Lab3

Elon Brange, Ludwig Thaung

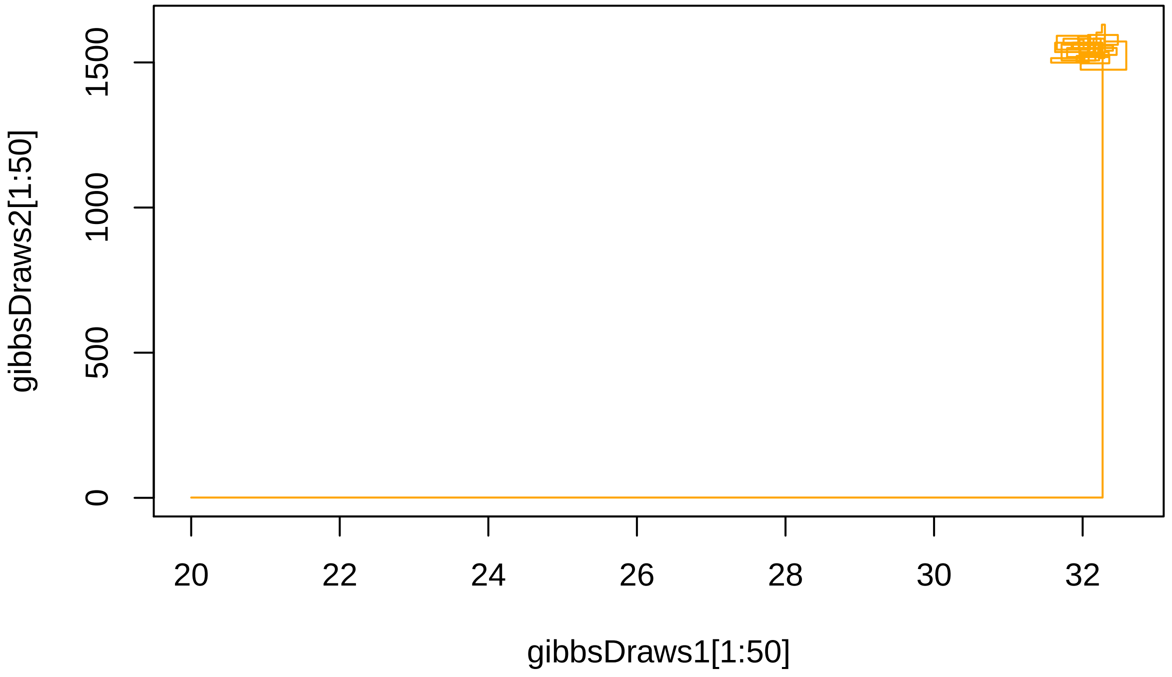
5/19/2019

Task 1

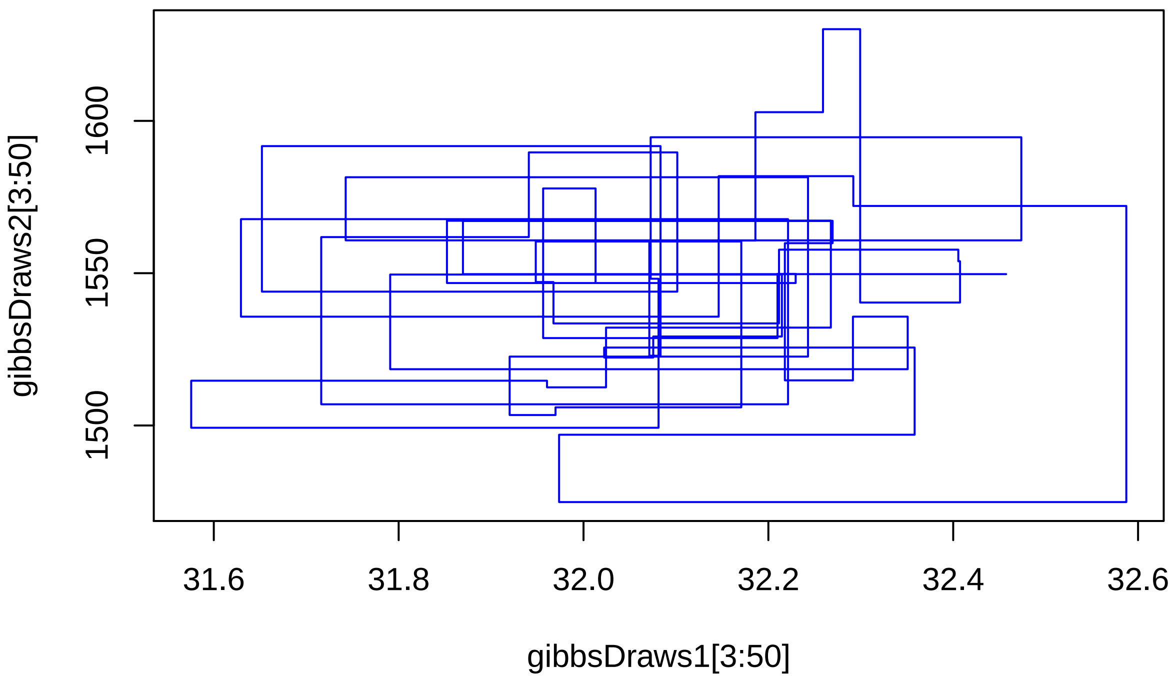
(a)

```r
rainfallData<-read.table("rainfall.dat",header=TRUE)
plotColors = list('red', 'green', 'blue', 'yellow', 'black', 'orange')
# Setup
mu1 <- 1
mu2 <- -1
rho <- 0.9
mu <- c(mu1,mu2)
Sigma = matrix(c(1,rho,rho,1),2,2)
nDraws <- 600 # Number of draws

v0 = 2
mu0 = 20
sigma0 = 1
tau0 = 20

currentMu = mu0
currentSigma = sigma0
currentTau = tau0
currentV = v0

n = nrow(rainfallData)
averageRain = sum(rainfallData)/nrow(rainfallData)
sigma = var(rainfallData-averageRain)[1]
gibbsDraws <- matrix(0,nDraws,2)
for (i in 1:nDraws) {
  currentTau = 1/((n/currentSigma) + 1/tau0)
  w = (n/currentSigma)/(n/currentSigma + 1/tau0)
  currentMu = w*averageRain + (1 - w)*mu0
  currentMu <- rnorm(1, currentMu, currentTau)
  gibbsDraws[i,1] <- currentMu

  currentV = n + v0
  currentSigma = (v0*sigma0 + sum((rainfallData - currentMu)^2)/(n + v0))
  currentSigma <- rinvchisq(n = 1, df = currentV, scale = currentSigma)
  gibbsDraws[i, 2] <- currentSigma
}
averages <- matrix(0, nDraws, 2)
for (i in (1:((nDraws-500)/10))) {
  averages[i, 1] = sum(gibbsDraws[(i*10):(i*10+500), 1])/500
  averages[i, 2] = sum(gibbsDraws[(i*10):(i*10+500), 2])/500
}
#plot(averages[1:(nDraws-500)/10, 1], type='l')
```

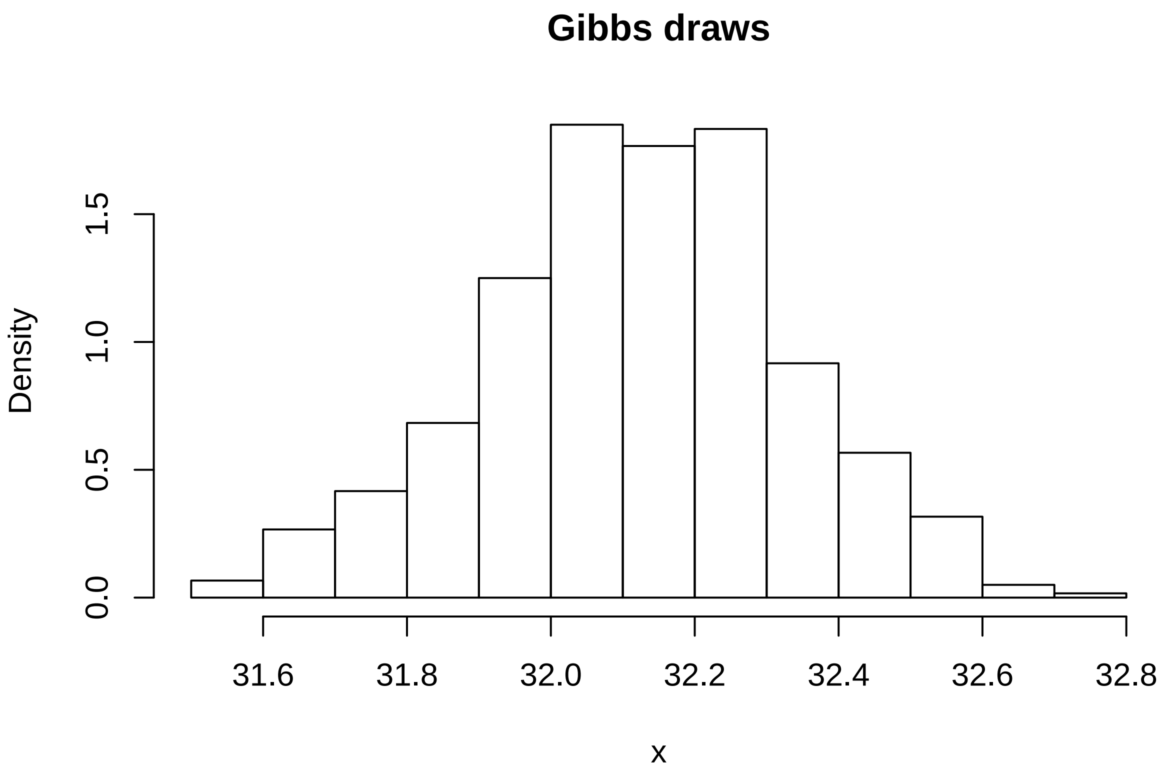
```r
#lines(1:nDraws, matrix(averageRain, (nDraws-500)/10, 1))

#plot(averages[1:(nDraws-500)/10, 2], type='l')
#lines(1:nDraws, matrix(sigma, (nDraws-500)/10, 1))

#plot(1:nDraws, gibbsDraws[,1], type = "l", col = 'red', ylab='y',
#     lwd = 1, axes=FALSE, xlab = 'MCMC iteration', xlim = c(0,nDraws), ylim = c(0, 10000),
#     main = 'Raw - Gibbs')
#axis(side = 1, at = seq(0, nDraws, by = 250))
#axis(side = 2, at = seq(0, 1000, by = 0.5))

gibbsDraws1 = c(mu0, gibbsDraws[,1])
gibbsDraws2 = c(sigma0, gibbsDraws[,2])
plot(gibbsDraws1[1:50],gibbsDraws2[1:50], type ='s', col ='orange')
```



```r
plot(gibbsDraws1[3:50],gibbsDraws2[3:50], type ='s', col ='blue')
```

