Lab1

Ludwig Thaung Elon Brange (ludth852, elobr959)

2019-04-01

Task 1

Let y1, . . . , yn|theta - Bern(theta), and assume that you have obtained a sample with s = 14 successes in n = 20 trials. Assume a Beta(alpha0, beta0) prior for theta and let alpha0 = beta0 = 2.

a) Posterior theta|y - Bern(alpha0 + s, beta0 + f).

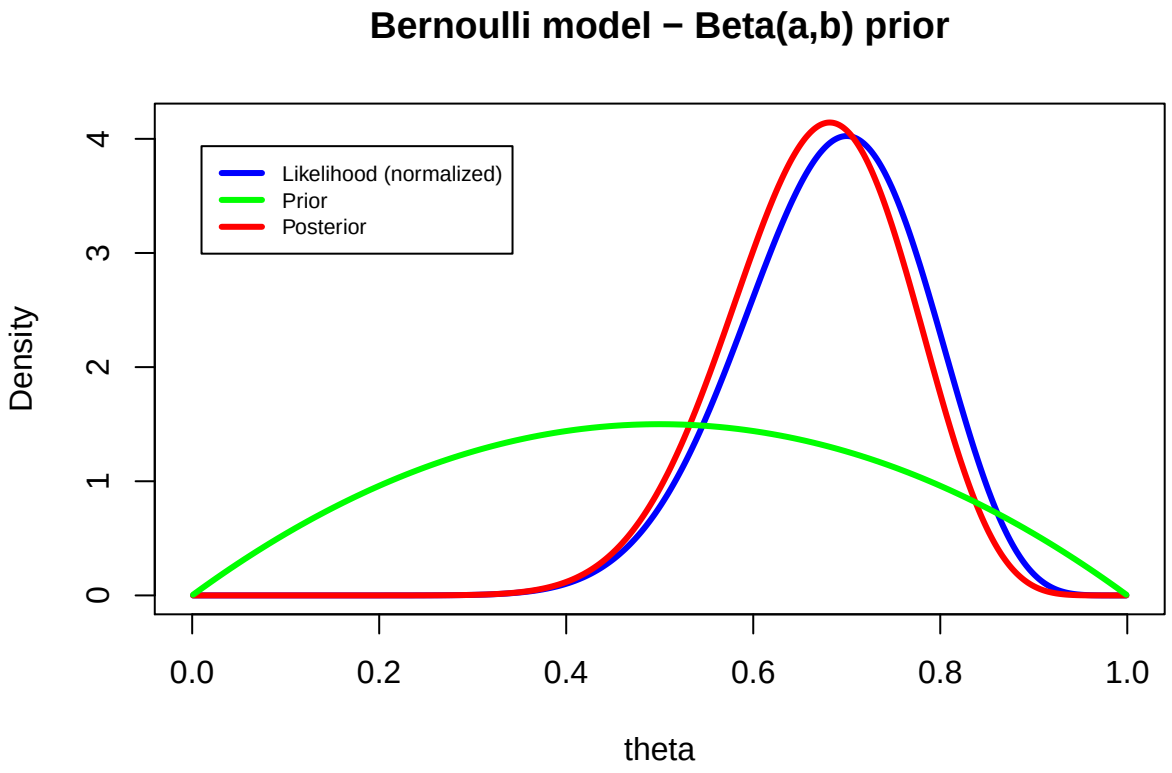
Verify graphically that the posterior mean and standard deviation converges to the true values as the number of random draws grows large.

For 20 draws we get:

```
## [1] "Posterior Mean GT: 0.666666666666667"
## [1] "ground truth std: 0.0942809041582063"
## [1] "std: 0.0923051801582707"
## [1] "Mean: 0.608671325942425"
```

For 10 0000 draws we get:

```
## [1] "Posterior Mean GT: 0.666666666666667"
## [1] "ground truth std: 0.0942809041582063"
## [1] "std: 0.0944912346276031"
## [1] "Mean: 0.666799432655026"
```



b)

Using 10 000 draws we seek to comupte the posterior probability Pr(theta $< 0.4|y$).

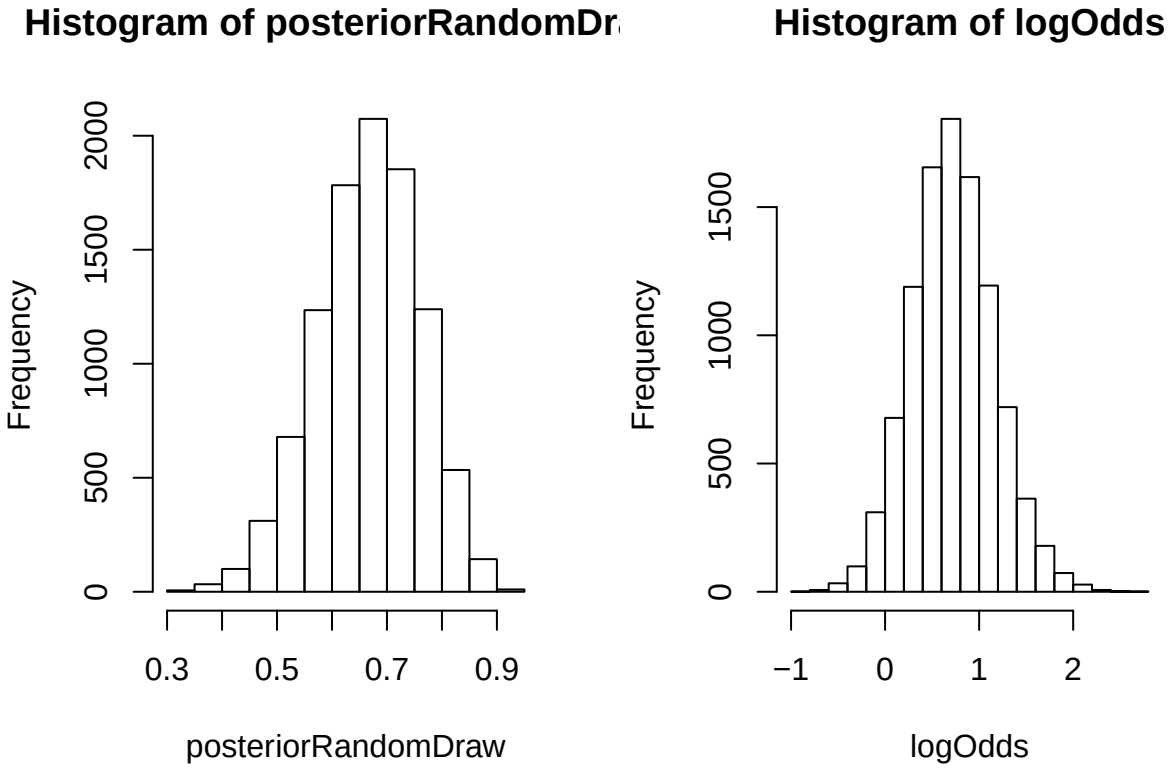
```
## [1] "propability condition with random: 0.0036"
```

```
## [1] "ground truth probability: 0.00397268082810898"
```

Looking at the plot above, the probability for theta $< 0.4|y$ is very small. The simulated value is relatively close to the ground truth. (Note: The further to the left on the tail, the larger sample we will need as the data points become more sparse.)

c)

Computing posterior distribution of the log-odds by simulating 10 000 random draws.

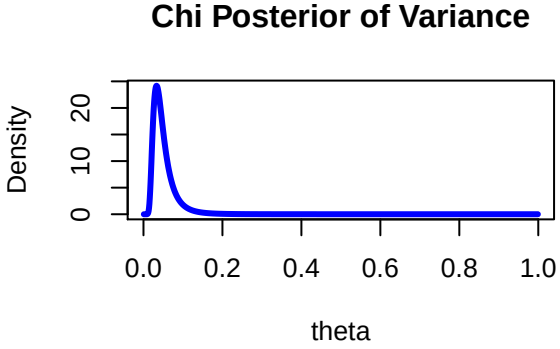


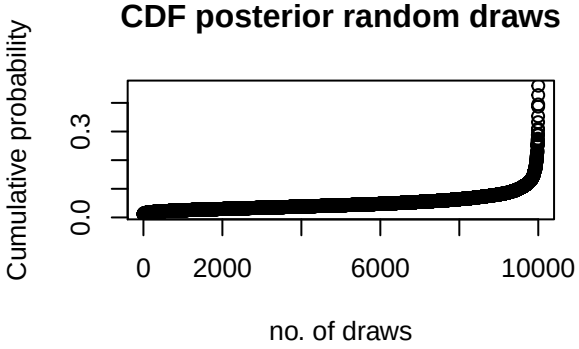

```
##
## Call:
##  density.default(x = logOdds)
##
## Data: logOdds (10000 obs.);  Bandwidth 'bw' = 0.06221
##
##        x                 y
##  Min.   :-1.02163   Min.   :0.0000074
##  1st Qu.:-0.06905   1st Qu.:0.0068038
##  Median : 0.88353   Median :0.0875281
##  Mean   : 0.88353   Mean   :0.2621891
##  3rd Qu.: 1.83611   3rd Qu.:0.4995155
##  Max.   : 2.78869   Max.   :0.9207366
```

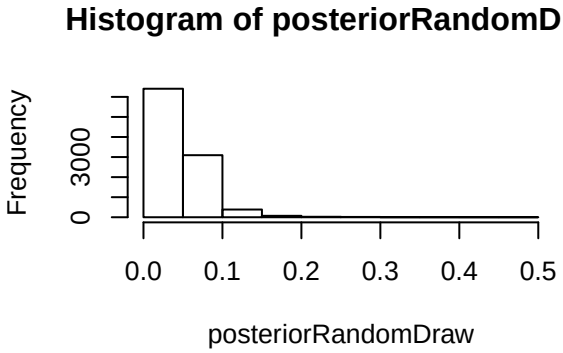
Task 2

Log-normal distribution and the Gini coefficient ###a) Simunaltion 10 000 draws from posterior of variance, assuming mean = 3.5 and comparing with the theoretical Inv Chi square posterior distribution.

"Chi posterior of variance" plot is the basis for comparison. The posterior CDF of the random draws together with the histogram of the random posterior shows that the randoms draws looks as expected in their cdf och histogram plots compared to the theoretical pdf, with the given look och the chi posterior pdf the cdf should have a relatively flat surface for a lot of values as a lot of values will be around 0.5, this is also shown when looking at the histogram.

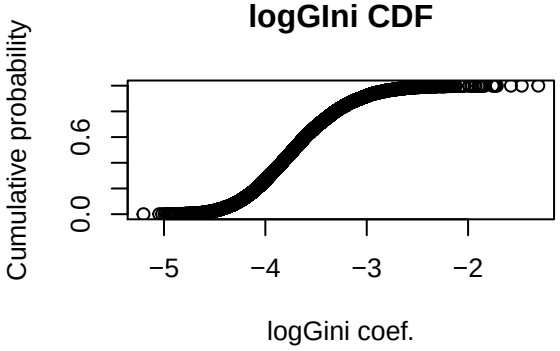


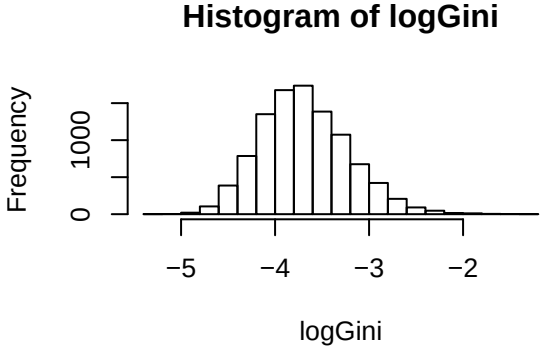




b)

Using posterior from a) to compute the posterior distribution of the Gini coefficient for the current dataset.





c)

Using posterior draws from b) to calculate a 95% euqal tail credible interval for the Gini coefficient G. In addition a kernel density estimate of the posterior of G to use that kernel density to compute a 95% HPD intercal for G.

```
##
## Call:
##  density.default(x = middleData)
##
## Data: middleData (9501 obs.);    Bandwidth 'bw' = 0.001584
##
##        x                  y
##  Min.   :0.006222   Min.   : 0.003
##  1st Qu.:0.023336   1st Qu.: 2.707
##  Median :0.040449   Median : 7.944
##  Mean   :0.040449   Mean   :14.594
##  3rd Qu.:0.057563   3rd Qu.:24.845
##  Max.   :0.074677   Max.   :44.743
## [1] "Lower end of interval: 0.0109745745699203"
## [1] "Upper end of interval: 0.0699240331269386"
```

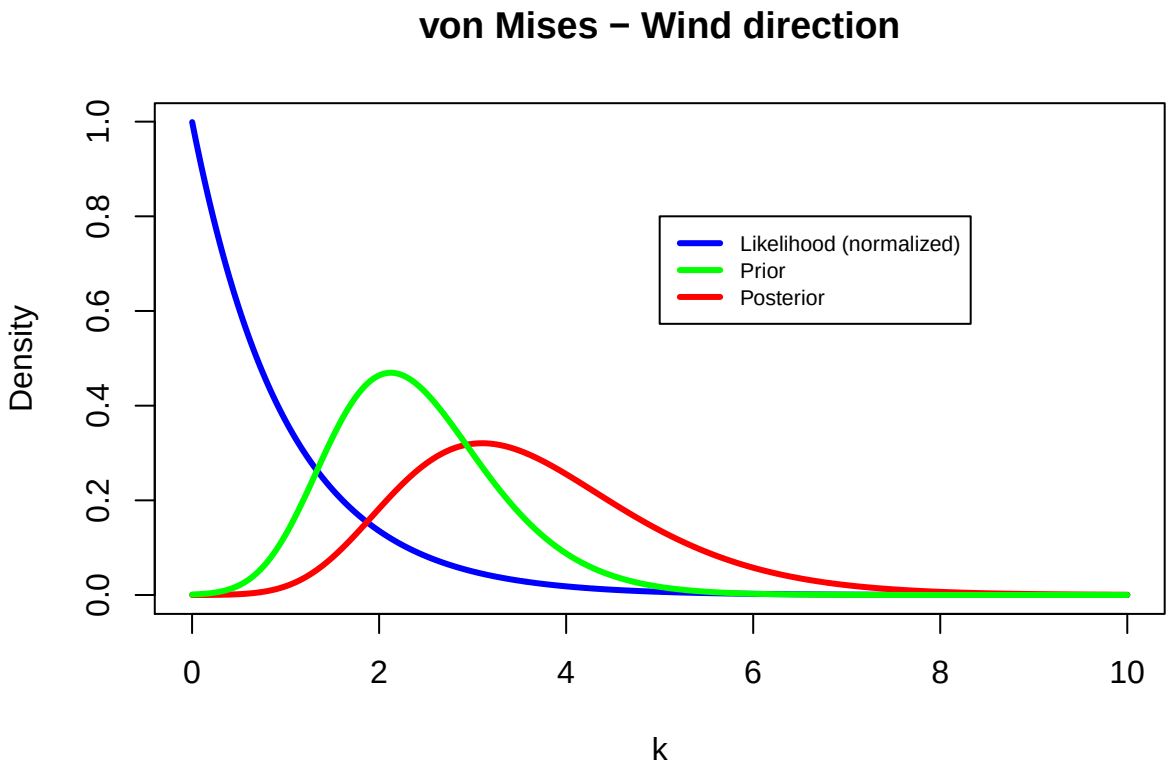
We can see that the interval for using our cutoff-method is more narrow than the interval for using the density-function as the kernel density estimate provides a interval of[0.006348, 0.074790] compared to the cutoff-interval of [0.01114952, 0.06998829].

Task 3

Bayesian inference for the concentration parameter in the von Mises distribution.

a)

Plot the posterior distribution of k (concentration parameter) for the wind direction.



b)

Approximate posterior mode of the concentration parameter k given the information in a).

```
## [1] 2.125
```

Output above shows the approximate mode of posterior, which is expected when comparing to the graph above(the green line).

TDDE07 - Lab 2

Ludwig Thaung (ludth852), Elon Brange (elobr959

5/4/2019

Task 1

(a)

```
TempLinkoping <- read_csv("TempLinkoping.csv")
```

```
## Parsed with column specification:
## cols(
##   time = col_double(),
##   temp = col_double()
## )
```

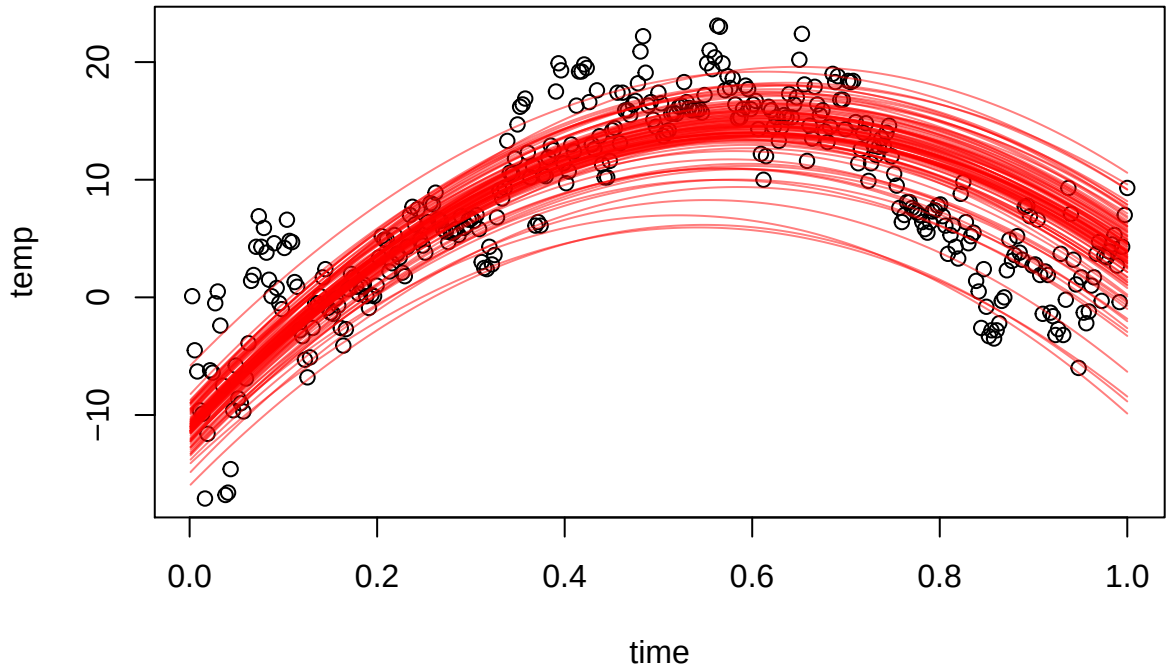
```
x = TempLinkoping['time']
y = TempLinkoping['temp']
x['time2'] = x^2
x['1'] = x['time']/x['time']
ph = x['time']
#Just to have the betas in right order
x['time'] = x['1']
x['1'] = x['time2']
x['time2'] = ph

matrix_x = data.matrix(x)
matrix_y = data.matrix(y)

mu0 = c(-11,85,-70)
omega0 = matrix(c(0.03, 0, 0, 0, 0.01, 0, 0, 0, 0.03), 3, 3)
v0 = 3
sigmasq0 = 0.03

e = rnorm(1, mean = 0, sd = sigmasq0)


betahat = inv((t(matrix_x)%*%matrix_x))%*%(t(matrix_x)%*%matrix_y)
randomSigma2 <- rinvchisq(n = 10, df = v0, scale = sigmasq0)
randomBetas <- c()
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
plot(TempLinkoping, col="black")
for(singleSigma in randomSigma2) {
  randomBeta <- rmvt(n = 10,mu = t(mu0), S = singleSigma*inv(omega0))
```

```
  for(k in 1:10) {
    ys = c()
    xs = cbind(matrix(1, 1000, 1), matrix(1:1000, 1000, 1)/1000, (matrix(1:1000, 1000, 1)/1000)^2)
    ys = xs%*%randomBeta[k, ]
    #for (i in 1:1000) {
    #  y = randomBeta[k, 1] + randomBeta[k, 2]*i/1000 + randomBeta[k, 3]*(i/1000)^2
    #  ys = c(ys, y)
    #  xs = c(xs, i/1000)
    #}
    lines(matrix(1:1000, 1000, 1)/1000, ys, col=alpha("red", 0.5))
  }
}
```

