lab1_block2

Thijs Quast

30-11-2018

Contents

Question 1 Ensemble Methods

```
# Loading packages and importing files ####
library(mboost)
```

```
## Loading required package: parallel
```

```
## Loading required package: stabs
```

```
## This is mboost 2.9-1. See 'package?mboost' and 'news(package  = "mboost")'
## for a complete list of changes.
```

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## The following object is masked from 'package:mboost':
##
##     %+%
```

```
sp <- read.csv2("spambase.csv", header = FALSE, sep = ",", stringsAsFactors = FALSE)
num_sp <- data.frame(data.matrix(sp))
num_sp$V58 <- factor(num_sp$V58)
```

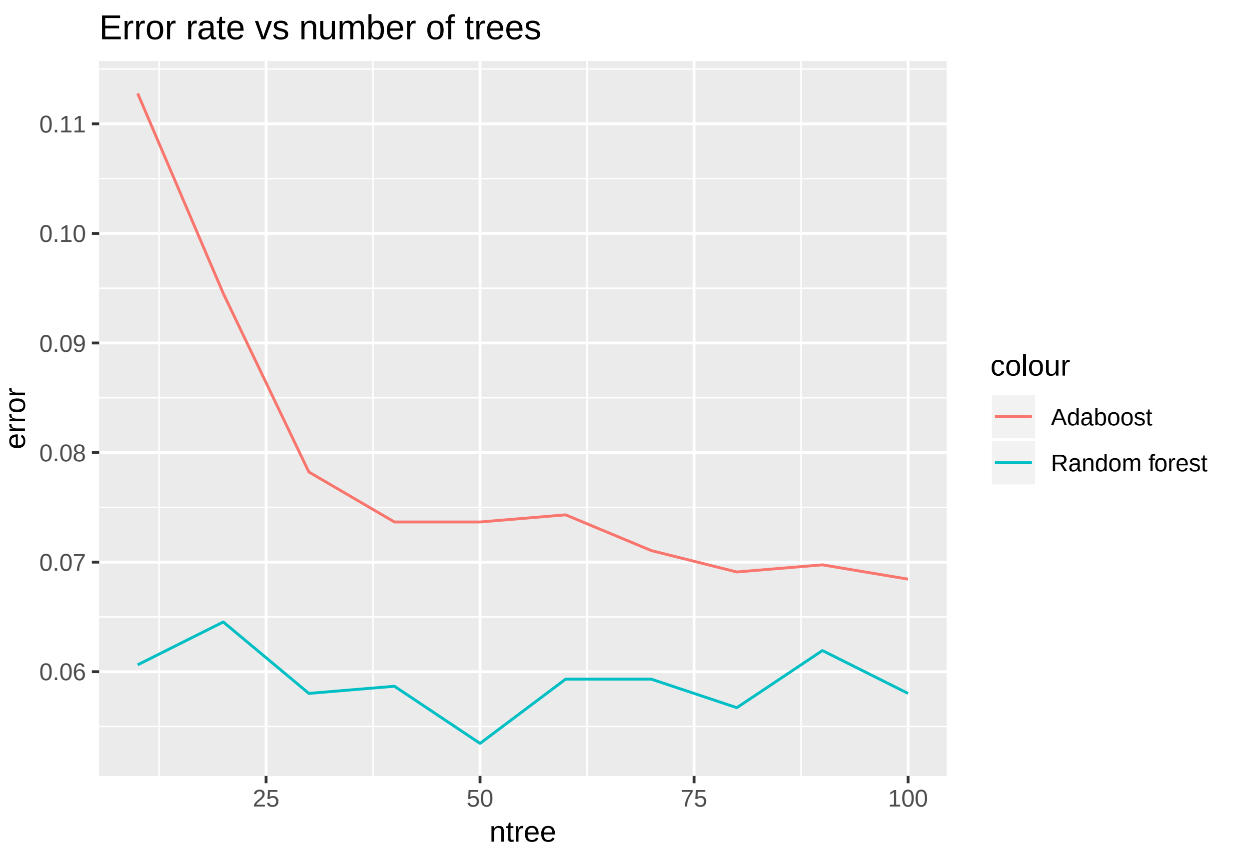
```
# shuffling data and dividing into train and test ####
n <- dim(num_sp)[1]
ncol <- dim(num_sp)[2]
set.seed(1234567890)
id <- sample(1:n, floor(n*(2/3)))
train <- num_sp[id,]
test <- num_sp[-id,]
```

```
# Adaboost
ntree <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
error <- c()

for (i in seq(from = 10, to = 100, by = 10)){
bb <- blackboost(V58 ~., data = train, control = boost_control(mstop = i), family = AdaExp())
bb_predict <- predict(bb, newdata = test, type = c("class"))
confusion_bb <- table(test$V58, bb_predict)
miss_class_bb <- (confusion_bb[1,2] + confusion_bb[2,1])/nrow(test)
error[(i/10)] <- miss_class_bb
}

error_df <- data.frame(cbind(ntree, error))
```

```
# Random forest ####
ntree_rf <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
```

```r
error_rf <- c()

for (i in seq(from = 10, to = 100, by = 10)){
rf <- randomForest(V58 ~., data = train, ntree= 10)
rf_predict <- predict(rf, newdata = test, type = c("class"))
confusion_rf <- table(test$V58, rf_predict)
miss_class_rf <- (confusion_rf[1,2] + confusion_rf[2,1])/nrow(test)
error_rf[i/10] <- miss_class_rf
}

error_df_rf <- data.frame(cbind(ntree_rf, error_rf))
```

```r
df <- cbind(error_df, error_df_rf)
df <- df[, -3]

plot_final <- ggplot(df, aes(ntree)) +
  geom_line(aes(y=error, color = "Adaboost")) +
  geom_line(aes(y=error_rf, color = "Random forest"))

plot_final <- plot_final + ggtitle("Error rate vs number of trees")
plot_final
```



The error rate for the AdaBoost model are clearly going down when the number of trees increases. Finally the model arrives at an error rate below 7% when 100 trees are included in the model. For the randomforest the pattern is less obvious, the error rate seems to go up and down as the number of trees in the model increases. 50 trees result in the lowest error rate. This error rate is also lower than the error rate produced by the best Adaboost model (100 trees). Therefore, for this spam classification, a randomforest with 50 trees seems to be most suitable.

Question 2 Mixture Models

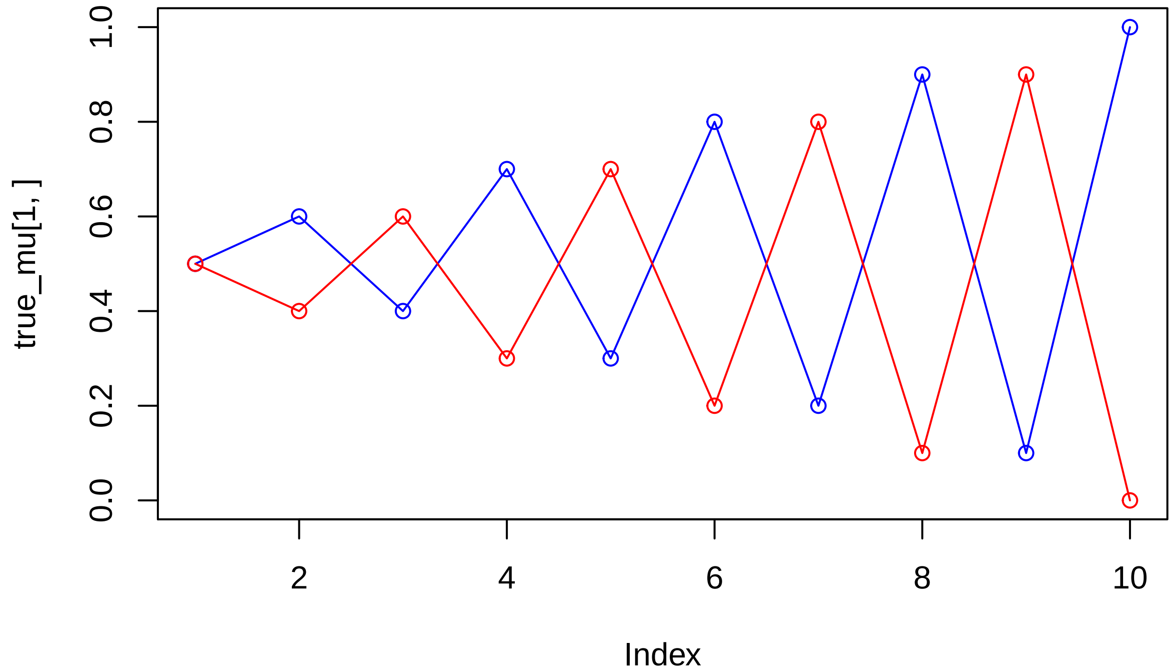
```
my_own_em <- function(K){
# 2 - Mixture Models ####
set.seed(1234567890)

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])
  }
}

 # number of guessed components
```

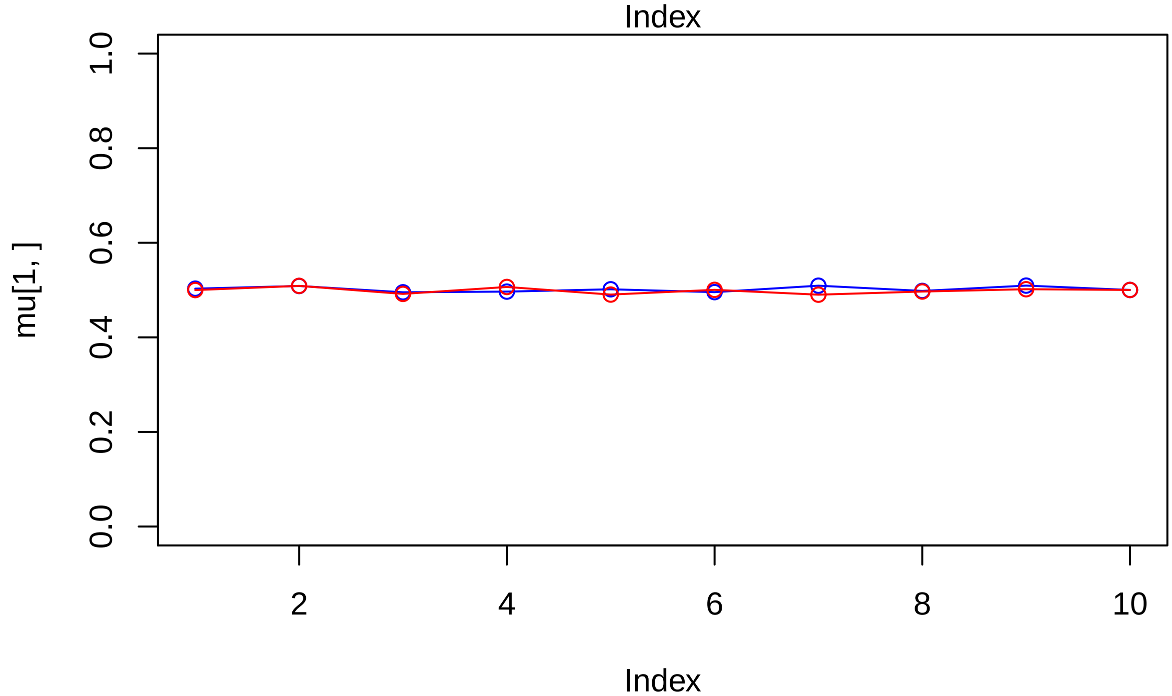
```r
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
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){
    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
output2 <- mu
```

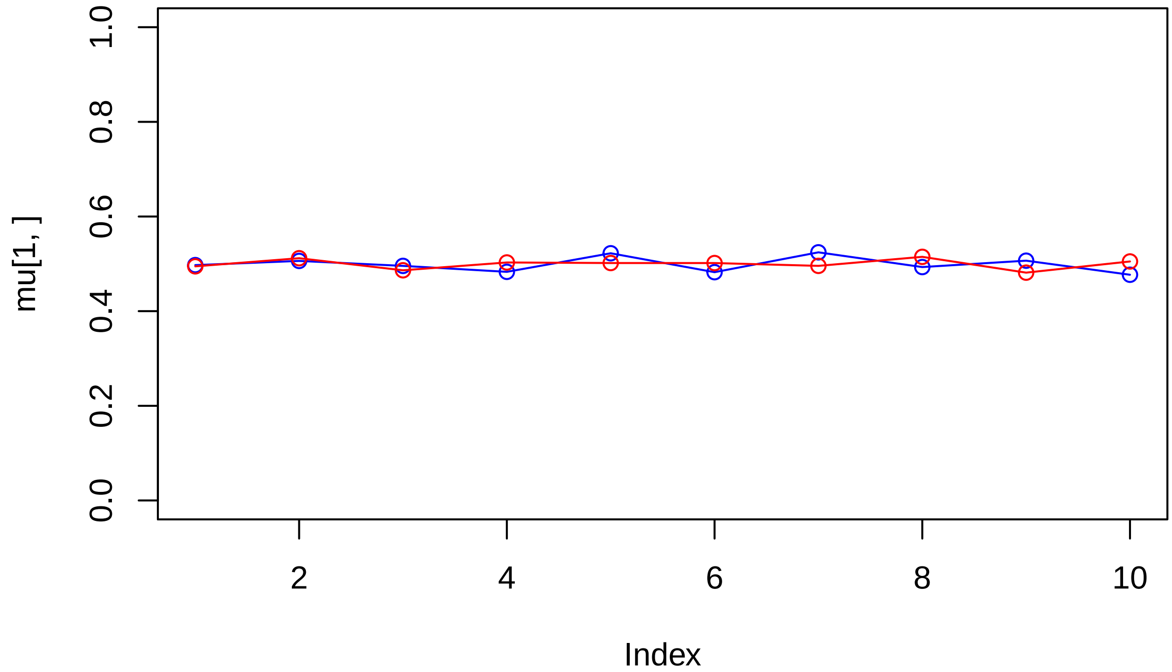
```
output3 <- plot(llik[1:it], type="o")
z
result <- list(c(output1, output2, output3))
return(result)
}
```

```
my_own_em(2)
```

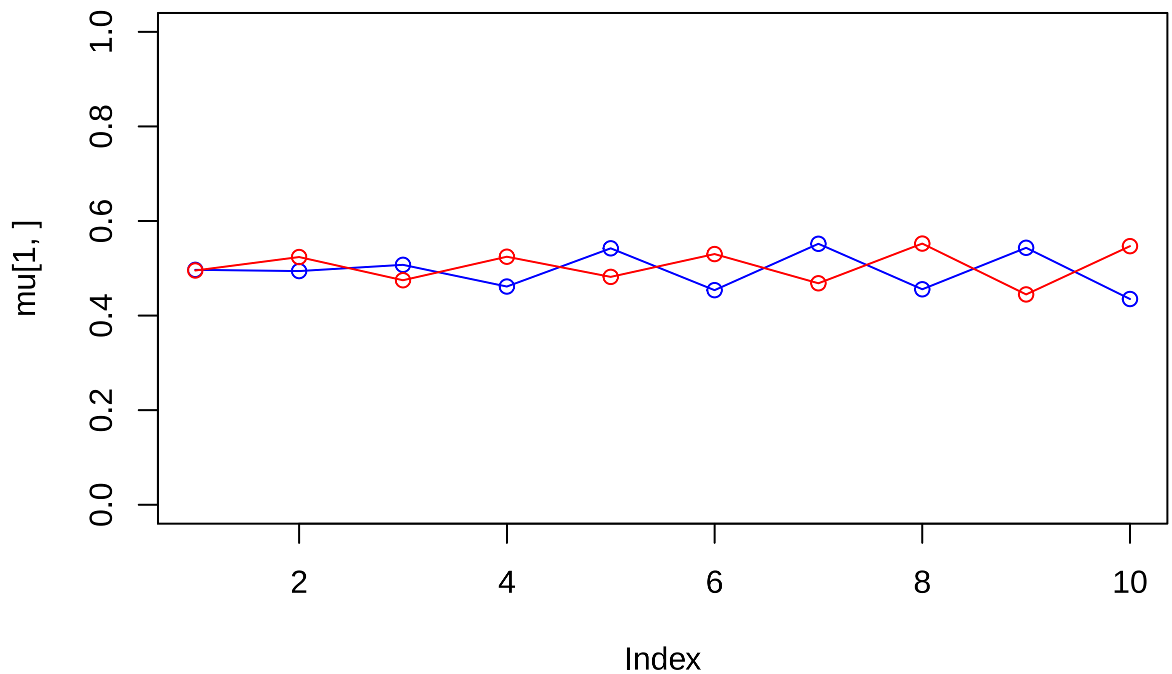




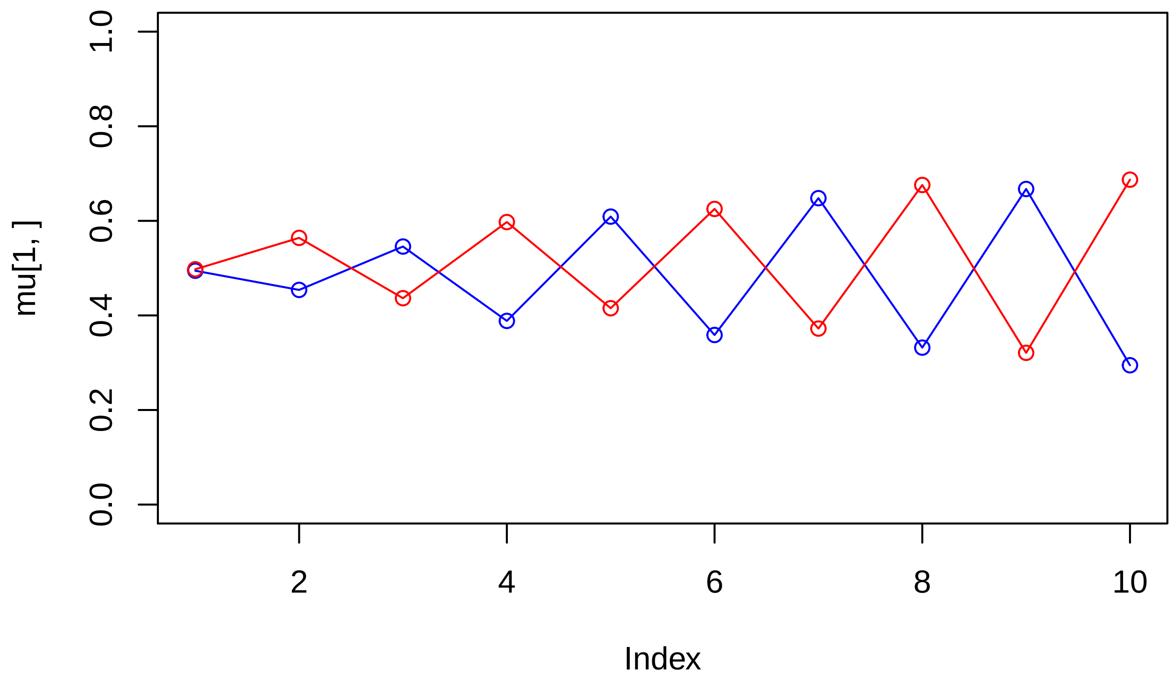

```
## iteration:  1 log likelihood:  -6430.751
```



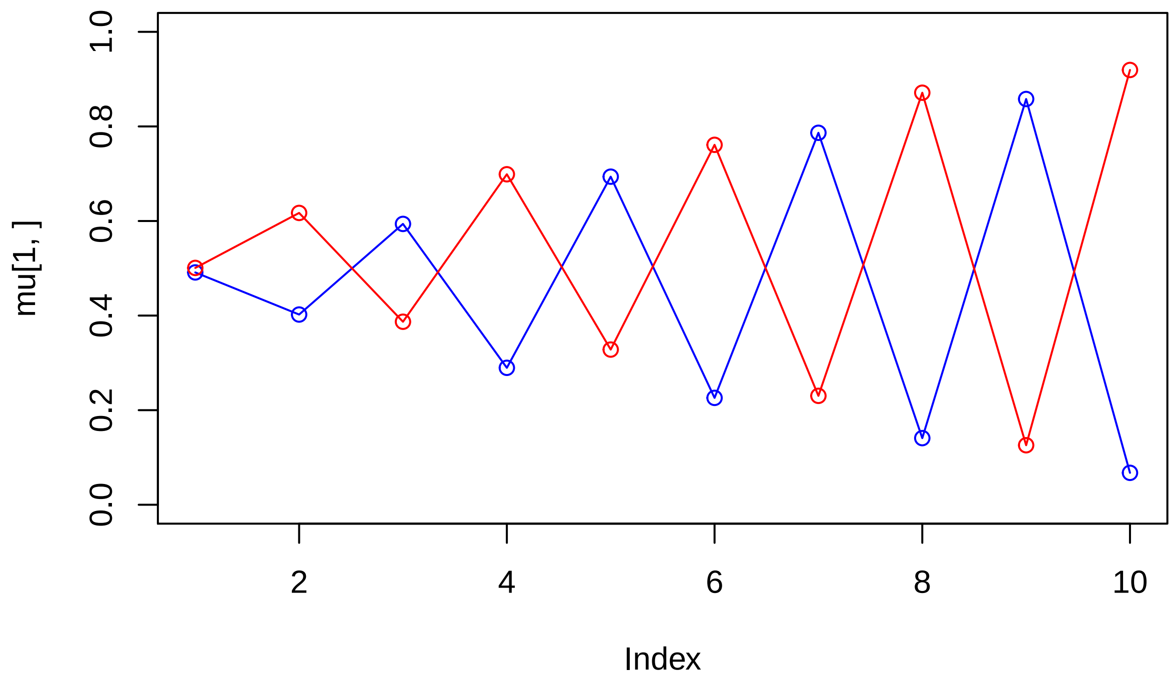

```
## iteration:  2 log likelihood:  -6417.599
```



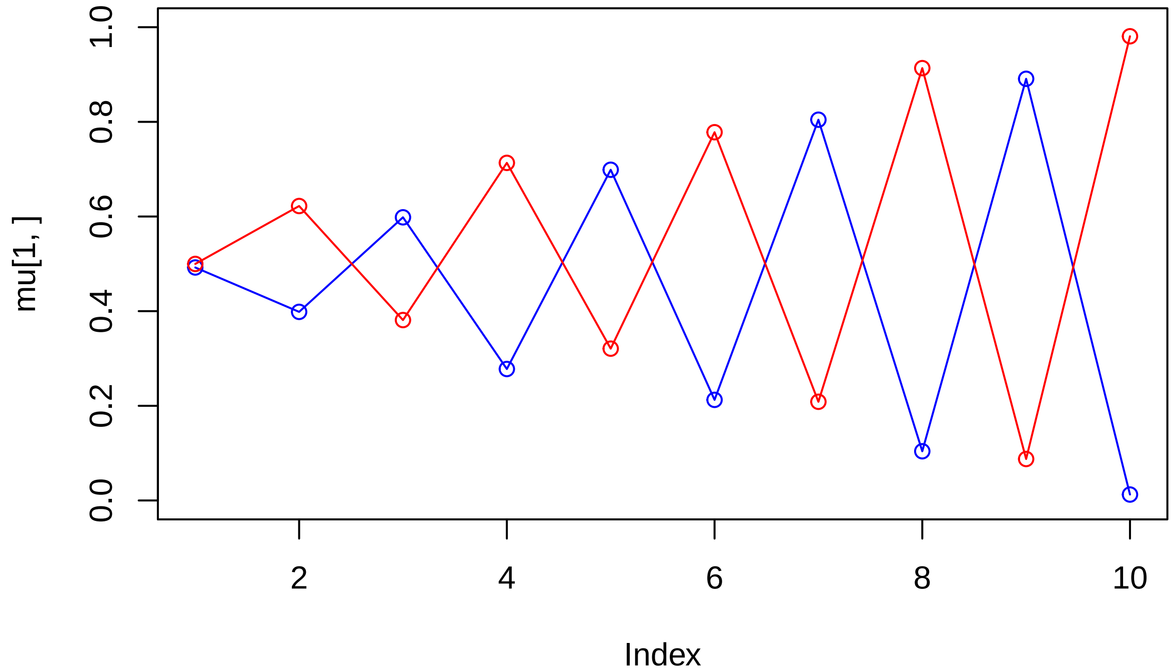

```
## iteration:  3 log likelihood:  -6270.298
```



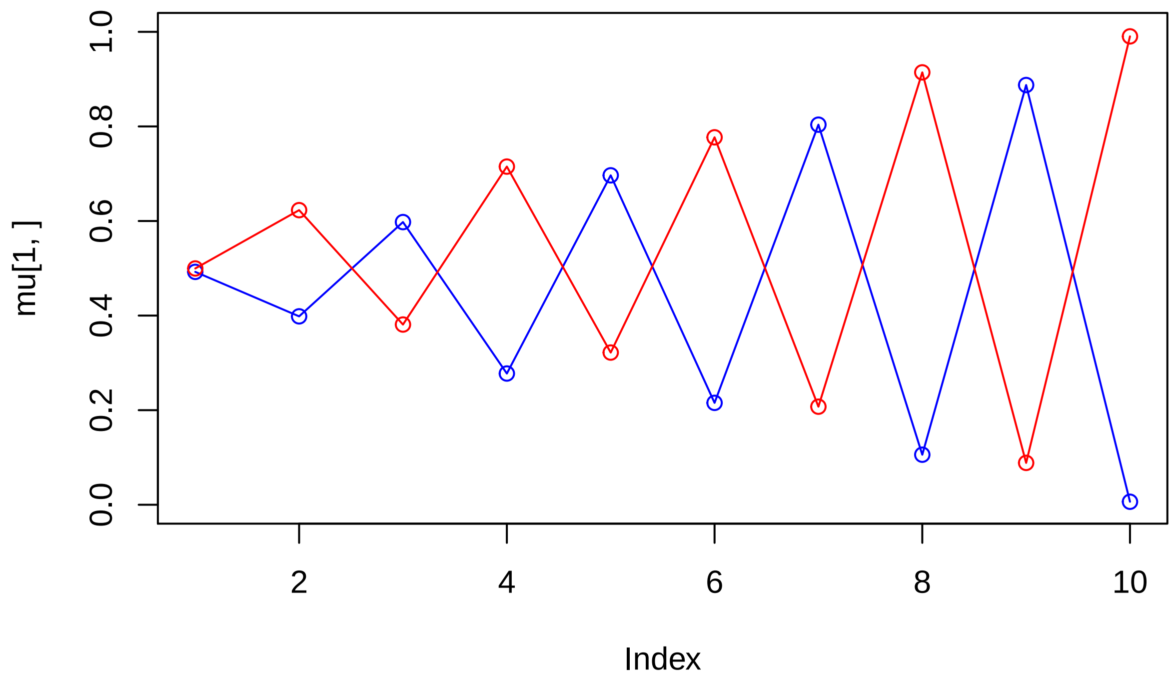

```
## iteration:  4 log likelihood:  -5381.969
```



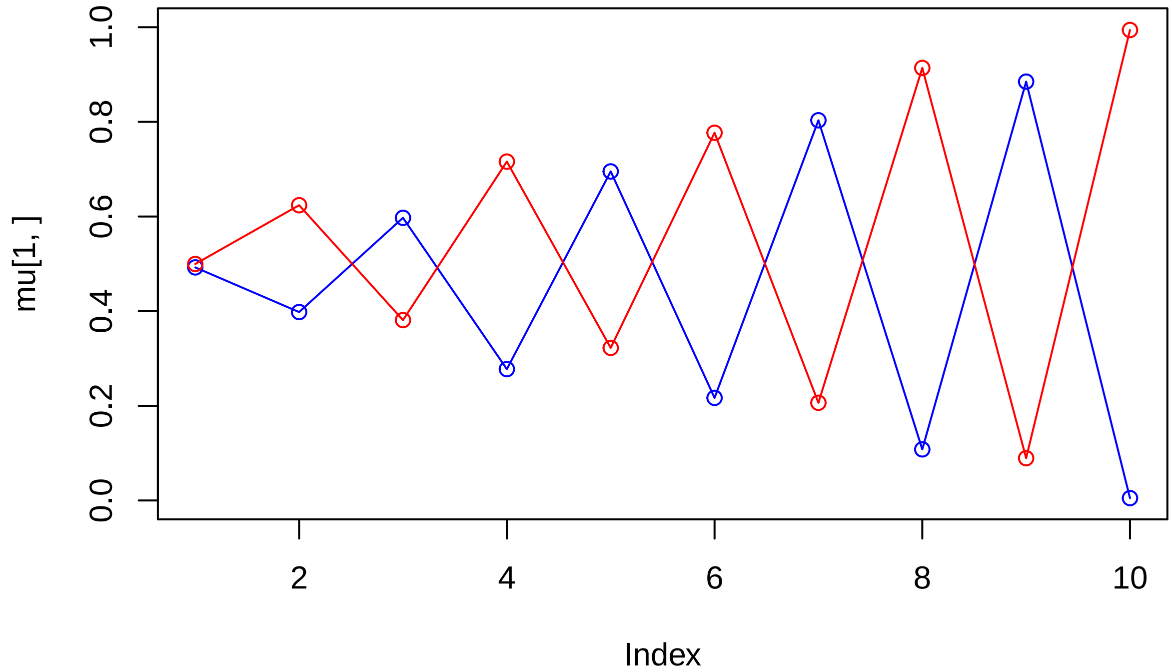

```
## iteration:  5 log likelihood:  -4538.463
```



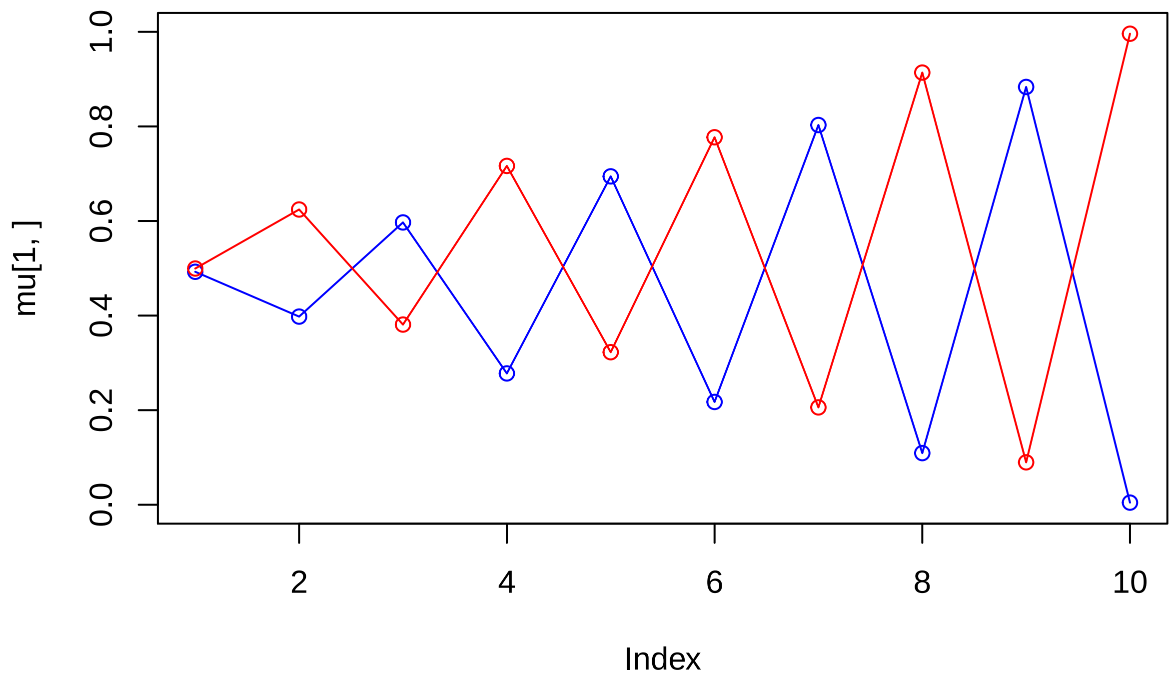

```
## iteration:  6 log likelihood:  -4463.134
```



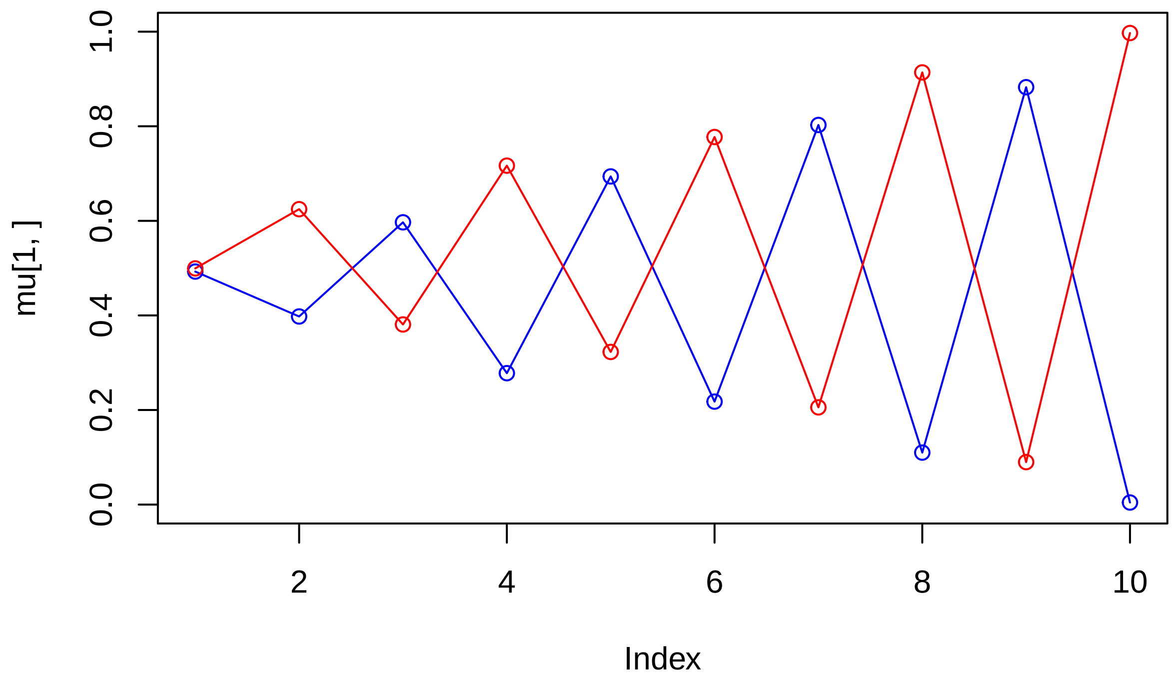

```
## iteration:  7 log likelihood:  -4455.903
```



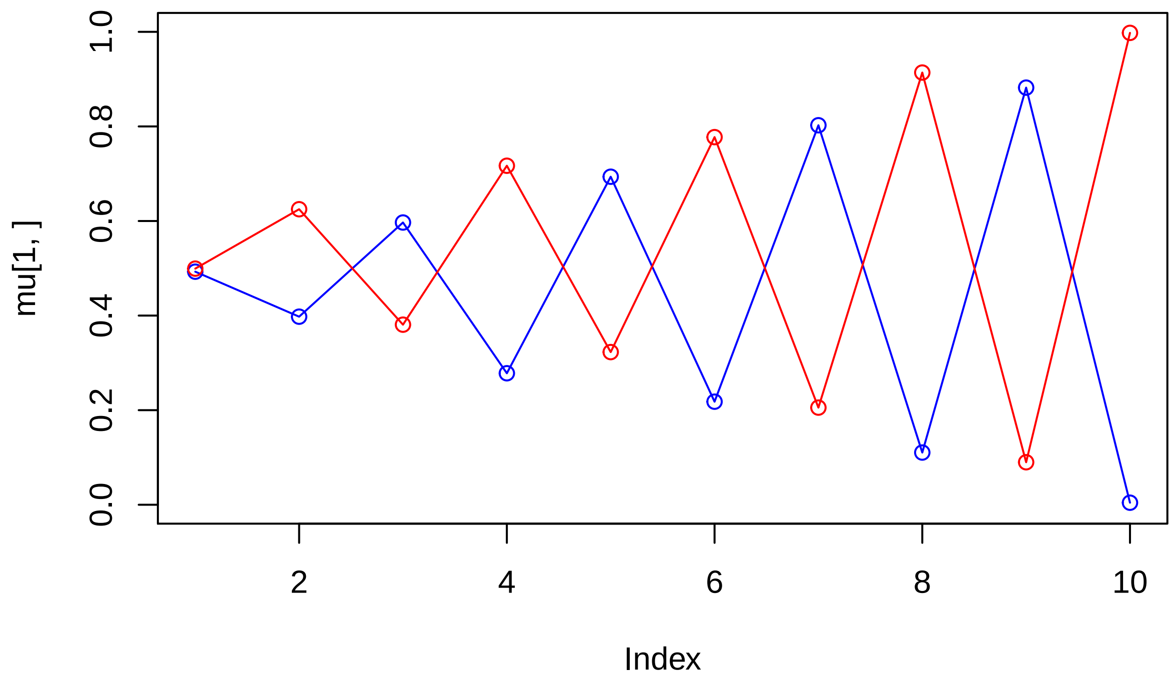

```
## iteration:  8 log likelihood:  -4453.165
```



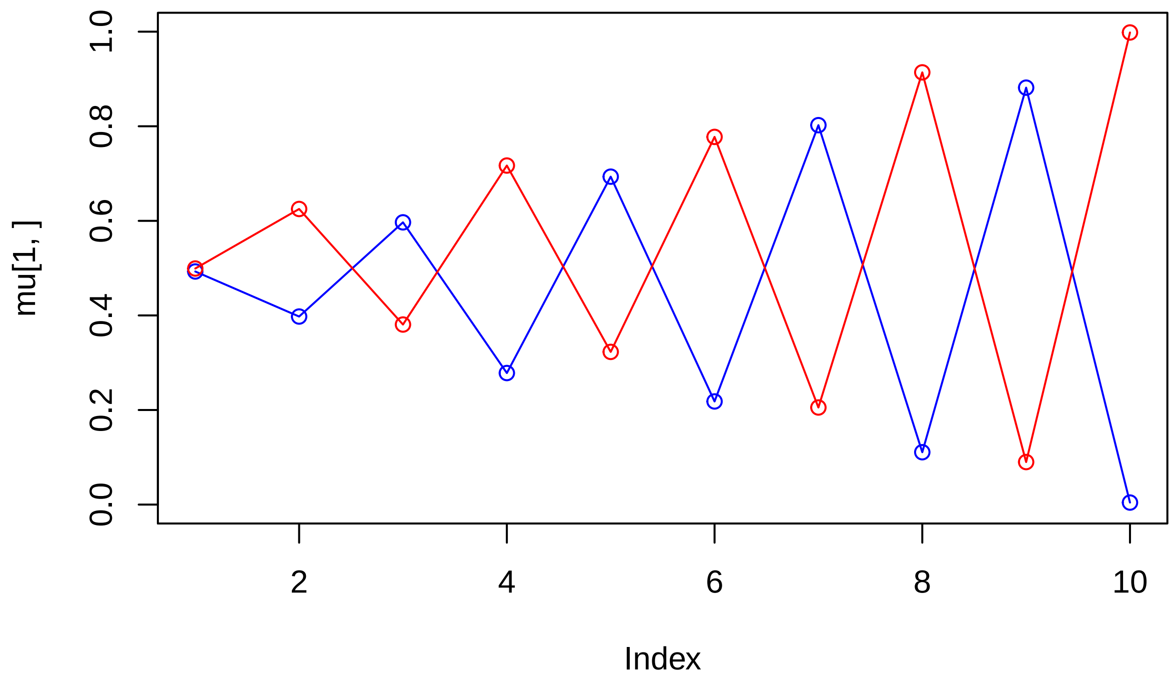

```
## iteration:  9 log likelihood:  -4451.653
```



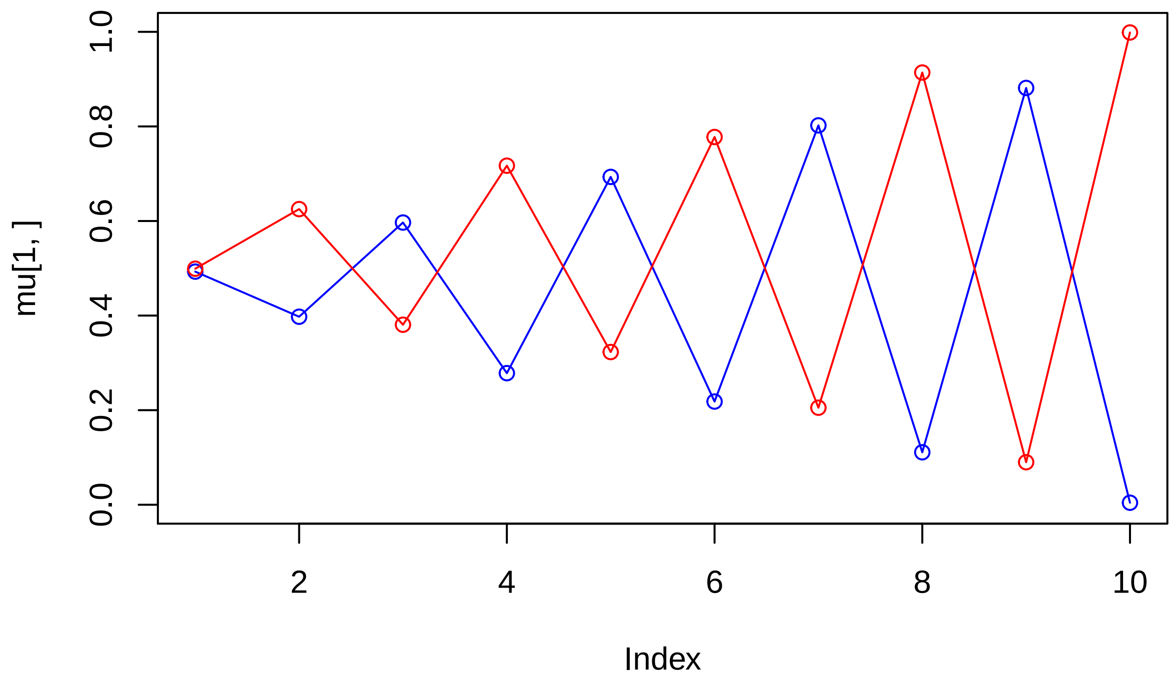

```
## iteration:  10 log likelihood:  -4450.674
```



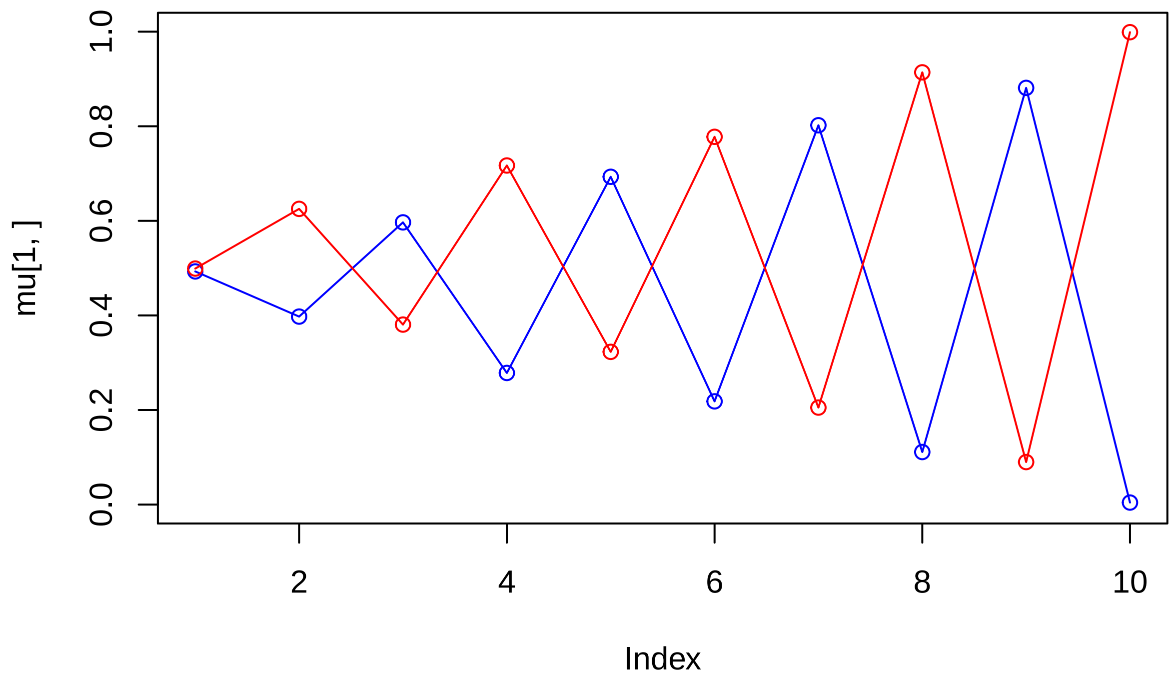

```
## iteration:  11 log likelihood:  -4449.981
```



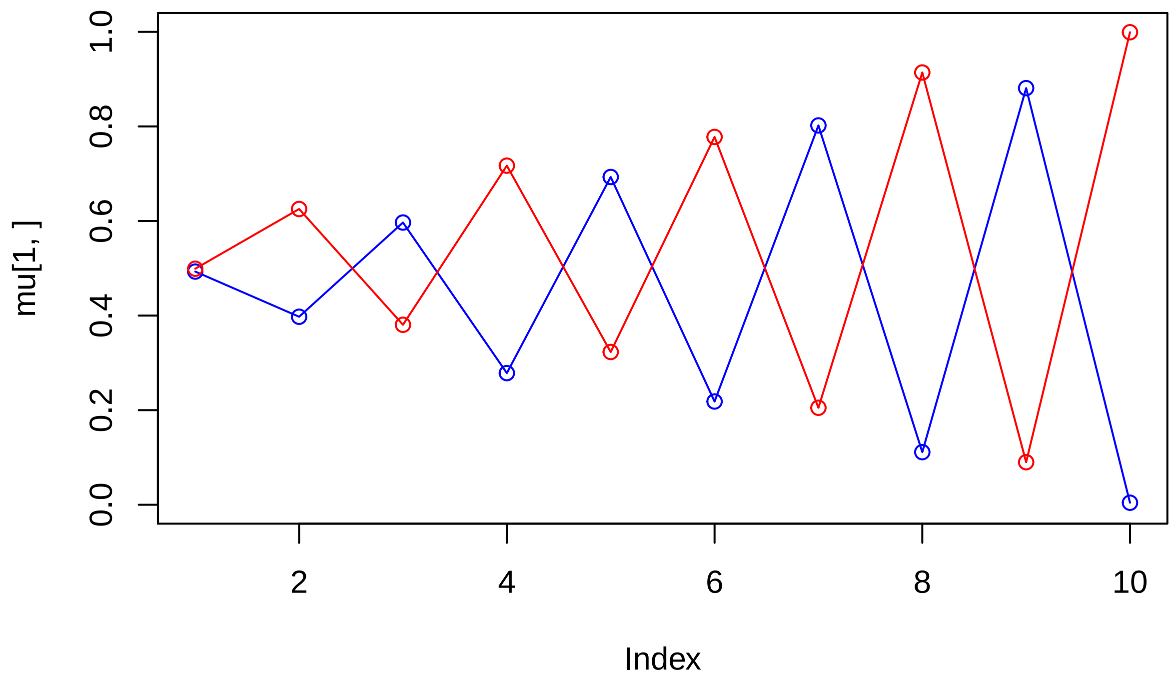

```
## iteration:  12 log likelihood:  -4449.461
```



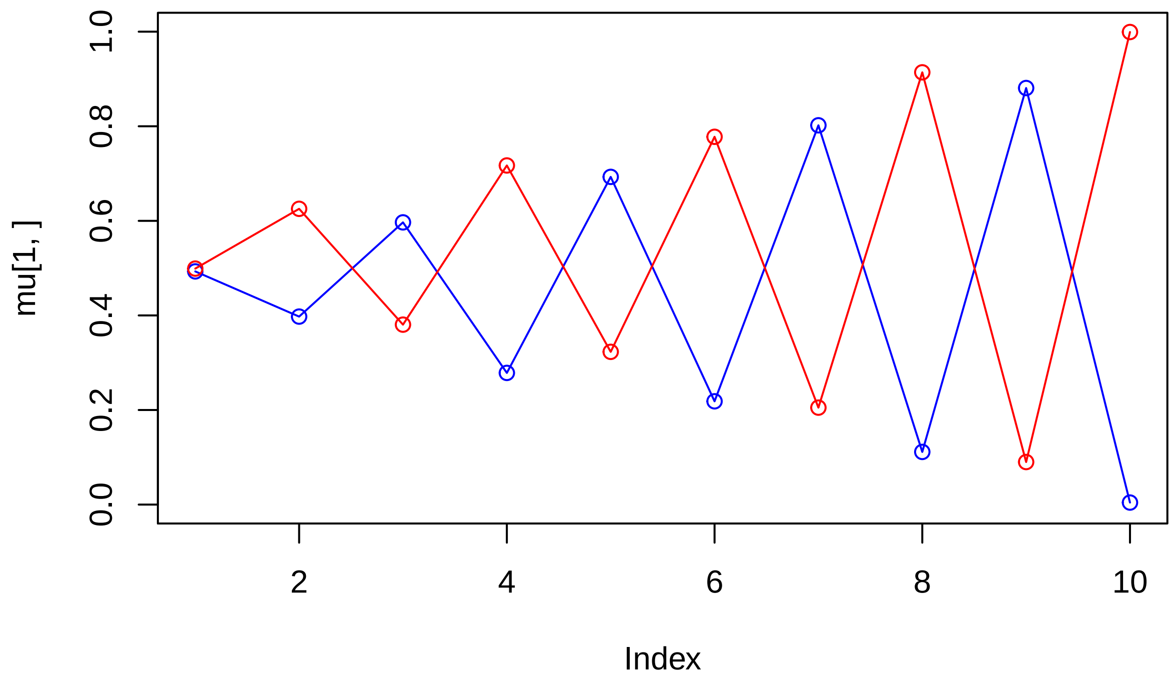

```
## iteration:  13 log likelihood:  -4449.057
```



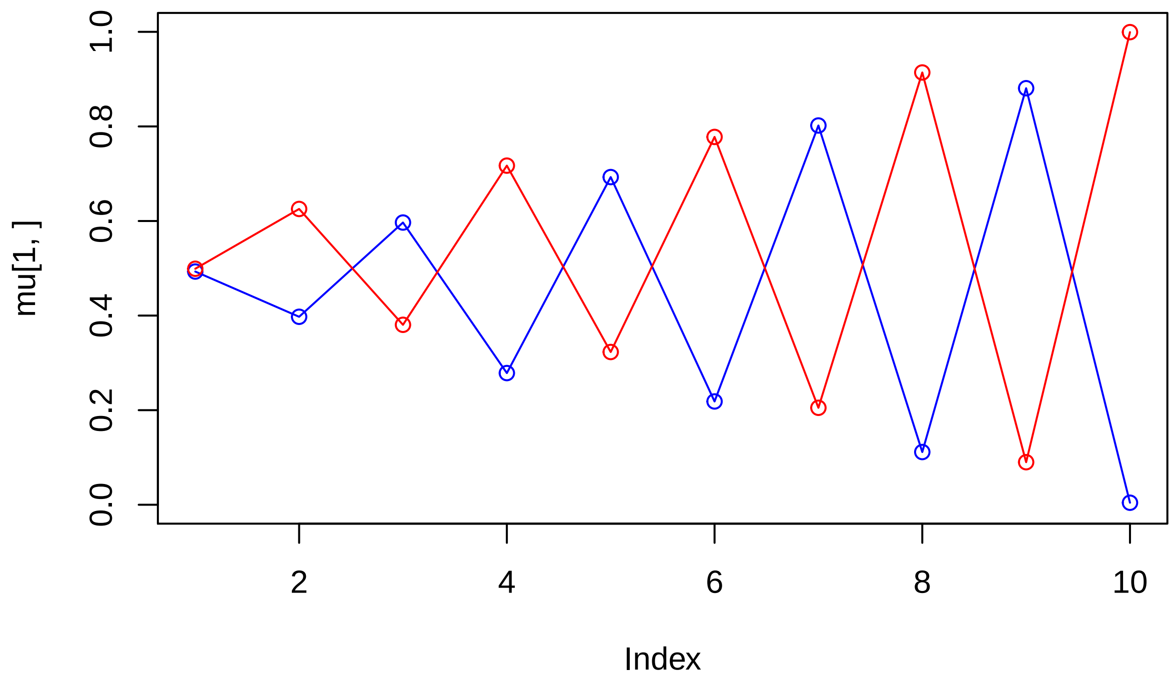

```
## iteration:  14 log likelihood:  -4448.734
```



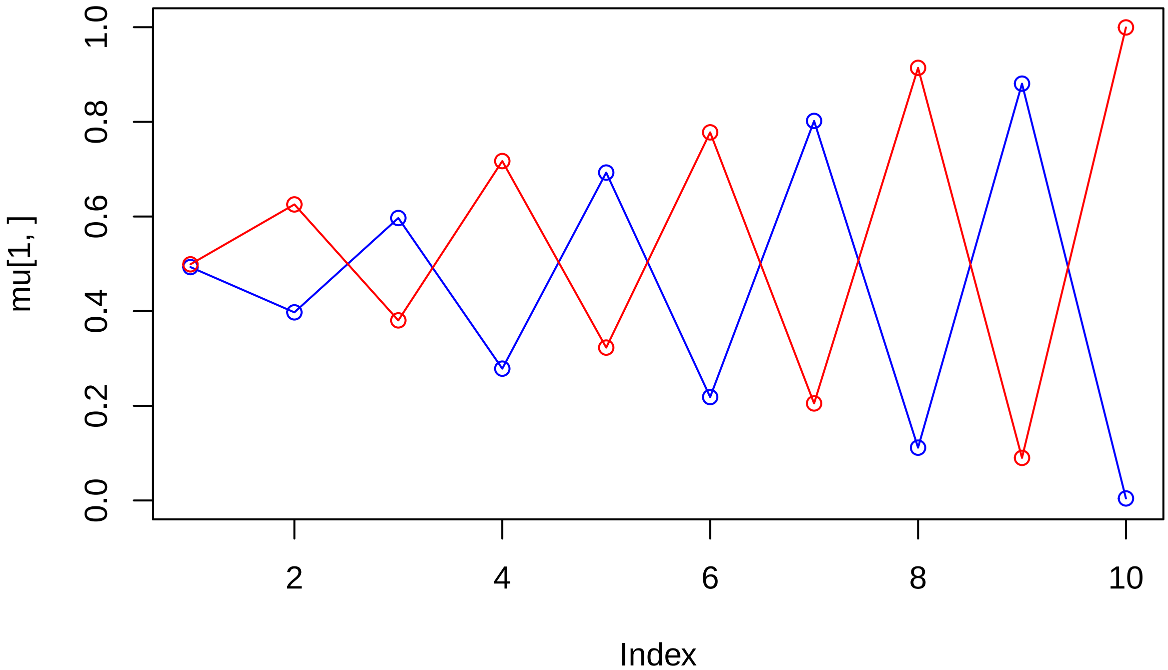

```
## iteration:  15 log likelihood:  -4448.47
```



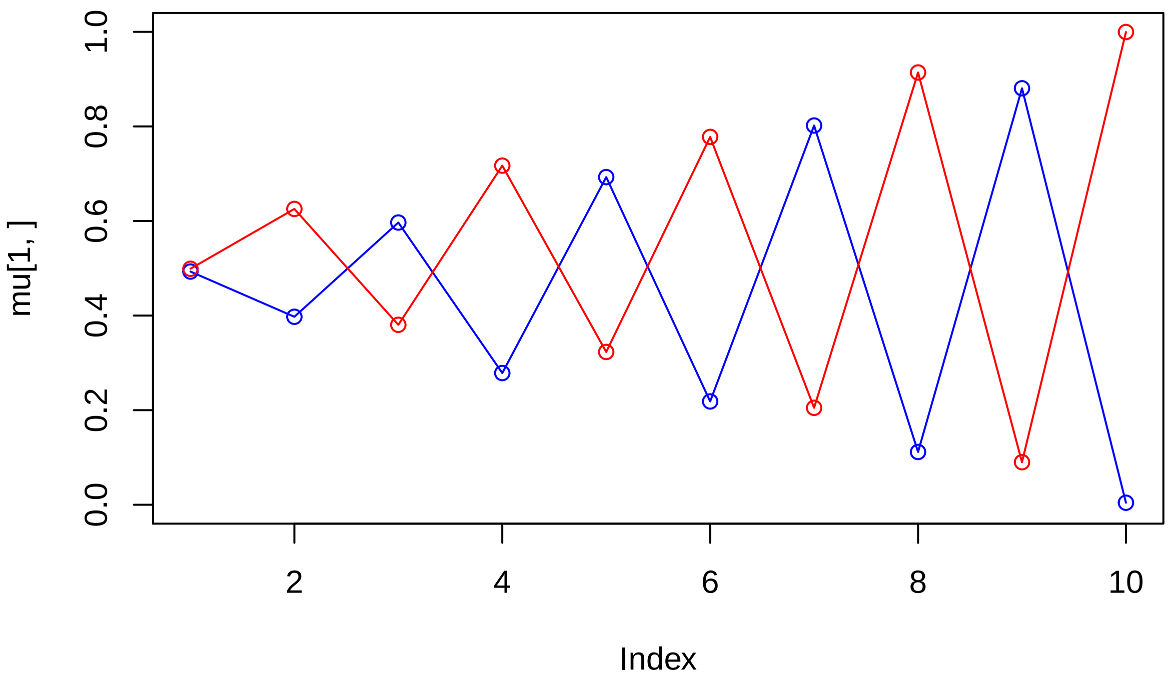

