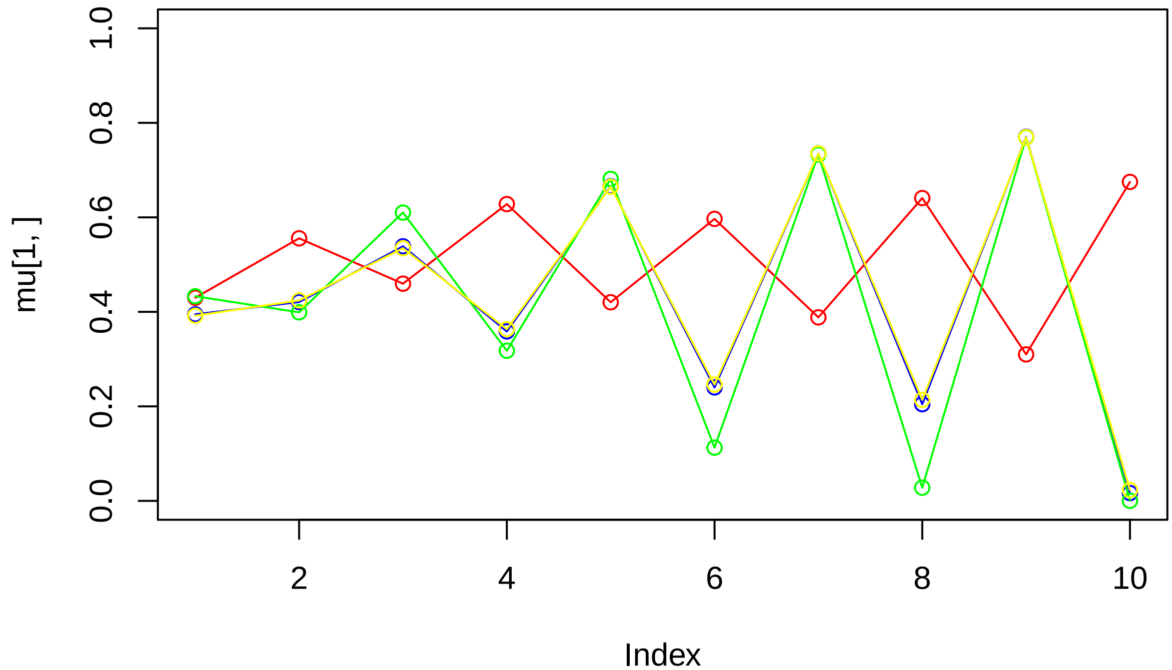

```
## iteration:  45 log likelihood:  -551.1577
```



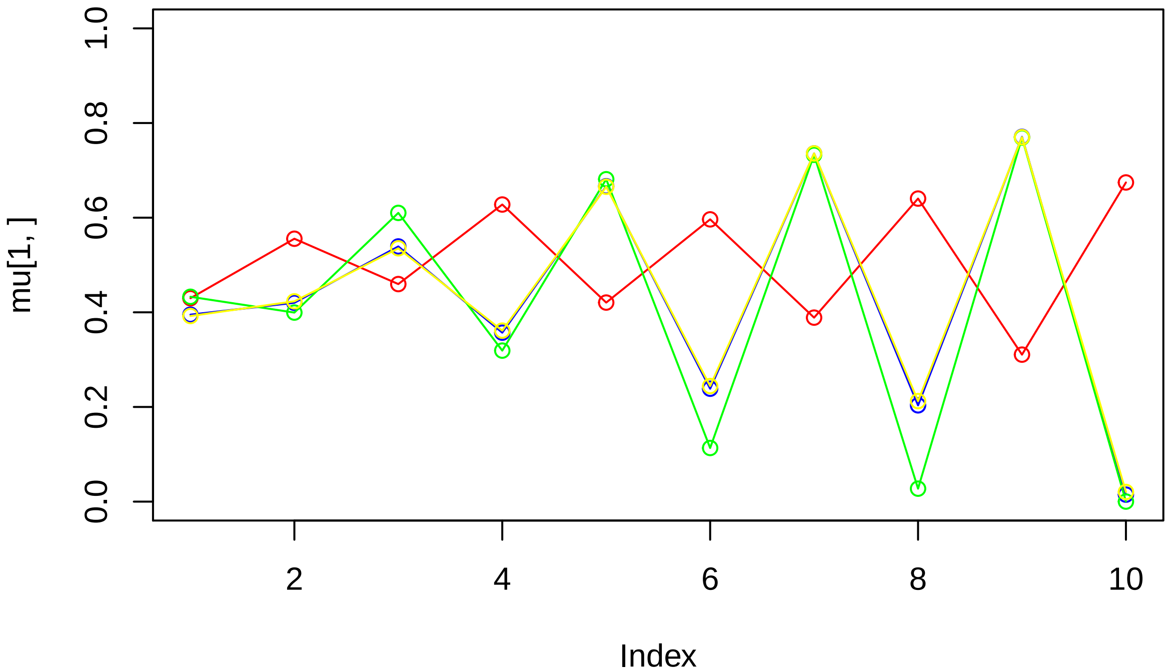

```
## iteration:  46 log likelihood:  -551.6789
```