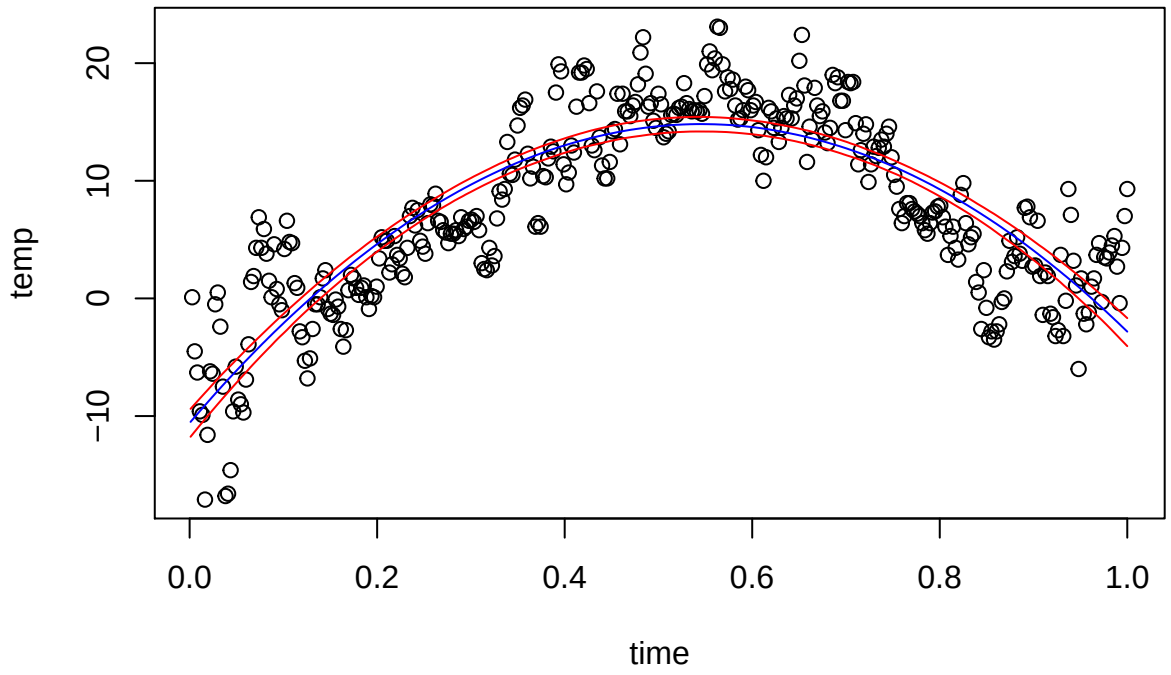


The collection of the curves look reasonable as they seem to follow the generalized curve of the datapoints.

(b)

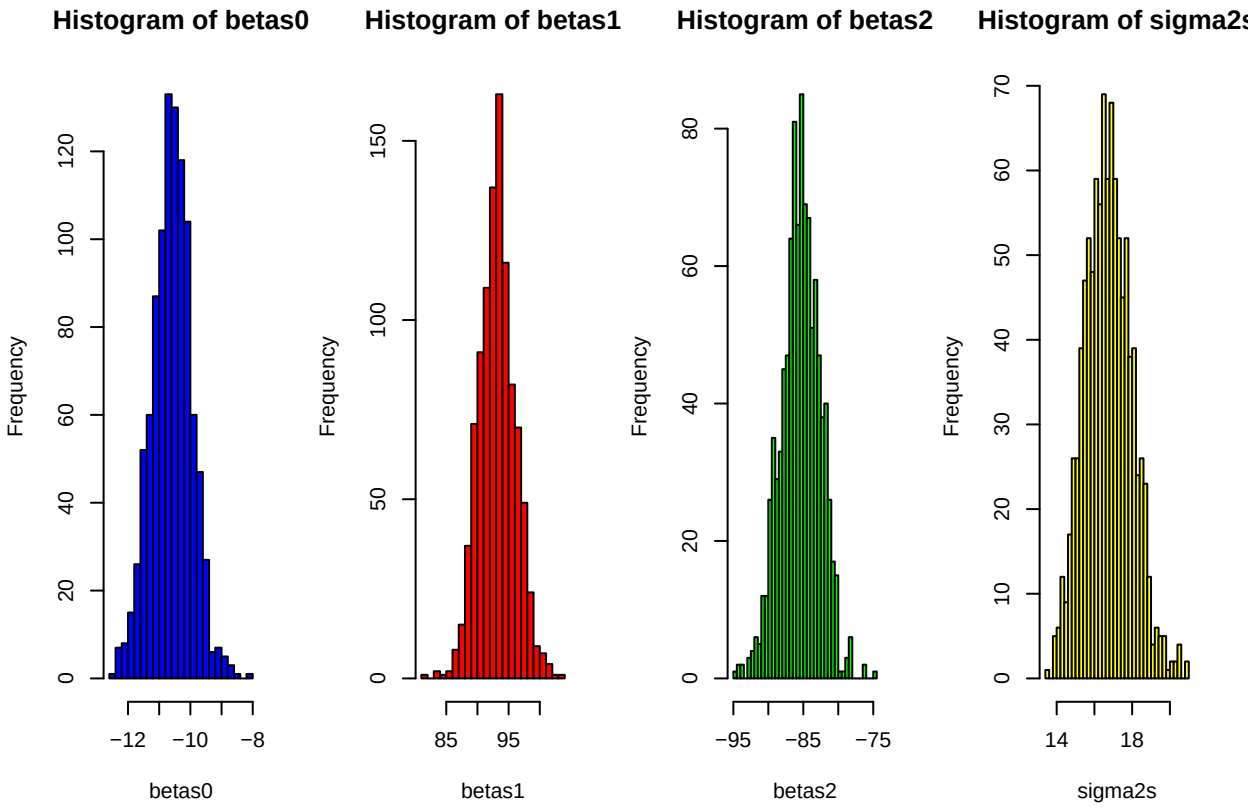
```
muN = inv(t(matrix_x)%*%matrix_x + omega0)%*%(t(matrix_x)%*%matrix_x%*%betahat + omega0%*%mu0)
omegaN = t(matrix_x)%*%matrix_x + omega0
vN = v0 + 366
vNsigmaN2 = v0*sigmasq0 + (t(matrix_y)%*%matrix_y + t(mu0)%*%omega0%*%mu0 - t(muN)%*%omegaN%*%muN)

randomSigma2 <- rinvchisq(n = 1000, df = vN, scale = (vNsigmaN2/vN))
randomBetas <- c()
plot(TempLinkoping, col="black")
sigmas = data.frame(randomSigma2)
y_df = data.frame(matrix(1, 1000, 1))
betas0 = c()
betas1 = c()
betas2 = c()
sigma2s = c()
```

```
ibeta = 0
for(singleSigma in randomSigma2) {
  randomBeta <- rmvt(n = 1,mu = t(muN), S = singleSigma*inv(omegaN))
  ibeta = ibeta + 1
  ys = c()
  for (i in 1:1000) {
    y = randomBeta[1, 1] + randomBeta[1, 2]*i/1000 + randomBeta[1, 3]*(i/1000)^2
    ys = c(ys, y)
  }
  betas0 = c(betas0, randomBeta[1, 1])
  betas1 = c(betas1, randomBeta[1, 2])
  betas2 = c(betas2, randomBeta[1, 3])
  sigma2s = c(sigma2s, singleSigma)
  y_df[paste0("trial", ibeta)] <- data.frame(ys)
}
xs = c()
for(i in 1:1000) {
  xs = c(xs, i/1000)
}
y_df = subset(y_df, select = -c(1) )

mediany = matrix(1, 1000, 1)
lowery = matrix(1, 1000, 1)
uppery = matrix(1, 1000, 1)
for (row in 1:nrow(y_df)) {
  mediany[row] = median(as.numeric(as.vector(y_df[row, ])))
  lowery[row] = quantile(x = as.numeric(as.vector(y_df[row, ])), probs = 0.025)
  uppery[row] = quantile(x = as.numeric(as.vector(y_df[row, ])), probs = 0.975)
}
plot(TempLinkoping, col="black")
lines(xs, mediany, col="blue")
lines(xs, lowery, col="red")
lines(xs, uppery, col="red")
```



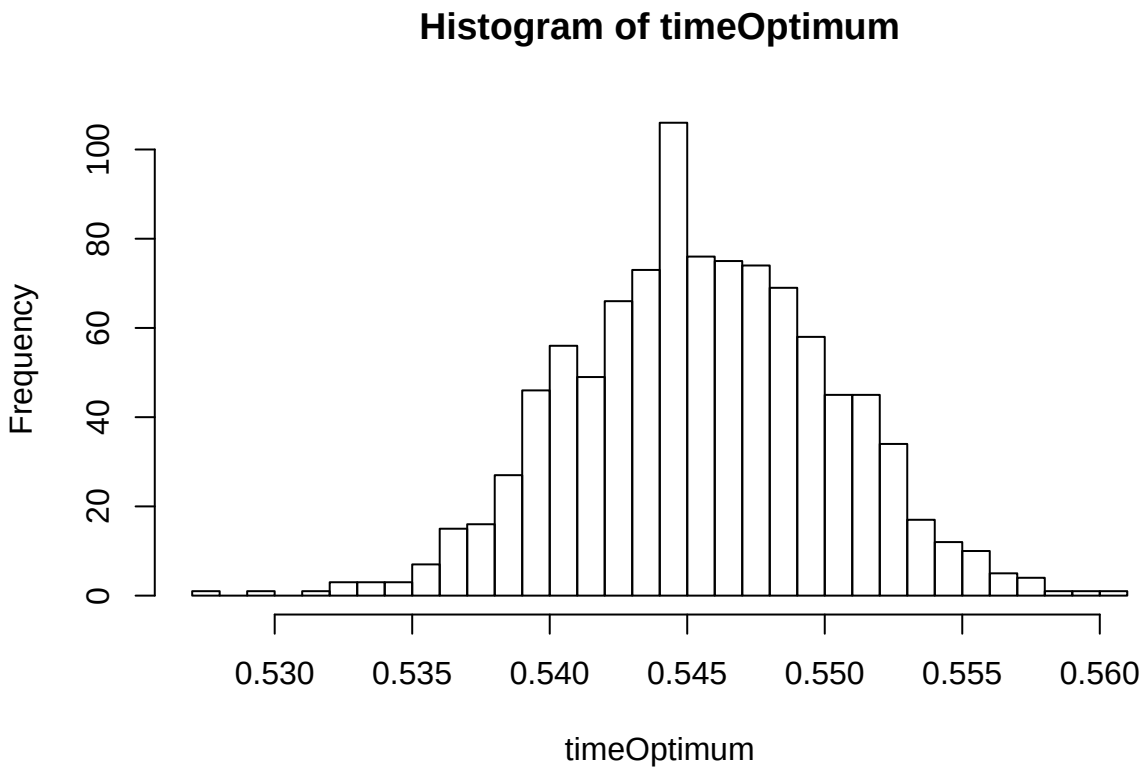
The interval band does not contain all the datapoints since it is a 95% approximation of the regression model and not the data points.

```
attach(mtcars)
par(mfrow=c(1,4))
hist(betas0, nclass=30, col='blue')
hist(betas1, nclass=30, col='red')
hist(betas2, nclass=30, col='green')
hist(sigma2s, nclass=30, col='yellow')
```


Histogram of betas0
Frequency
0
20
40
60
80
100
120
−12
−10
−8
betas0
Histogram of betas1
Frequency
0
50
100
150
85
95
betas1
Histogram of betas2
Frequency
0
20
40
60
80
−95
−85
−75
betas2
Histogram of sigma2s
Frequency
0
10
20
30
40
50
60
70
14
18
sigma2s

(c)

```
timeOptimum = -betas1/(2*betas2)
hist(timeOptimum, nclass=30)
```



(d)

If there is a higher order but we are more certain that higher order parameters are not needed we can set the omega-values high as it creates a stronger prior and the mu-prior values at 0. Which would mitigate the problem of an unneccesary high order polynom.

Task 2

(a)

```
womenWork<-read.table("WomenWork.dat",header=TRUE)
glmModel <- glm(Work ~ 0 + ., data = womenWork, family = binomial)
print(glmModel)
```

```
##
## Call:  glm(formula = Work ~ 0 + ., family = binomial, data = womenWork)
##
## Coefficients:
##    Constant   HusbandInc    EducYears     ExpYears    ExpYears2
##     0.64430     -0.01977      0.17988      0.16751     -0.14436
##         Age  NSmallChild    NBigChild
##    -0.08234     -1.36250     -0.02543
##
## Degrees of Freedom: 200 Total (i.e. Null);  192 Residual
## Null Deviance:       277.3
## Residual Deviance: 222.7     AIC: 238.7
```

(b)

```
chooseCov <- c(1:8) # Here we choose which covariates to include in the model
tau <- 10; # Prior scaling factor such that Prior Covariance = (tau^2)*I

# install.packages("mvtnorm") # Loading a package that contains the multivariate normal pdf
library("mvtnorm") # This command reads the mvtnorm package into R's memory. NOW we can use dmvnorm fun

# Loading data from file
Data<-read.table("WomenWork.dat",header=TRUE)  # Spam data from? Hastie et al.
y <- as.vector(womenWork[,1]); # Data from the read.table function is a data frame. Let's convert y and
X <- as.matrix(womenWork[,2:9]);
covNames <- names(womenWork)[2:length(names(womenWork))];
X <- X[,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);

LogPostLogistic <- function(betaVect,y,X,mu,Sigma){
  nPara <- length(betaVect);
  linPred <- X%*%betaVect;

  logLik <- sum( linPred*y -log(1 + exp(linPred)));
  if (abs(logLik) == Inf) logLik = -20000; # Likelihood is not finite, stear the optimizer away from he
  logPrior <- dmvnorm(betaVect, matrix(0,nPara,1), Sigma, log=TRUE);
  return(logLik + logPrior)
}
```

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

```
# Printing the results to the screen
names(OptimResults$par) <- covNames # Naming the coefficient by covariates
approxPostStd <- sqrt(diag(-solve(OptimResults$hessian))) # Computing approximate standard deviations.
approxPostStd <- sqrt(diag(-solve(OptimResults$hessian)))
names(approxPostStd) <- covNames # Naming the coefficient by covariates

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

```
##    Constant  HusbandInc   EducYears    ExpYears   ExpYears2         Age
##  0.62672884 -0.01979113  0.18021897  0.16756670 -0.14459669 -0.08206561
## NSmallChild   NBigChild
## -1.35913317 -0.02468351
```

```
# Jacobiany beta:
print(approxPostStd)
```

```
##    Constant  HusbandInc   EducYears    ExpYears   ExpYears2         Age
##  1.50533138  0.01589983  0.07885556  0.06596754  0.23575129  0.02680412
## NSmallChild   NBigChild
##  0.38892439  0.14132327
```

```
#Intervall NSmallChild
upperb = betatilde["NSmallChild"] + 1.96*approxPostStd["NSmallChild"]
lowerb = betatilde["NSmallChild"] - 1.96*approxPostStd["NSmallChild"]
print(upperb)
```

```
## NSmallChild
##  -0.5968414
```

```
print(lowerb)
```

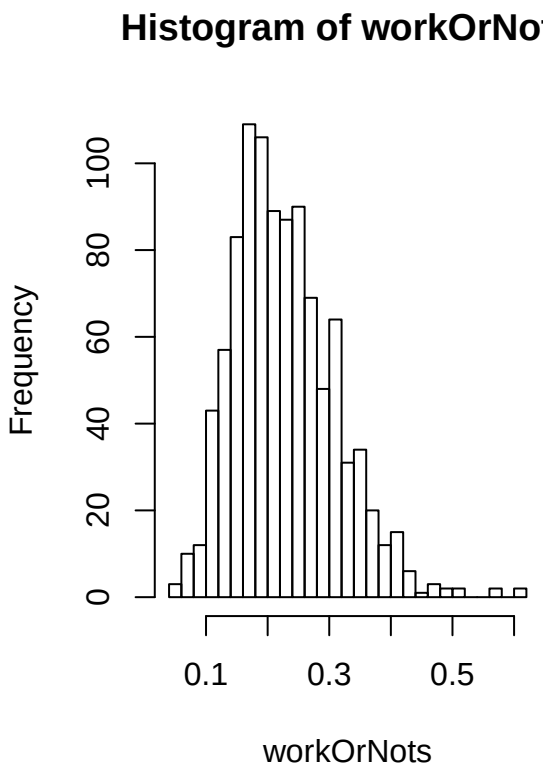
```
## NSmallChild
##   -2.121425
```

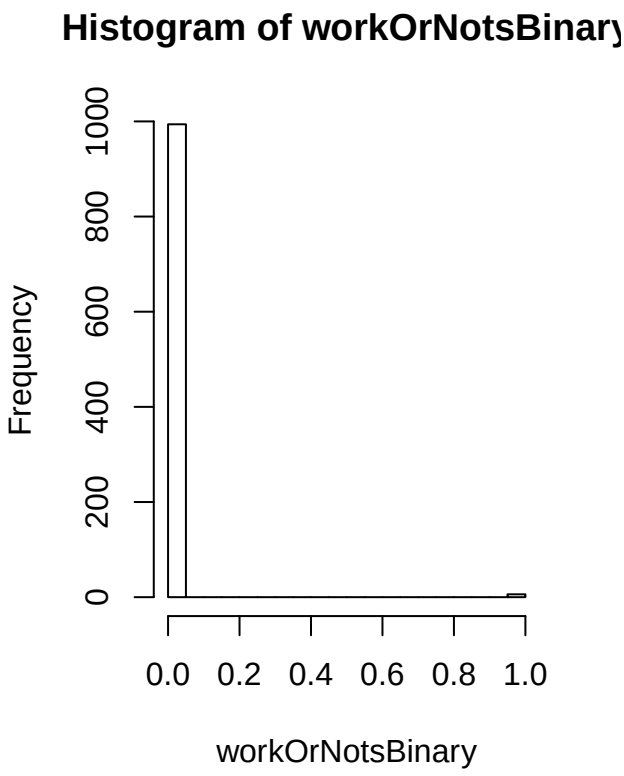
(c)

```
covarMatrix = -inv(OptimResults$hessian)

ladyInput = c(1, 10, 8, 10, (10/10)^2, 40, 1, 1)
workOrNots = c()
workOrNotsBinary = c()
betas = rmvt(n = 1000,mu = matrix(betatilde), S = covarMatrix)
for(row in 1:nrow(betas)) {
  working = exp(ladyInput %*% betas[row, ])/(1 + exp(ladyInput %*% betas[row, ]))
  workingBinary = round(working)
  workOrNots = c(workOrNots, working)
  workOrNotsBinary = c(workOrNotsBinary, workingBinary)
}
attach(mtcars)
```

```
## The following objects are masked from mtcars (pos = 3):
##
##     am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt
```