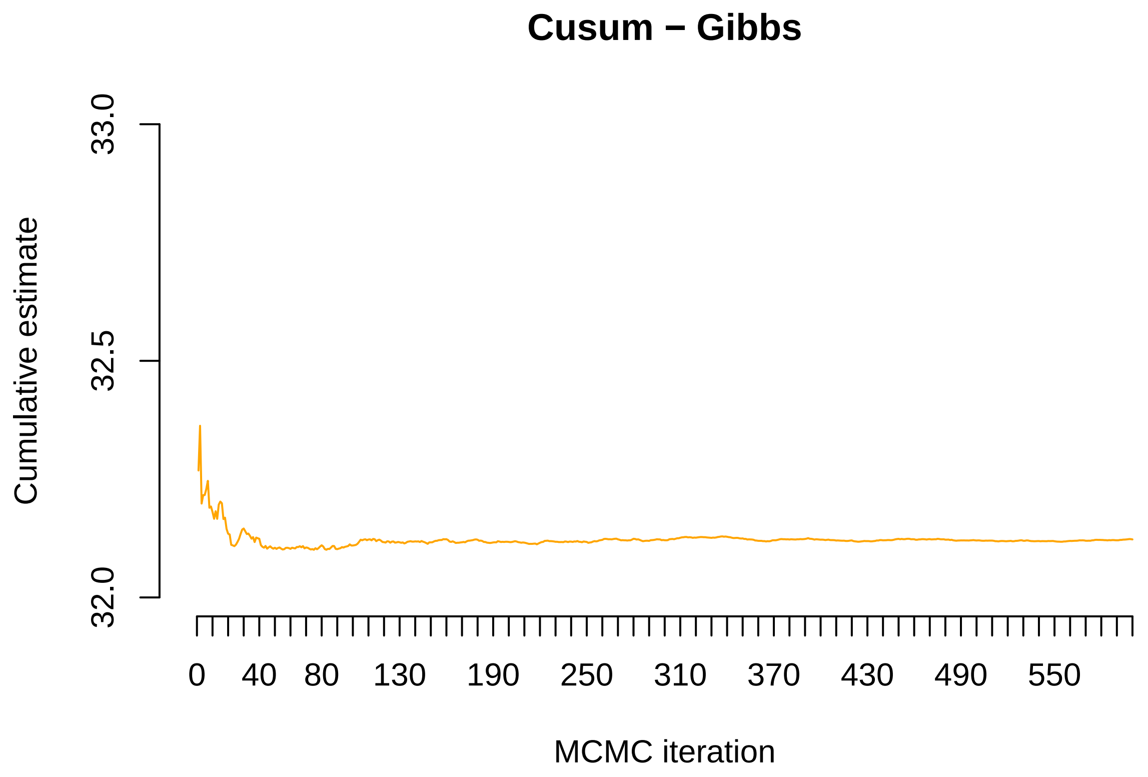


```
hist(gibbsDraws[,1], freq = FALSE, main='Gibbs draws', xlab='x')
#hist(gibbsDraws[,1], freq = FALSE, main='Gibbs draws', ylim = c(0,0.5), xlab='x')

lines(seq(-2,4,by=0.01),dnorm(seq(-2,4,by=0.01), mean = 1), col = 'orange',
      lwd = 1)
```

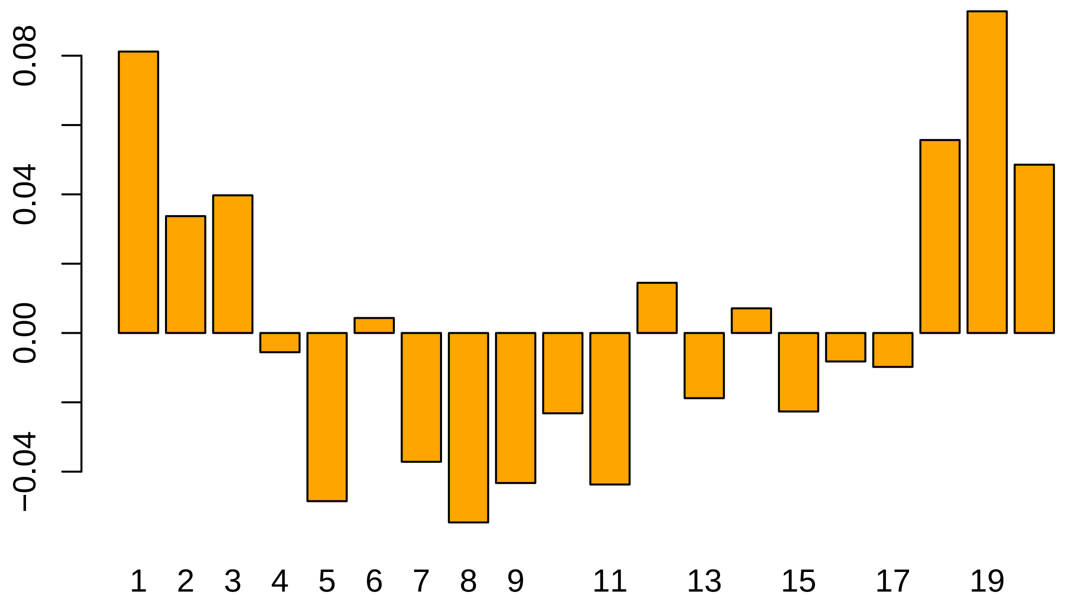


```
cusumData =  cumsum(gibbsDraws[,1])/seq(1,nDraws)
minY = floor(min(cusumData))
```

```r
maxY = ceiling(max(cusumData))
plot(1:nDraws, cusumData, type = "l", col = 'orange', ylab='Cumulative estimate',
     lwd = 1, axes=FALSE, xlab = 'MCMC iteration', xlim = c(0,nDraws),
     ylim = c(minY,maxY), main = 'Cusum - Gibbs')
lines(seq(1,nDraws),1*matrix(1,1,nDraws),col = 'orange', lwd=1)
axis(side = 1, at = seq(0, nDraws, by = 10))
axis(side = 2, at = seq(minY, maxY, by = 0.5))
```

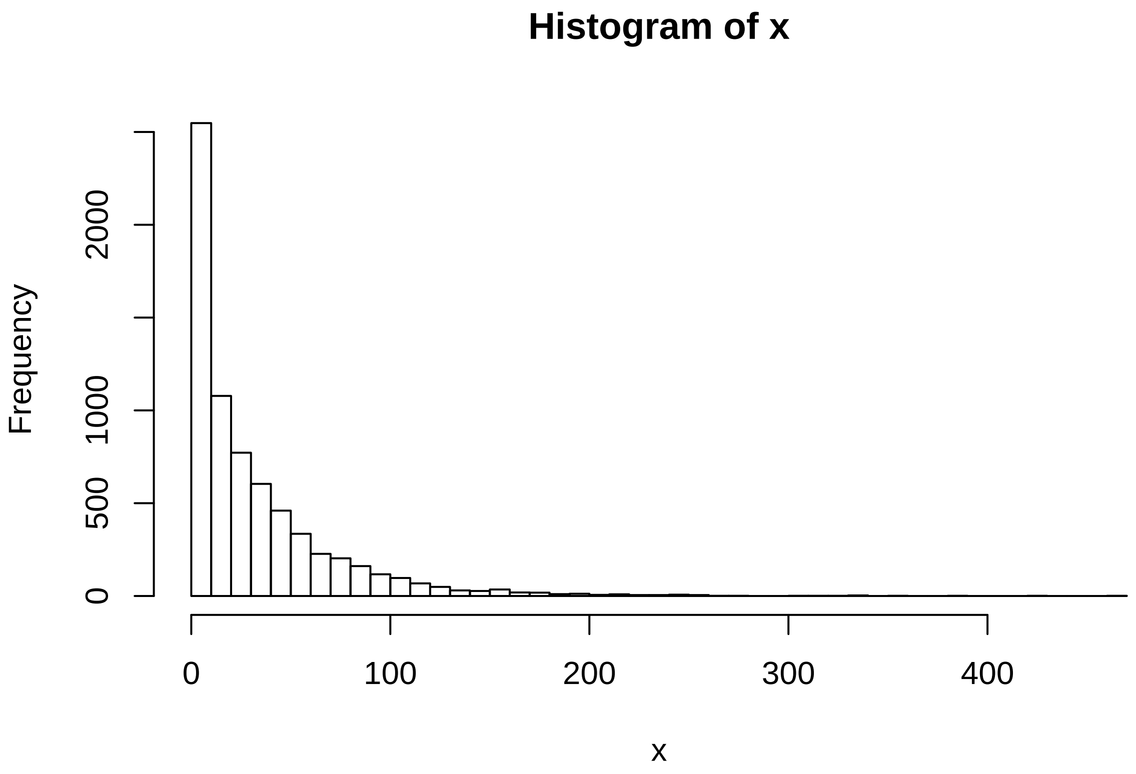


```r
a = acf(gibbsDraws[,1], main='Gibbs draws', lag.max = 20, plot = F)
barplot(height = a$acf[-1], names.arg=seq(1,20), col = 'orange')
```



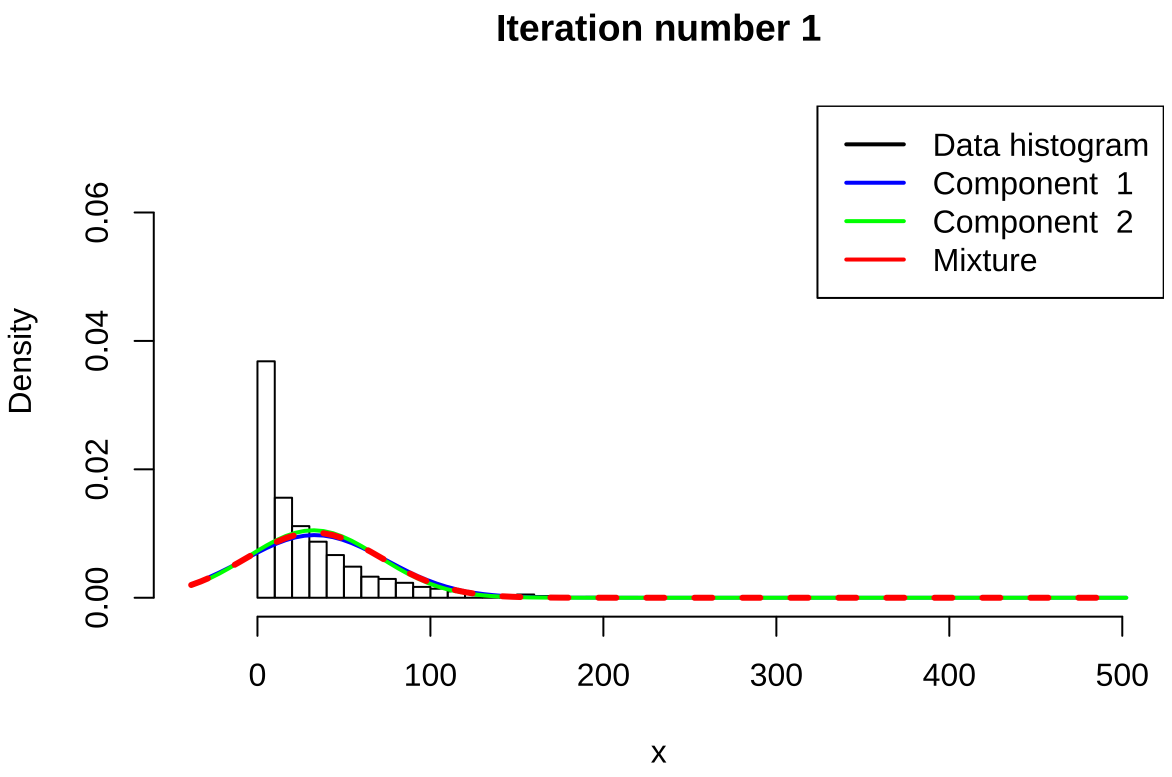
(b)

```r
# Estimating a simple mixture of normals
# Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com

##########    BEGIN USER INPUT #################
# Data options
rawData <- read.table("rainfall.dat",header=TRUE)
x <- as.matrix(rawData)

# Model options
nComp <- 2    # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(30,nComp) # Prior mean of mu
tau2Prior <- rep(10,nComp) # Prior std of mu
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(4,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 10 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0 # Adding sleep time between iterations for plotting
################   END USER INPUT ###############

###### Defining a function that simulates from the
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

####### Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
  nCat <- length(param)
  piDraws <- matrix(NA,nCat,1)
  for (j in 1:nCat){
    piDraws[j] <- rgamma(1,param[j],1)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum of the elements in that co
  return(piDraws)
}

# Simple function that converts between two different representations of the mixture allocation
S2alloc <- function(S){
  n <- dim(S)[1]
  alloc <- rep(0,n)
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component all
mu <- quantile(x, probs = seq(0.2,0.8,length = nComp))
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)

# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x, n = 50)$density))
```