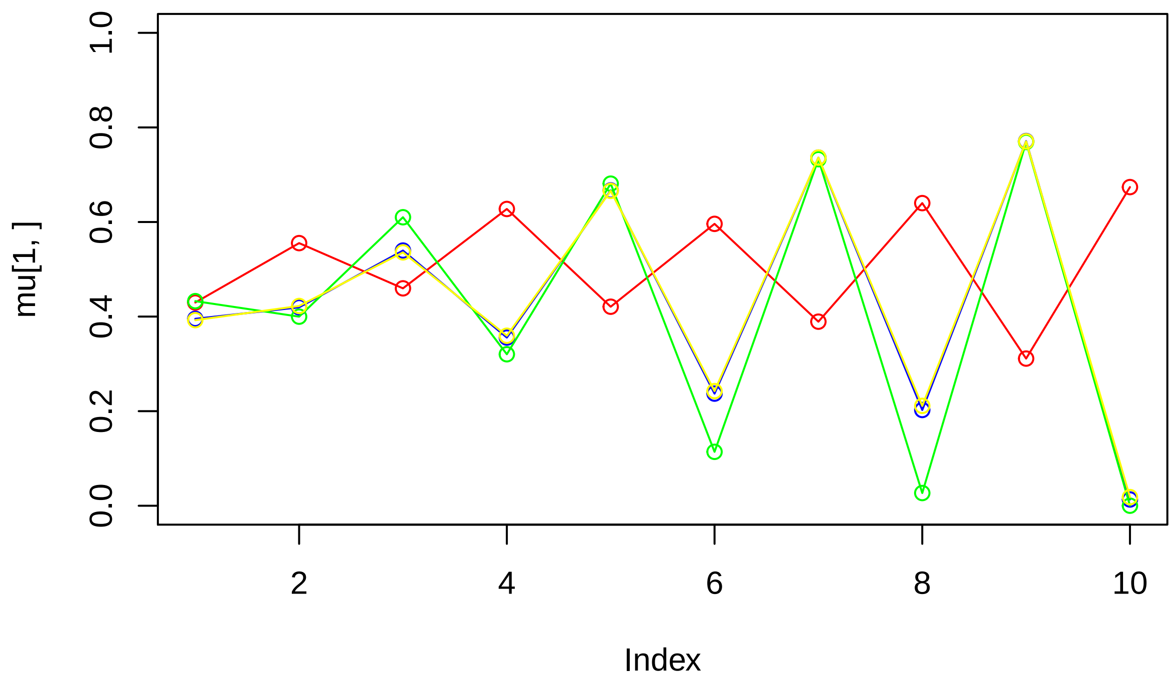

```
## iteration:  47 log likelihood:  -552.1471
```



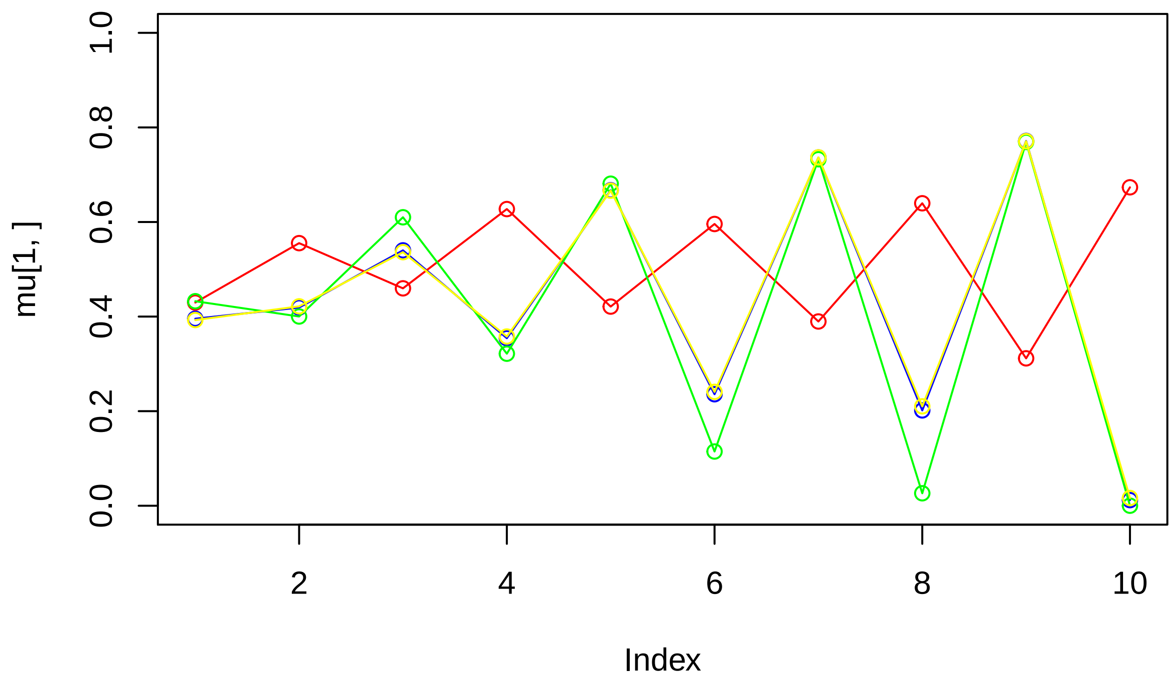

```
## iteration:  48 log likelihood:  -552.5674
```



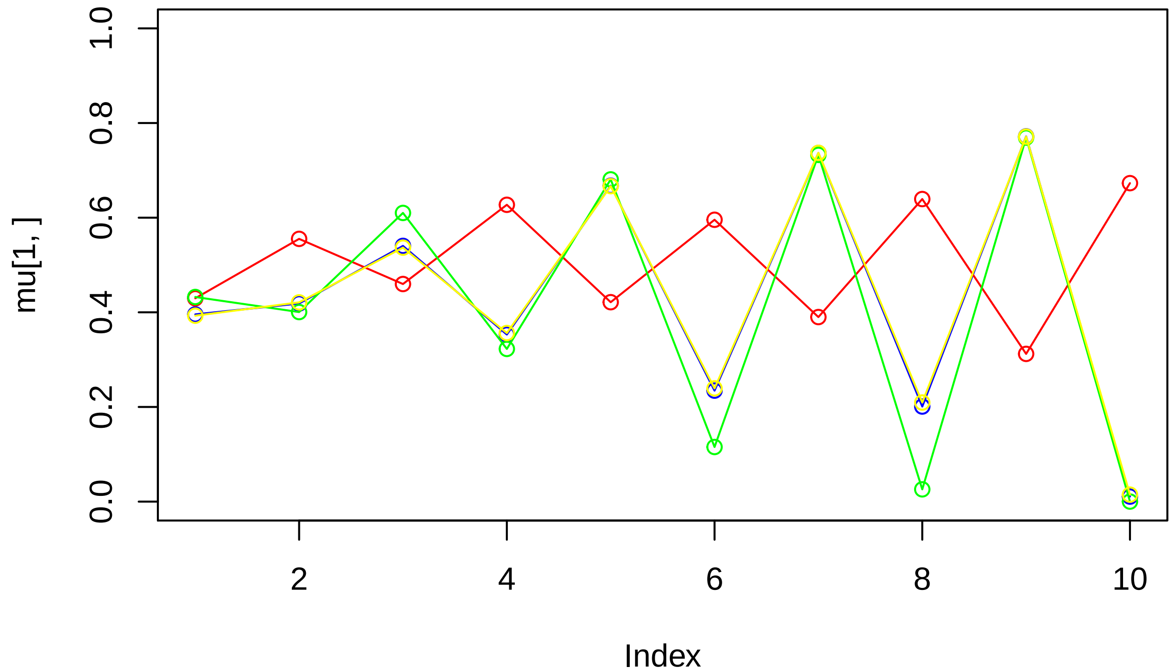

```
## iteration:  49 log likelihood:  -552.9443
```