```
## iteration:  16 log likelihood:  -4448.251
```



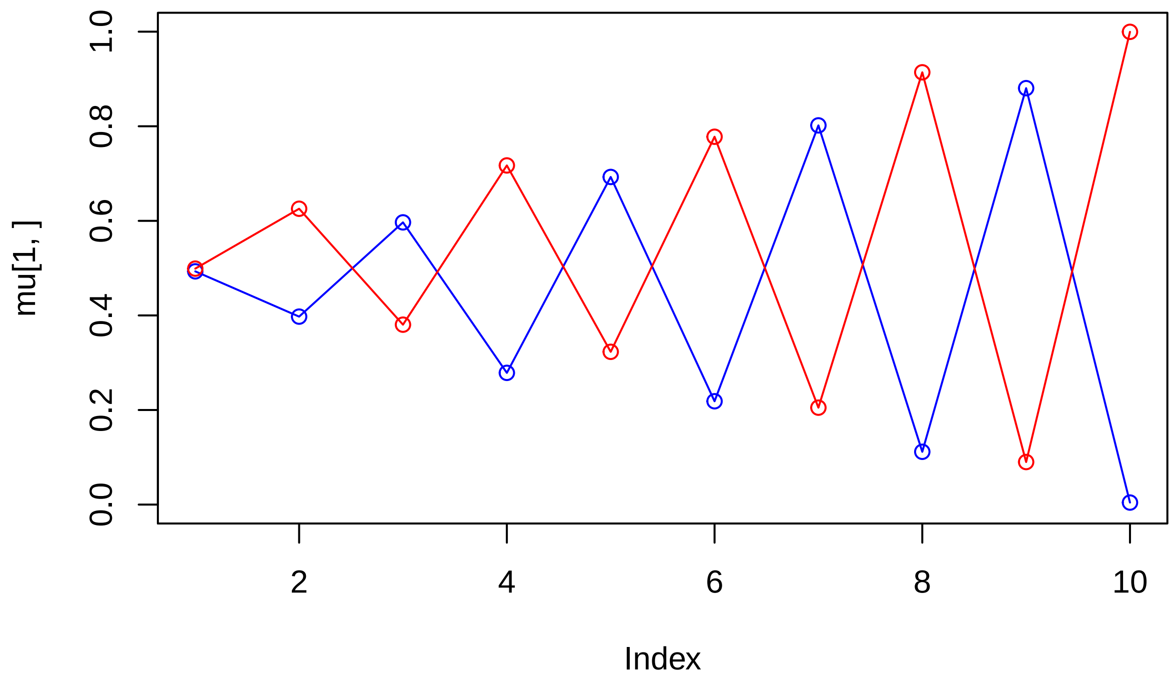

```
## iteration:  17 log likelihood:  -4448.068
```



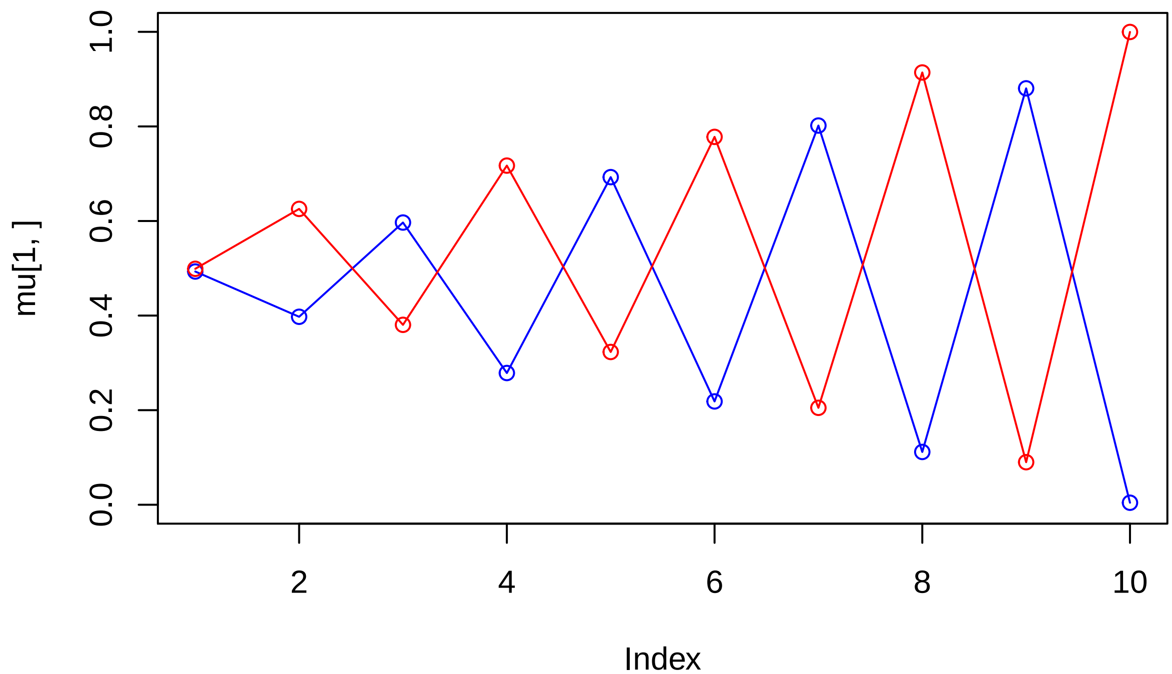

```
## iteration:  18 log likelihood:  -4447.913
```



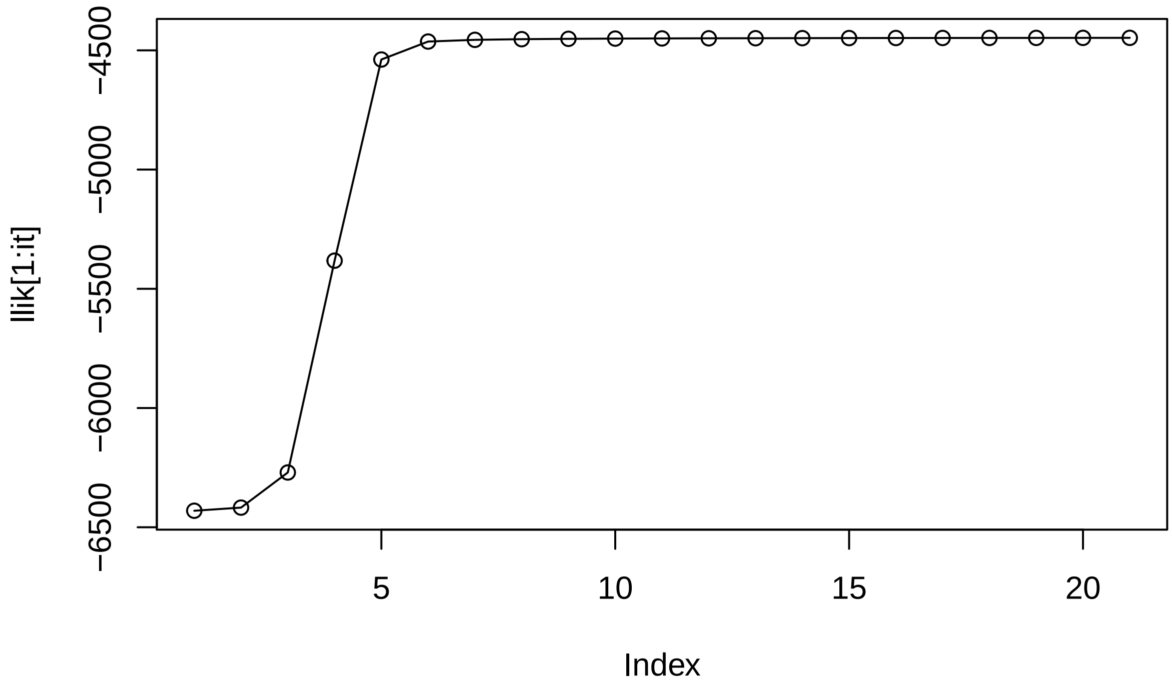

```
## iteration:  19 log likelihood:  -4447.781
```




```
## iteration:  20 log likelihood:  -4447.669
```

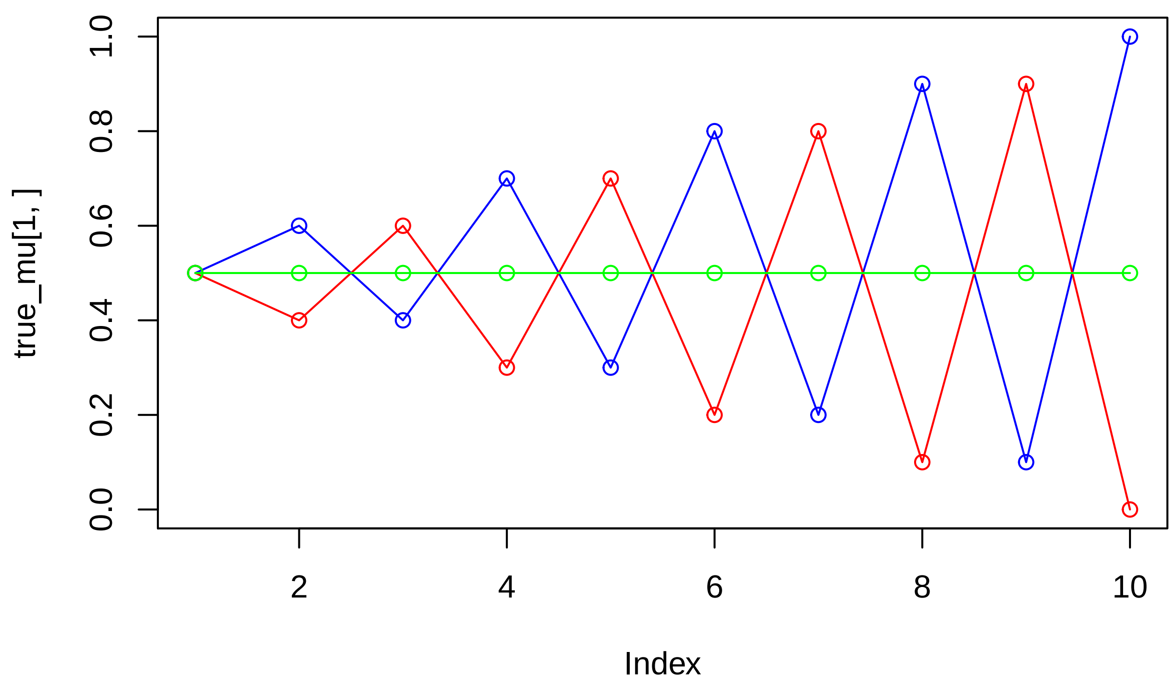


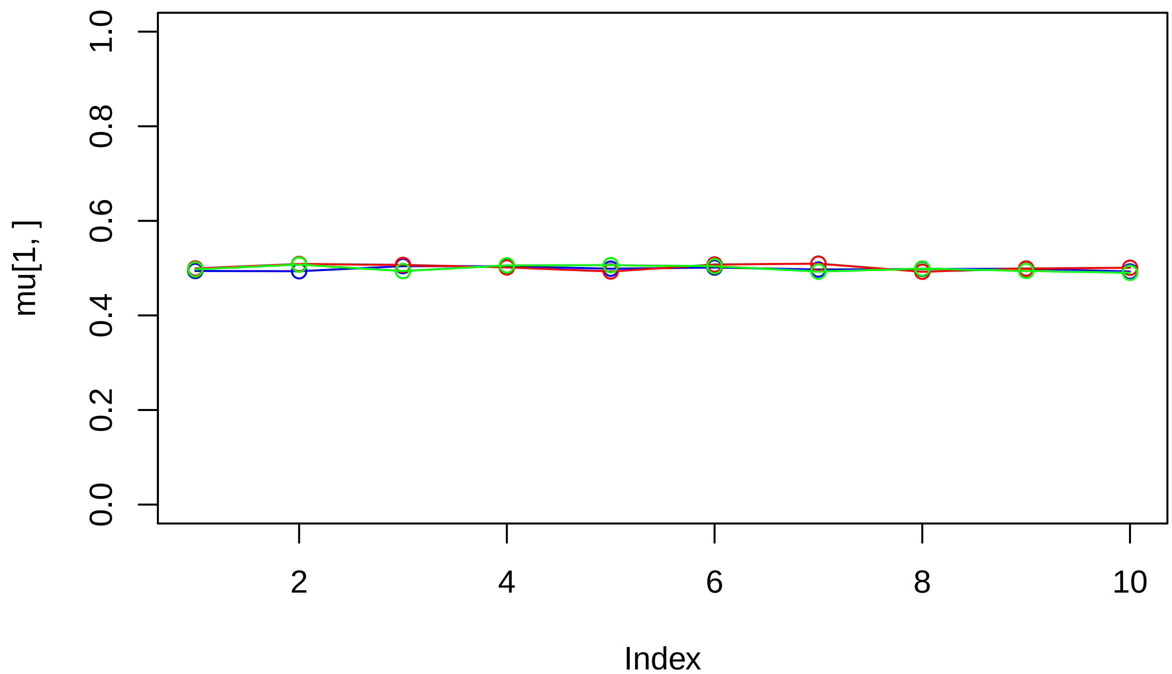

```
## iteration:  21 log likelihood:  -4447.571
```



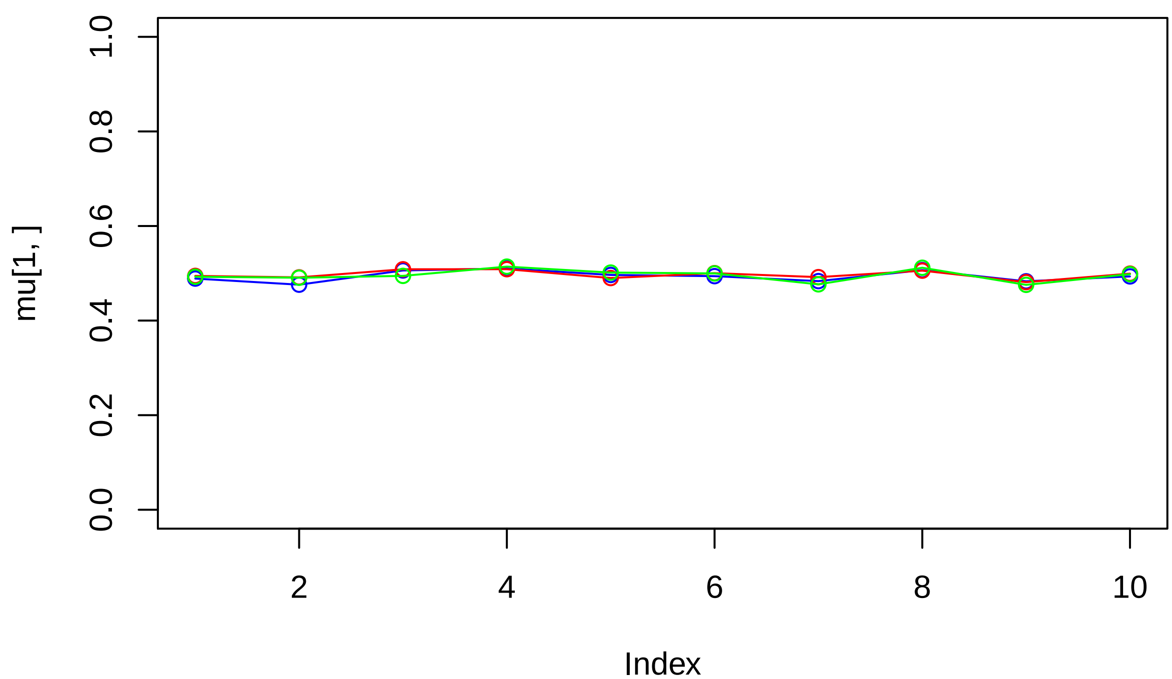

```
## [[1]]
##  [1] 0.511053119 0.488946881 0.493173475 0.498954318 0.397460637
##  [6] 0.625582274 0.596781124 0.380436306 0.278547959 0.717147834
## [11] 0.692791708 0.323034348 0.218495730 0.777869929 0.801849083
## [16] 0.204955853 0.111647688 0.914091323 0.880544386 0.089979192
## [21] 0.004290353 0.999714736
```

```r
my_own_em(3)
```

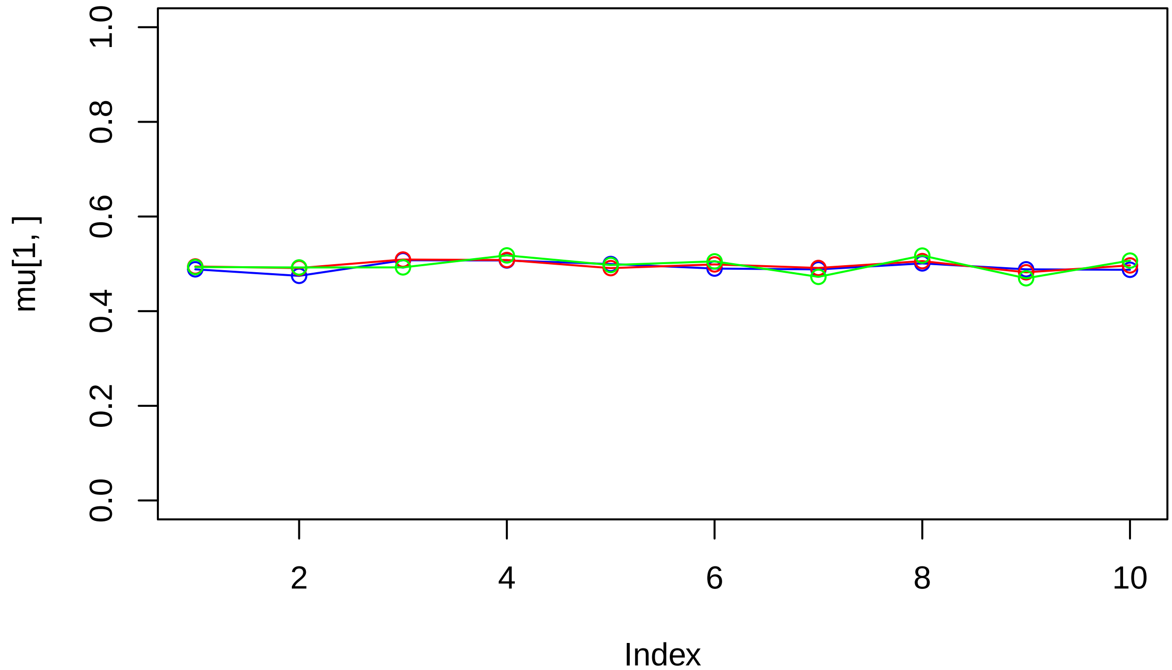




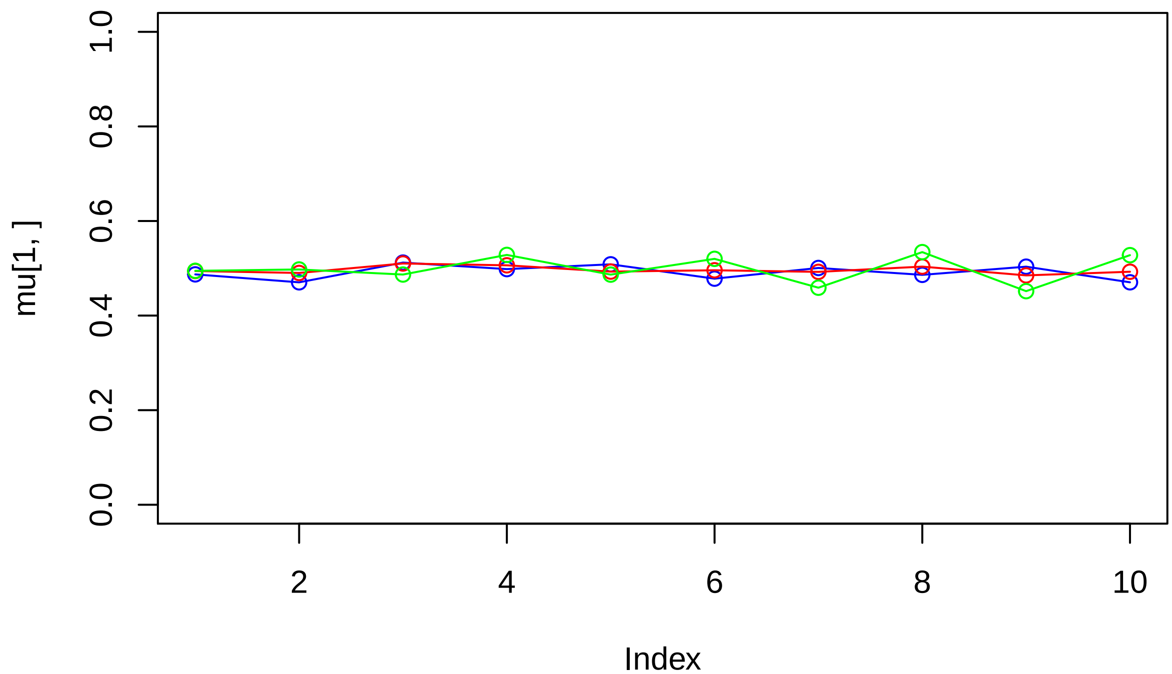

```
## iteration:  1 log likelihood:  -6597.778
```



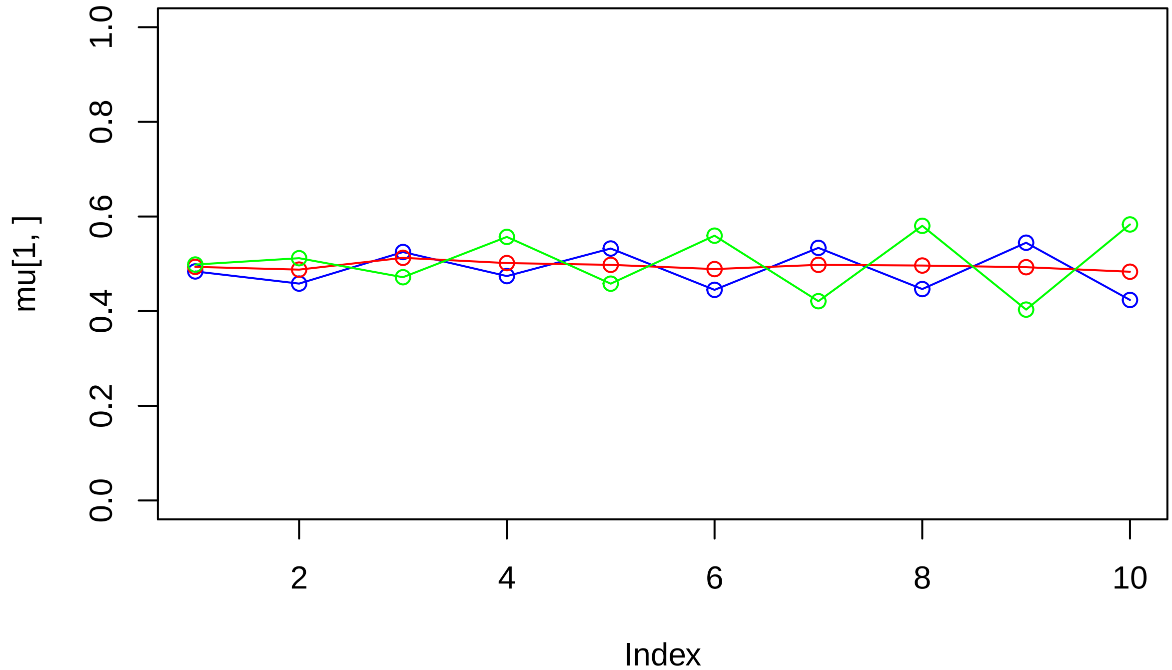

```
## iteration:  2 log likelihood:  -6595.239
```



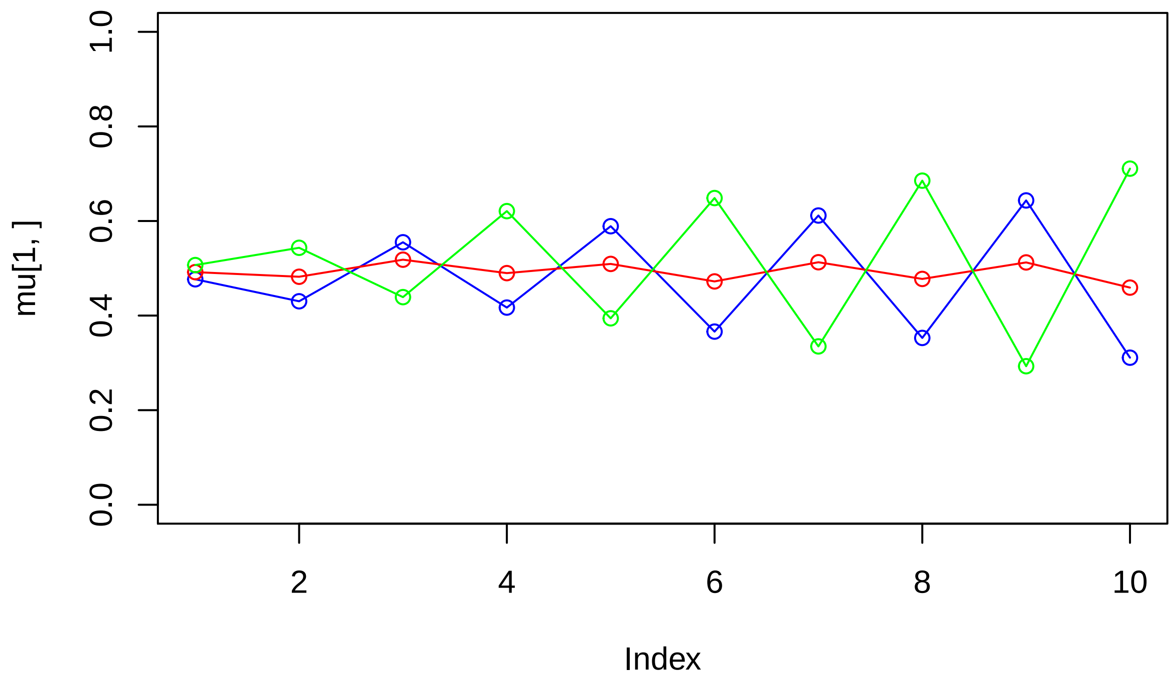

```
## iteration:  3 log likelihood:  -6592.753
```



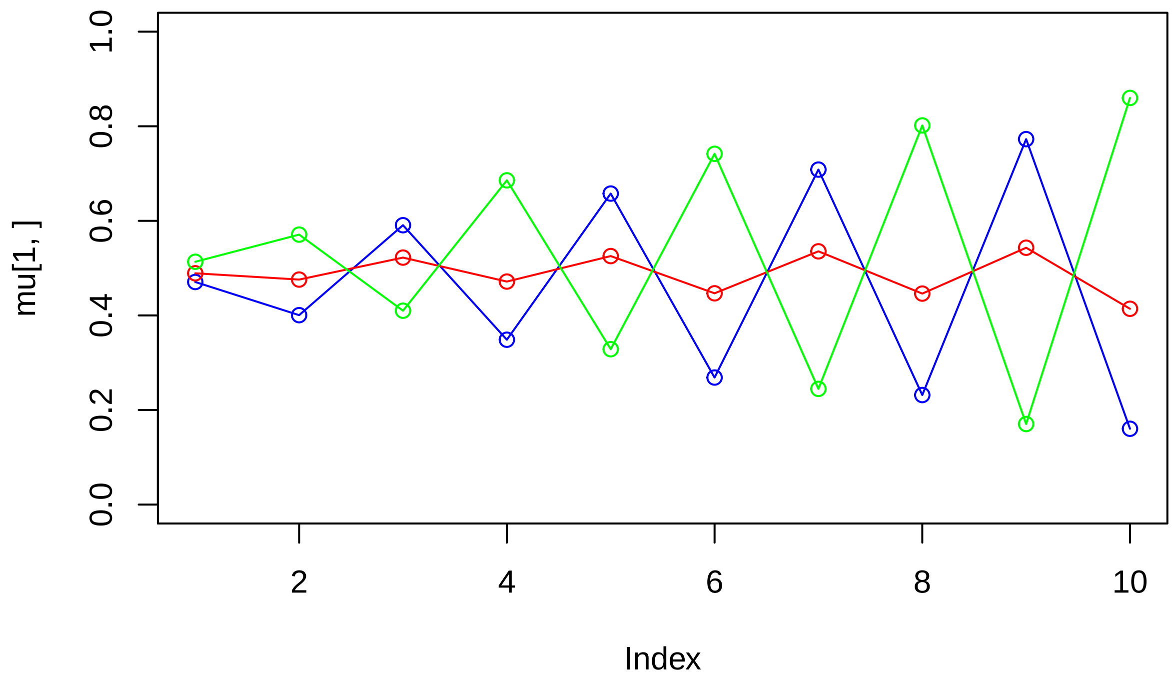

```
## iteration:  4 log likelihood:  -6573.7
```



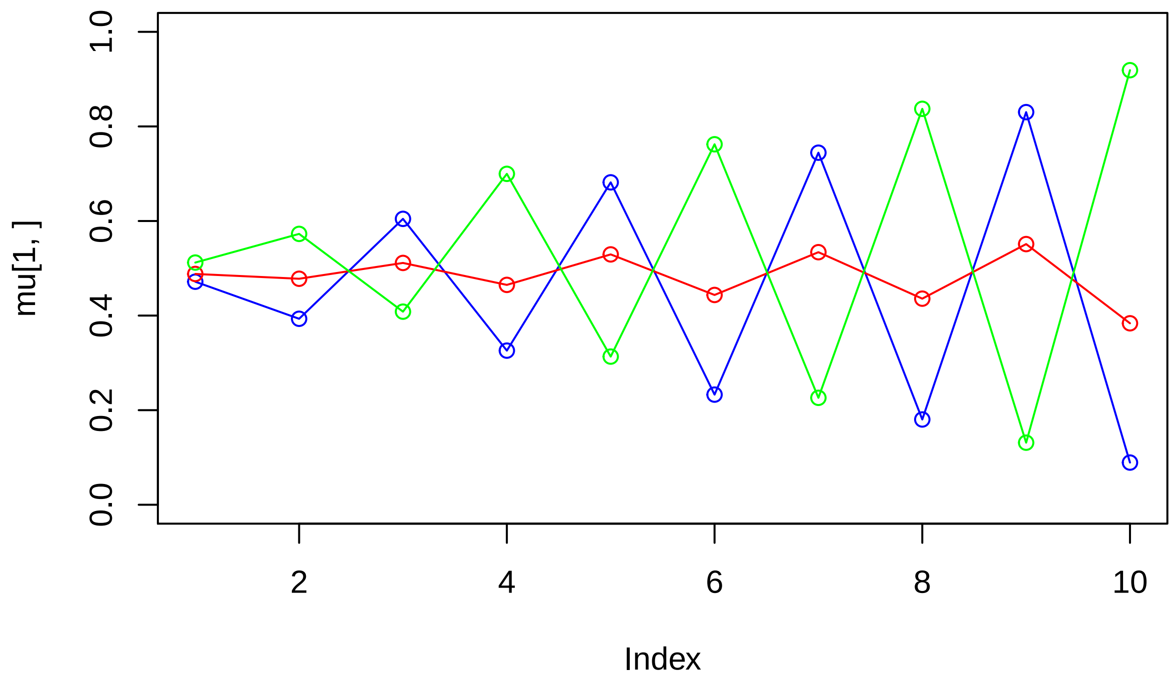
iteration: 5 log likelihood: -6446.022



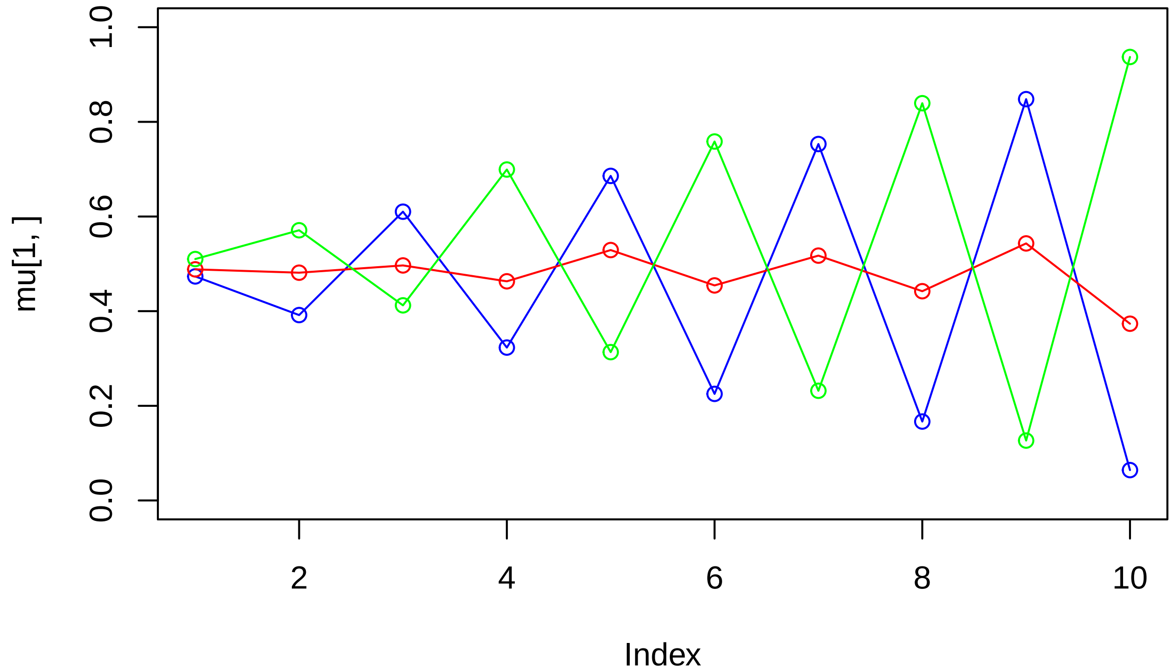
iteration: 6 log likelihood: -5978.865



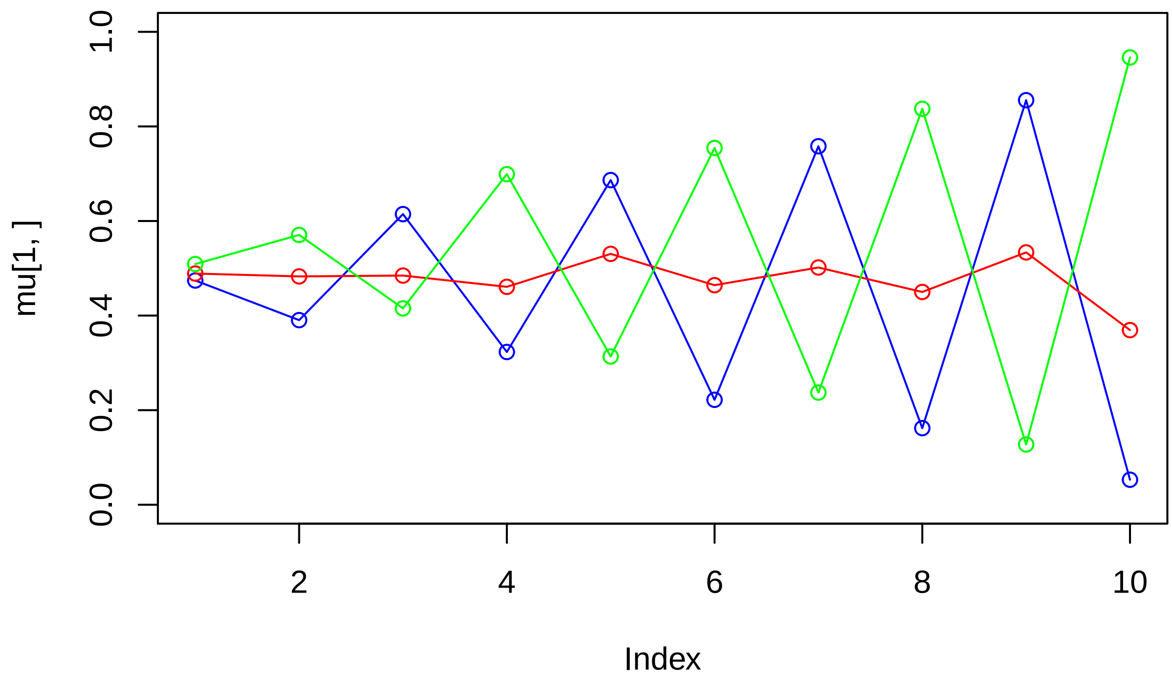

```
## iteration:  7 log likelihood:  -5537.074
```



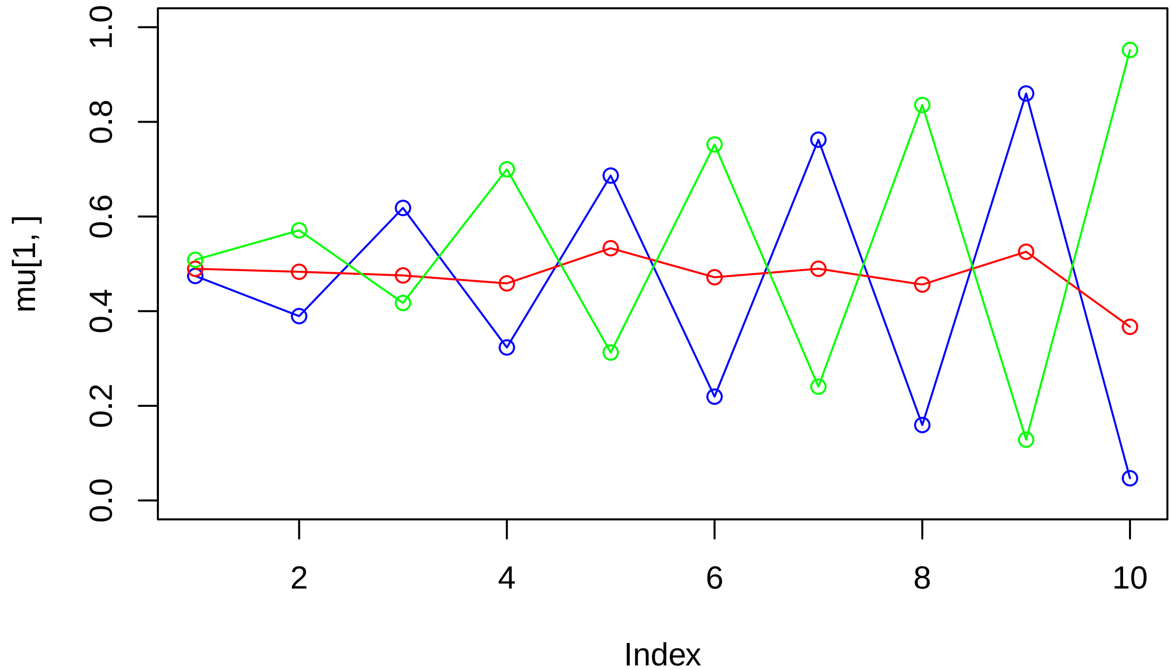

```
## iteration:  8 log likelihood:  -5429.225
```



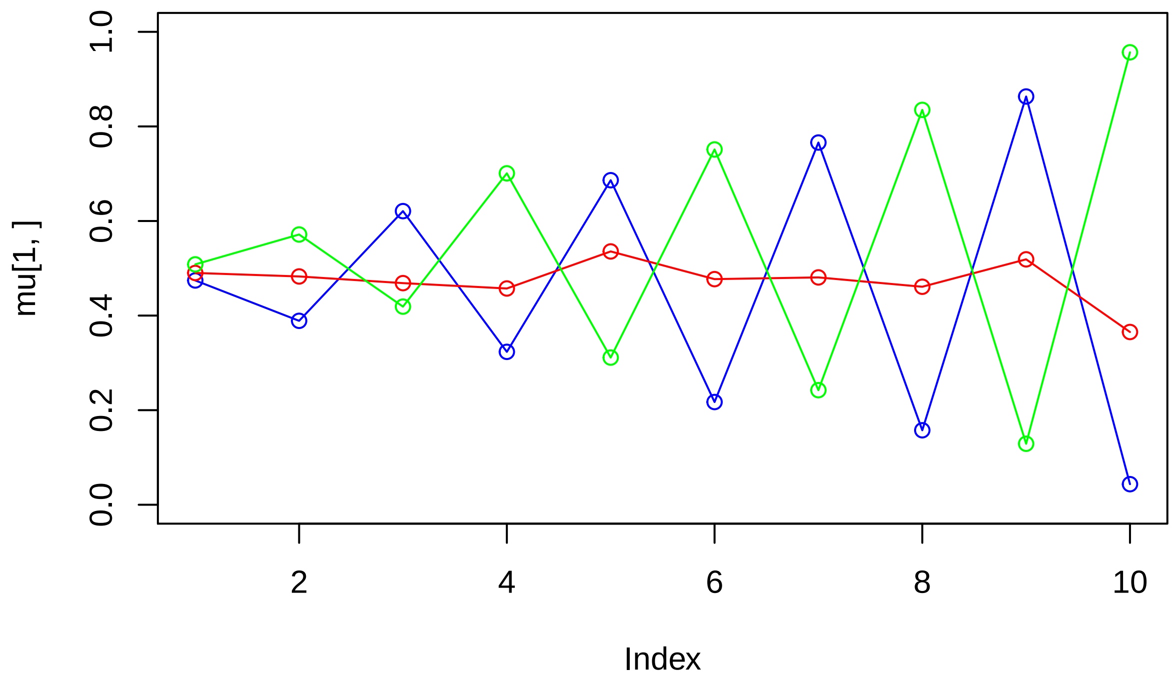

```
## iteration:  9 log likelihood:  -5401.95
```



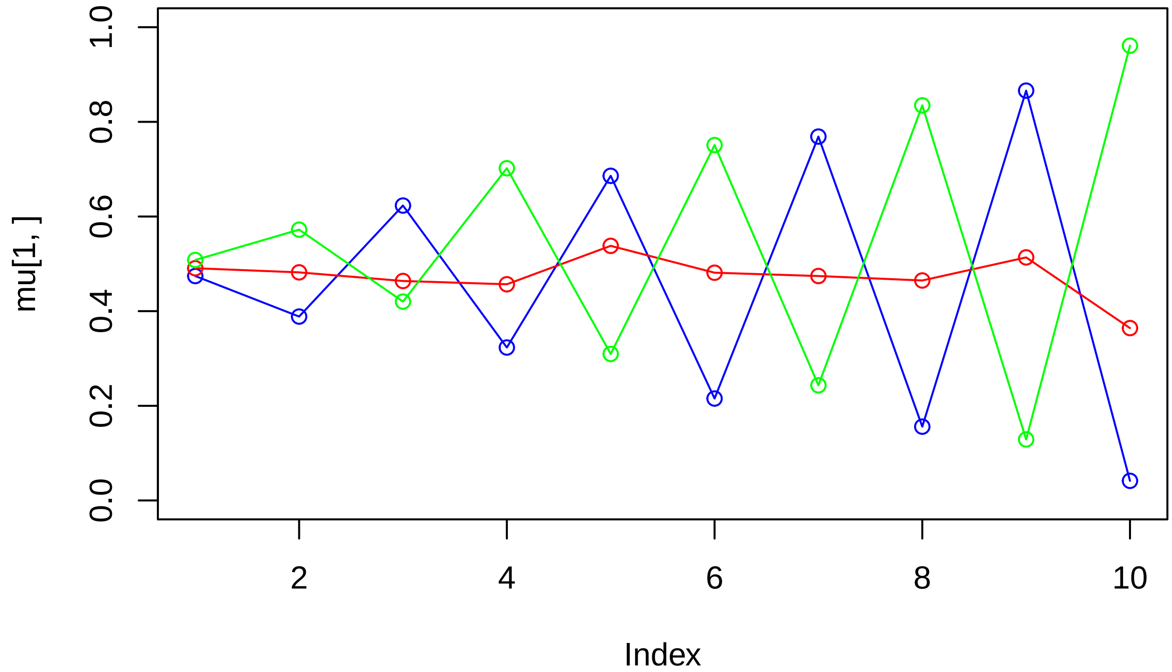

```
## iteration:  10 log likelihood:  -5389.023
```



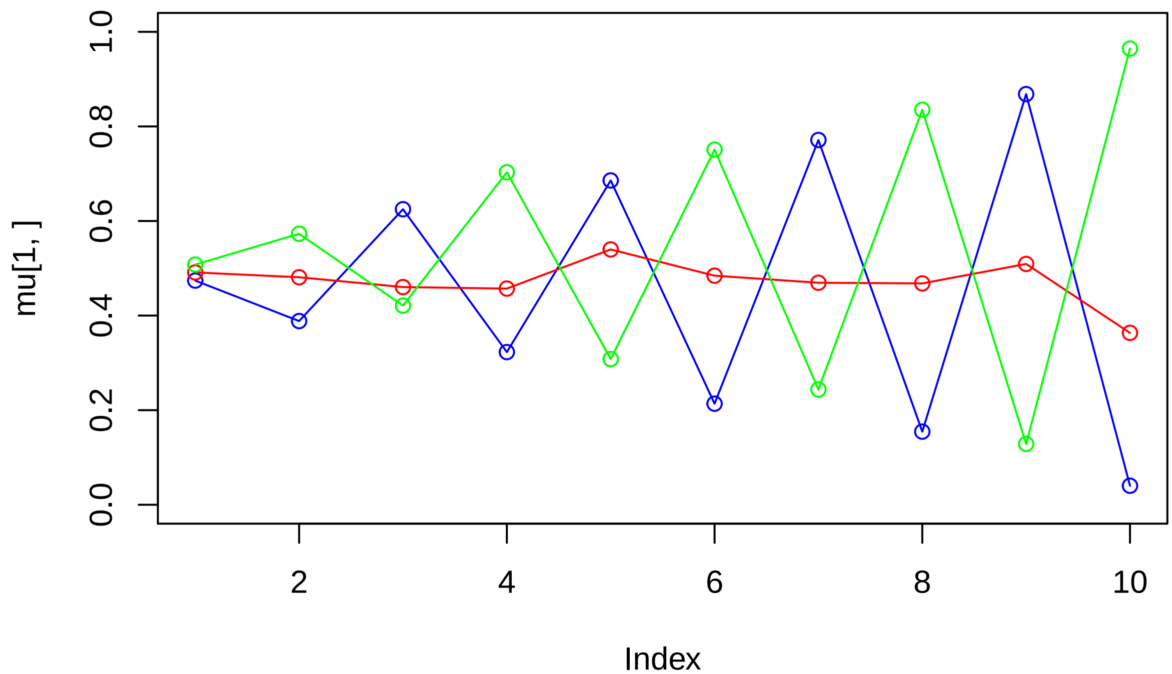

```
## iteration:  11 log likelihood:  -5380.443
```



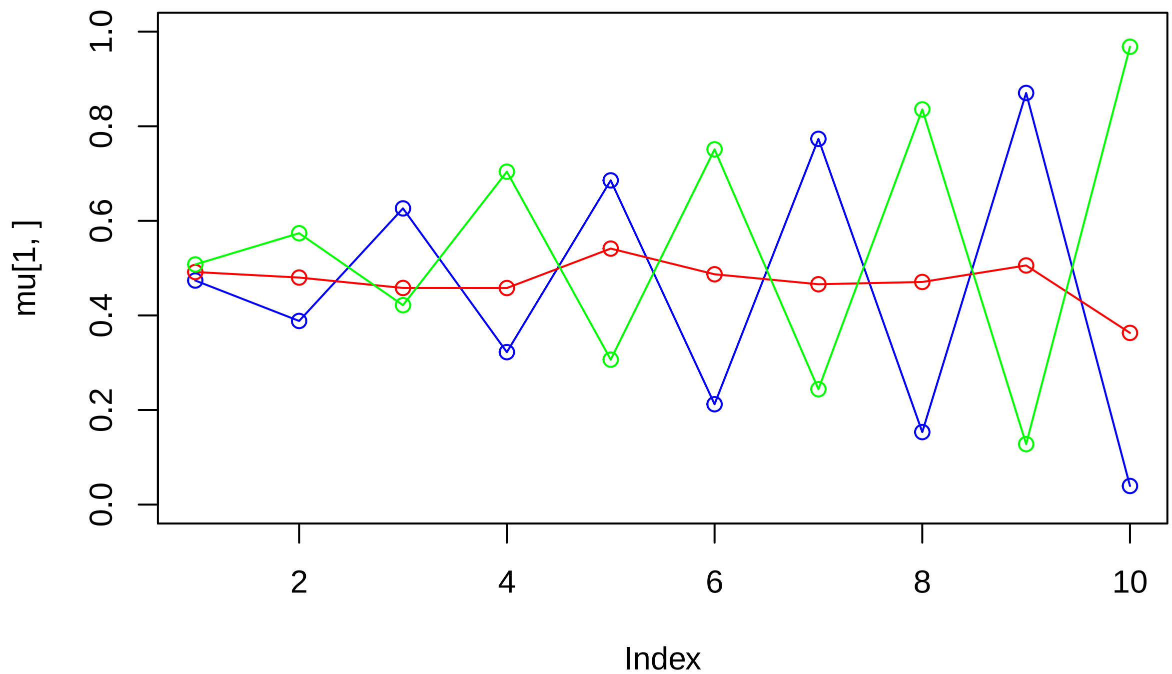

```
## iteration:  12 log likelihood:  -5373.845
```



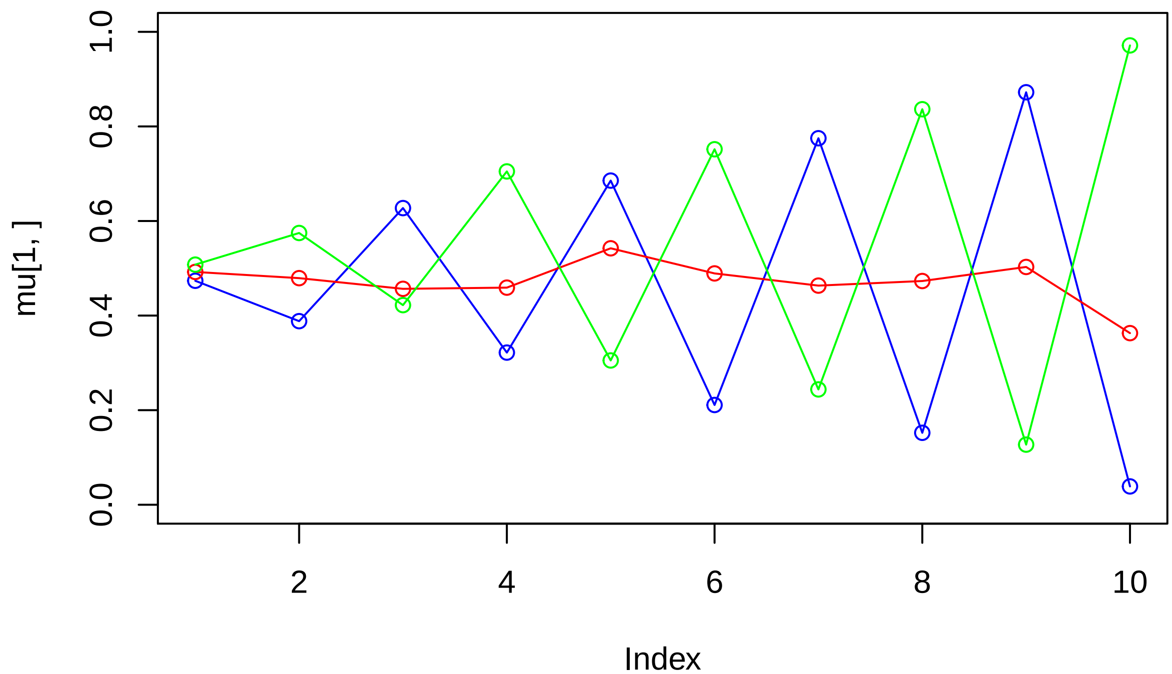

```
## iteration:  13 log likelihood:  -5368.41
```



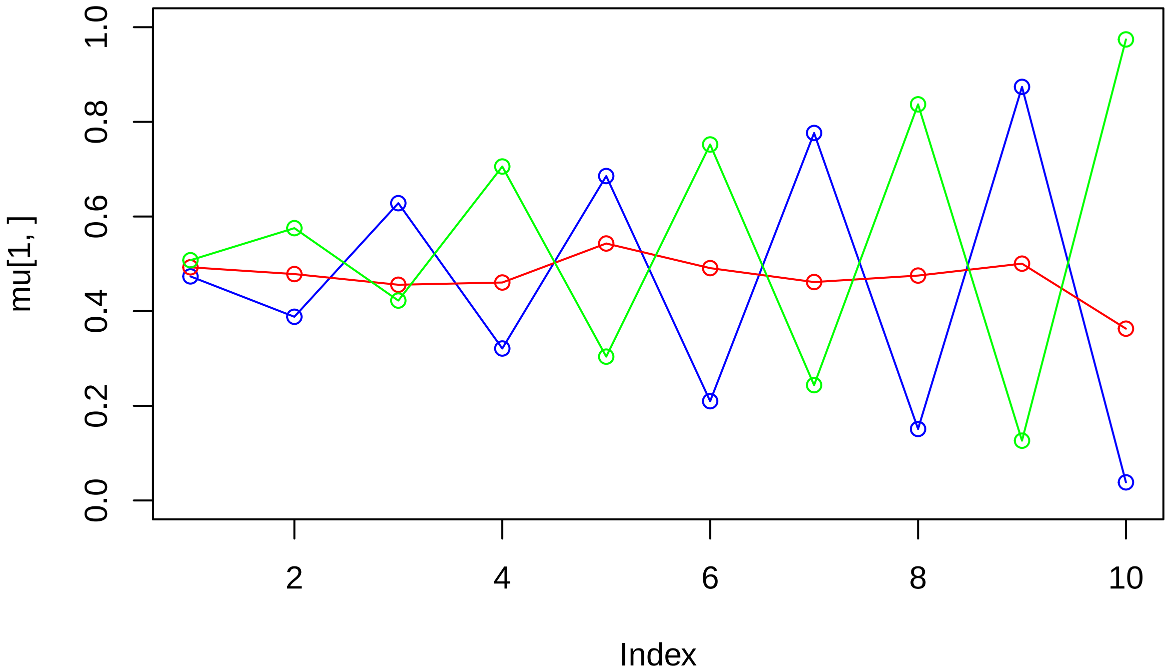

```
## iteration:  14 log likelihood:  -5363.759
```



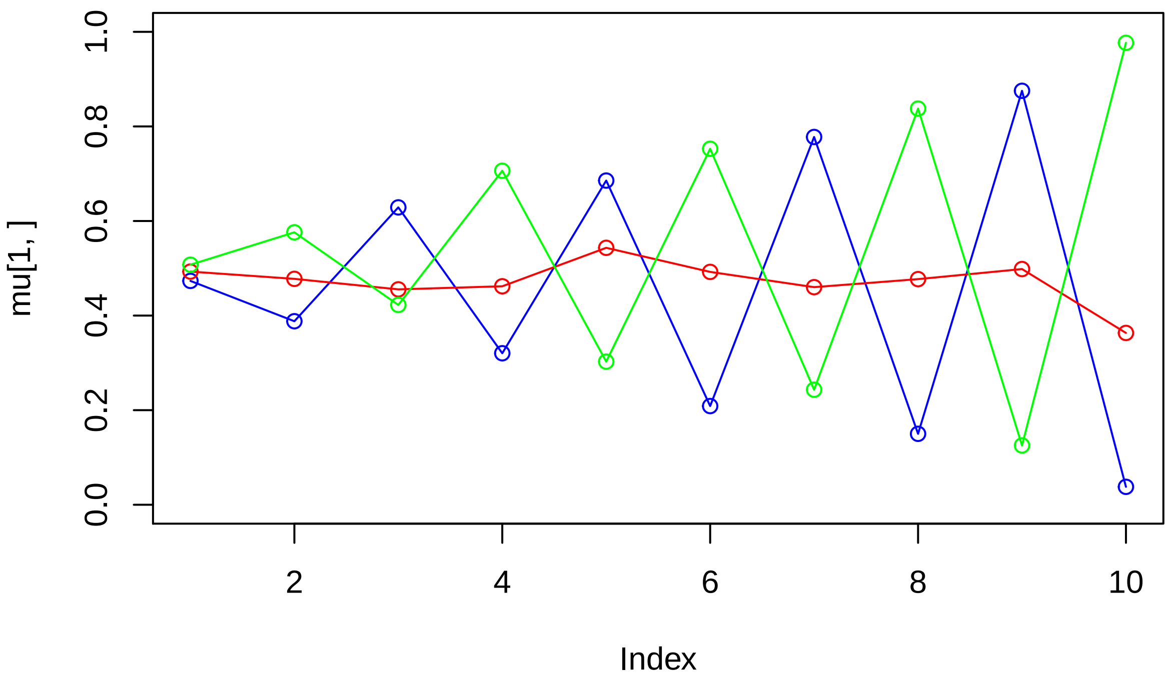

```
## iteration:  15 log likelihood:  -5359.682
```



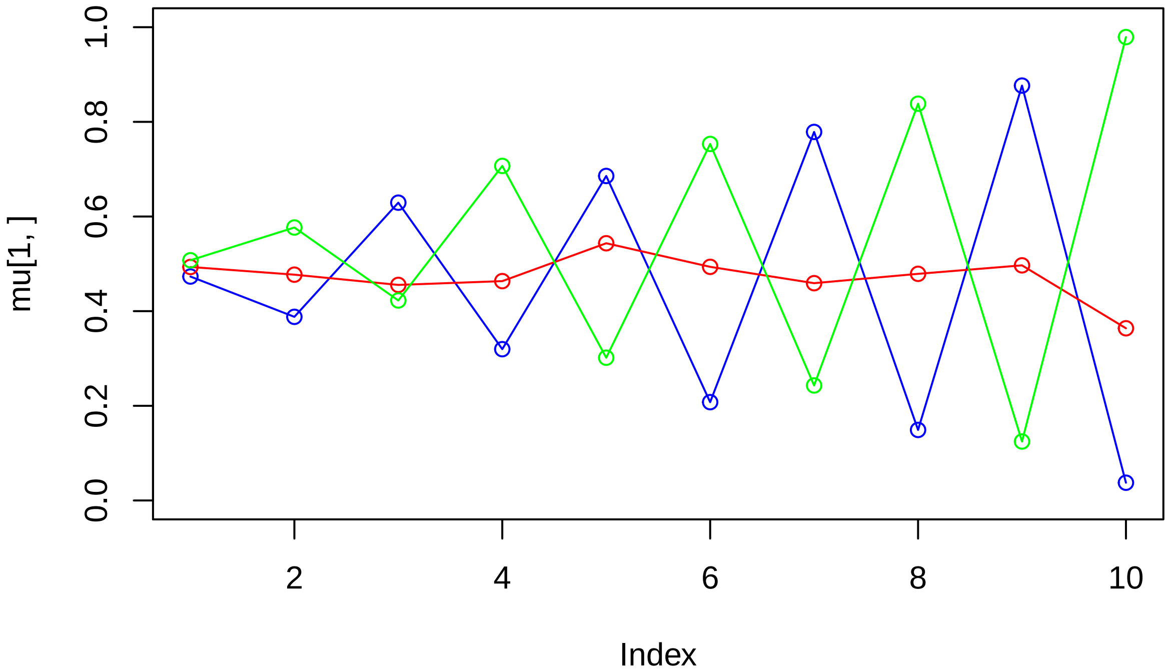