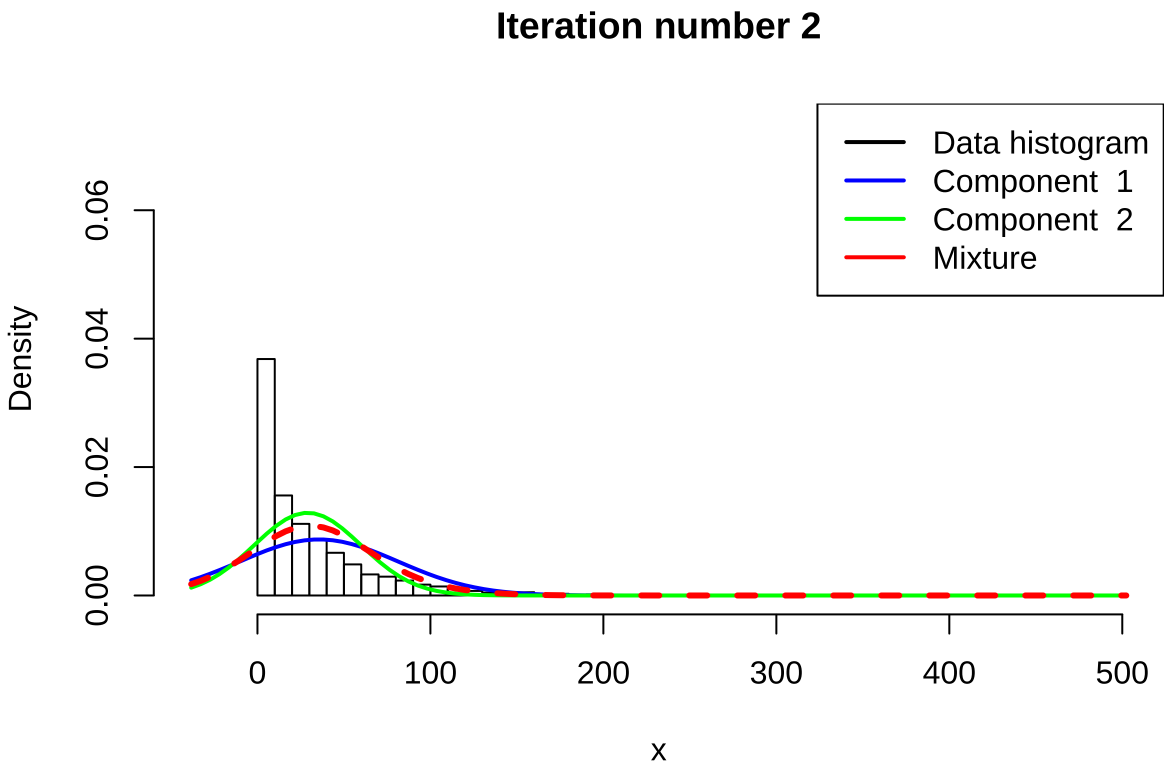


```r
for (k in 1:nIter){
  message(paste('Iteration number:',k))
  alloc <- S2alloc(S) # Just a function that converts between different representations of the group al
  nAlloc <- colSums(S)
  print(nAlloc)
  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)

  # Update mu's
  for (j in 1:nComp){
    precPrior <- 1/tau2Prior[j]
    precData <- nAlloc[j]/sigma2[j]
    precPost <- precPrior + precData
    wPrior <- precPrior/precPost
    muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
    tau2Post <- 1/precPost
    mu[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }

  # Update sigma2's
  for (j in 1:nComp){
    sigma2[j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j], scale = (nu0[j]*sigma2_0[j] + sum((x[alloc
  }

  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean = mu[j], sd = sqrt(sigma2[j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }

  # Printing the fitted density against data histogram
  if (plotFit && (k%%1 ==0)){
    effIterCount <- effIterCount + 1
    hist(x, breaks = 50, freq = FALSE, xlim = c(xGridMin,xGridMax), main = paste("Iteration number",k),
    mixDens <- rep(0,length(xGrid))
    components <- c()
    for (j in 1:nComp){
      compDens <- dnorm(xGrid,mu[j],sd = sqrt(sigma2[j]))
      mixDens <- mixDens + pi[j]*compDens
      lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
      components[j] <- paste("Component ",j)
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount

    lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
    legend("topright", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    Sys.sleep(sleepTime)
  }

}
```

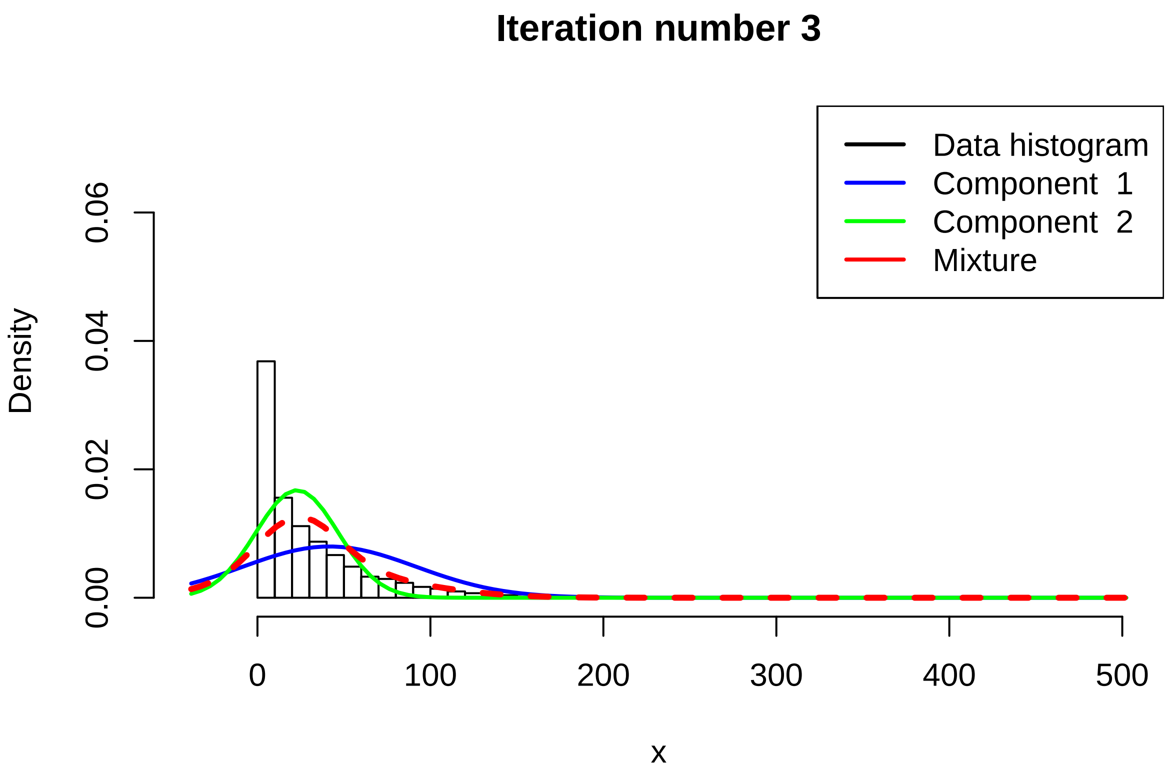
```
## Iteration number: 1

## [1] 3427 3492

## Iteration number: 2
```



```
## [1] 3363 3556
## Iteration number: 3
```

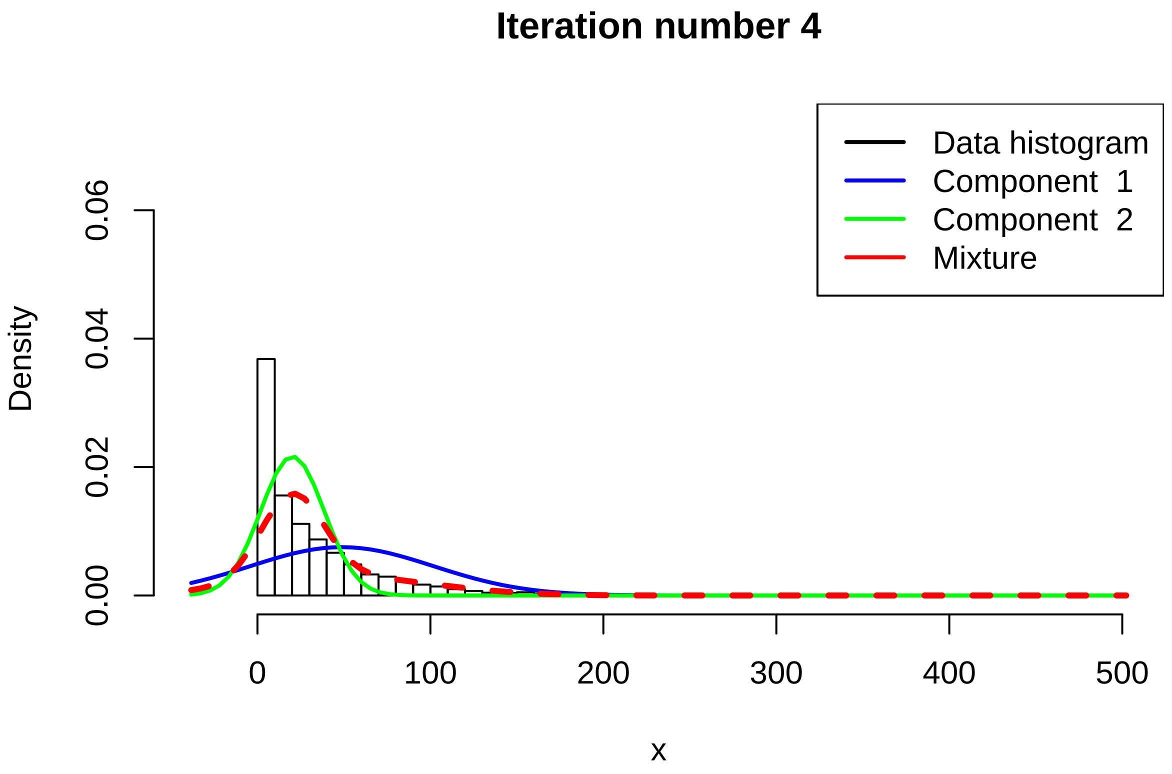


```
## [1] 3108 3811
## Iteration number: 4
```