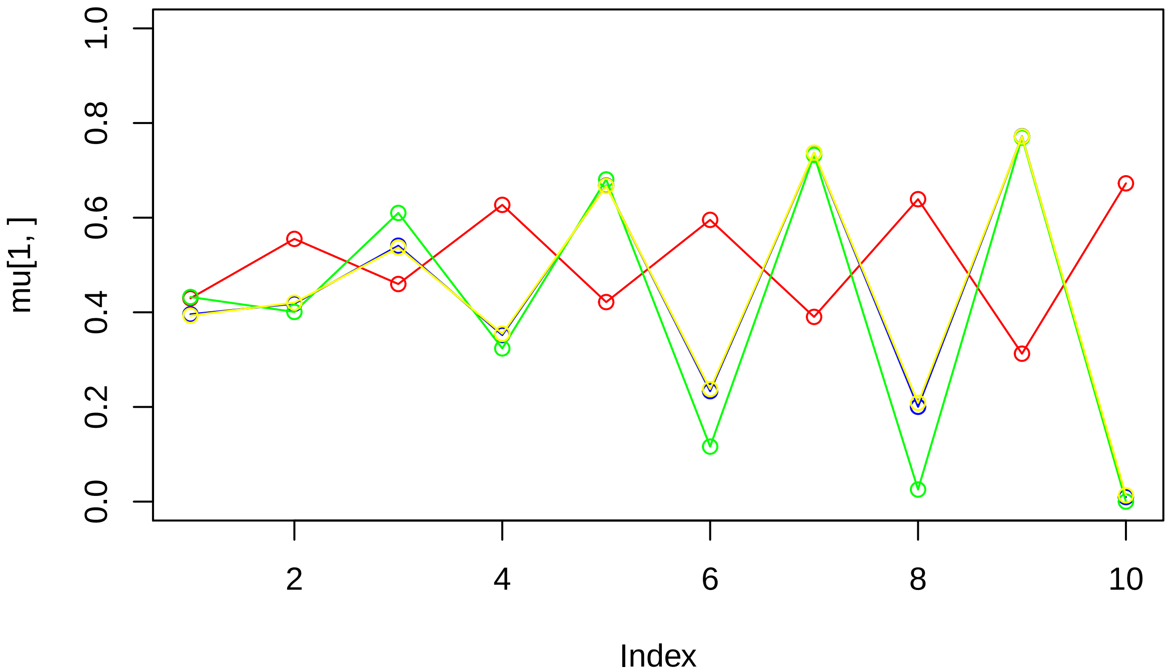

```
## iteration:  50 log likelihood:  -553.2824
```



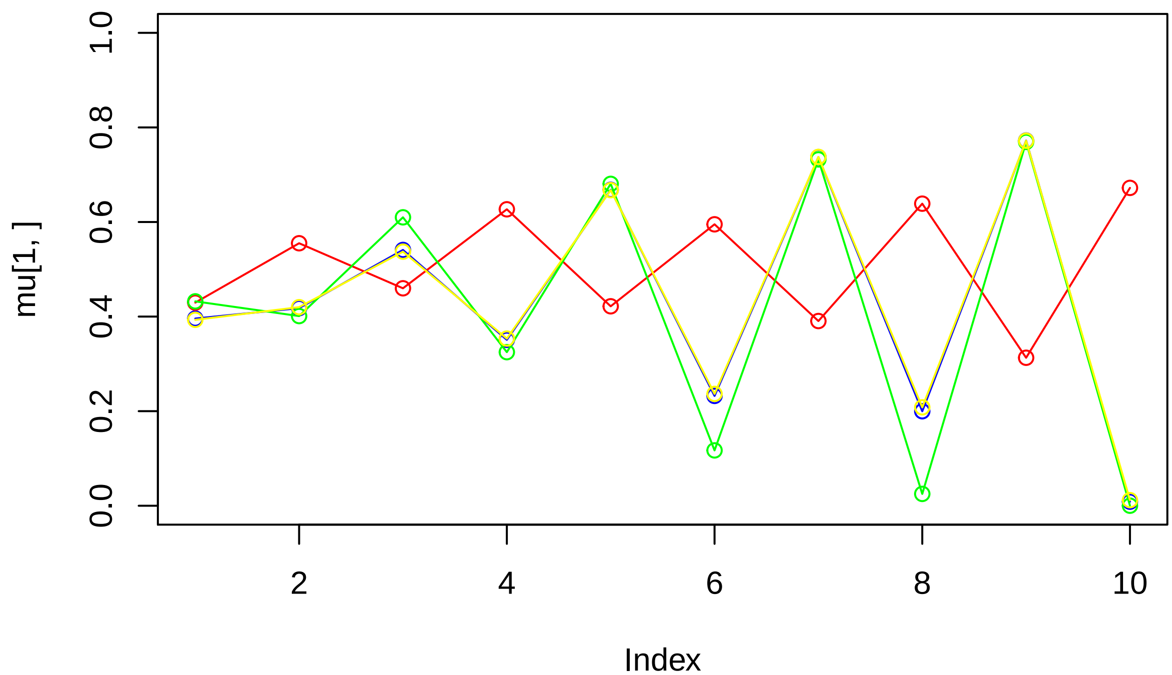

```
## iteration:  51 log likelihood:  -553.5855
```