```
## iteration:  16 log likelihood:  -5356.051
```



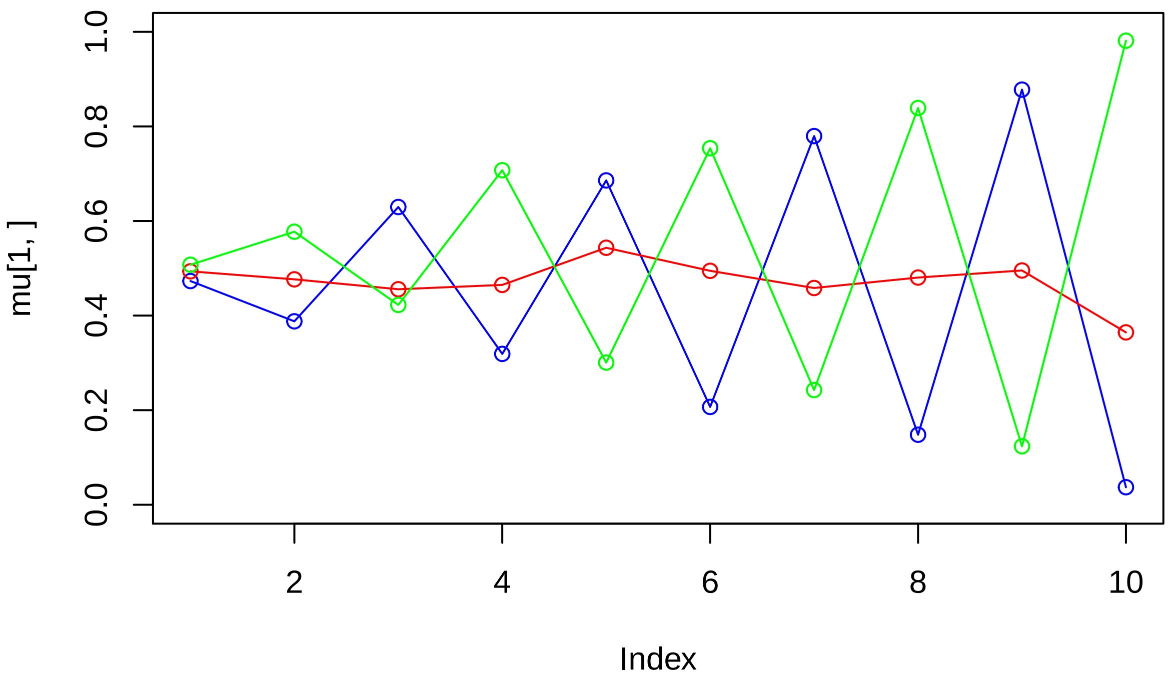

```
## iteration:  17 log likelihood:  -5352.782
```



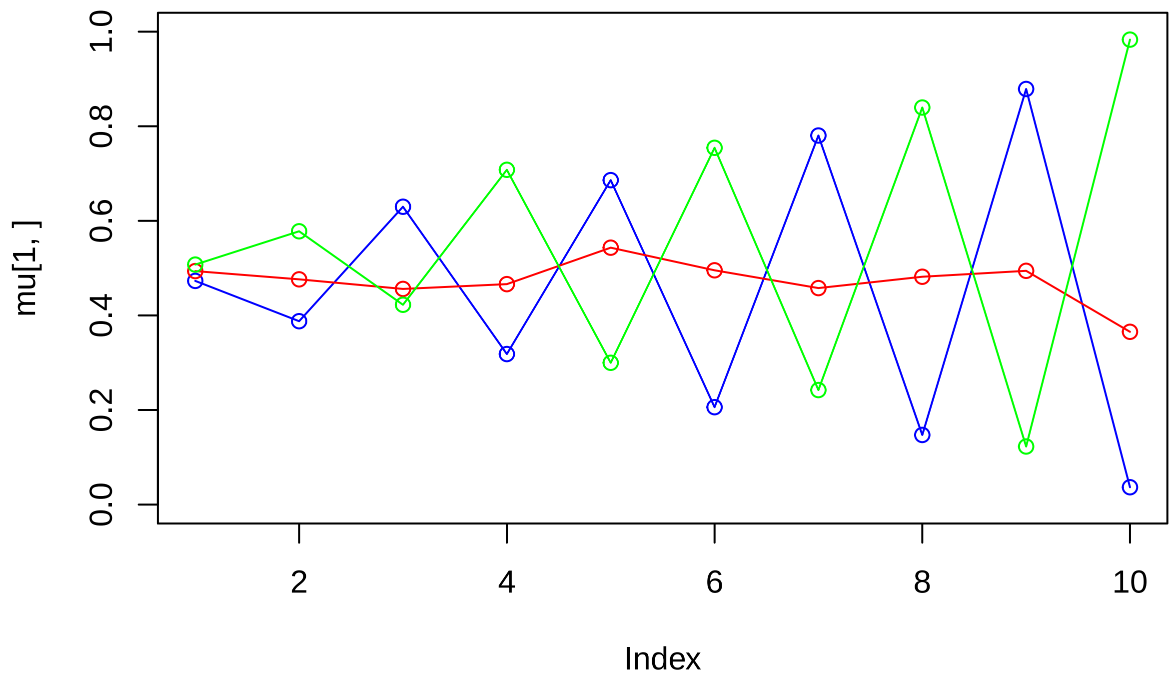

```
## iteration:  18 log likelihood:  -5349.816
```



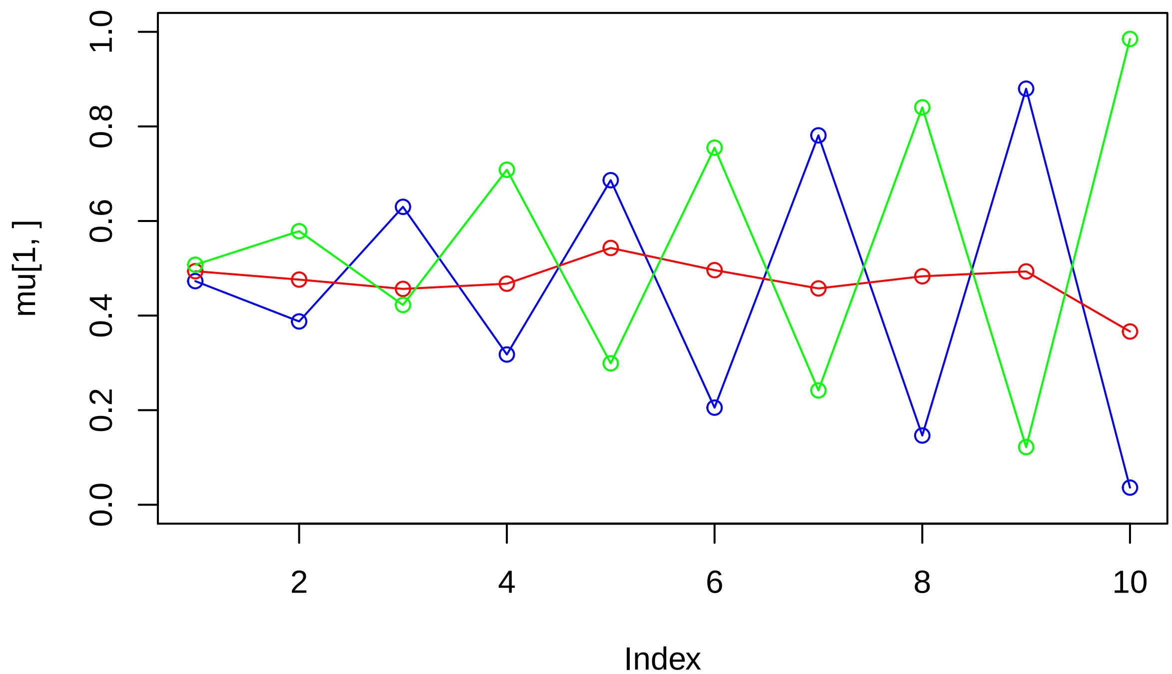

```
## iteration:  19 log likelihood:  -5347.113
```



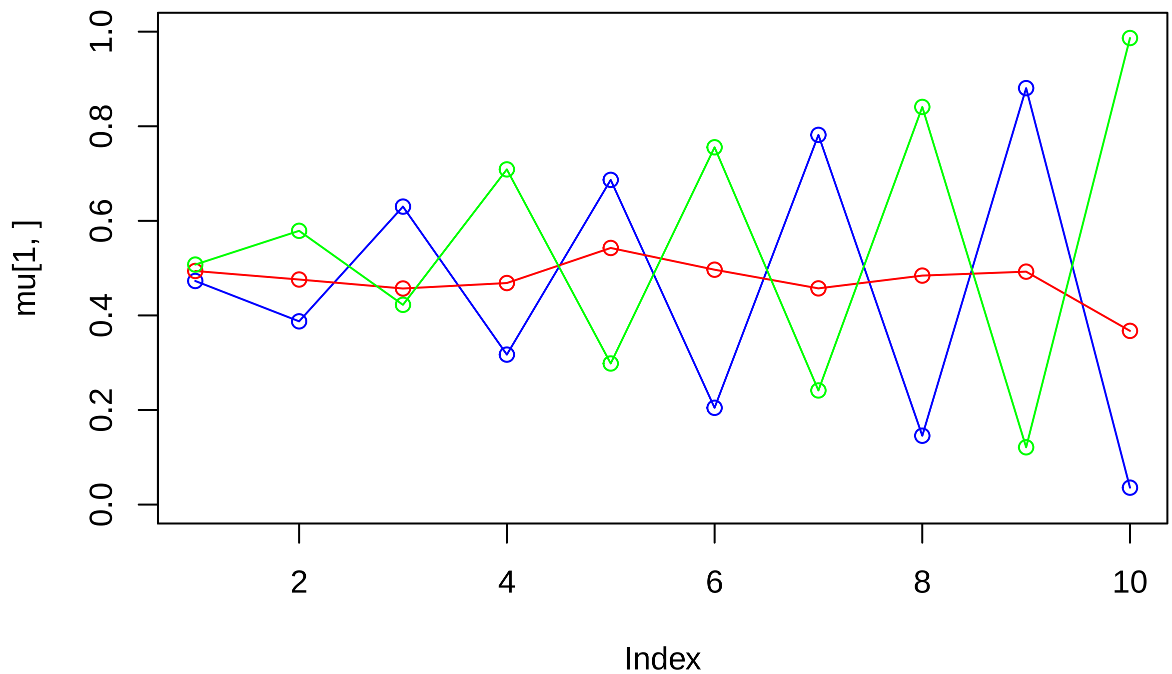

```
## iteration:  20 log likelihood:  -5344.641
```



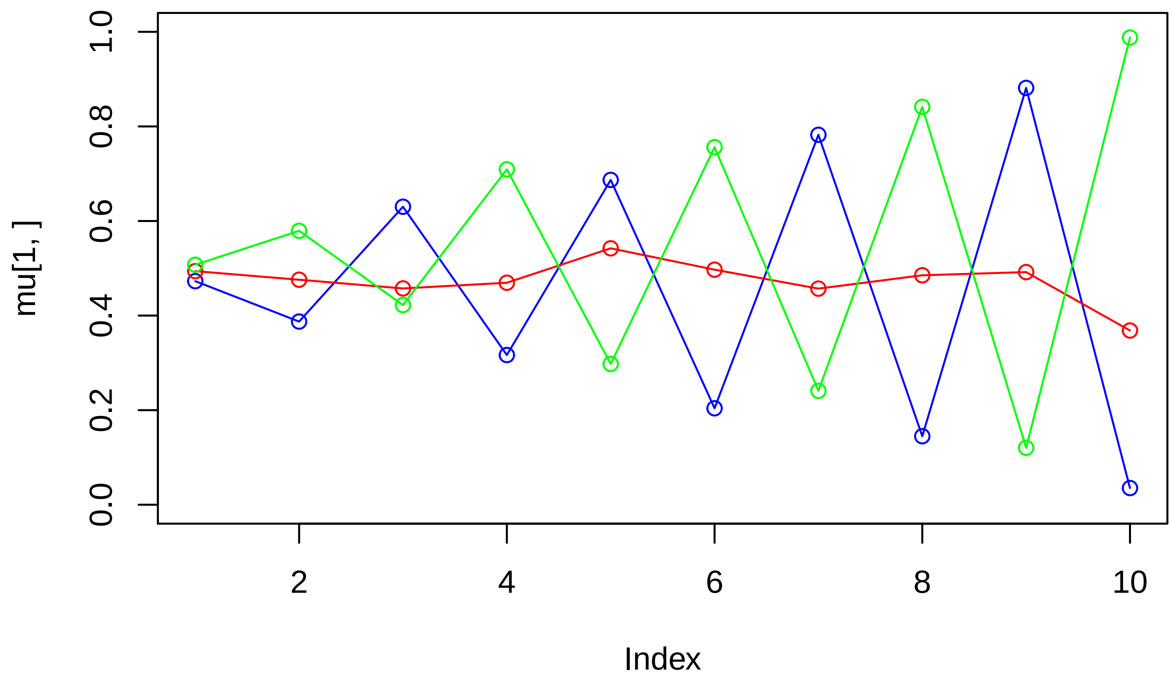

```
## iteration:  21 log likelihood:  -5342.375
```



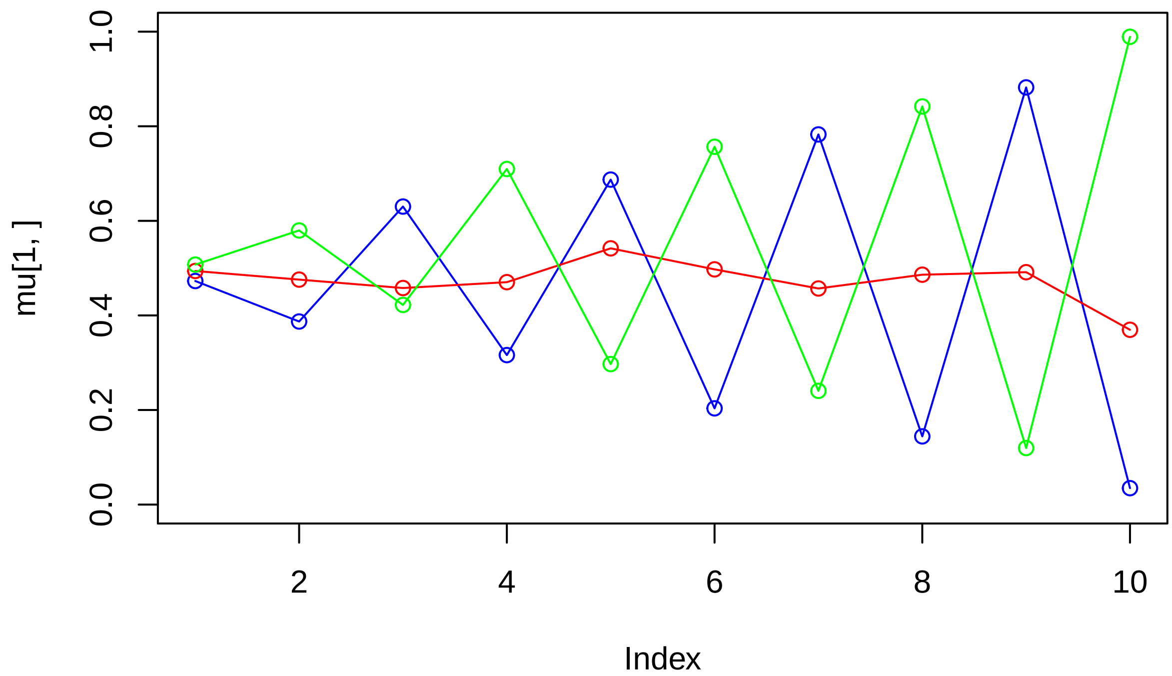

```
## iteration:  22 log likelihood:  -5340.295
```



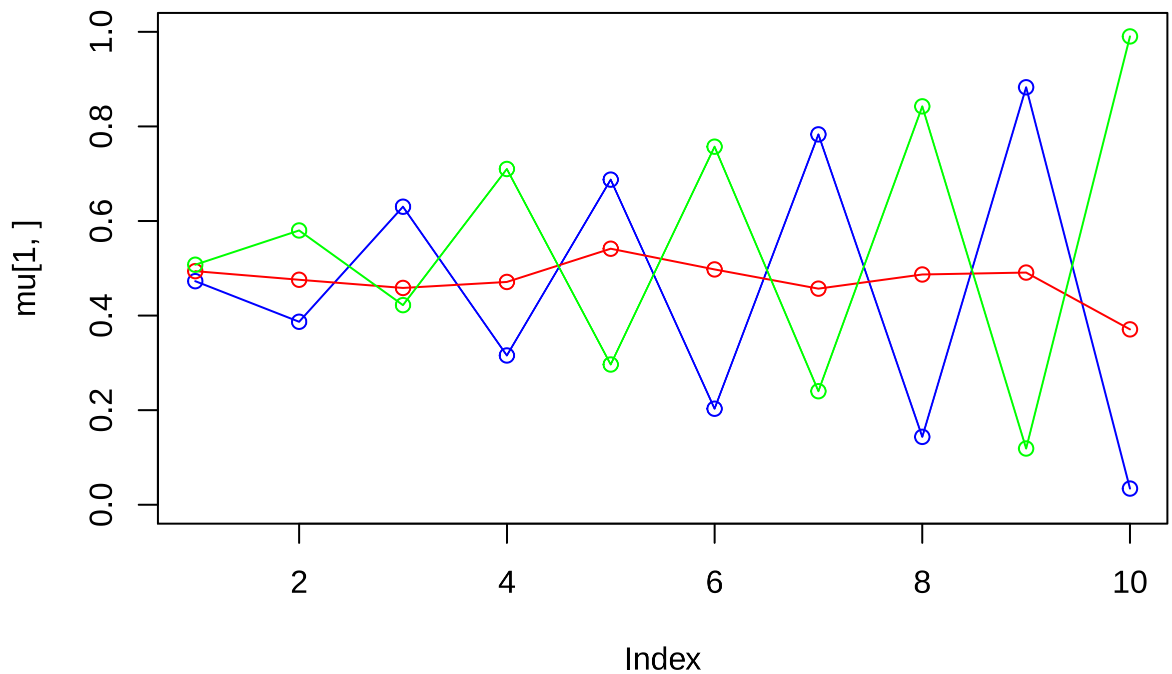

```
## iteration:  23 log likelihood:  -5338.385
```



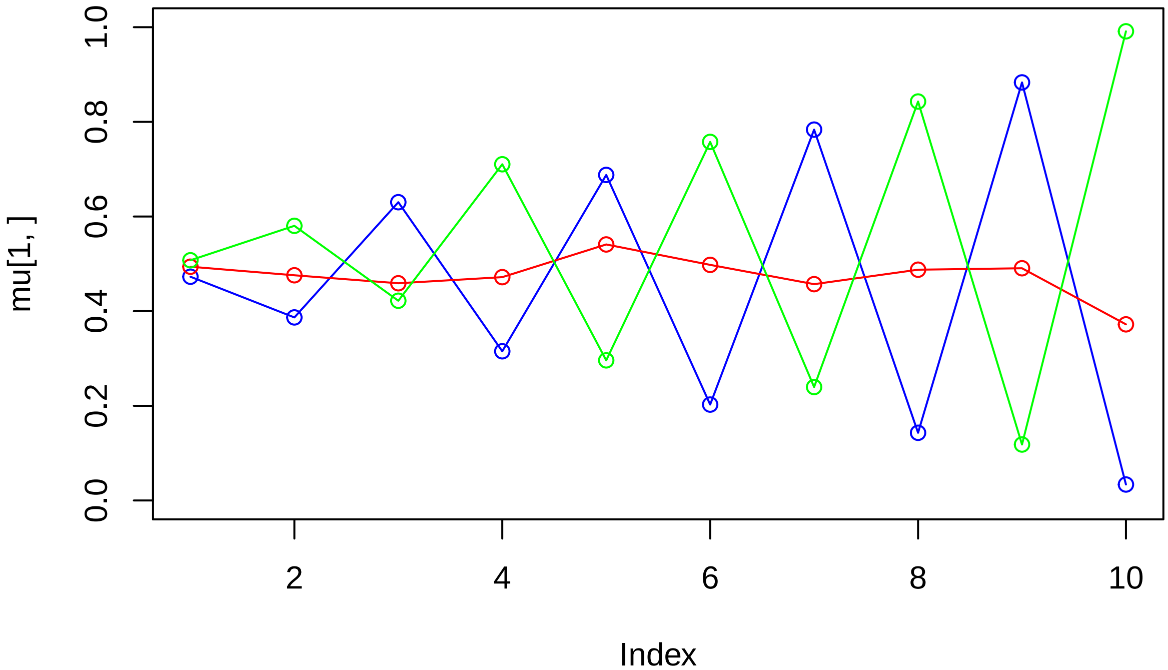

```
## iteration:  24 log likelihood:  -5336.63
```



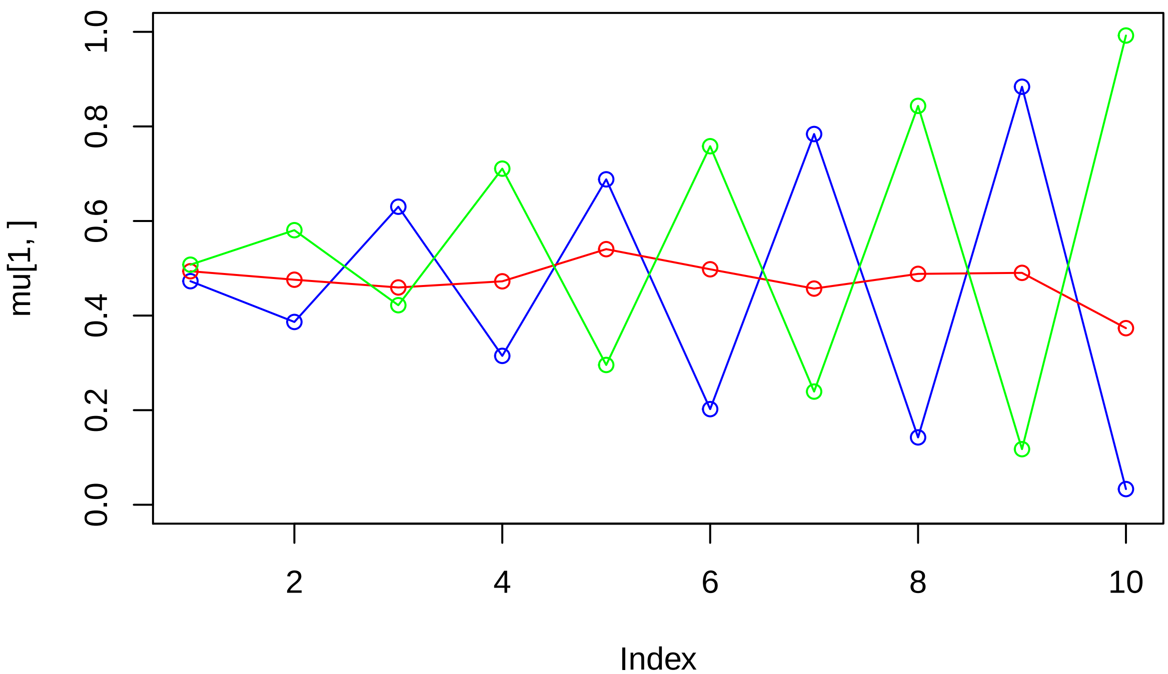

```
## iteration:  25 log likelihood:  -5335.015
```



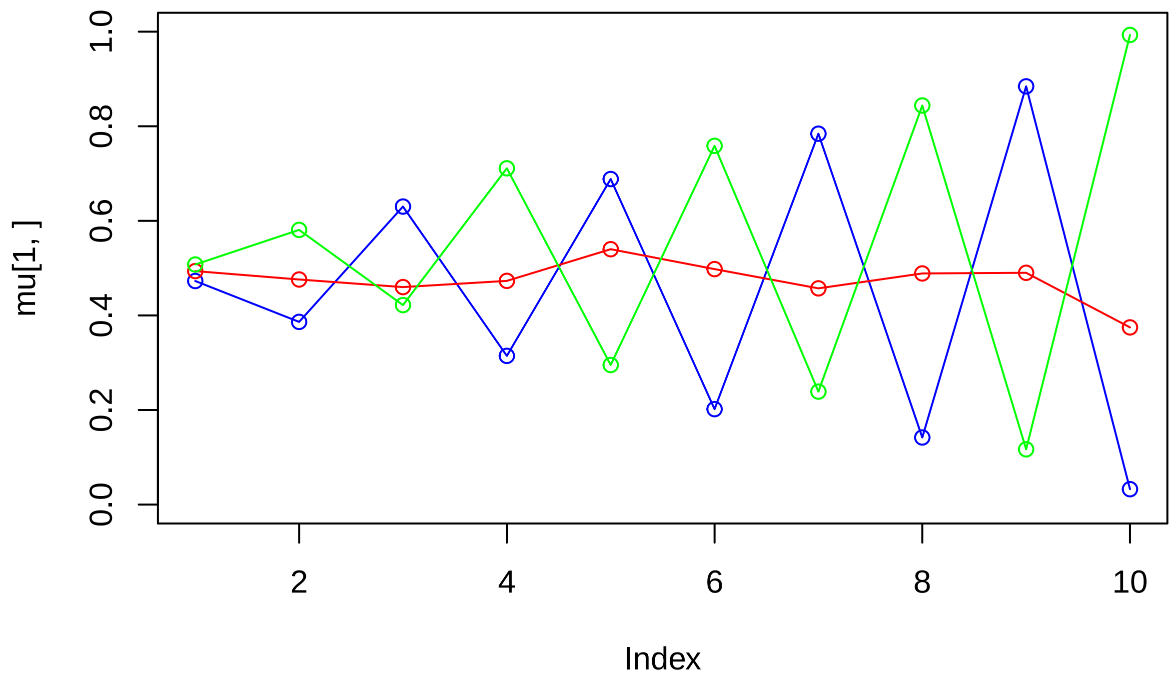

```
## iteration:  26 log likelihood:  -5333.529
```



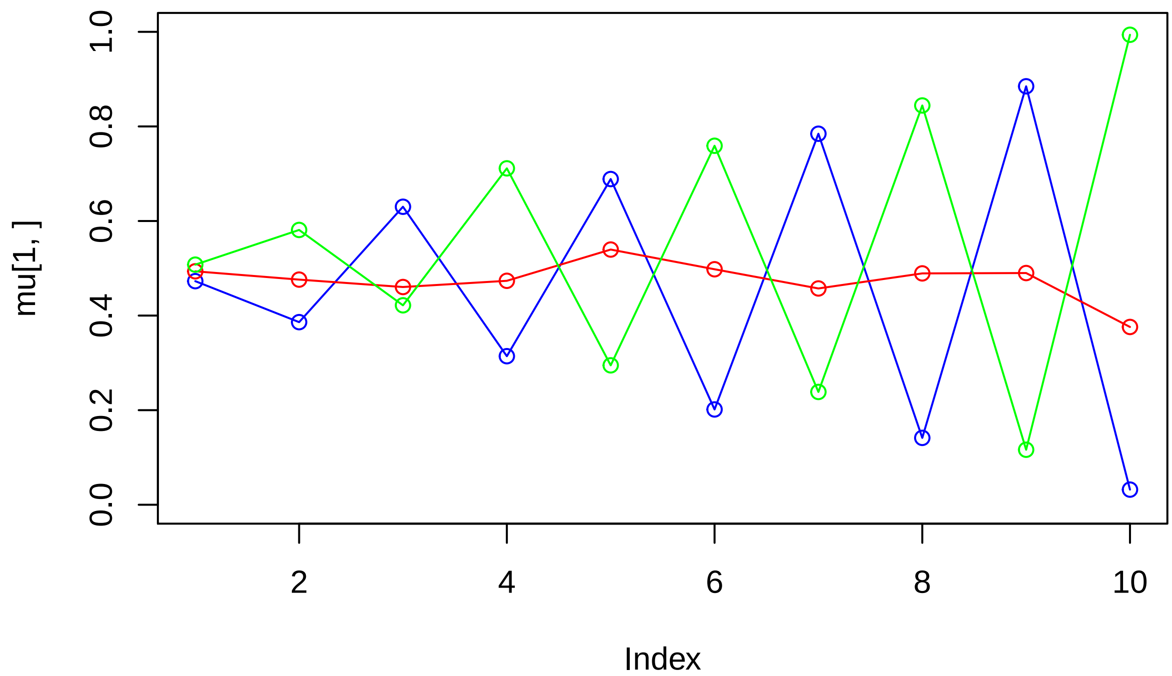

```
## iteration:  27 log likelihood:  -5332.16
```



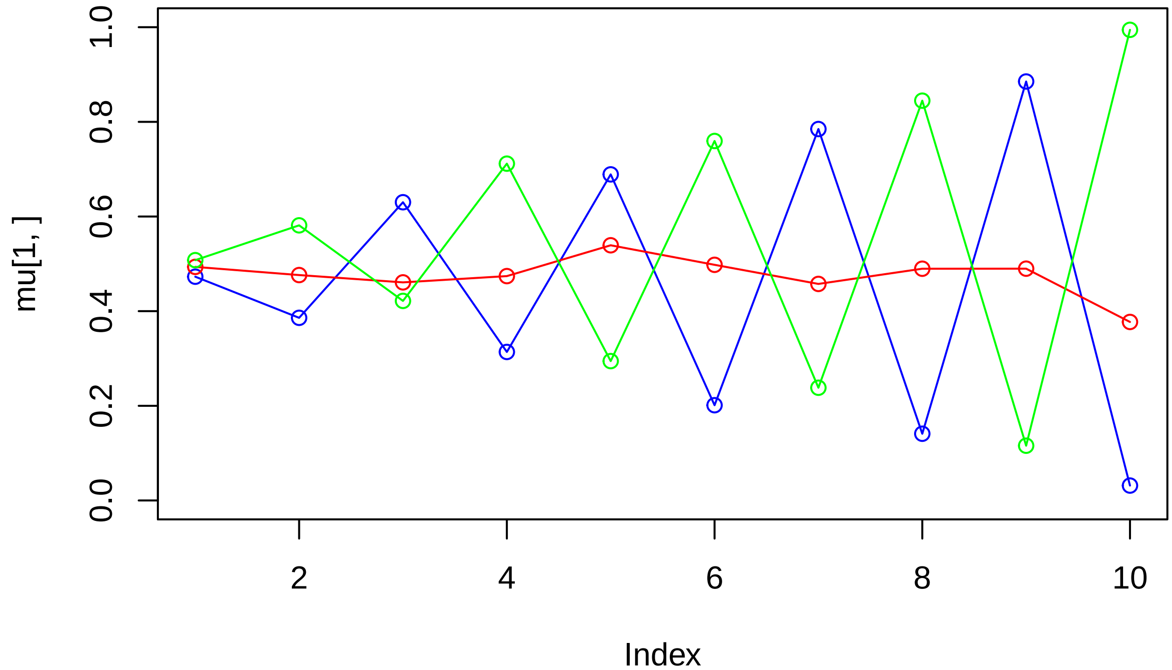

```
## iteration:  28 log likelihood:  -5330.9
```



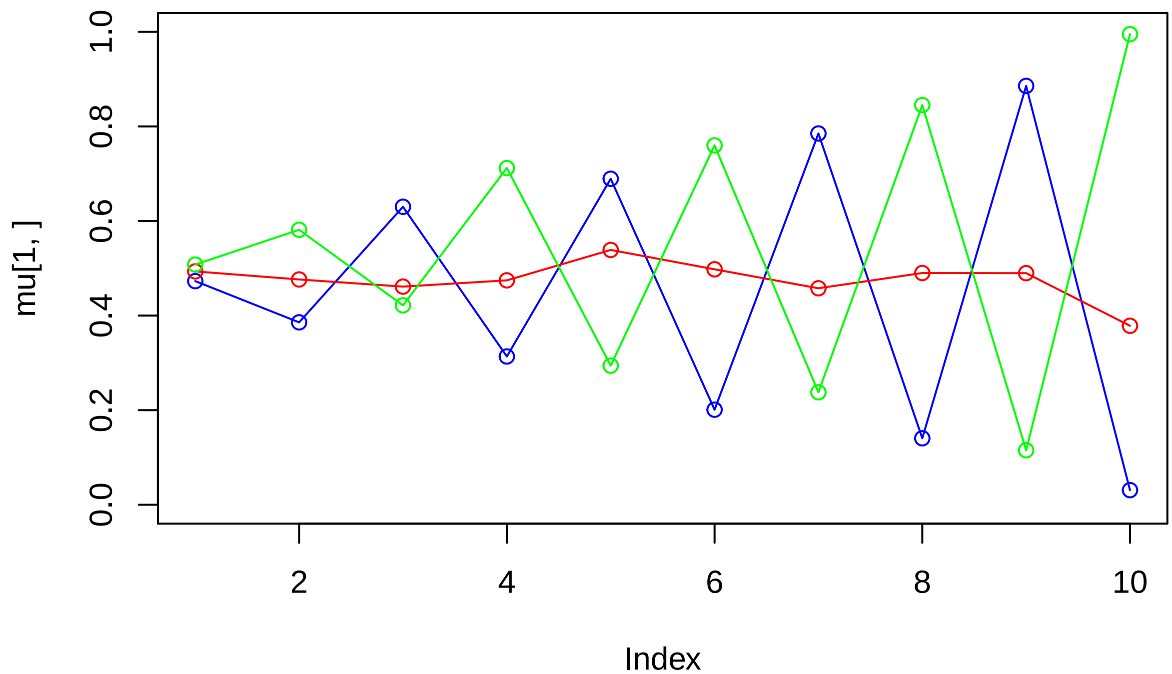

```
## iteration:  29 log likelihood:  -5329.738
```



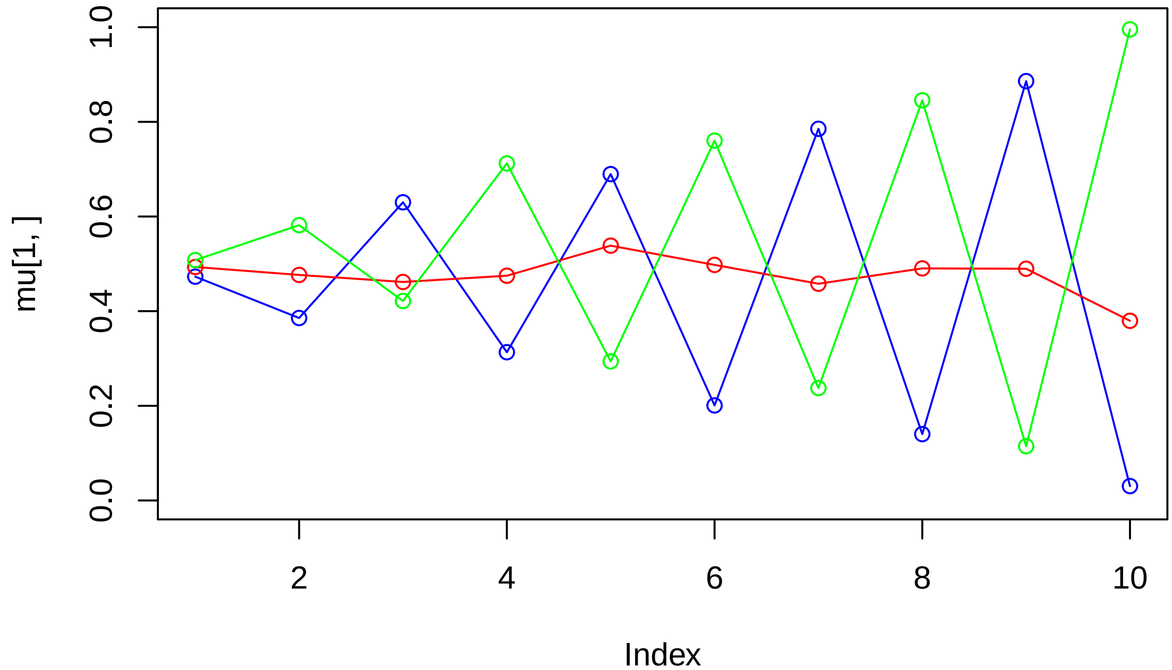

```
## iteration:  30 log likelihood:  -5328.666
```



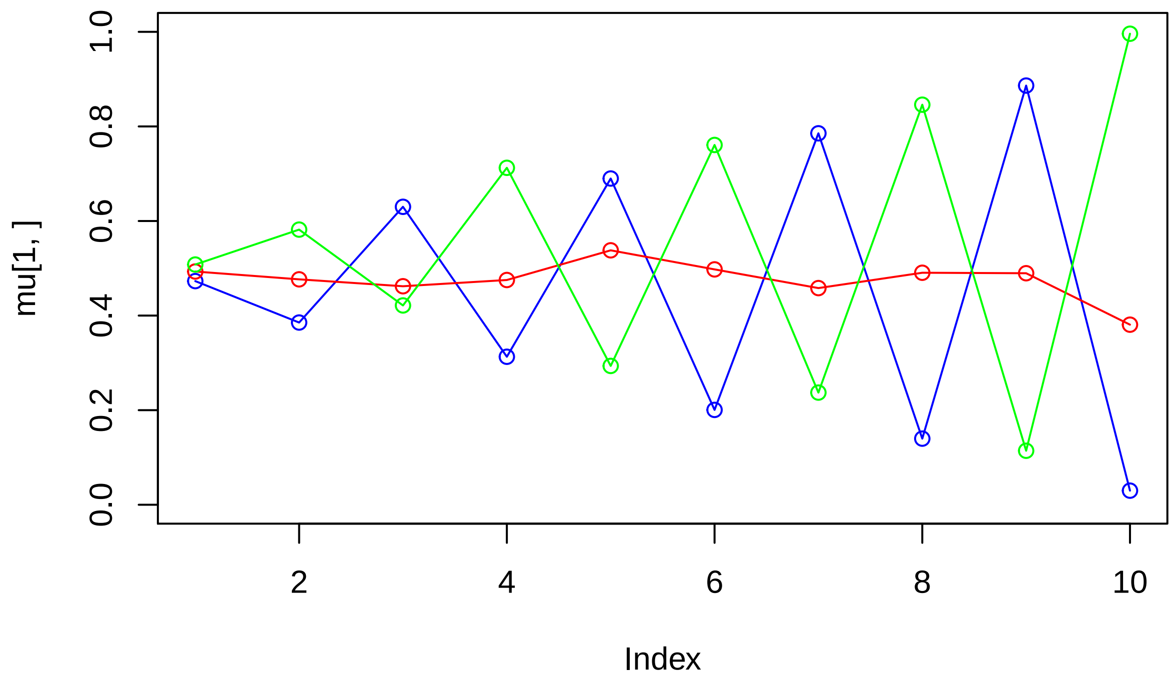

```
## iteration:  31 log likelihood:  -5327.676
```



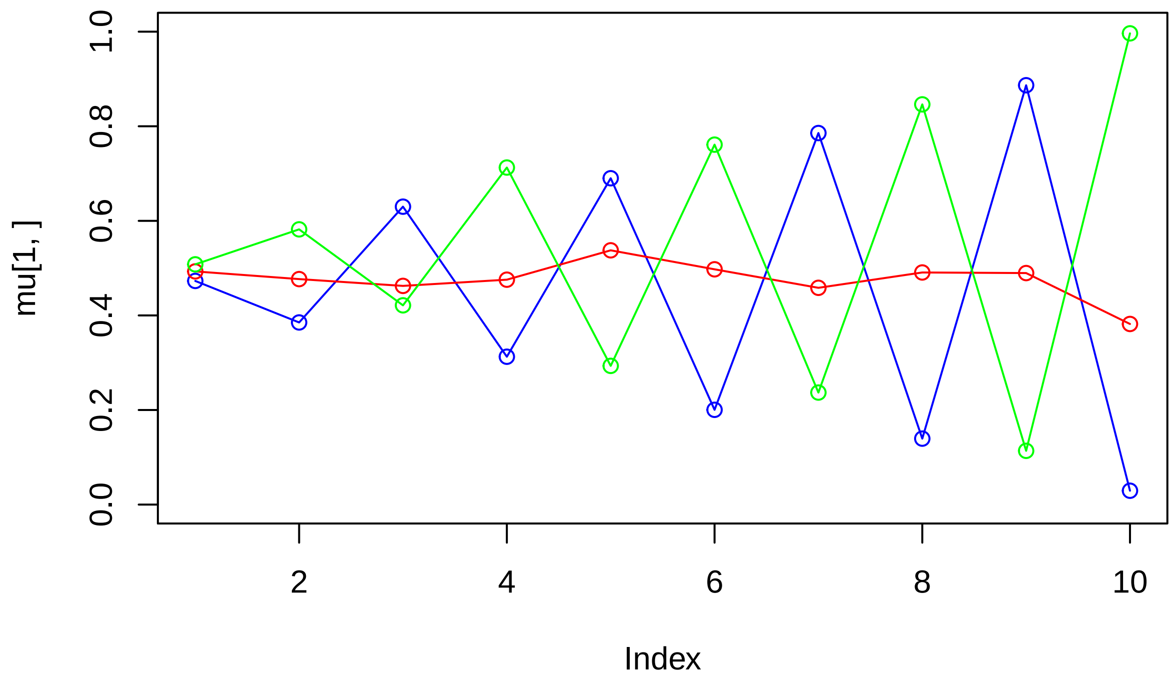

```
## iteration:  32 log likelihood:  -5326.762
```



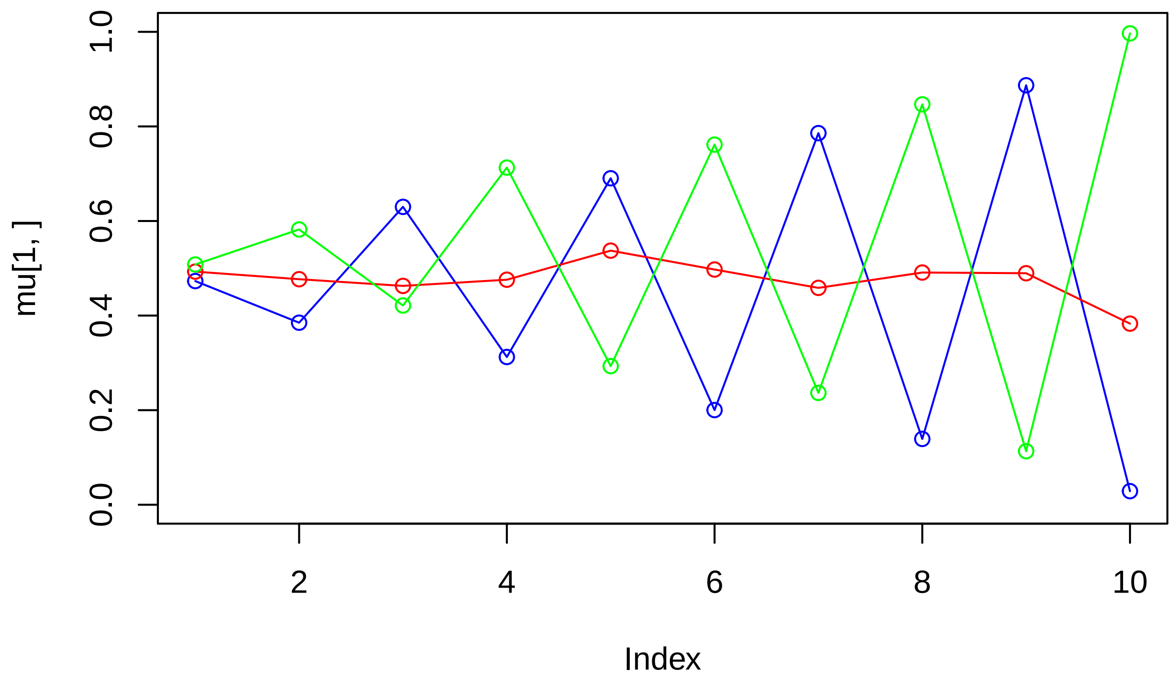

```
## iteration:  33 log likelihood:  -5325.917
```



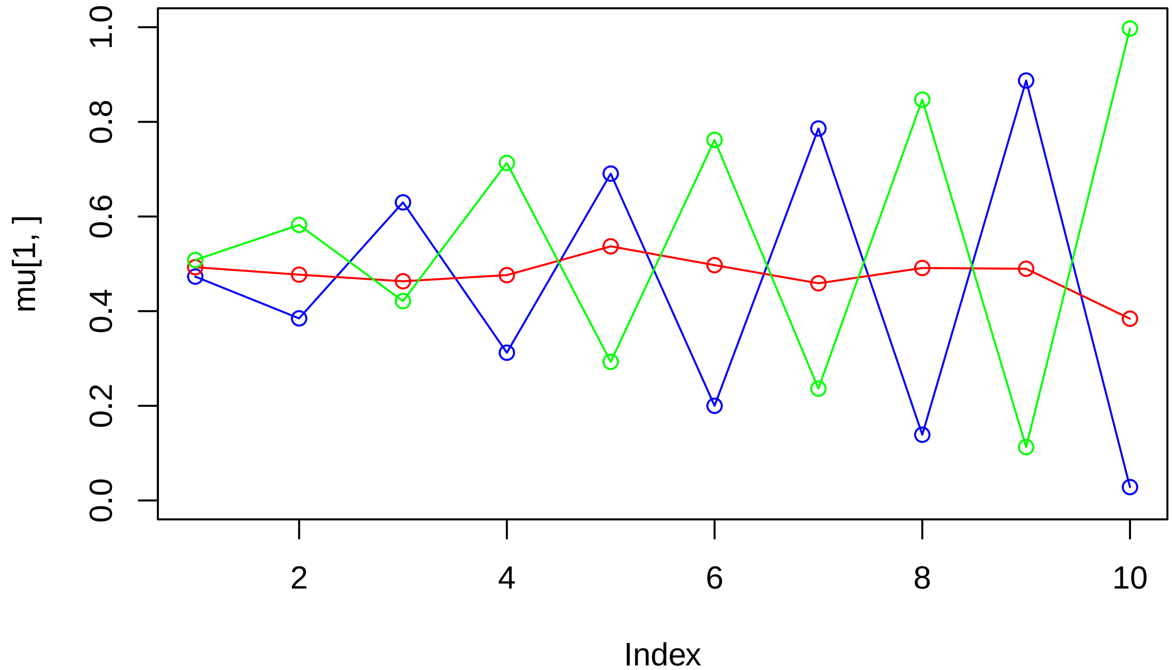

```
## iteration:  34 log likelihood:  -5325.135
```



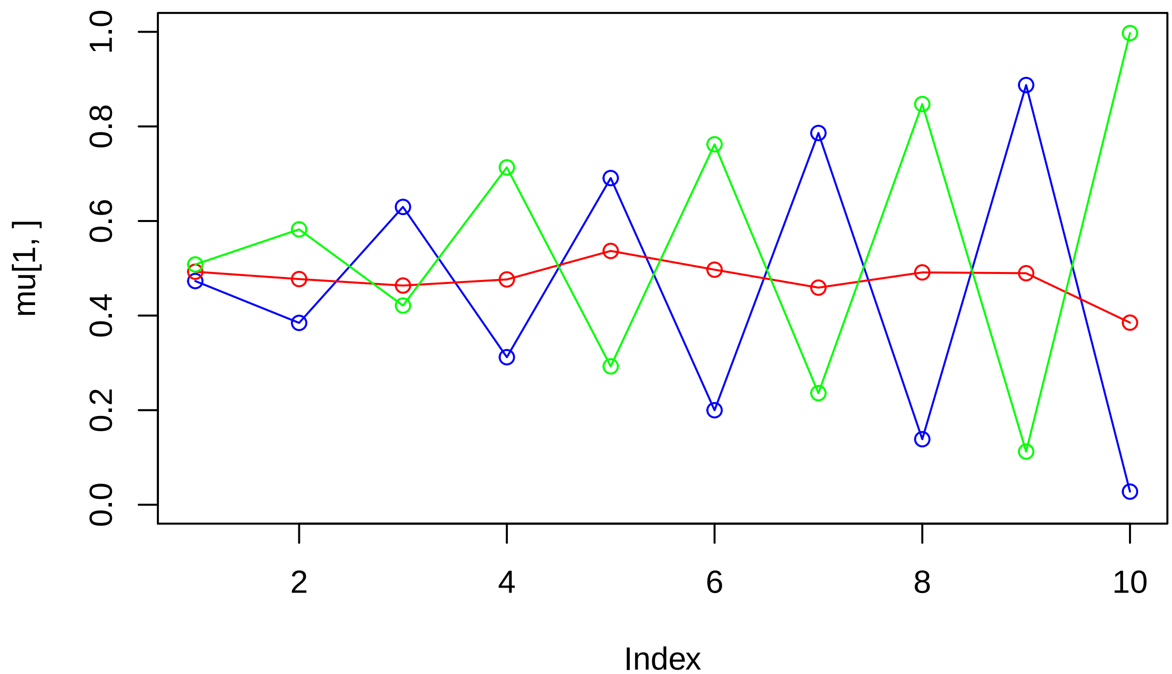

```
## iteration:  35 log likelihood:  -5324.41
```



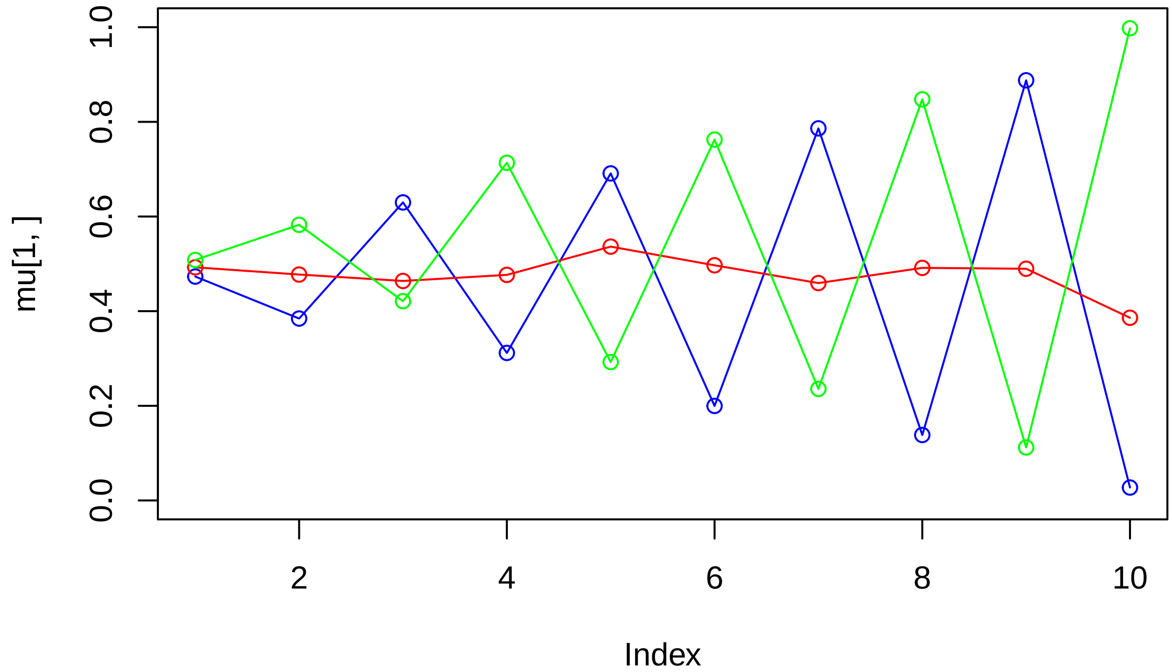

```
## iteration:  36 log likelihood:  -5323.739
```



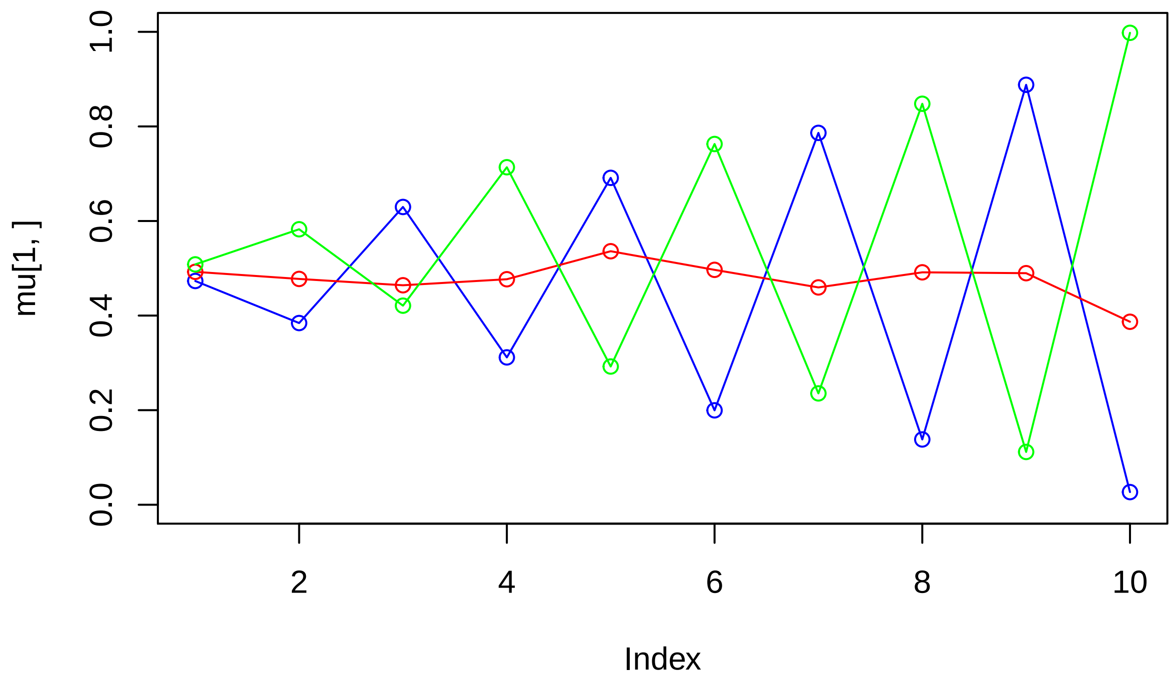

```
## iteration:  37 log likelihood:  -5323.115
```



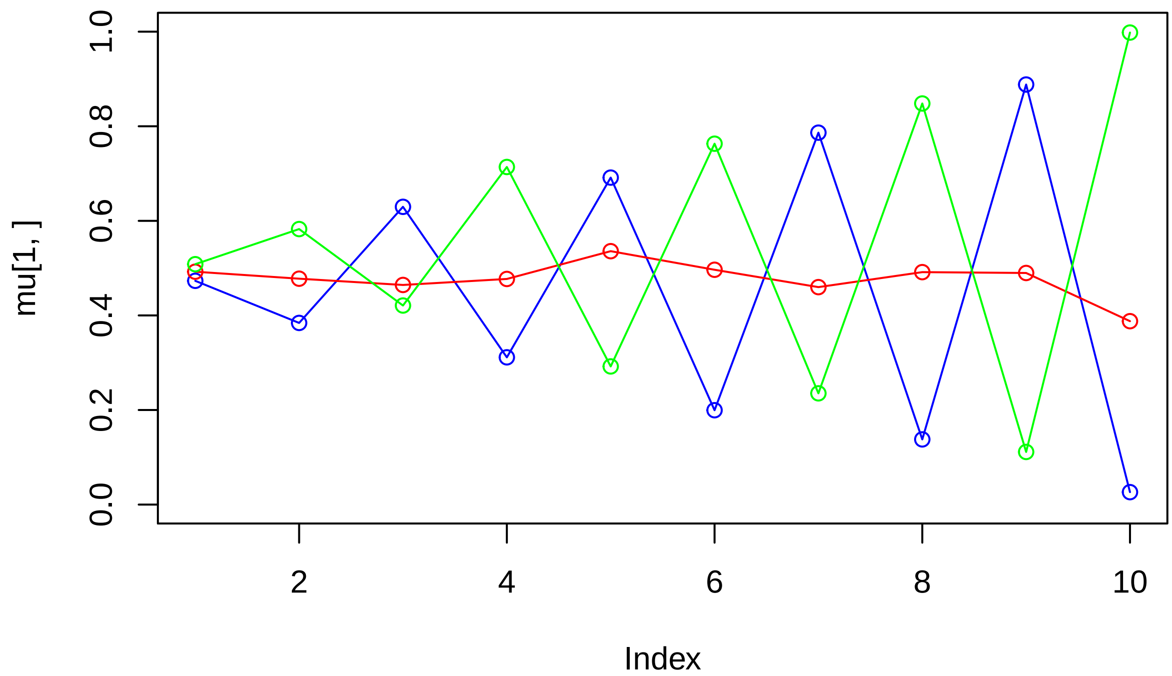

```
## iteration:  38 log likelihood:  -5322.537
```



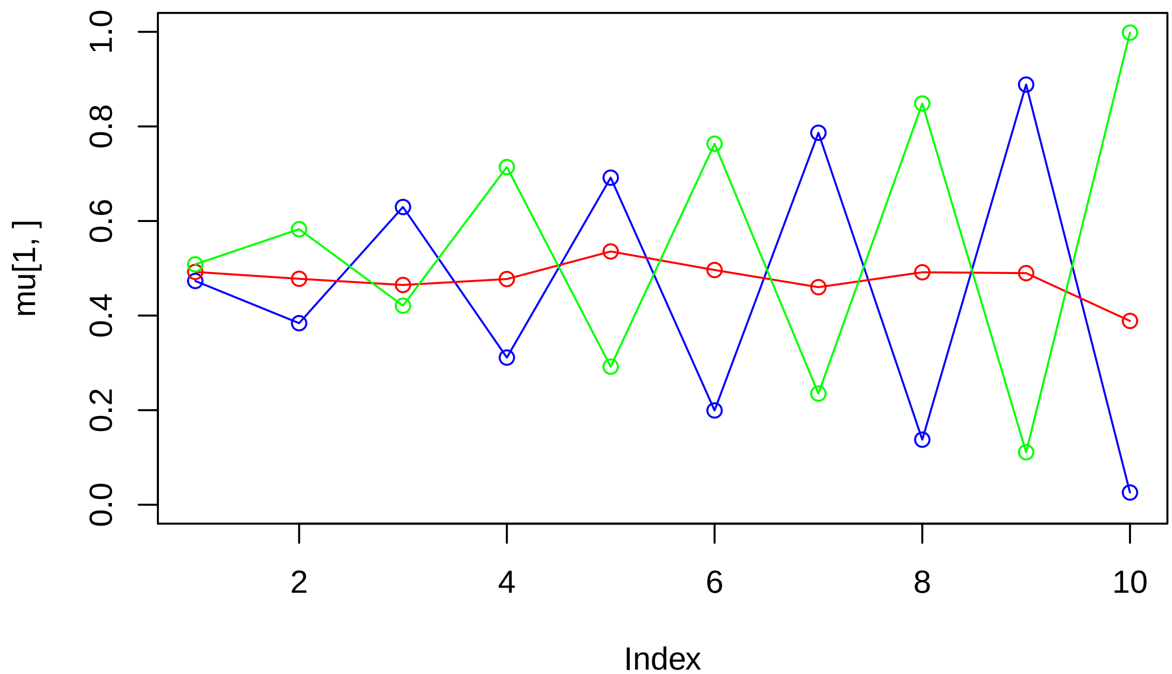

