Lab_03

Thijs Quast

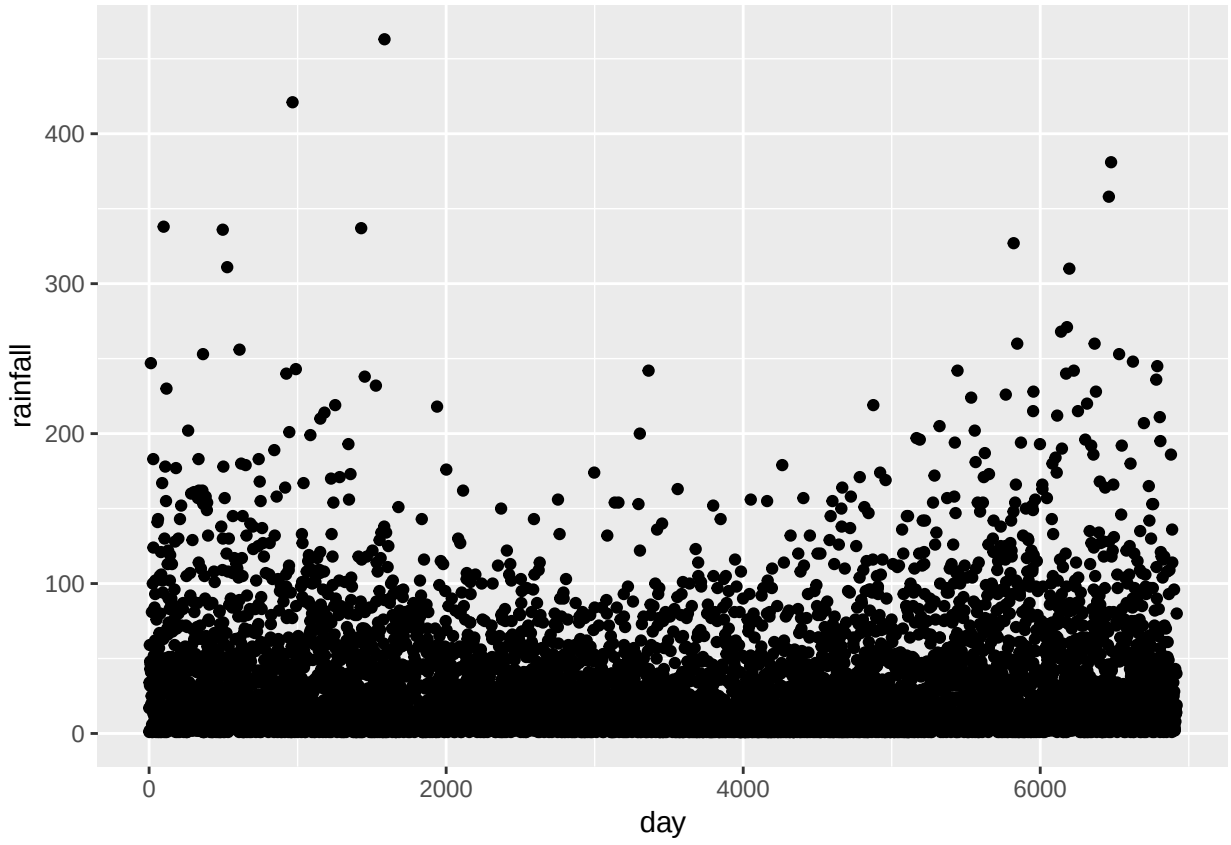
5-5-2019

Contents

```
library(mvtnorm)
```

```
rainfall <- read.delim("rainfall.dat")
rainfall$day <- c(1:nrow(rainfall))
colnames(rainfall) <- c("rainfall", "day")
```

```
library(ggplot2)
rainfall_plot <- ggplot(rainfall, aes(x=day, y = rainfall)) + geom_point()
rainfall_plot
```



Question 1

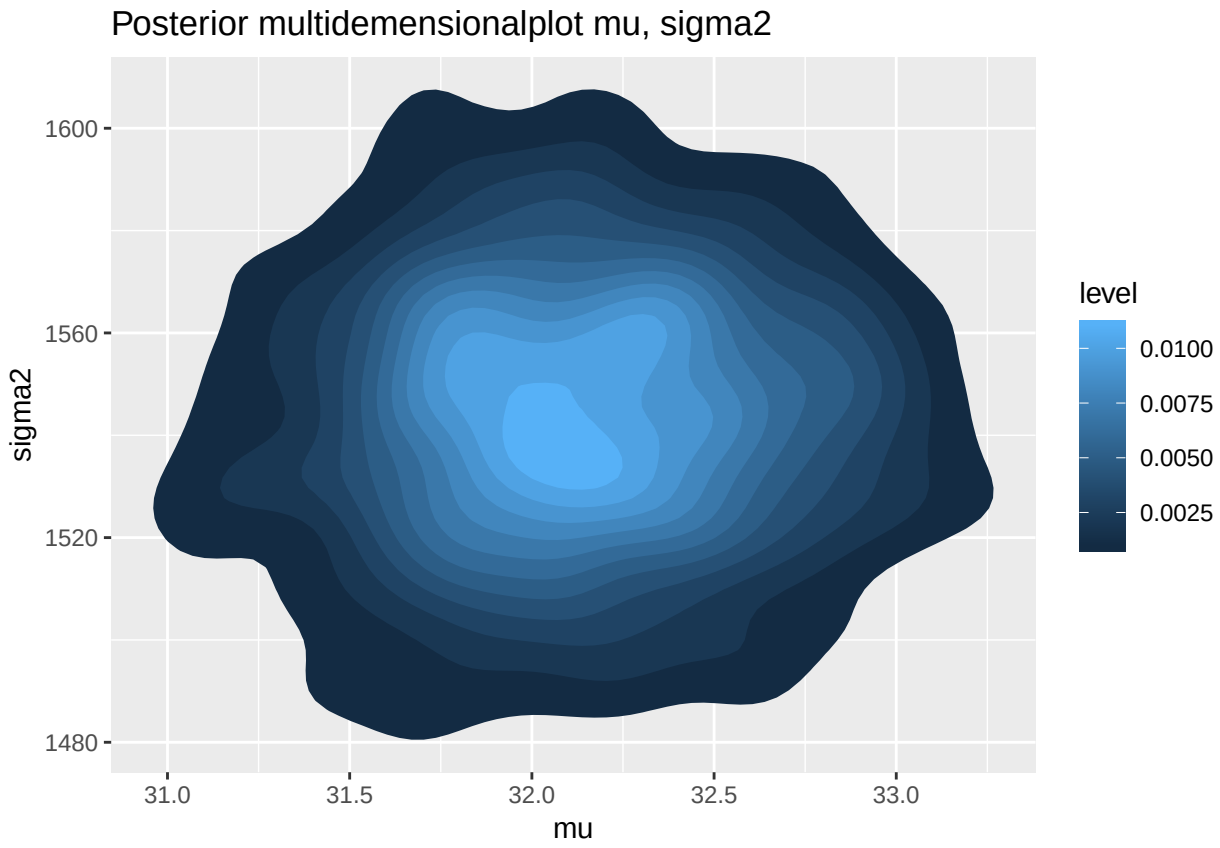
a

```
GibbsSampler <- function(data, N, mu_0, tau2_0, nu_0, sigma2_0){
  n <- nrow(data)
  xbar <- mean(data[,1])

  # parameters
  nu_n <- nu_0 + n #fixed

  mu <- c()
  sigma2 <- c()
```

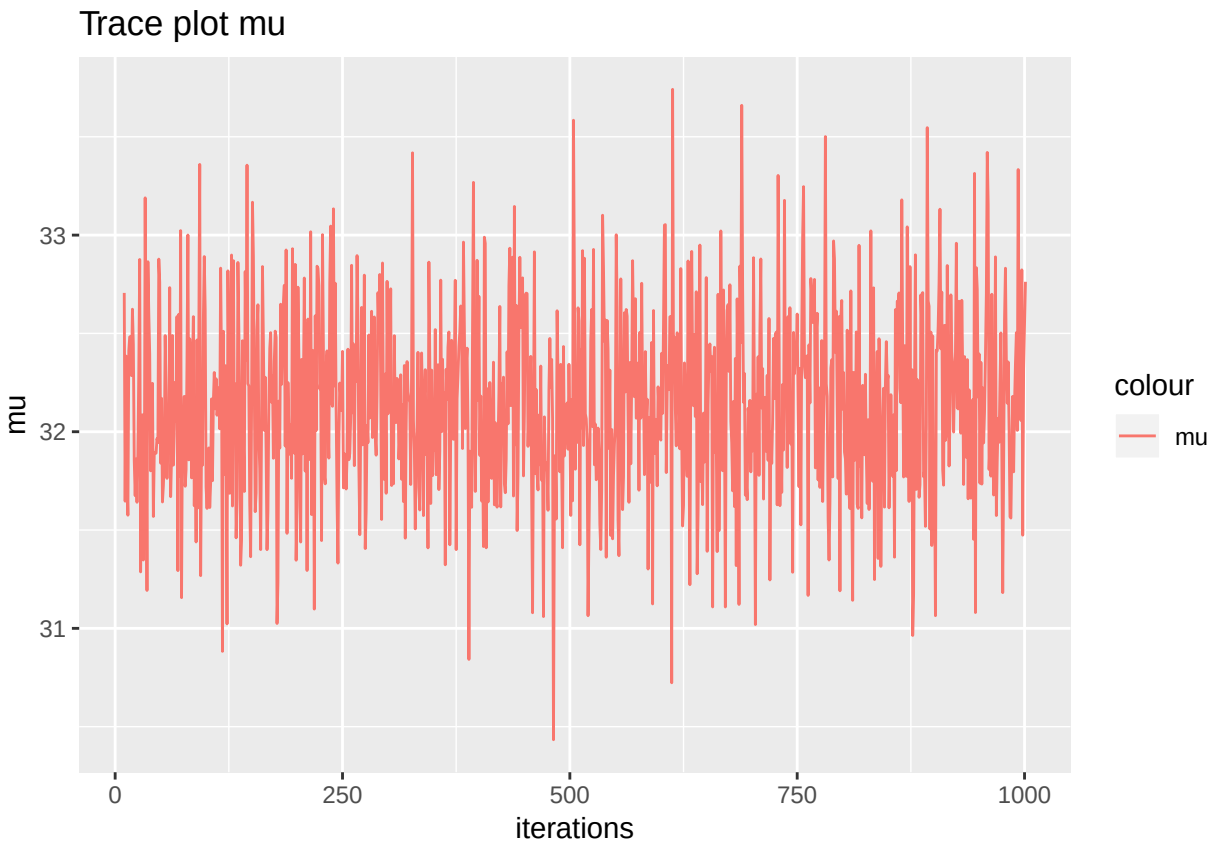
```
  mu[1] <- rnorm(1, mu_0, sqrt(tau2_0))
  sigma2[1] <- (nu_0 * sigma2_0)/rchisq(n = 1, df = nu_0)

  for (i in 1:N){

    # mu
    w <- (n/sigma2[i])/((n/sigma2[i]) + (1/tau2_0))
    mu_n <- w*xbar + (1-w)*mu_0
    tau2_n <- (n/sigma2[i] + 1/tau2_0)^-1
    mu[i+1] <- rnorm(n = 1, mu_n, sd = sqrt(tau2_n))


    # sigma
    sigma2_n <- ((nu_0*sigma2_0) + sum((data[,1] - mu[i])^2))/ (n+nu_0)
    sigma2[i+1] <- (nu_n * sigma2_n)/rchisq(1, df = nu_n)
  }

  df <- data.frame("mu" = mu, "sigma2" = sigma2)
  return(df)
}
```

```
sample1 <- GibbsSampler(data = rainfall,
                        N = 1000,
                        mu_0 = 0,
                        tau2_0 = 50,
                        nu_0 = 5,
                        sigma2_0 = 20)

sample1$iterations <- c(1:nrow(sample1))

sample1WOBurnin <- sample1[10:1001, ]
```

```
posterior_plot <- ggplot(data = sample1WOBurnin, aes(x = mu, y = sigma2)) + stat_density_2d(aes(fill=..
                                                                                             geom = "pol

  ggtitle("Posterior multidemensionalplot mu, sigma2")
posterior_plot
```