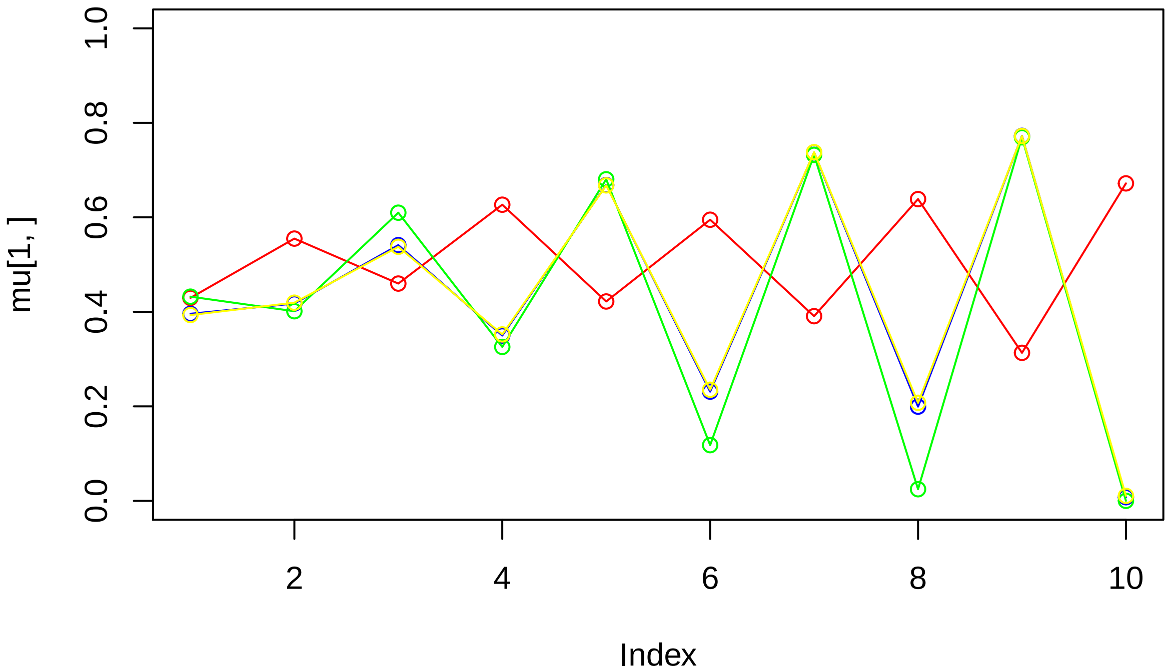

```
## iteration:  52 log likelihood:  -553.8573
```



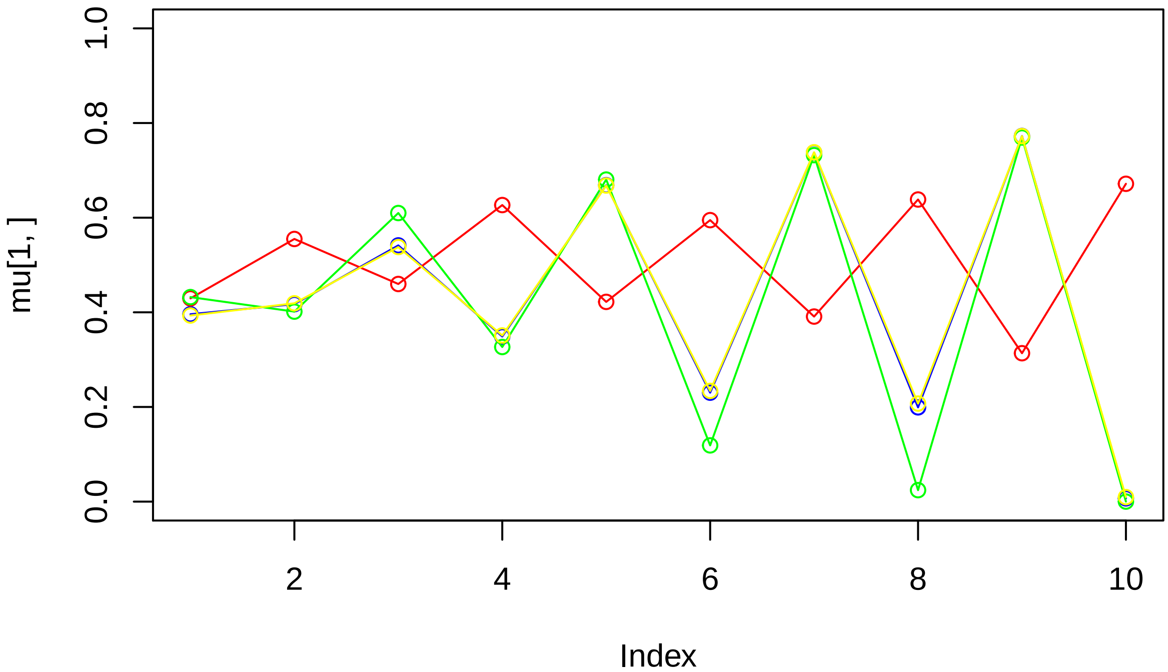

```
## iteration:  53 log likelihood:  -554.101
```