```
## iteration:  39 log likelihood:  -5321.999
```



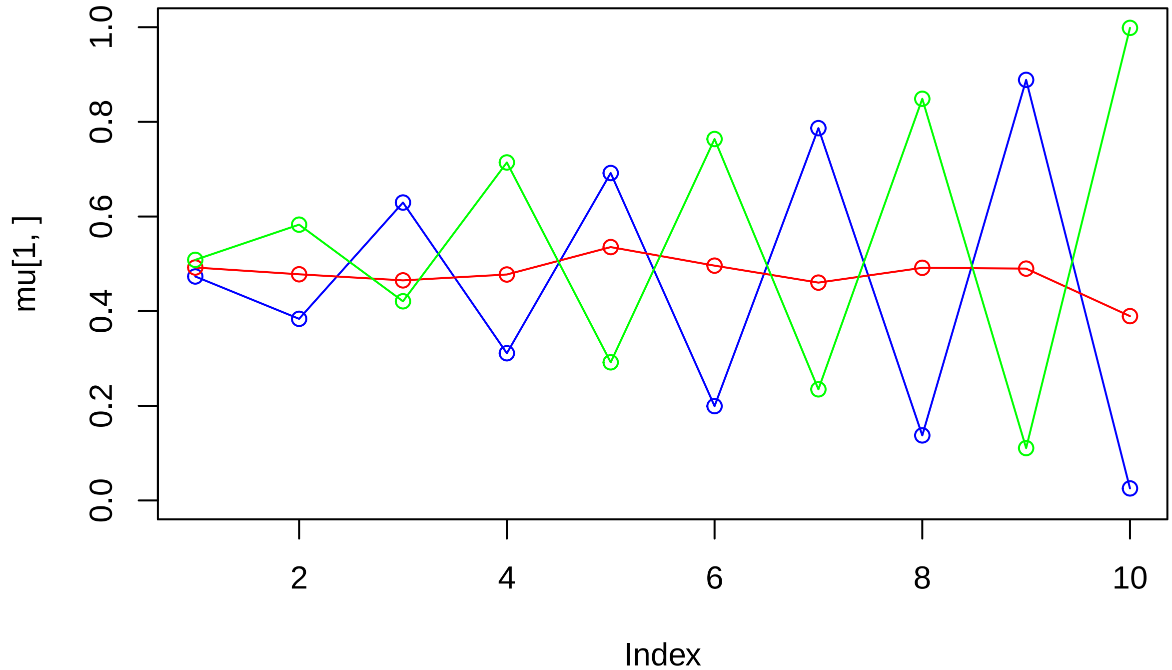

```
## iteration:  40 log likelihood:  -5321.498
```



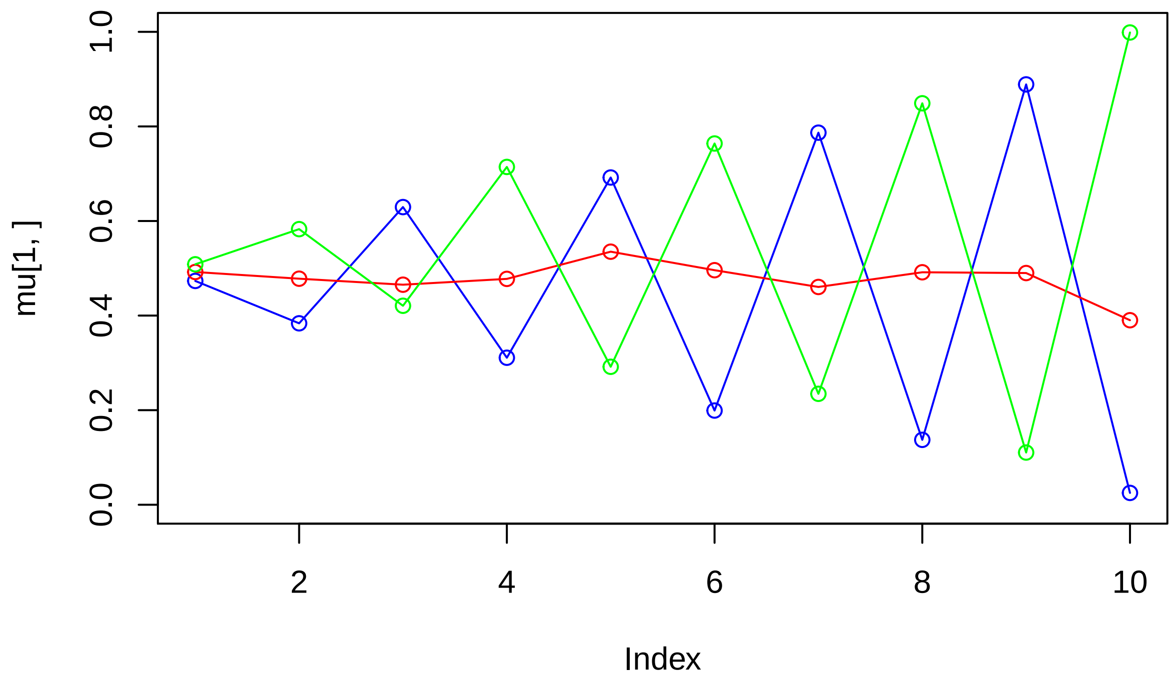

```
## iteration:  41 log likelihood:  -5321.031
```



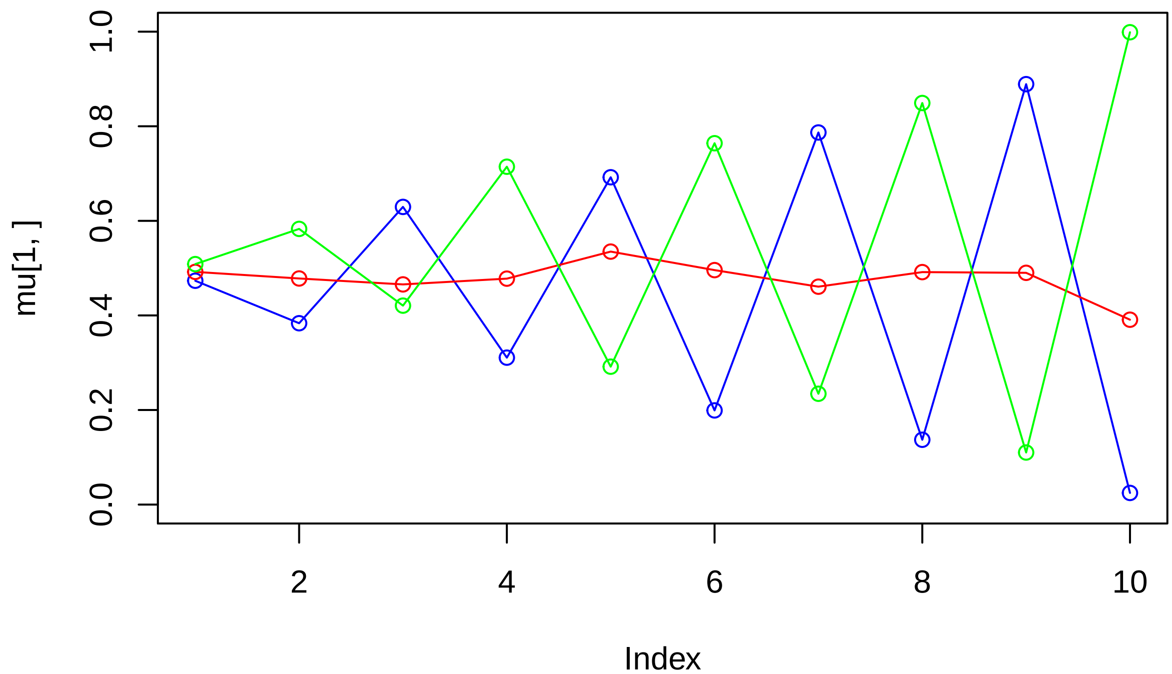

```
## iteration:  42 log likelihood:  -5320.596
```



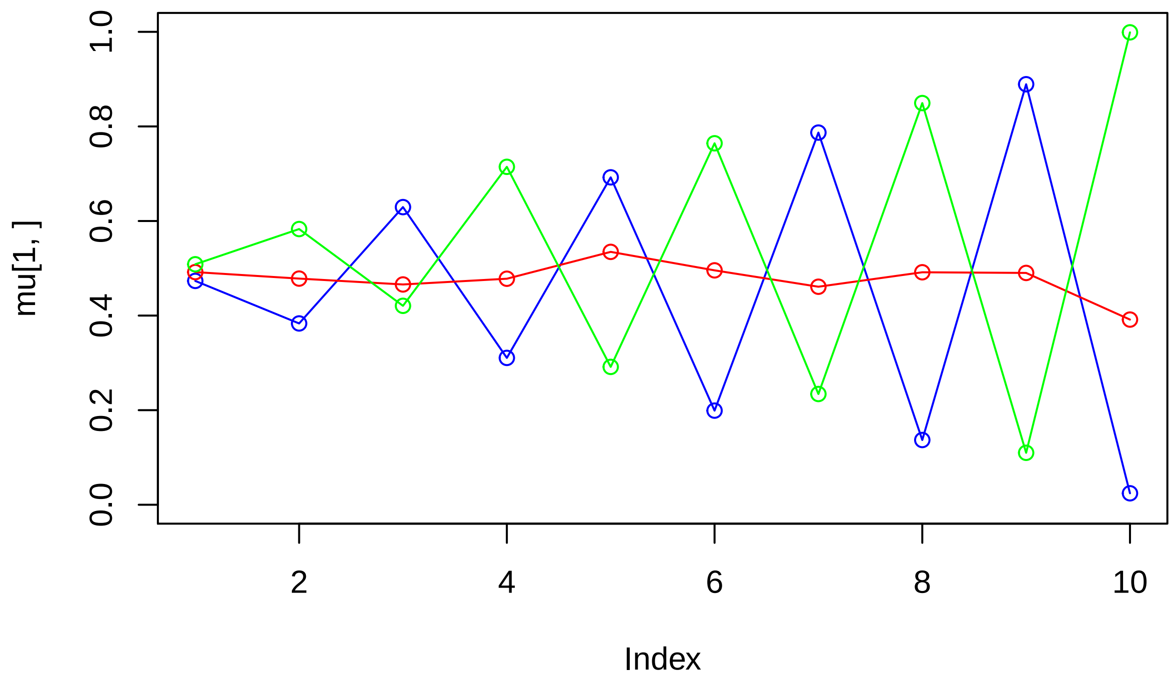

```
## iteration:  43 log likelihood:  -5320.19
```



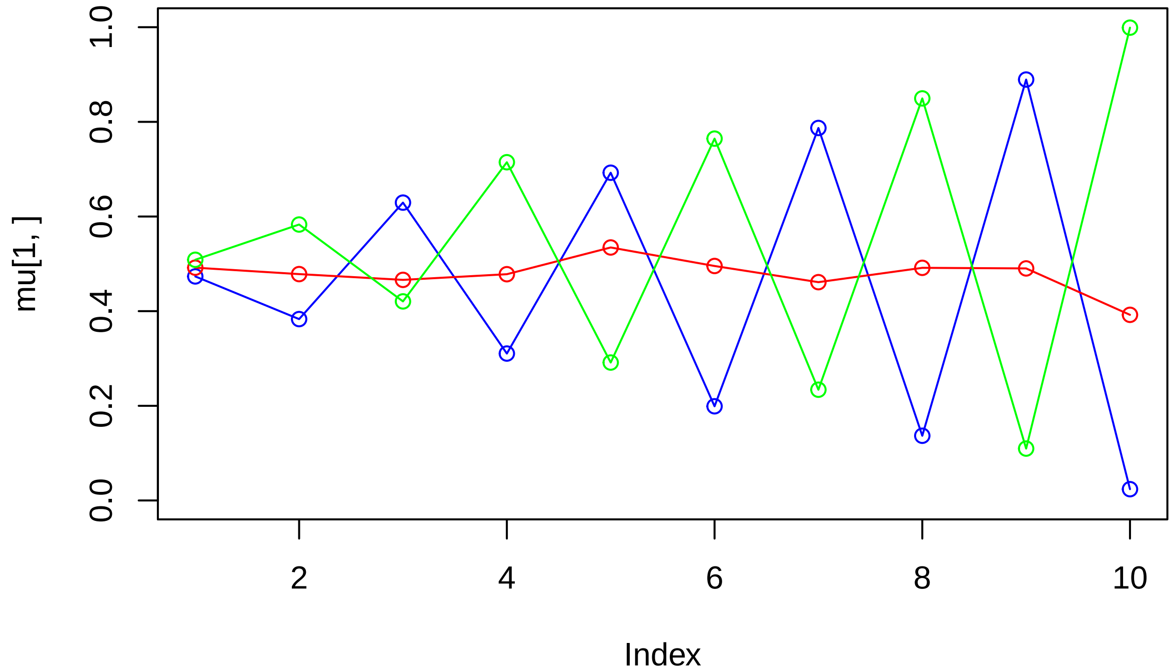

```
## iteration:  44 log likelihood:  -5319.81
```



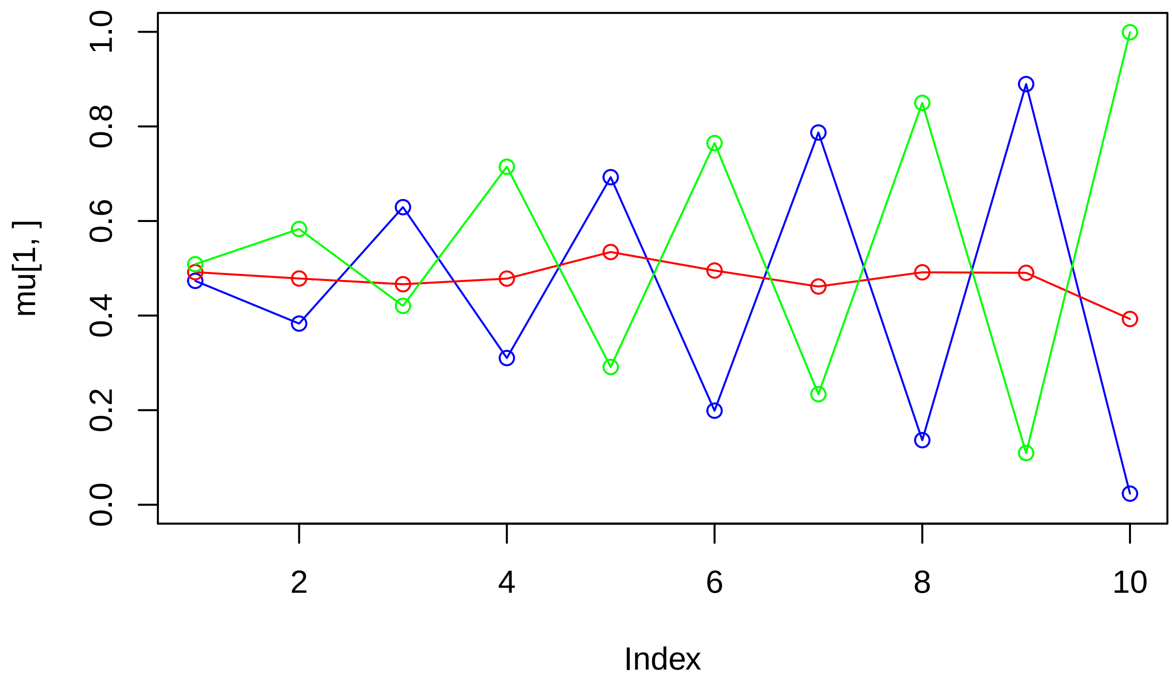

```
## iteration:  45 log likelihood:  -5319.454
```



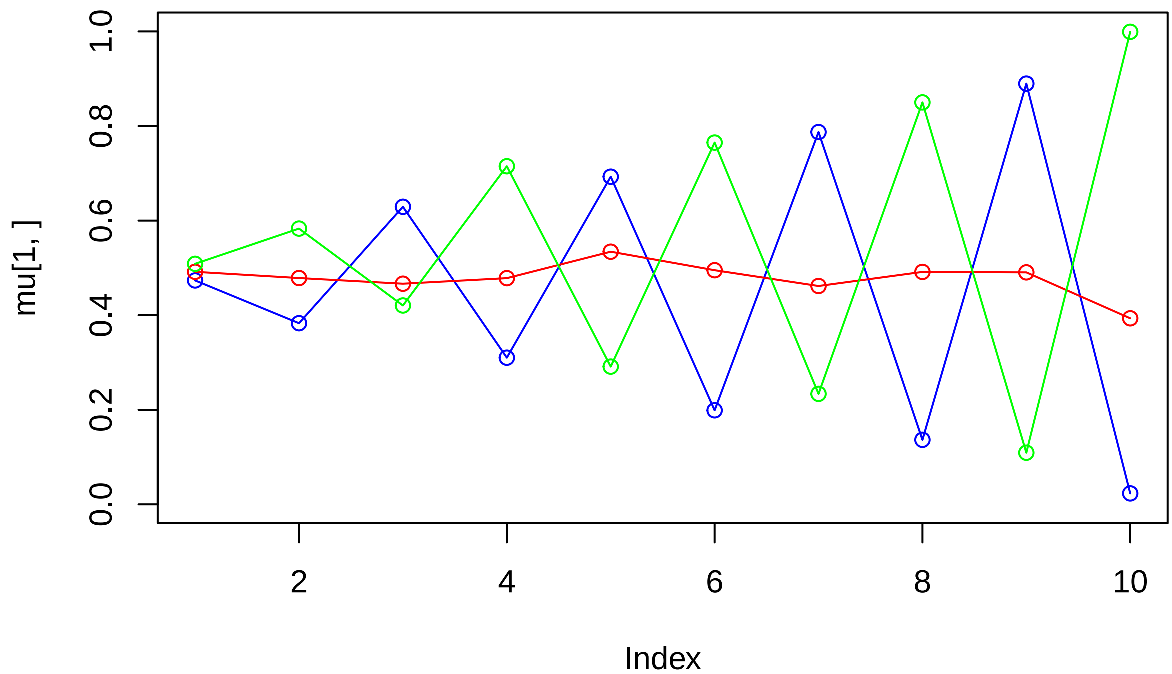

```
## iteration:  46 log likelihood:  -5319.121
```



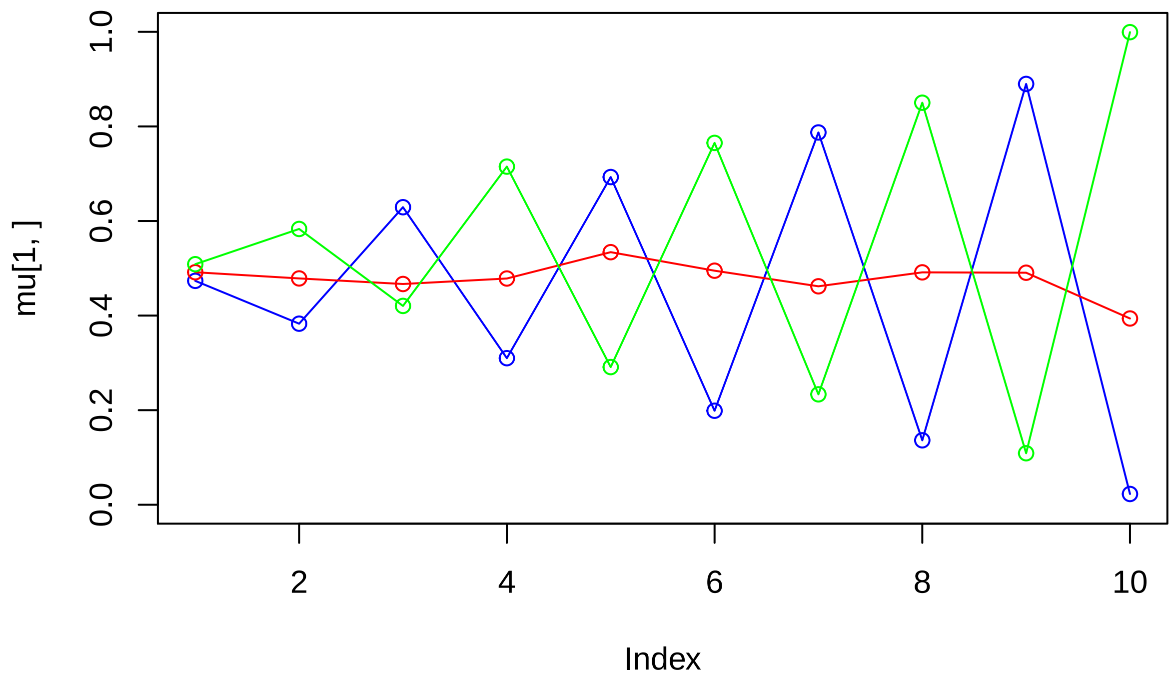

```
## iteration:  47 log likelihood:  -5318.809
```



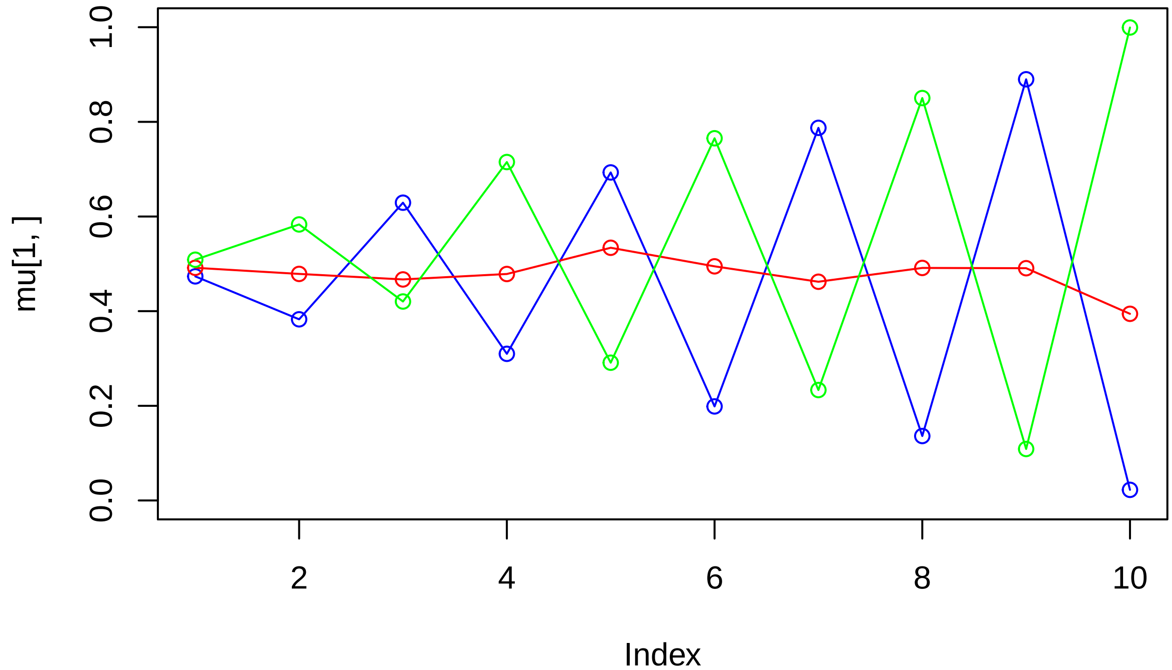

```
## iteration:  48 log likelihood:  -5318.515
```



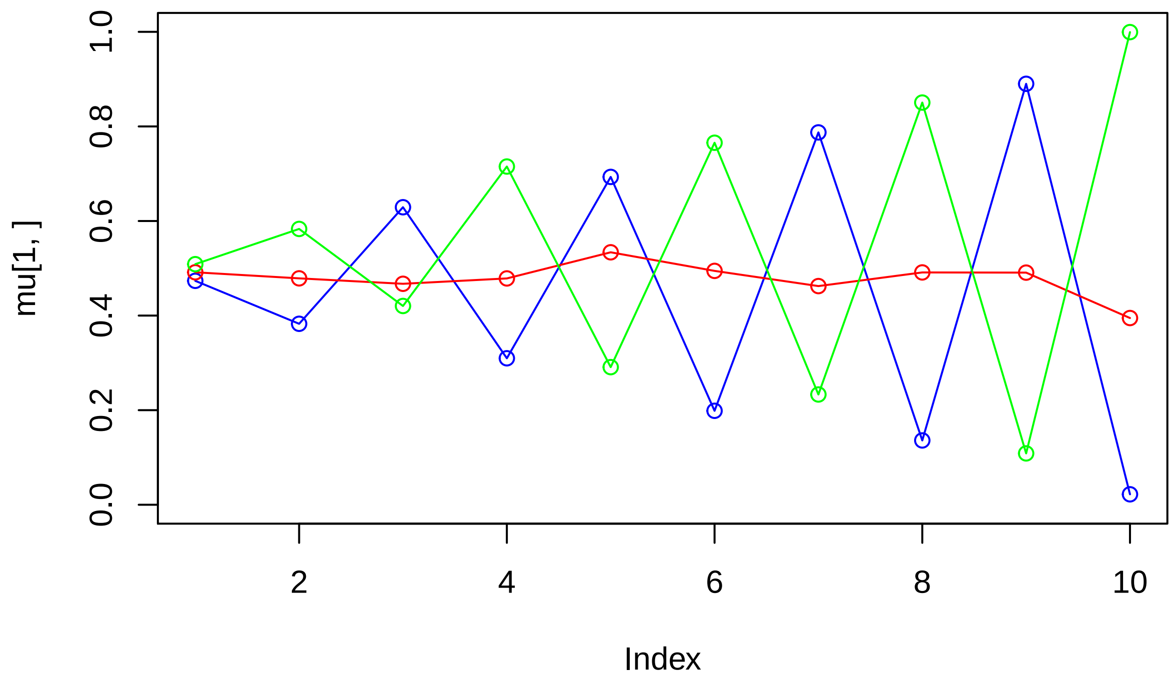

```
## iteration:  49 log likelihood:  -5318.239
```



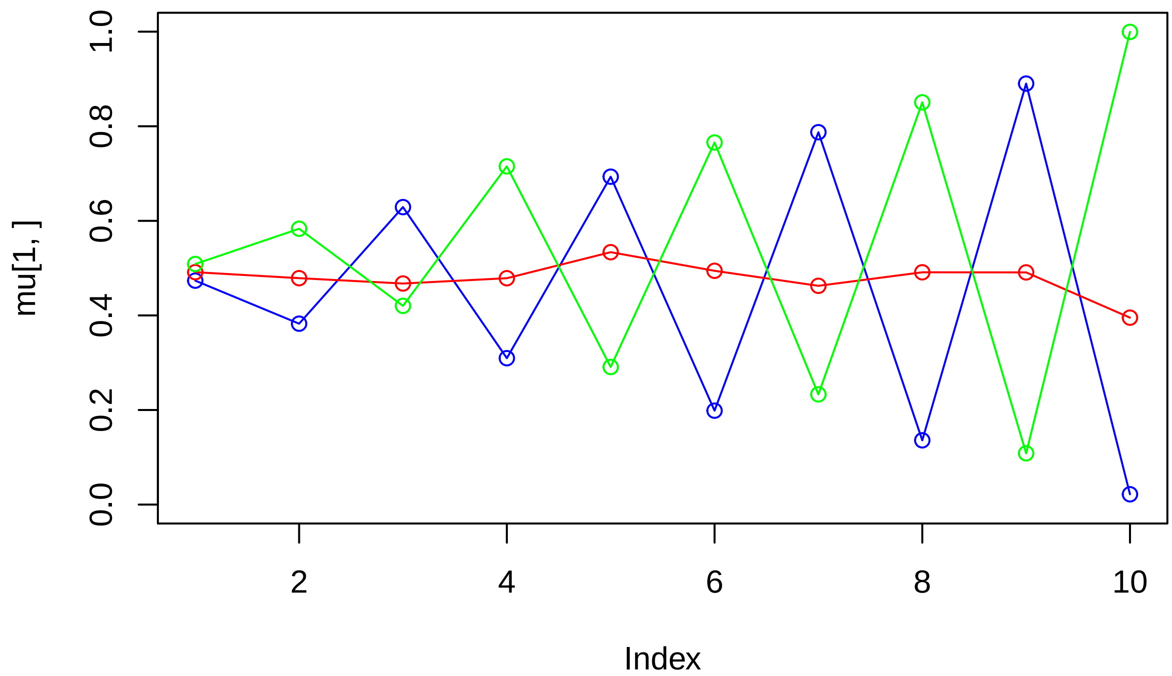

```
## iteration:  50 log likelihood:  -5317.979
```



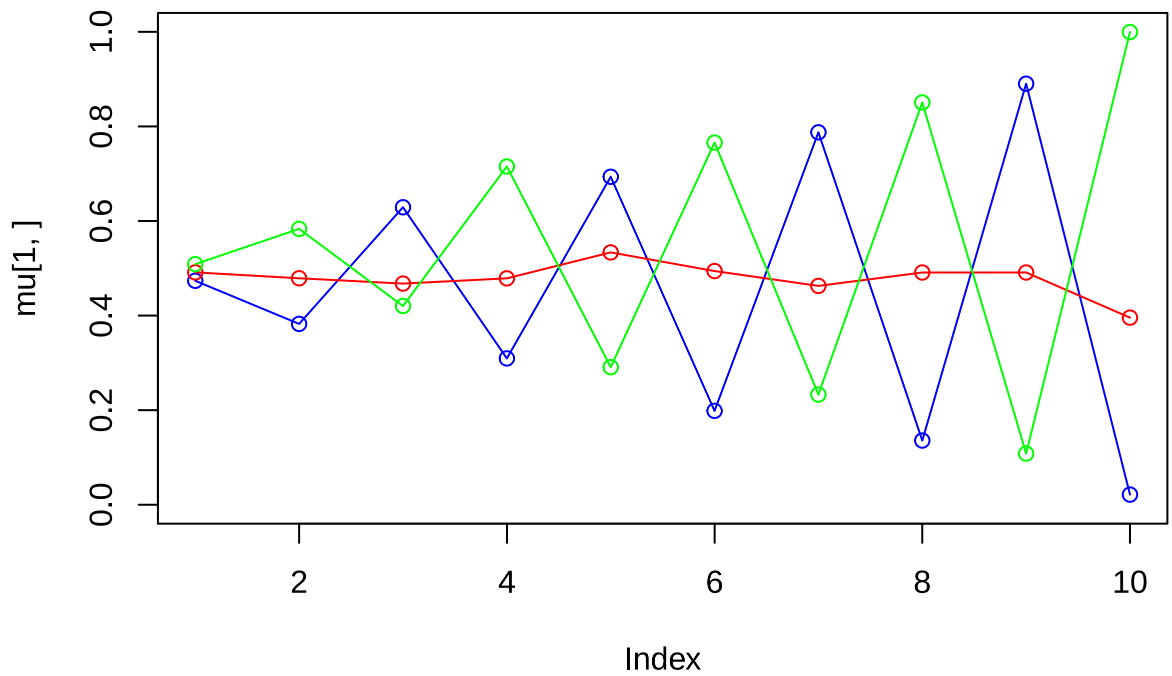

```
## iteration:  51 log likelihood:  -5317.734
```



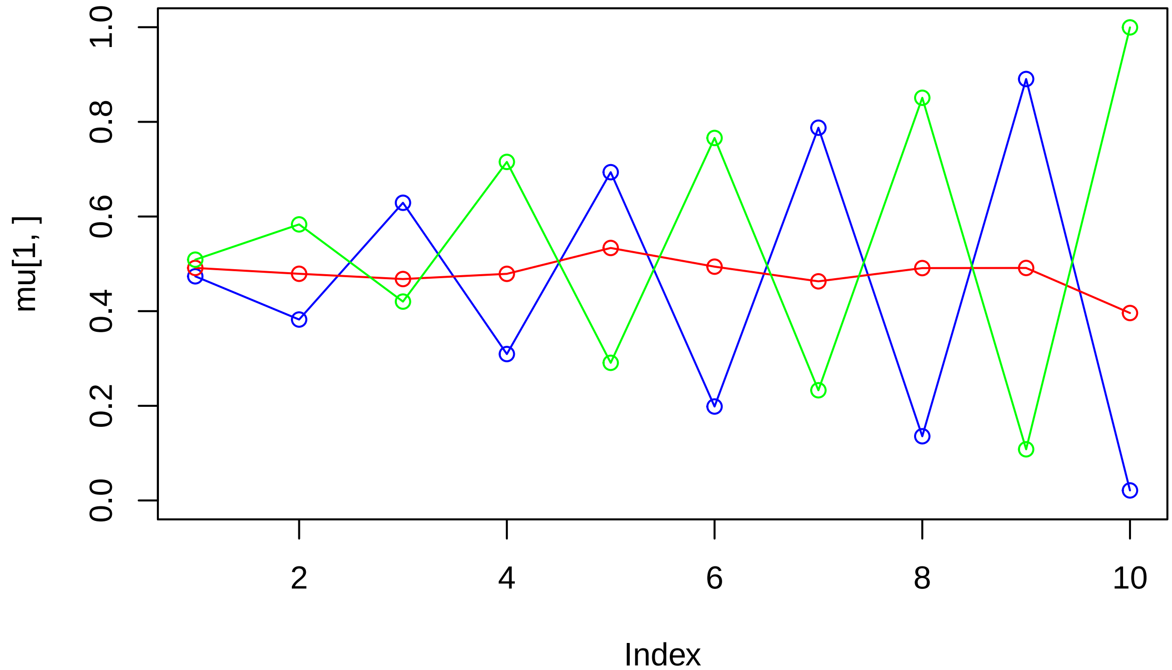

```
## iteration:  52 log likelihood:  -5317.503
```



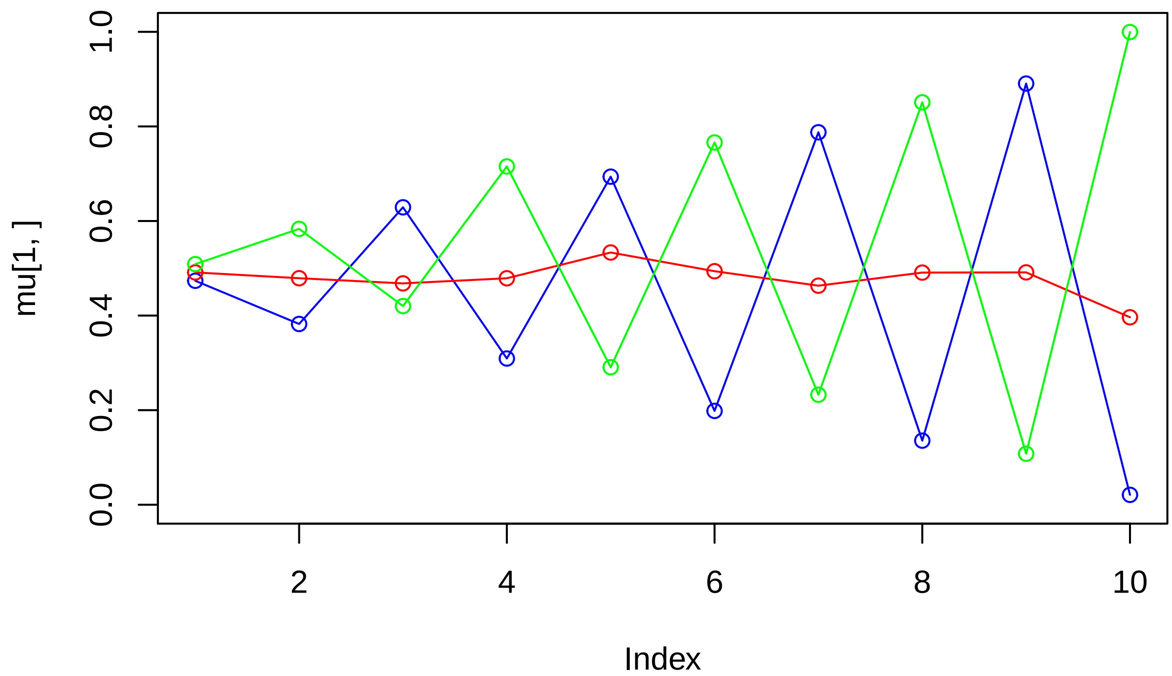

```
## iteration:  53 log likelihood:  -5317.284
```



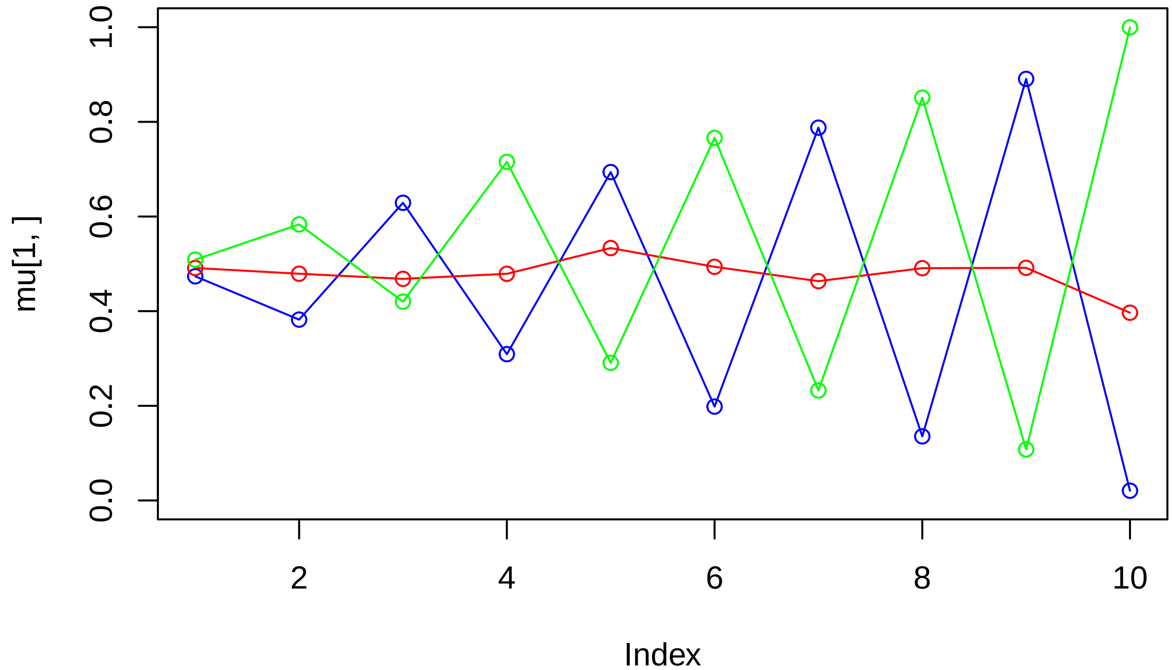

```
## iteration:  54 log likelihood:  -5317.077
```



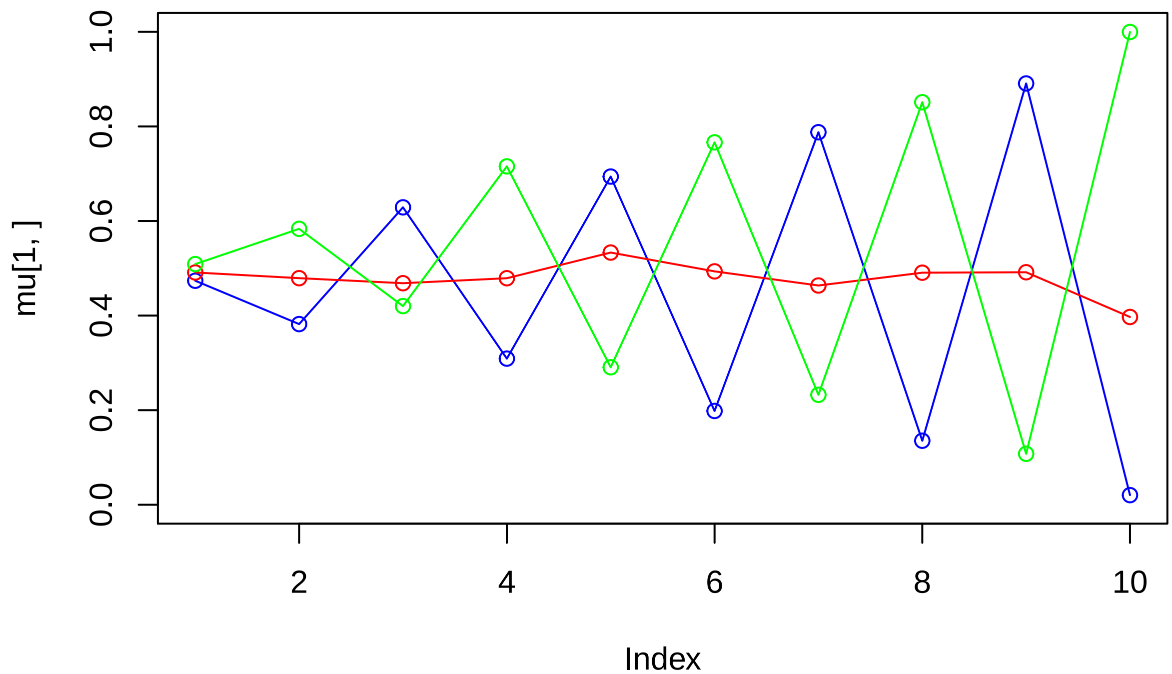

```
## iteration:  55 log likelihood:  -5316.881
```



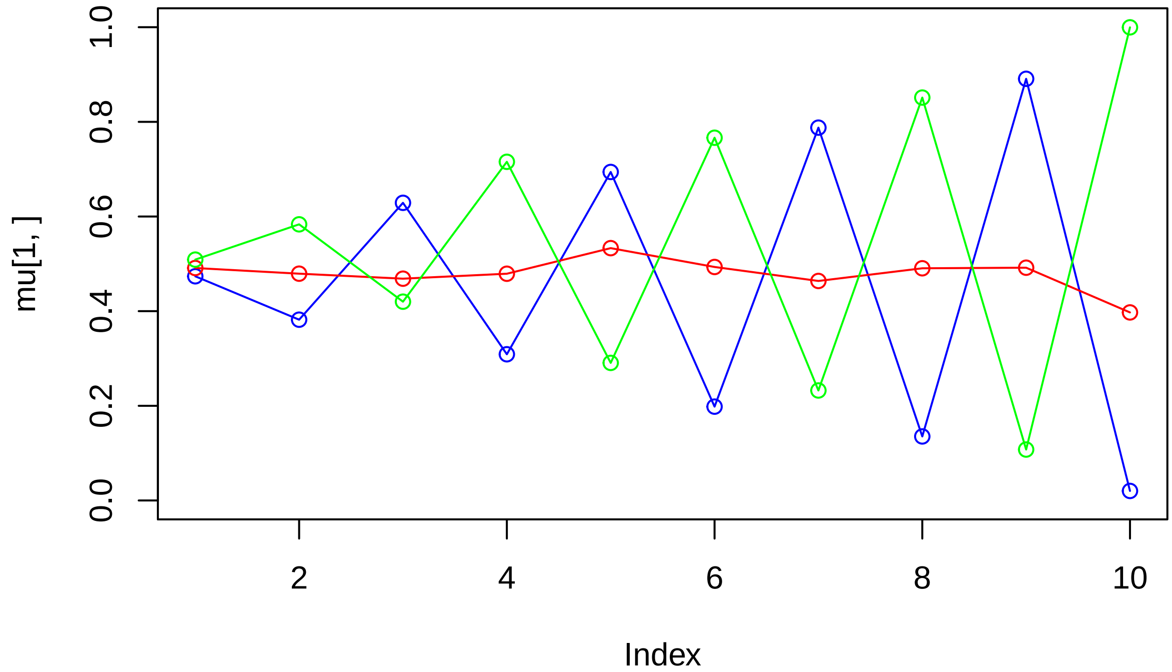

```
## iteration:  56 log likelihood:  -5316.695
```



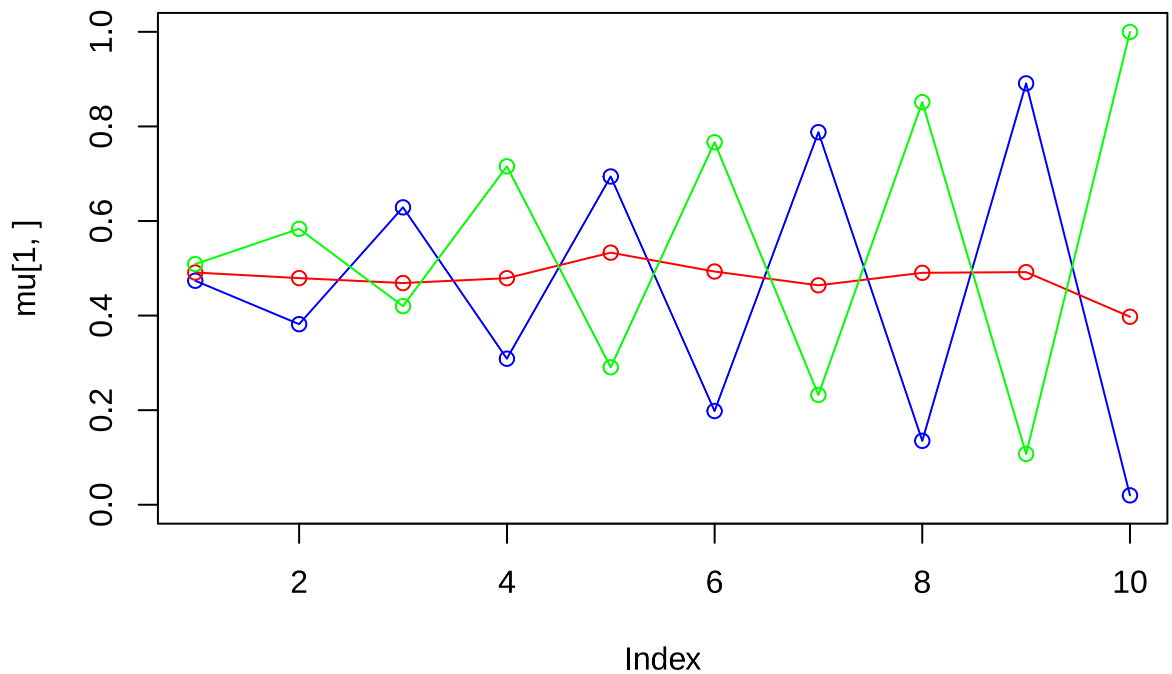

```
## iteration:  57 log likelihood:  -5316.518
```



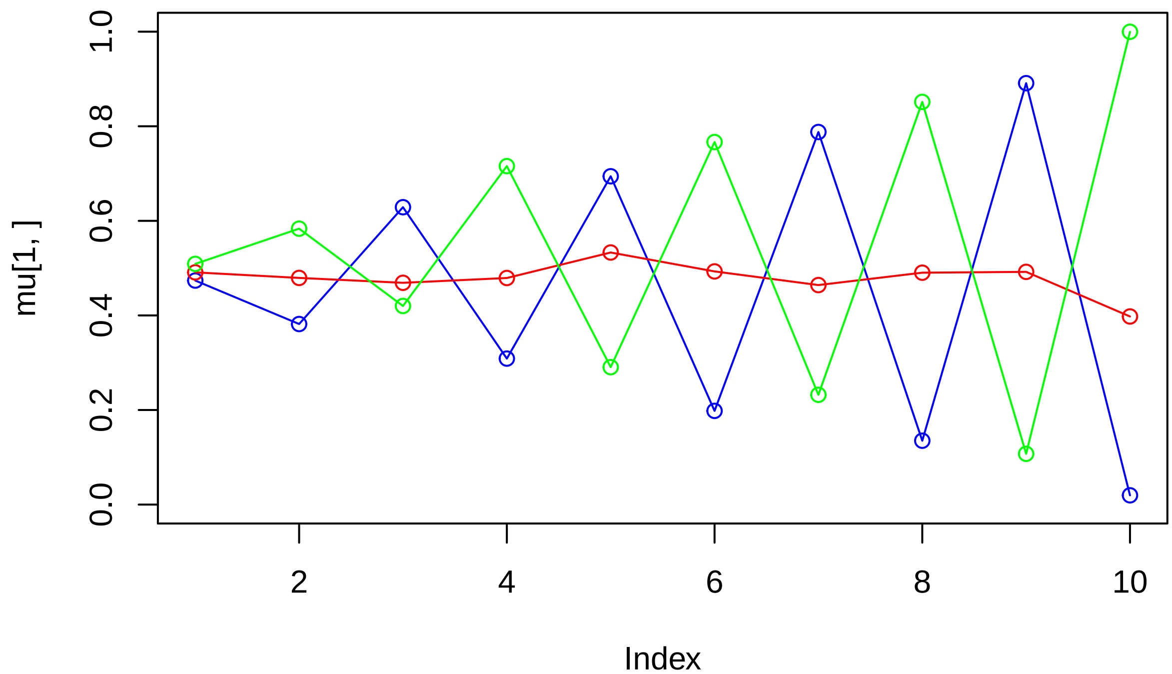

```
## iteration:  58 log likelihood:  -5316.349
```



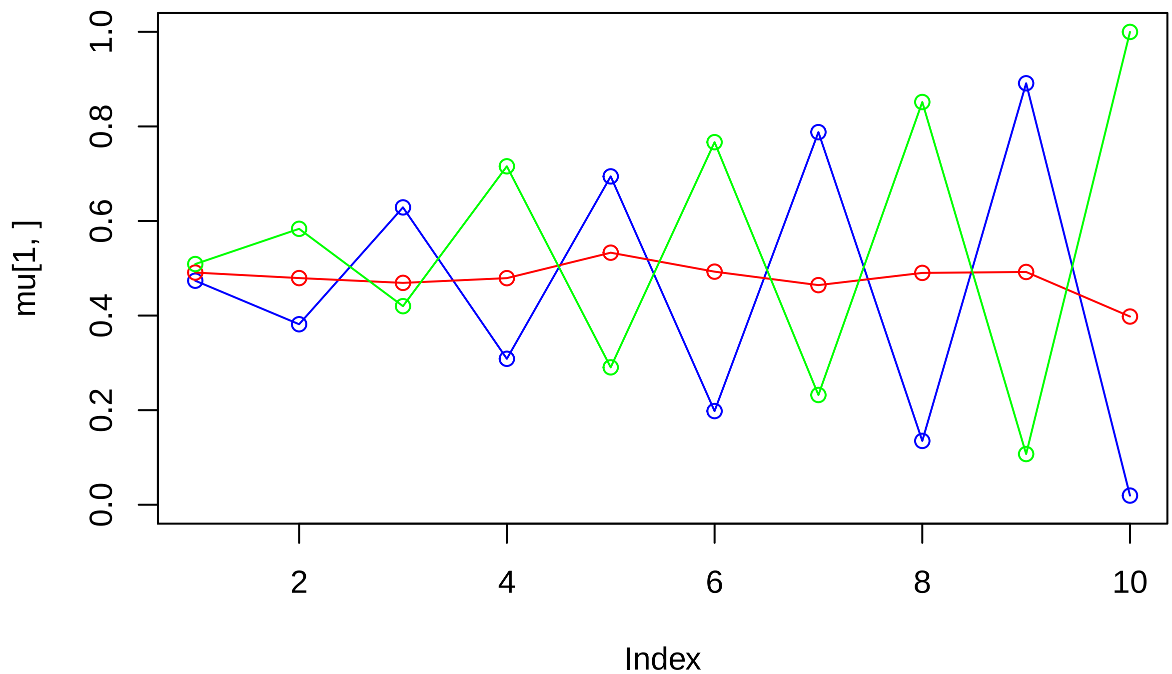

```
## iteration:  59 log likelihood:  -5316.189
```



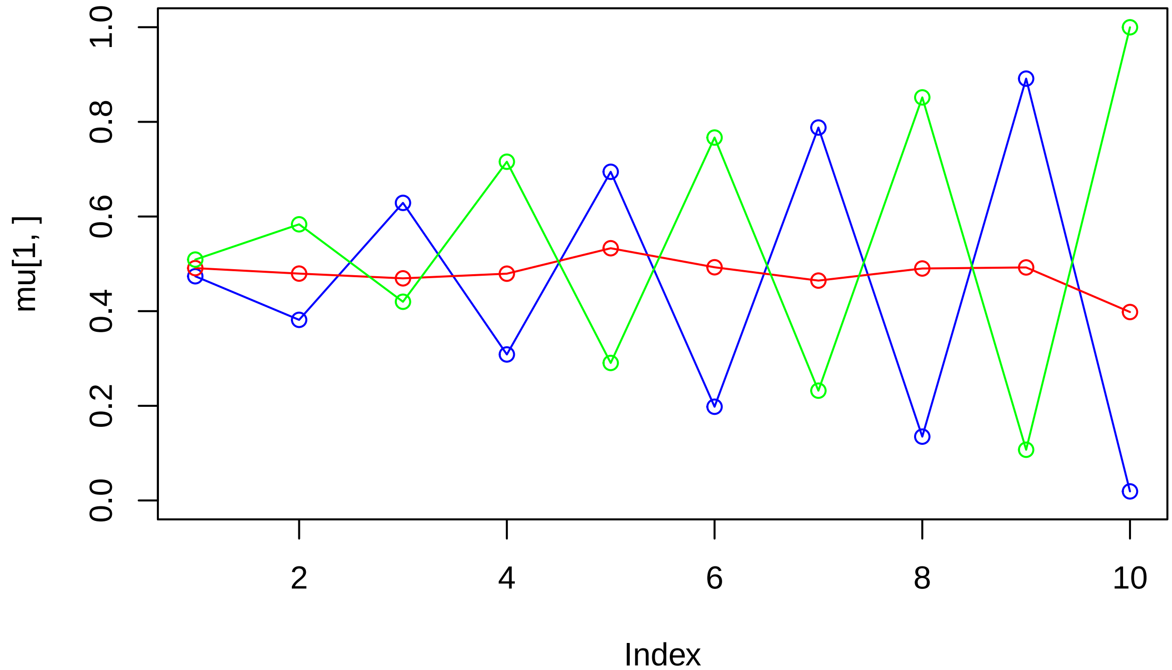

```
## iteration:  60 log likelihood:  -5316.036
```



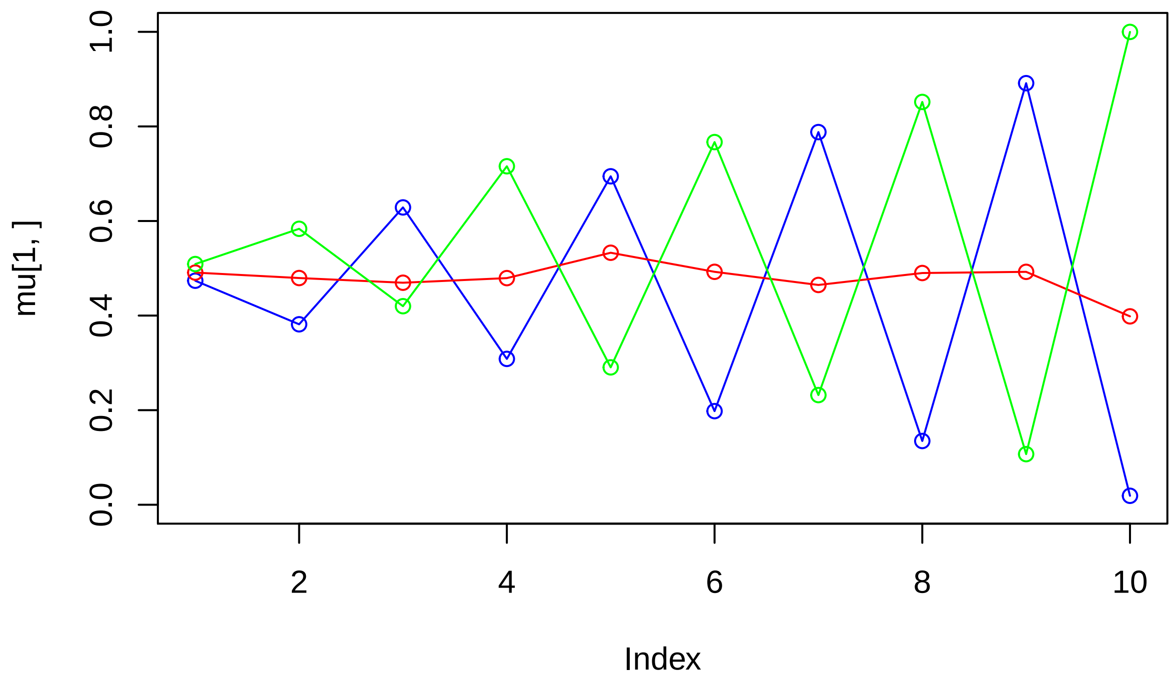

```
## iteration:  61 log likelihood:  -5315.89
```



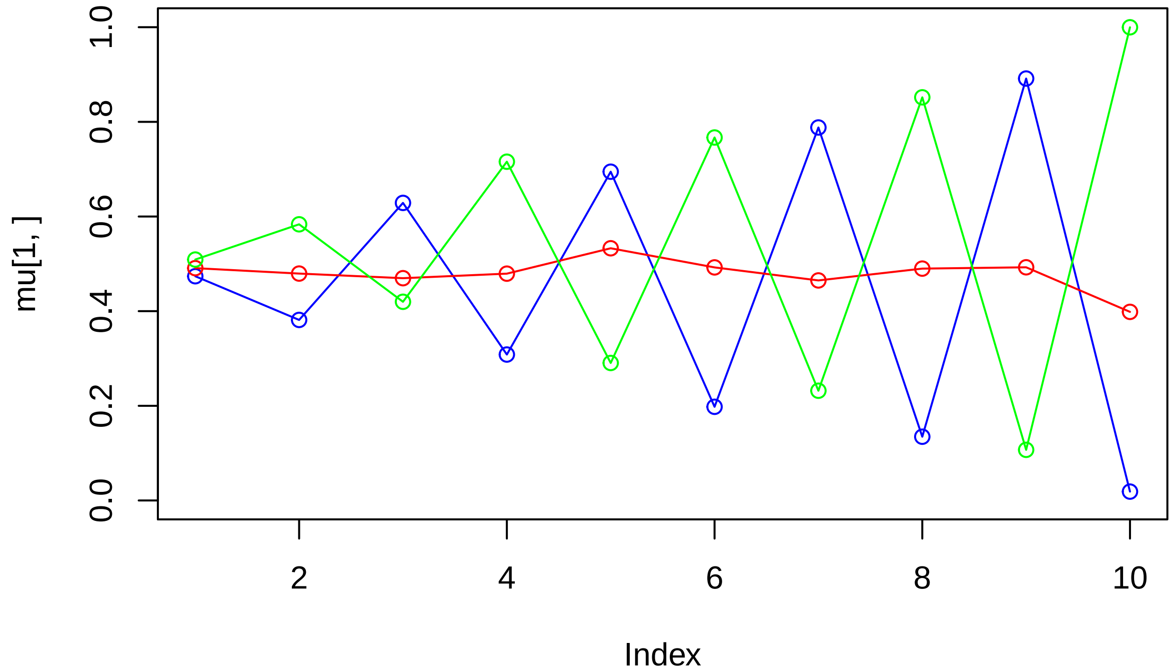