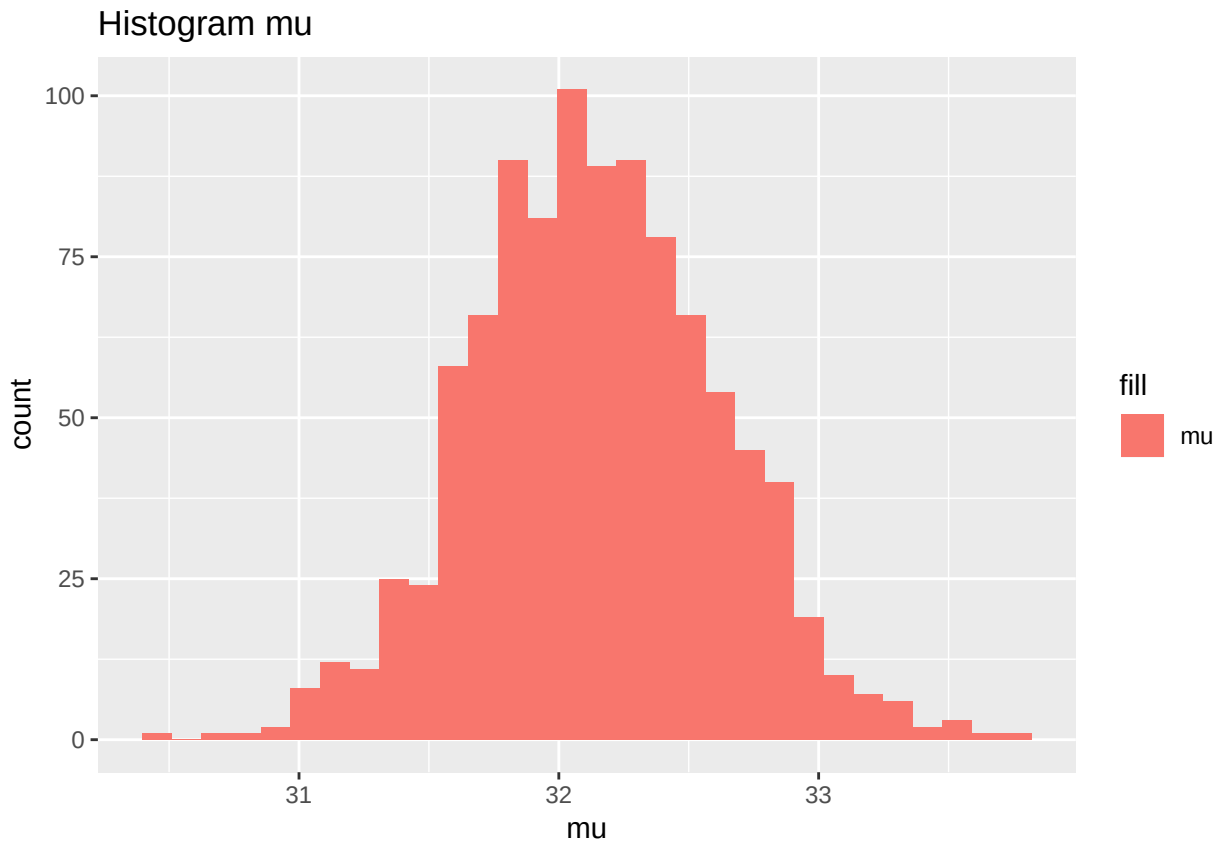



```
# Trace plots
trace_plot_mu <- ggplot(data = sample1WOBurnin, aes(x = iterations, y = mu, col="mu")) + geom_line() +
  ggtitle("Trace plot mu")

trace_plot_sigma2 <- ggplot(data = sample1WOBurnin, aes(x = iterations, y = sigma2, col="sigma2")) + ge
  ggtitle("Trace plot sigma2")

trace_plot_mu
```

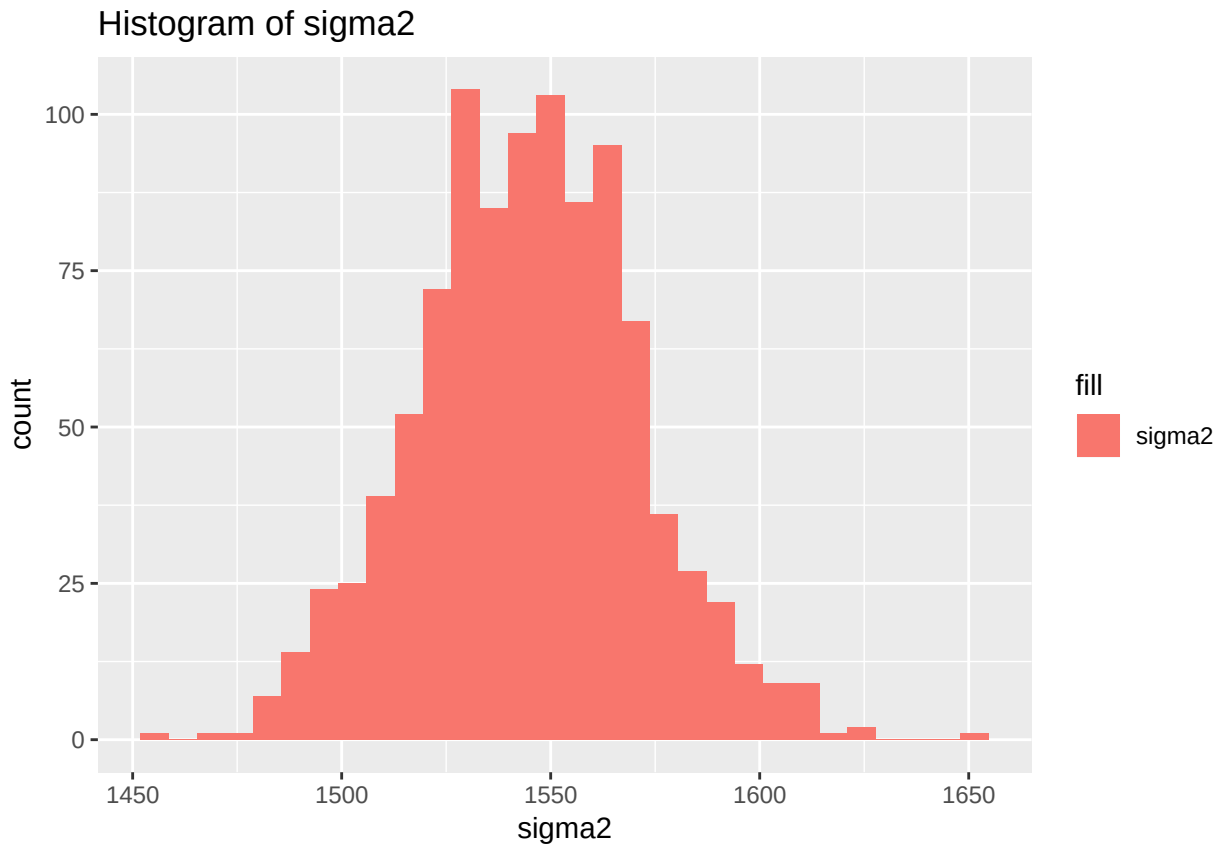



```
trace_plot_sigma2
```


```
# Histograms
histogram_mu <- ggplot(data = sample1WOBurnin, aes(x= mu, fill="mu")) + geom_histogram(bins = 30) +
  ggtitle("Histogram mu")
histogram_mu
```