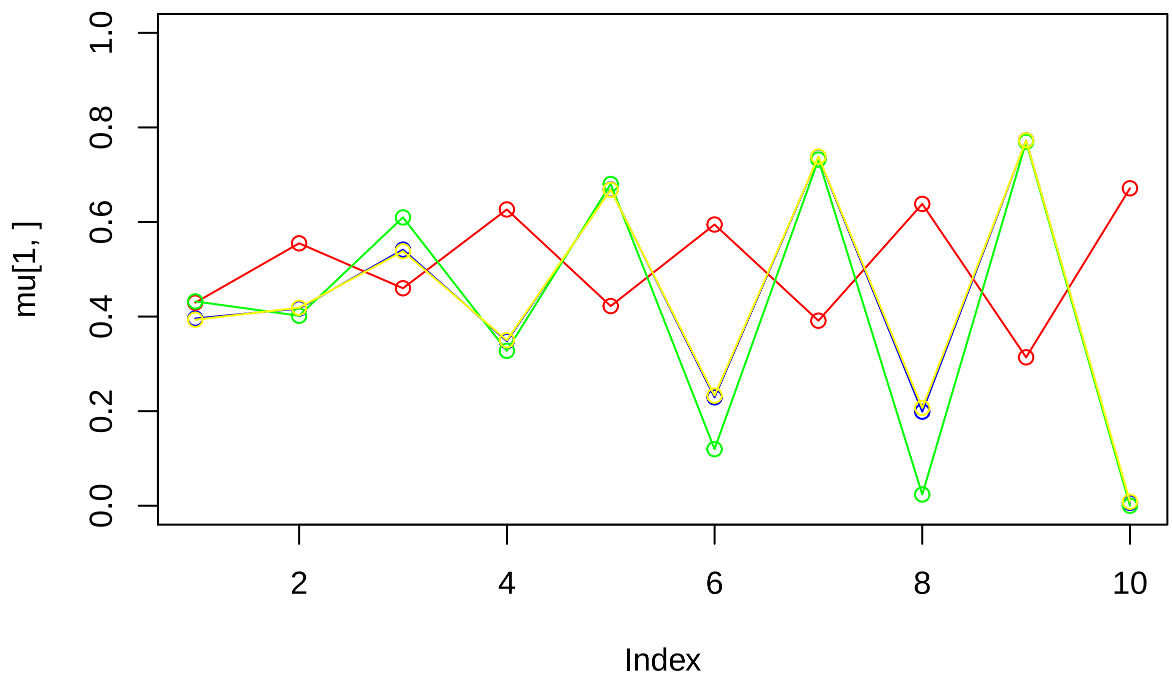

```
## iteration:  54 log likelihood:  -554.3194
```



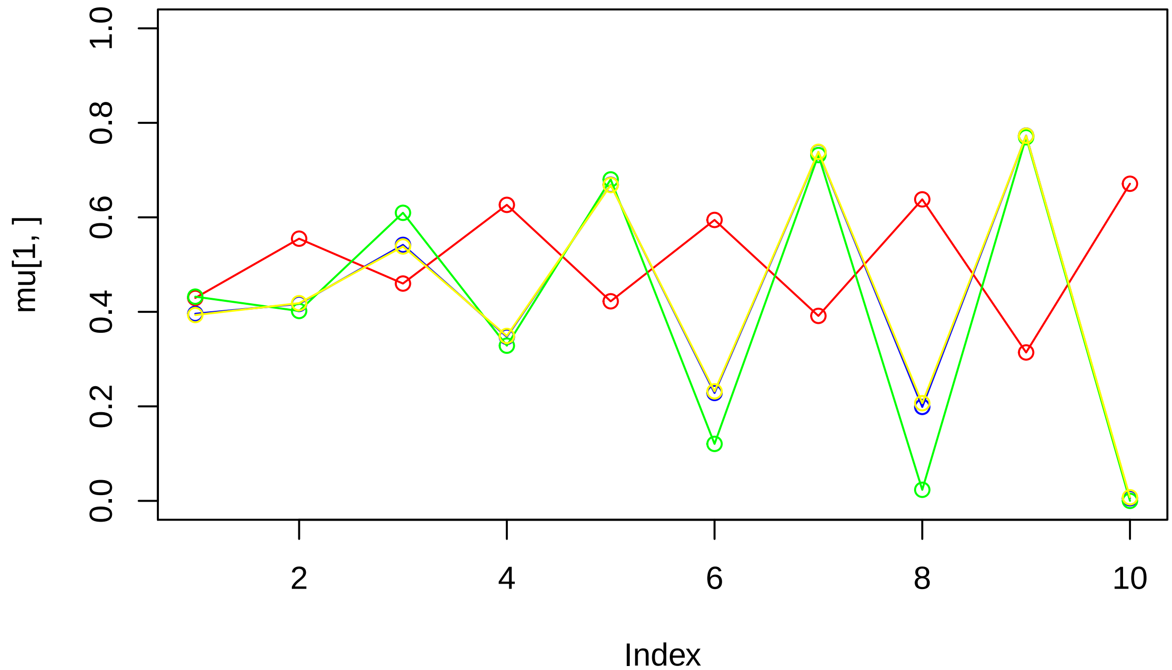

```
## iteration:  55 log likelihood:  -554.5153
```



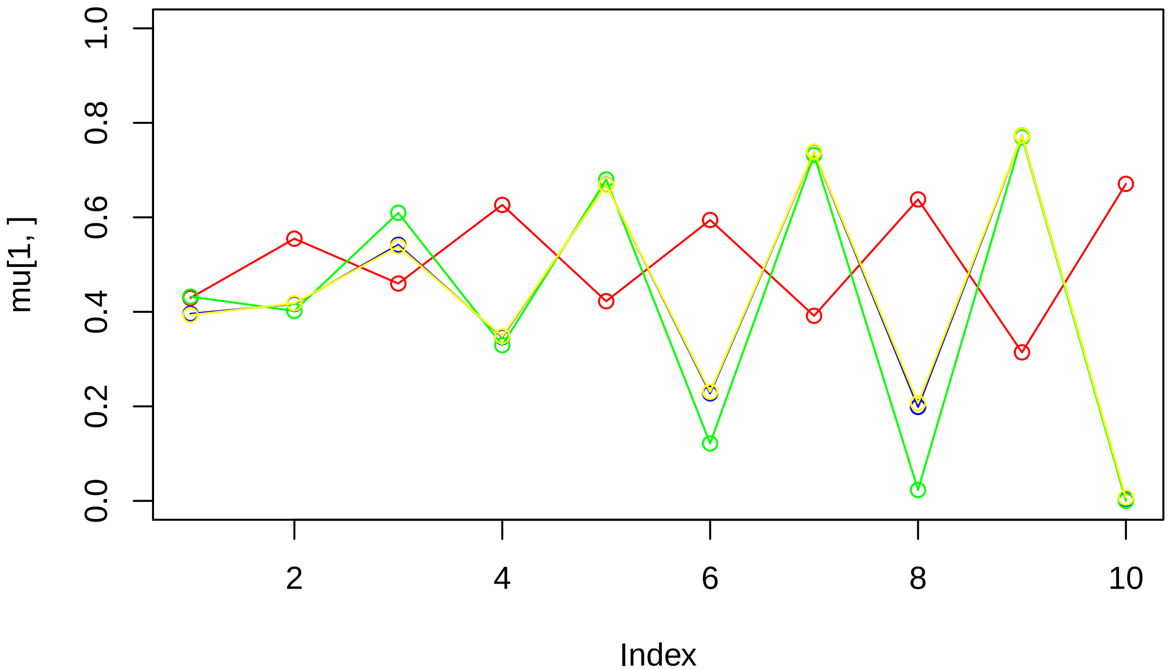

```
## iteration:  56 log likelihood:  -554.691
```