```
par(mfrow=c(1,2))
hist(workOrNots, n=30)
hist(workOrNotsBinary, n=30)
```





Lab3

Elon Brange, Ludwig Thaung

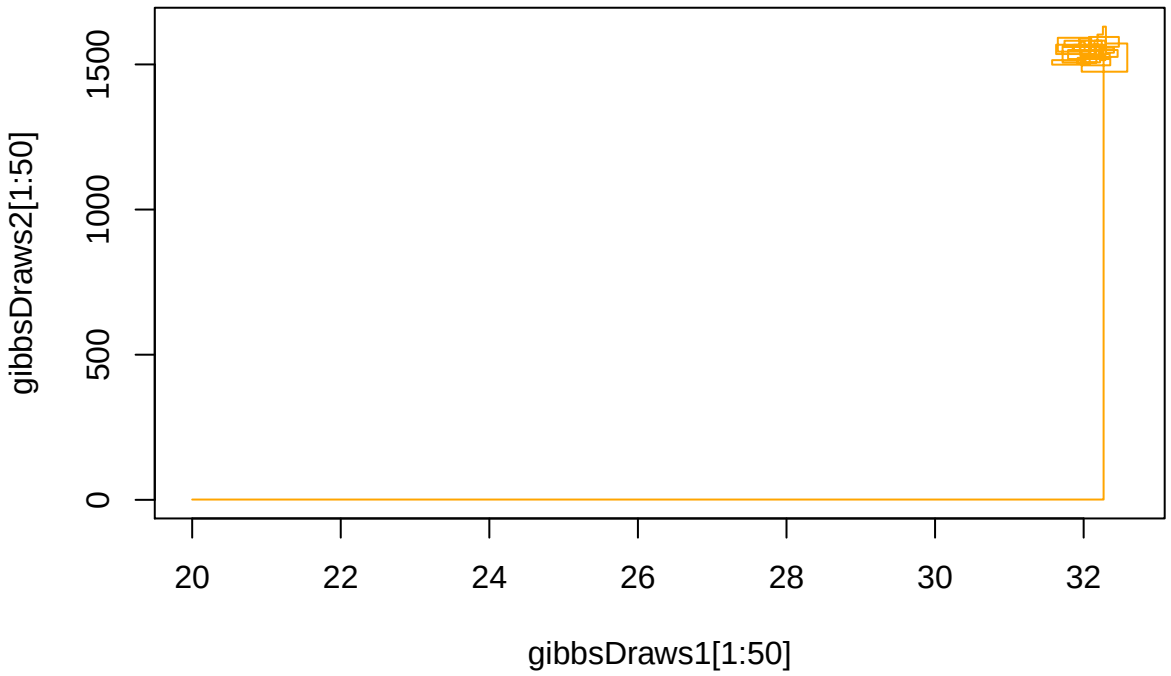
5/19/2019

Task 1

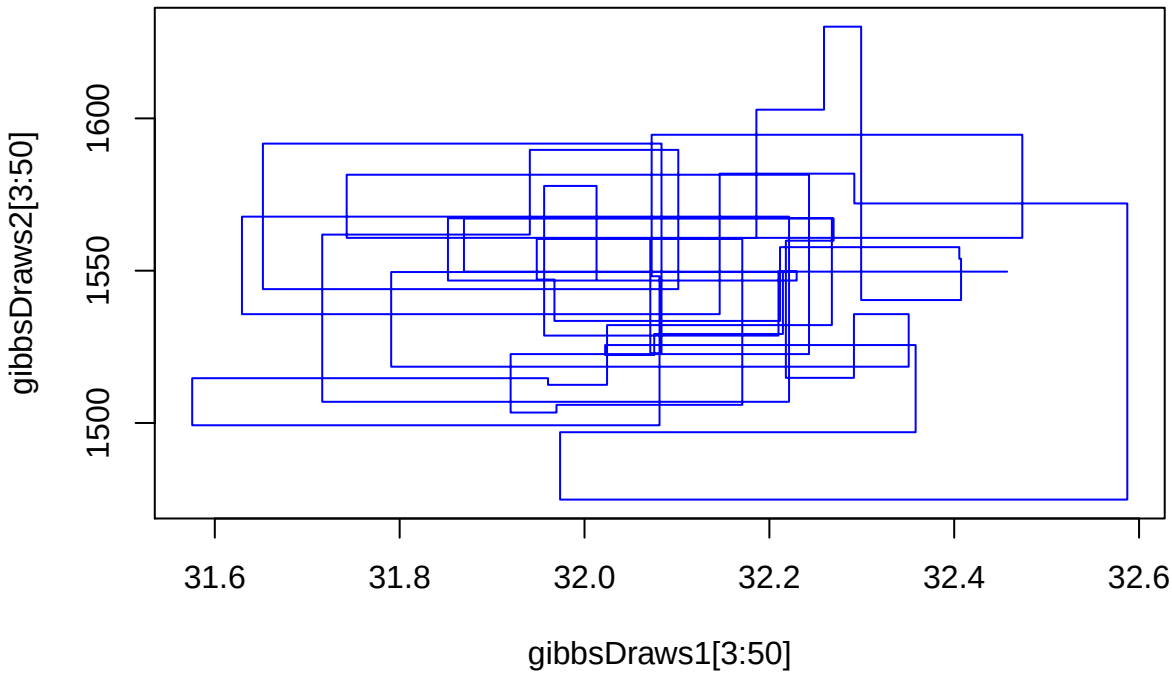
(a)

```
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')
```

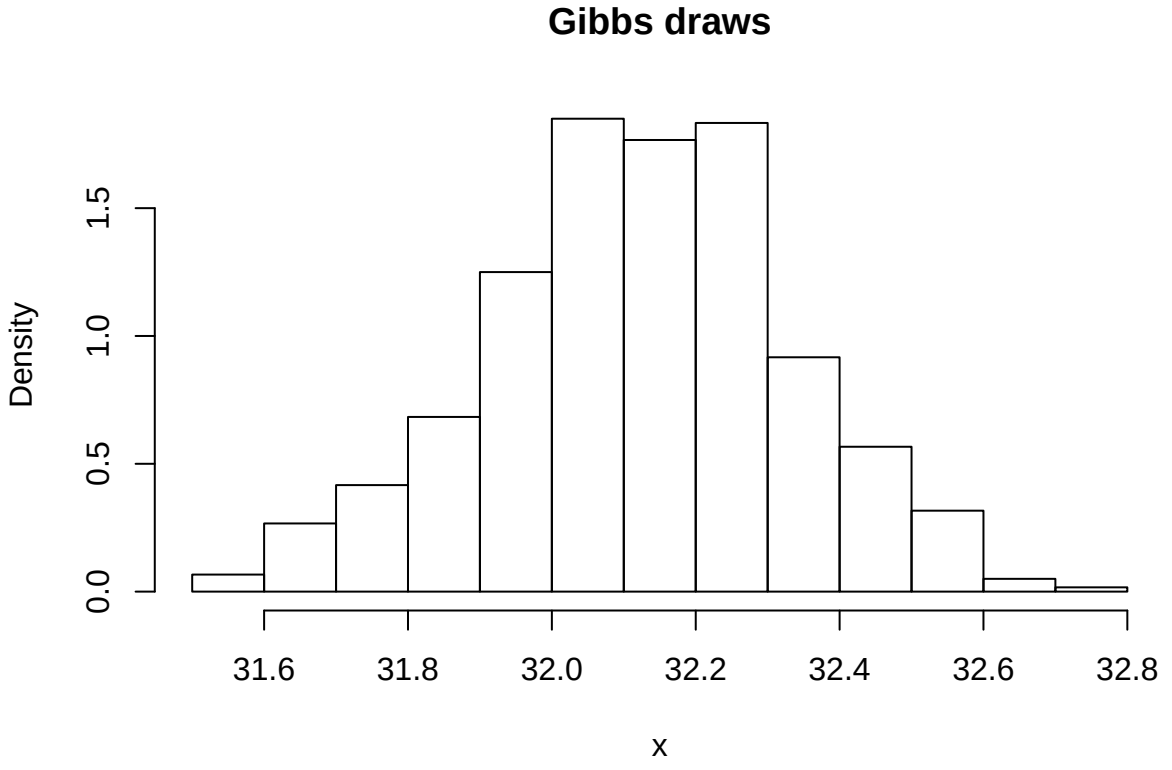
```
#lines(1:nDraws, matrix(averageRain, (nDraws-500)/10, 1))

#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')
```




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

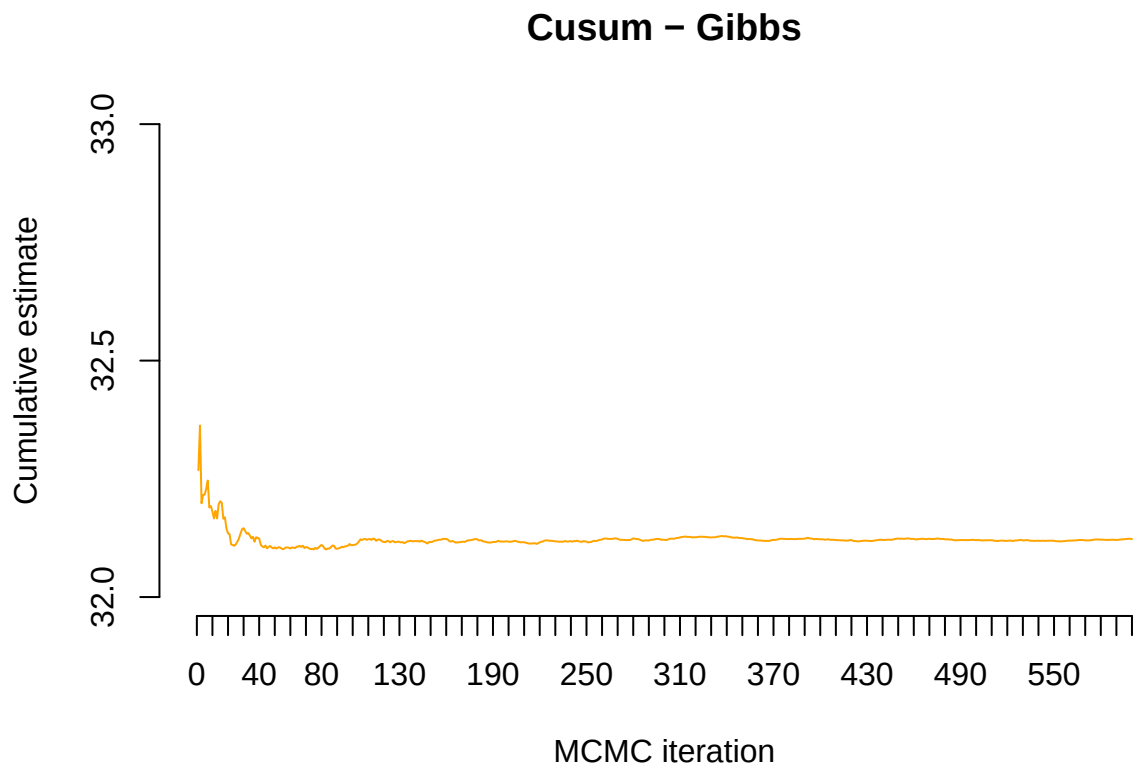



```
hist(gibbsDraws[,1], freq = FALSE, main='Gibbs draws', xlab='x')
#hist(gibbsDraws[,1], freq = FALSE, main='Gibbs draws', ylim = c(0,0.5), xlab='x')

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

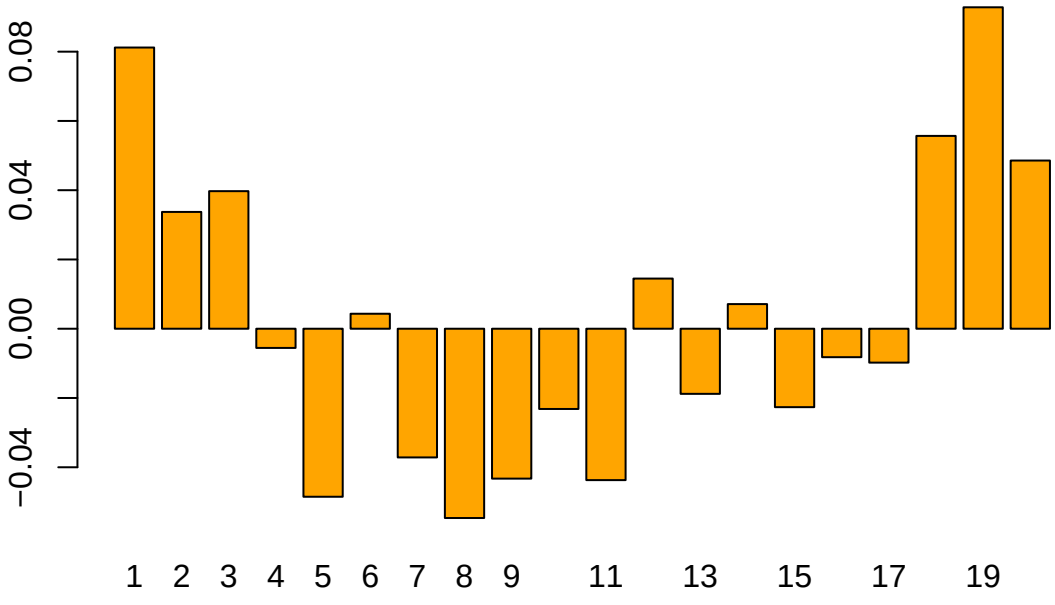



```
cusumData =  cumsum(gibbsDraws[,1])/seq(1,nDraws)
minY = floor(min(cusumData))
```

```
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))
```



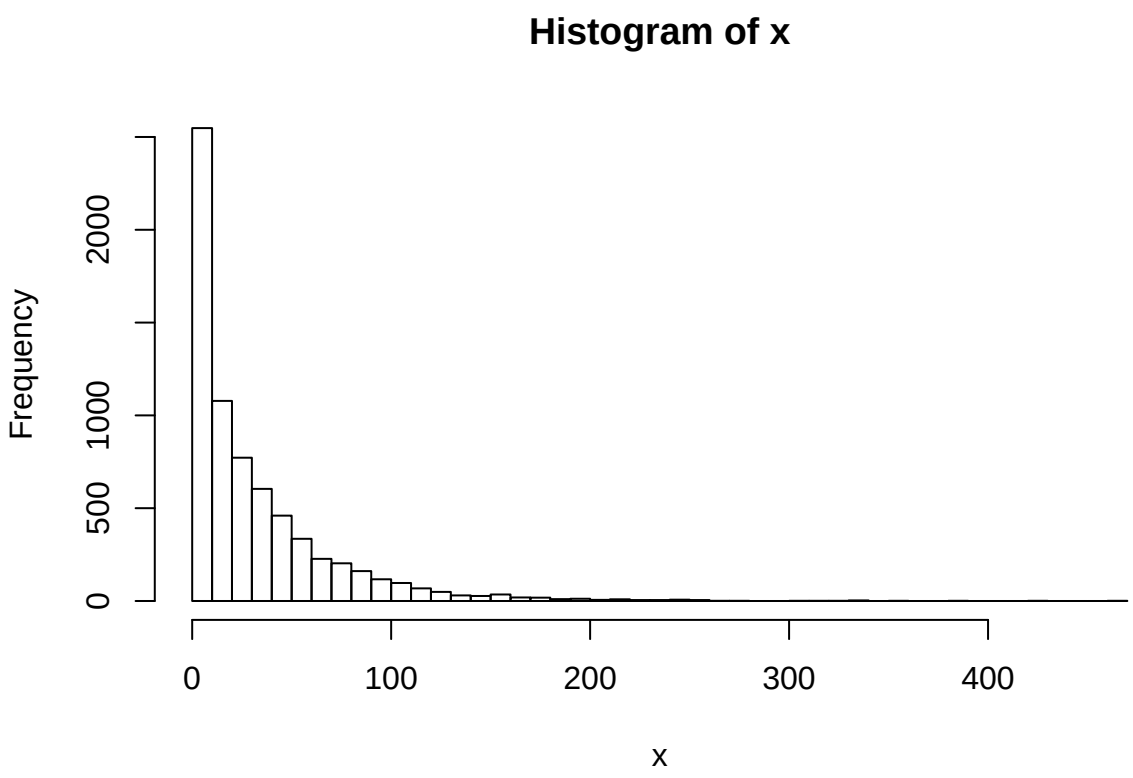

```
a = acf(gibbsDraws[,1], main='Gibbs draws', lag.max = 20, plot = F)
barplot(height = a$acf[-1], names.arg=seq(1,20), col = 'orange')
```



(b)

```
# 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))
```



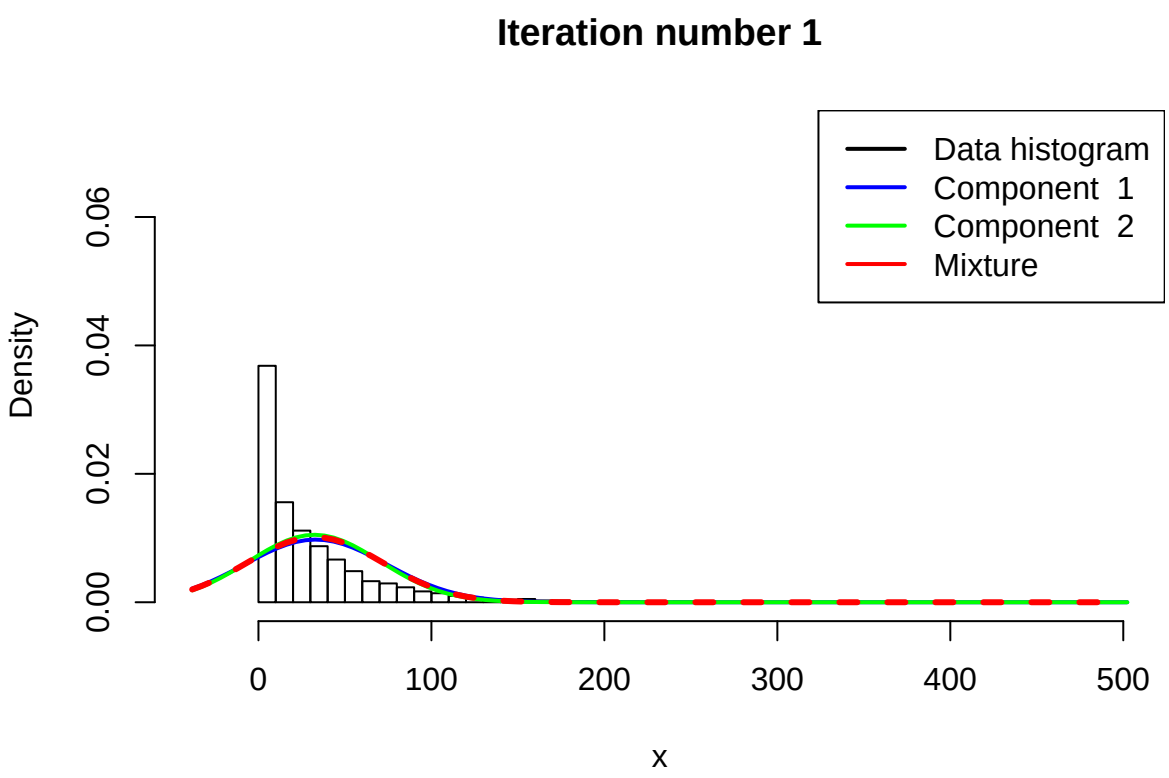

```
for (k in 1:nIter){
  message(paste('Iteration number:',k))
  alloc <- S2alloc(S) # Just a function that converts between different representations of the group al
  nAlloc <- colSums(S)
  print(nAlloc)
  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)

  # 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
```