```
## iteration:  62 log likelihood:  -5315.75
```



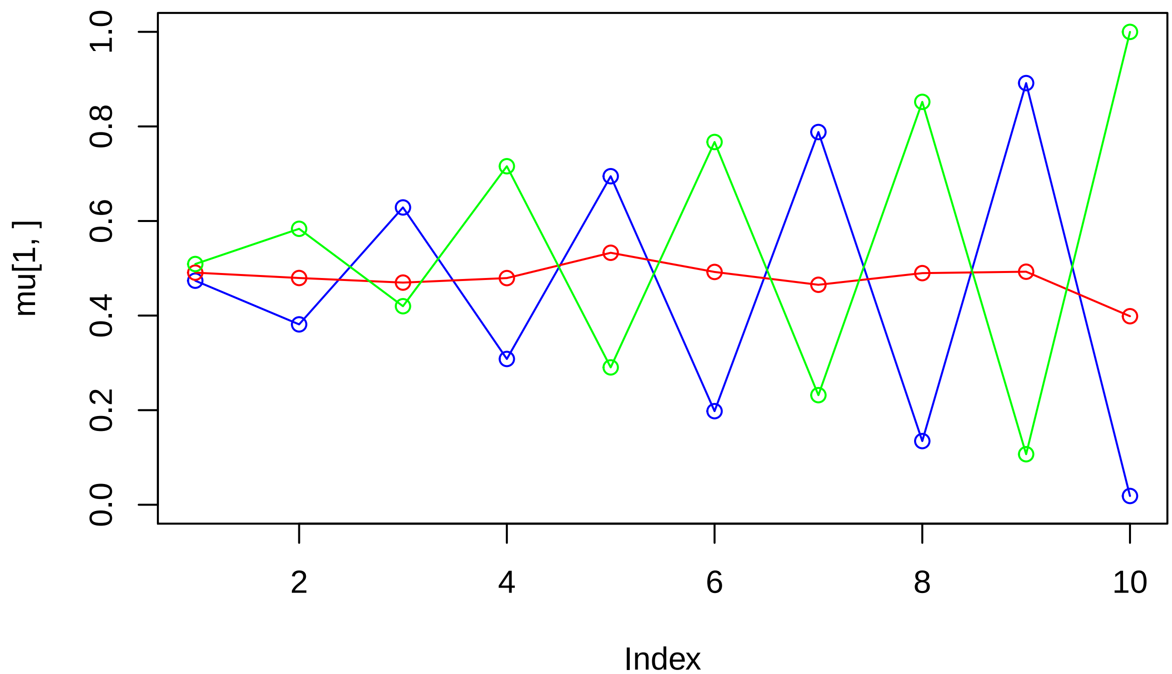

```
## iteration:  63 log likelihood:  -5315.616
```



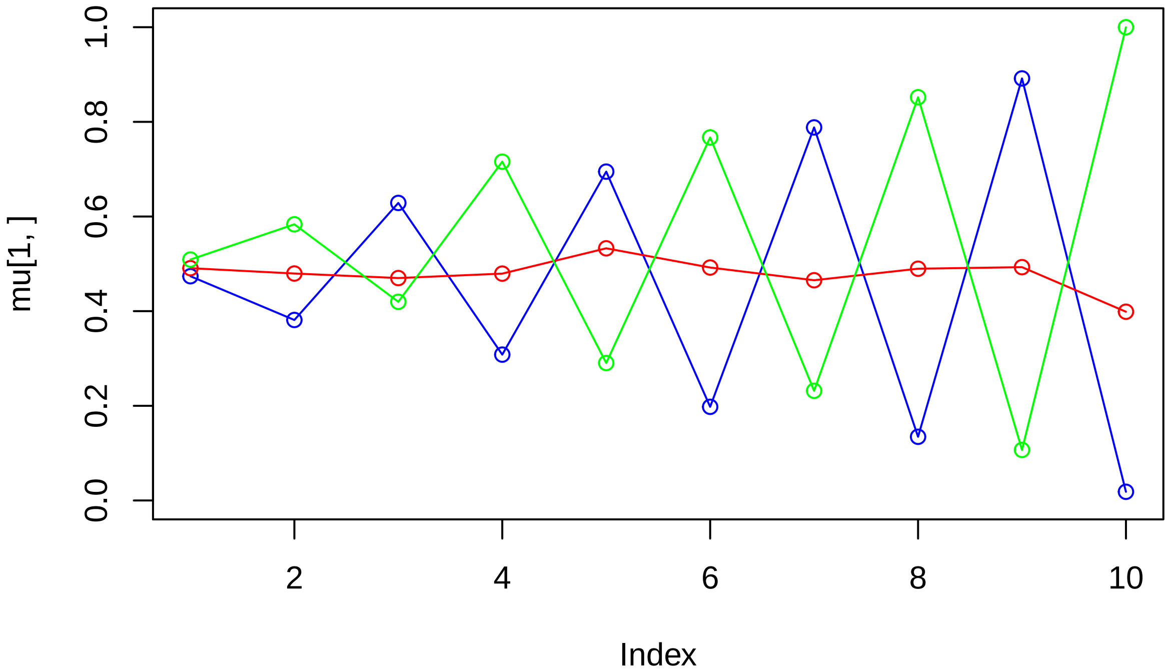

```
## iteration:  64 log likelihood:  -5315.487
```



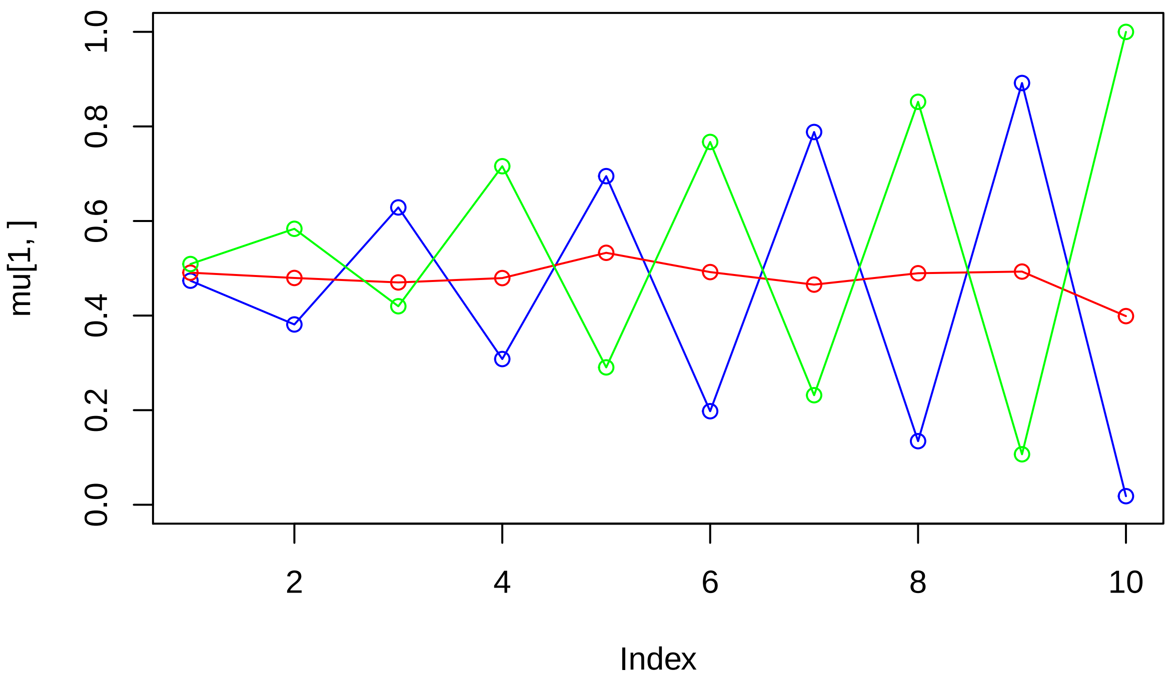

```
## iteration:  65 log likelihood:  -5315.364
```



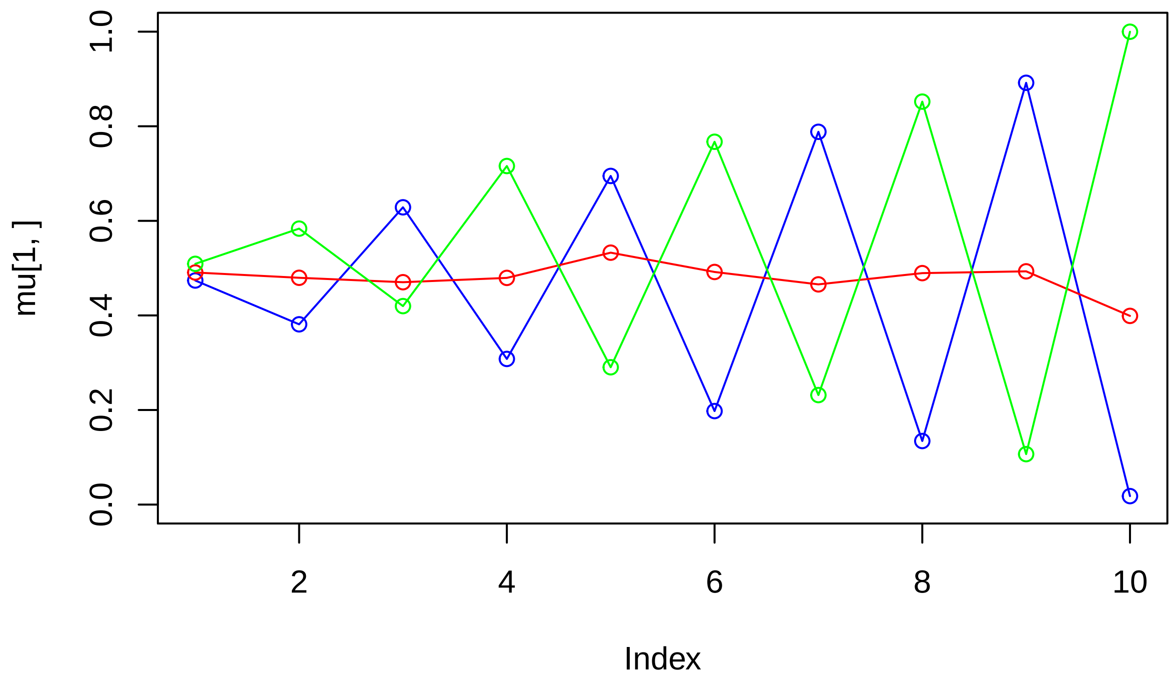

```
## iteration:  66 log likelihood:  -5315.246
```



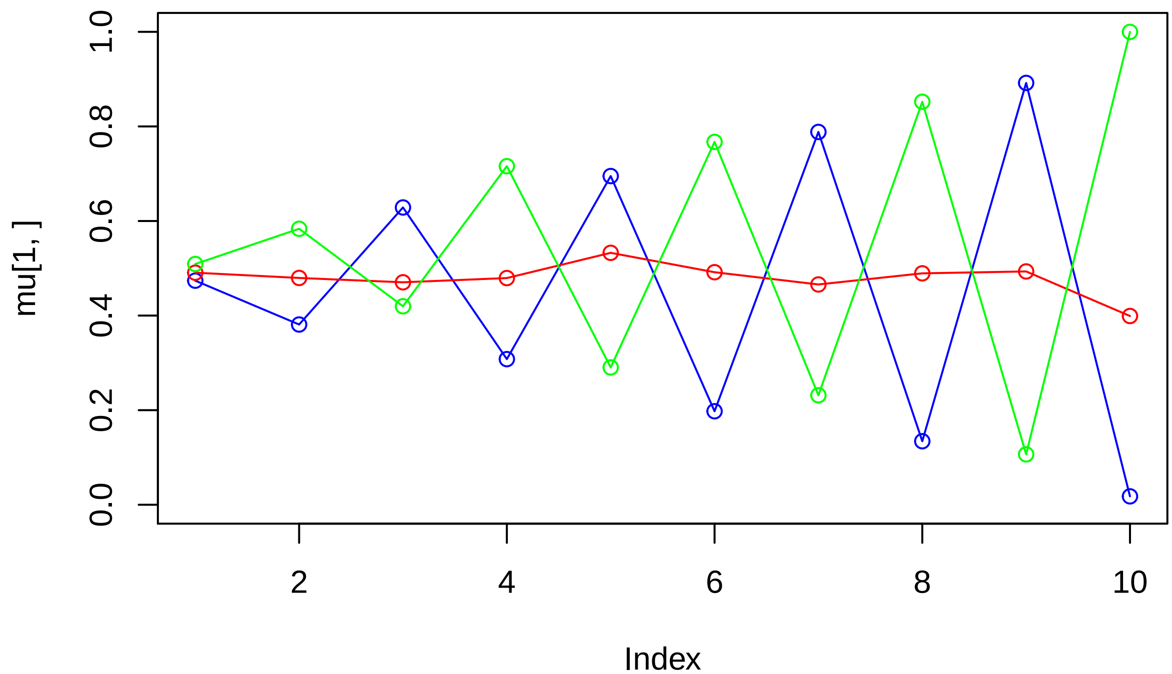

```
## iteration:  67 log likelihood:  -5315.132
```



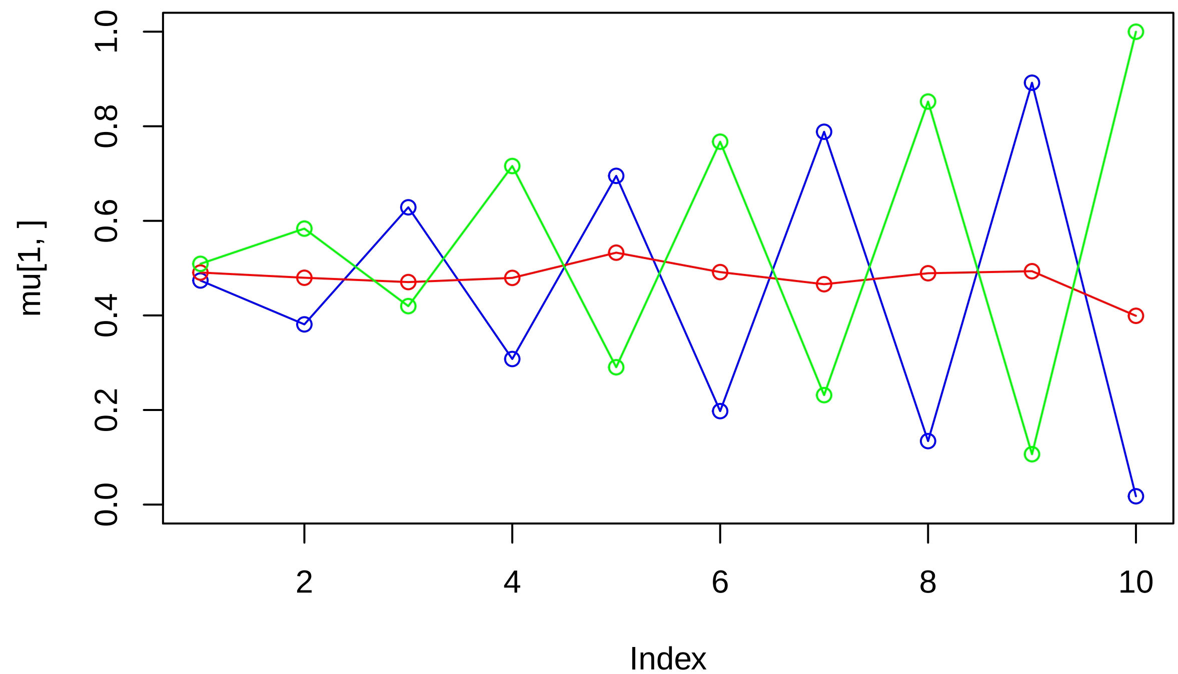

```
## iteration:  68 log likelihood:  -5315.022
```



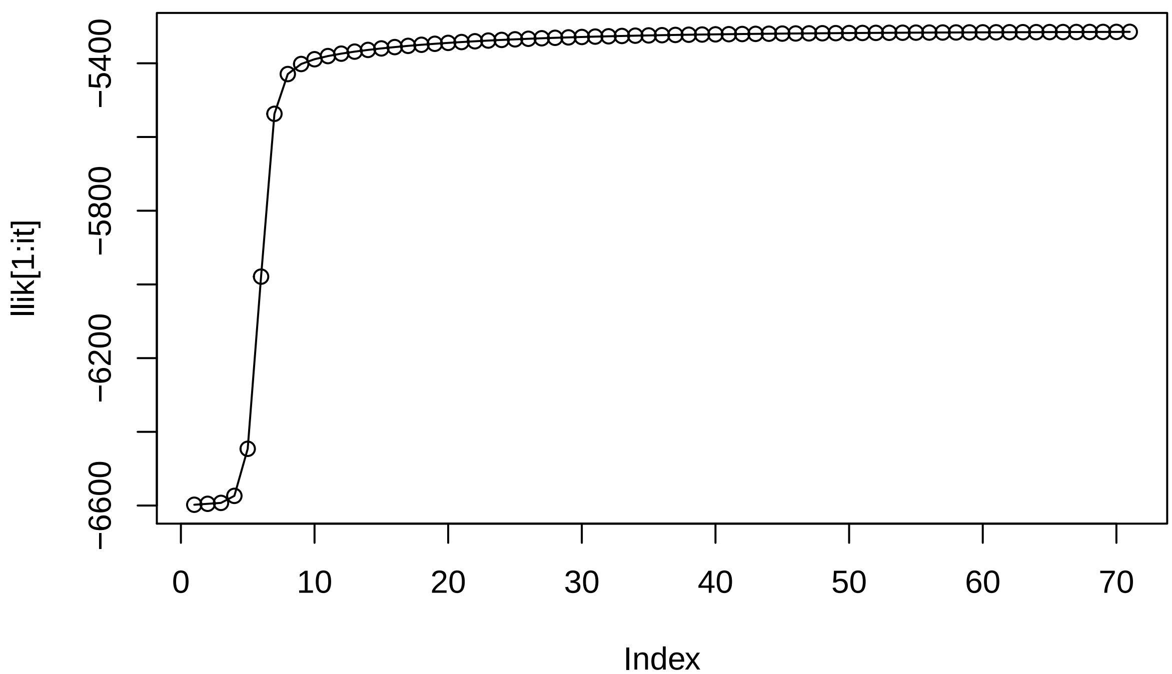

```
## iteration:  69 log likelihood:  -5314.916
```




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

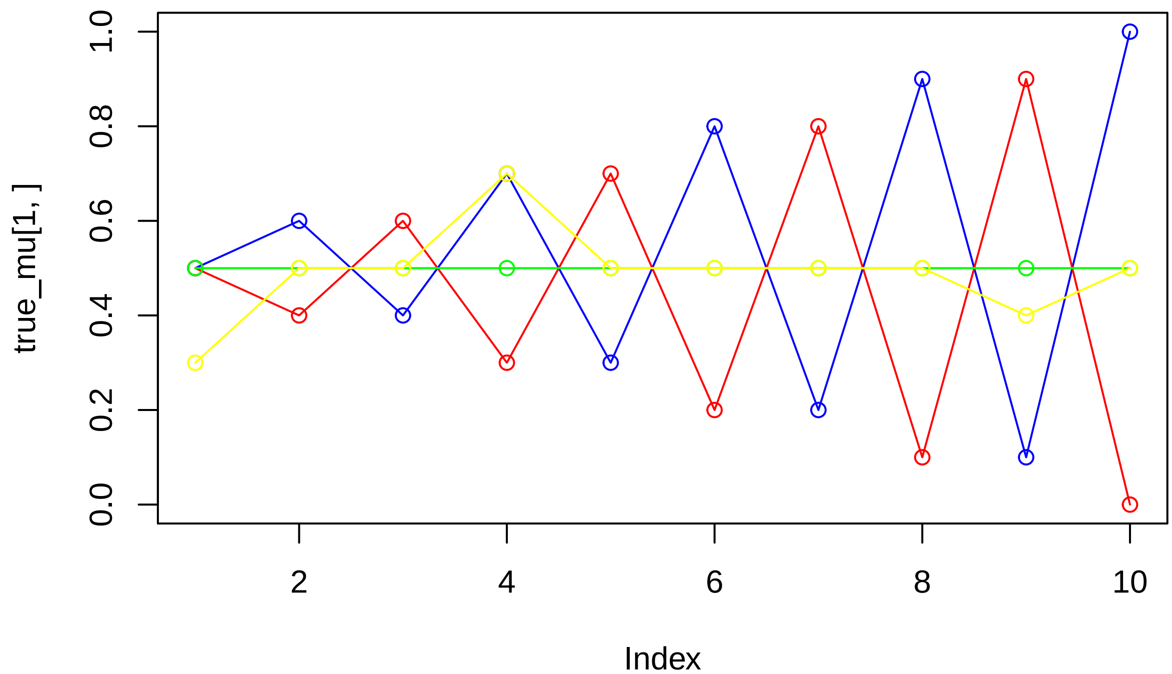


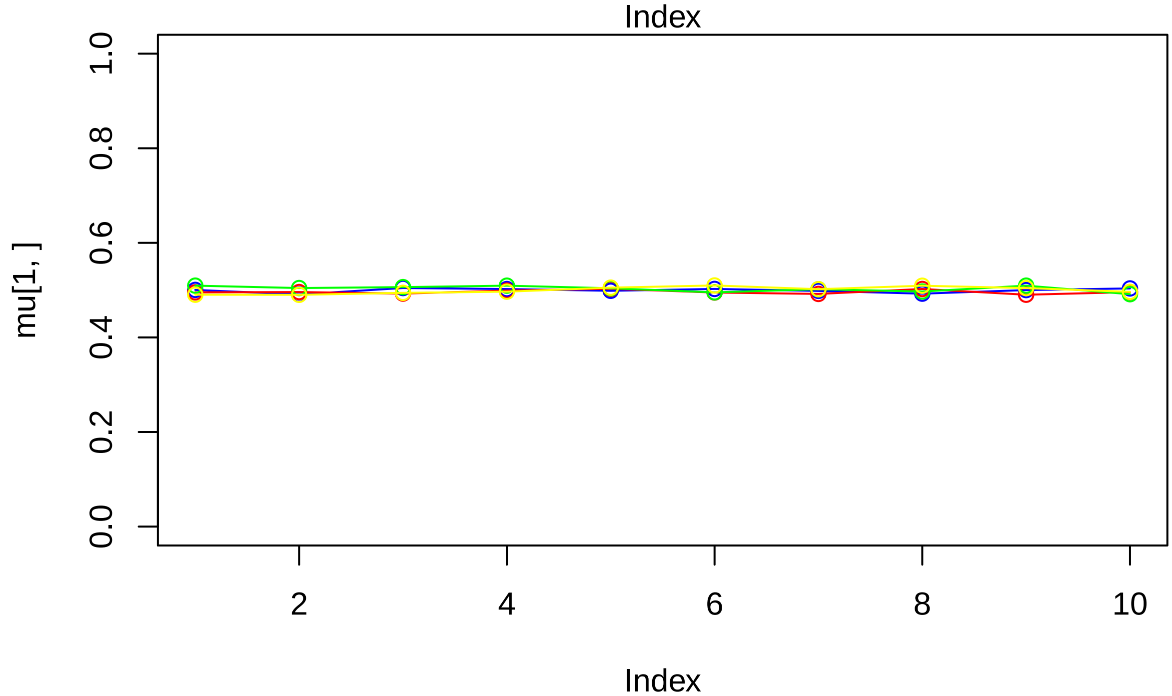

```
## iteration:  71 log likelihood:  -5314.715
```



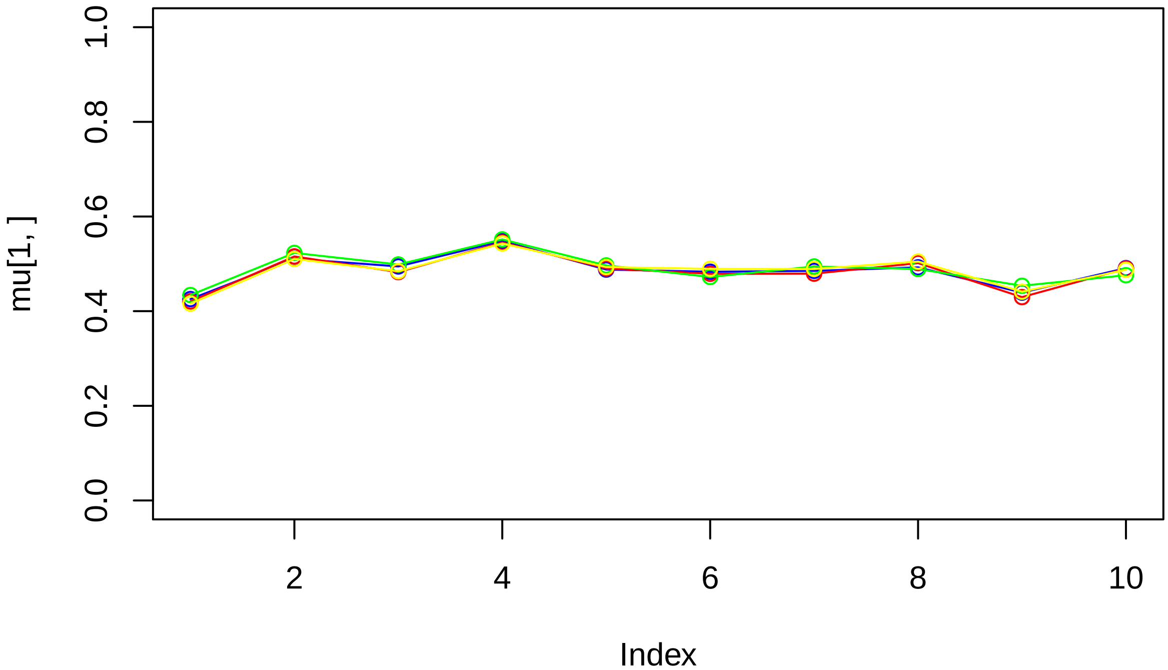

```
## [[1]]
##  [1] 0.32411556 0.30717327 0.36871116 0.47383554 0.49067297 0.50907301
##  [7] 0.38106095 0.47962547 0.58355730 0.62860603 0.47050854 0.41965465
## [13] 0.30784064 0.47936473 0.71594258 0.69512193 0.53269167 0.29039403
## [19] 0.19764738 0.49158731 0.76736739 0.78836950 0.46597001 0.23146477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

```
my_own_em(4)
```

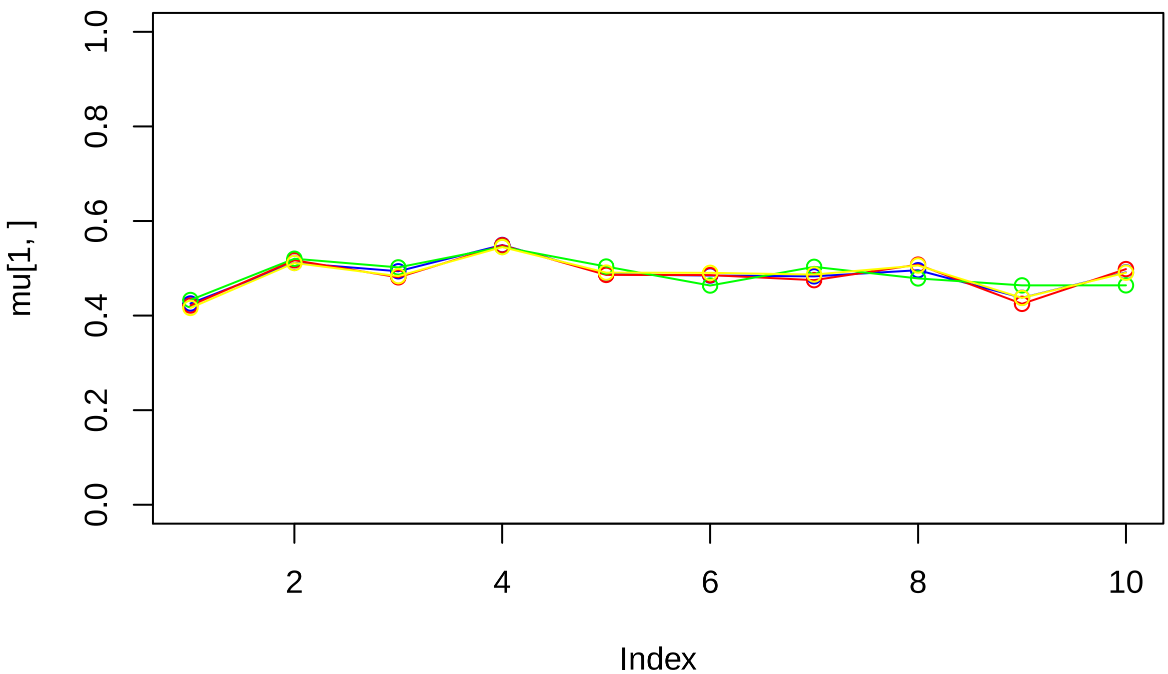




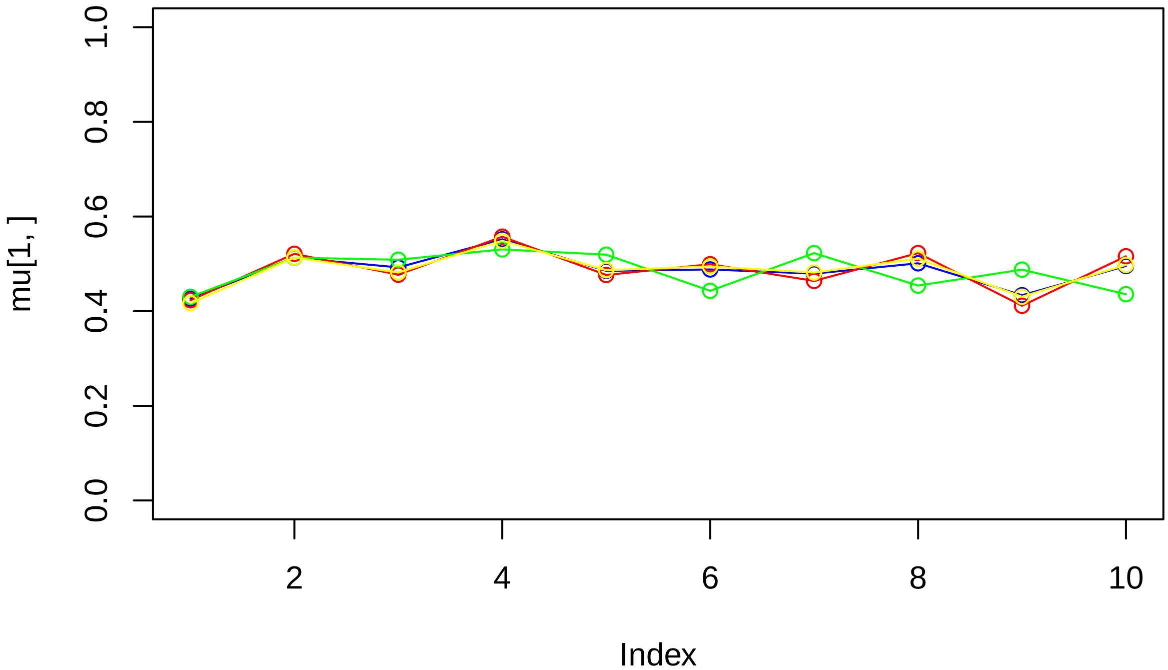

```
## iteration:  1 log likelihood:  -6680.657
```



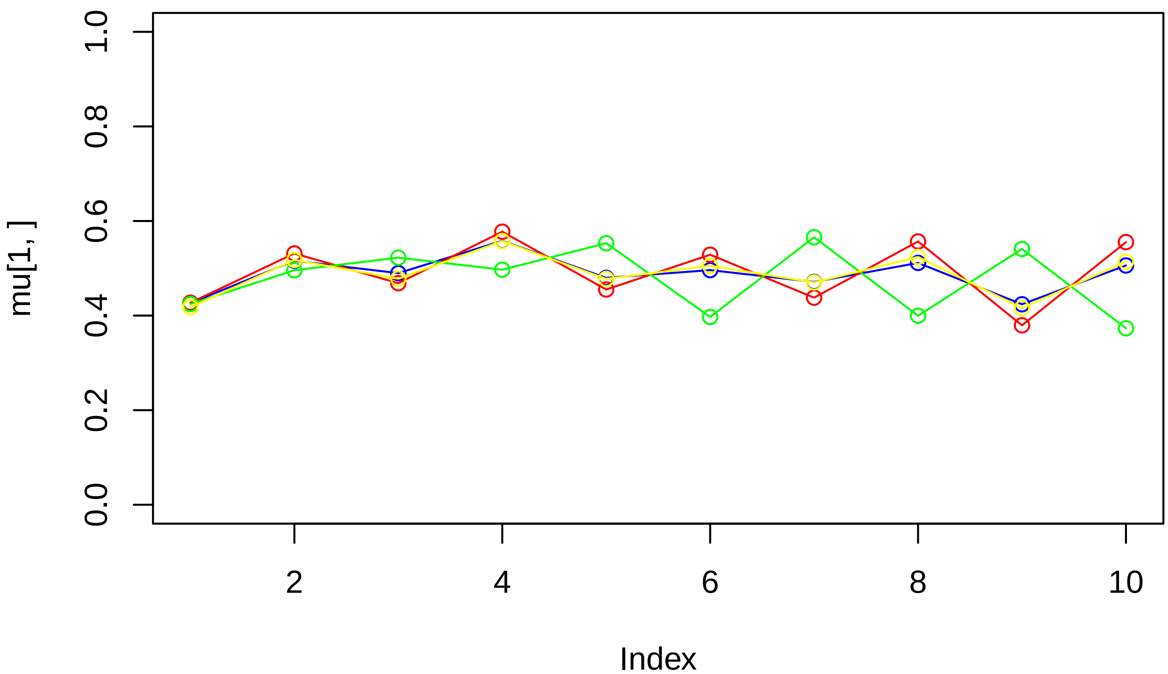

```
## iteration:  2 log likelihood:  -6654.874
```



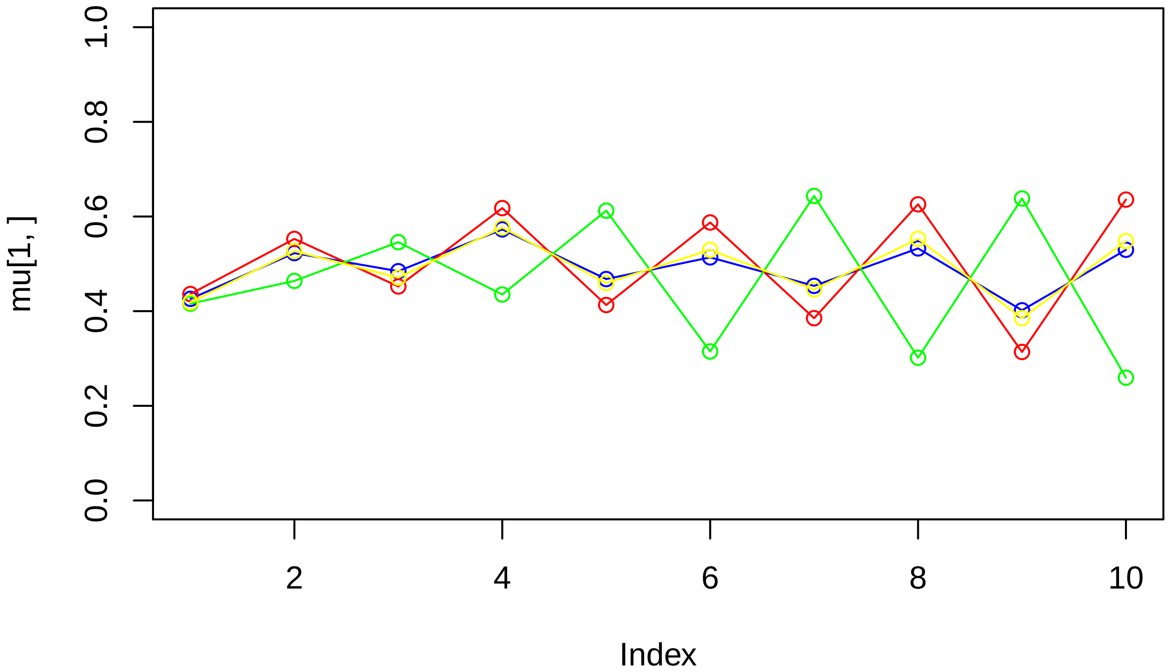

```
## iteration:  3 log likelihood:  -6650.741
```



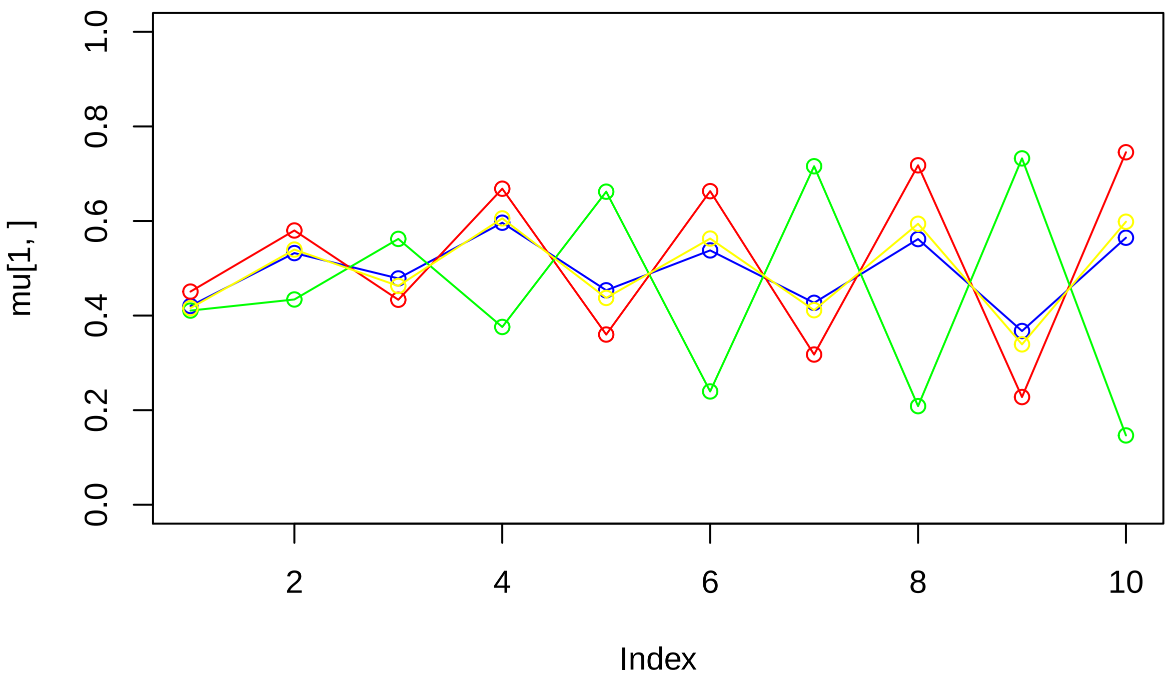

```
## iteration:  4 log likelihood:  -6628.679
```



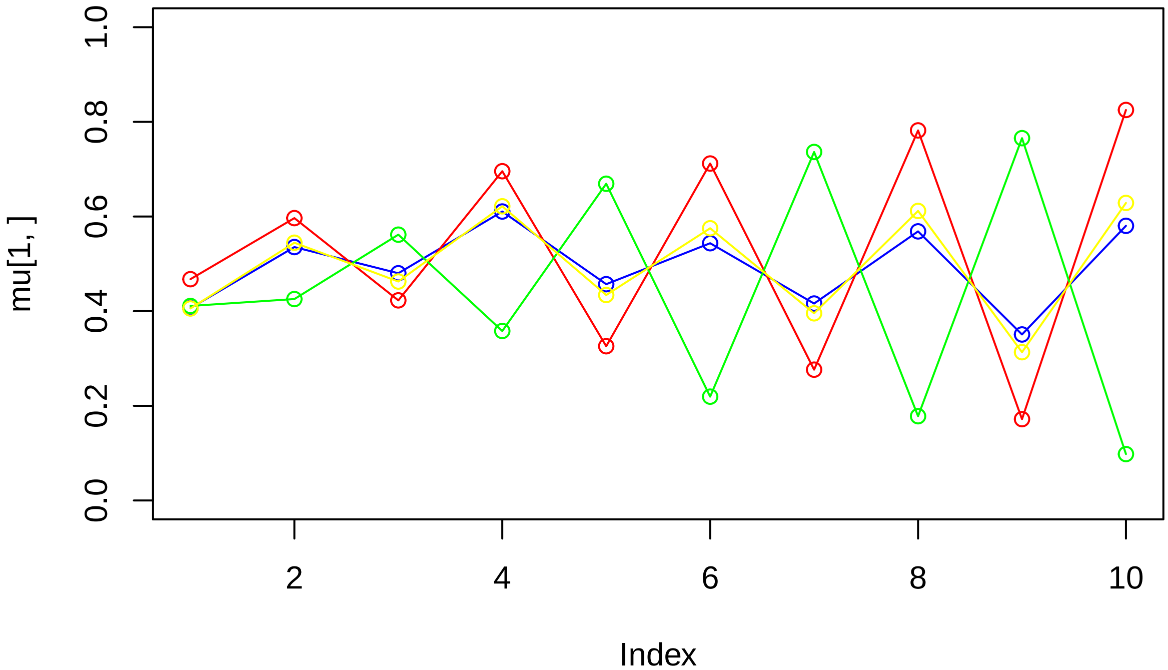

```
## iteration:  5 log likelihood:  -6525.998
```



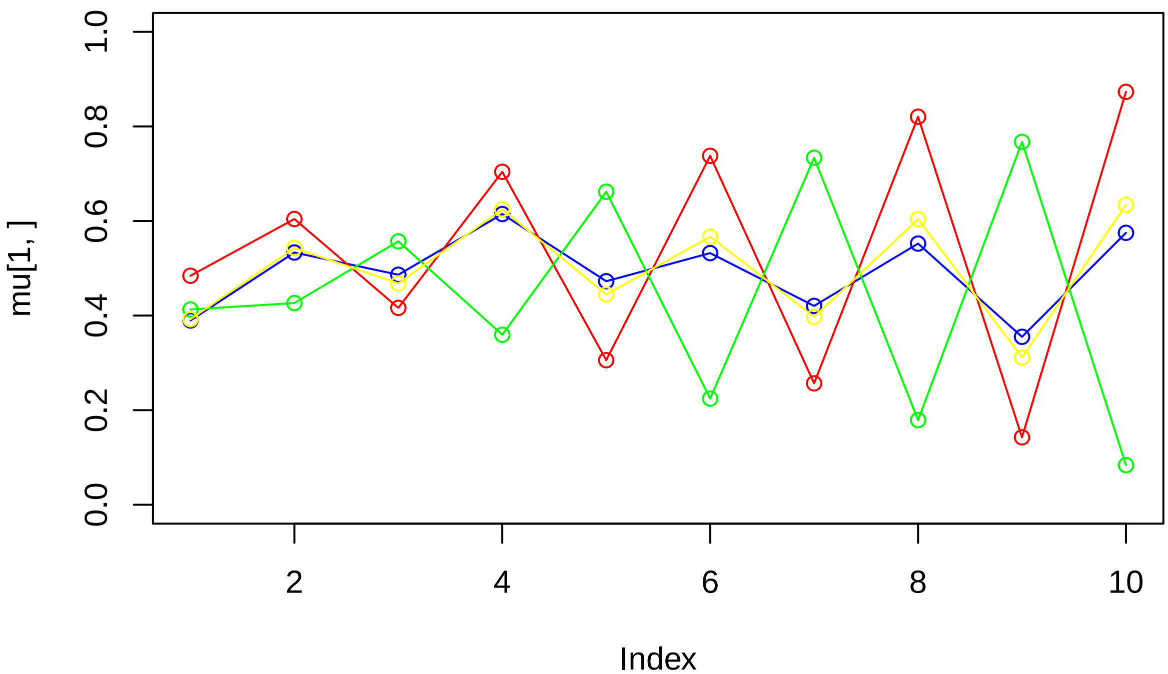

```
## iteration:  6 log likelihood:  -6241.218
```



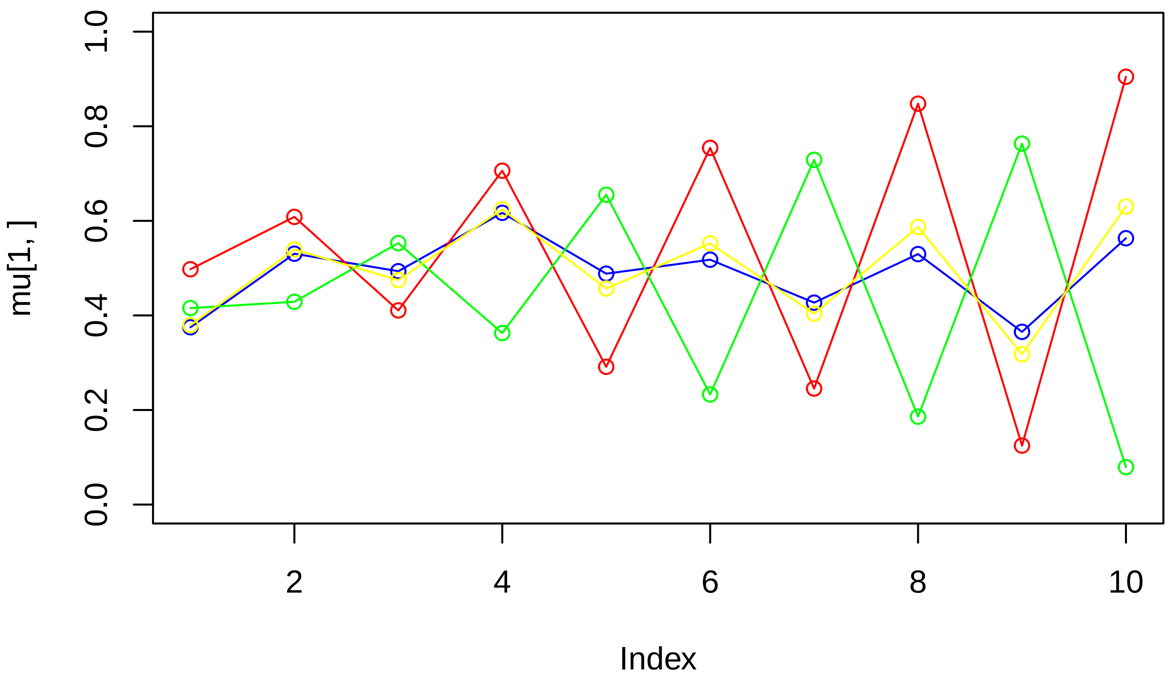

```
## iteration:  7 log likelihood:  -5962.695
```



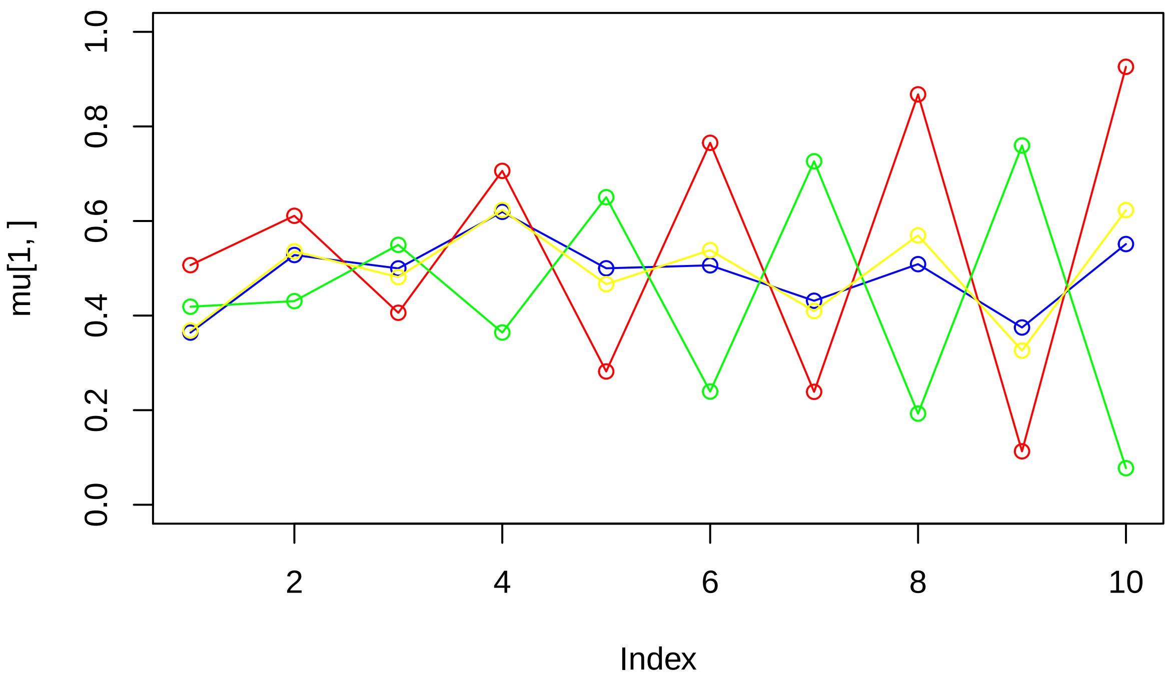

```
## iteration:  8 log likelihood:  -5852.656
```



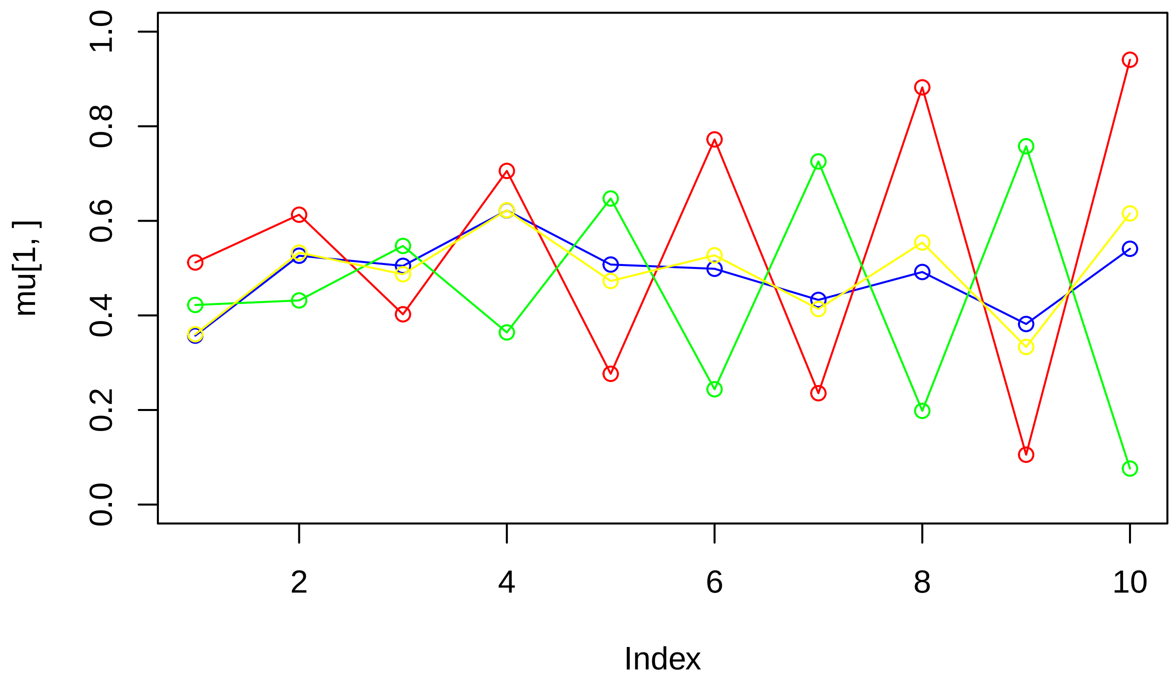

```
## iteration:  9 log likelihood:  -5804.099
```



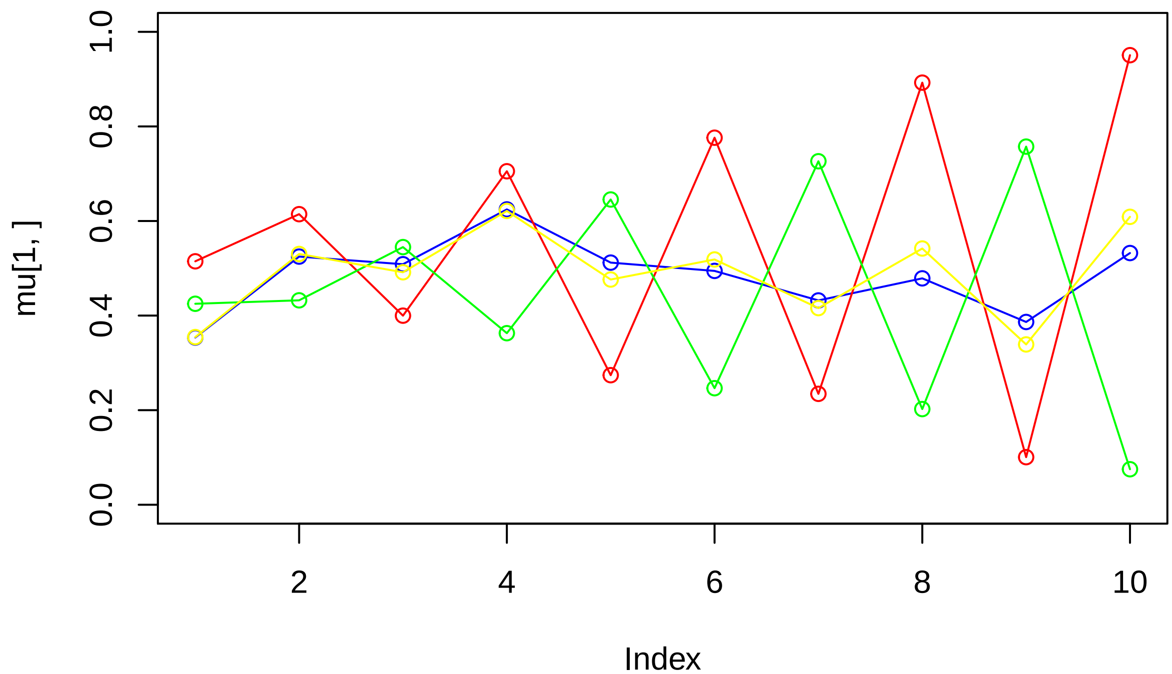

```
## iteration:  10 log likelihood:  -5774.053
```



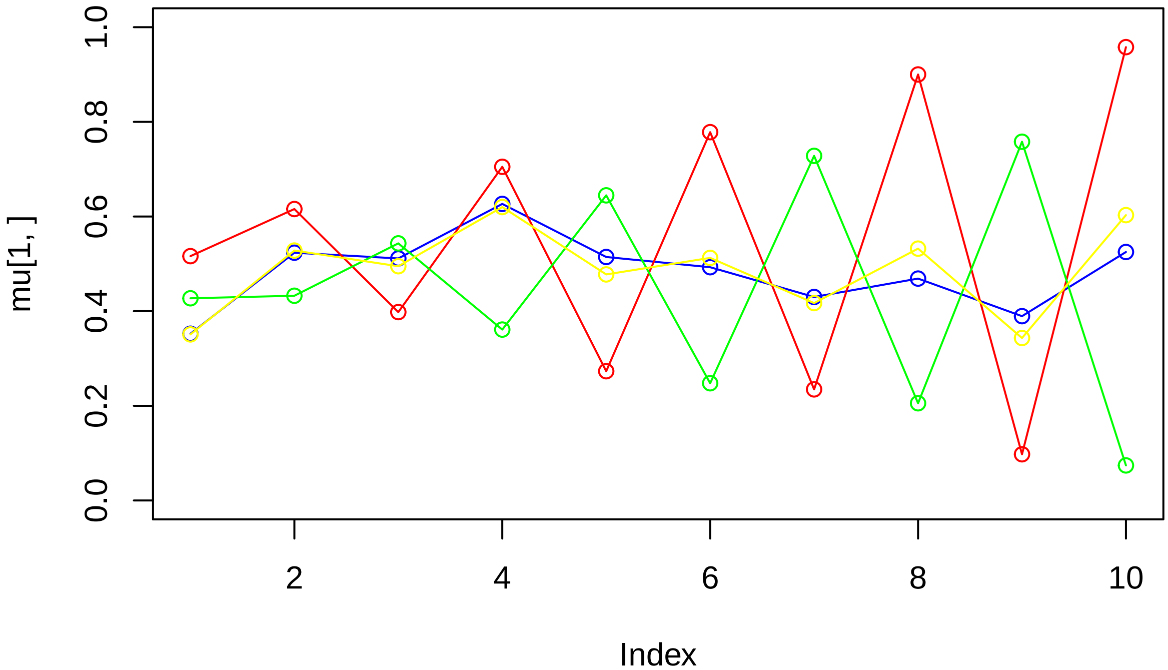

```
## iteration:  11 log likelihood:  -5754.55
```