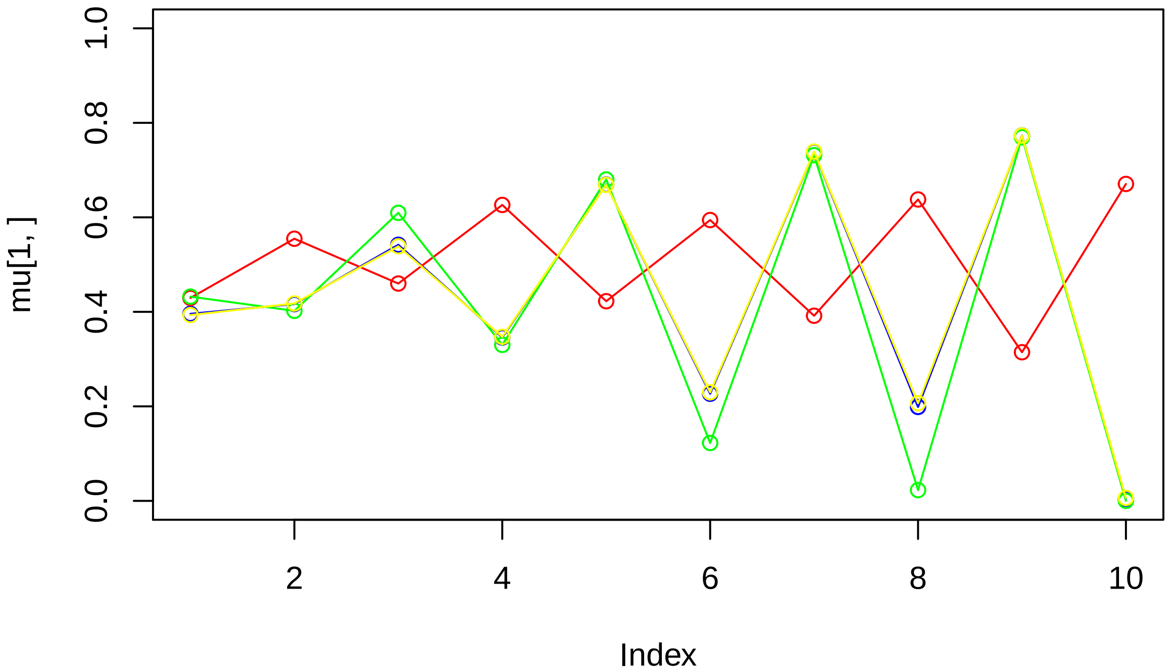

```
## iteration:  57 log likelihood:  -554.8485
```



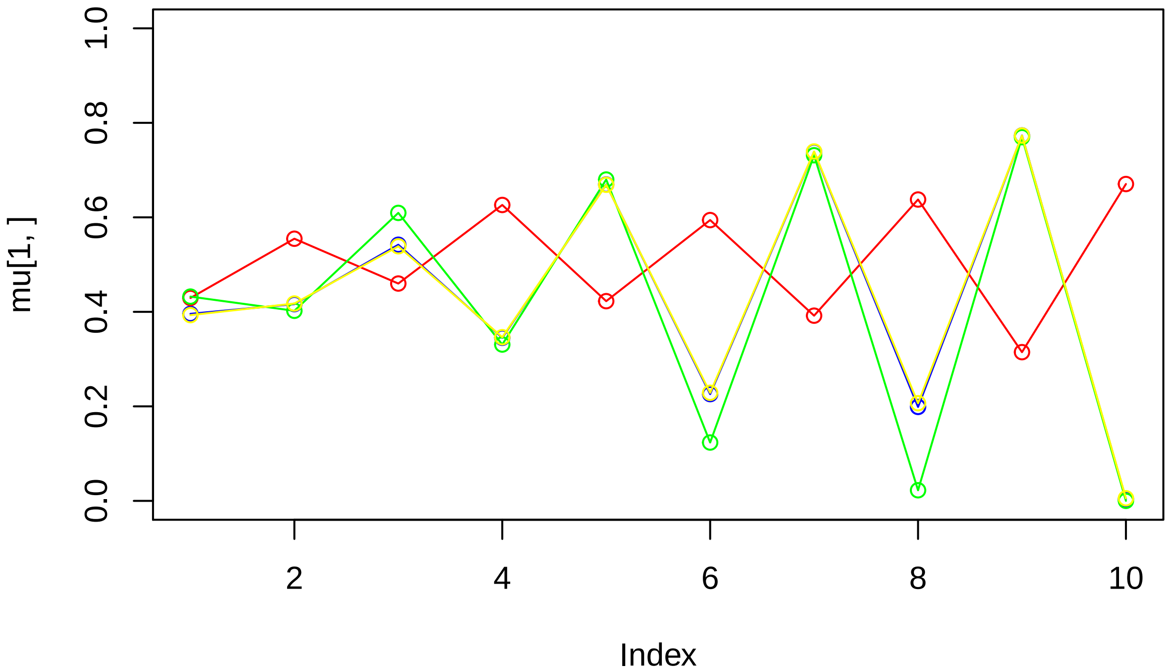

```
## iteration:  58 log likelihood:  -554.9898
```