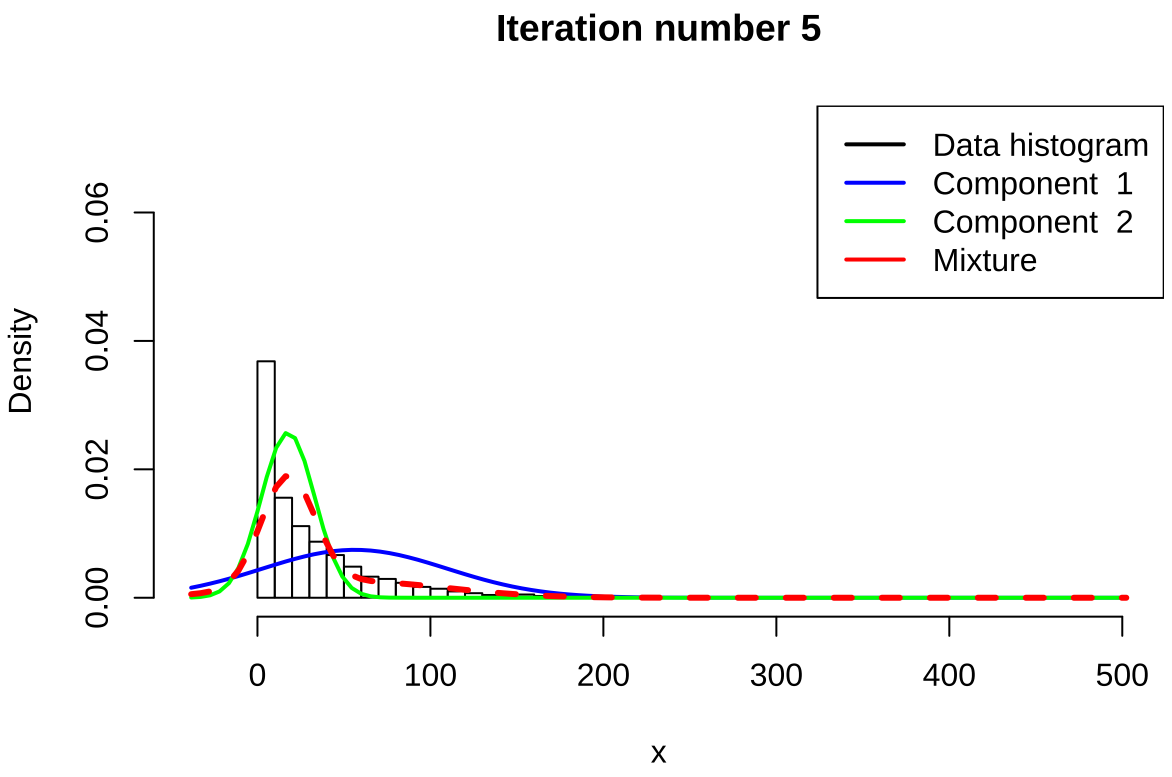
```
## [1] 2761 4158
```

```
## Iteration number: 5
```



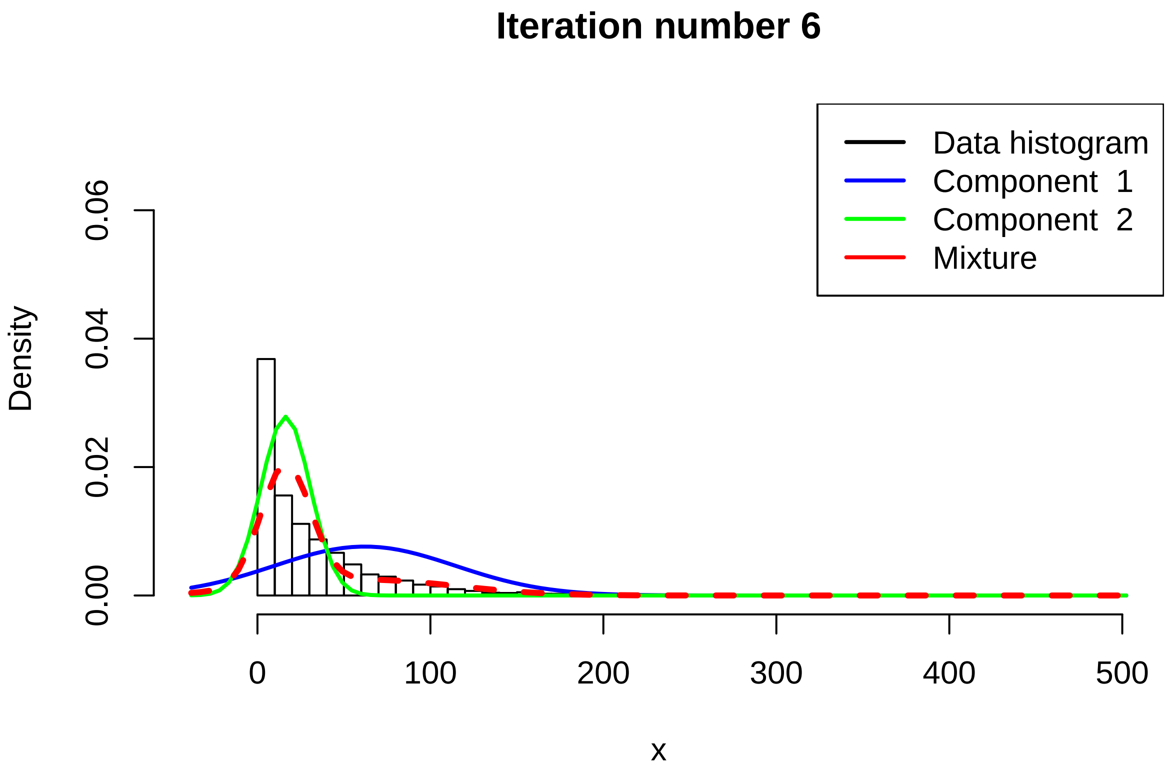
```
## [1] 2360 4559
```

```
## Iteration number: 6
```



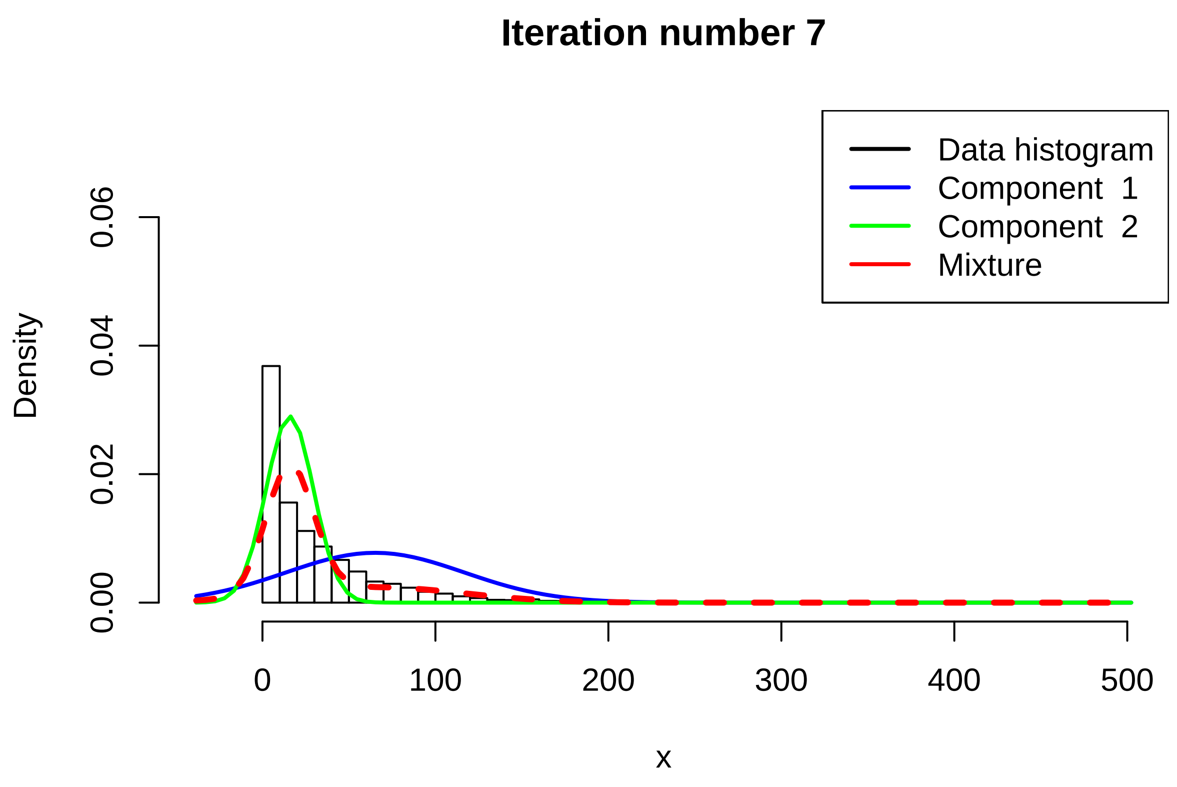
```
## [1] 2206 4713
```

```
## Iteration number: 7
```



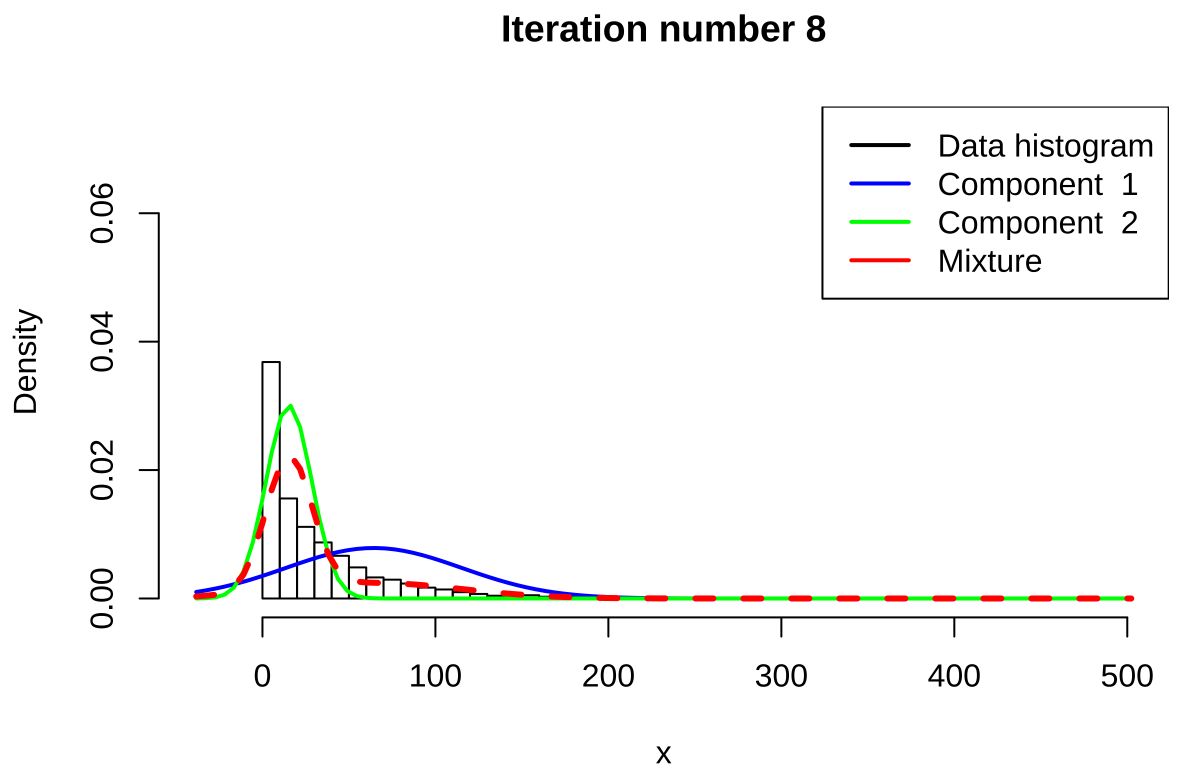
```
## [1] 2129 4790
```

```
## Iteration number: 8
```