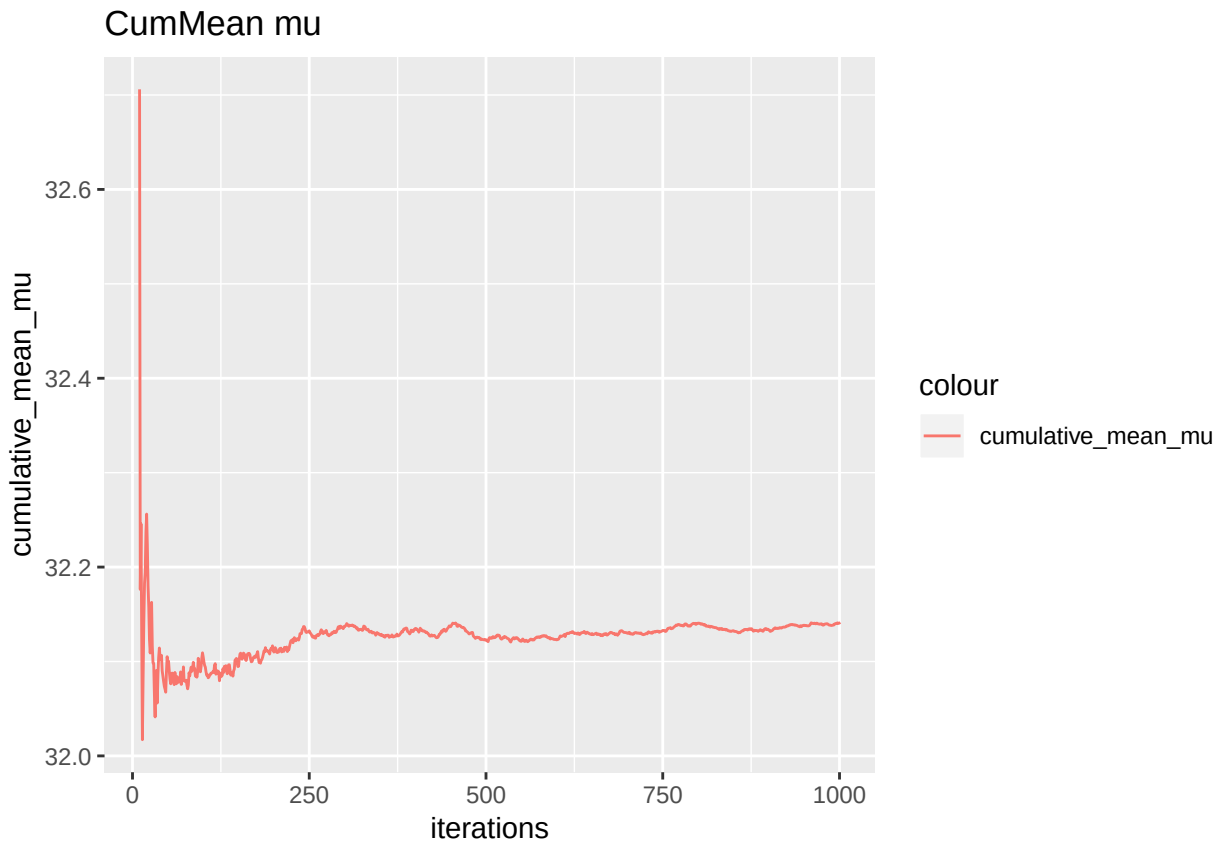



```
histogram_sigma2 <- ggplot(data = sample1WOBurnin, aes(x= sigma2, fill="sigma2")) + geom_histogram(bins
  ggtitle("Histogram of sigma2")
histogram_sigma2
```

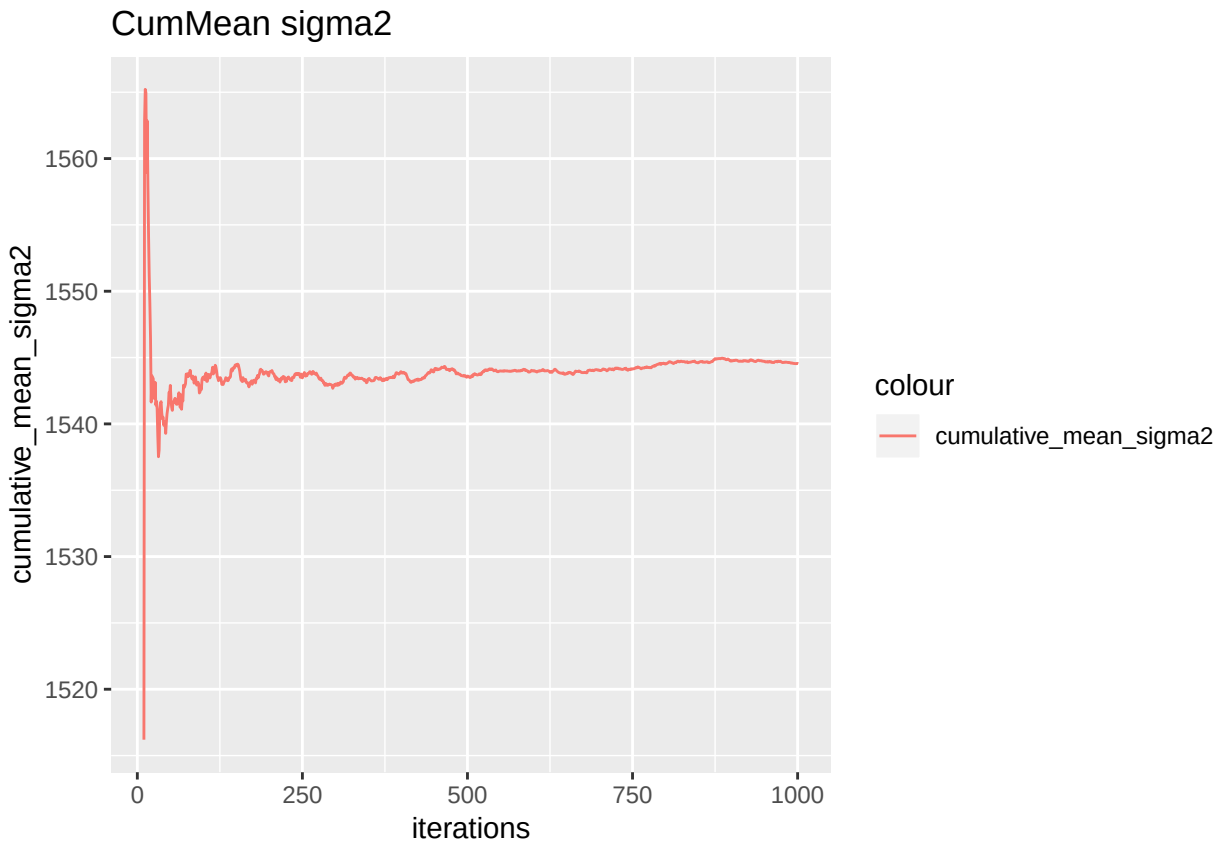



```
cumulative_mean_mu <- c()
cumulative_mean_sigma2 <- c()
for (i in 1:nrow(sample1WOBurnin)){
  cumulative_mean_mu[i] <- sum(sample1WOBurnin$mu[1:i])/i
  cumulative_mean_sigma2[i] <- sum(sample1WOBurnin$sigma2[1:i])/i
}
sample1WOBurnin$cumulative_mean_mu <- cumulative_mean_mu
sample1WOBurnin$cumulative_mean_sigma2 <- cumulative_mean_sigma2
```

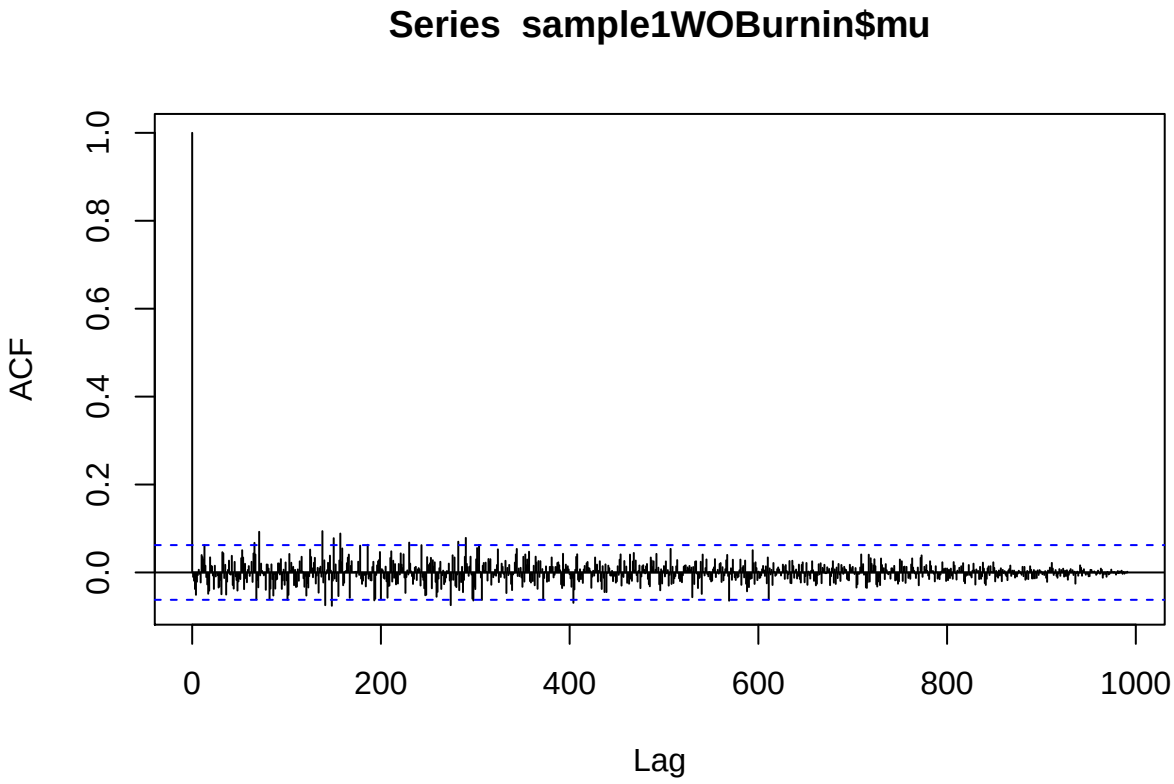
```
cummean_plot_mu <- ggplot(data = sample1WOBurnin, aes(x = iterations, y = cumulative_mean_mu,
                                                      col="cumulative_mean_mu")) + geom_line() +
  ggtitle("CumMean mu")

cummean_plot_sigma2 <- ggplot(data = sample1WOBurnin, aes(x = iterations, y = cumulative_mean_sigma2,
                                                          col="cumulative_mean_sigma2")) + geom_line() +
  ggtitle("CumMean sigma2")

cummean_plot_mu
```