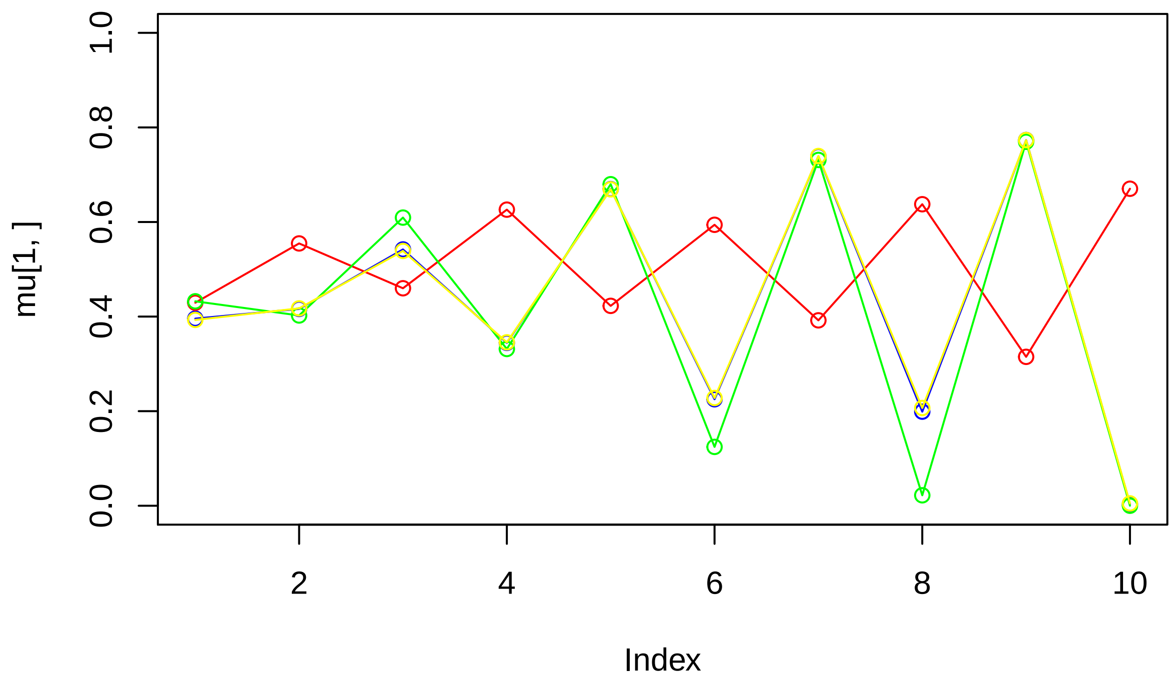

```
## iteration:  59 log likelihood:  -555.1165
```