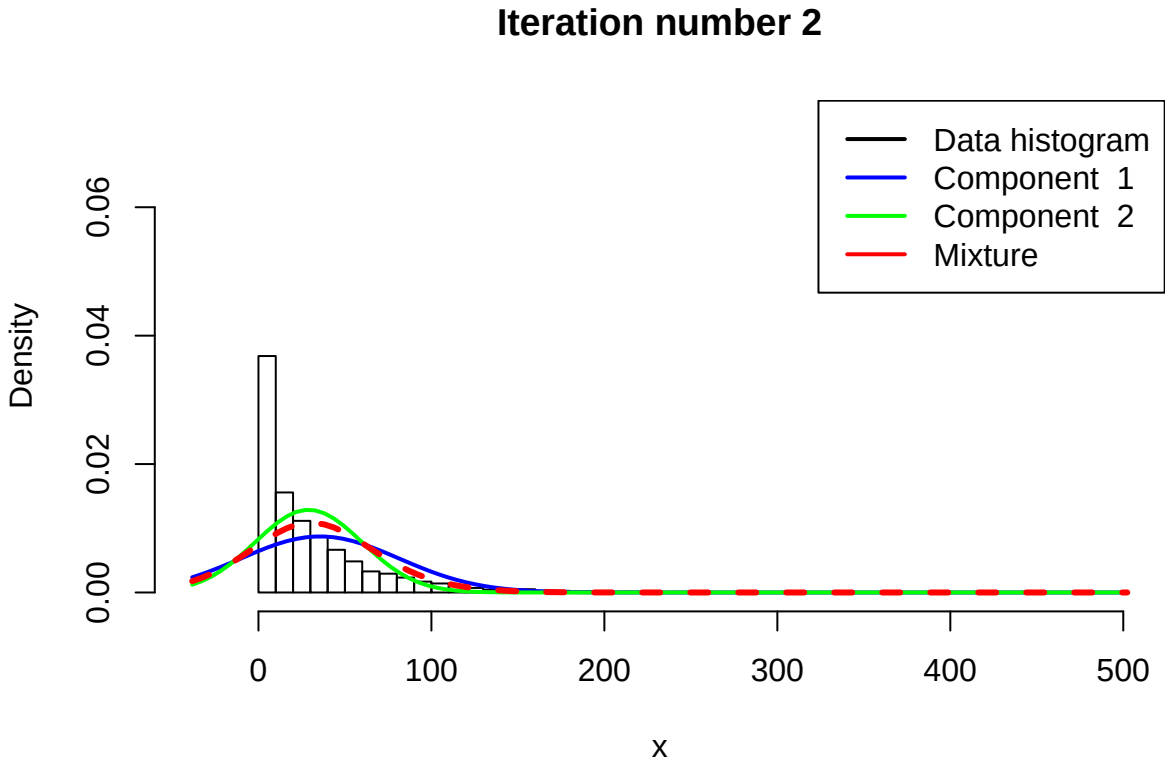
```
## [1] 3427 3492
```

```
## Iteration number: 2
```



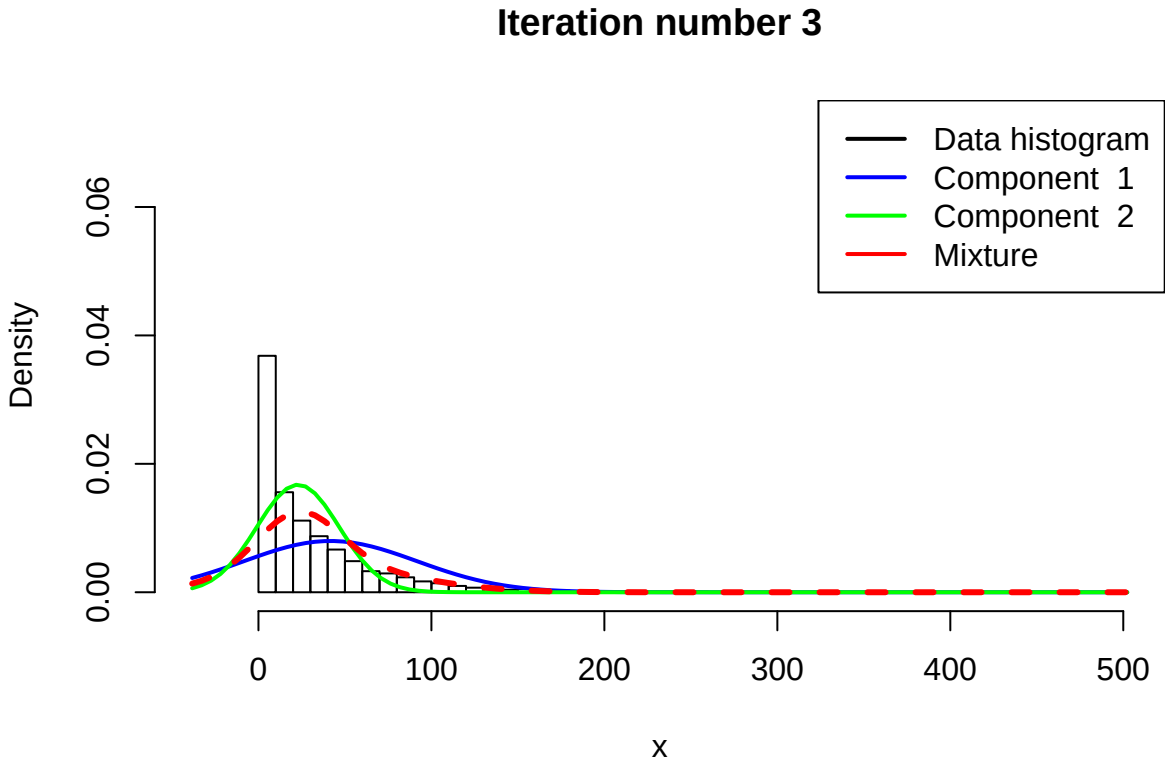

```
## [1] 3363 3556
```

```
## Iteration number: 3
```



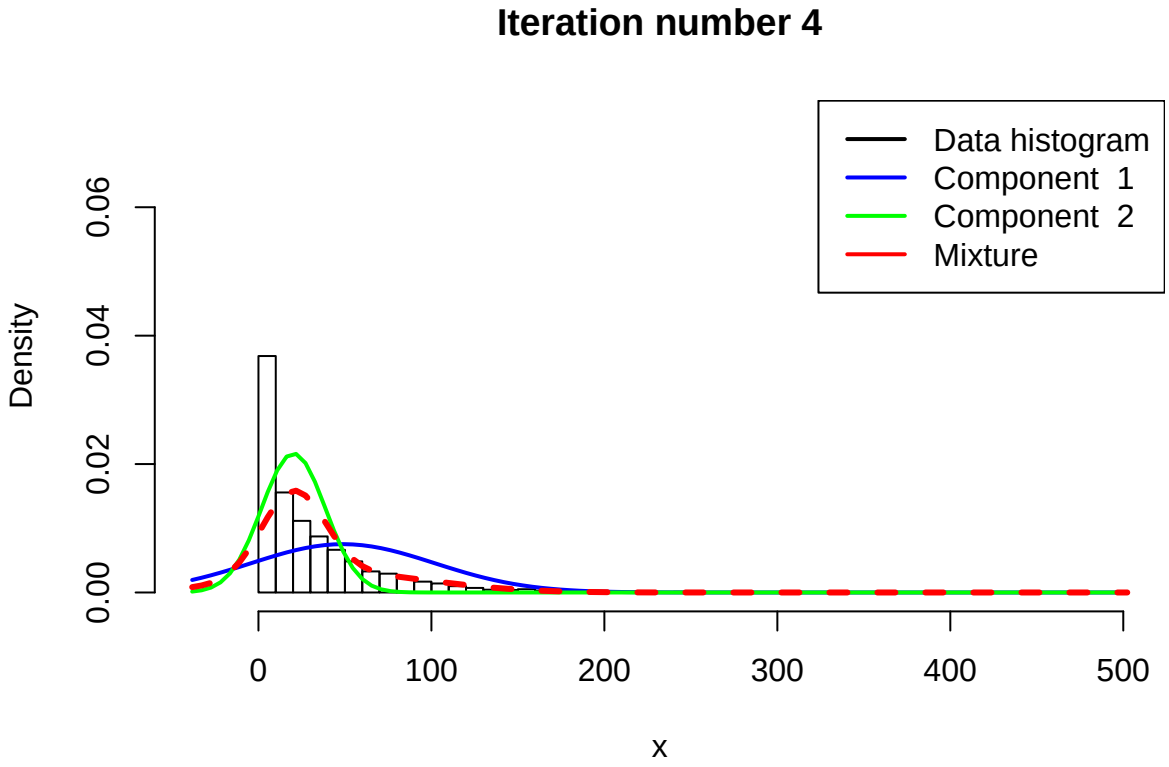

```
## [1] 3108 3811
```

```
## Iteration number: 4
```