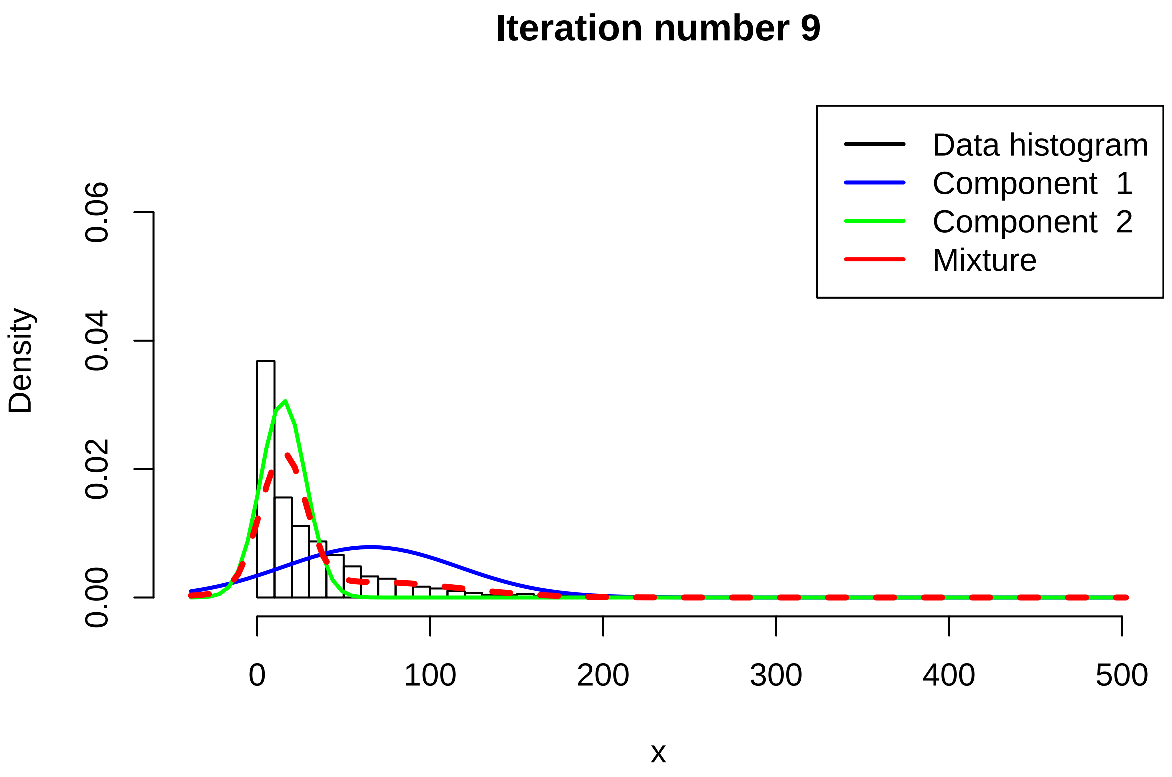
```
## [1] 2144 4775
```

```
## Iteration number: 9
```

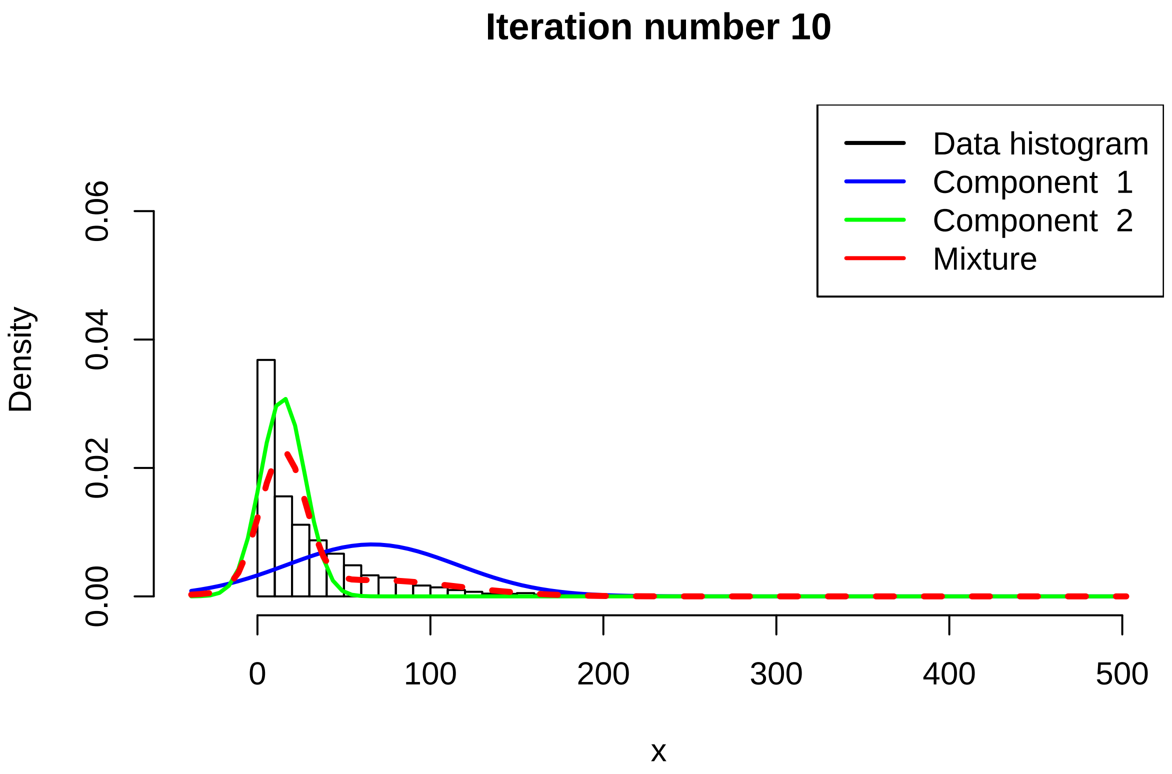


```
## [1] 2195 4724
```

```
## Iteration number: 10
```



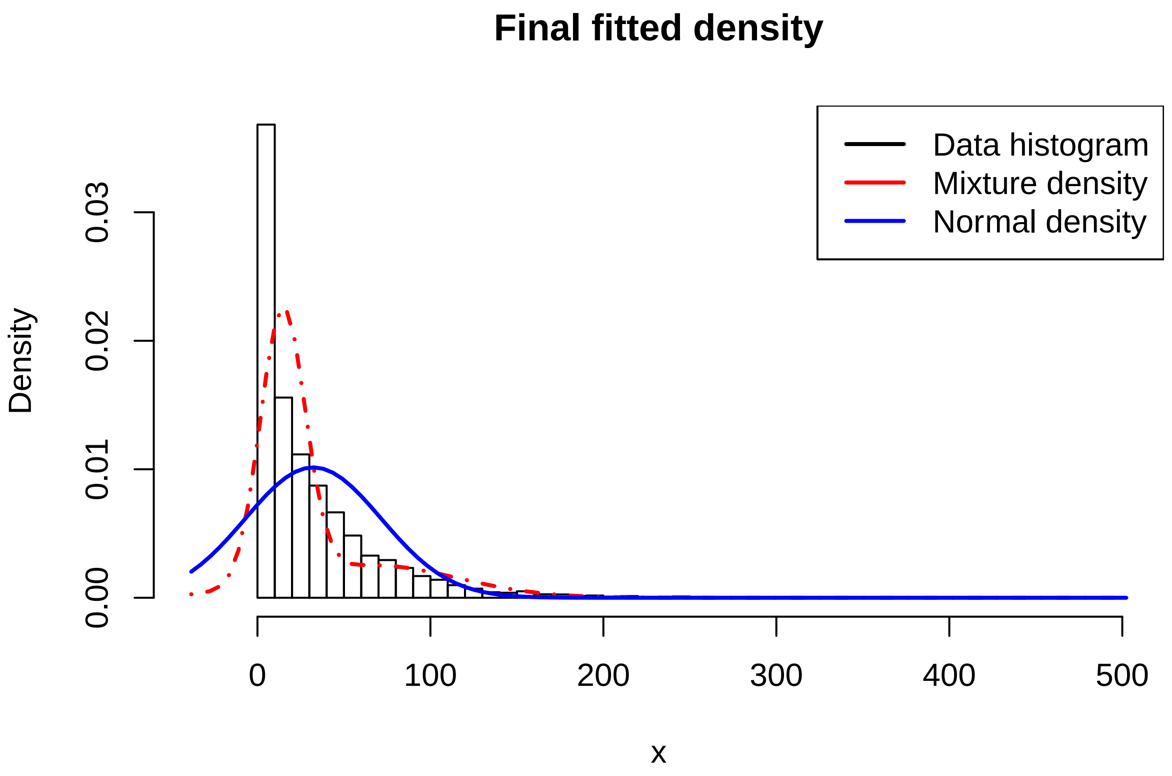
```
## [1] 2158 4761
```



The plots above shows the samples covnerging on the distribution of the data. The multinomial model fits better to the data then the model in part (a). It is always atleast as good or better then the other model but instead risks to overfit on the given data.