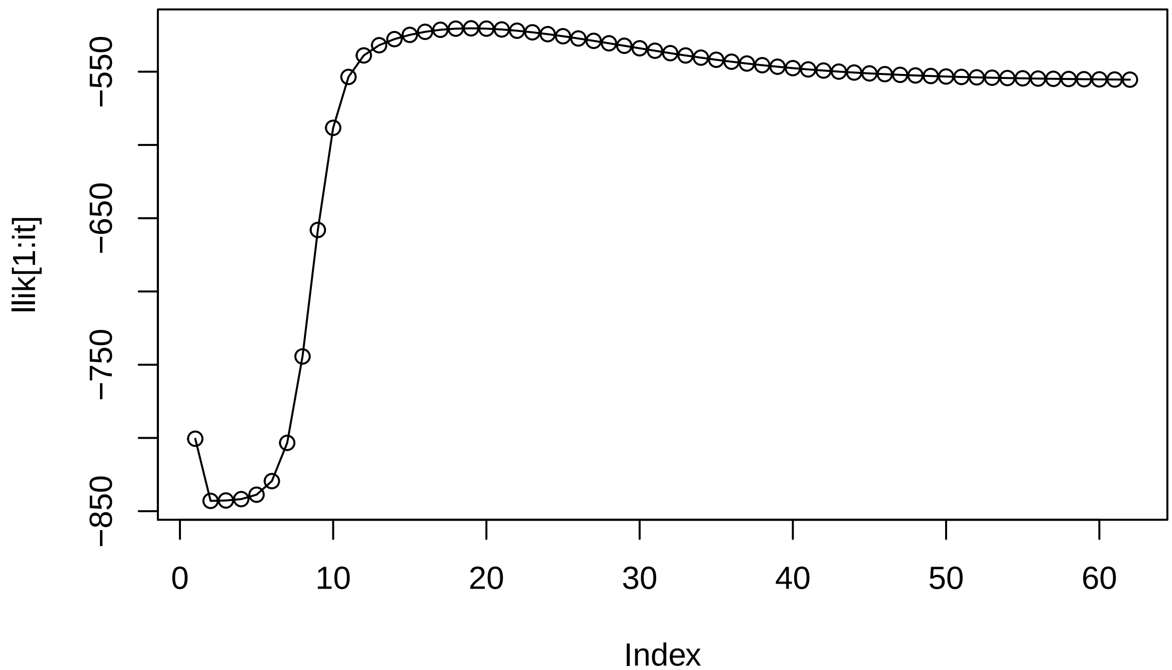

```
## iteration:  60 log likelihood:  -555.2301
```




```
## iteration:  61 log likelihood:  -555.3319
```




```
## iteration:  62 log likelihood:  -555.4231
```




```
## [[1]]
##  [1] 0.0681207107 0.7239375809 0.1144285078 0.0935132006 0.3956837783
##  [6] 0.4293539072 0.4323432998 0.3929703185 0.4162506305 0.5547107213
## [11] 0.4023142592 0.4174015318 0.5420279926 0.4599340095 0.6093481972
## [16] 0.5388153590 0.3444983068 0.6261407708 0.3315033116 0.3455369524
## [21] 0.6696117509 0.4227075553 0.6799271777 0.6690887905 0.2251983009
## [26] 0.5941304803 0.1244291484 0.2278577477 0.7389032359 0.3920562642
## [31] 0.7312641782 0.7396138061 0.1989570475 0.6376090927 0.0220675386
## [36] 0.2067362122 0.7733978028 0.3148516085 0.7696255465 0.7733194550
## [41] 0.0036278205 0.6703401502 0.0000374818 0.0049635197
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
if (!require("pacman")) install.packages("pacman")
pacman::p_load(mboost, randomForest, dplyr, ggplot2)

options(scipen = 999)

spam_data <- read.csv(file = "spambase.data", header = FALSE)
colnames(spam_data)[58] <- "Spam"
spam_data$Spam <- factor(spam_data$Spam, levels = c(0,1), labels = c("0", "1"))
set.seed(12345)
n =  NROW(spam_data)
```

```r
id = sample(1:n, floor(n*(2/3)))
train = spam_data[id,]
test = spam_data[-id,]

final_result <- NULL
for(i in seq(from = 10, to = 100, by = 10)){

ada_model <- mboost::blackboost(Spam~.,
                                 data = train,
                                 family = AdaExp(),
                               control=boost_control(mstop=i))

forest_model <- randomForest(Spam~., data = train, ntree = i)


prediction_function <- function(model, data){
  predicted <- predict(model, newdata = data, type = c("class"))
  predict_correct <- ifelse(data$Spam == predicted, 1, 0)
  score <- sum(predict_correct)/NROW(data)
  return(score)
}


train_ada_model_predict <- predict(ada_model, newdata = train, type = c("class"))
test_ada_model_predict <- predict(ada_model, newdata = test, type = c("class"))
train_forest_model_predict <- predict(forest_model, newdata = train, type = c("class"))
test_forest_model_predict <- predict(forest_model, newdata = test, type = c("class"))

test_predict_correct <- ifelse(test$Spam == test_forest_model_predict, 1, 0)
train_predict_correct <- ifelse(train$Spam == train_forest_model_predict, 1, 0)


train_ada_score <-  prediction_function(ada_model, train)
test_ada_score <-  prediction_function(ada_model, test)
train_forest_score <-  prediction_function(forest_model, train)
test_forest_score <-  prediction_function(forest_model, test)

iteration_result <- data.frame(number_of_trees = i,
                               accuracy = c(train_ada_score,
                                            test_ada_score,
                                            train_forest_score,
                                            test_forest_score),
                               type  = c("train", "test", "train", "test"),
                               model = c("ADA", "ADA",  "Forest", "Forest"))


final_result <- rbind(iteration_result, final_result)
}

final_result$error_rate_percentage <- 100*(1 - final_result$accuracy)
ggplot(data = final_result, aes(x = number_of_trees,
                                y = error_rate_percentage,
                                group = type, color = type)) +
```

```
  geom_point() +
  geom_line() +
  ggtitle("Error Rate vs. increase in trees") + facet_grid(rows = vars(model))

myem <- 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 = K) # true mixing coefficients
true_mu <- matrix(nrow=K, ncol=D) # true conditional distributions
true_pi=c(rep(1/3, K))

if(K == 2){
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")

  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)
}else if(K == 3){
    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")

  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)
}else {
    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")
    points(true_mu[4,], type="o", col="yellow")

    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)
    true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)}

# Producing the training data
for(n in 1:N) {
k <- sample(1:K,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}

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
```

```r
# 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) {


if(K == 2){
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
}else if(K == 3){
  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")
}else{
    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(k in 1:K)
prod <- exp(x %*% log(t(mu))) * exp((1-x) %*% t(1-mu))

num = matrix(rep(pi,N), ncol = K, byrow = TRUE) * prod
dem = rowSums(num)
poster = num/dem

#Log likelihood computation.
llik[it] = sum(log(dem))
# Your code here
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
# Your code here
num_pi = colSums(poster)
pi = num_pi/N
mu = (t(poster) %*% x)/num_pi
}
```

```
a <- pi
b <- mu
c <- plot(llik[1:it], type="o")
result <- list(c(a,b,c))
return(result)
}
myem_loop <- function(K){
# 2 - Mixture Models ####
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 = K) # true mixing coefficients
true_mu <- matrix(nrow=K, ncol=D) # true conditional distributions
true_pi=c(rep(1/3, K))


if (K == 2){
  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)
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")
}else if (K == 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")
}else{
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)
true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)
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")
points(true_mu[4,], type="o", col="yellow")
}


# Producing the training data
for(n in 1:N) {
  k <- sample(1:K,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}
```

```r
 # number of guessed components
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)
}
pi
mu
for(it in 1:max_it) {
  if (K == 2){
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
  }else if (K == 3){
    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")
  }else{
    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
  m <- matrix(NA, nrow = 1000, ncol = k)

  #Here I create the Bernouilli probabilities, lecture 1b, slide 7. I use 3 loops to do it for the thre
  # not very efficient, but it works.
  for (j in 1:k){
    for(each in 1:nrow(x)){
      row <- x[each,]
      vec <- c()
      for (i in 1:10) {
        a <- mu[j,i]^row[i]
        b <- a * ((1-mu[j,i])^(1-row[i]))
        vec[i] <- b
        c <- prod(vec)
      }
      m[each, j] <- c
    }
  }

  # Here I create a empty matrix, to store all values for the numerator of the formula on the bottom of
  # slide 9, lecture 1b.
  m2 <- matrix(NA, ncol = k, nrow = 1000)

  # m2 stores all the values for the numerator of the formula on the bottom of slide 9, lecture 1b.
  for (i in 1:1000){
```

```r
    a <- pi * m[i,]
    m2[i,] <- a
  }

  # Sum m2 to get the denominator of the formula on the bottom of slide 9, lecture 1b.
  m2_sum <- rowSums(m2)
  m_final <- m2 / m2_sum

  #Log likelihood computation.
  ll <- matrix(nrow = 1000, ncol = K)
  for (j in 1:K){
    for (i in 1:1000){
      ll[i, j] <- sum(((x[i,] * log(mu[j,])) + (1 - x[i,])*log(1-mu[j,])))
    }
  }

  ll <- ll + pi
  llnew <- m_final * ll
  llik[it] <- sum(rowSums(llnew))

  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

  # Create the numerator for pi, slide 9, lecture 1b.
  numerator_pi <- colSums(m_final)

  # Create new values for pi, stored in the vector pi_new
  pi_new <- numerator_pi / N
  pi_new
  mnew <- matrix(NA, nrow = 1000, ncol = 10)
  mu_new <- matrix(NA, nrow = K, ncol = 10)

  for (j in 1:k){
    for (i in 1:1000){
      row <- x[i,] * m_final[i,j]
      mnew[i,] <- row
    }
    mnewsum <- colSums(mnew)/numerator_pi[j]
    mu_new[j,] <- mnewsum
  }

  # Now, to create the iterations, I have to run the code again and again, and specifying mu as new the
  # created for mu. Same goes for the other variables.
  mu <- mu_new
  pi <- pi_new
}
z <- m_final
output1 <- pi
```

```r
output2 <- mu
output3 <- plot(llik[1:it], type="o")

result <- list(c(output1, output2, output3))
return(result)
}
myem_loop(K=3)

myem(K=2)
myem(K=3)
myem(K=4)
```