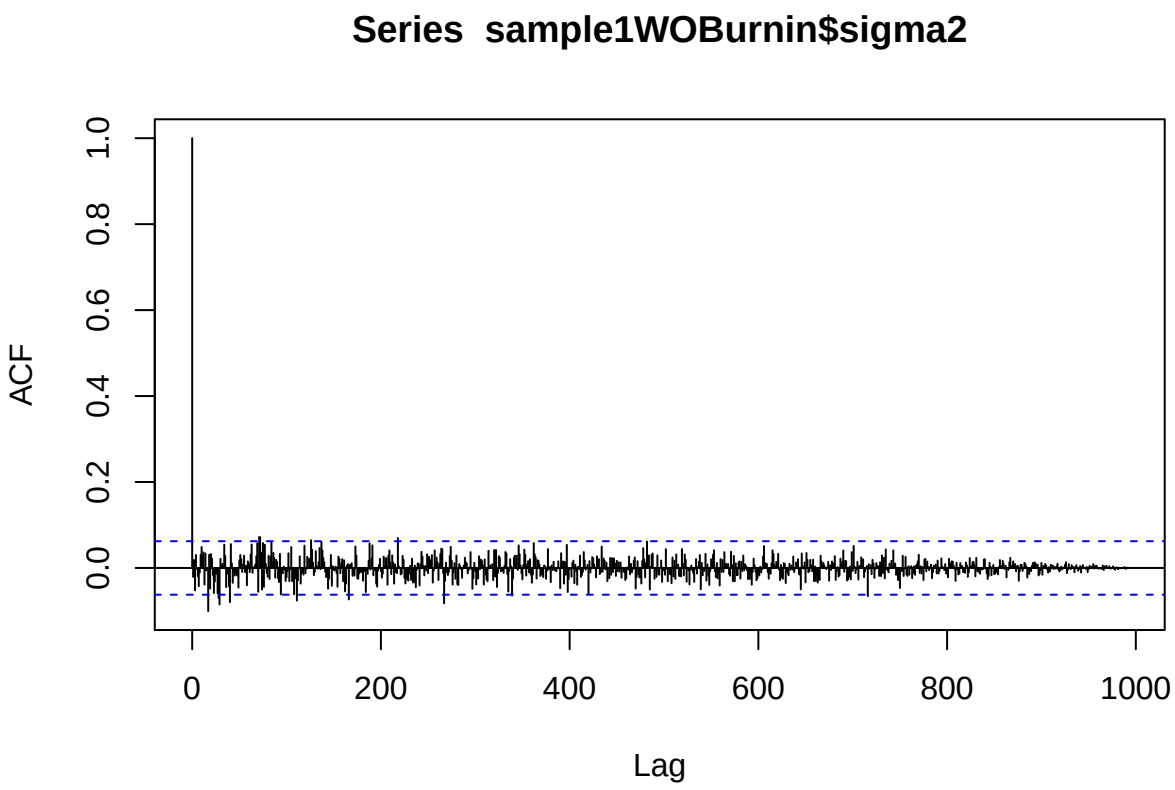

```
cummean_plot_sigma2
```




```
# Autocorrelation plots
autocorrelation_mu <- acf(x = sample1WOBurnin$mu, lag.max = 1000)
```



```r
autocorrelation_sigma2 <- acf(x = sample1WOBurnin$sigma2, lag.max = 1000)
```



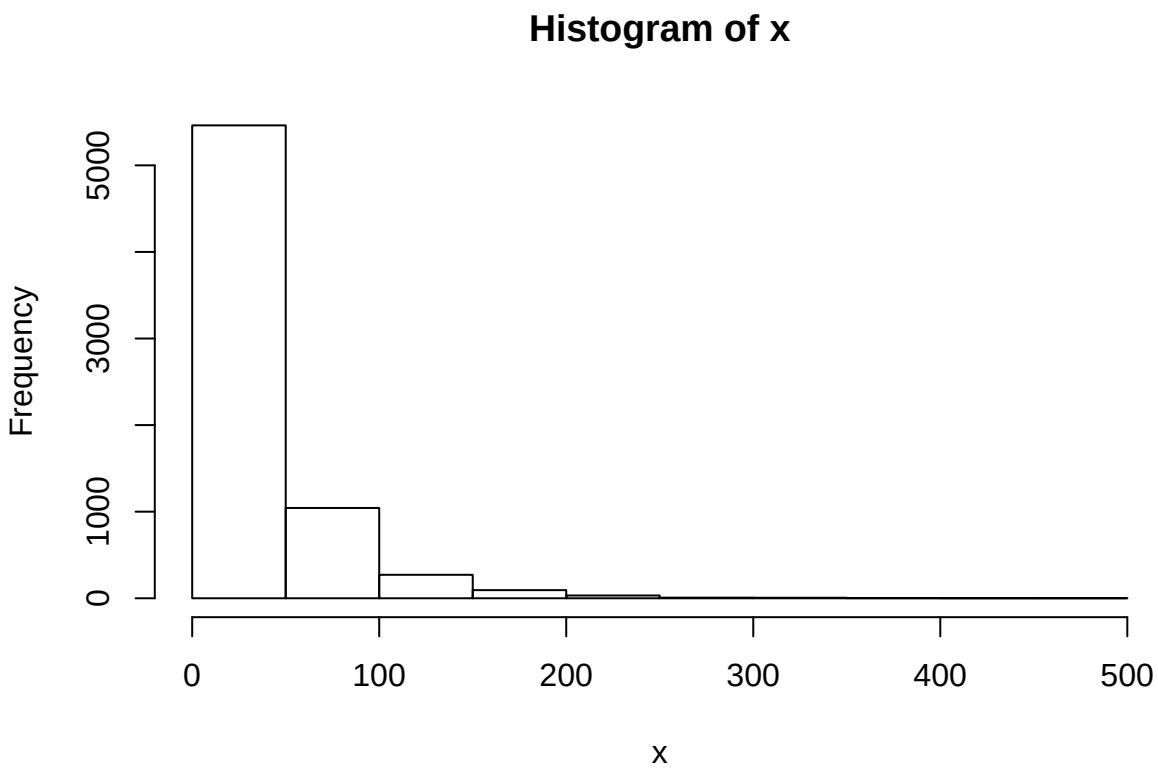
b

```r
# Data
x <- as.matrix(rainfall$rainfall)

# 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(5,nComp) # Prior std of mu
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(2,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")
sleepTime <- 0.1 # Adding sleep time between iterations for plotting
```

```r
###### 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(0,1,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)$density))
```



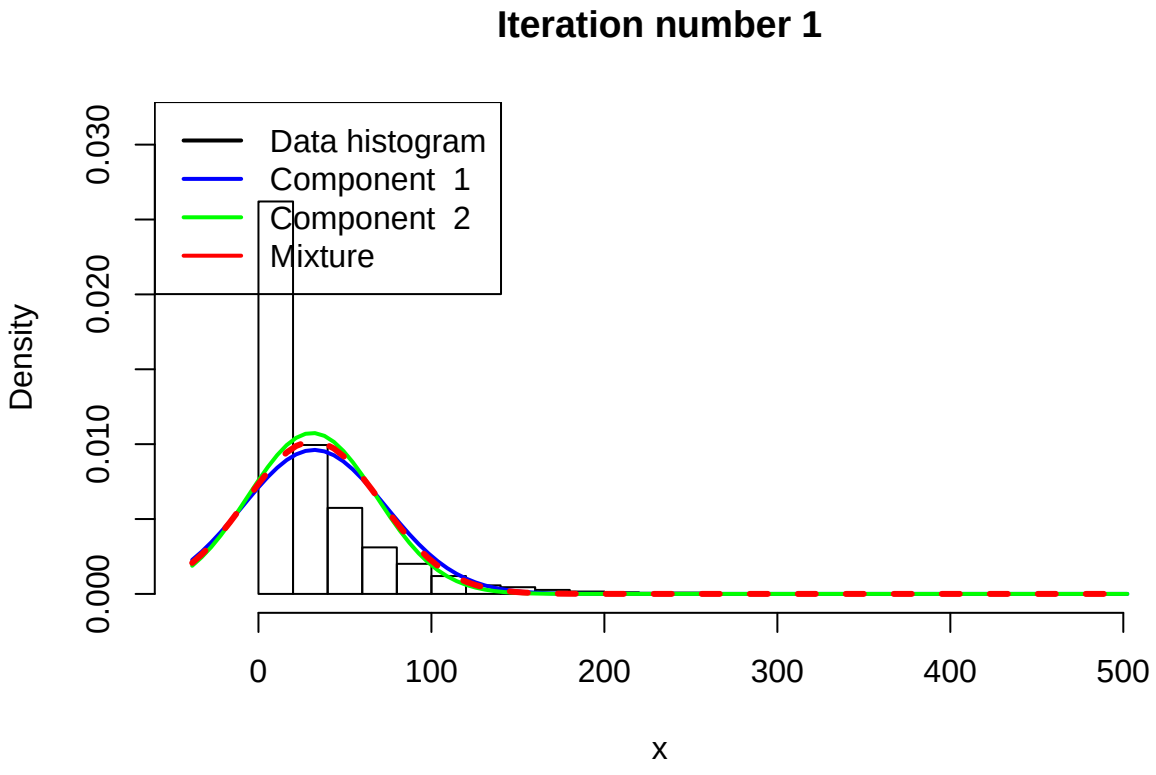

```
for (k in 1:nIter){
  message(paste('Iteration number:',k))
  alloc <- S2alloc(S) # Just a function that converts between different representations of the group al
  nAlloc <- colSums(S)
  print(nAlloc)
  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)

  # 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 = 20, 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("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    Sys.sleep(sleepTime)
  }

}
```

```
## Iteration number: 1
```