(c)

```r
hist(x, breaks = 50, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")
lines(xGrid, mixDens, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"), col=c(
```



Task 2

(a)

```r
rawData <- read.table("ebayNumberOfBidderData.dat",header=TRUE)

y <- as.vector(rawData[,1]); # Data from the read.table function is a data frame. Let's convert y and X
X <- rawData[,2:10];

model <- glm(y ~ PowerSeller+VerifyID+Sealed+Minblem+MajBlem+LargNeg+LogBook+MinBidShare, data=X, famil
print(model)
```

```
##
## Call:  glm(formula = y ~ PowerSeller + VerifyID + Sealed + Minblem +
##     MajBlem + LargNeg + LogBook + MinBidShare, family = poisson(),
##     data = X)
##
## Coefficients:
## (Intercept)  PowerSeller     VerifyID       Sealed      Minblem
##     1.07244     -0.02054     -0.39452      0.44384     -0.05220
##     MajBlem      LargNeg      LogBook  MinBidShare
##    -0.22087      0.07067     -0.12068     -1.89410
##
## Degrees of Freedom: 999 Total (i.e. Null);  991 Residual
## Null Deviance:       2151
## Residual Deviance: 867.5     AIC: 3610
```

```r
print(summary(model))
```

```
##
## Call:
## glm(formula = y ~ PowerSeller + VerifyID + Sealed + Minblem +
##     MajBlem + LargNeg + LogBook + MinBidShare, family = poisson(),
##     data = X)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
```

```
##
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

The covariates that are significant is the variables: VerifyID, Sealed, MajBlem, LogBook and MinBidShare. I.e. those that have a Pr(abs(z)) > 2, also denothed with "*" in the output above.

(b)

```r
xMatrix = as.matrix(X)
sigmaGPrior = 100*solve((t(xMatrix)%*%xMatrix))
tau = 10
chooseCov <- c(1:9)

covNames <- names(rawData)[2:length(names(rawData))];
xMatrix <- xMatrix[,chooseCov]; # Here we pick out the chosen covariates.
covNames <- covNames[chooseCov];
nPara <- dim(X)[2];

# Setting up the prior
mu <- as.vector(rep(0,nPara)) # Prior mean vector
Sigma <- tau^2*diag(nPara);

PoiPost <- function(theta,y,X, mu, SigmaGPrior) {
  nPara <- length(theta);
  linPred <- X%*%theta;
  logPoiLik <- sum( linPred*y -exp(linPred) - log(factorial(y)));
  if (abs(logPoiLik) == Inf) logPoiLik = -20000; # Likelihood is not finite, stear the optimizer away f
  logBetaPrior <- dmvnorm(theta, matrix(0,nPara,1), SigmaGPrior, log=TRUE);
  return(logPoiLik + logBetaPrior)
}

initVal <- as.vector(rep(0,dim(X)[2]));
logPost = PoiPost;
OptimResults<-optim(initVal,logPost,gr=NULL,y,xMatrix,mu,Sigma,method=c("BFGS"),control=list(fnscale=-1

approxPostStd <- sqrt(diag(-solve(OptimResults$hessian)))
names(approxPostStd) <- covNames # Naming the coefficient by covariates

betatilde = OptimResults$par
print("Betatilde: ")
```

```
## [1] "Betatilde: "
```

```r
print(betatilde)
```

```
## [1]  1.07245146 -0.02054160 -0.39448259  0.44382930 -0.05219690 -0.22084701
## [7]  0.07066972 -0.12065535 -1.89401063
```

```r
print("Jacobiany beta: ")
```

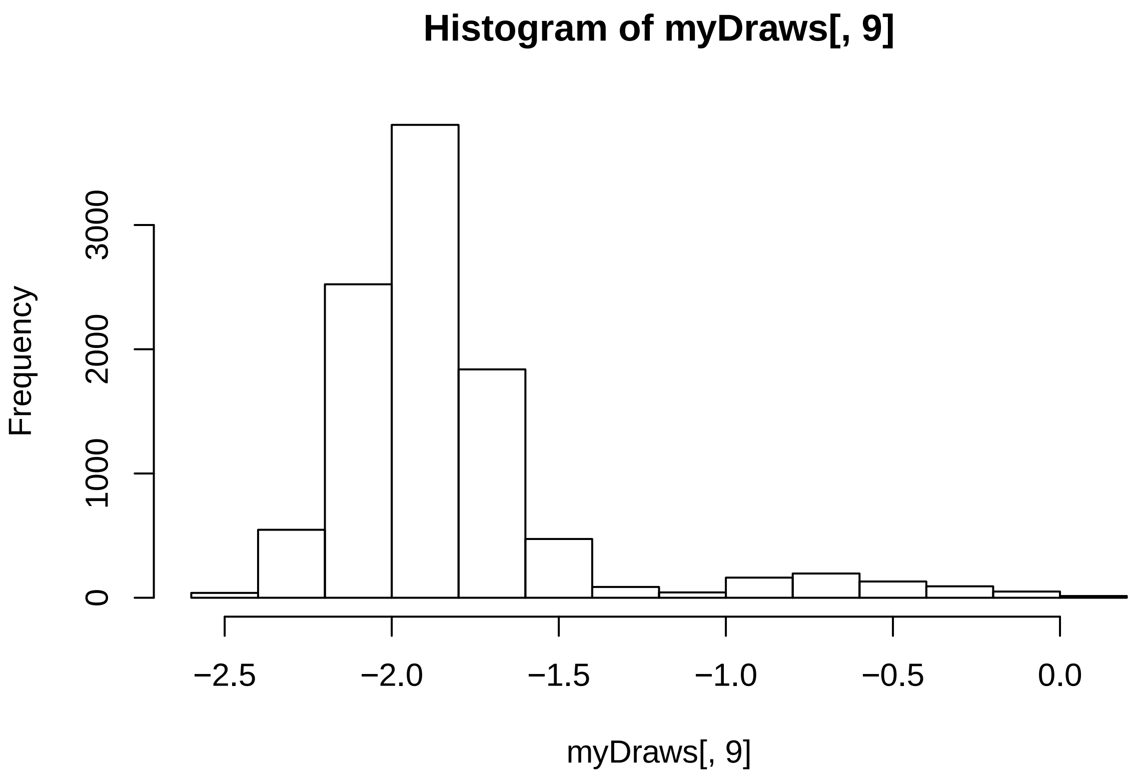
```
## [1] "Jacobiany beta: "
```

```r
print(approxPostStd)
```

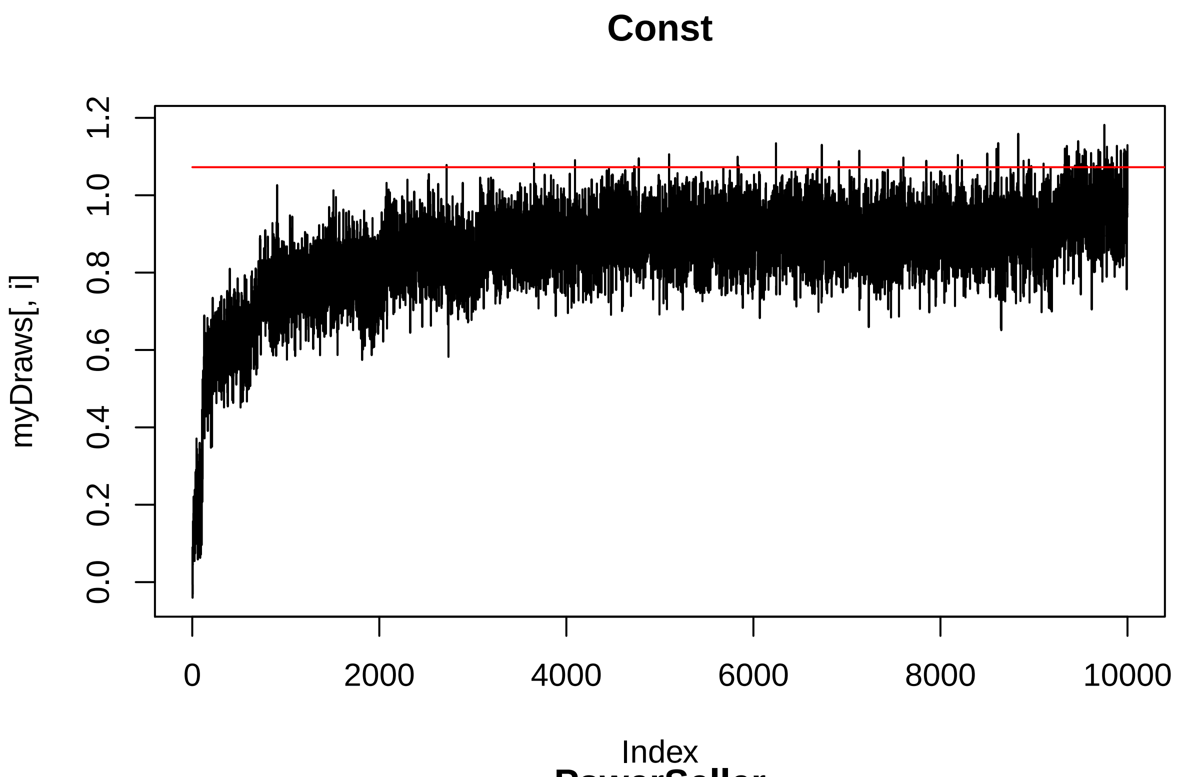
```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem 
##  0.03077417  0.03678186  0.09242044  0.05056213  0.06019999  0.09143186 
##     LargNeg     LogBook MinBidShare 
##  0.05633101  0.02896411  0.07123677
```

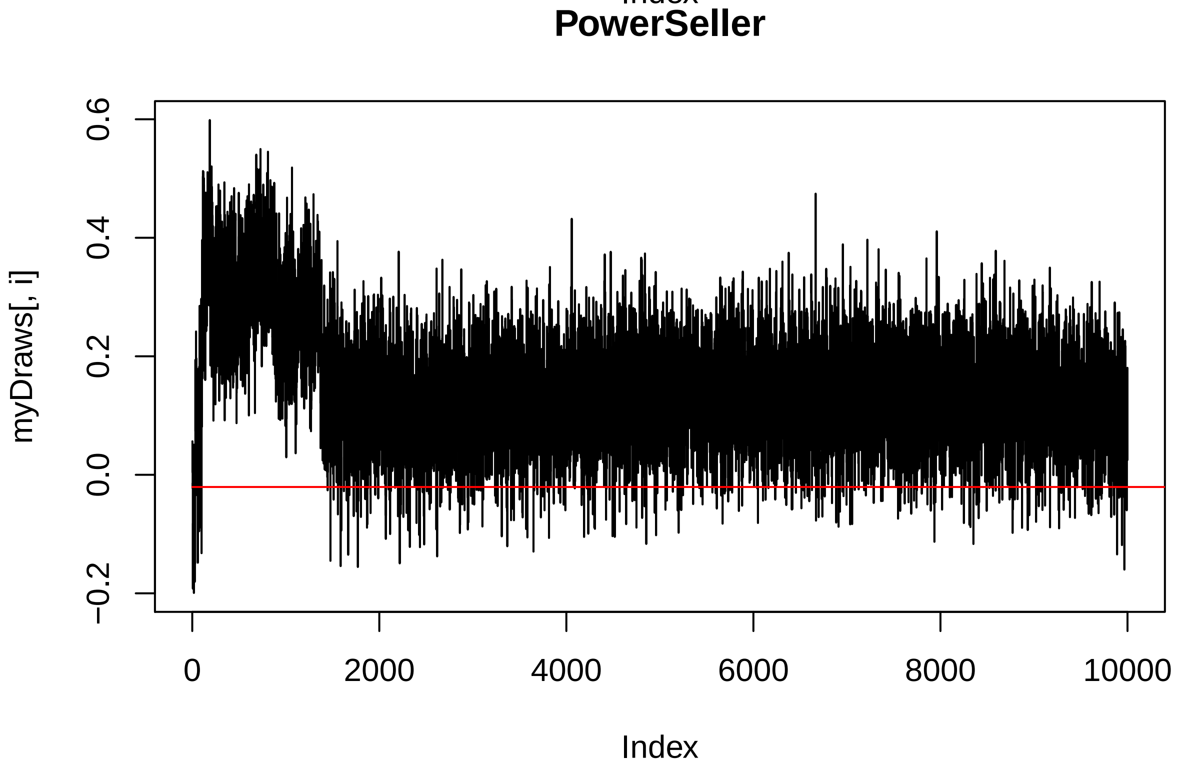
(c)

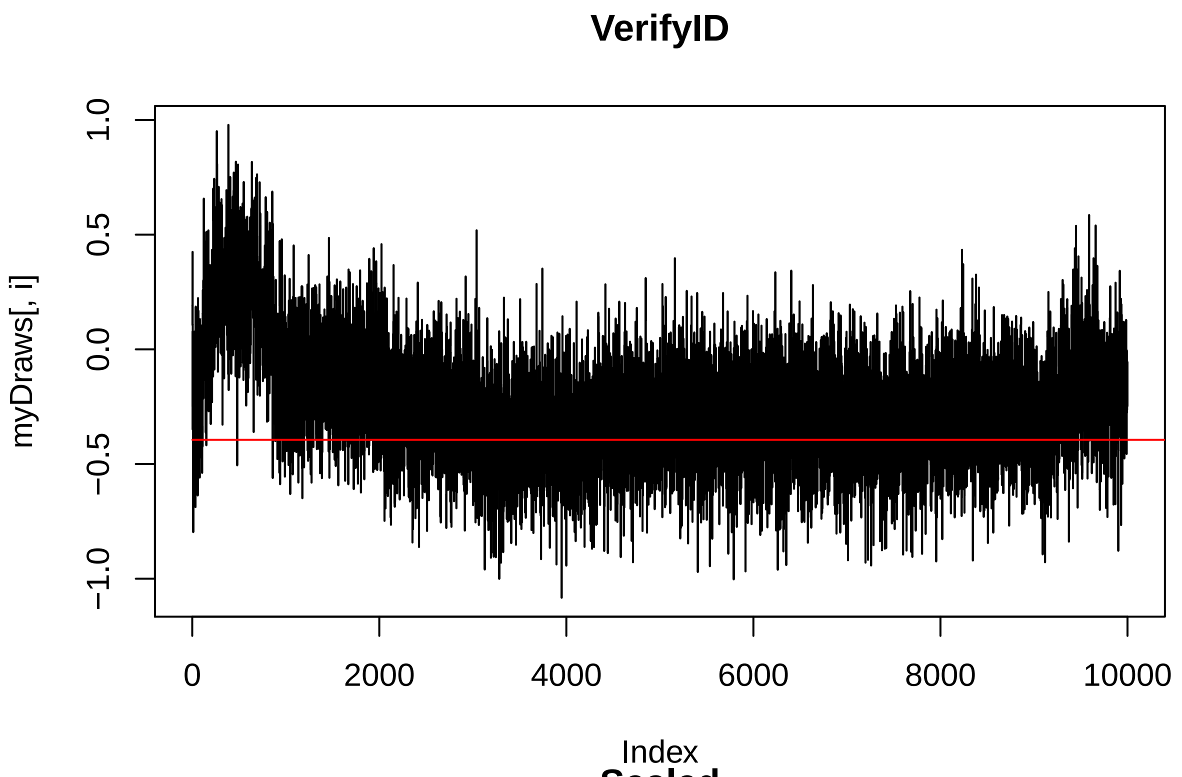
```r
RWMSampler <- function(logPostFunc, n,  c, covar,  ...) {
  currentTheta = as.vector(rep(0, dim(covar)[1]))
  draws = matrix(0, nrow = n, ncol = dim(covar)[1])
  oldProbability = logPostFunc(currentTheta, ...)
  for(i in 1:n) {
    currentDraw <- rmvt(1, mu = currentTheta, S = c*covar)
    newProbability <- logPostFunc(as.vector(currentDraw), ...)