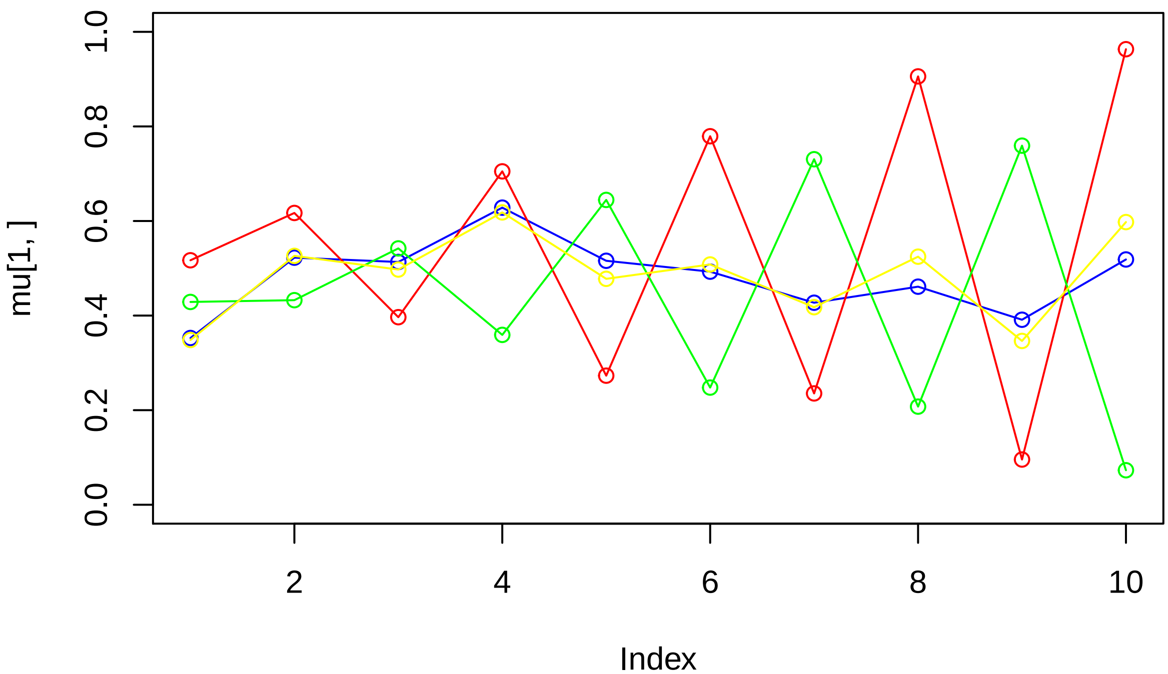

```
## iteration:  12 log likelihood:  -5741.968
```



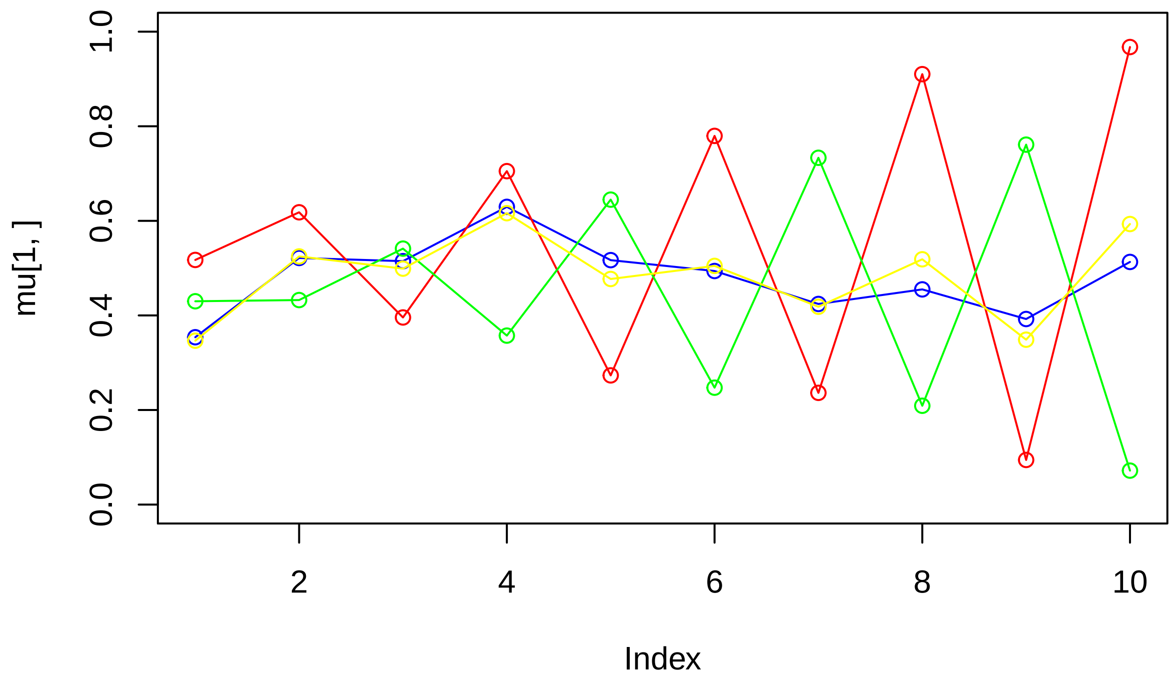

```
## iteration:  13 log likelihood:  -5733.572
```



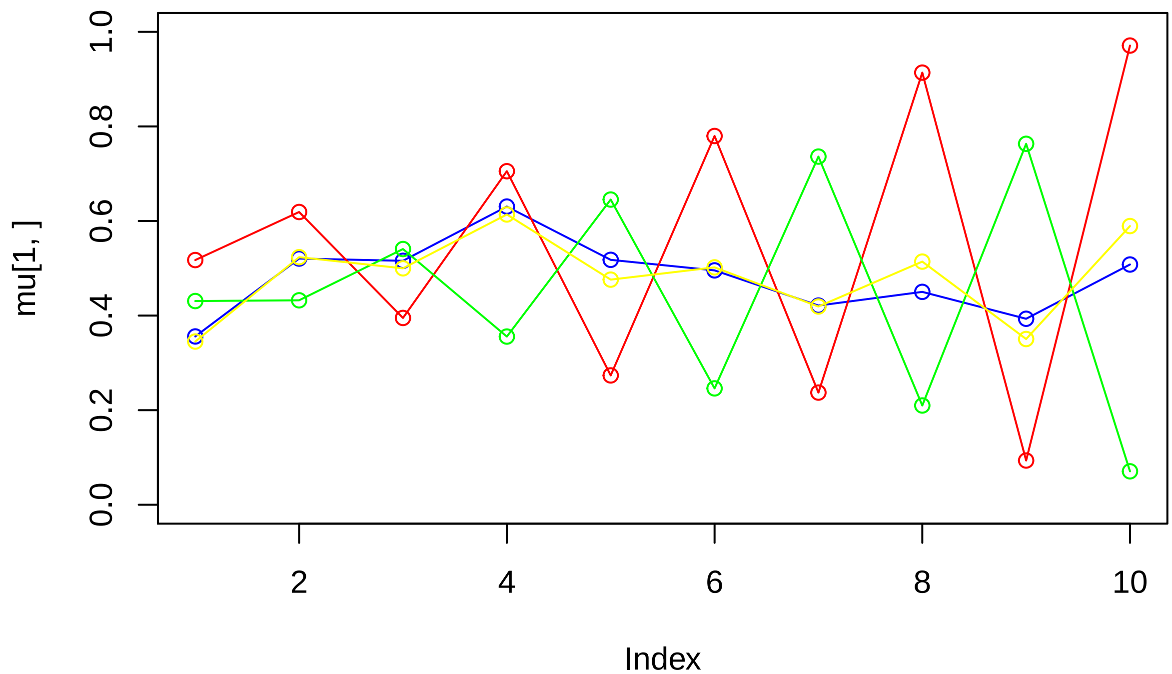

```
## iteration:  14 log likelihood:  -5727.573
```



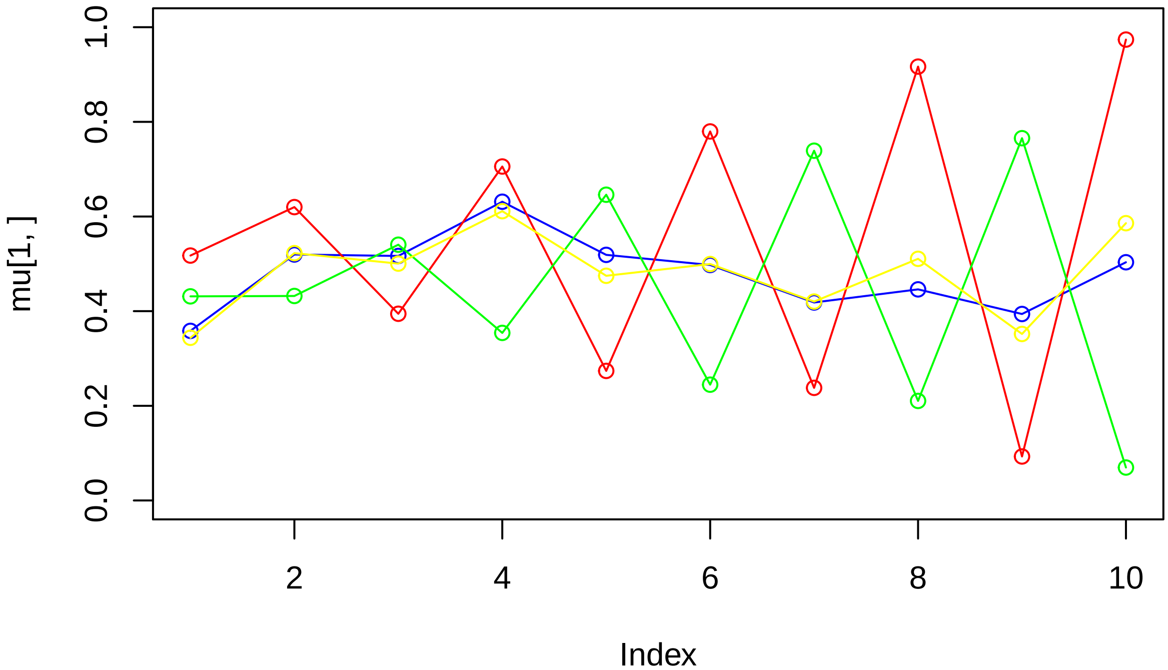

```
## iteration:  15 log likelihood:  -5722.953
```



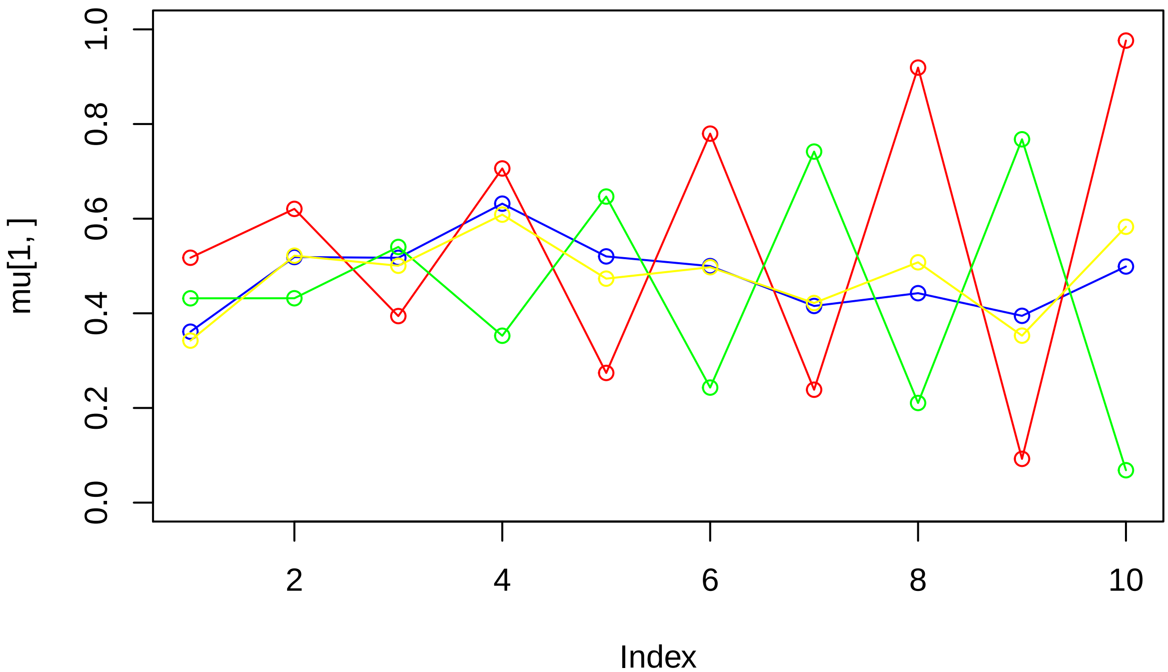

```
## iteration:  16 log likelihood:  -5719.164
```



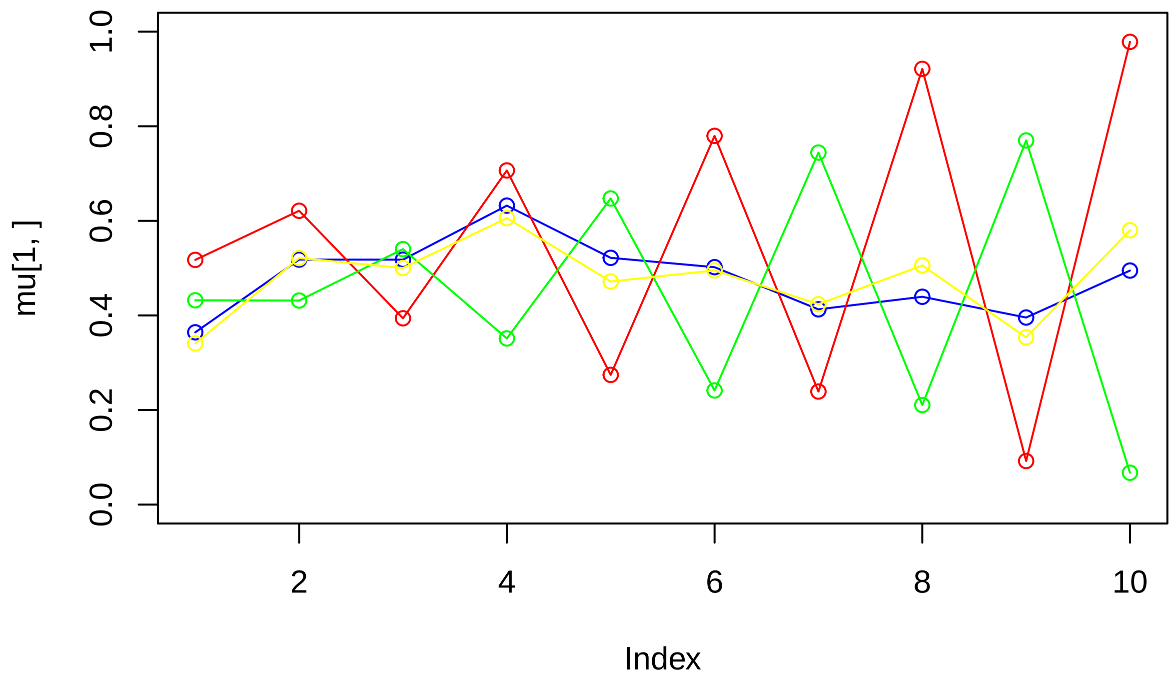

```
## iteration:  17 log likelihood:  -5715.91
```



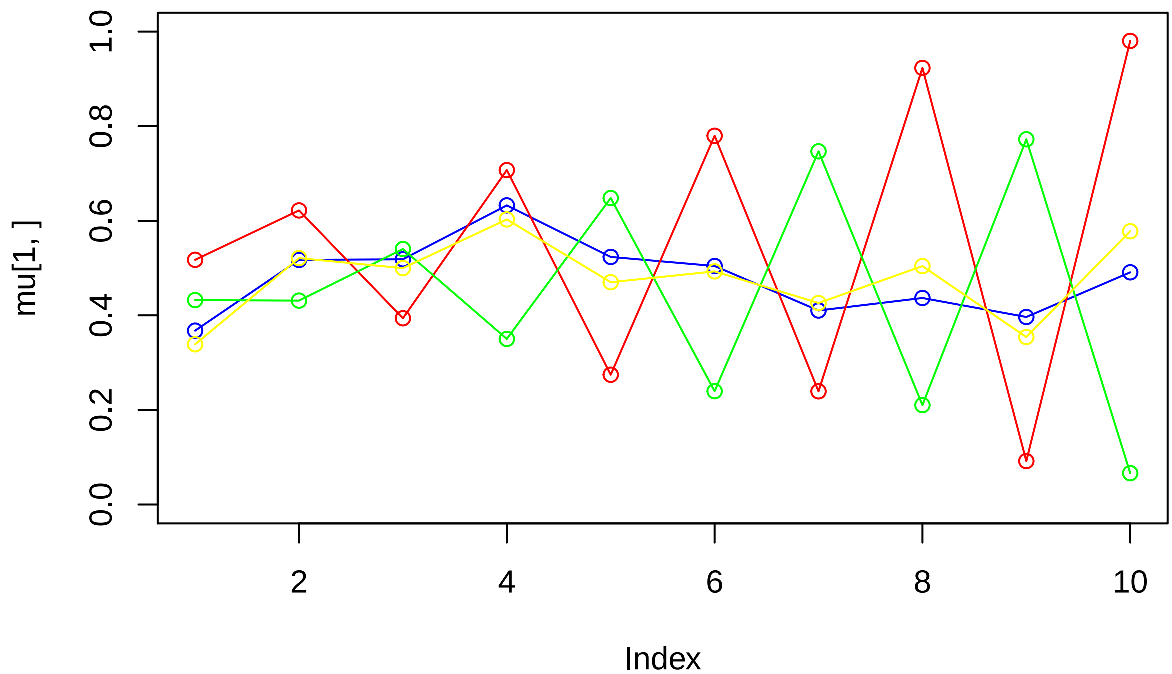

```
## iteration:  18 log likelihood:  -5713.019
```



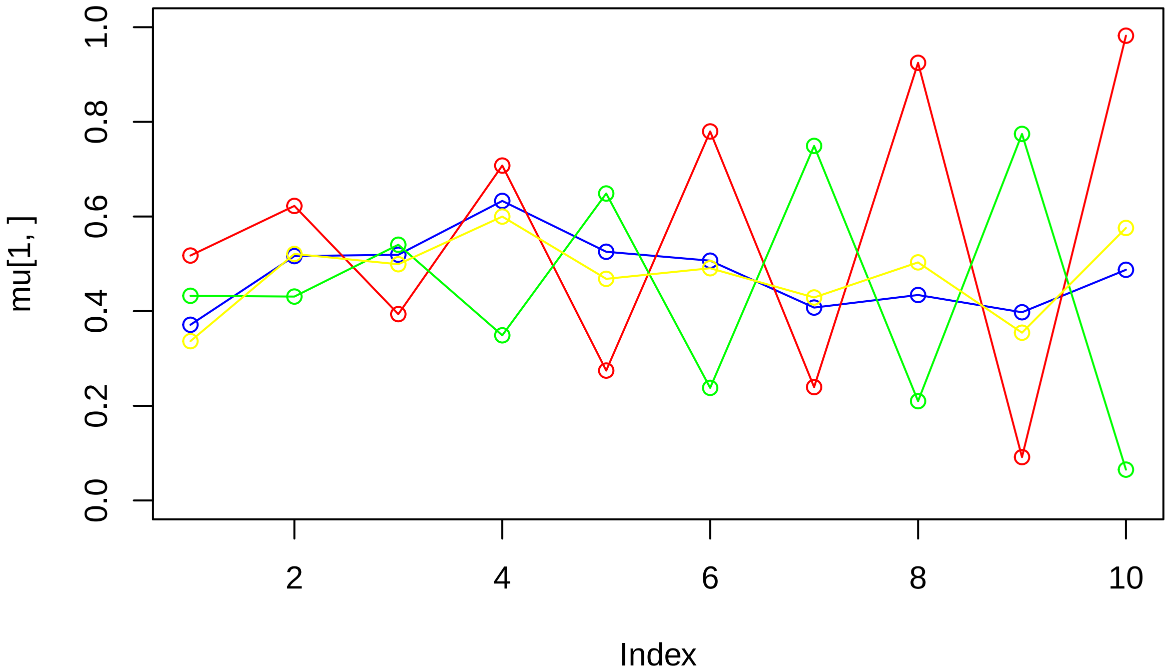

```
## iteration:  19 log likelihood:  -5710.383
```



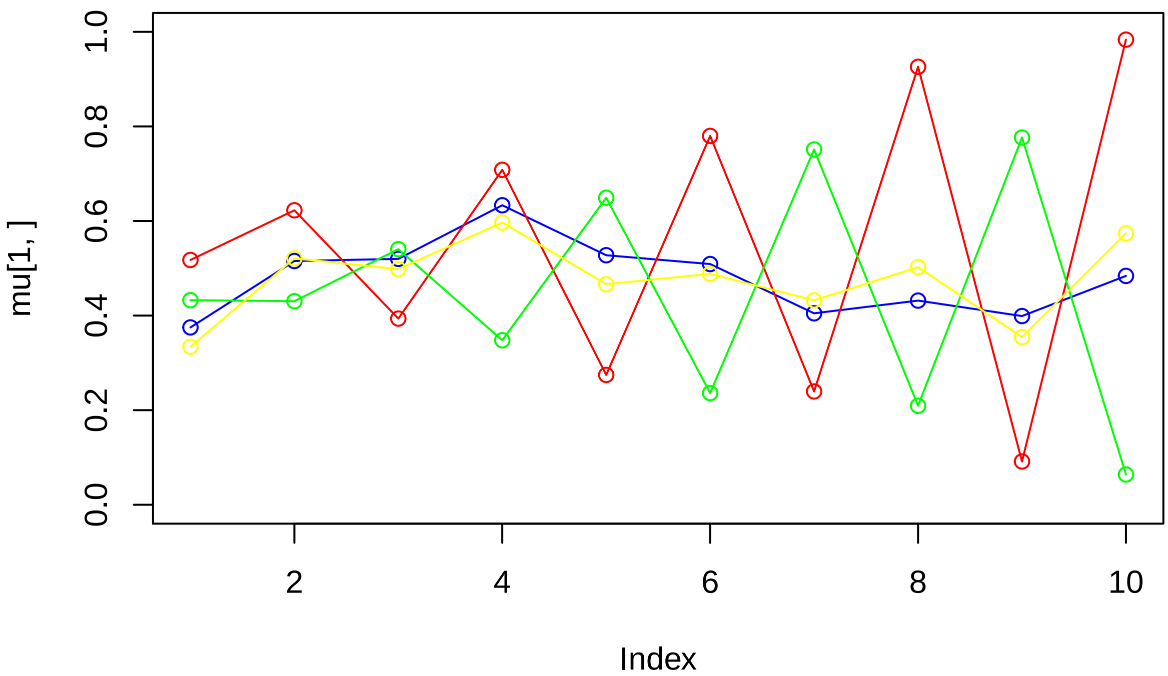

```
## iteration:  20 log likelihood:  -5707.926
```



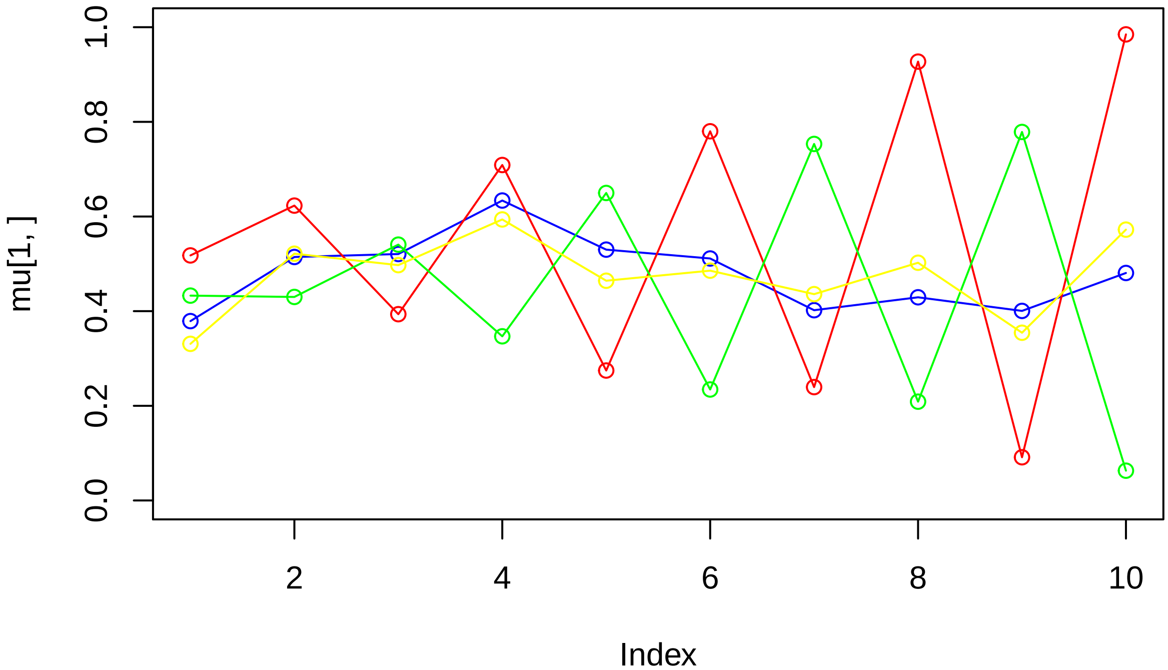

```
## iteration:  21 log likelihood:  -5705.593
```



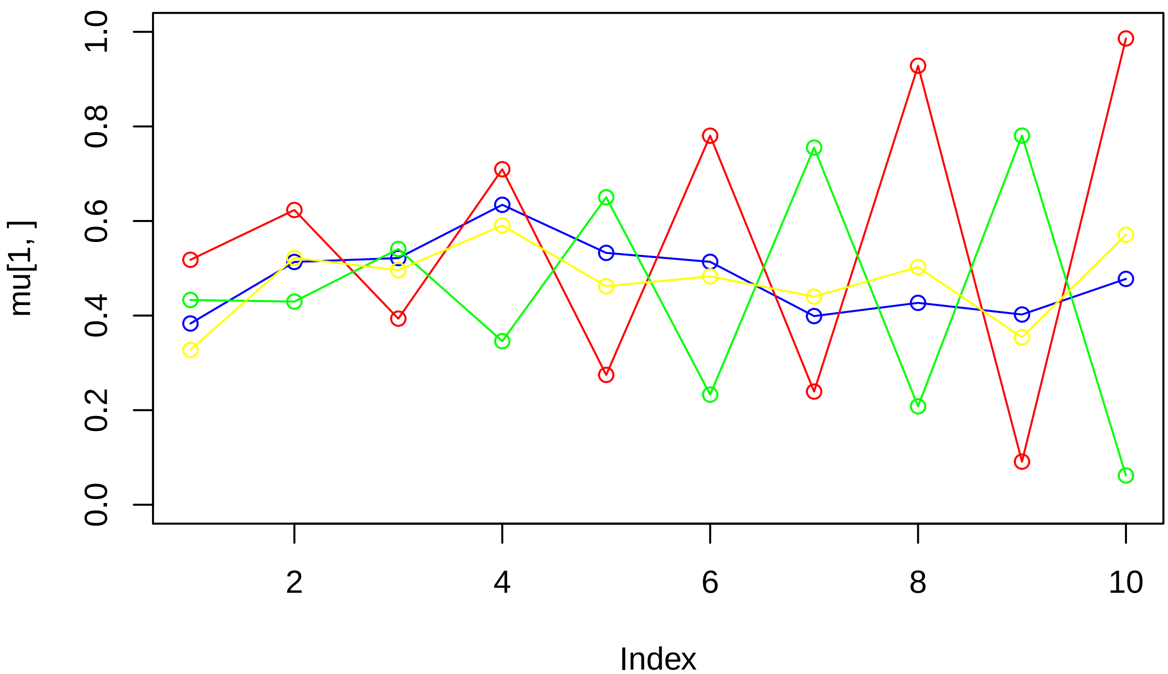

```
## iteration:  22 log likelihood:  -5703.335
```



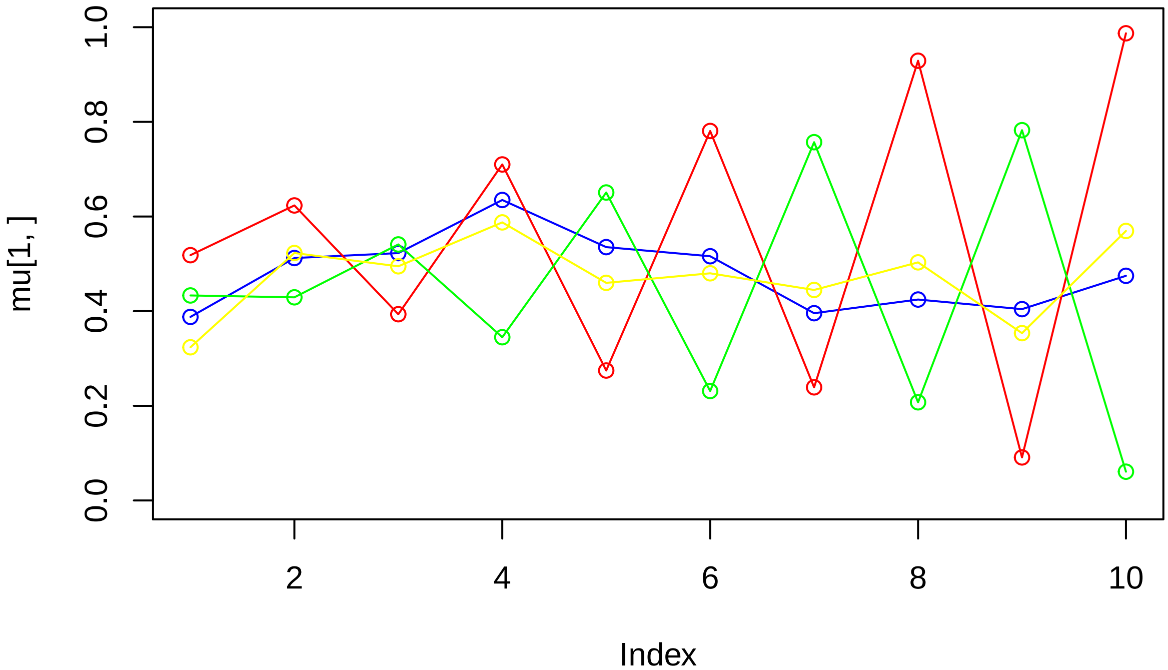

```
## iteration:  23 log likelihood:  -5701.111
```



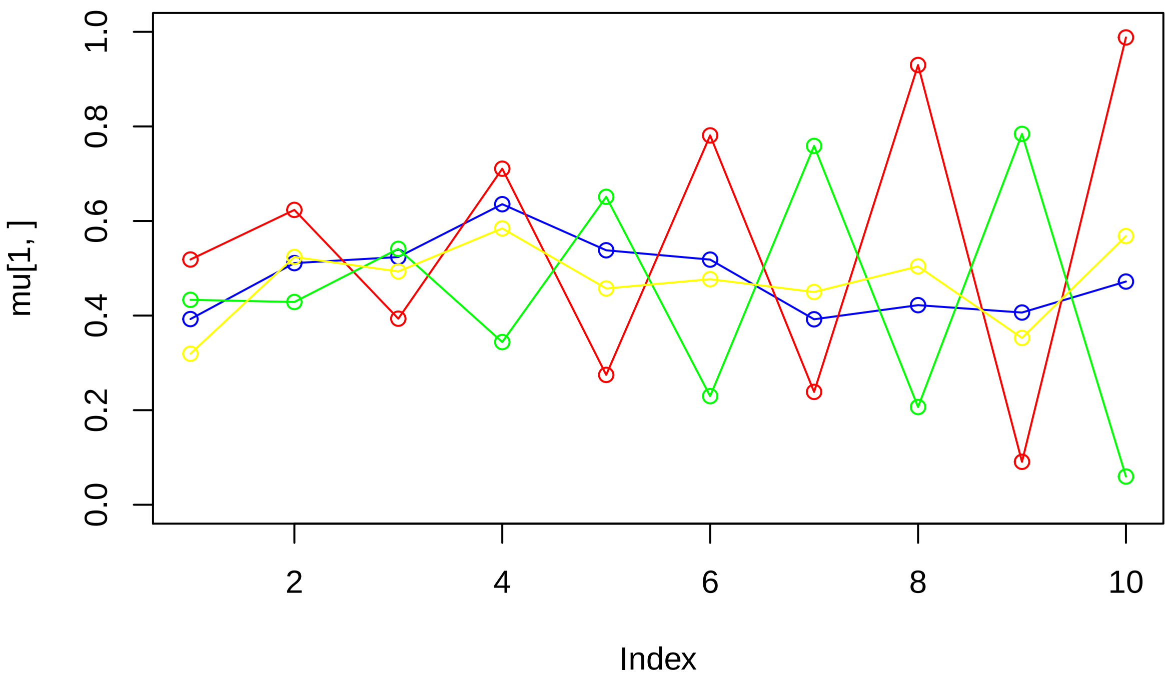
iteration: 24 log likelihood: -5698.879



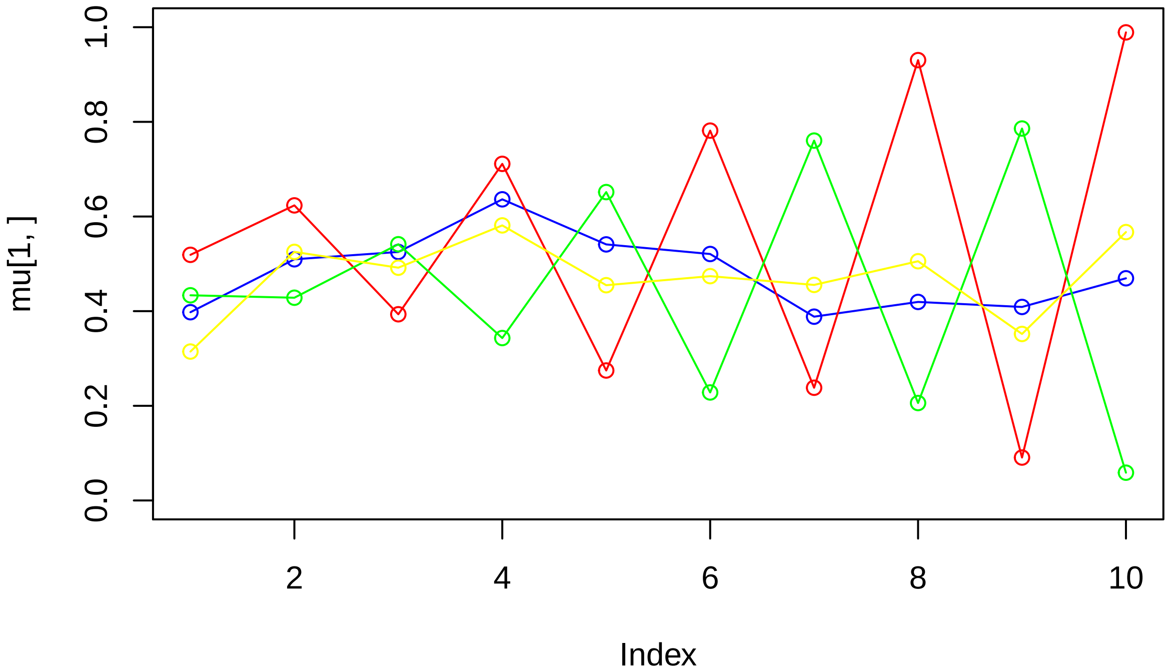
iteration: 25 log likelihood: -5696.603



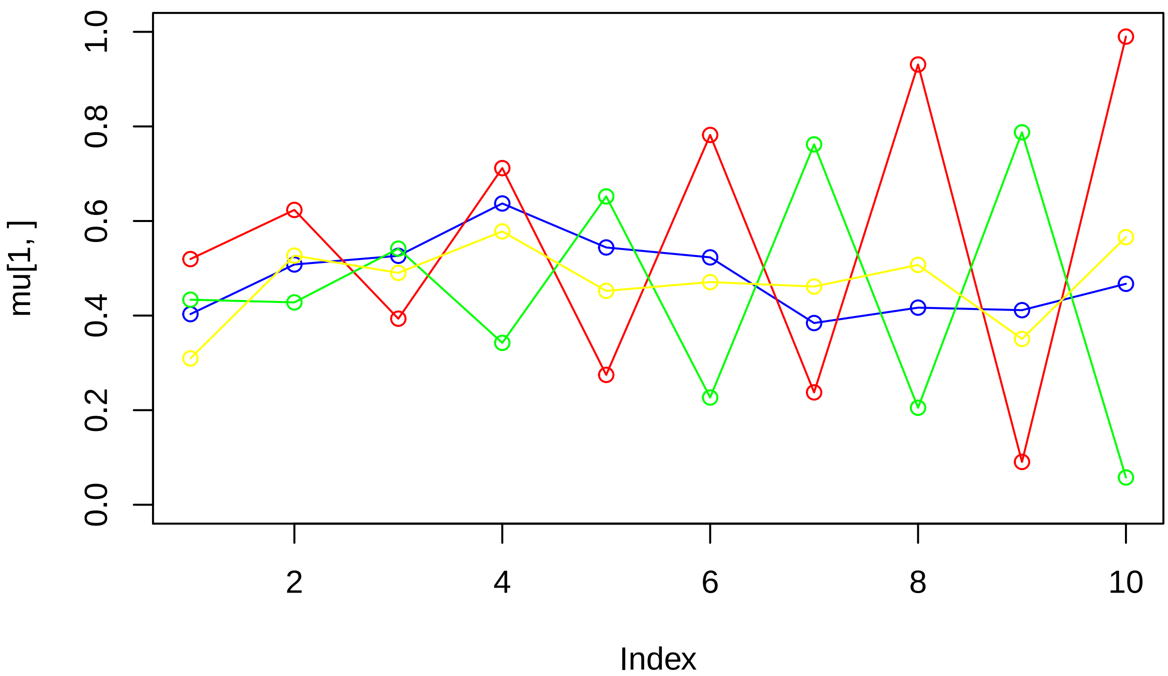

```
## iteration:  26 log likelihood:  -5694.243
```



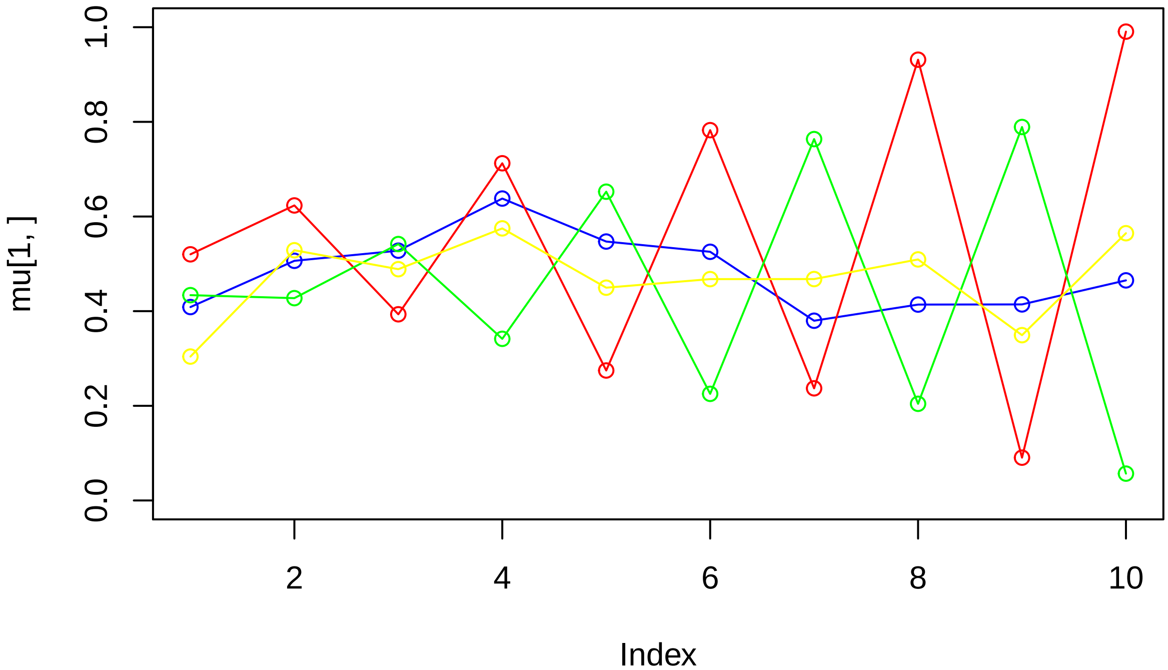

```
## iteration:  27 log likelihood:  -5691.767
```



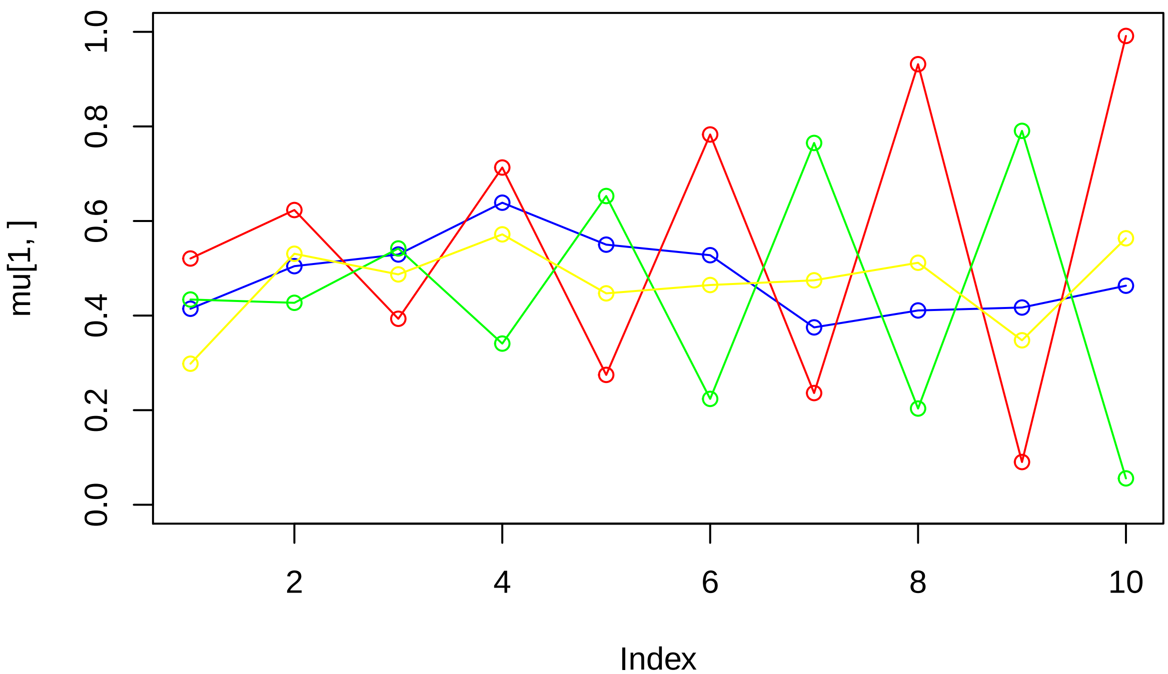

```
## iteration:  28 log likelihood:  -5689.141
```



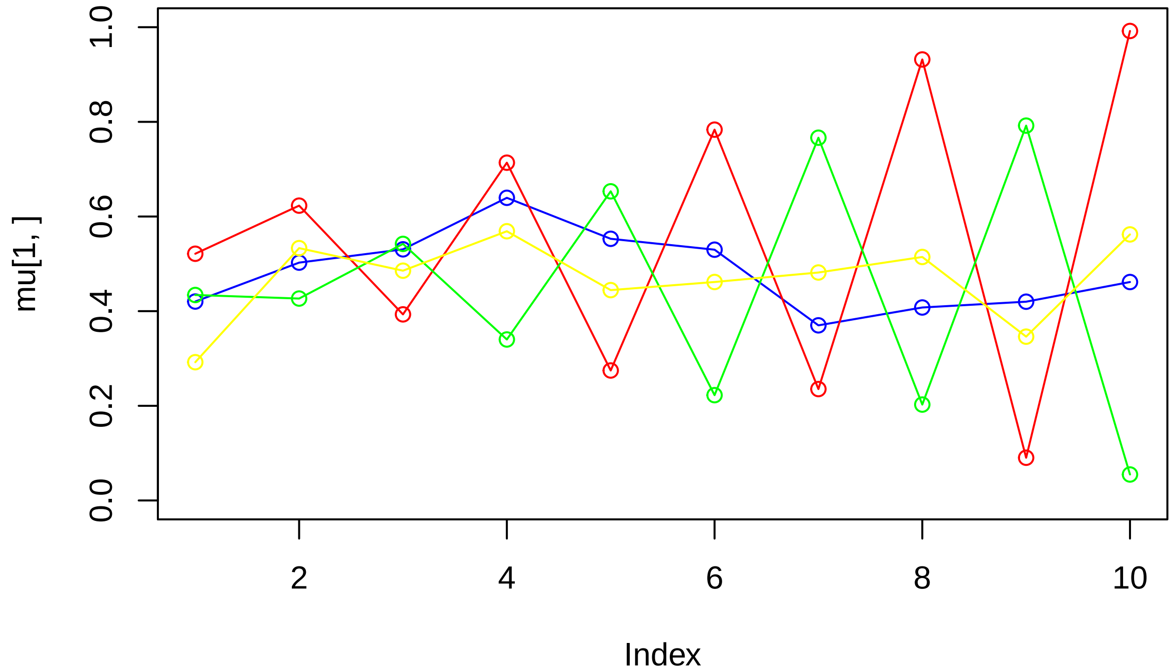

```
## iteration:  29 log likelihood:  -5686.341
```



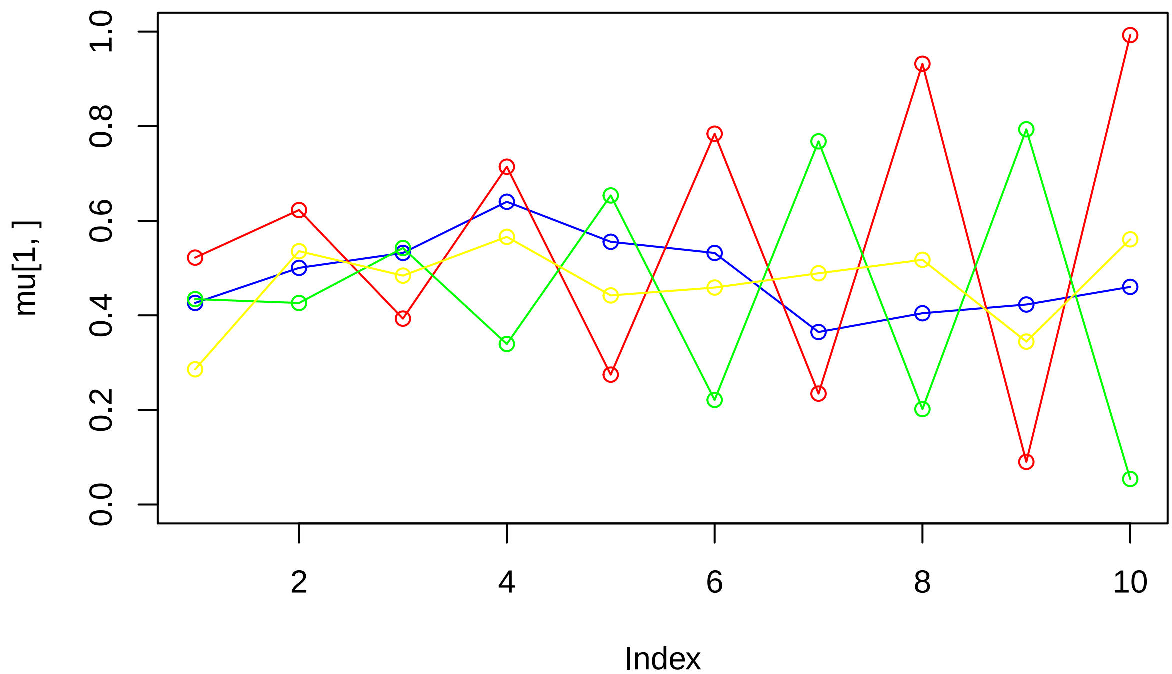

```
## iteration:  30 log likelihood:  -5683.349
```



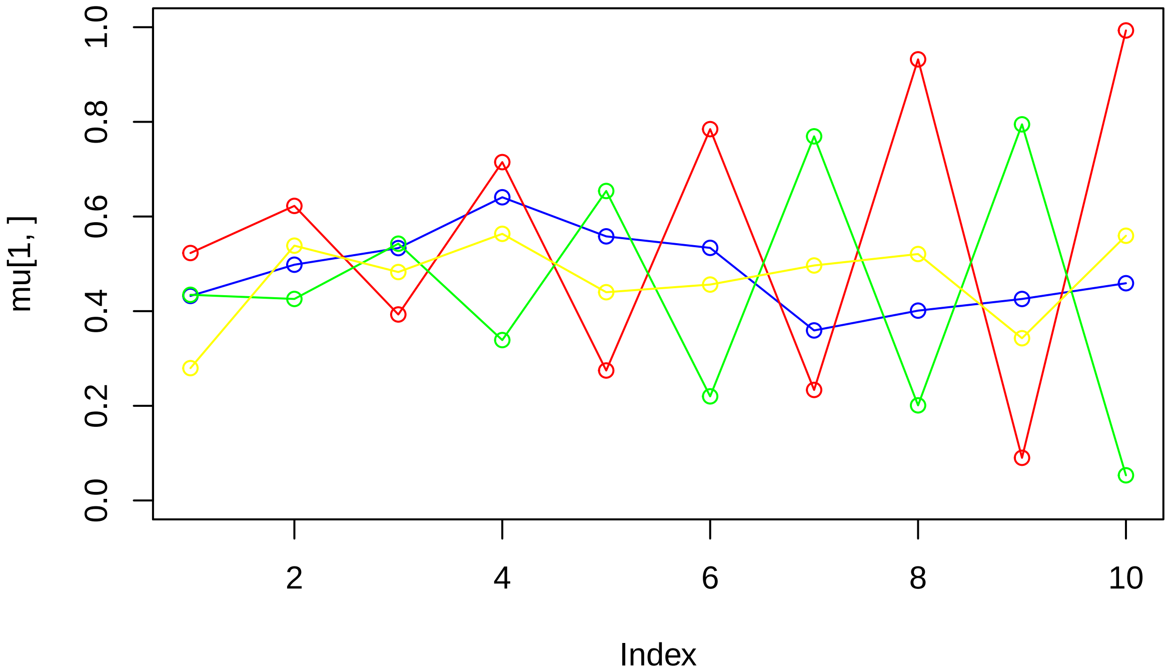

```
## iteration:  31 log likelihood:  -5680.162
```



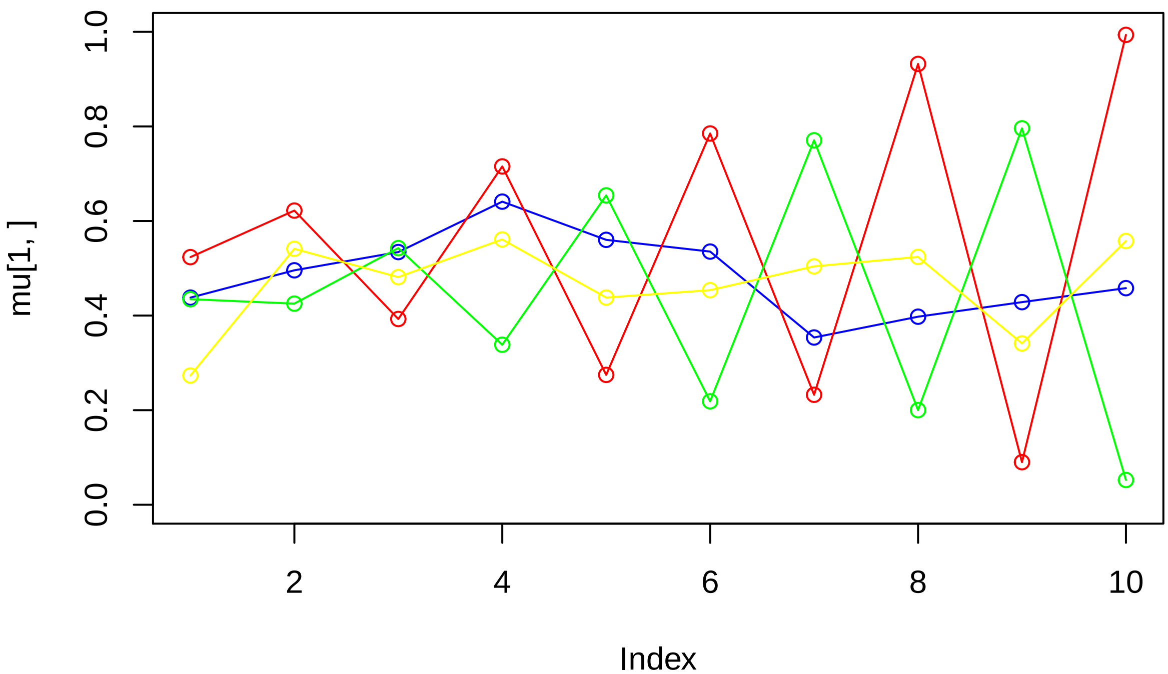

```
## iteration:  32 log likelihood:  -5676.789
```



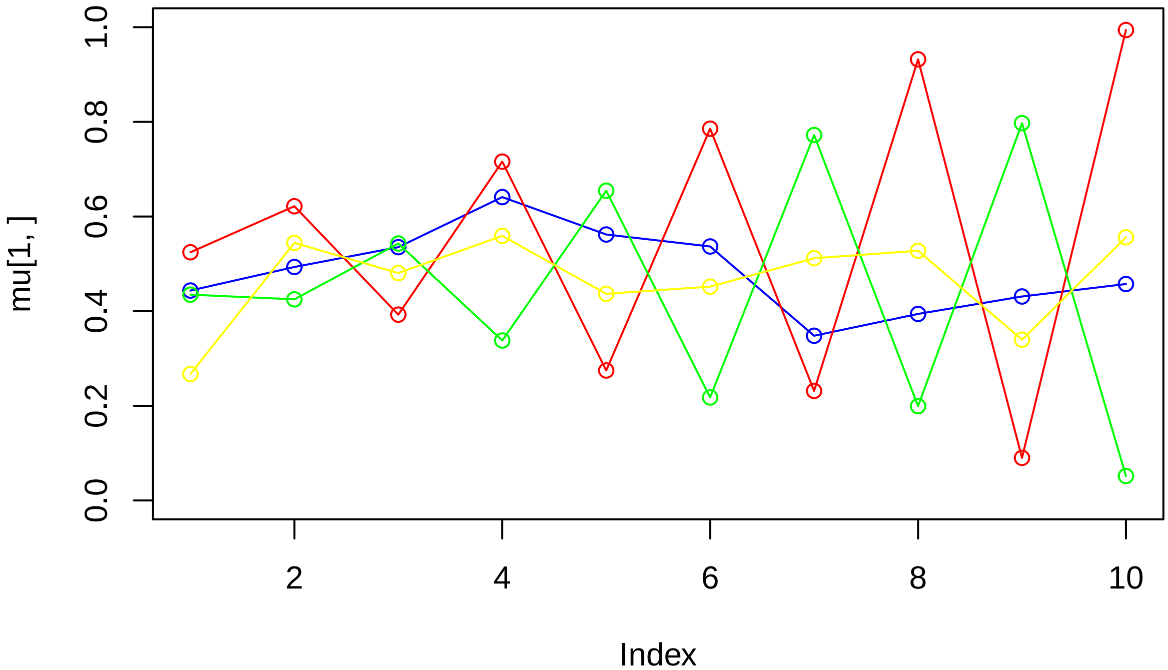

```
## iteration:  33 log likelihood:  -5673.256
```



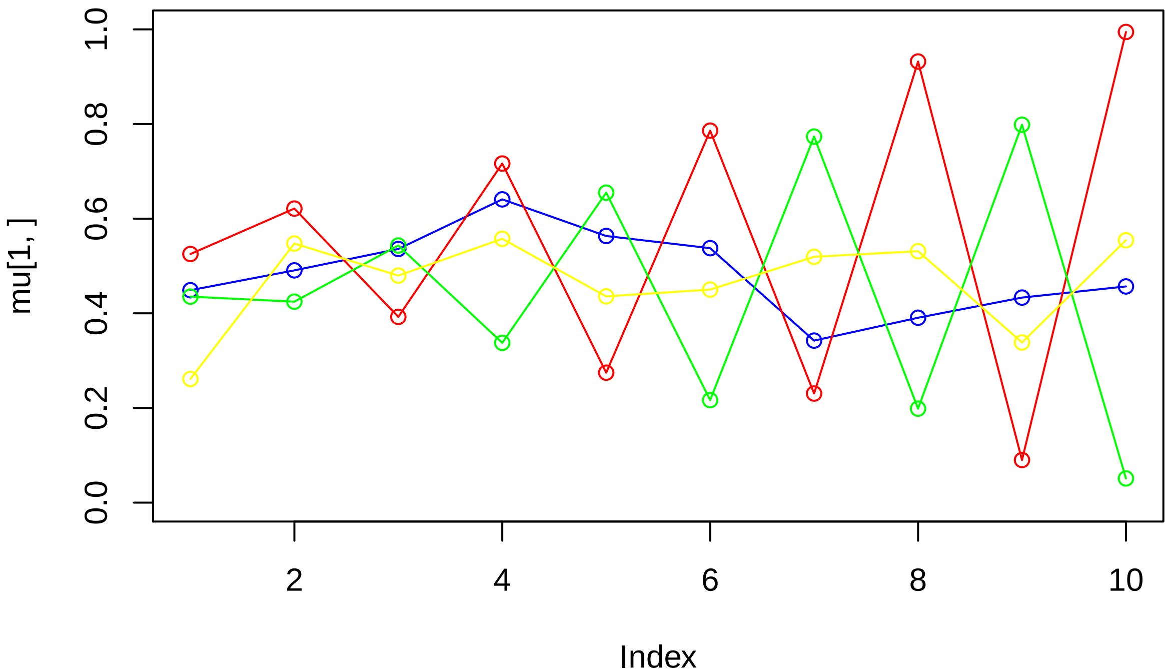

```
## iteration:  34 log likelihood:  -5669.605
```



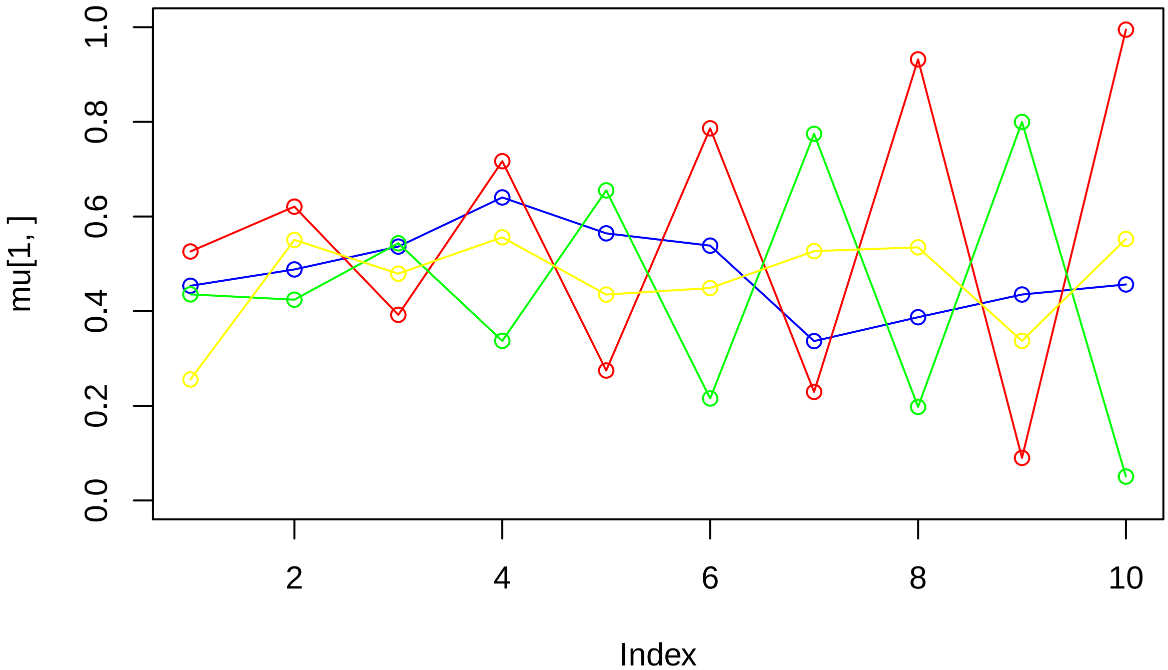

```
## iteration:  35 log likelihood:  -5665.886
```