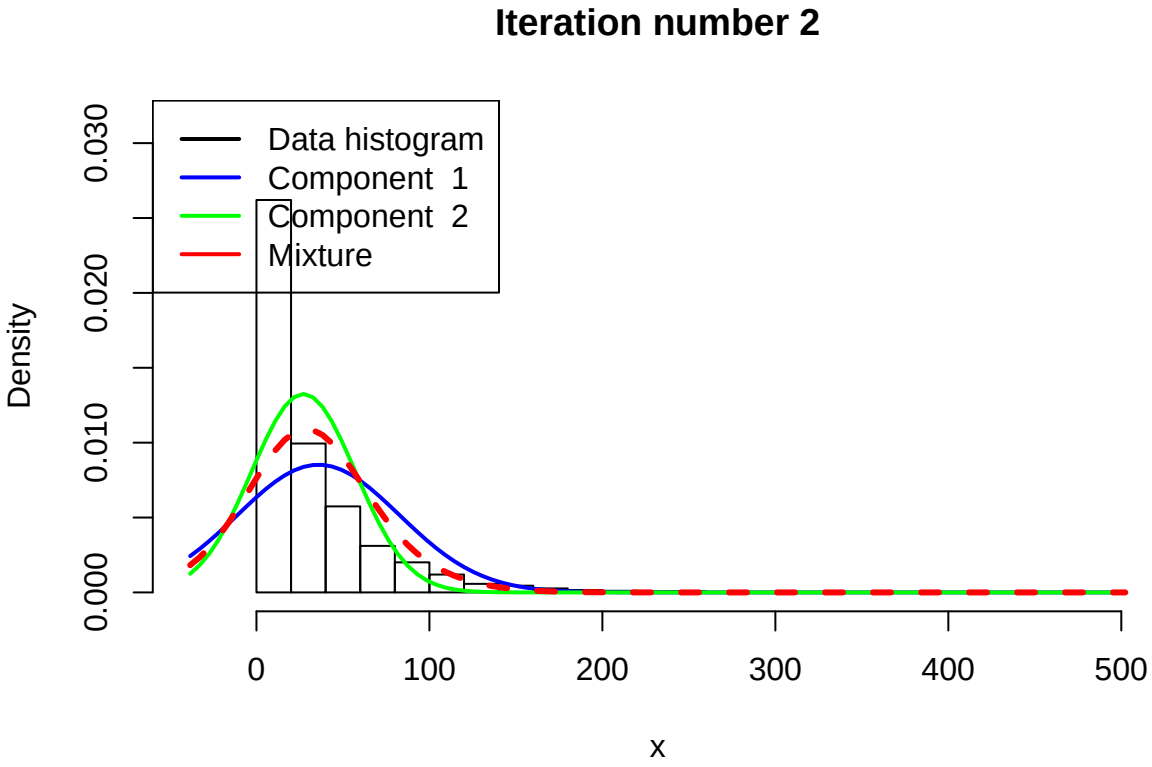
```
## [1] 3435 3484
```

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

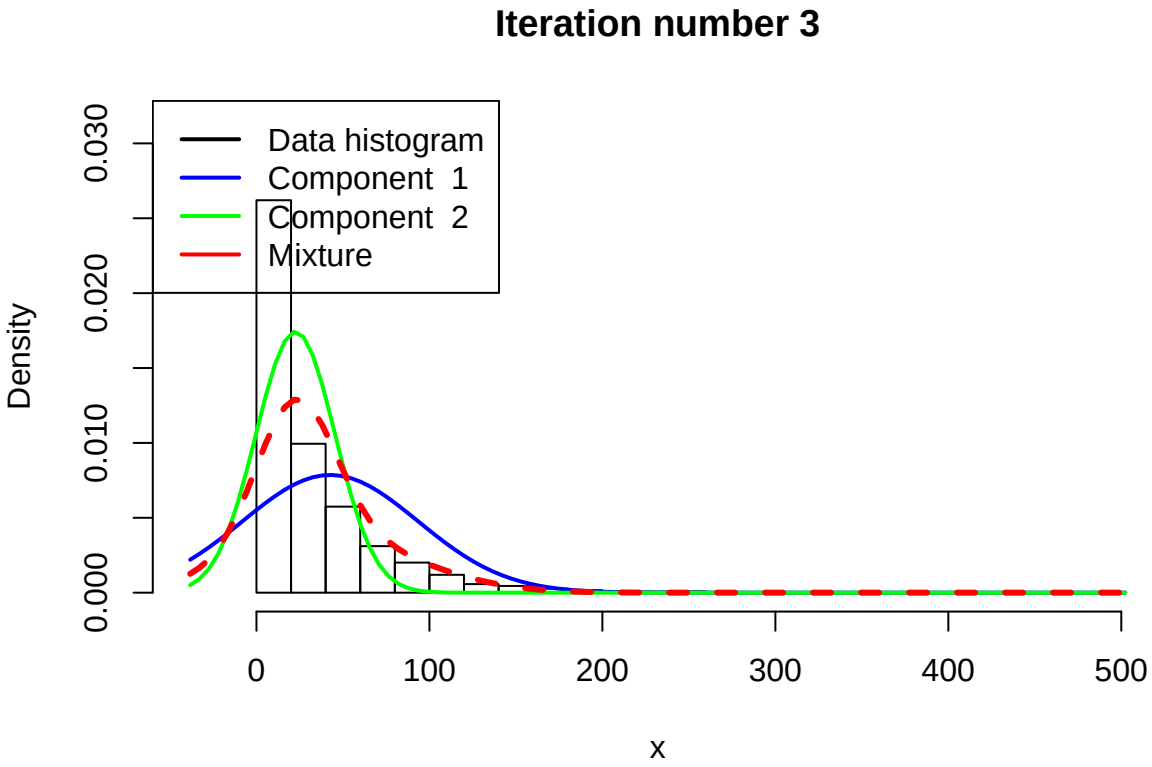



```
## [1] 3388 3531
```

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

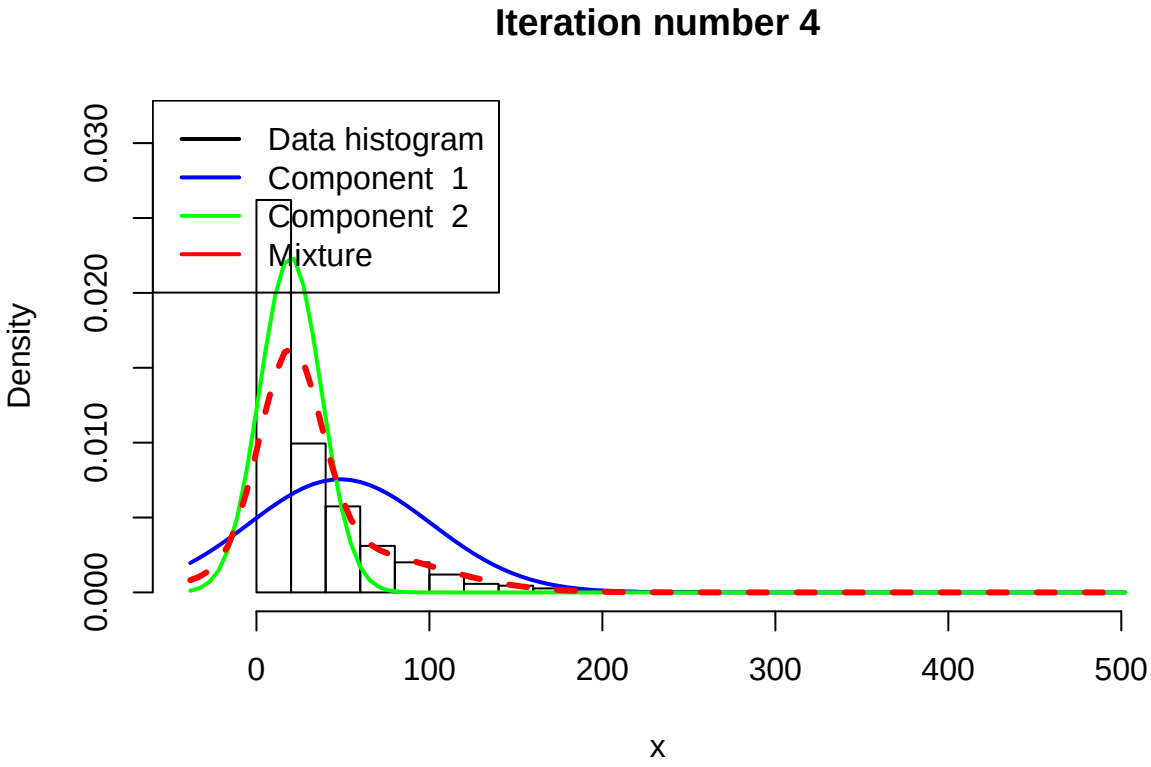



```
## [1] 3069 3850
## Iteration number: 4
```