    alpha = min(1, newProbability/oldProbability)
    uniformDraw = runif(1, 0, 1)
    if(uniformDraw >= alpha) {
      oldProbability = newProbability
      currentTheta = currentDraw
    }
    draws[i,] = currentDraw
  }
  return(draws)
}

myDraws <- RWMSampler( PoiPost, 10000, 5, diag(diag(-solve(OptimResults$hessian))),
                       y, xMatrix, mu, Sigma)

hist(myDraws[,9])
```



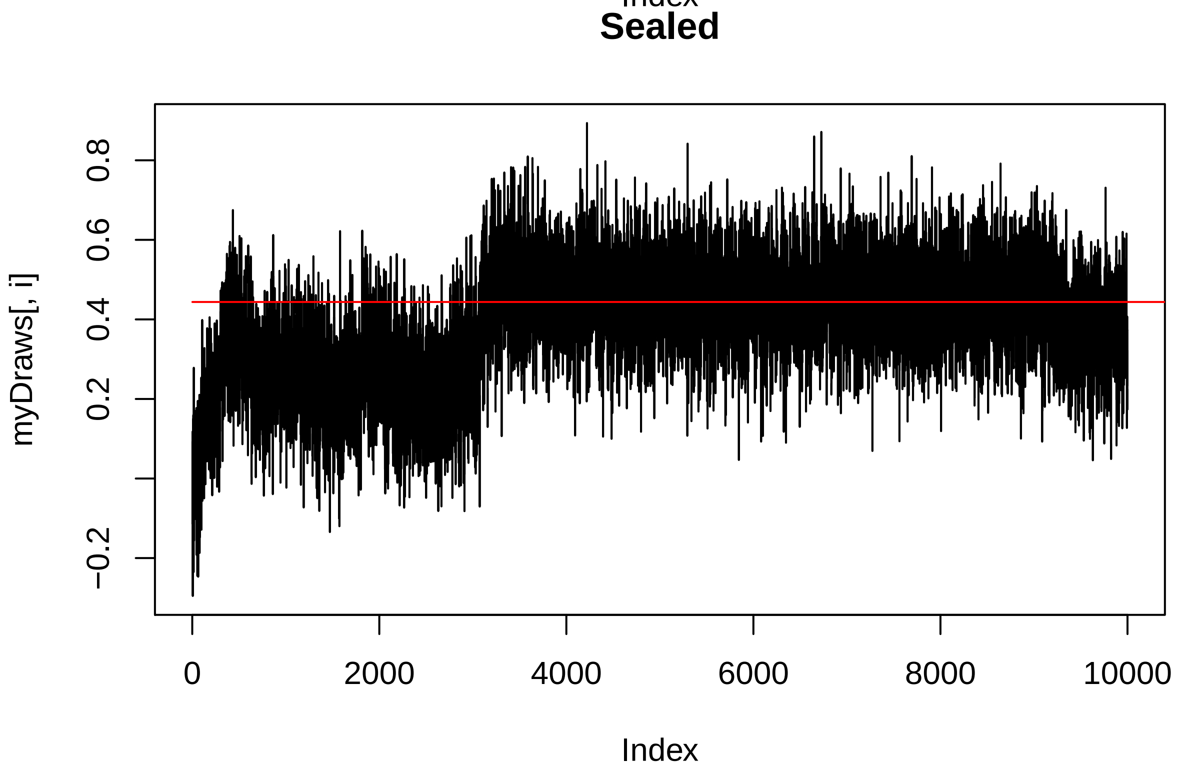
```
for(i in 1:9){
  plot(myDraws[,i], type='s')
  a = c(rep(betatilde[i],length(myDraws)))
  lines(a, col='red')
  title(covNames[i])
}
```


Const
myDraws[, i]
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2000
4000
6000
8000
10000
Index

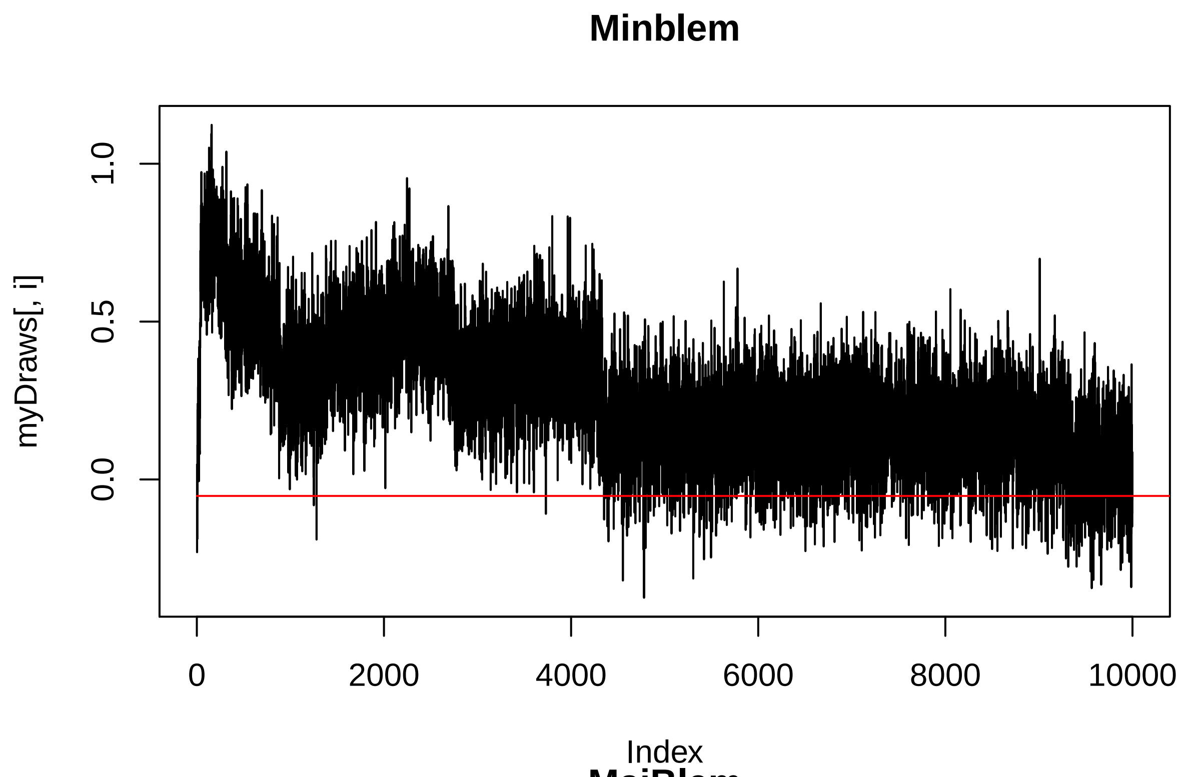

PowerSeller
myDraws[, i]
−0.2
0.0
0.2
0.4
0.6
0
2000
4000
6000
8000
10000
Index

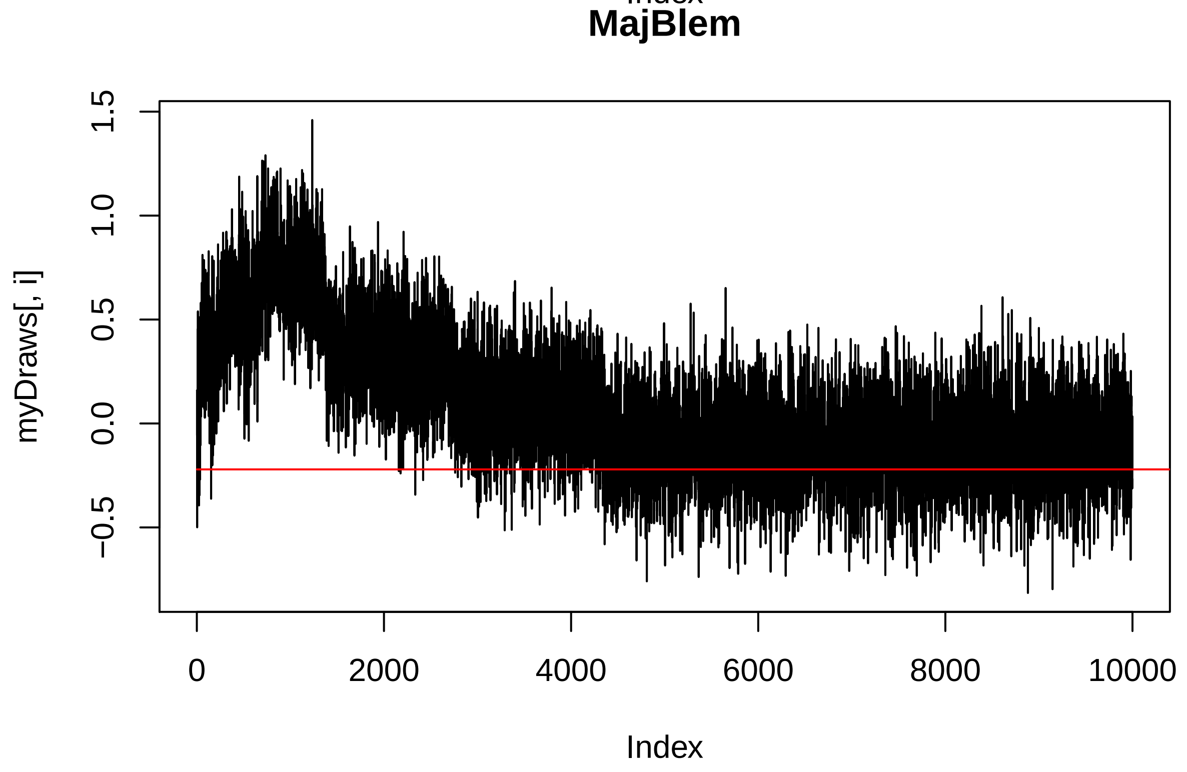