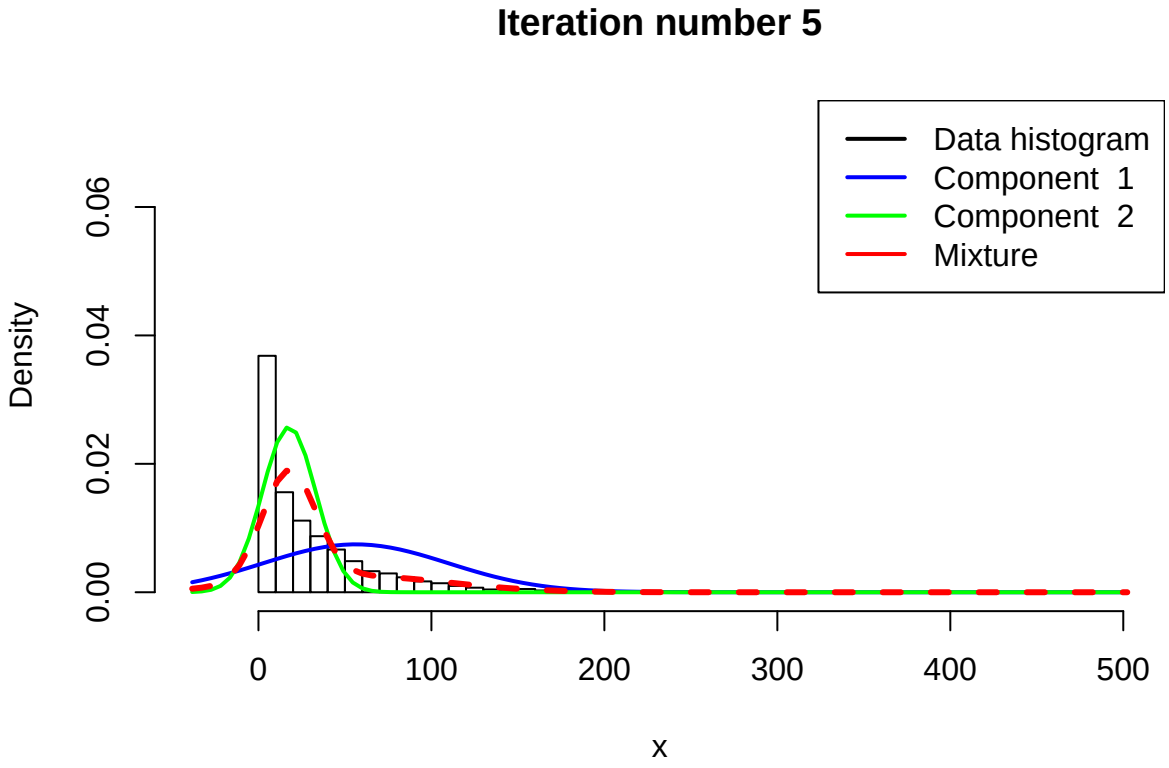

```
## [1] 2761 4158
```

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



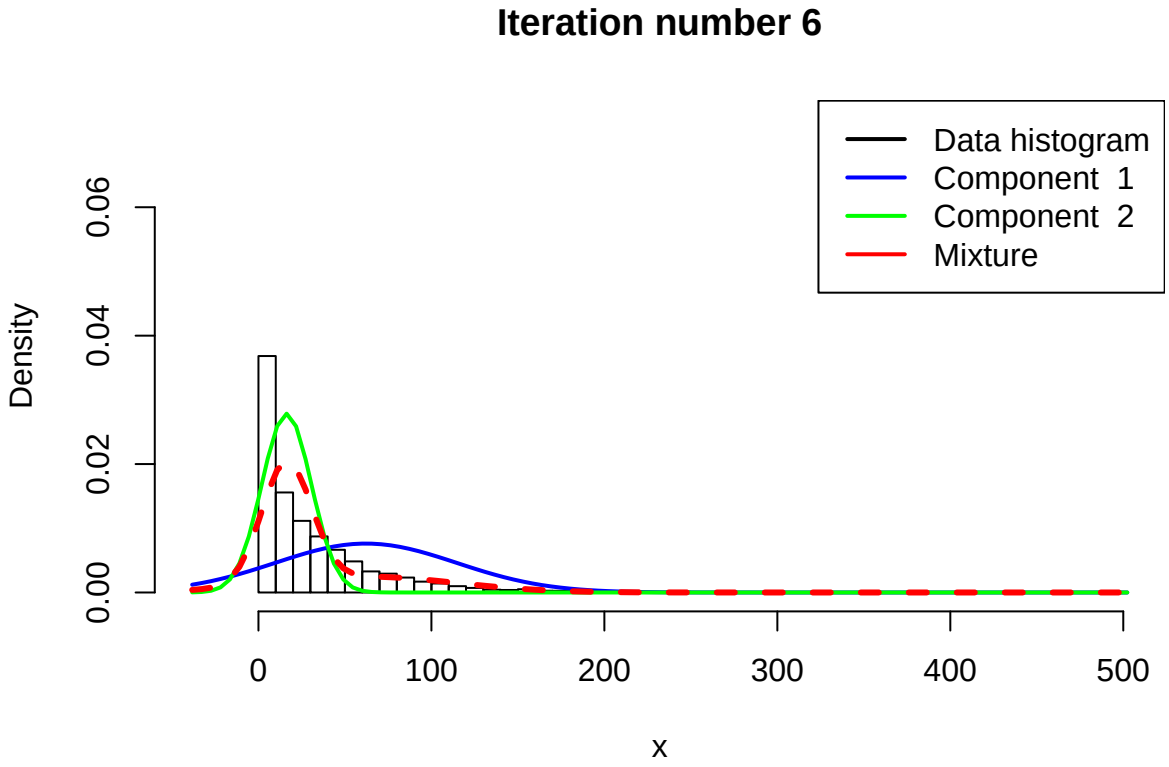

```
## [1] 2360 4559
```

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



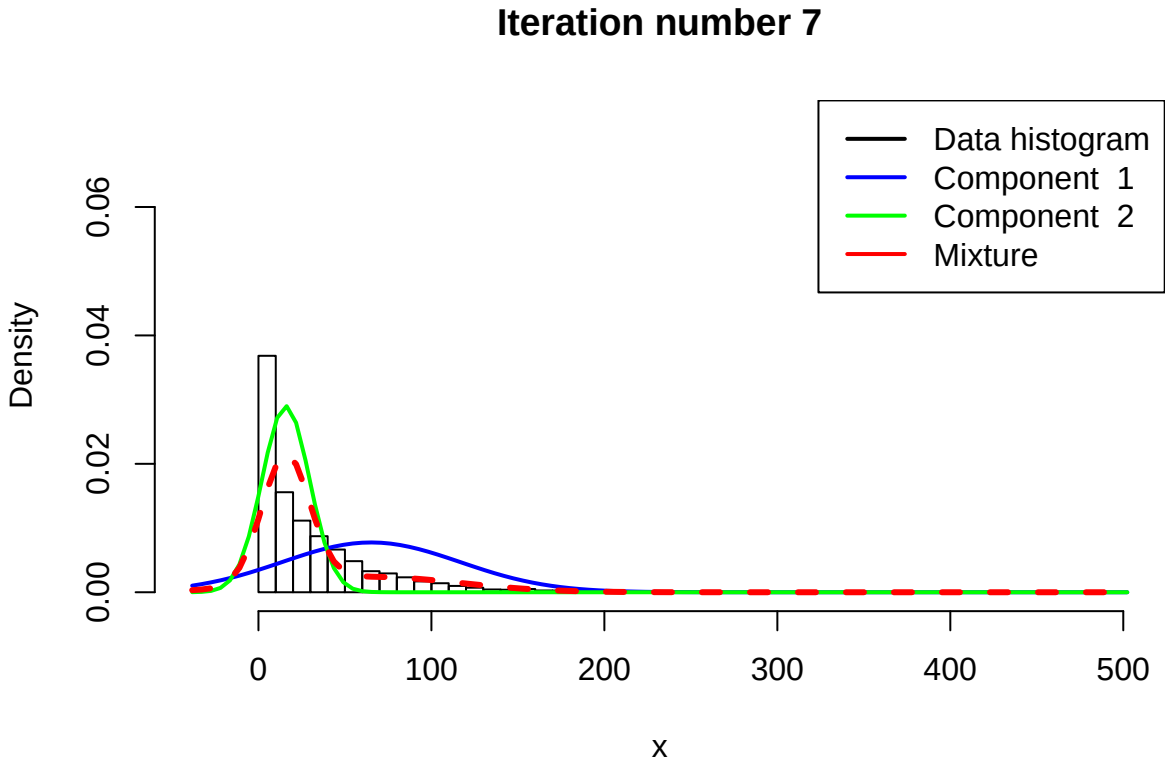

```
## [1] 2206 4713
```

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



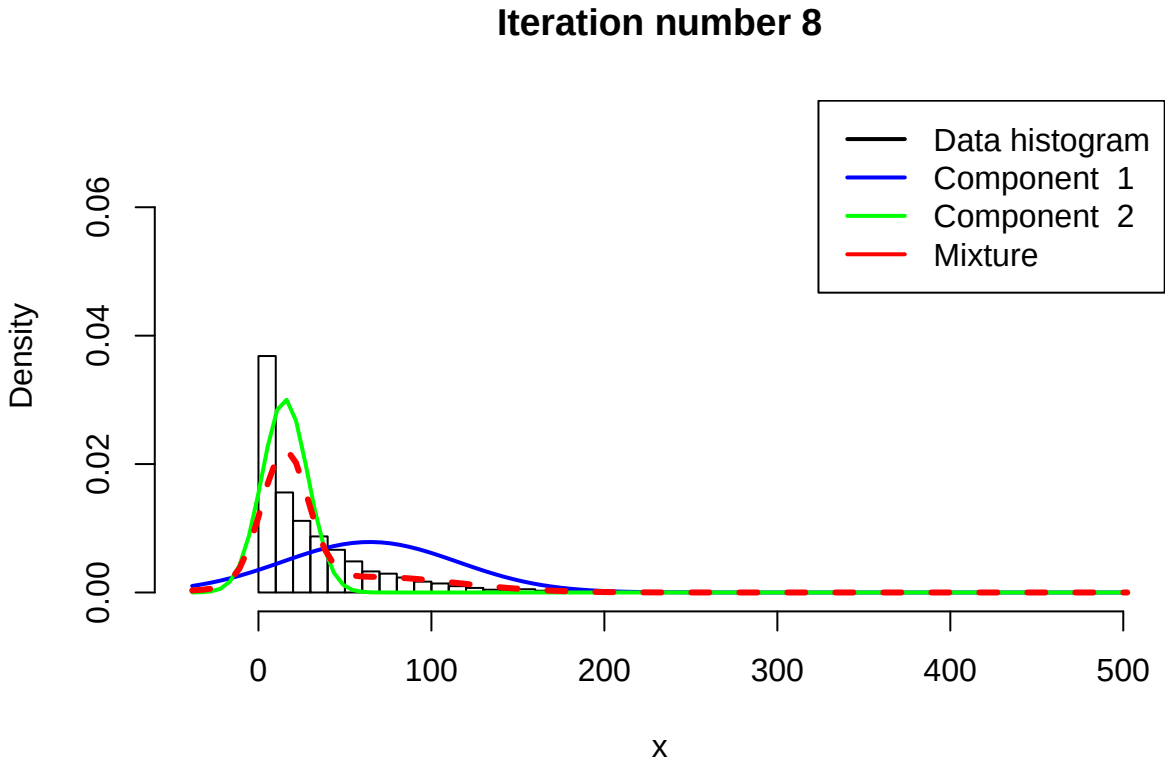

```
## [1] 2129 4790
```

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



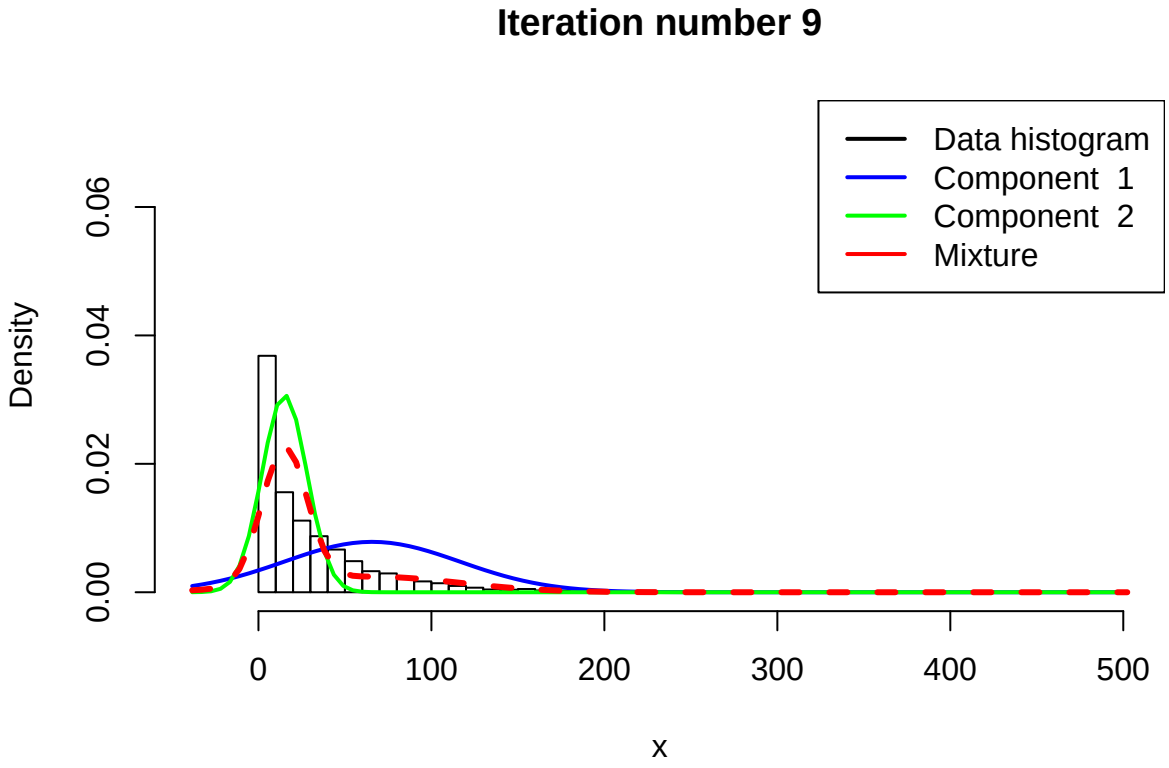

```
## [1] 2144 4775
```

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

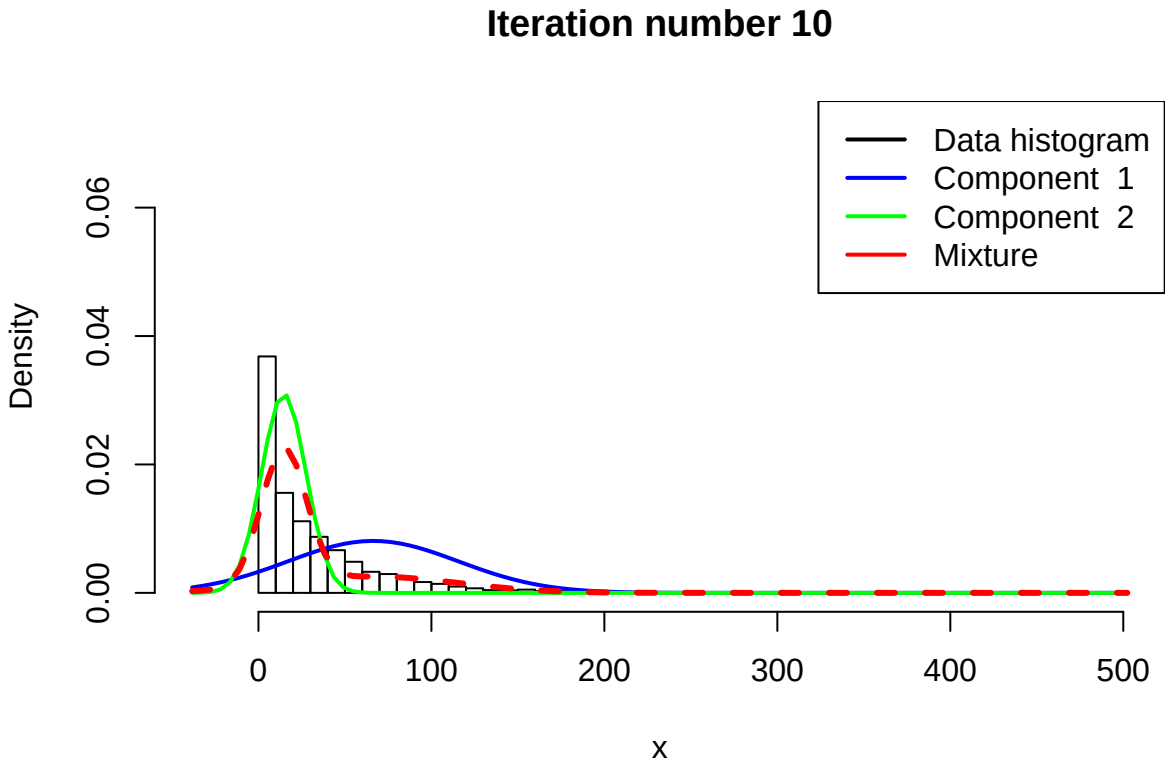



```
## [1] 2195 4724
```

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



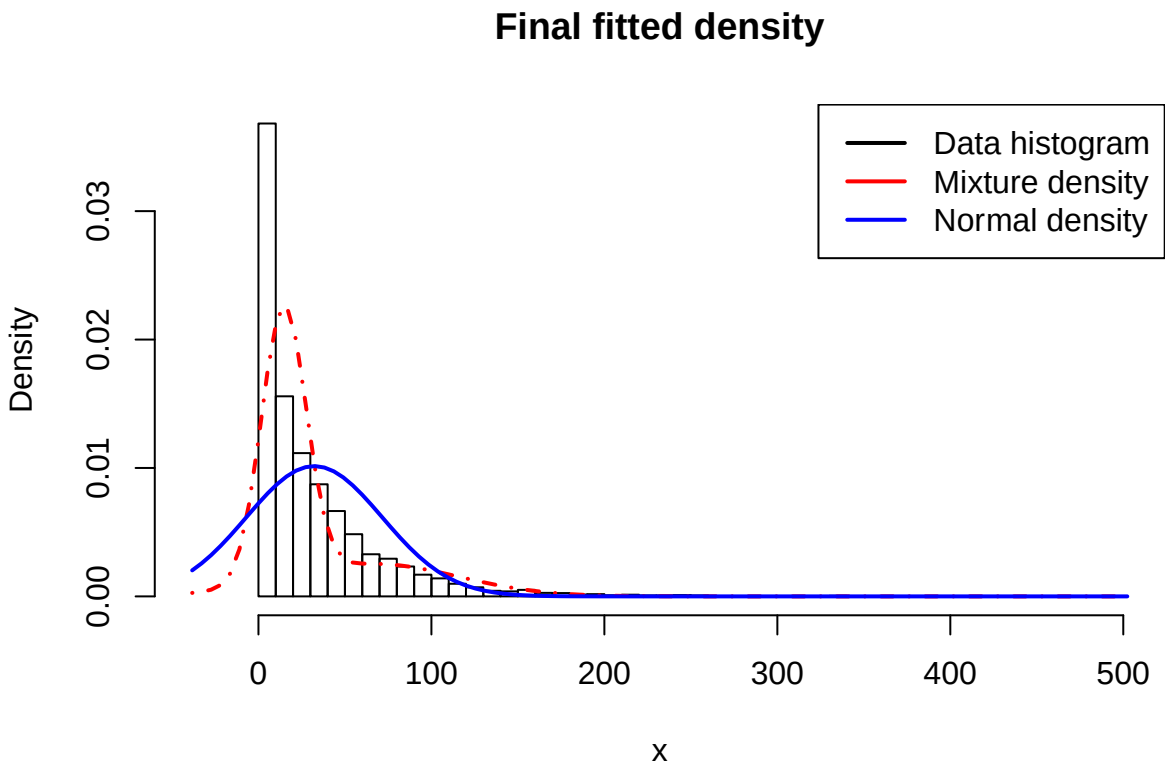

```
## [1] 2158 4761
```



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)

```
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)

```
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, family
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
```

```
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)

```
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: "
```

```
print(betatilde)
```

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

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

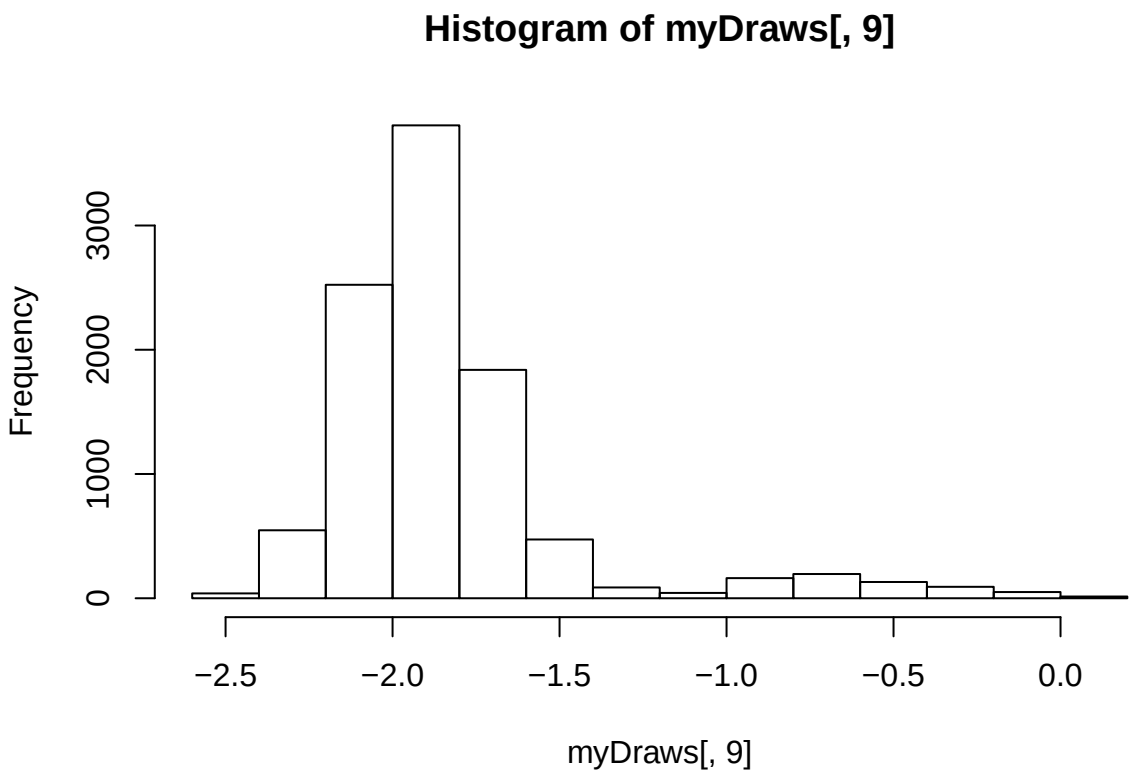
```
## [1] "Jacobiany beta: "
```

```
print(approxPostStd)
```

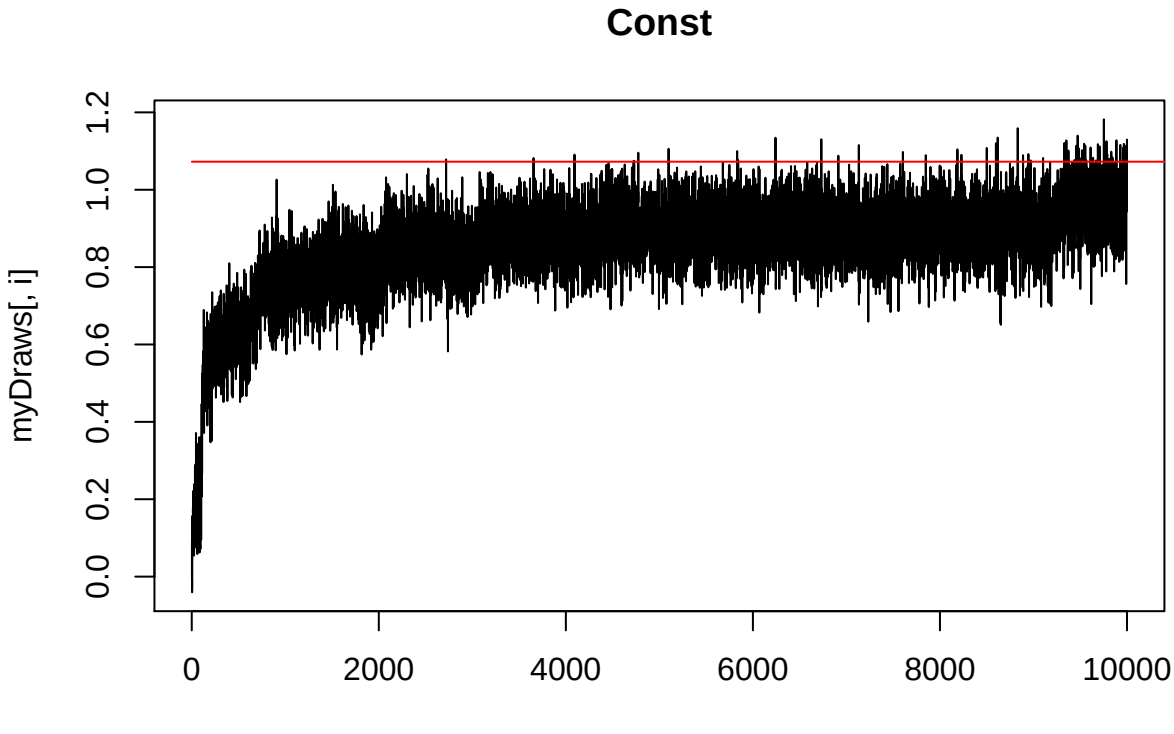
```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03077417  0.03678186  0.09242044  0.05056213  0.06019999  0.09143186
##     LargNeg     LogBook MinBidShare
##  0.05633101  0.02896411  0.07123677
```

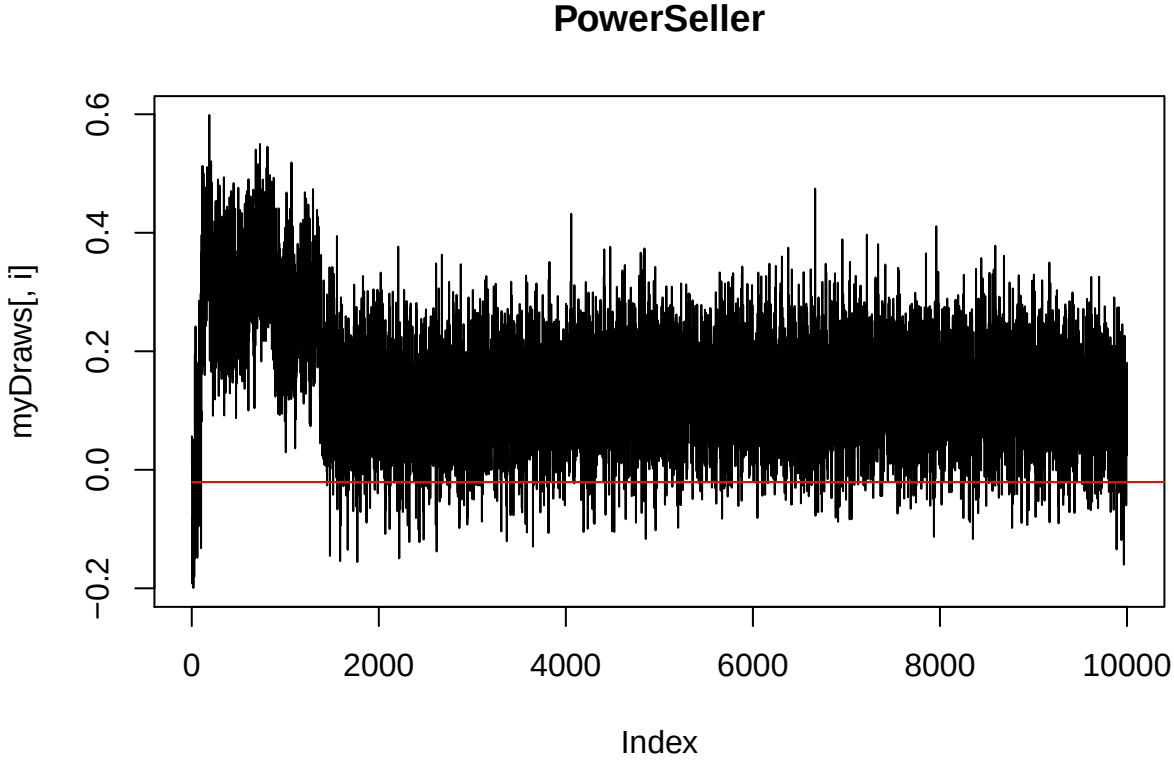
(c)