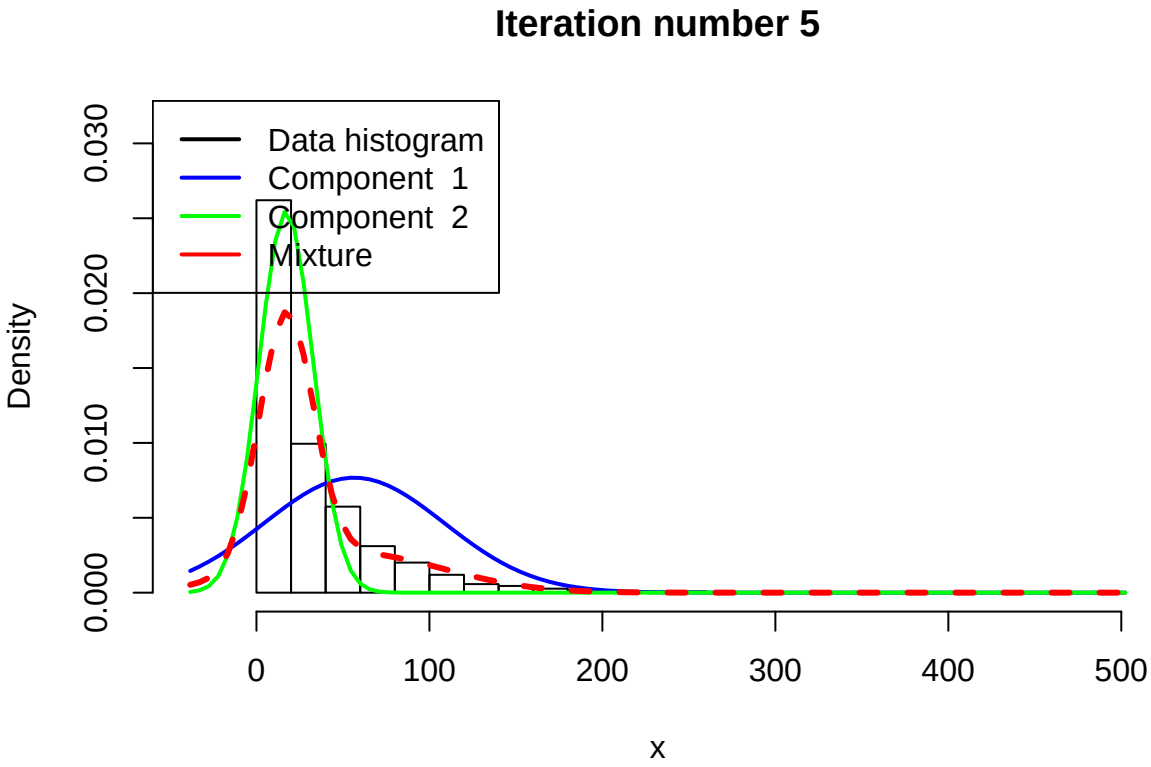



```
## [1] 2668 4251
```

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

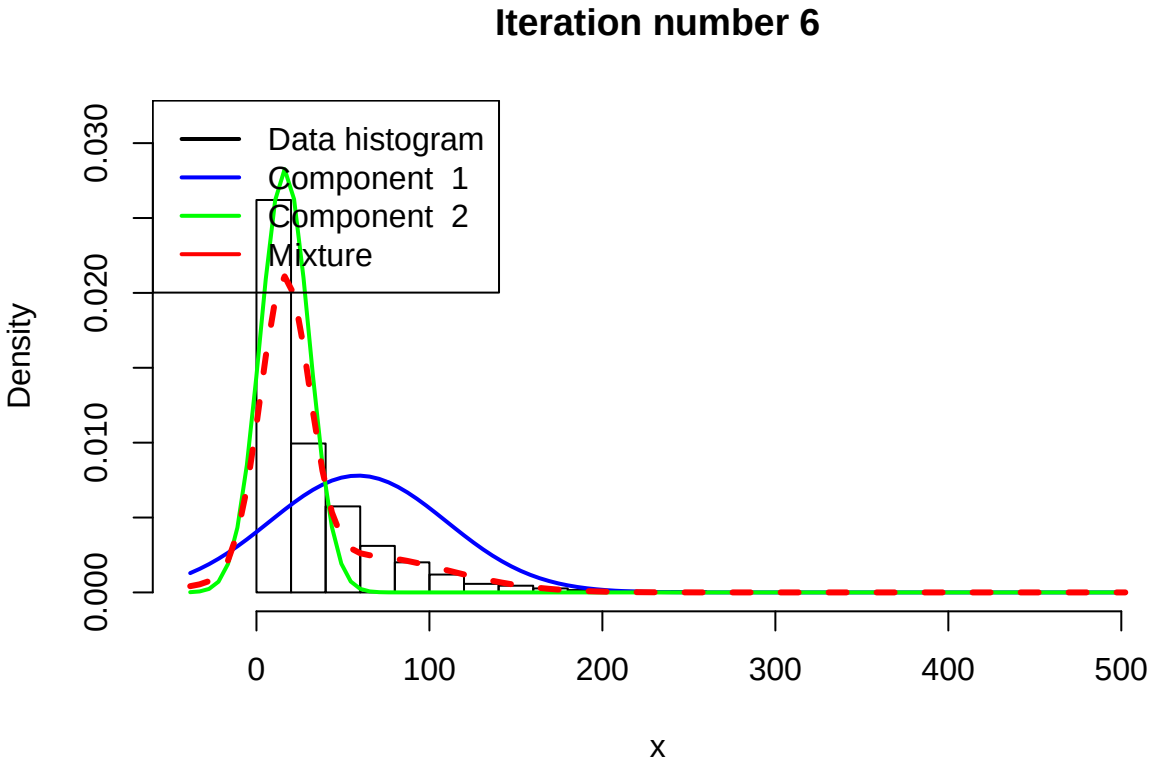



```
## [1] 2304 4615
## Iteration number: 6
```

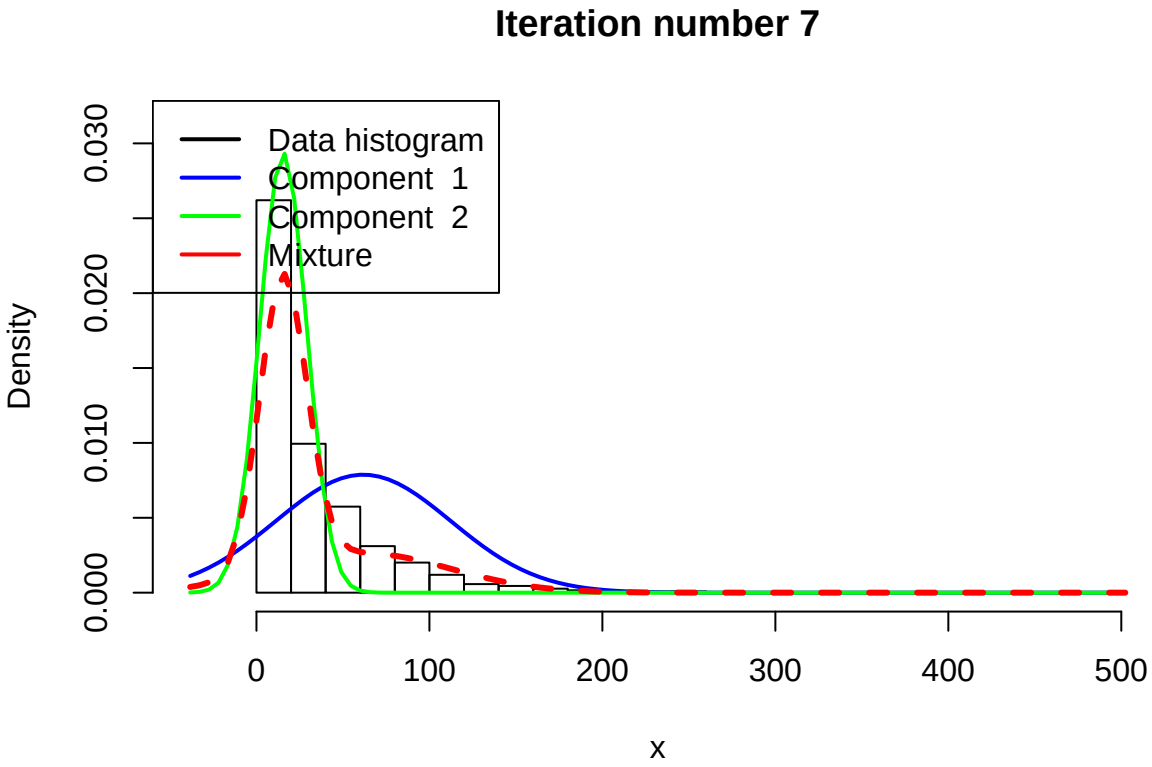



```
## [1] 2214 4705
```

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

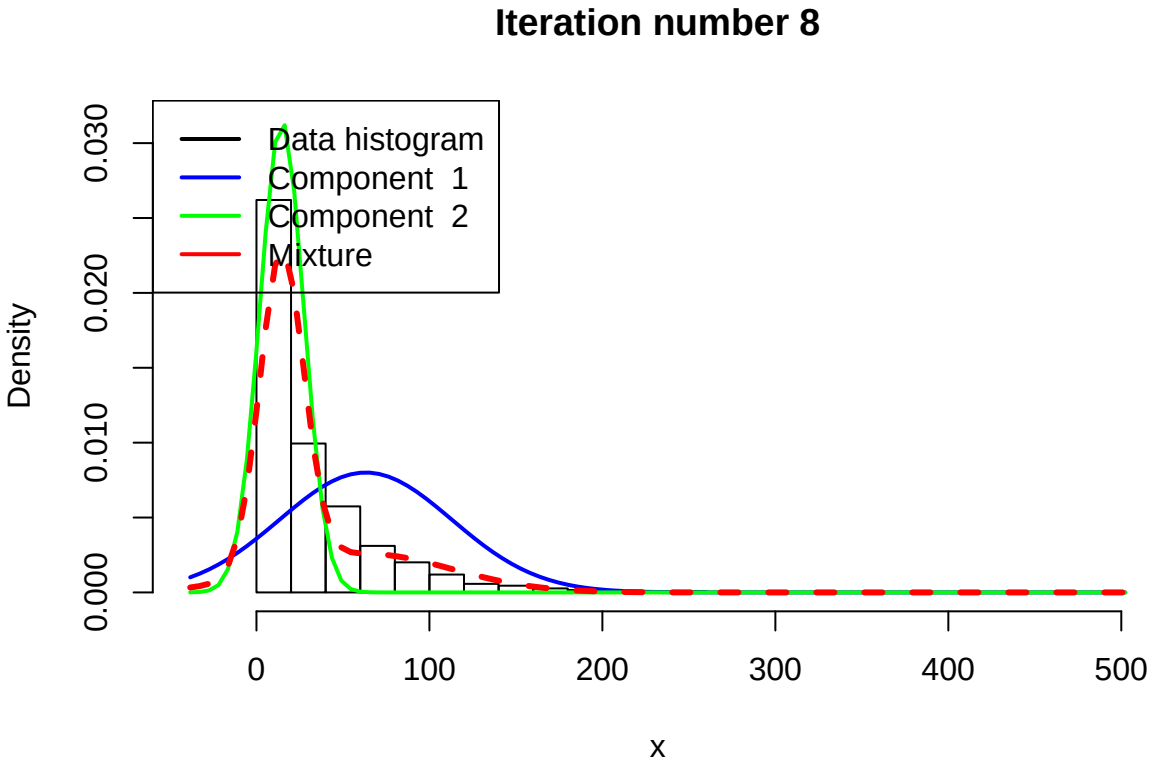



```
## [1] 2187 4732
## Iteration number: 8
```