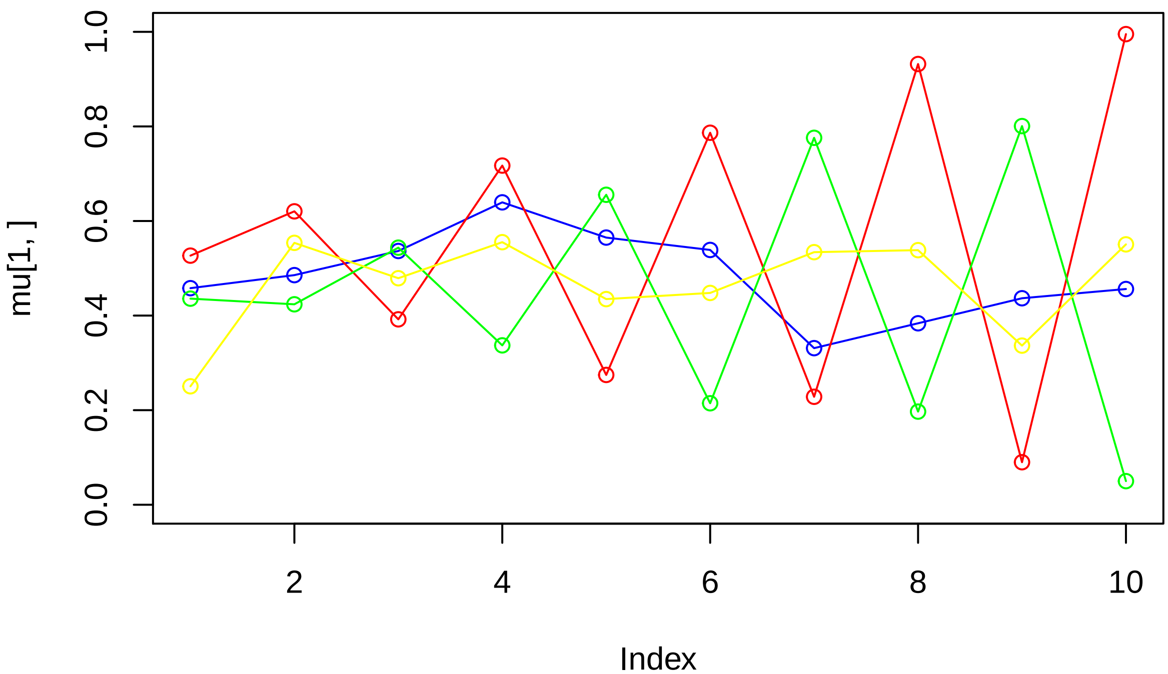

```
## iteration:  36 log likelihood:  -5662.158
```



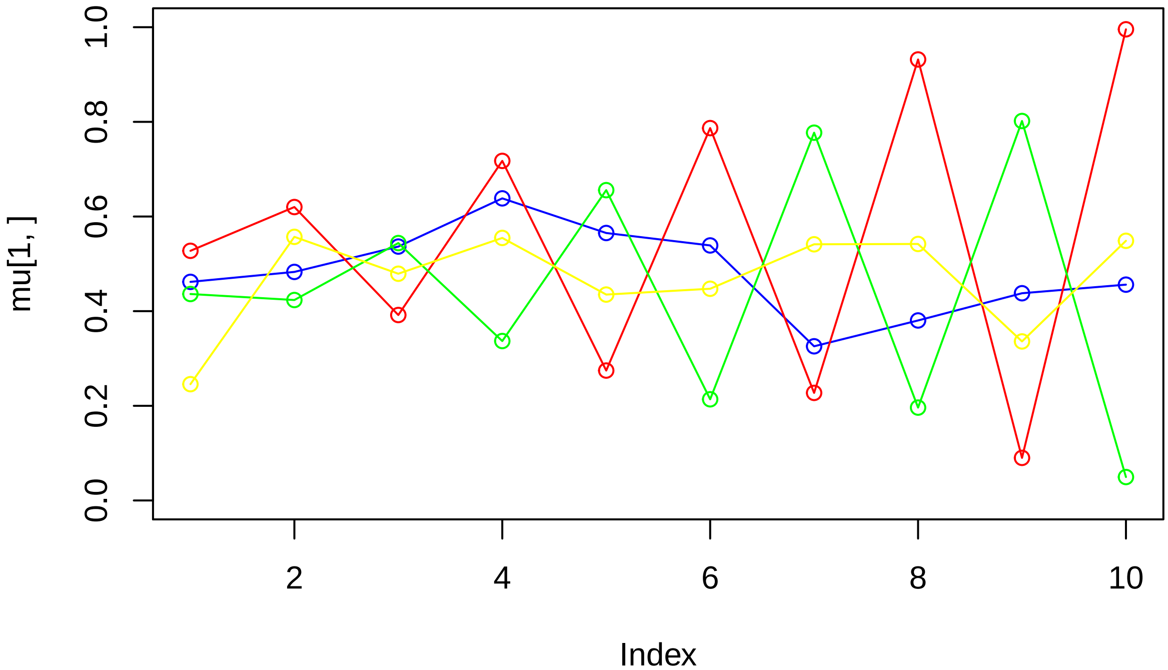

```
## iteration:  37 log likelihood:  -5658.48
```



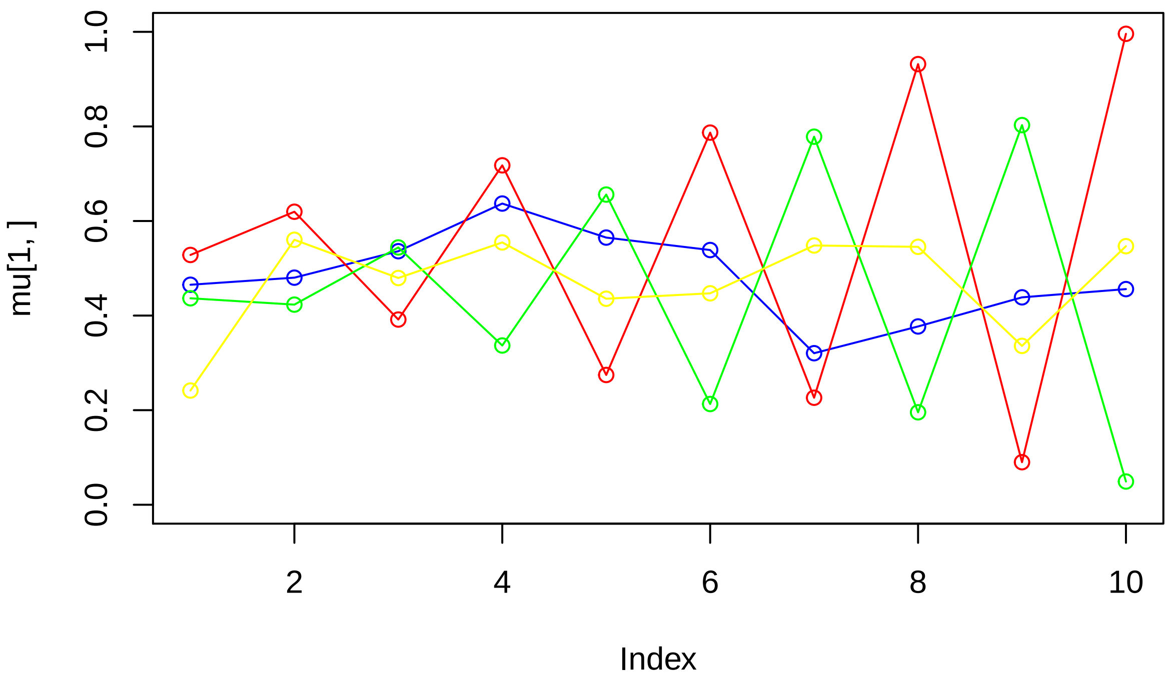

```
## iteration:  38 log likelihood:  -5654.903
```



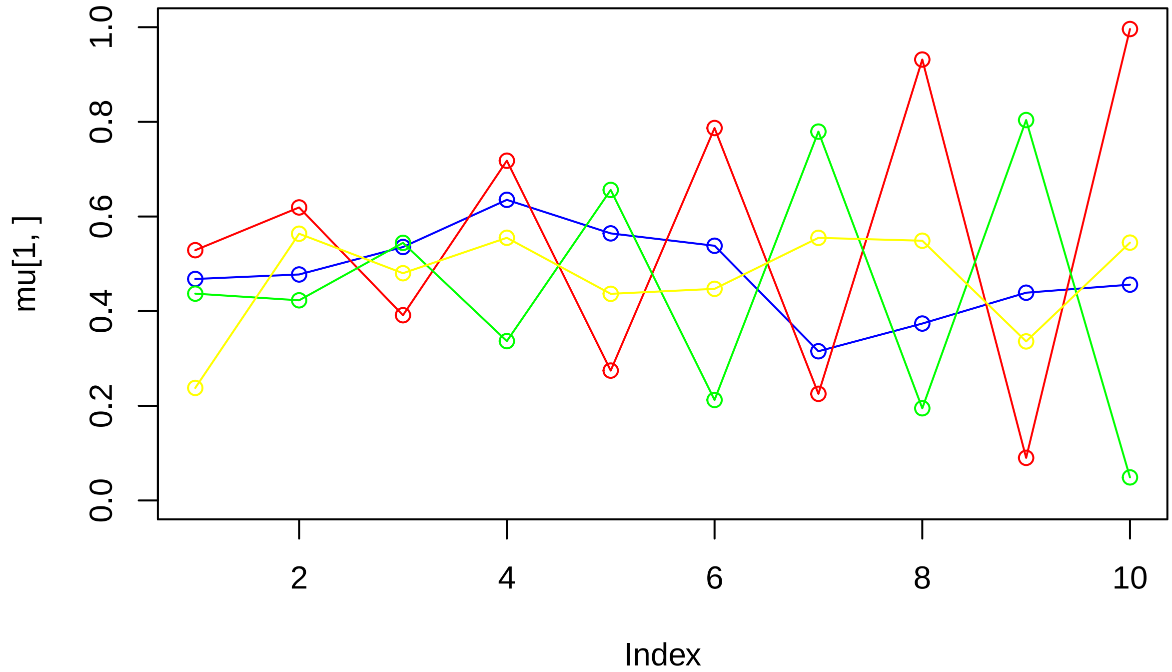

```
## iteration:  39 log likelihood:  -5651.468
```



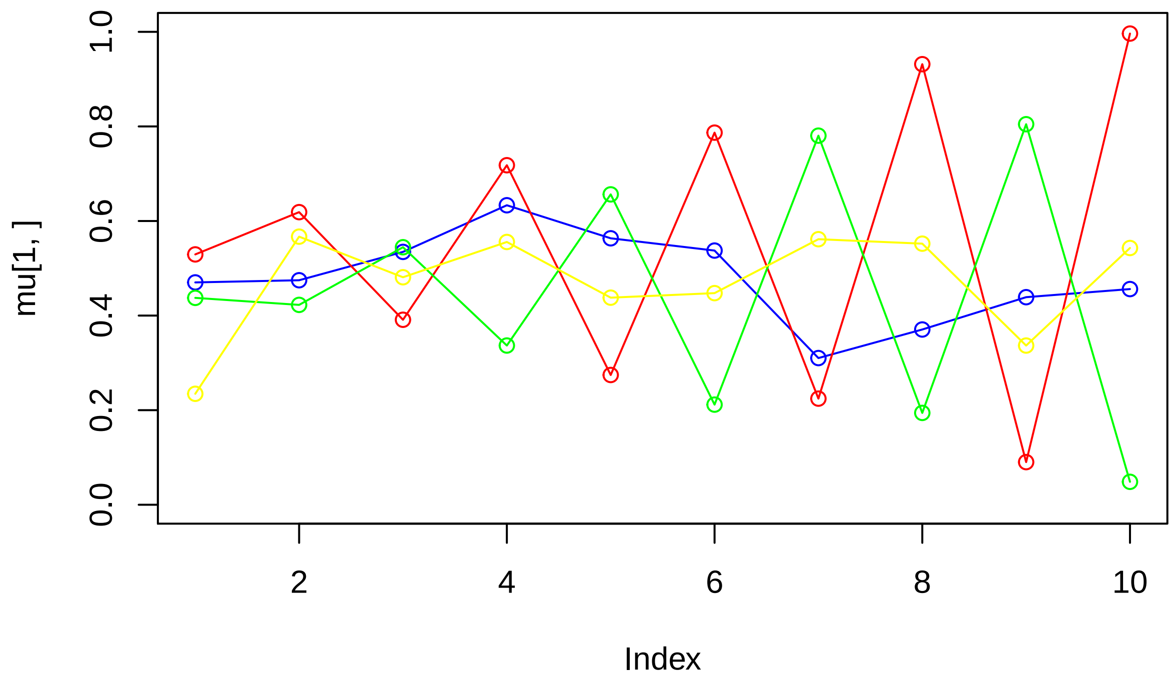

```
## iteration:  40 log likelihood:  -5648.201
```



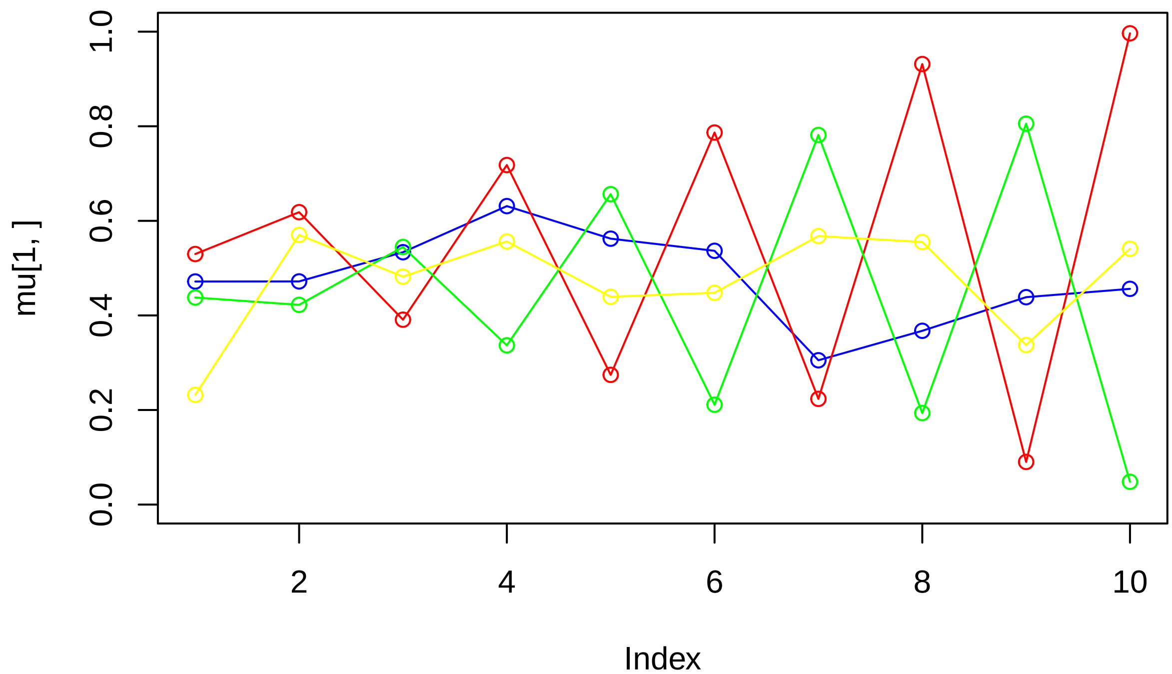

```
## iteration:  41 log likelihood:  -5645.114
```



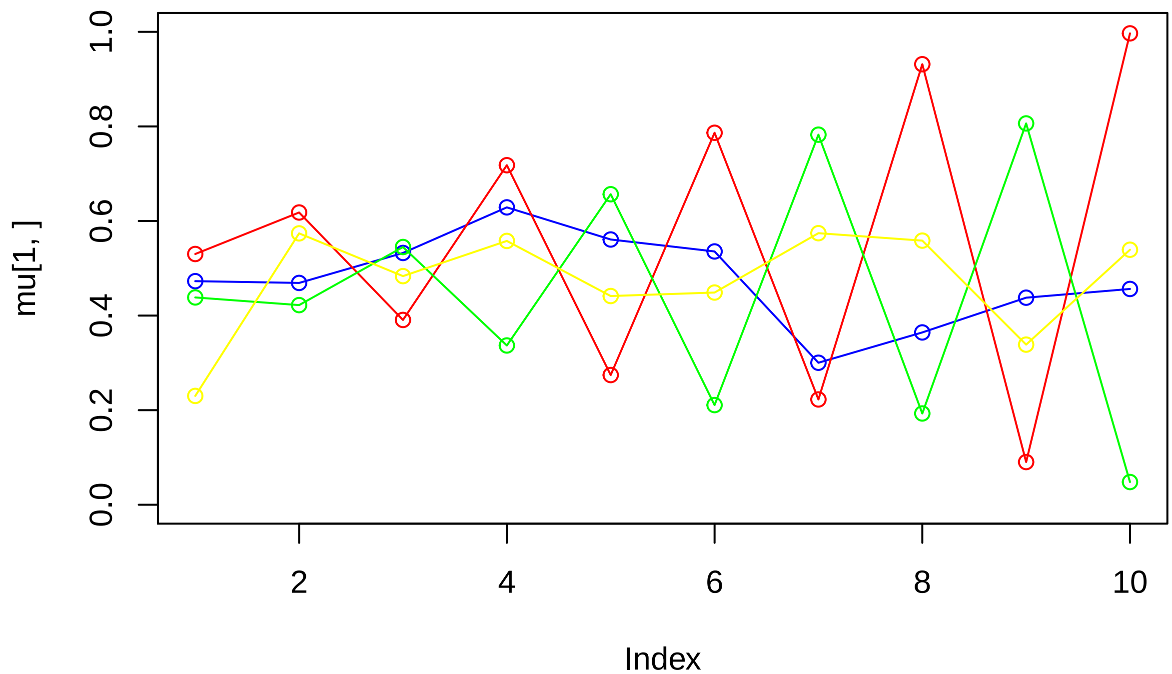

```
## iteration:  42 log likelihood:  -5642.206
```



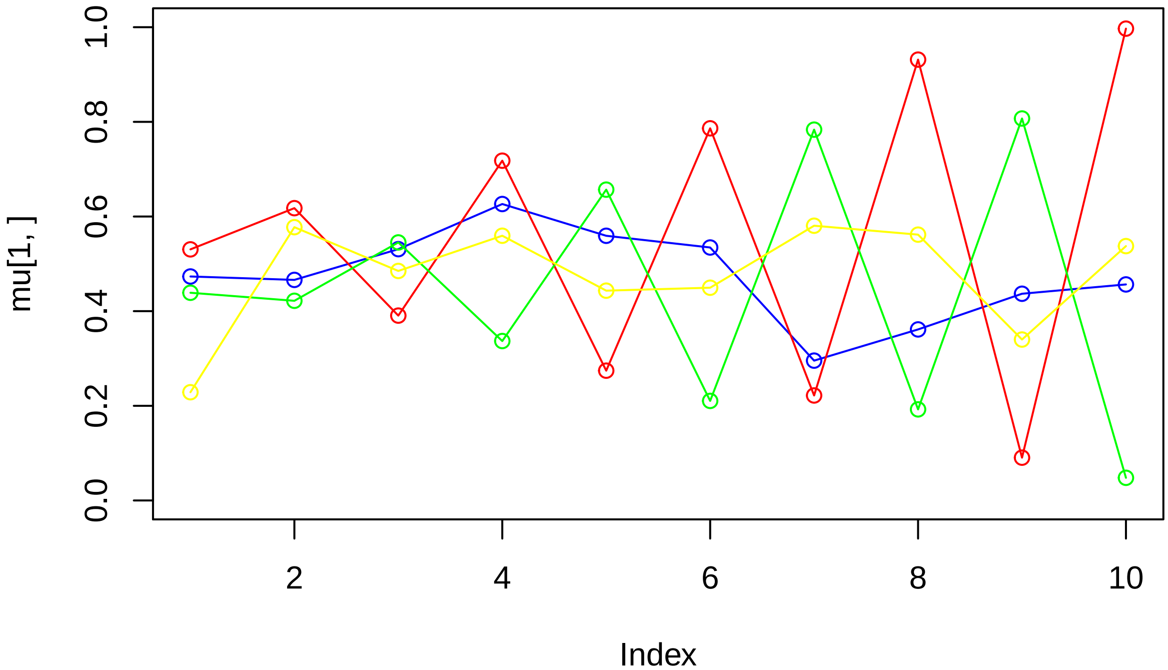

```
## iteration:  43 log likelihood:  -5639.468
```



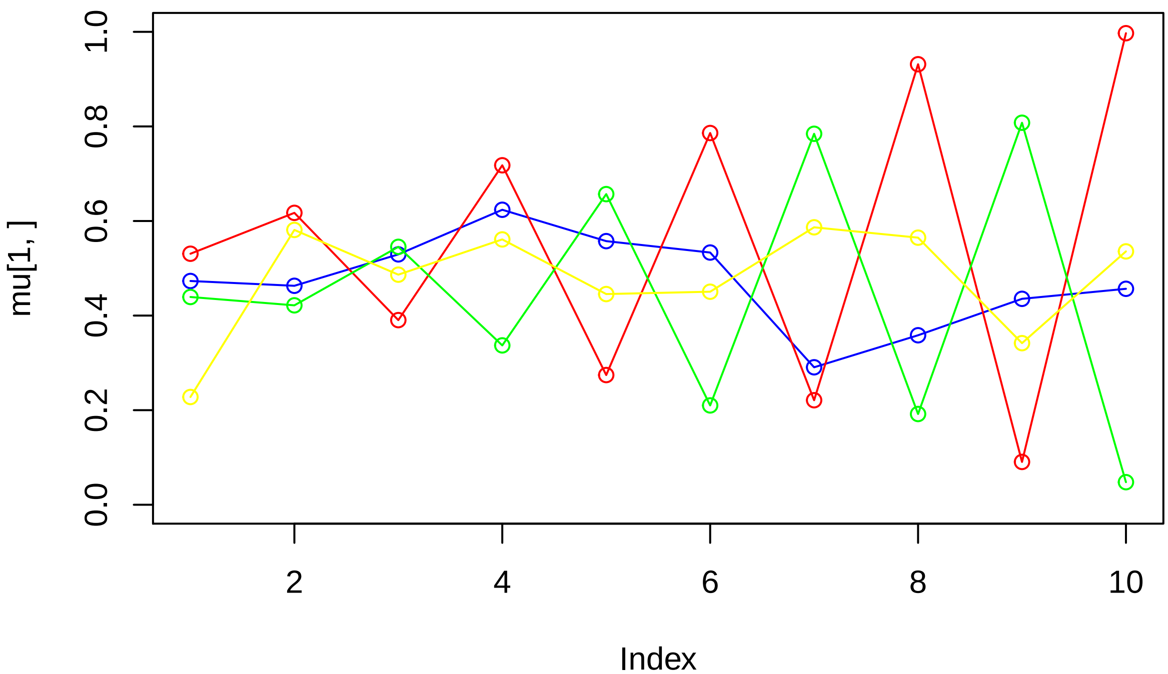

```
## iteration:  44 log likelihood:  -5636.879
```



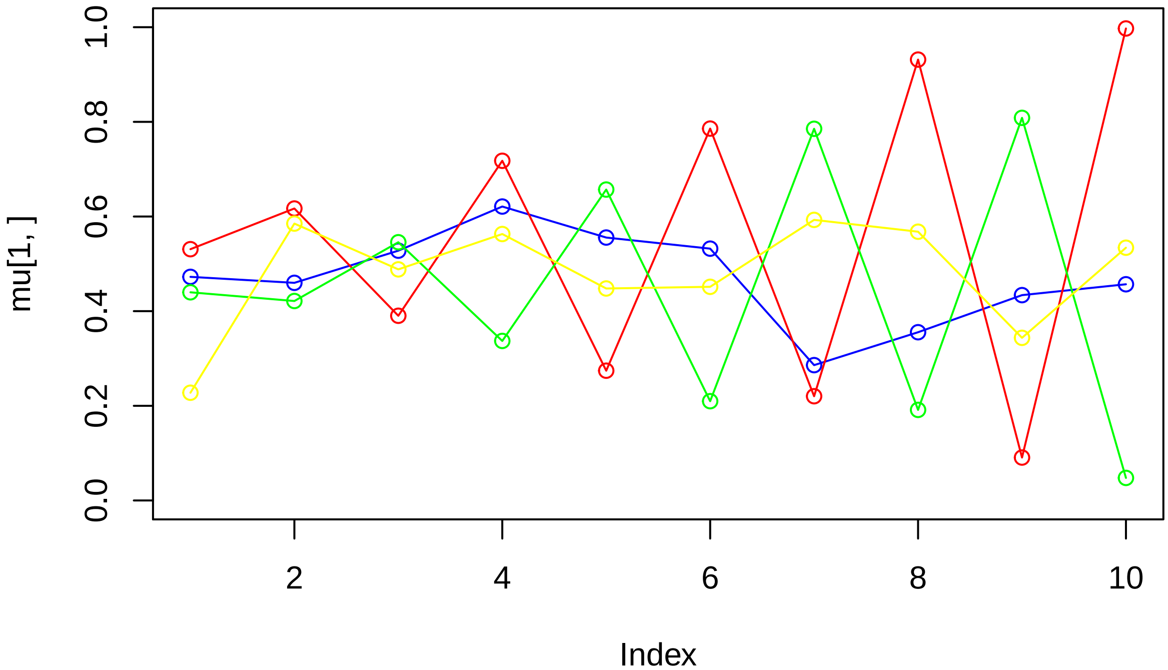

```
## iteration:  45 log likelihood:  -5634.418
```



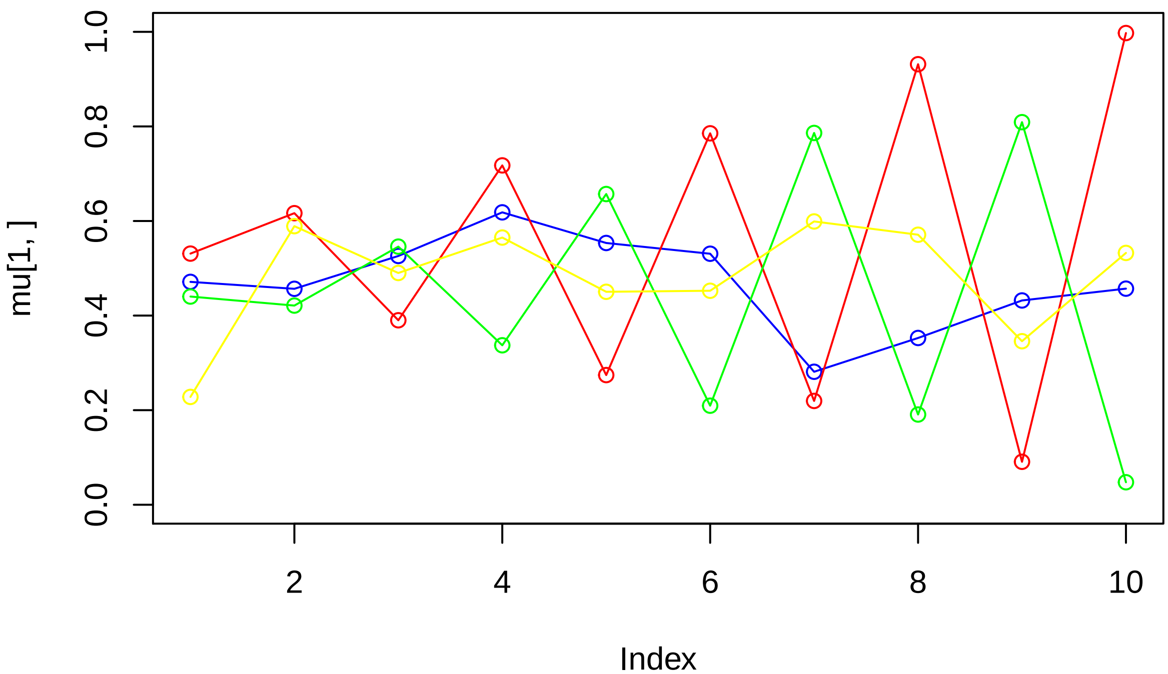

```
## iteration:  46 log likelihood:  -5632.056
```



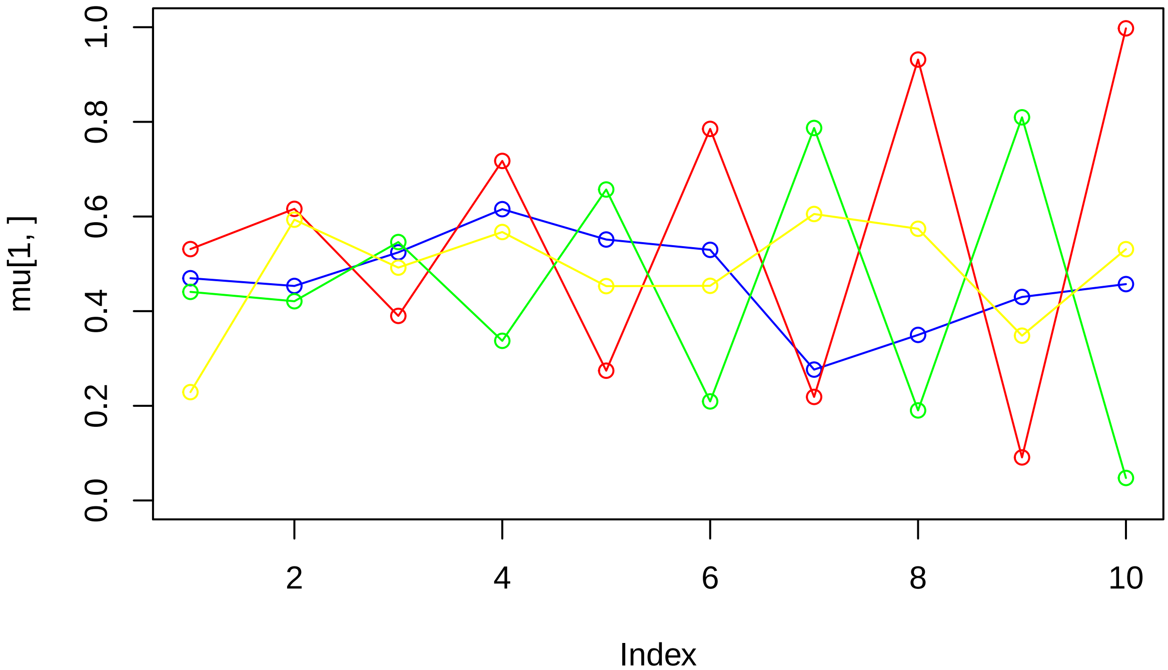

```
## iteration:  47 log likelihood:  -5629.768
```



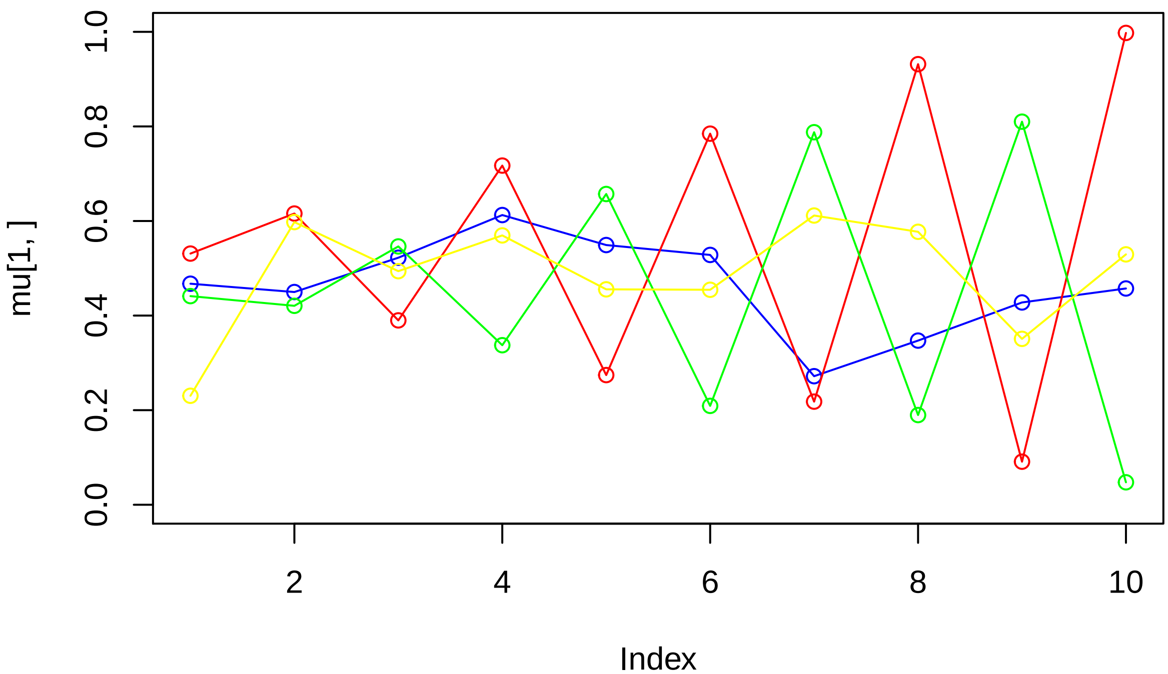

```
## iteration:  48 log likelihood:  -5627.526
```



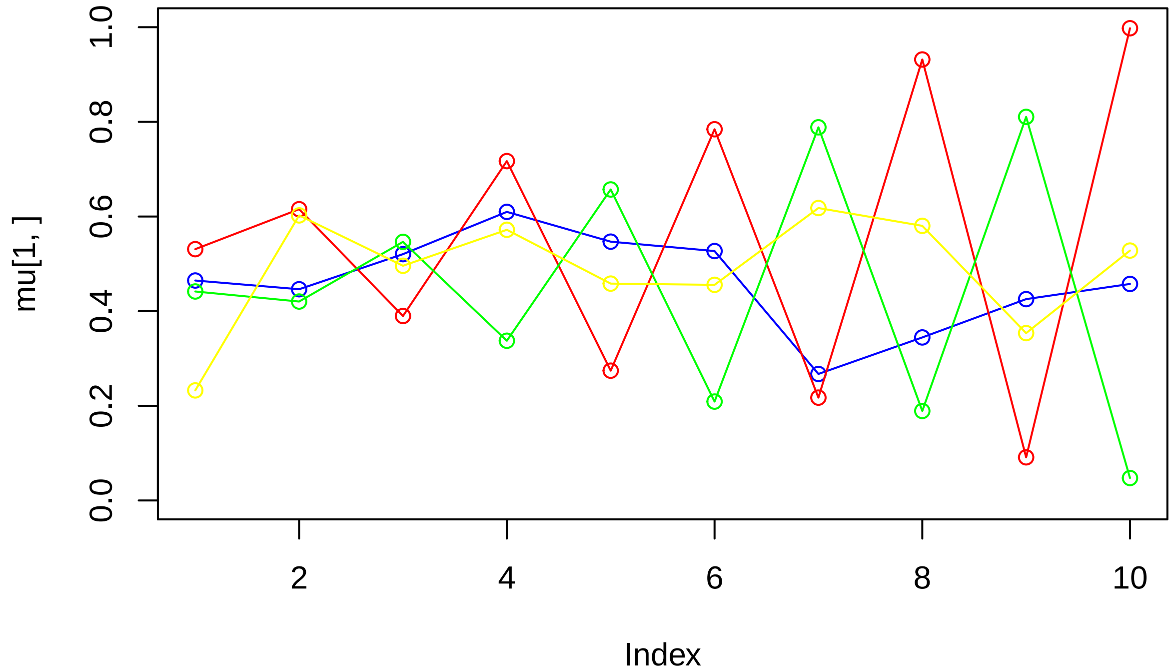

```
## iteration:  49 log likelihood:  -5625.304
```



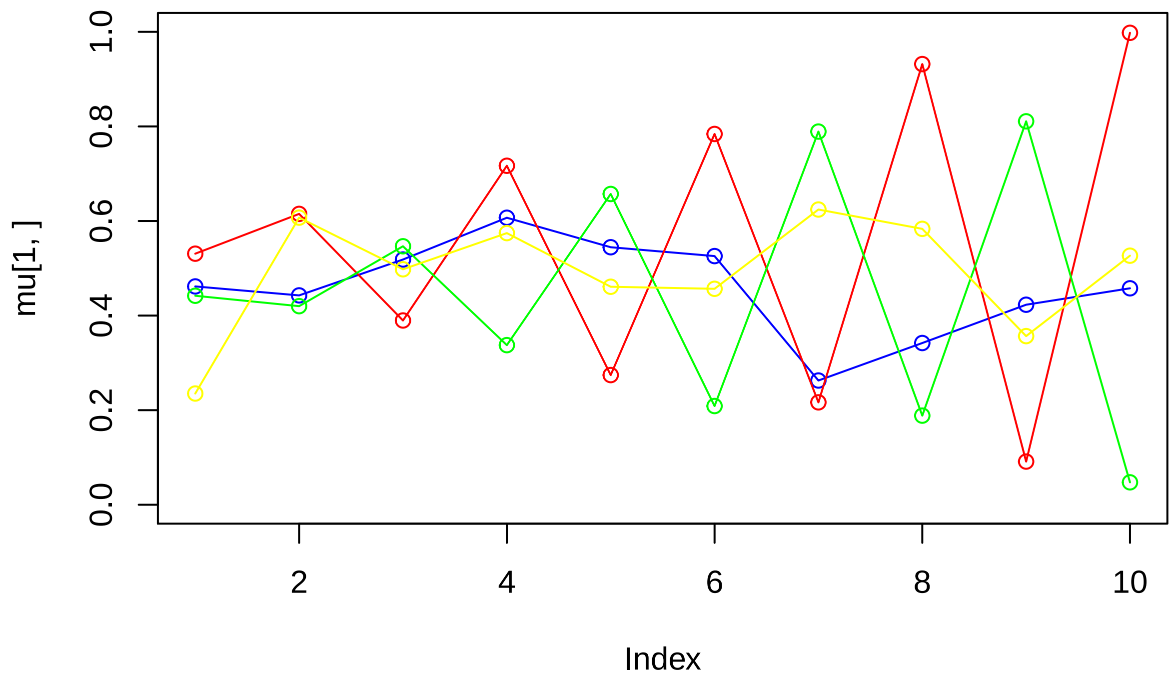

```
## iteration:  50 log likelihood:  -5623.08
```



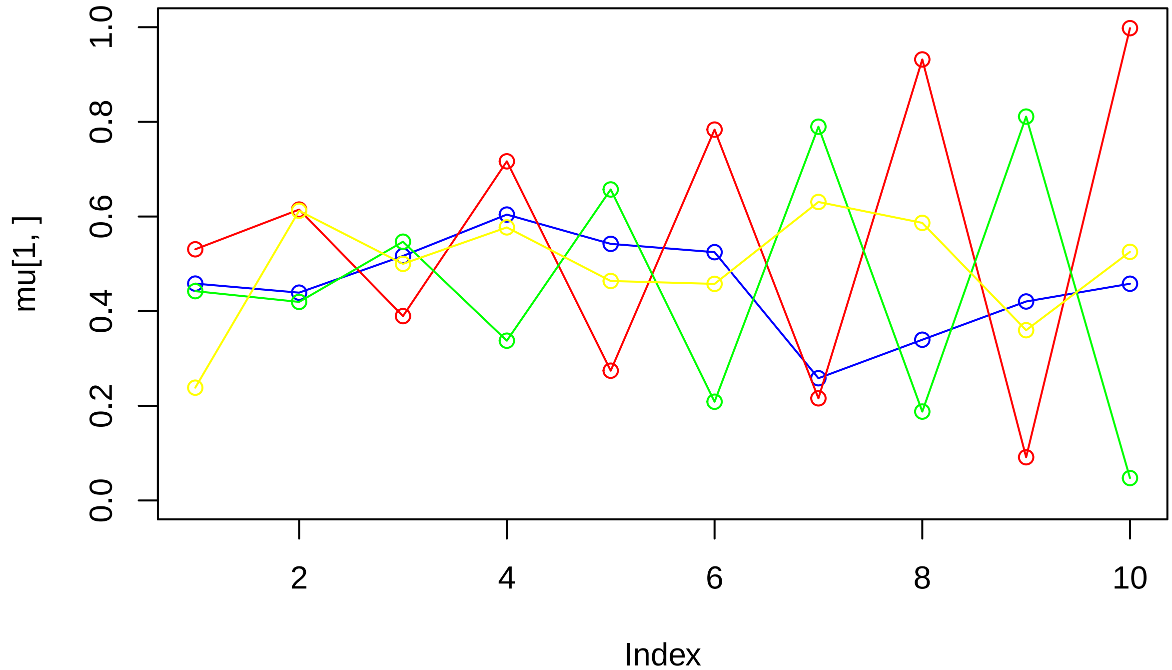

```
## iteration:  51 log likelihood:  -5620.832
```



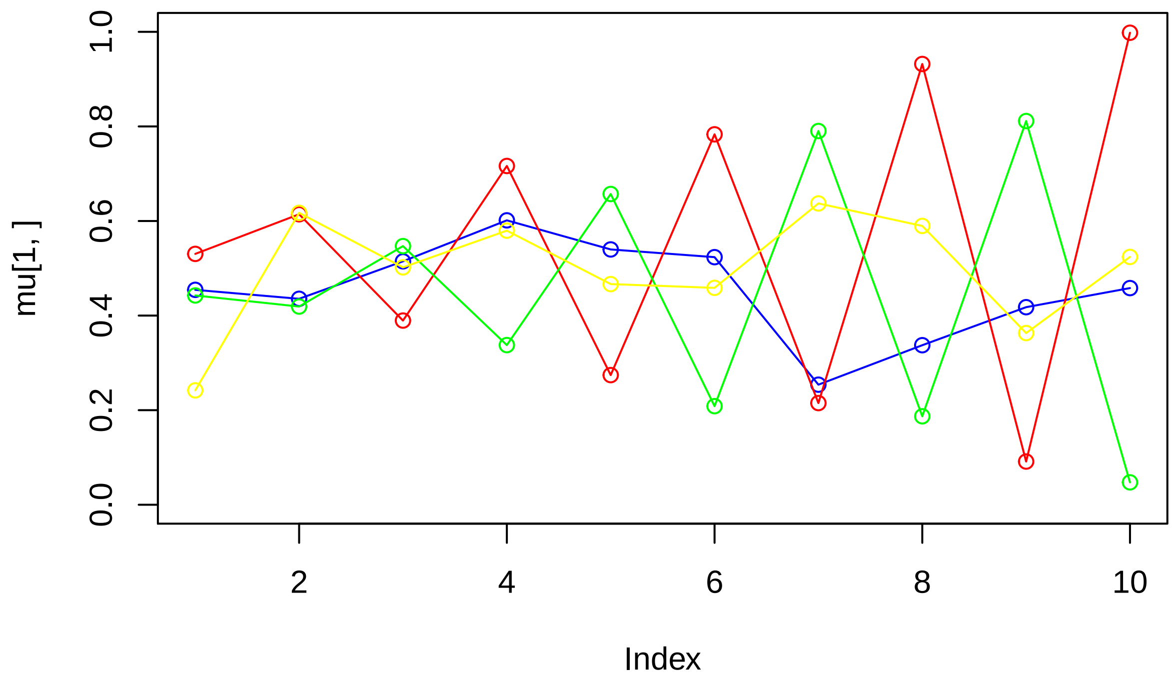

```
## iteration:  52 log likelihood:  -5618.543
```



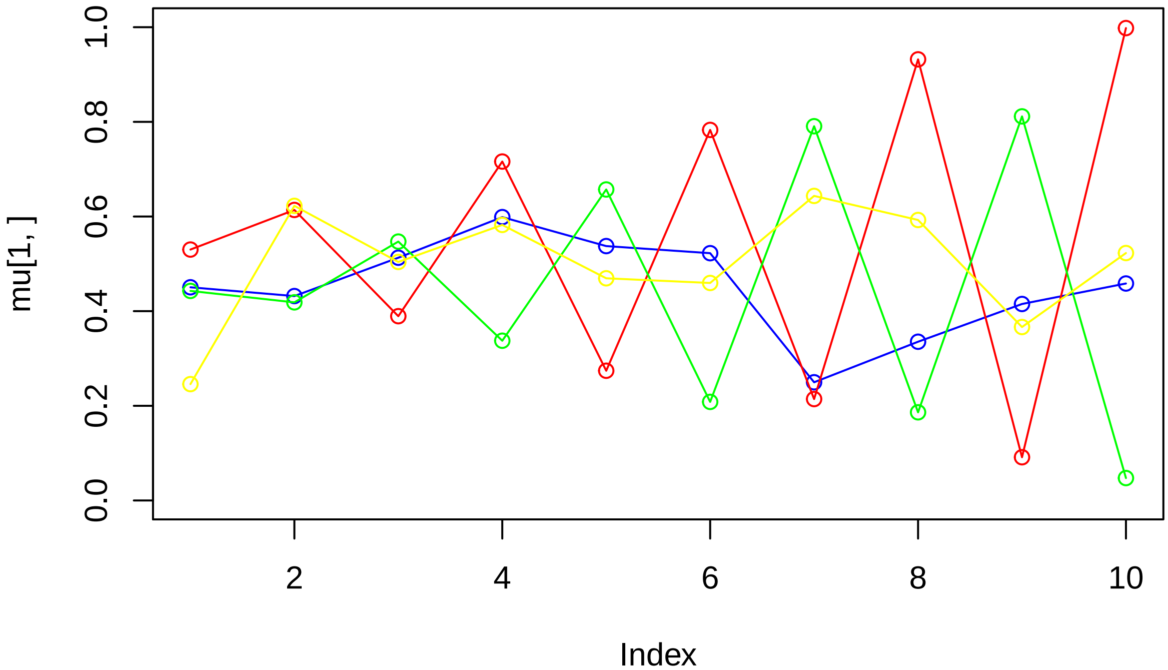

```
## iteration:  53 log likelihood:  -5616.204
```



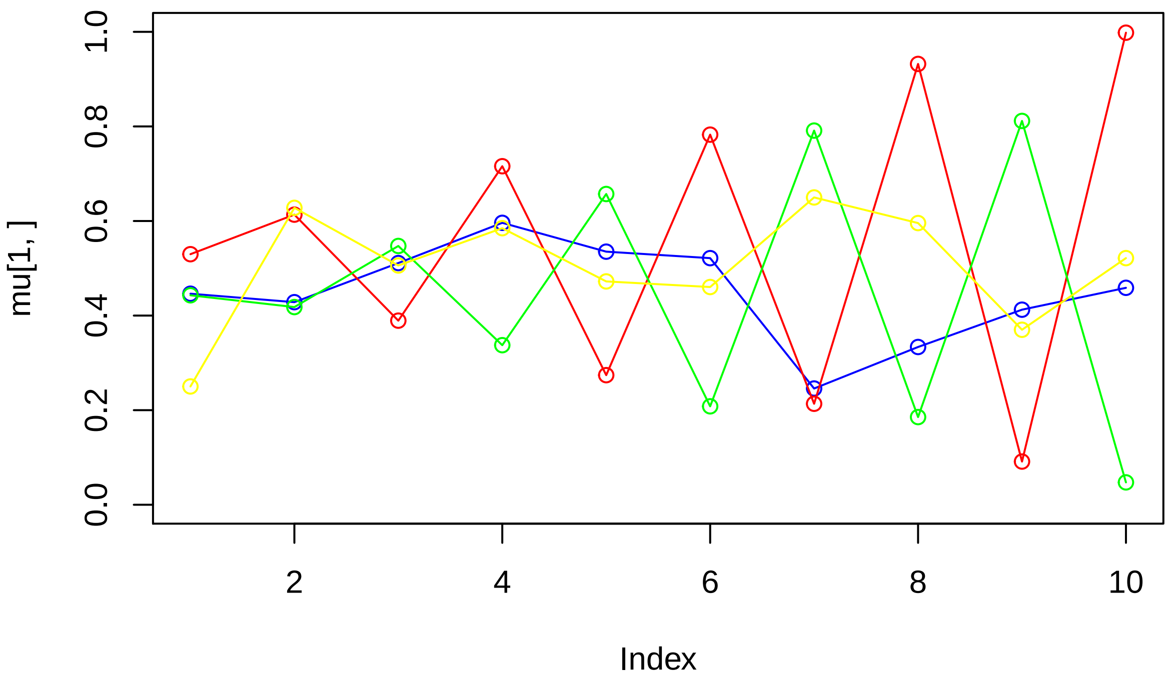

```
## iteration:  54 log likelihood:  -5613.809
```



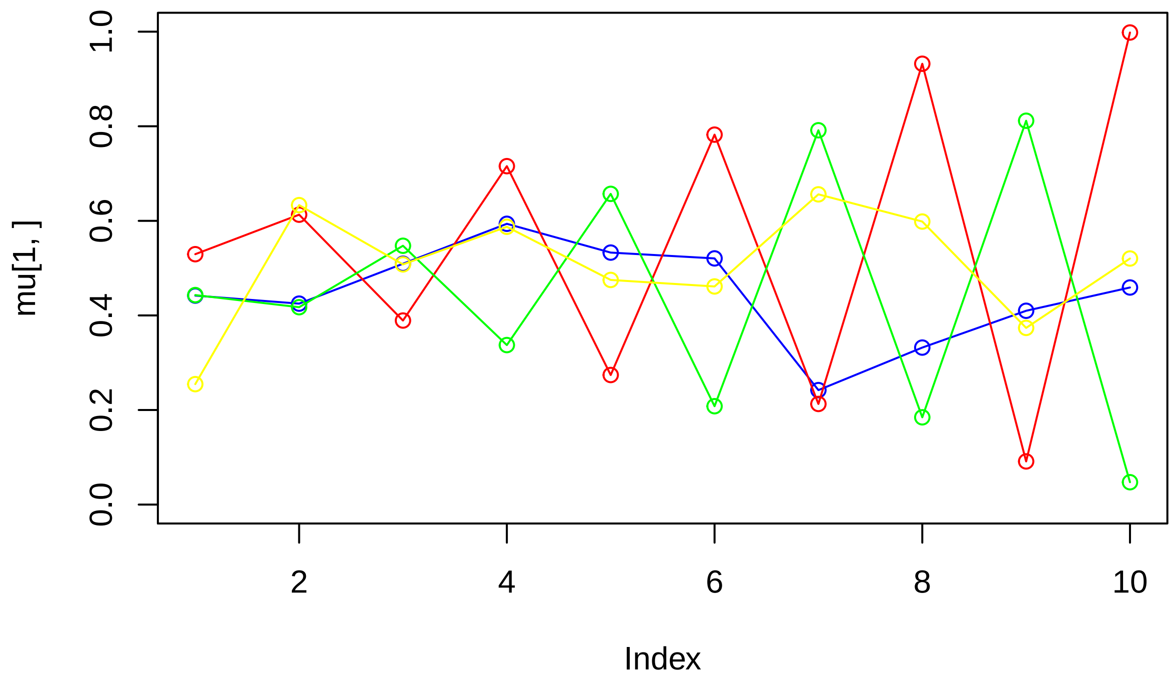

```
## iteration:  55 log likelihood:  -5611.36
```



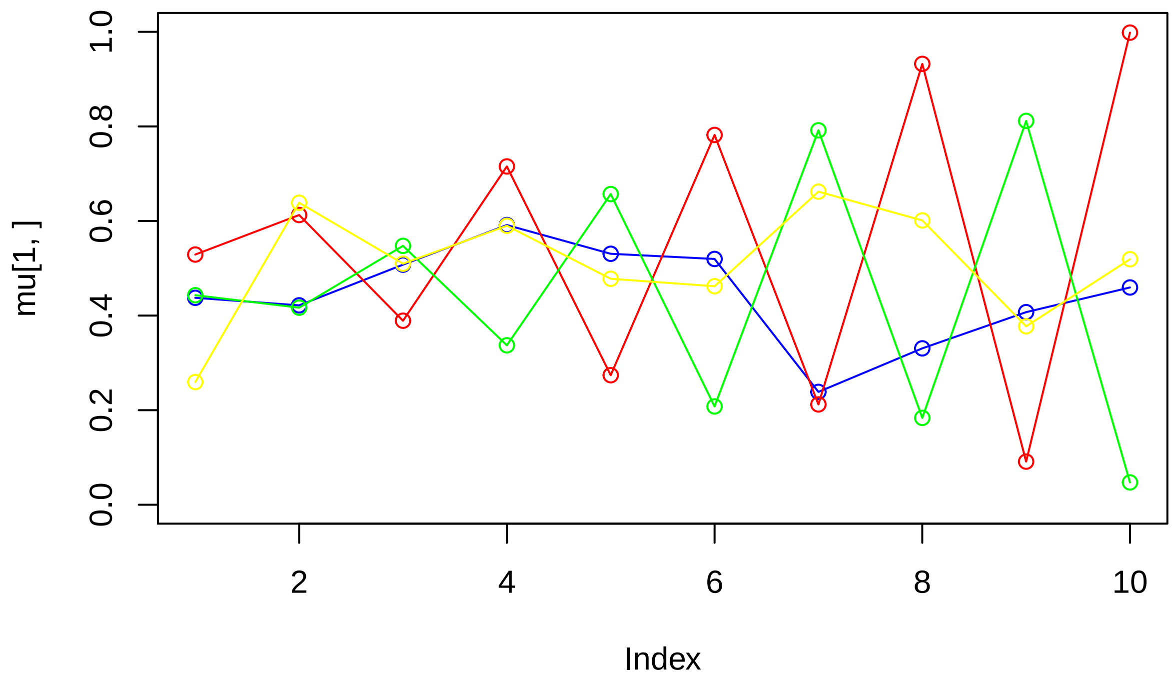

```
## iteration:  56 log likelihood:  -5608.87
```



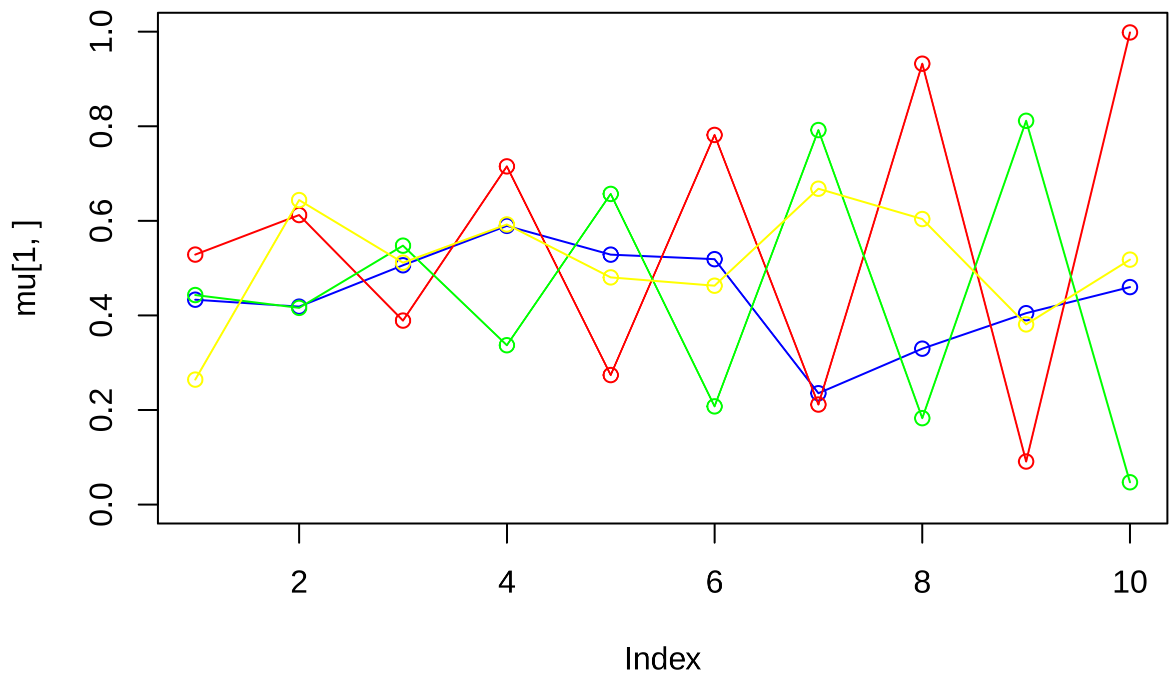

```
## iteration:  57 log likelihood:  -5606.356
```