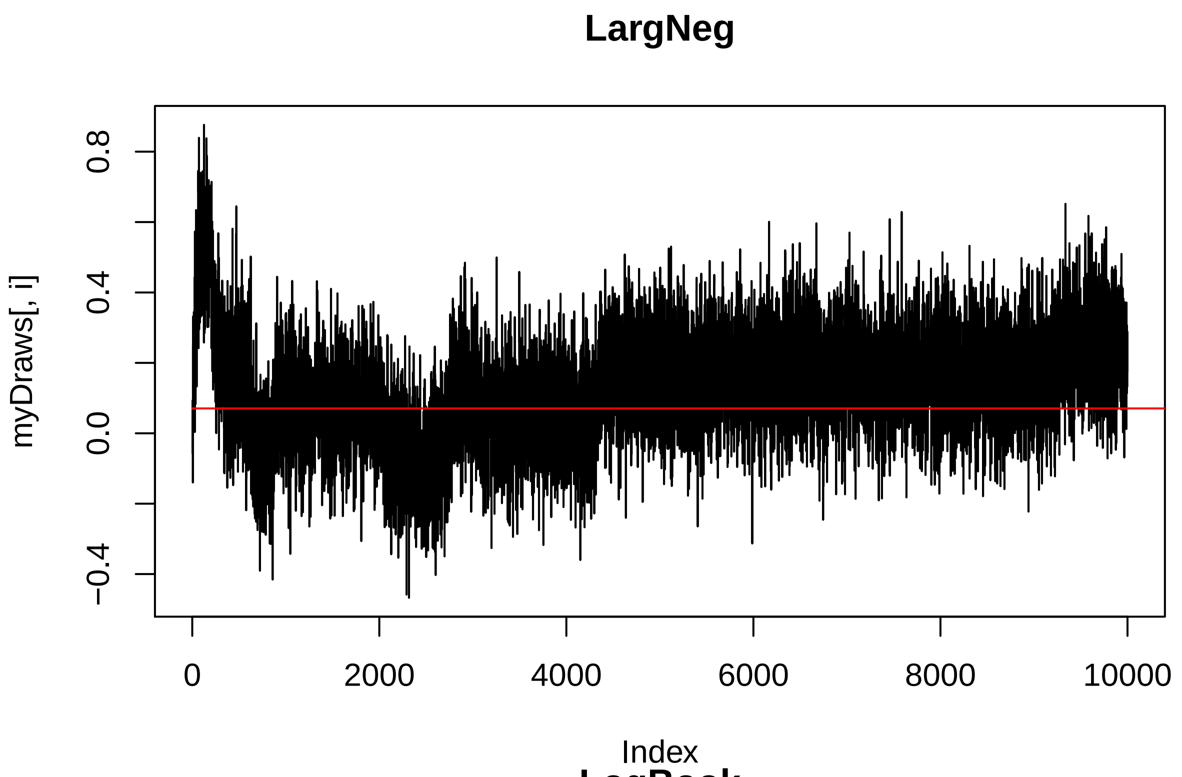
VerifyID
myDraws[, i]
1.0
0.5
0.0
−0.5
−1.0
0
2000
4000
6000
8000
10000
Index

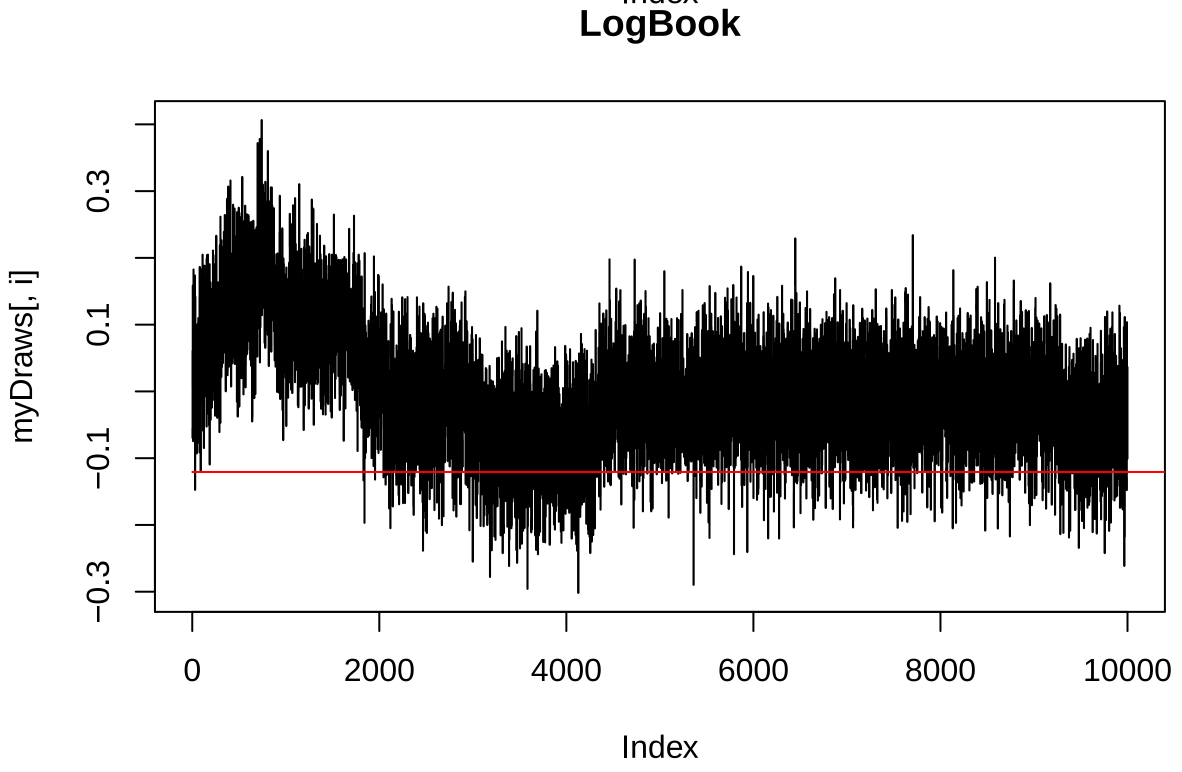


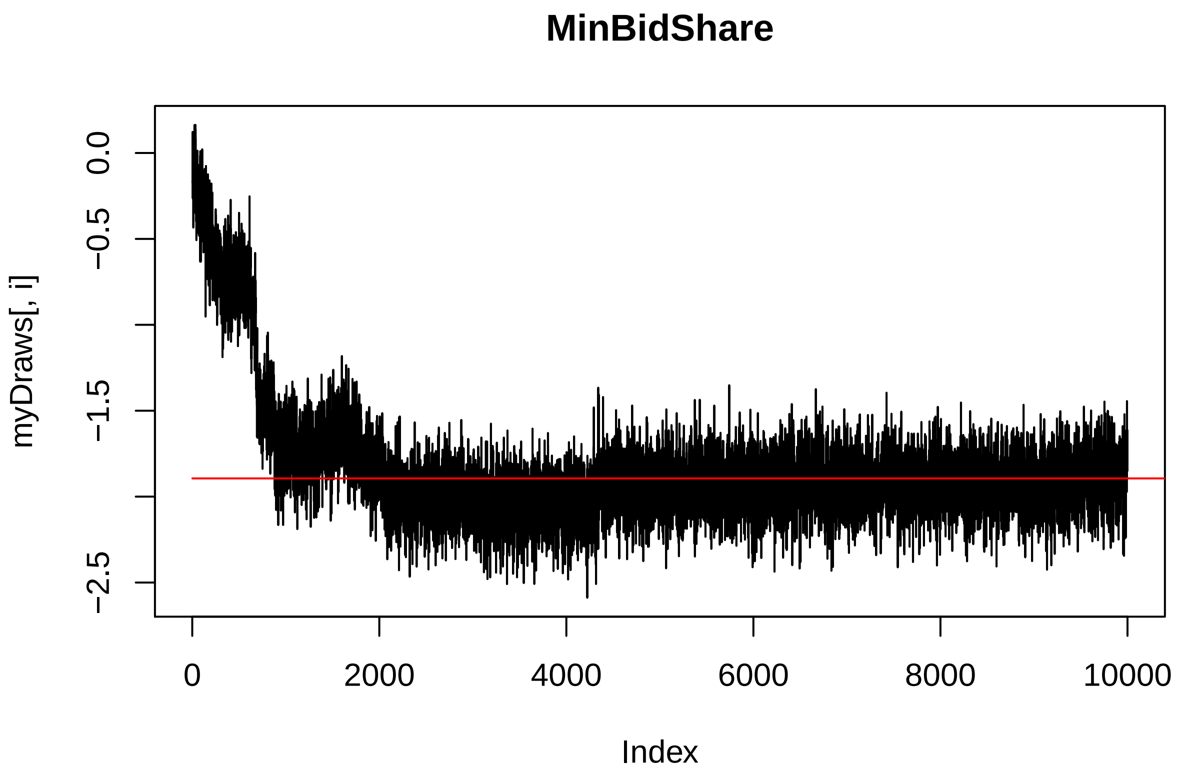
Sealed
myDraws[, i]
0.8
0.6
0.4
0.2
−0.2
0
2000
4000
6000
8000
10000
Index


Minblem
myDraws[, i]
1.0
0.5
0.0
0
2000
4000
6000
8000
10000
Index


MajBlem
myDraws[, i]
1.5
1.0
0.5
0.0
−0.5
0
2000
4000
6000
8000
10000
Index


LargNeg
myDraws[, i]
0.8
0.4
0.0
−0.4
0
2000
4000
6000
8000
10000
Index


LogBook
myDraws[, i]
0.3
0.1
−0.1
−0.3
0
2000
4000
6000
8000
10000
Index



The draws seem to move in the right direction but not all variables are close to the approximated corresponding betatilde(red line) as in the earlier models in (a) nad (b), might be due to local optima or possible dependency on other vairables.