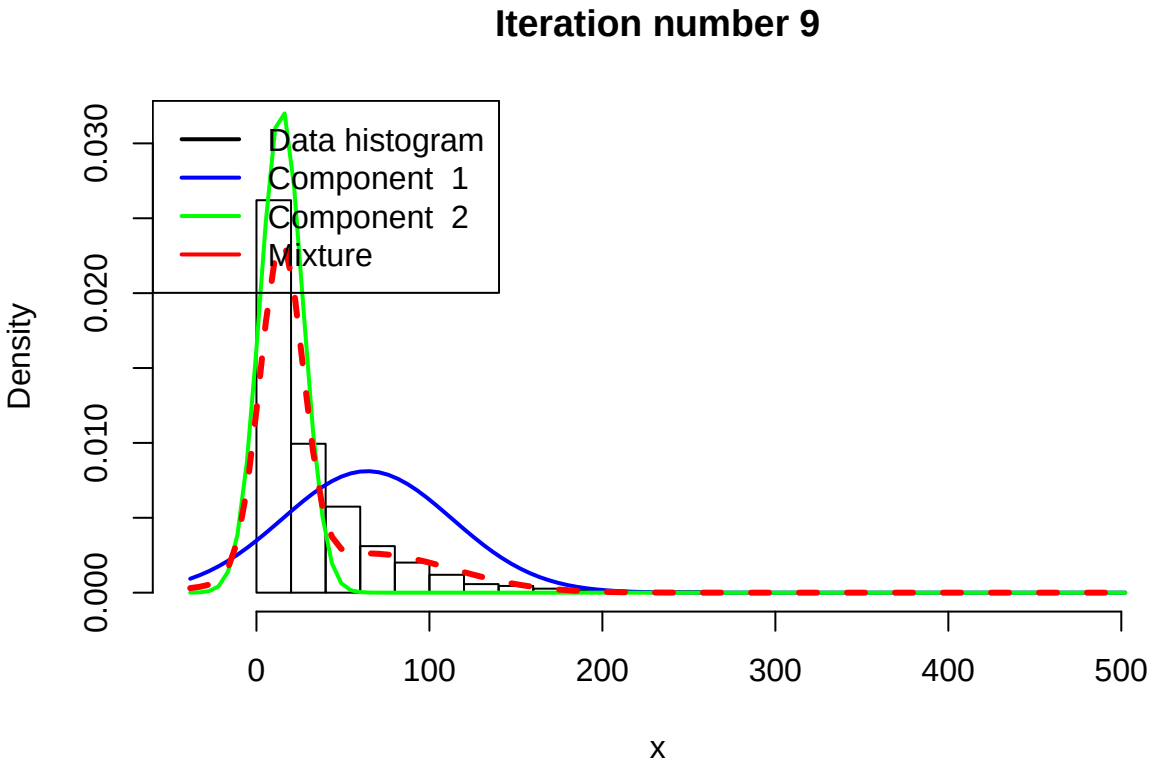



```
## [1] 2212 4707
```

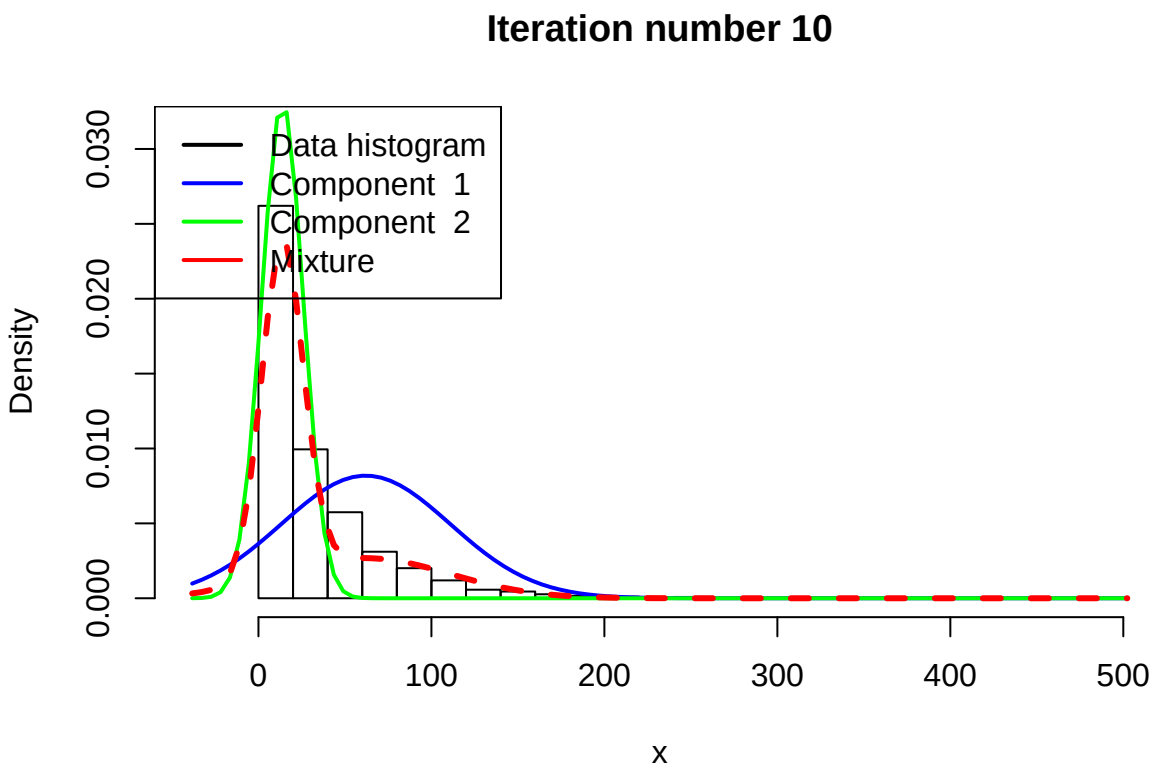
```
## Iteration number: 9
```



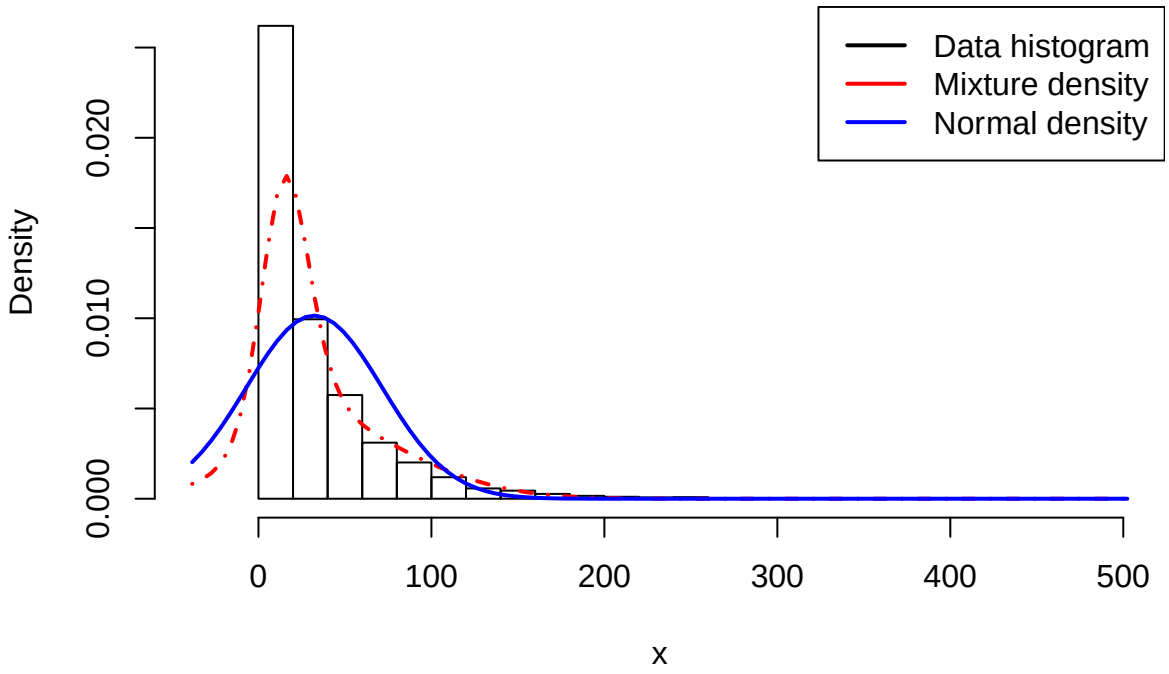

```
## [1] 2265 4654
## Iteration number: 10
```




```
## [1] 2307 4612
```




```
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")
lines(xGrid, mixDensMean, 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(
```



c

```
posterior_mean_mu <- mean(sample1$mu)
posterior_mean_sigma2 <- mean(sample1$sigma2)
final_mu <- c()

for (i in 1:nrow(rainfall)){
  final_mu[i] <- rnorm(1, mean = posterior_mean_mu, sd = sqrt(posterior_mean_sigma2))
}

final_df <- as.data.frame(cbind(final_mu, rainfall$rainfall))
colnames(final_df) <- c("final_mu", "rainfall")
```