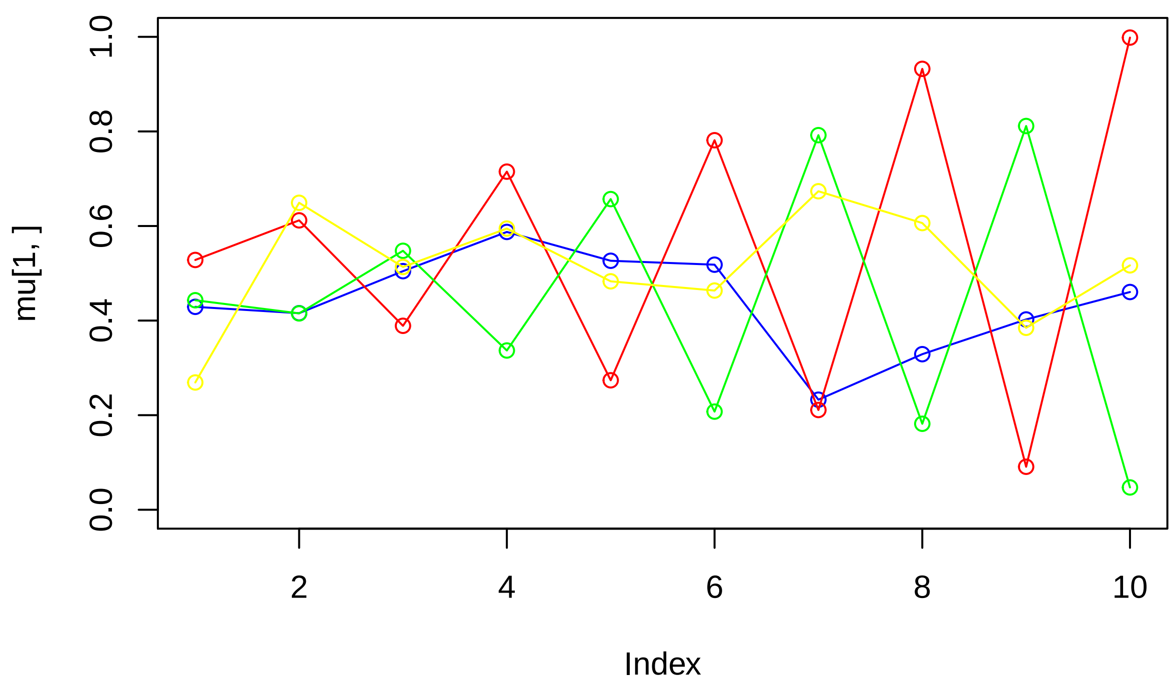

```
## iteration:  58 log likelihood:  -5603.846
```



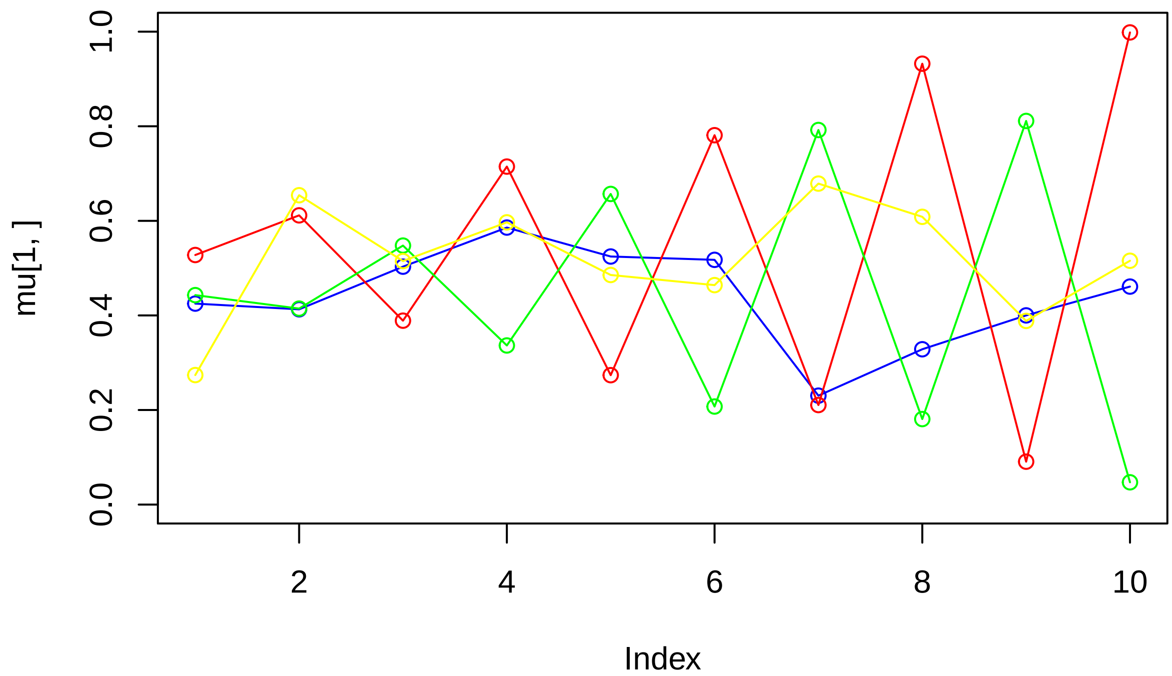

```
## iteration:  59 log likelihood:  -5601.369
```



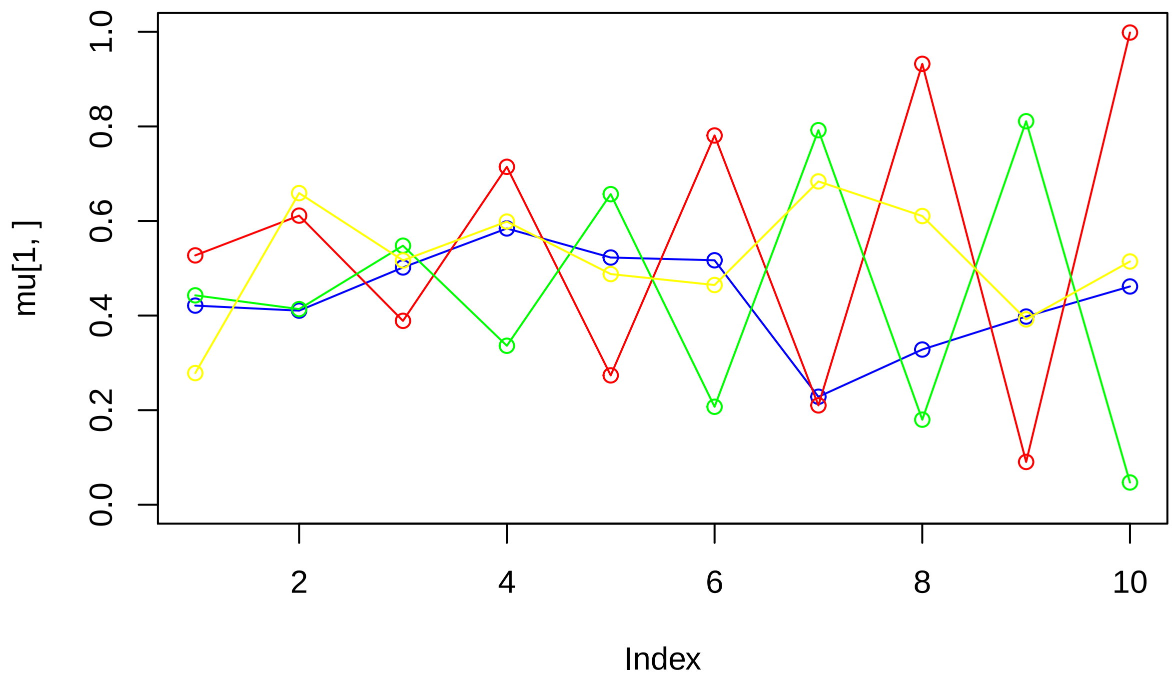

```
## iteration:  60 log likelihood:  -5598.96
```



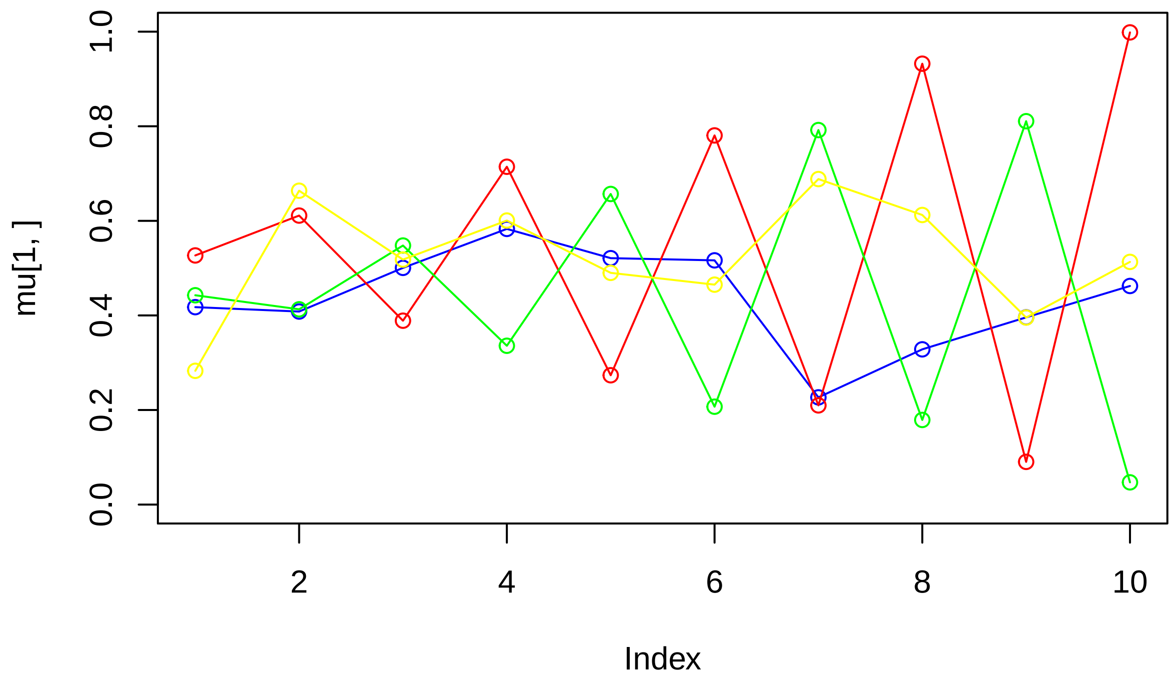

```
## iteration:  61 log likelihood:  -5596.652
```



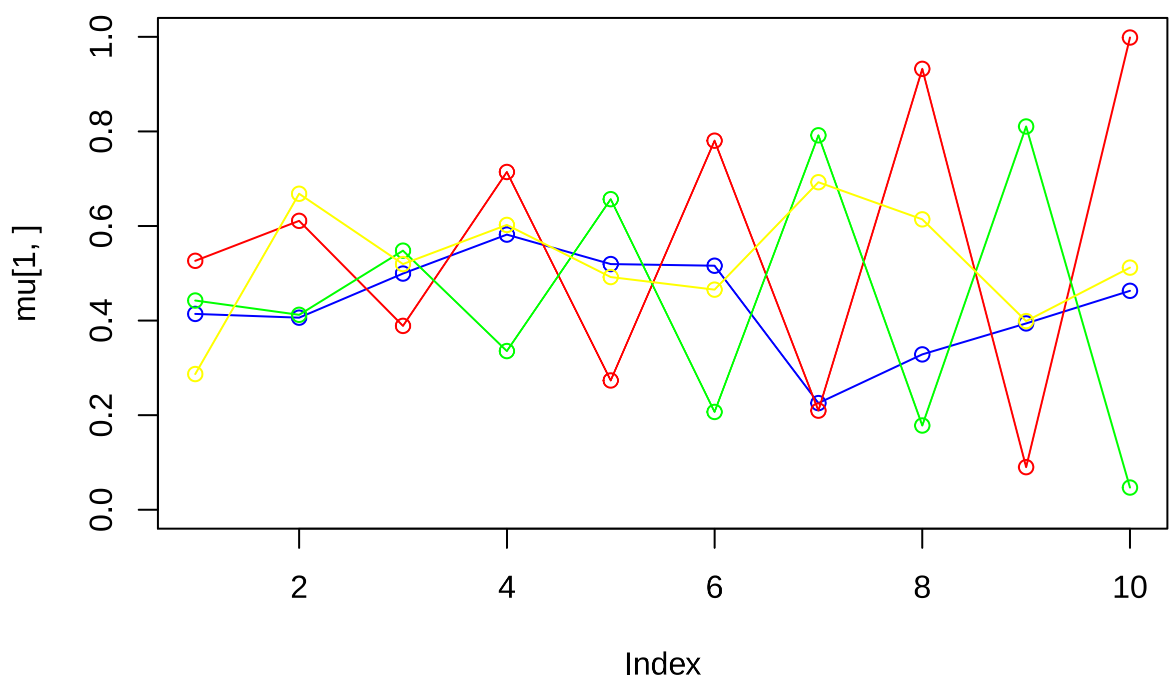

```
## iteration:  62 log likelihood:  -5594.475
```



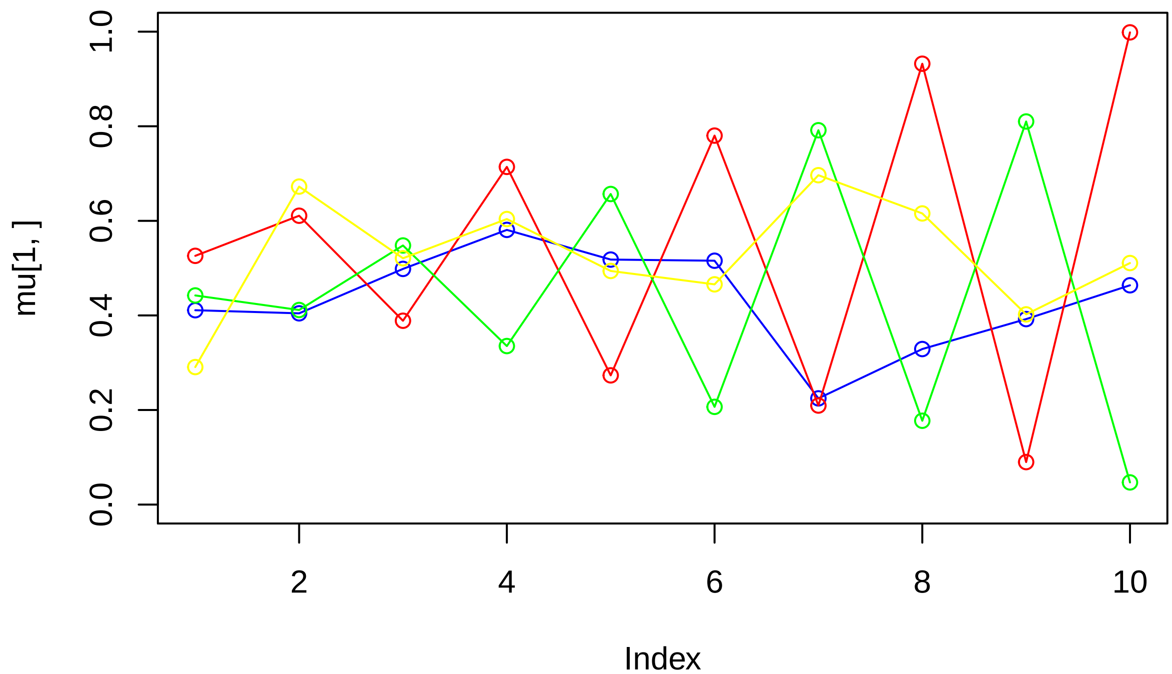

```
## iteration:  63 log likelihood:  -5592.454
```



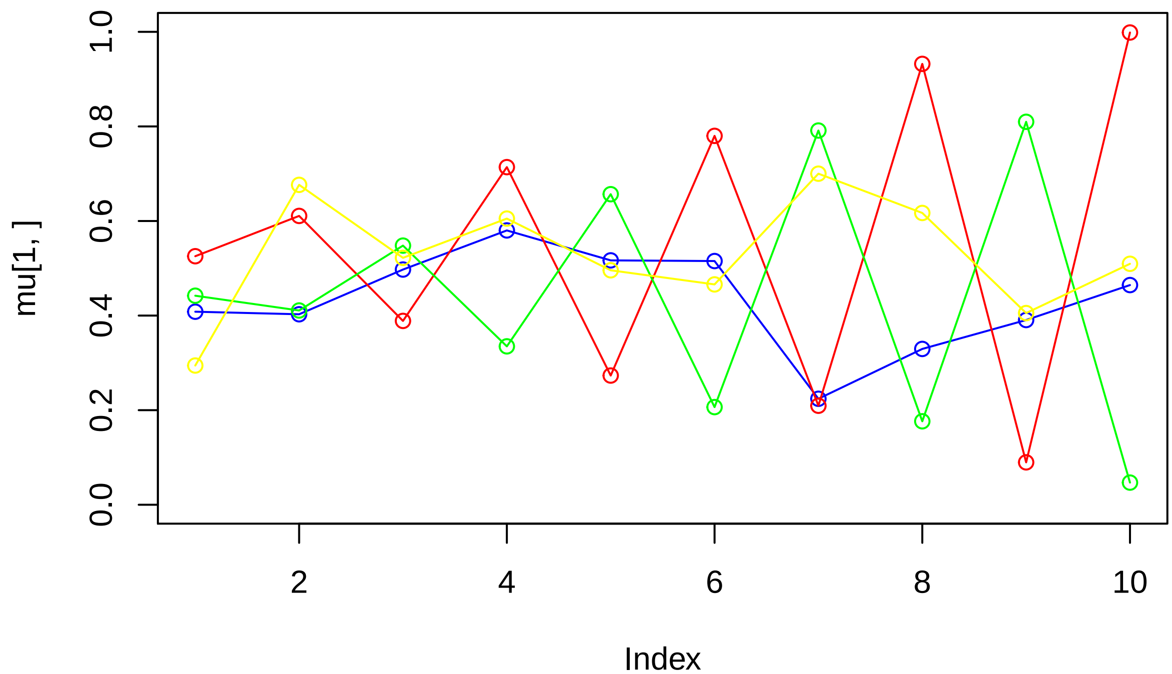

```
## iteration:  64 log likelihood:  -5590.605
```



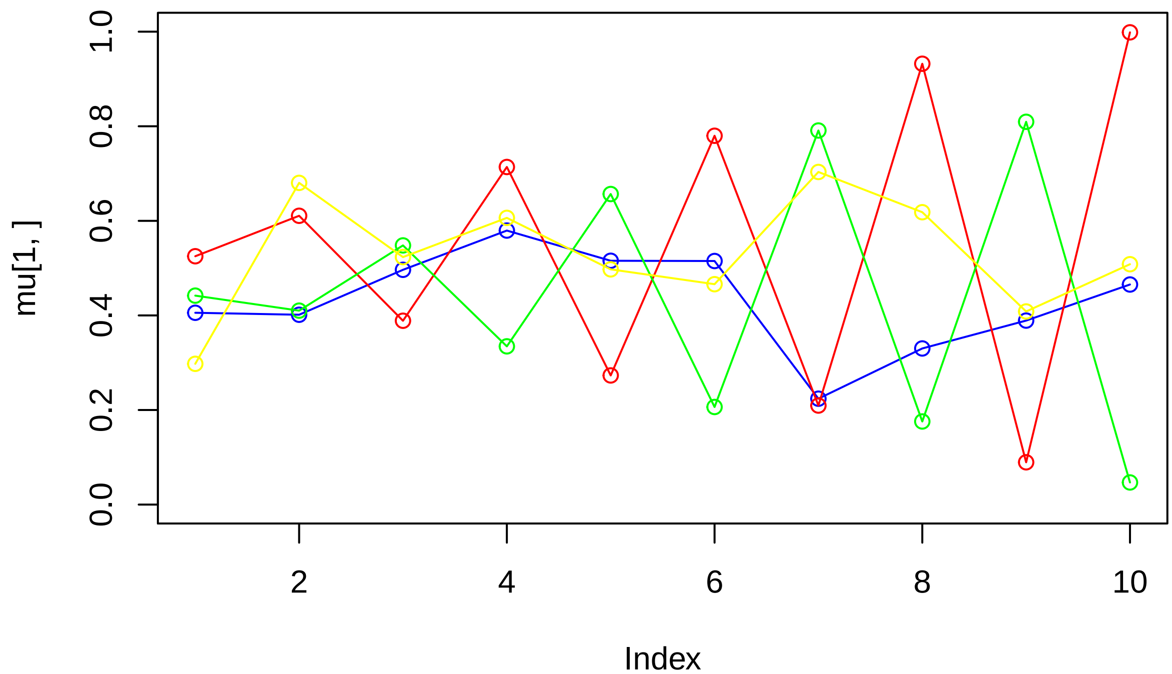

```
## iteration:  65 log likelihood:  -5588.94
```



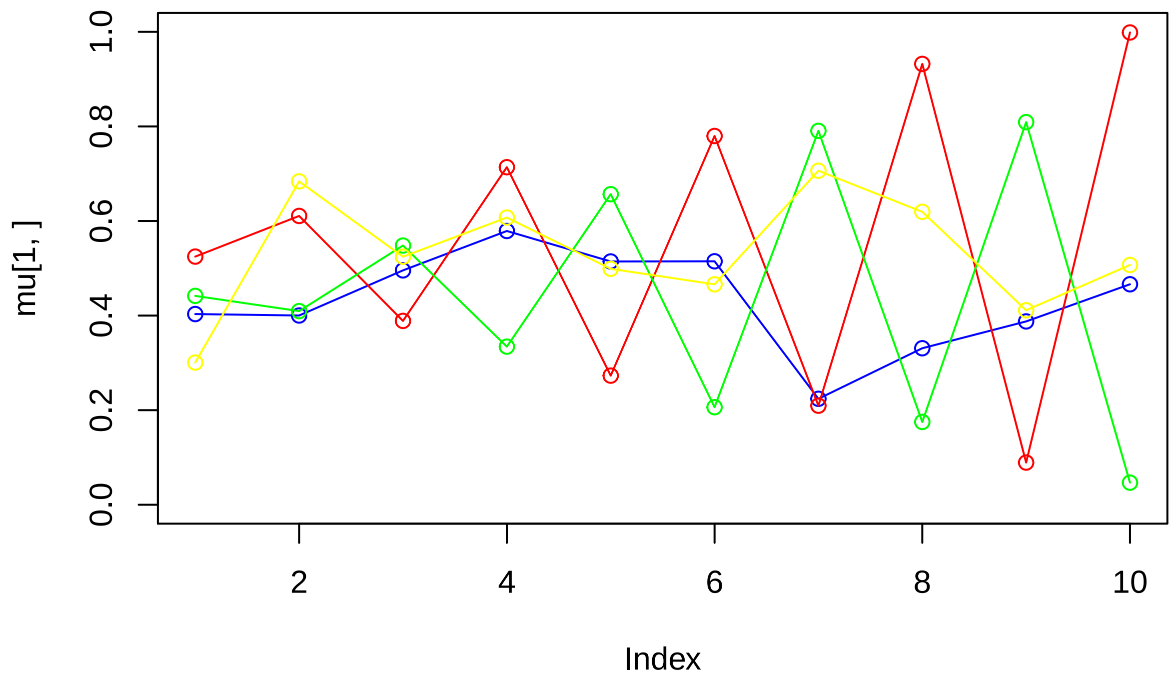

```
## iteration:  66 log likelihood:  -5587.461
```



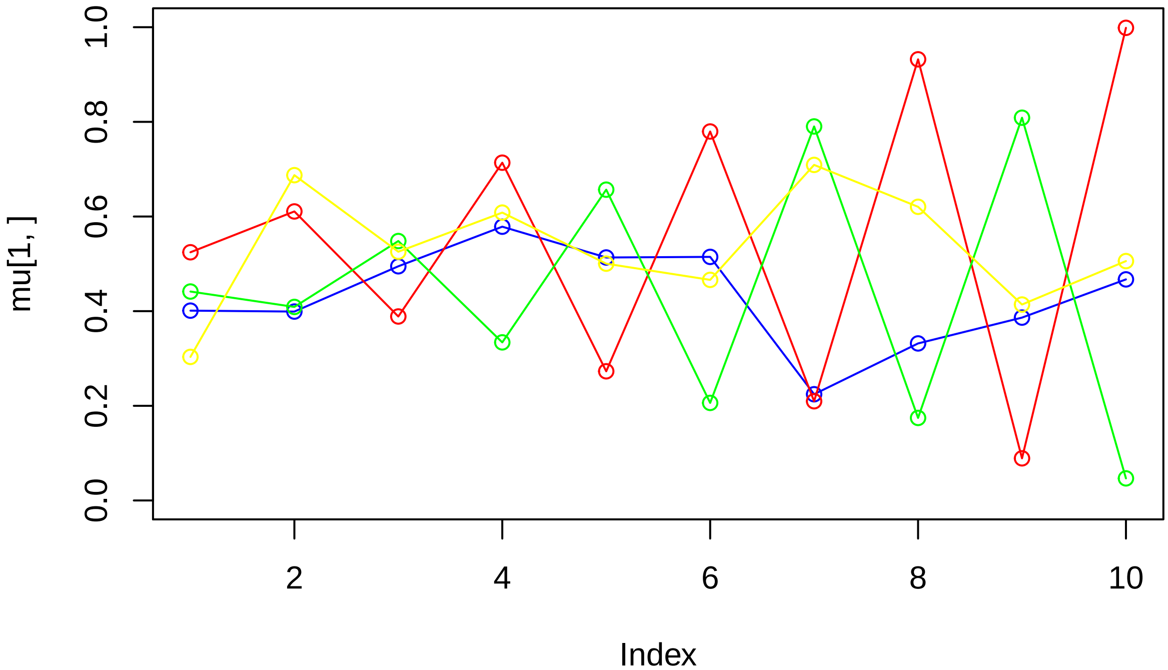

```
## iteration:  67 log likelihood:  -5586.164
```



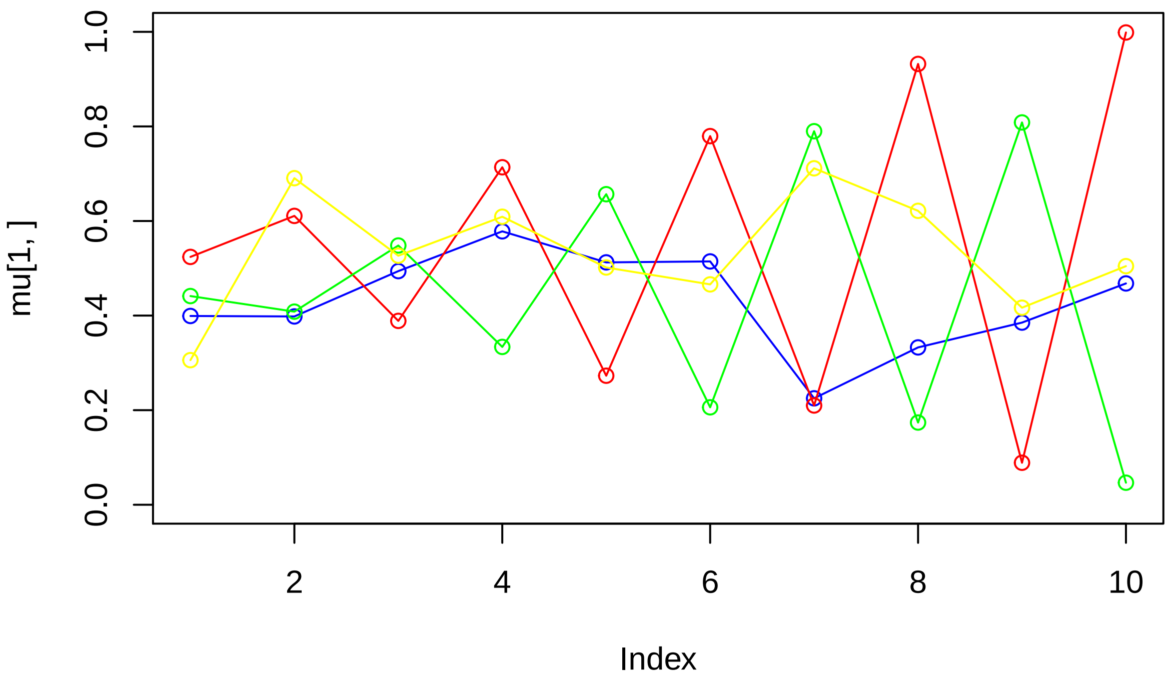

```
## iteration:  68 log likelihood:  -5585.04
```



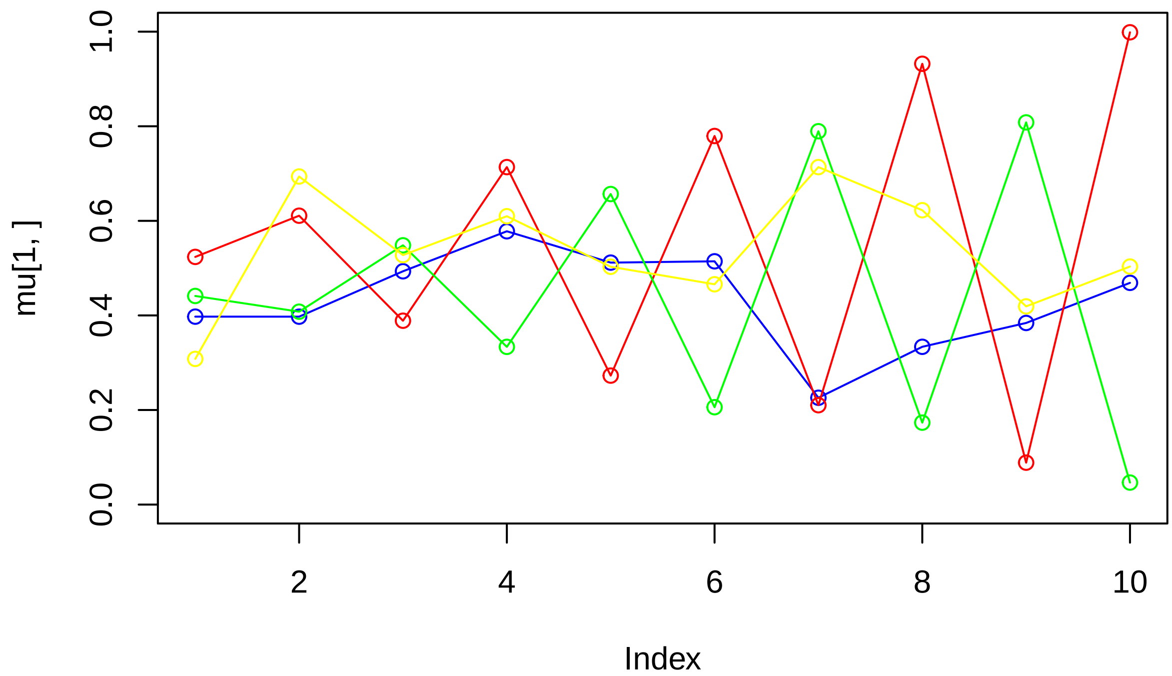

```
## iteration:  69 log likelihood:  -5584.076
```



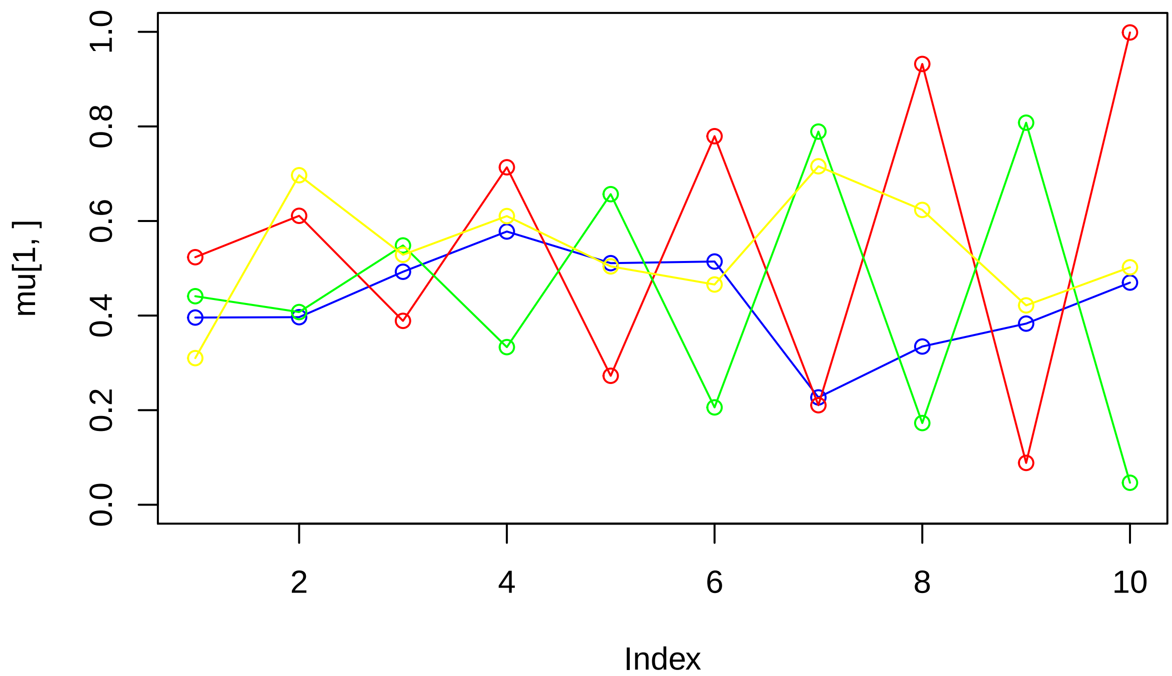

```
## iteration:  70 log likelihood:  -5583.259
```



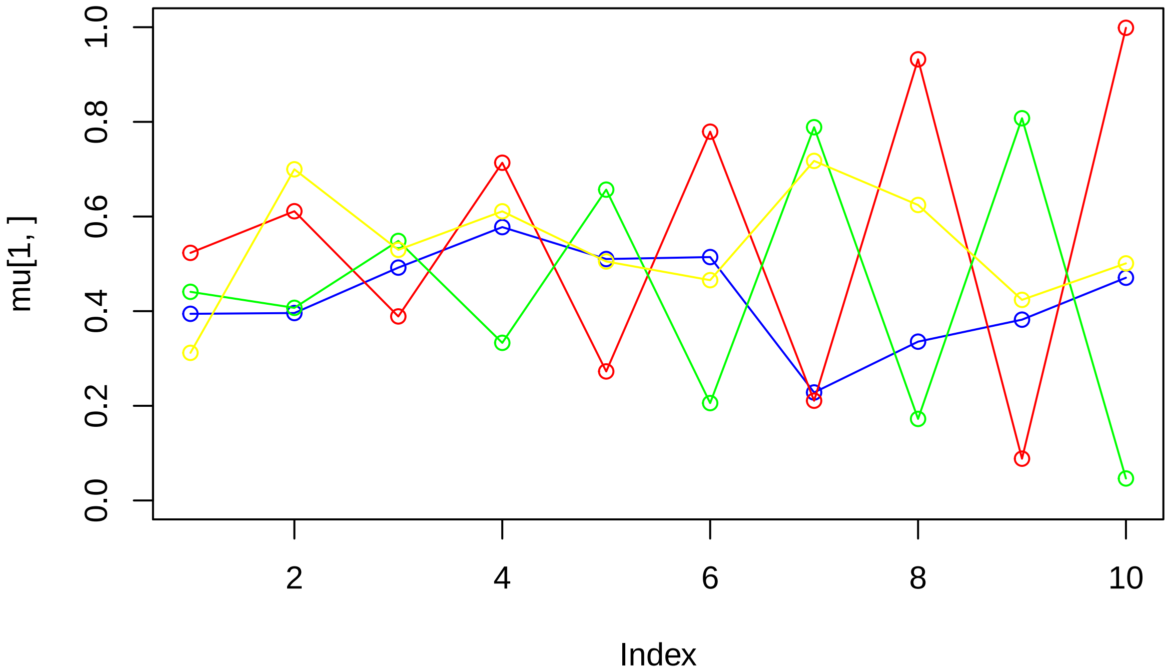

```
## iteration:  71 log likelihood:  -5582.571
```



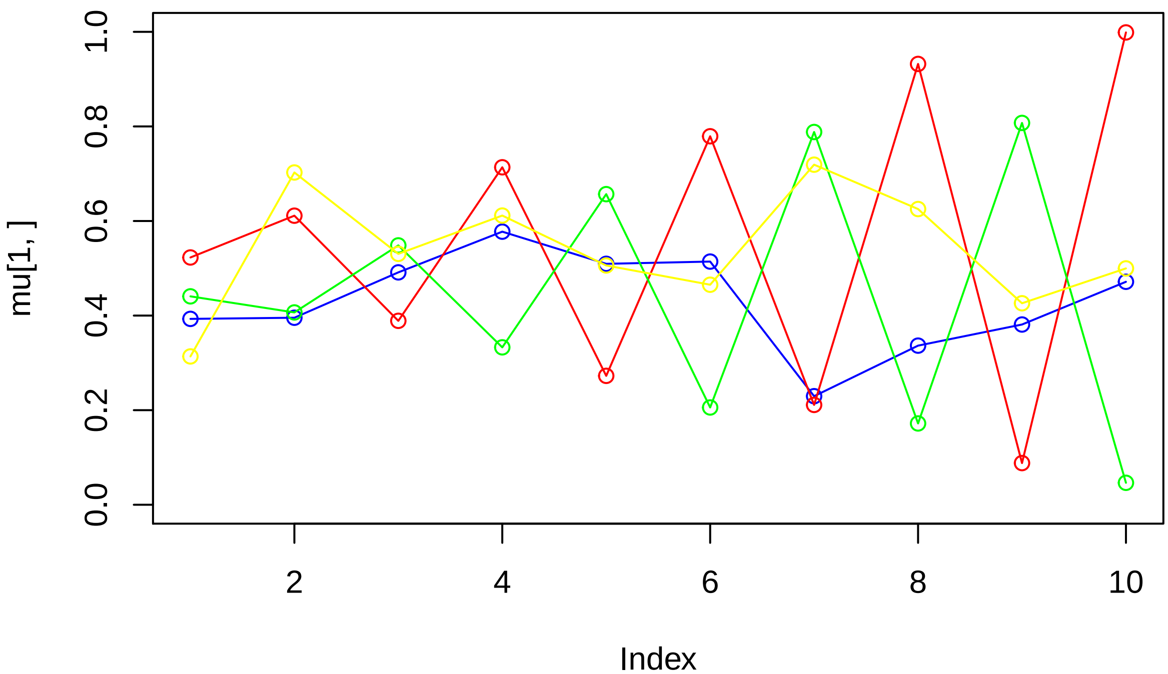

```
## iteration:  72 log likelihood:  -5581.998
```



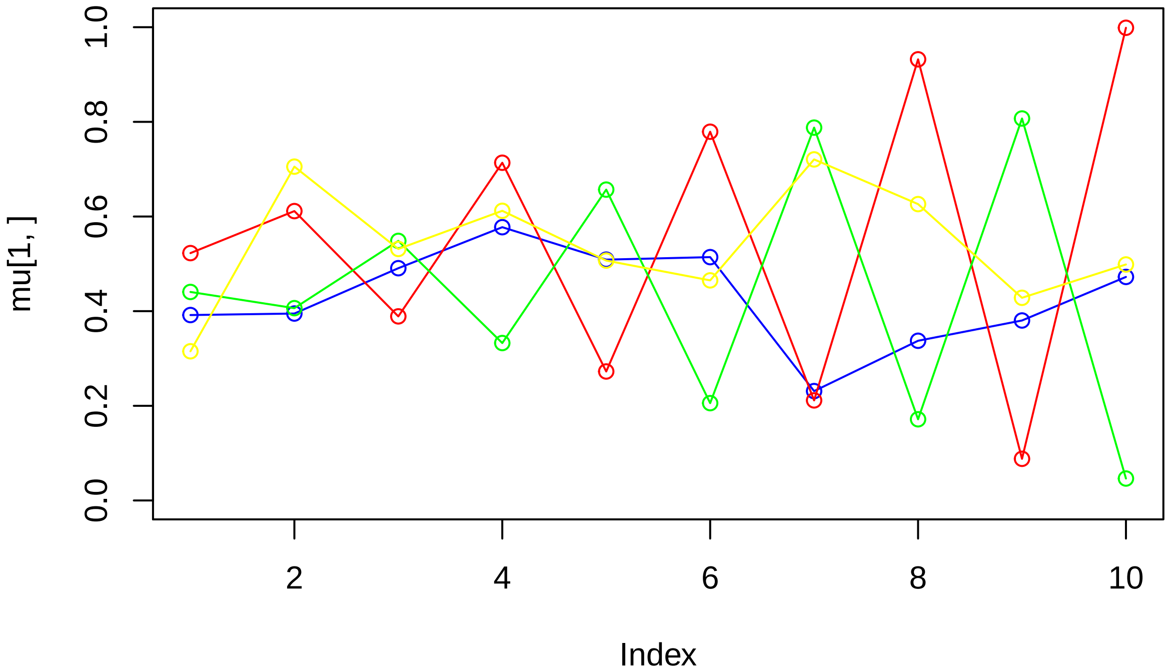

```
## iteration:  73 log likelihood:  -5581.523
```



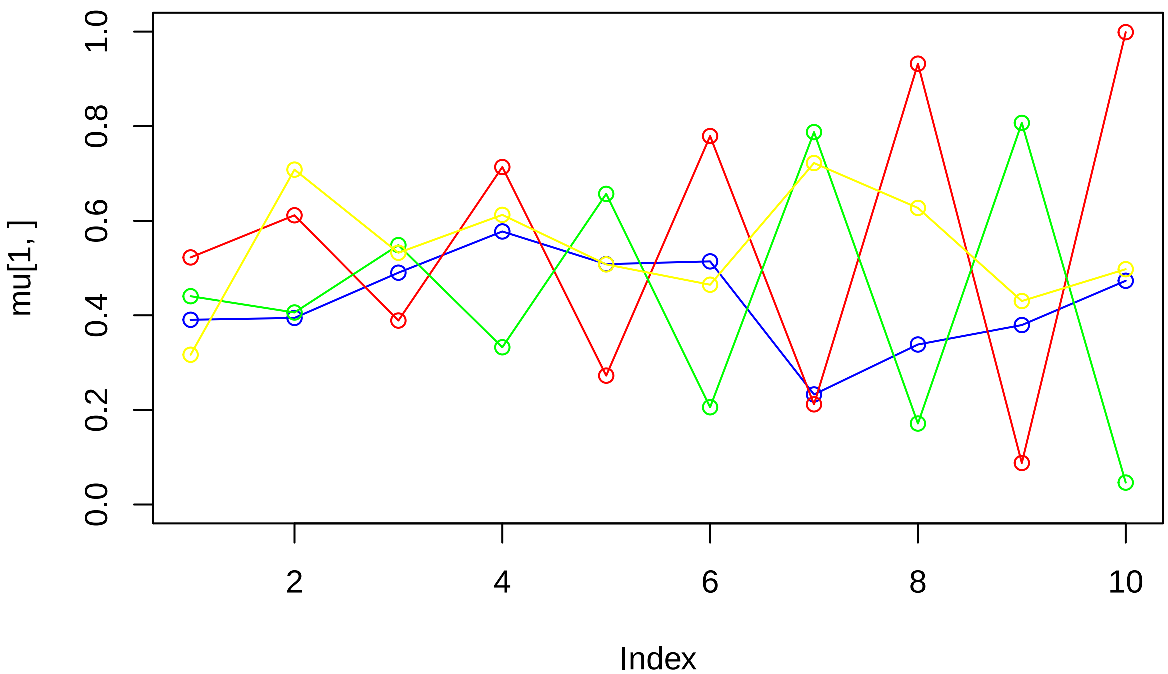

```
## iteration:  74 log likelihood:  -5581.132
```



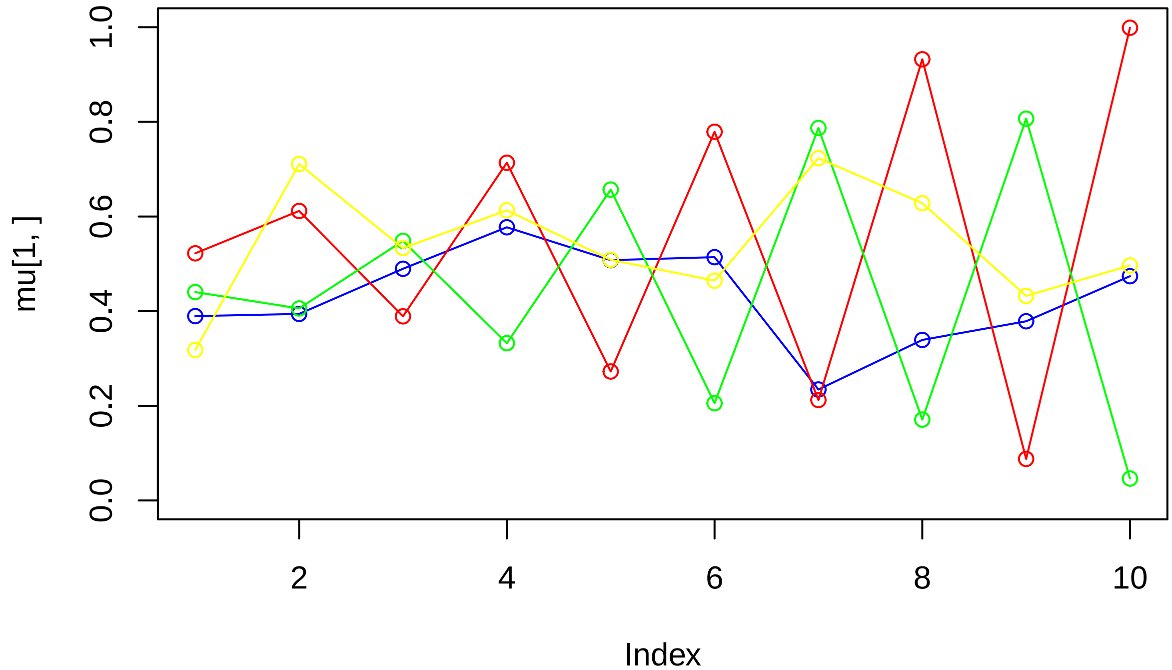

```
## iteration:  75 log likelihood:  -5580.812
```



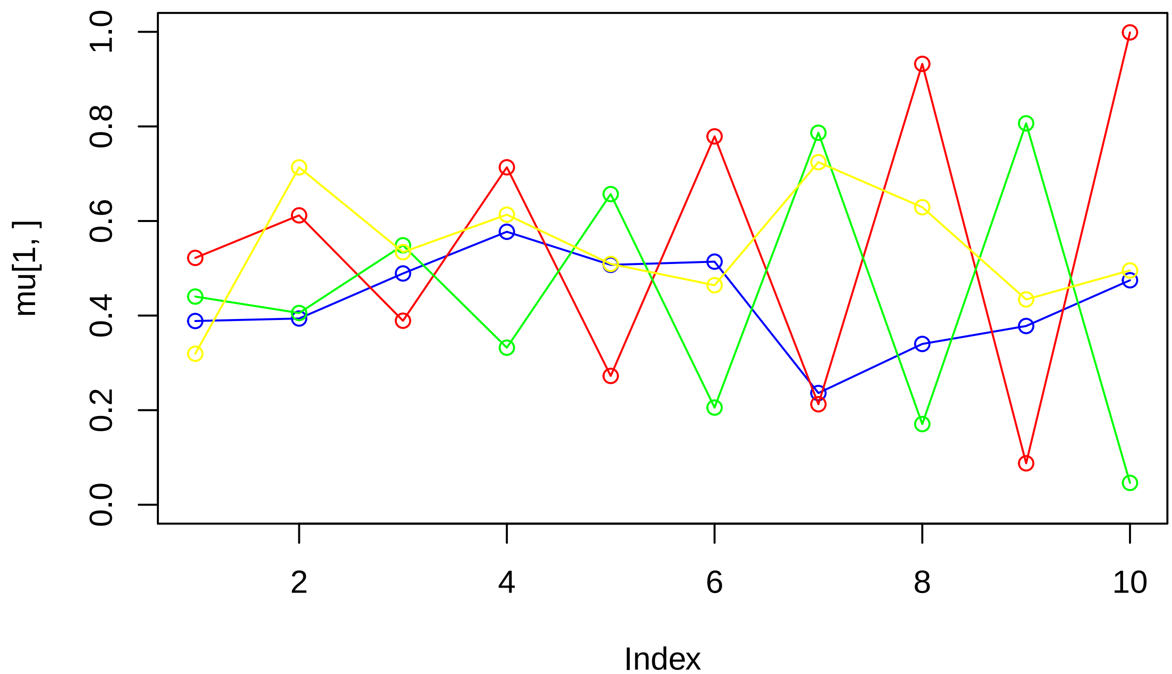

```
## iteration:  76 log likelihood:  -5580.552
```



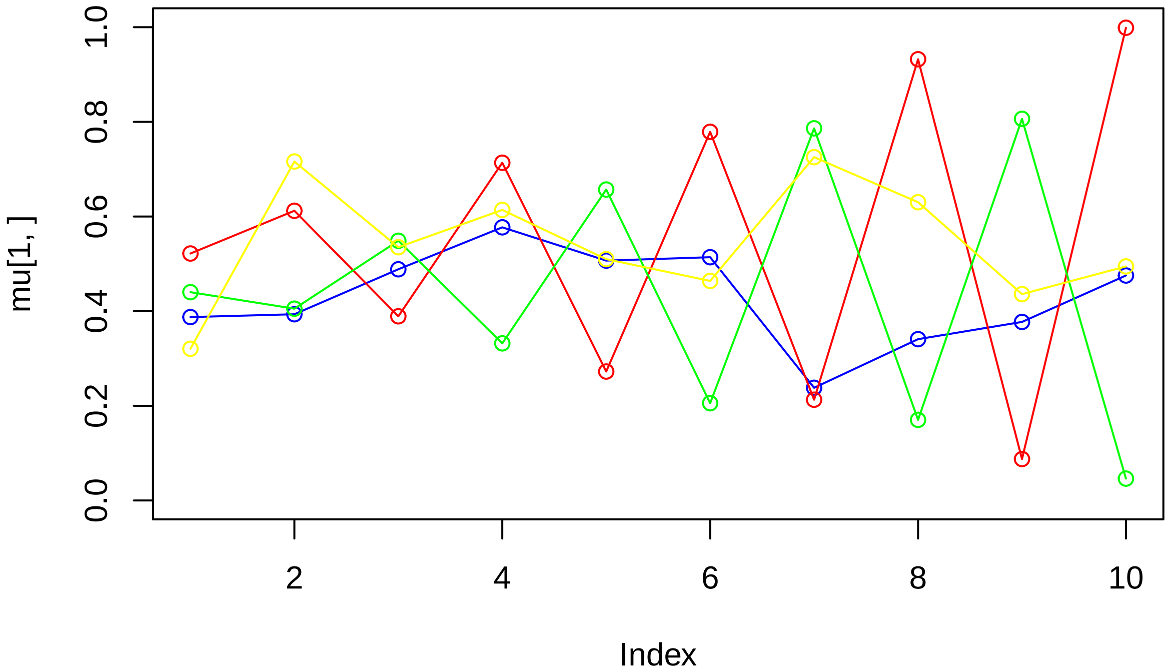

```
## iteration:  77 log likelihood:  -5580.341
```



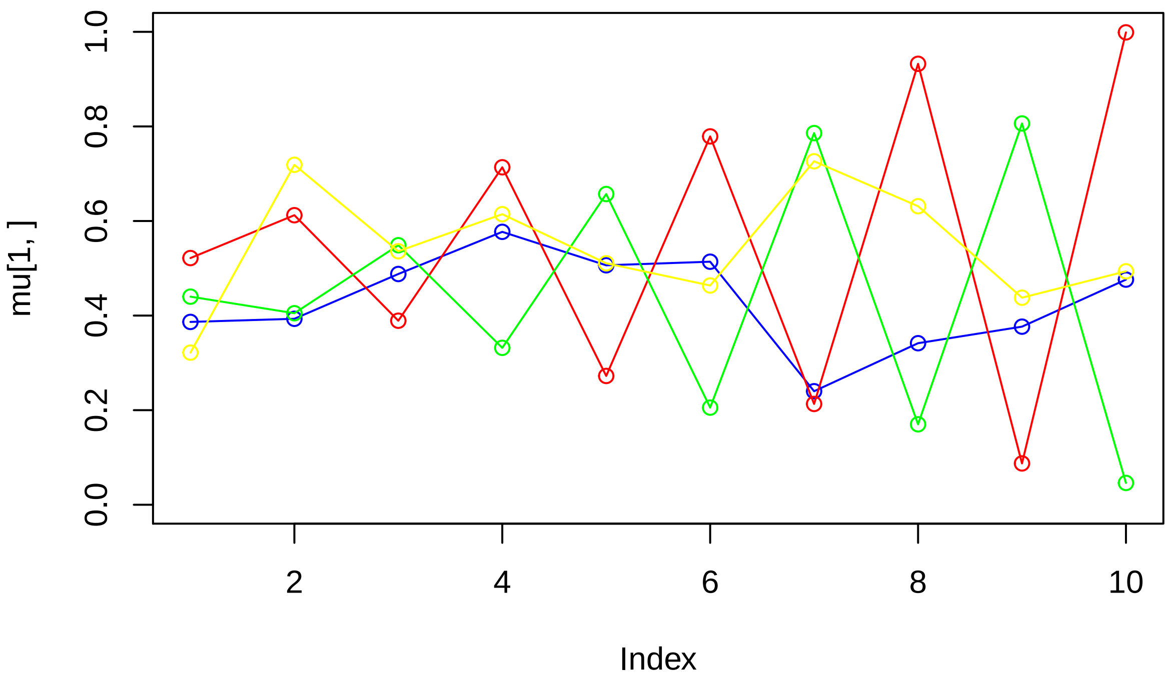

```
## iteration:  78 log likelihood:  -5580.169
```



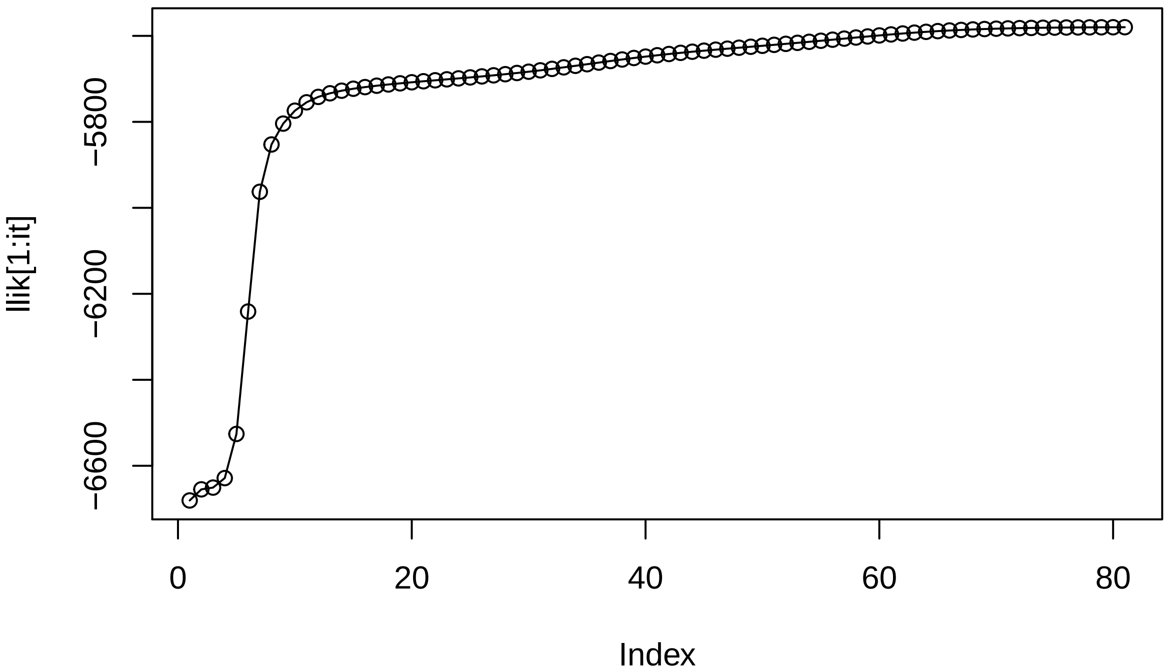

```
## iteration:  79 log likelihood:  -5580.029
```




```
## iteration:  80 log likelihood:  -5579.915
```




```
## iteration:  81 log likelihood:  -5579.821
```




```
## [[1]]
##  [1] 0.26041646 0.25364281 0.29318139 0.19275935 0.38658629 0.52163416
##  [7] 0.44004028 0.32167671 0.39321260 0.61225557 0.40500112 0.71886505
## [13] 0.48795286 0.38916318 0.54876590 0.53607077 0.57710846 0.71360156
## [19] 0.33179850 0.61441593 0.50636776 0.27235697 0.65703184 0.51059837
## [25] 0.51395263 0.77891604 0.20540033 0.46364689 0.24003794 0.21316485
## [31] 0.78592225 0.72631841 0.34159808 0.93237925 0.17009121 0.63127091
## [37] 0.37660293 0.08737204 0.80624121 0.43780078 0.47606925 0.99886761
## [43] 0.04609842 0.49363597
```

Appendix

```r
# Loading packages and importing files ####
library(mboost)
library(randomForest)
library(ggplot2)
sp <- read.csv2("spambase.csv", header = FALSE, sep = ",", stringsAsFactors = FALSE)
num_sp <- data.frame(data.matrix(sp))
num_sp$V58 <- factor(num_sp$V58)
# shuffling data and dividing into train and test ####
n <- dim(num_sp)[1]
ncol <- dim(num_sp)[2]
set.seed(1234567890)
id <- sample(1:n, floor(n*(2/3)))
train <- num_sp[id,]
test <- num_sp[-id,]
# Adaboost
ntree <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
error <- c()
```

```r
for (i in seq(from = 10, to = 100, by = 10)){
bb <- blackboost(V58 ~., data = train, control = boost_control(mstop = i), family = AdaExp())
bb_predict <- predict(bb, newdata = test, type = c("class"))
confusion_bb <- table(test$V58, bb_predict)
miss_class_bb <- (confusion_bb[1,2] + confusion_bb[2,1])/nrow(test)
error[(i/10)] <- miss_class_bb
}

error_df <- data.frame(cbind(ntree, error))
# Random forest ####
ntree_rf <- c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
error_rf <- c()

for (i in seq(from = 10, to = 100, by = 10)){
rf <- randomForest(V58 ~., data = train, ntree= 10)
rf_predict <- predict(rf, newdata = test, type = c("class"))
confusion_rf <- table(test$V58, rf_predict)
miss_class_rf <- (confusion_rf[1,2] + confusion_rf[2,1])/nrow(test)
error_rf[i/10] <- miss_class_rf
}

error_df_rf <- data.frame(cbind(ntree_rf, error_rf))

df <- cbind(error_df, error_df_rf)
df <- df[, -3]

plot_final <- ggplot(df, aes(ntree)) +
  geom_line(aes(y=error, color = "Adaboost")) +
  geom_line(aes(y=error_rf, color = "Random forest"))

plot_final <- plot_final + ggtitle("Error rate vs number of trees")
plot_final

my_own_em <- 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")
```

```r
}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])
  }
}

 # 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){
  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}
}
```

```r
  #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
output2 <- mu
output3 <- plot(llik[1:it], type="o")
z
result <- list(c(output1, output2, output3))
return(result)
}
my_own_em(2)
my_own_em(3)
my_own_em(4)
```