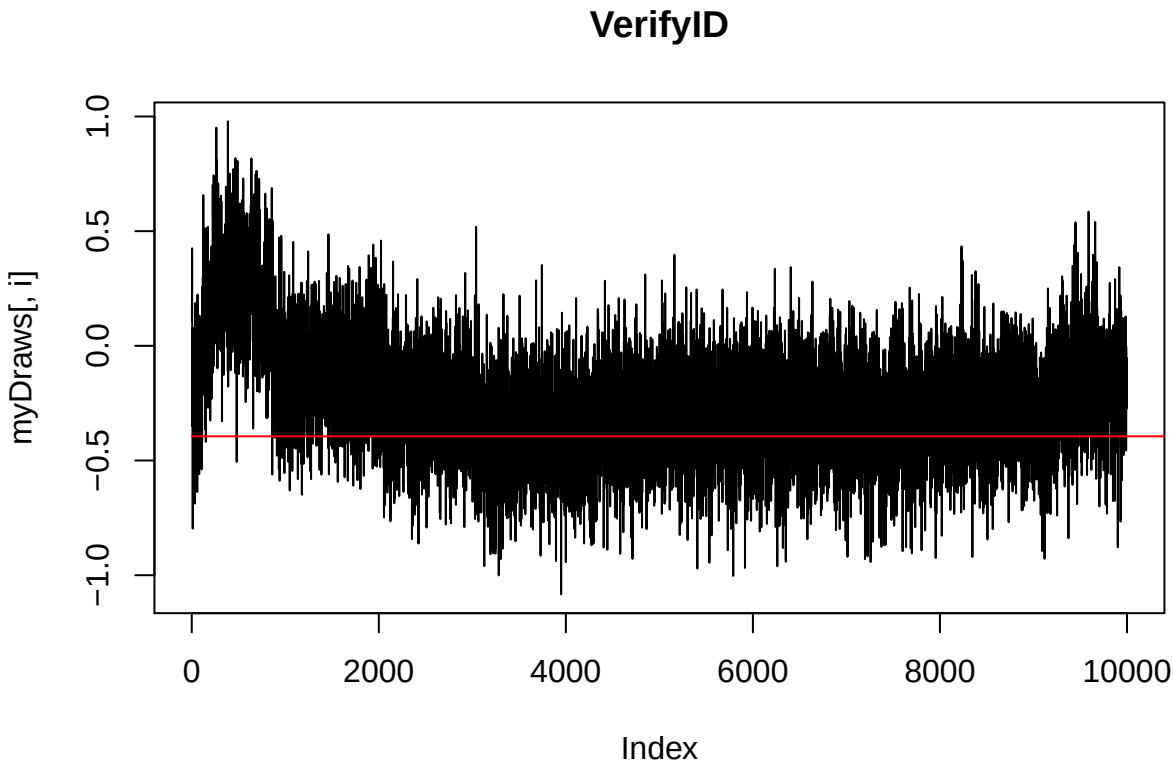
```
RWMSampler <- function(logPostFunc, n,  c, covar,  ...) {
  currentTheta = as.vector(rep(0, dim(covar)[1]))
  draws = matrix(0, nrow = n, ncol = dim(covar)[1])
  oldProbability = logPostFunc(currentTheta, ...)
  for(i in 1:n) {
    currentDraw <- rmvt(1, mu = currentTheta, S = c*covar)
    newProbability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

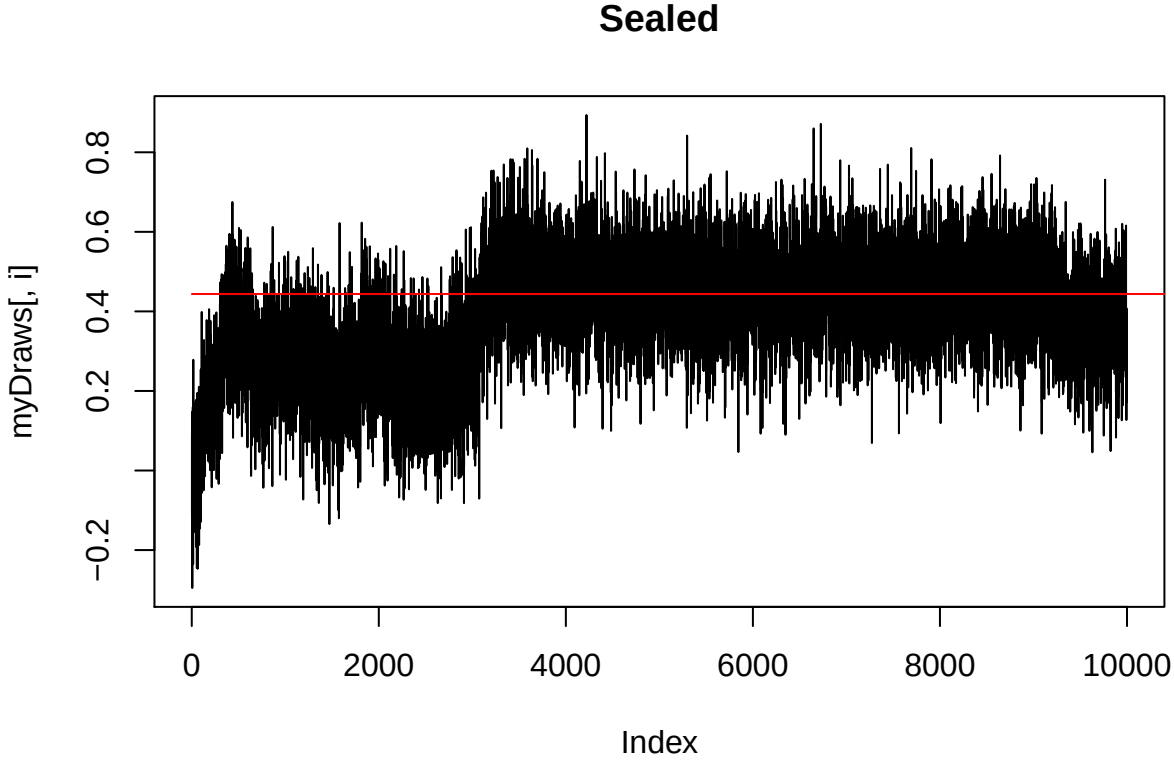


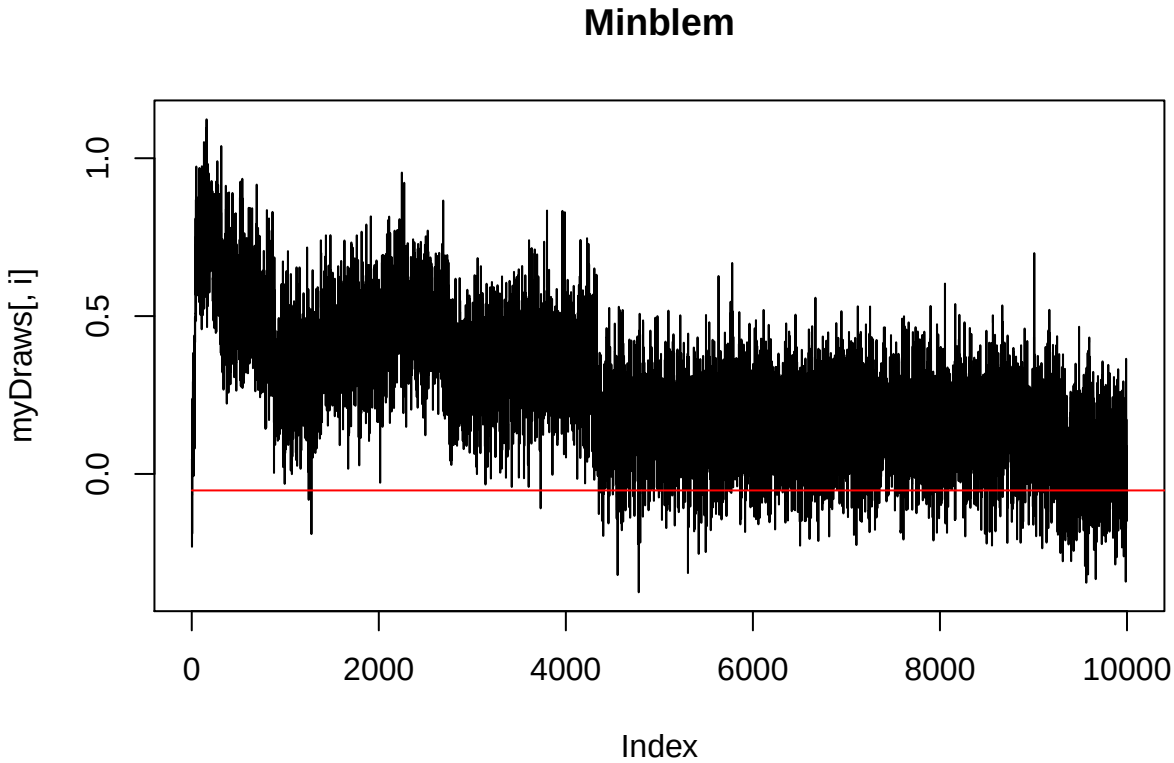

```
for(i in 1:9){
  plot(myDraws[,i], type='s')
  a = c(rep(betatilde[i],length(myDraws)))
  lines(a, col='red')
  title(covNames[i])
}
```

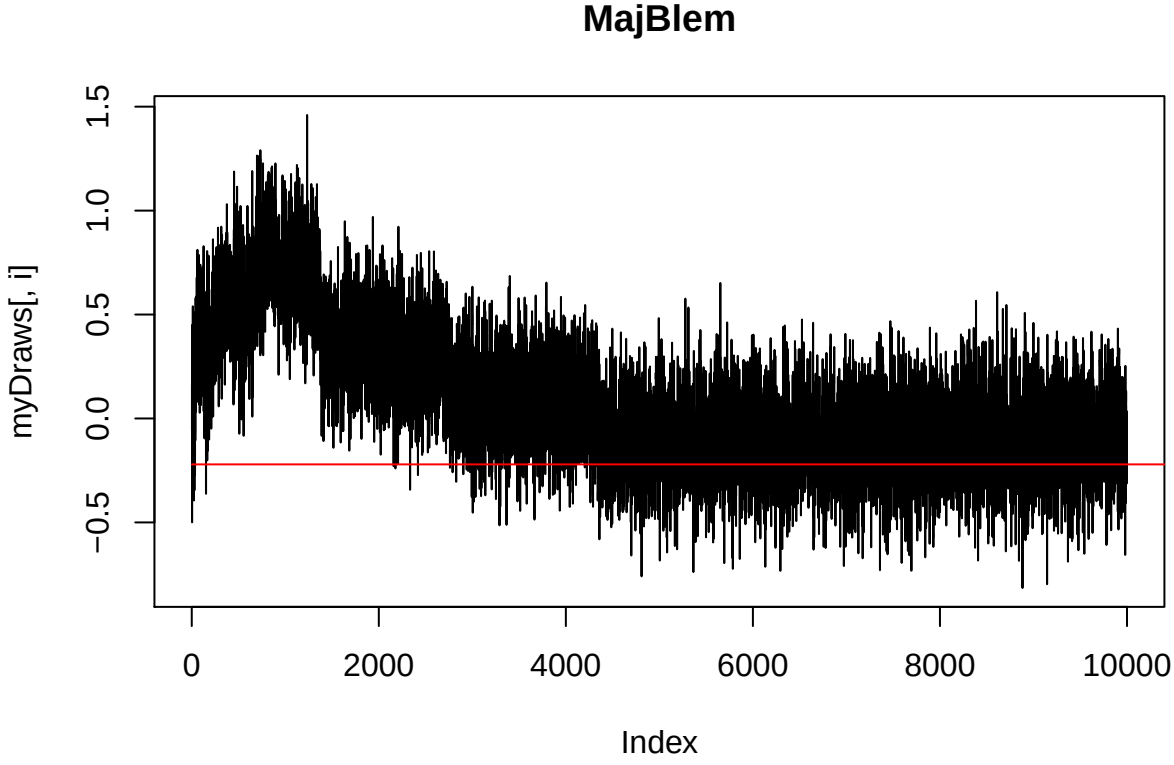

Const
myDraws[, i]
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2000
4000
6000
8000
10000
Index

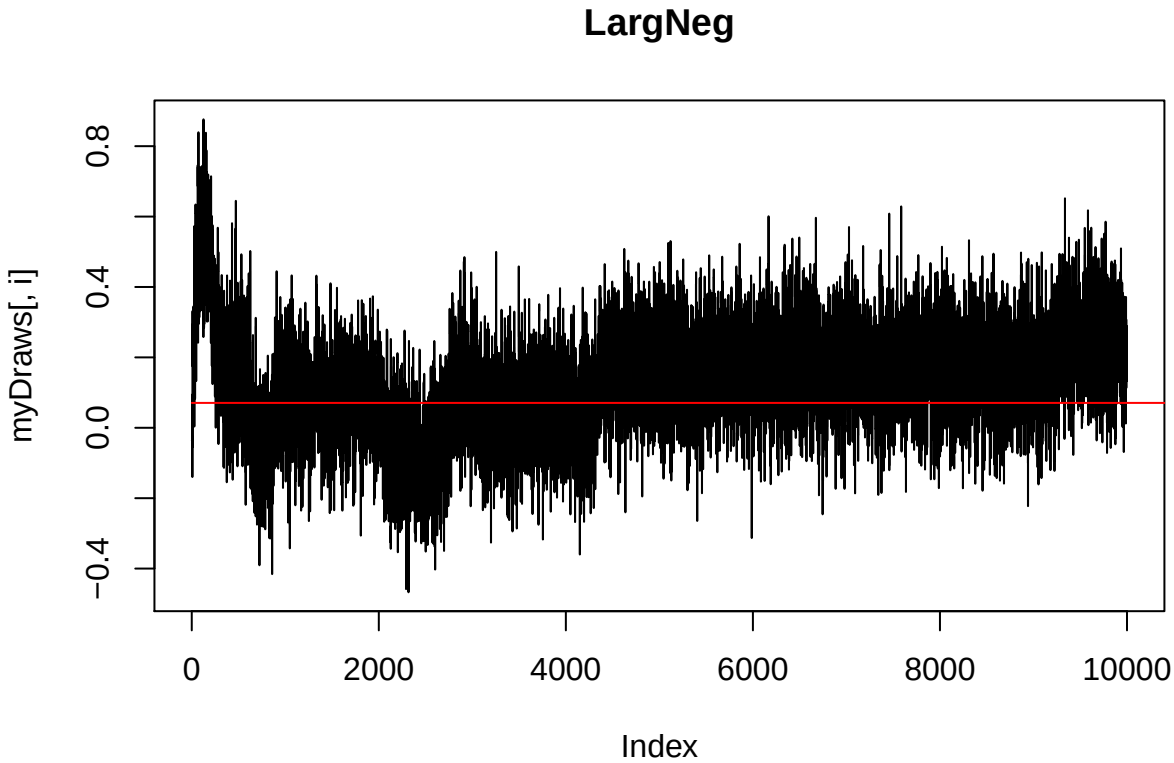

PowerSeller
myDraws[, i]
−0.2
0.0
0.2
0.4
0.6
0
2000
4000
6000
8000
10000
Index

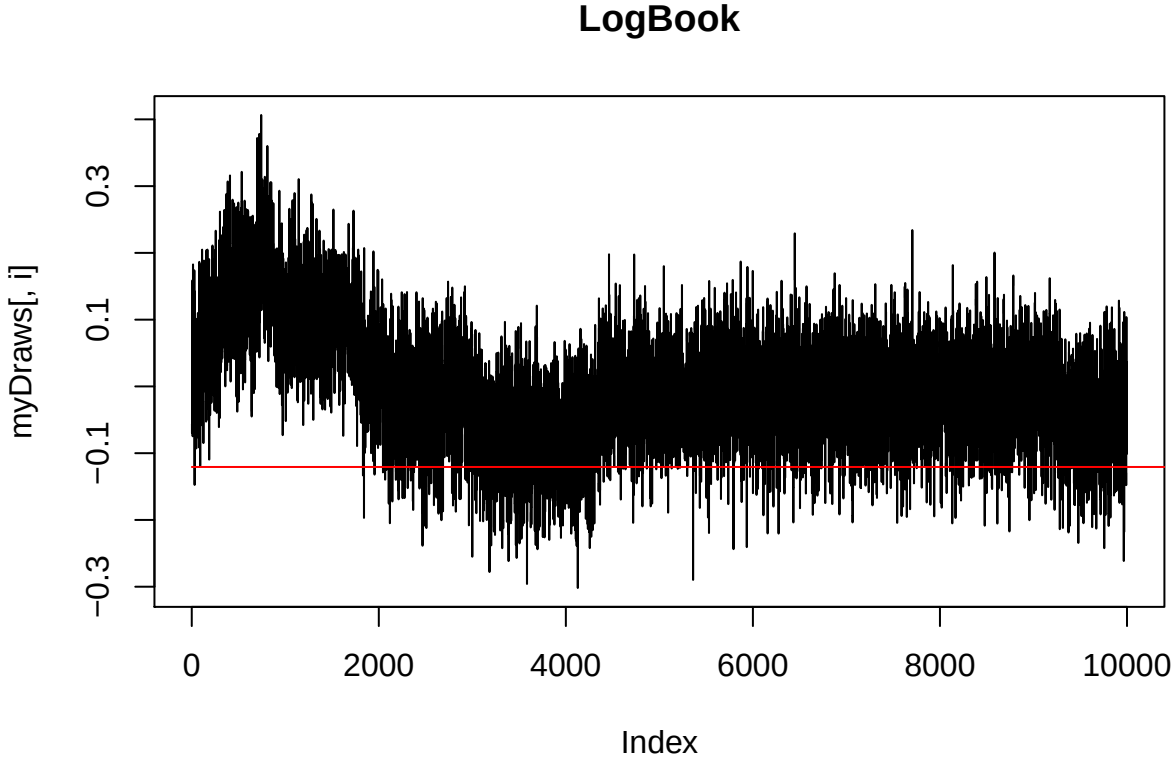

VerifyID
myDraws[, i]
1.0
0.5
0.0
−0.5
−1.0
0
2000
4000
6000
8000
10000
Index

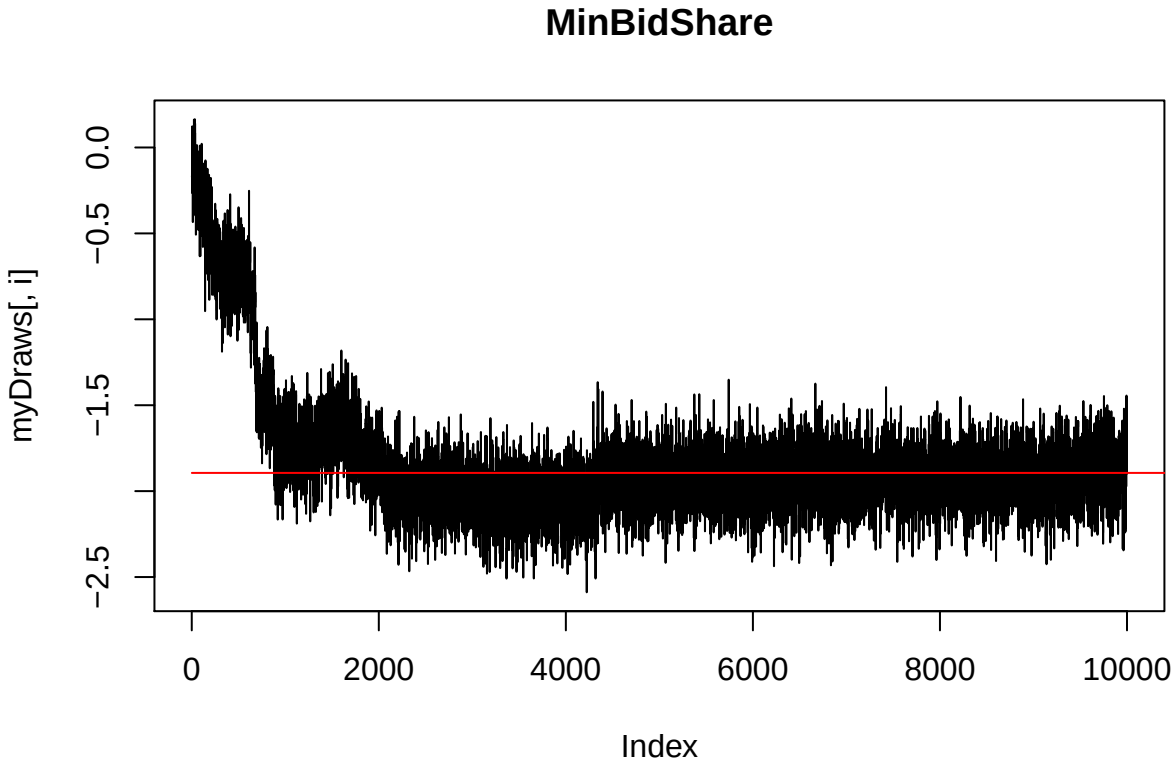

Sealed
myDraws[, i]
0.8
0.6
0.4
0.2
−0.2
0
2000
4000
6000
8000
10000
Index


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.

Computer Lab 4

Elon Brange, Ludwig Thaung

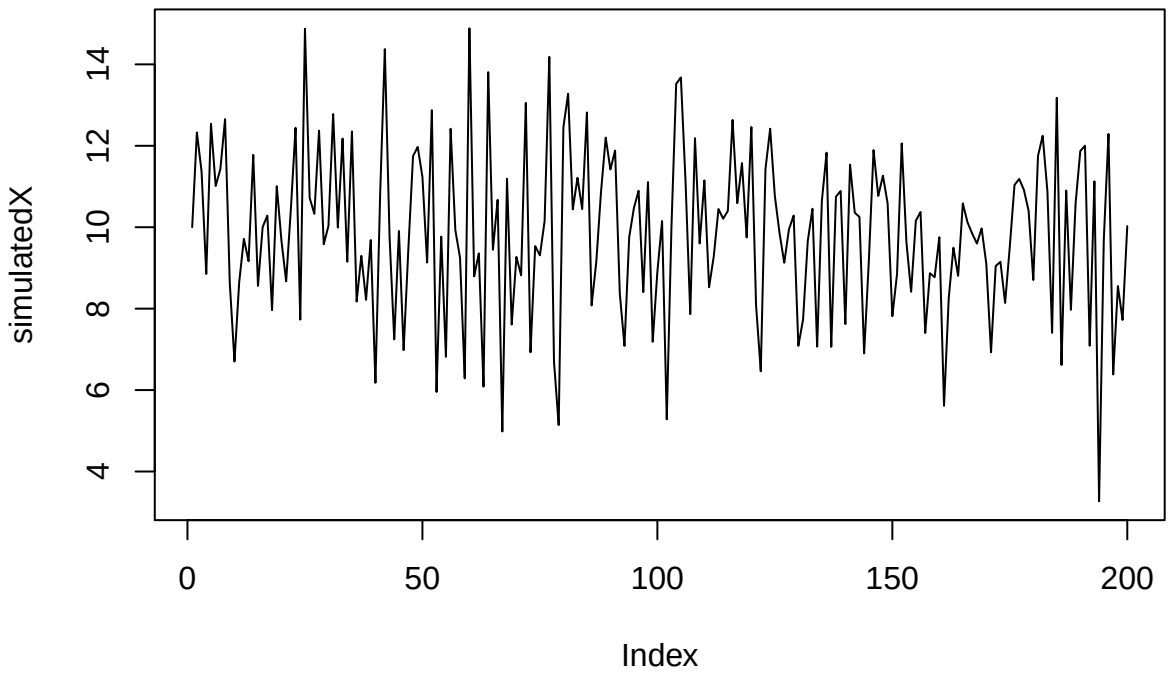
5/23/2019

Computer Lab 4

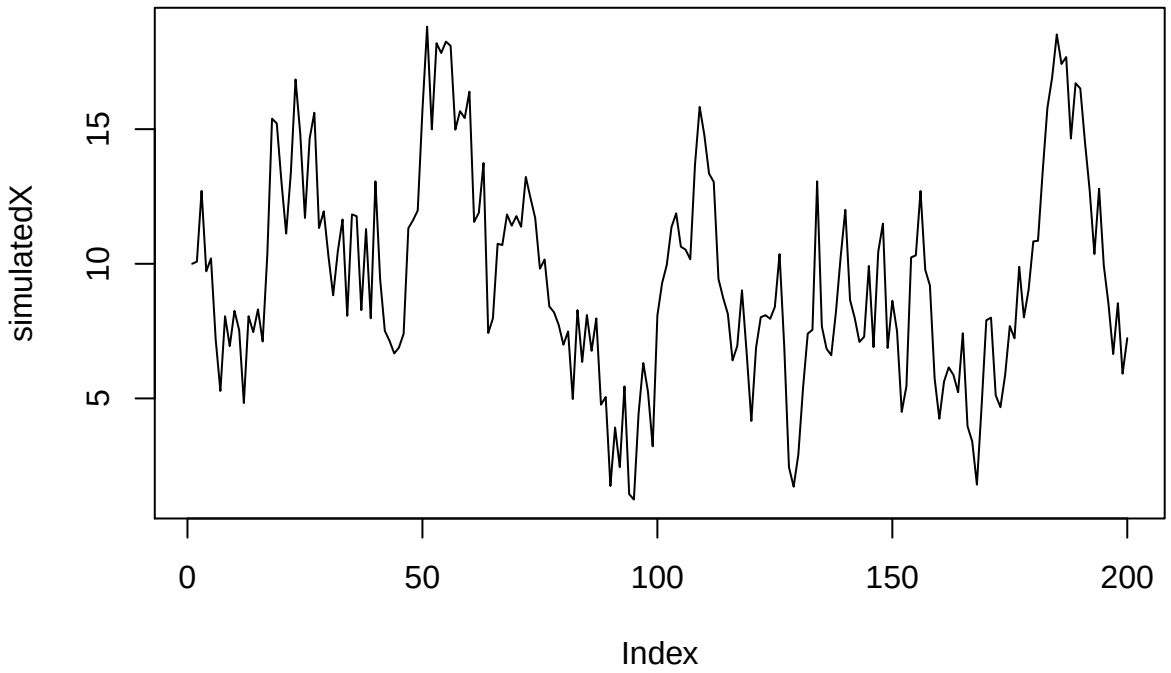
##1

1 a)