Question 2

a

```
library(mvtnorm)
ebay <- read.table("ebayNumberOfBidderData.dat", header = TRUE)
ebay2 <- ebay
ebay <- ebay[,-2]
```

```
poisson <- glm(formula = nBids ~., data = ebay, family = "poisson")
summary(poisson)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = ebay)
##
## 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
```

b

```
# Data preparation:
X <- as.matrix(ebay2[,-1])
XtX <- t(X)%*%X
XtX_inv <- solve(XtX)
y <- ebay$nBids
covNames <- names(ebay)[2:ncol(ebay)]
```

```
# Prior parameters
mu <- rep(0, ncol(X))
PriorCov<- 100 * XtX_inv
```

```
# Function to optimize over
LogPostPoisson <- function(betaVect, y, X, mu, PriorCov){
  nPara <- length(betaVect)
  lambda <- exp(X%*%betaVect)

  logLik <- sum(-log(factorial(y)) + y * X%*%betaVect - lambda)

  if (abs(logLik) == Inf) logLik = -20000

  logPrior <- dmvnorm(betaVect, mean = mu, sigma =  PriorCov, log = TRUE)

  return(logLik + logPrior)
}
```

```
initVal <- rep(0, dim(X)[2])
logPost <- LogPostPoisson
```

```
OptimResults <- optim(initVal,
                      logPost,
                      gr = NULL,
                      method = c("BFGS"),
                      control = list(fnscale=-1),
                      hessian = TRUE,

                      y = y,
                      X = X,
                      mu = mu,
                      PriorCov = PriorCov
                      )

# We don't specify betaVect because this is the parameter we want to optimize over
```

```
PostMode <- OptimResults$par
postCov <- -solve(OptimResults$hessian)
names(PostMode) <- covNames
approxPostStd <- sqrt(diag(postCov))
names(approxPostStd) <- covNames
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(mvtnorm)
library(tidyr)
posteriors <- rmvnorm(2000, mean = PostMode, sigma = postCov) %>% data.frame()
```

c

Given:

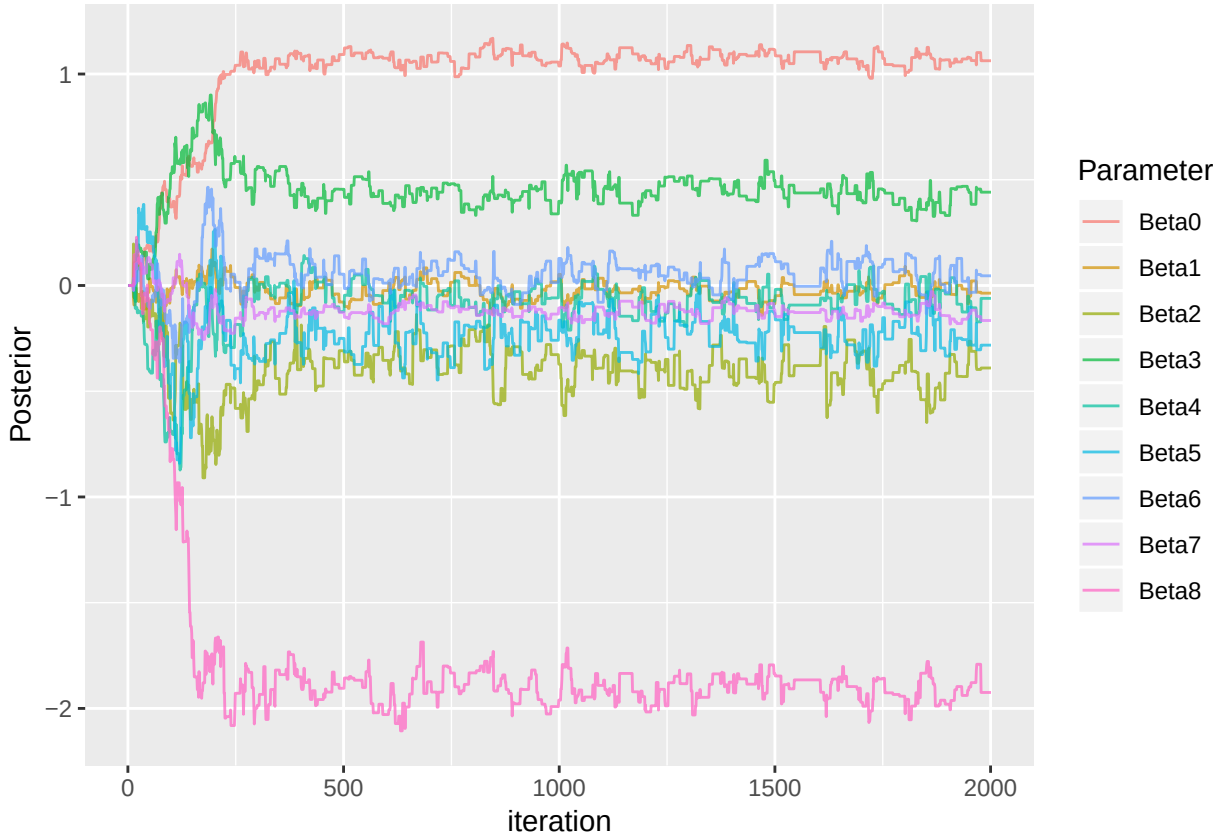
$$\frac{p(\theta_p|y)}{p(\theta^{(i-1)}|y)} = exp[log(p(\theta^{(i-1)})) - log(\theta^{(i-1)}|y)]$$

```r
metropolis_sampler <- function(theta, c, n, PostCov, wildcard, ...){

  npar <- length(theta)

  # Theta matrix
  thetas <- matrix(theta, nrow = n+1, ncol = npar)

  # Alpha vector
  alpha <- c()

  # Posterior density of current thetas
  current_posterior_density <- wildcard(theta, ...)

  for (i in 2:n+1){

    proposed_thetas <- rmvnorm(1, mean = thetas[i-1,], sigma = c*PostCov) %>% as.vector()
    proposed_posterior_density <- wildcard(proposed_thetas, ...)

    alpha[i] <- min(1, exp(proposed_posterior_density-current_posterior_density))
    probability <- runif(1)

    if (probability <= alpha[i]){
      thetas[i,] <- proposed_thetas
      current_posterior_density <- proposed_posterior_density

    } else {
      thetas[i,] <- thetas[i-1,]
    }
  }
```

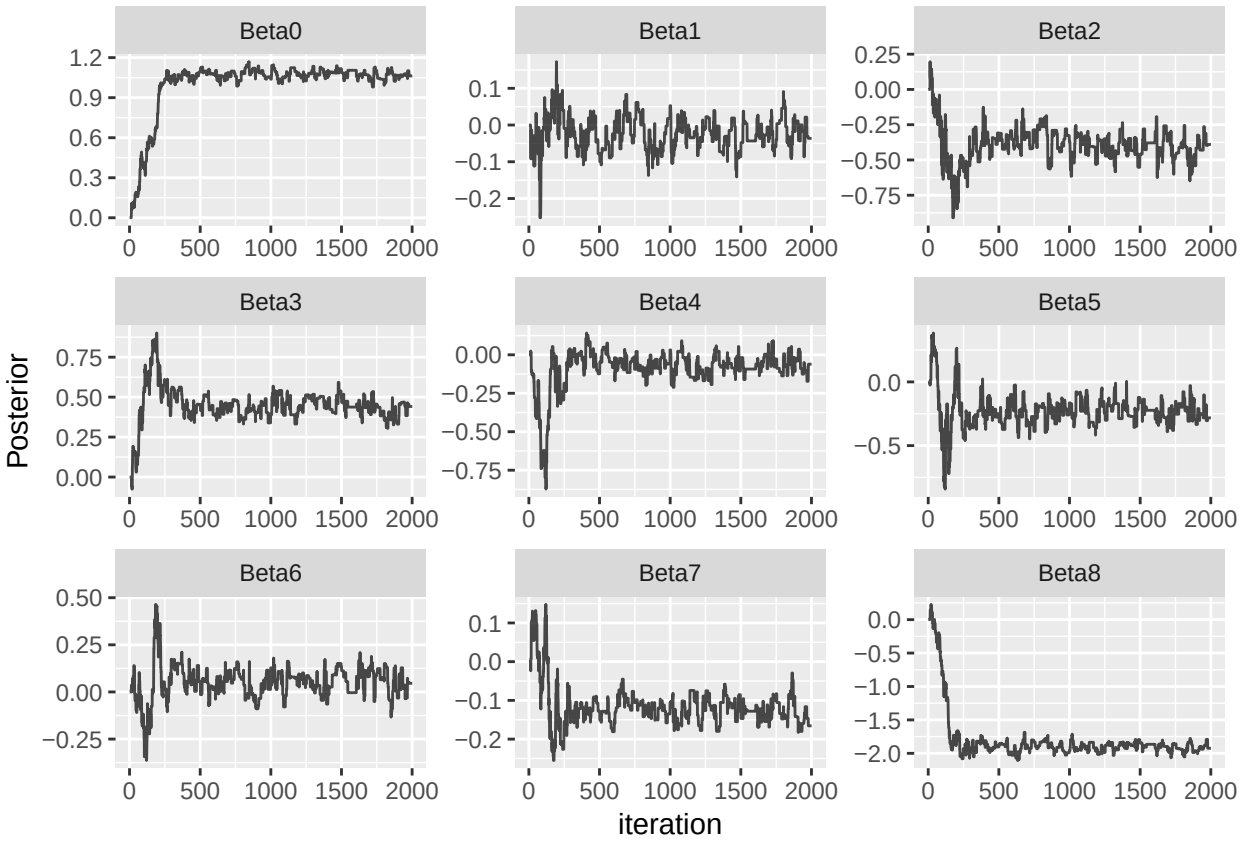
```
  return(thetas)
}
```

```
thetas <- metropolis_sampler(theta = rep(0, 9),
                             c = 1,
                             n = 2000,
                             PostCov = postCov,

                             wildcard = LogPostPoisson,
                             y=y,
                             X=X,
                             mu=mu,
                             PriorCov = PriorCov)

thetas <- as.data.frame(cbind(thetas, c(1:2001)))
colnames(thetas) <- c("Beta0","Beta1", "Beta2", "Beta3", "Beta4",
                      "Beta5", "Beta6", "Beta7", "Beta8", "iteration")
```

```
thetas %>% gather(., key = "Parameter", value = "Posterior", -iteration) %>%
  ggplot(., aes(iteration, Posterior, col=Parameter)) + geom_line(alpha=0.7)
```




```
thetas %>% gather(., key = "Parameter", value = "Posterior", -iteration) %>%
  ggplot(., aes(iteration, Posterior)) + geom_line(alpha=0.7) +
  facet_wrap(~Parameter, nrow = 3, scales = "free")
```


Beta0
Beta1
Beta2
Beta3
Beta4
Beta5
Beta6
Beta7
Beta8
Posterior
iteration