```
ARfunction <- function(n, mu, omega, sigma2) {
  x = matrix(0, n, 1)
  x[1, 1] = mu
  noise <- rnorm(n-1,0,sigma2)
  for(i in 1:(n-1)) {
    x[i+1, 1] = mu + omega*(x[i, 1] - mu) + noise[i]
  }
  return(x)
}

simulatedX <- ARfunction(200, 10, -11/10 + 1, 2)
plot(simulatedX, type='l')
```




```
simulatedX <- ARfunction(200, 10, -11/10 + 2, 2)
plot(simulatedX, type='l')
```



The effect of Phi on X1:T is that, depending on the value of Phi determines X1:T's autocorrelation with the previous value of the difference between X1:T-1 and the mean(mu). I.e. what omega does is how much the previous value affect the current value and in which direction.

1 b)

```
simulatedX <- c(ARfunction(1000, 10, 0.3, 1))
simulatedY <- c(ARfunction(1000, 10, 0.95, 1))

Nx = length(simulatedX)
Ny = length(simulatedY)

x_dat <- list(N = Nx,
              X = simulatedX)

y_dat <- list(N = Ny,
              X = simulatedY)

fitX <- stan(file = 'mystan.stan', data = x_dat)
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info (non-fatal): Comments beginning with # are deprecated.  Please use // in place of # for line co
```

```
print(fitX)
```

```
## Inference for Stan model: mystan.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## muRandom       9.98    0.00 0.05    9.89    9.95    9.98   10.01   10.07
## sigma2Random   1.00    0.00 0.02    0.96    0.98    1.00    1.01    1.05
```

```
## omegaRandom     0.32    0.00 0.03    0.26    0.30    0.32    0.34    0.38
## lp__         -496.97    0.03 1.24 -500.31 -497.50 -496.64 -496.07 -495.56
##              n_eff Rhat
## muRandom      2771    1
## sigma2Random  4347    1
## omegaRandom   4303    1
## lp__          2019    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 11:52:22 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
summaryFitX = summary(fitX)$summary
dataX = extract(fitX)

meanX = summaryFitX[,1]
highX = summaryFitX[,8]
lowX = summaryFitX[,4]
efficientX = summaryFitX[, 9]

print(paste0("The 95% interval for mu is: ", lowX[1], " - ", highX[1]))
```

```
## [1] "The 95% interval for mu is: 9.88810295021301 - 10.066740566065"
```

```
print(paste0(" and the mean for mu is: ", meanX[1]))
```

```
## [1] " and the mean for mu is: 9.9788840975677"
```

```
print(paste0("The 95% interval for sigma is: ", lowX[2], " - ", highX[2]))
```

```
## [1] "The 95% interval for sigma is: 0.955280658829604 - 1.04557304166958"
```

```
print(paste0(" and the mean for sigma is: ", meanX[2]))
```

```
## [1] " and the mean for sigma is: 0.998334194814575"
```

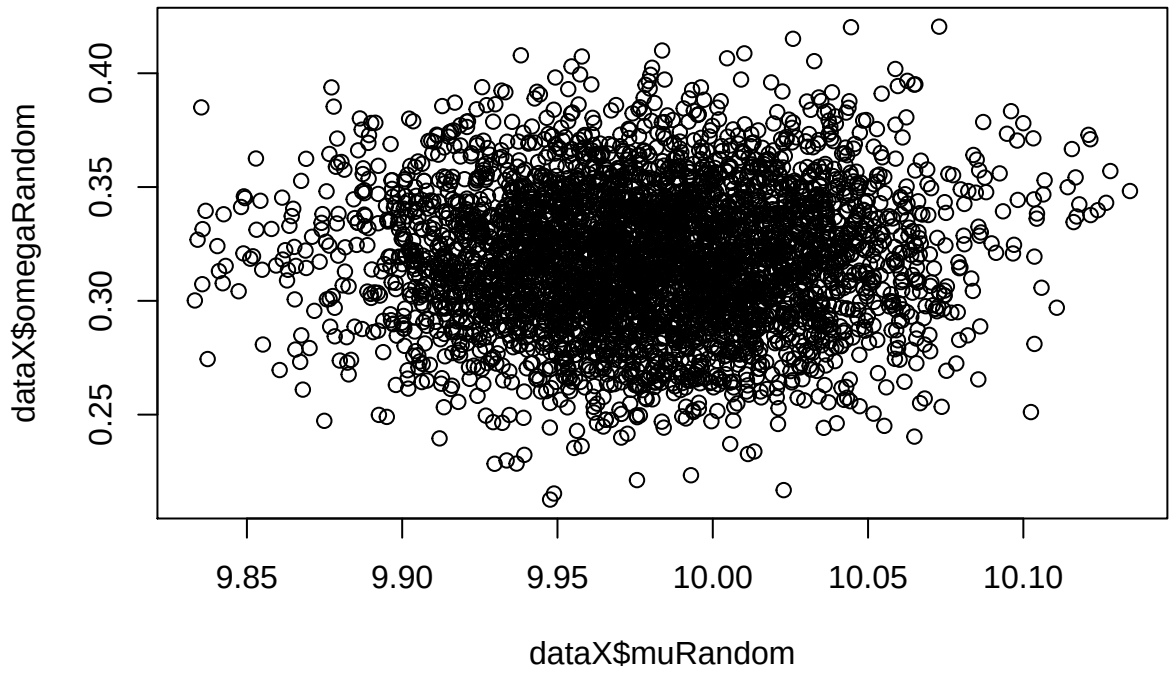
```
print(paste0("The 95% interval for omega is: ", lowX[3], " - ", highX[3]))
```

```
## [1] "The 95% interval for omega is: 0.26124444813333 - 0.37852625241563"
```

```
print(paste0(" and the mean for omega is: ", meanX[3]))
```

```
## [1] " and the mean for omega is: 0.318332952302275"
```

```
#print(meanX)
#print(highX)
#print(lowX)
#print(efficientX)

plot(dataX$muRandom, dataX$omegaRandom)
```




```
library(MASS)
den3d <- kde2d(dataX$muRandom, dataX$omegaRandom)
#library(plotly)
#plot_ly(x=den3d$x, y=den3d$y, z=den3d$z/length(dataX$muRandom)) %>% add_surface()

library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: dotCall64
```

```
## Loading required package: grid
```

```
## Spam version 2.2-2 (2019-03-07) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

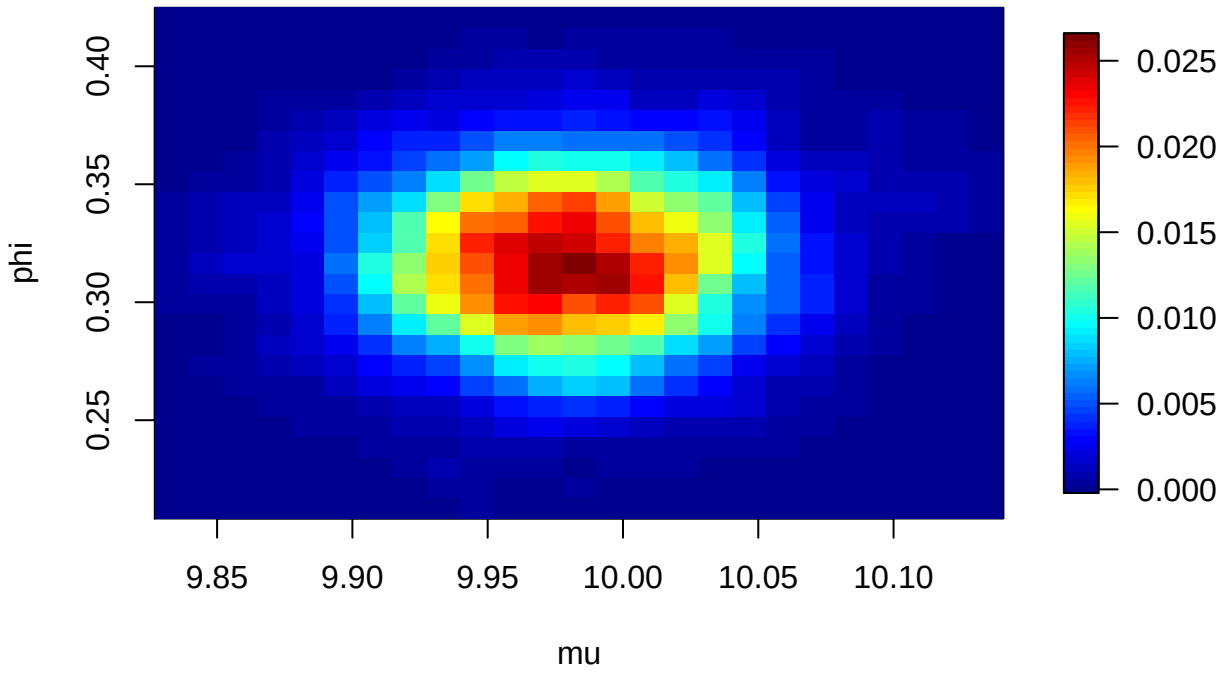
```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

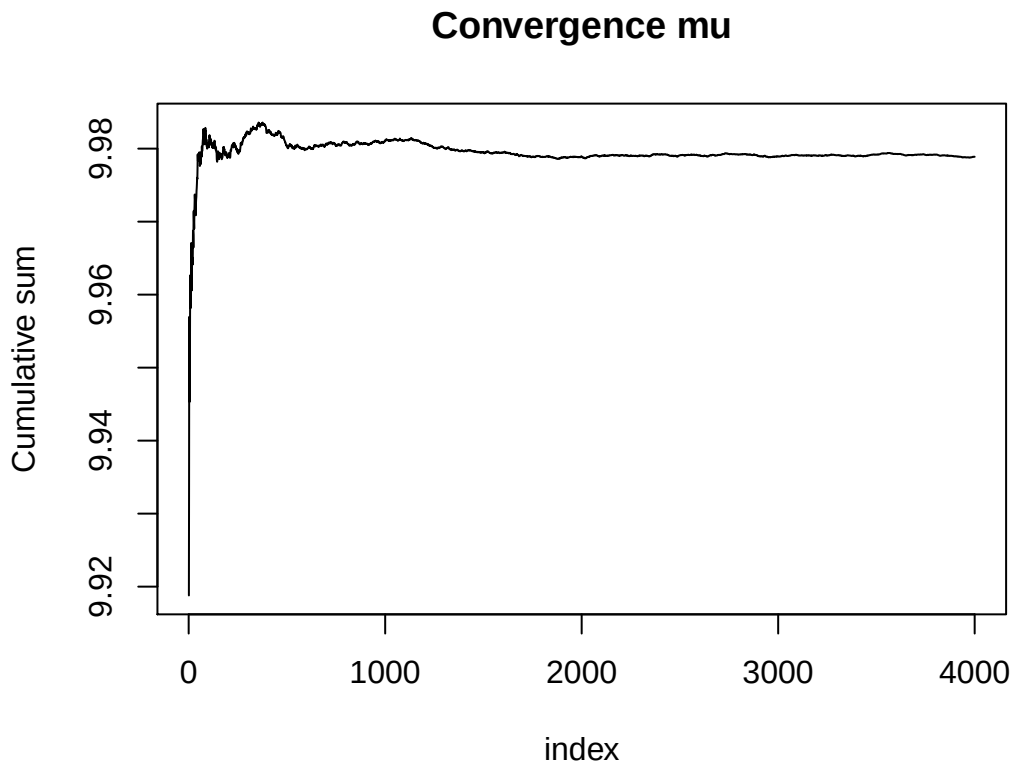
```
## Loading required package: maps
```

```
## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code
```

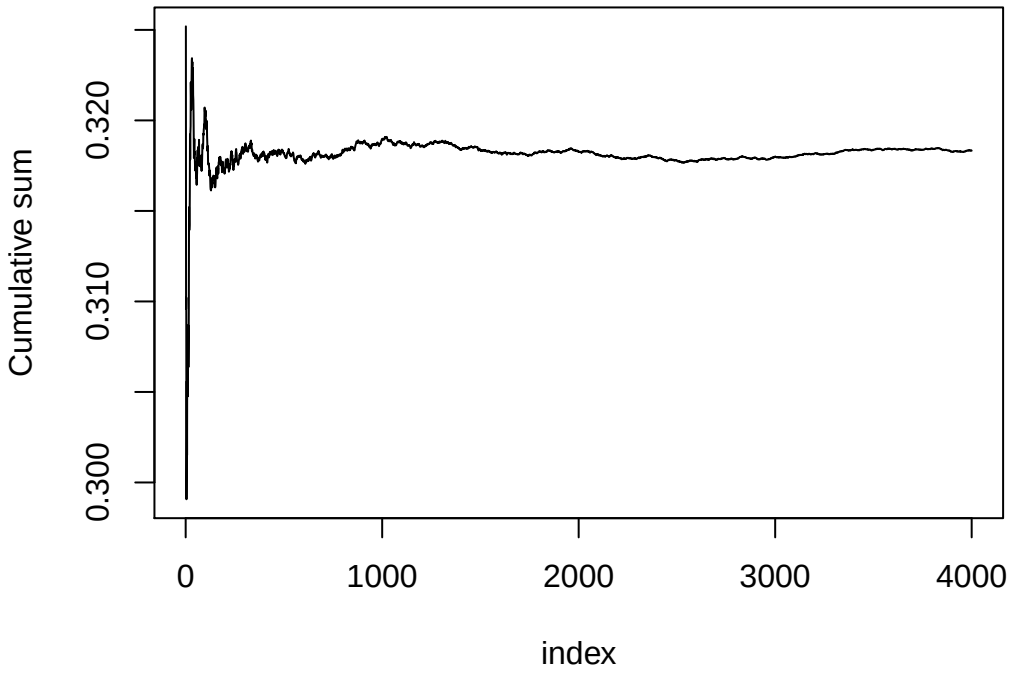
```
image.plot(den3d$x,den3d$y,den3d$z/length(dataX$muRandom),xlab="mu", ylab="phi")
```




```
plot(cumsum(dataX$muRandom)/seq(1,length(dataX$muRandom)), type='l', xlab ="index", ylab = "Cumulative s
```




```
plot(cumsum(dataX$omegaRandom)/seq(1,length(dataX$omegaRandom)), type='l', xlab = "index", ylab="Cumulat
```




```
fitY <- stan(file = 'mystan.stan', data = y_dat)
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info (non-fatal): Comments beginning with # are deprecated.  Please use // in place of # for line co
```

```
print(fitY)
```

```
## Inference for Stan model: mystan.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                 mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## muRandom        9.09    0.01 0.67    7.74    8.67    9.10    9.51   10.41
## sigma2Random    0.99    0.00 0.02    0.95    0.98    0.99    1.01    1.04
## omegaRandom     0.95    0.00 0.01    0.93    0.94    0.95    0.96    0.97
## lp__         -492.00    0.03 1.27 -495.36 -492.60 -491.66 -491.06 -490.54
##              n_eff Rhat
## muRandom      2625    1
## sigma2Random  3193    1
## omegaRandom   3159    1
## lp__          1801    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 11:52:28 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
summaryFitY = summary(fitY)$summary
dataY = extract(fitY)

meanY = summaryFitY[,1]
```

```r
highY = summaryFitY[,8]
lowY = summaryFitY[,4]
efficientY = summaryFitY[, 9]

print(paste0("The 95% interval for mu is: ", lowY[1], " - ", highY[1]))
```

```
## [1] "The 95% interval for mu is: 7.73562081703205 - 10.4123363175313"
```

```r
print(paste0(" and the mean for mu is: ", meanY[1]))
```

```
## [1] " and the mean for mu is: 9.0941361223012"
```

```r
print(paste0("The 95% interval for sigma is: ", lowY[2], " - ", highY[2]))
```

```
## [1] "The 95% interval for sigma is: 0.94886108395219 - 1.03711159445509"
```

```r
print(paste0(" and the mean for sigma is: ", meanY[2]))
```

```
## [1] " and the mean for sigma is: 0.992469917416263"
```

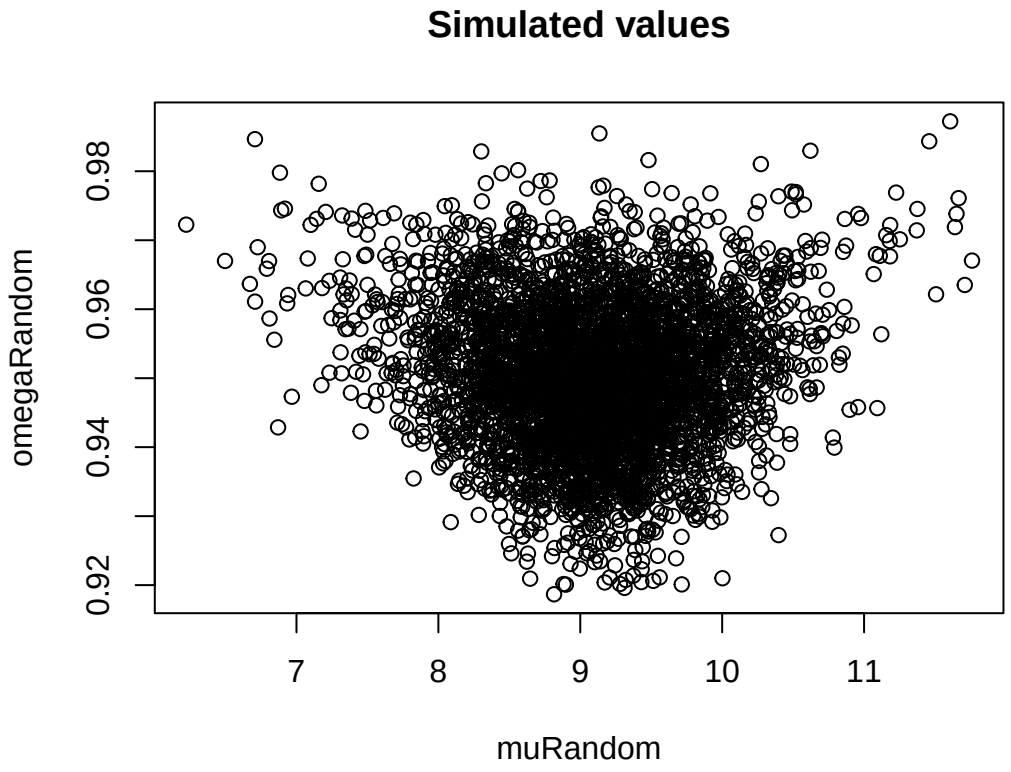
```r
print(paste0("The 95% interval for omega is: ", lowY[3], " - ", highY[3]))
```

```
## [1] "The 95% interval for omega is: 0.930474879736404 - 0.971066969694039"
```

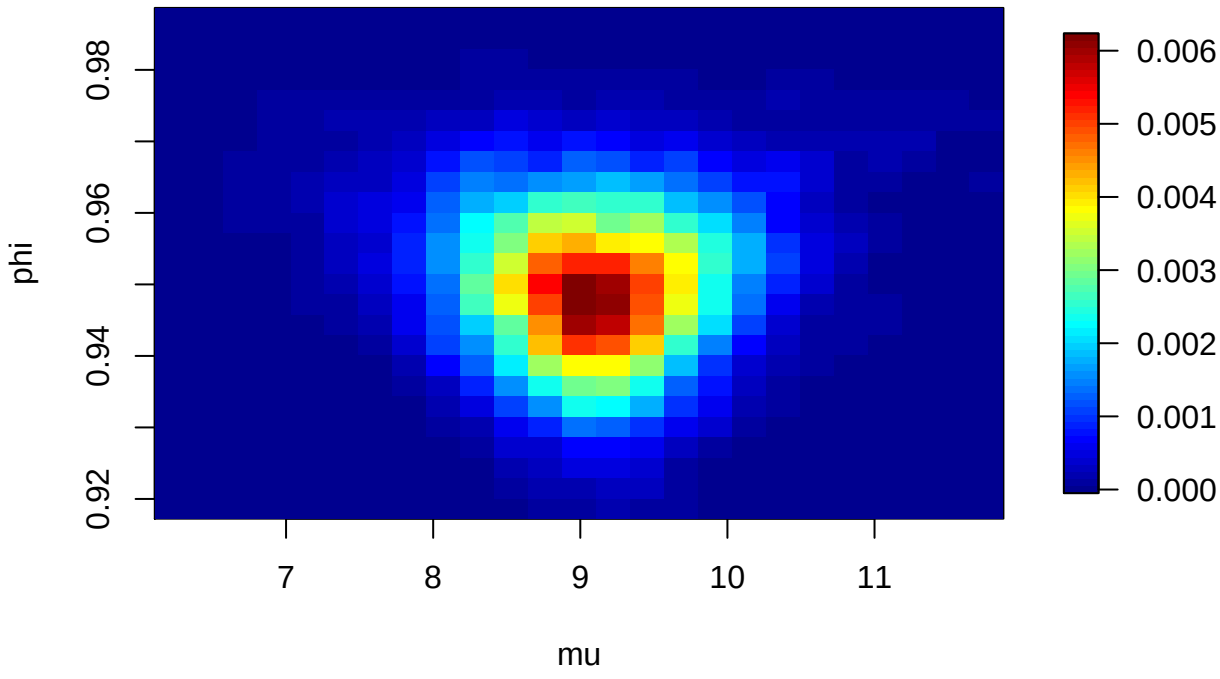
```r
print(paste0(" and the mean for omega is: ", meanY[3]))
```

```
## [1] " and the mean for omega is: 0.950358506193888"
```

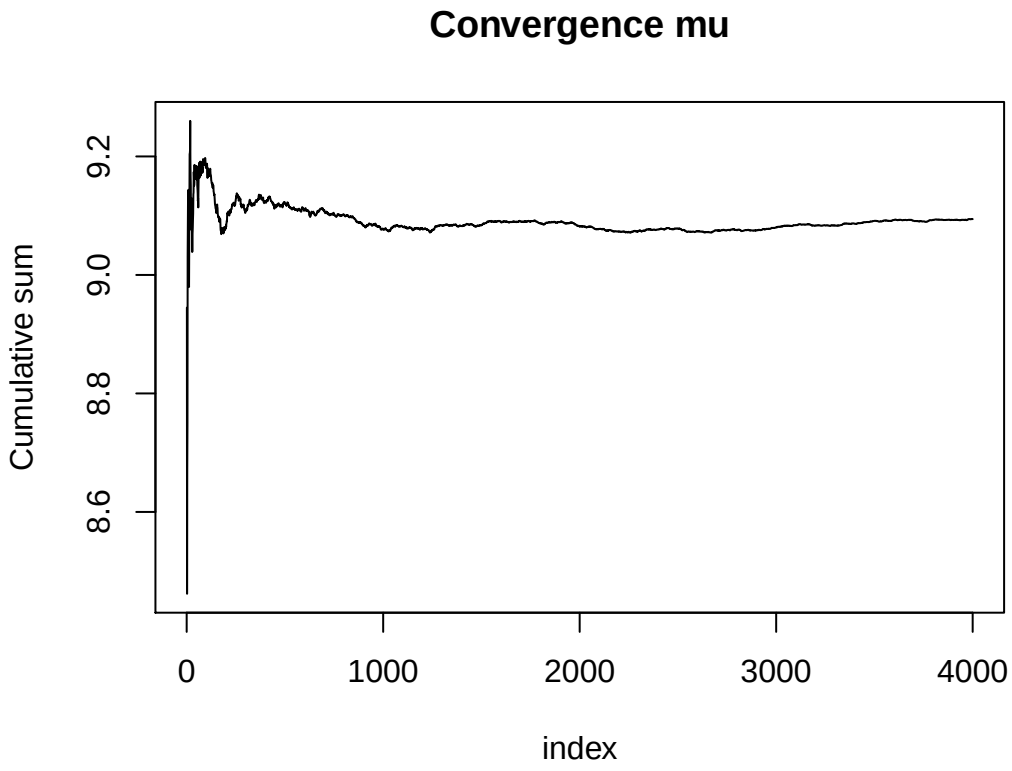
```r
#print(meanY)
#print(highY)
#print(lowY)
#print(efficientY)

plot(dataY$muRandom, dataY$omegaRandom, xlab ="muRandom", ylab ="omegaRandom", main = "Simulated values
```



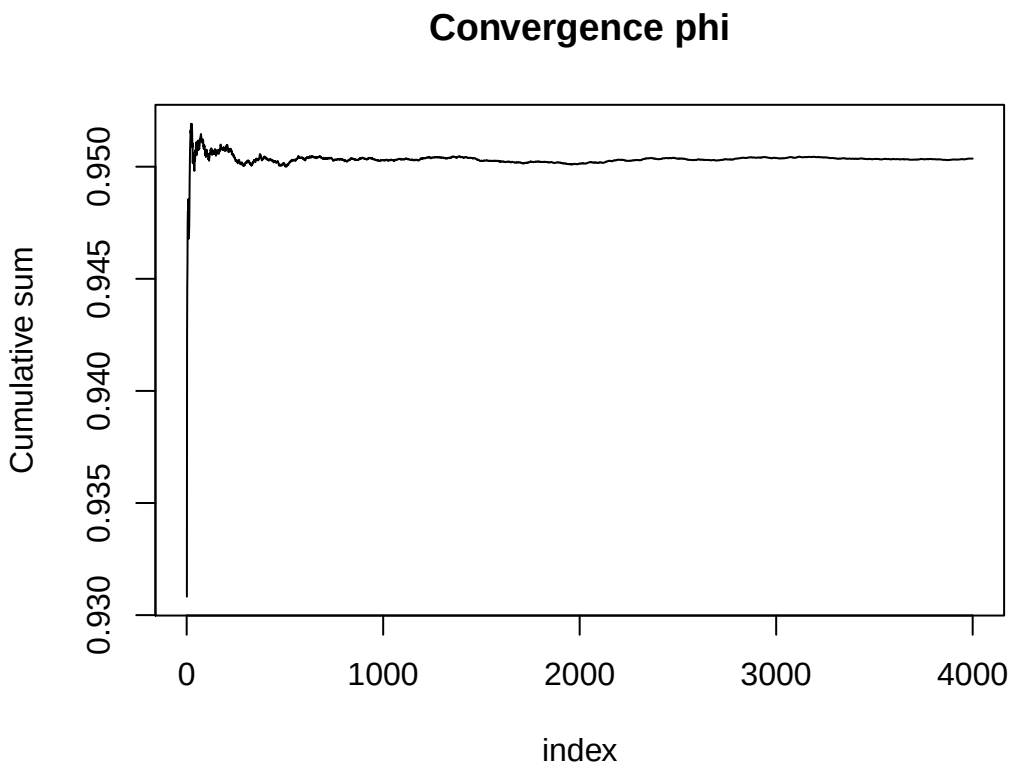

```
library(MASS)
den3d <- kde2d(dataY$muRandom, dataY$omegaRandom)
#library(plotly)
#plot_ly(x=den3d$x, y=den3d$y, z=den3d$z/length(dataY$muRandom)) %>% add_surface()

library(fields)
image.plot(den3d$x,den3d$y,den3d$z/length(dataX$muRandom), xlab="mu", ylab="phi")
```



```
plot(cumsum(dataY$muRandom)/seq(1,length(dataY$muRandom)), type='l', , xlab ="index", ylab ="Cumulative
```




```
plot(cumsum(dataY$omegaRandom)/seq(1,length(dataY$omegaRandom)), type='l', , xlab ="index", ylab ="Cumul
```



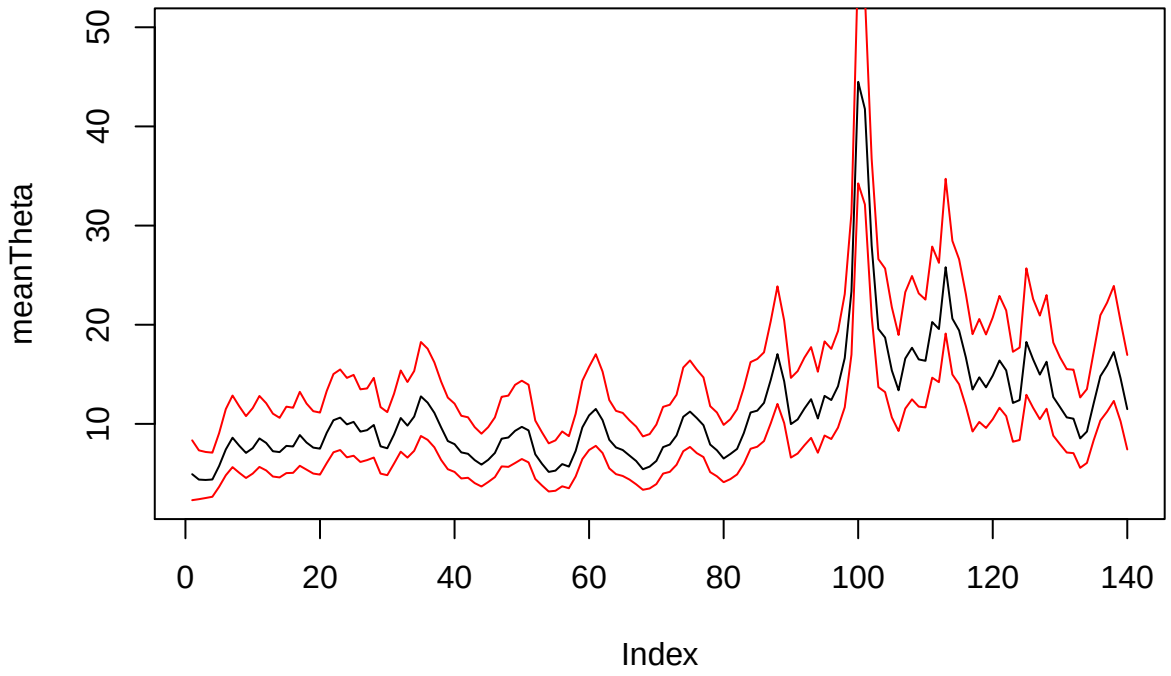
i) When compared to the values in SimulatedX & SimulatedY we can see that we are able to estimate the

true values.

ii) The two plots show that the parameters converges close to the real value quite quickly.

1 c)

```
campyData<-c(t(read.table("campy.dat",header=TRUE)))

Ny = length(campyData)

campy_dat <- list(N = Ny,
                  y = campyData)
fitP <- stan(file = 'poisson.stan', data = campy_dat)

summaryFitP = summary(fitP)$summary
rowsCols = dim(summaryFitP)
highTheta = c()
meanTheta = c()
lowTheta = c()
for(i in 1:(rowsCols[1]-4)) {
  highTheta = c(highTheta, exp(summaryFitP[i, 8]))
  meanTheta = c(meanTheta, exp(summaryFitP[i, 1]))
  lowTheta = c(lowTheta, exp(summaryFitP[i, 4]))
}

plot(meanTheta, type='l', ylim=c(min(lowTheta), 50))
lines(highTheta, col='red')
lines(lowTheta, col='red')
```



1

d)

```
campy_dat <- list(N = Ny,
                  y = campyData,
```

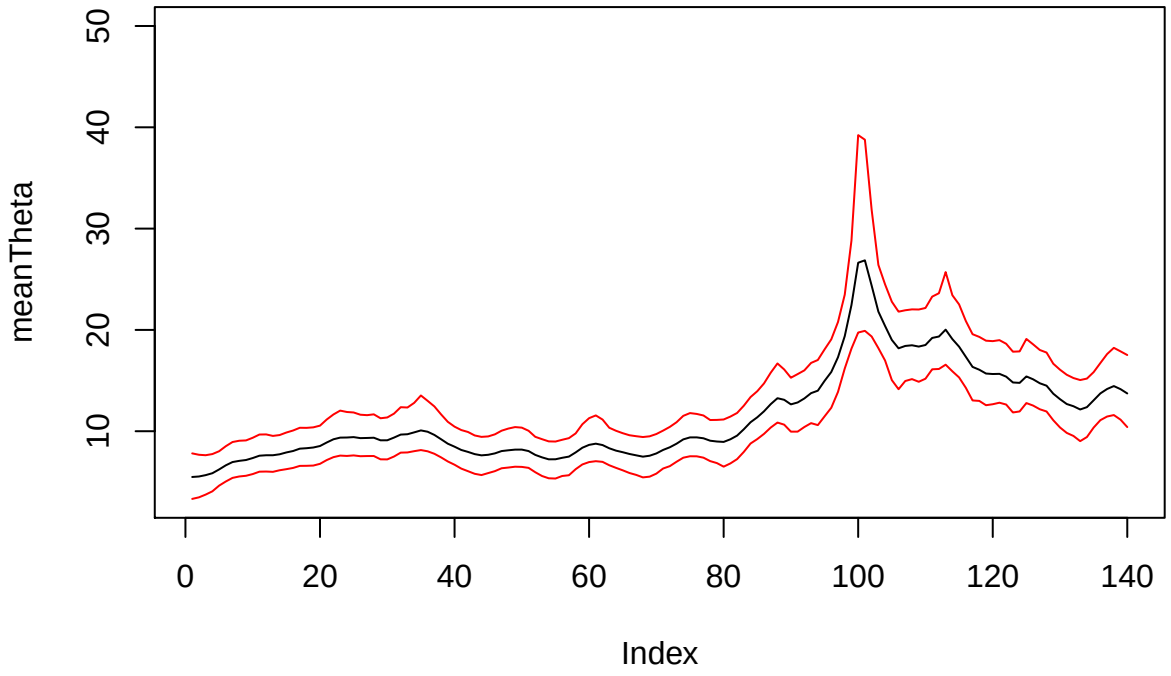
```
                  sigma2Nu = 50,
                  sigma2Sigma = 0.01)
fitP <- stan(file = 'poissonPrior.stan', data = campy_dat)
```

```
## Warning: There were 28 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was low. S
## http://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
summaryFitP = summary(fitP)$summary
rowsCols = dim(summaryFitP)
rowsCols[1] = rowsCols[1] - 4
highTheta = c()
meanTheta = c()
lowTheta = c()
for(i in 1:rowsCols[1]) {
  highTheta = c(highTheta, exp(summaryFitP[i, 8]))
  meanTheta = c(meanTheta, exp(summaryFitP[i, 1]))
  lowTheta = c(lowTheta, exp(summaryFitP[i, 4]))
}

plot(meanTheta, type='l', ylim=c(min(lowTheta), 50))
lines(highTheta, col='red')
lines(lowTheta, col='red')
```



The posterior has become less volatile.