Bayesian Learning - Lab 03

Lakshidaa Saigiridharan (laksa656) and Maximilian Pfundstein (maxpf364)

2019-05-27

Contents

1 Normal Model, Mixture of Normal Model with Semi-Conjugate Prior

Exercise: The data `rainfall.dat` consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation (rain or snow in units of 1 inch, and records of zero 100 precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

a) Assume the daily precipitation $y_1, ..., y_n$ are independent normally distributed, $y_1, ..., y_n|\mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$

- Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2|y_1, ..., y_n)$. The full conditional posteriors are given on the slides from Lecture 7.
- Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

b) Let us now instead assume that the daily precipitation $y_1, ..., y_n$ follow an iid two-component **mixture of normals** model:

$$p(y_i|\mu, \sigma^2, \pi) = \pi\mathcal{N}(y_i|\mu_1, \sigma_1^2) + (1-\pi)\mathcal{N}(y_i|\mu_2, \sigma_2^2)$$

where

$$\mu = (\mu_1, \mu_2) \text{ and } \sigma^2 = (\sigma_1^2, \sigma_2^2)$$

Use the Gibbs sampling data augmentation algorithm in `NormalMixtureGibbs.R` (available under Lecture 7 on the course page) to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

c) Plot the following densities in one figure:

- A histogram or kernel density estimate of the data.
- Normal density $\mathcal{N}(\mu, \sigma^2)$ in (a)
- Mixture of normals density $p(y_i|\mu, \sigma^2, \pi)$ in (b)

Use the posterior mean value for all the parameters.

1.1 Normal Model

1.1.1 Gibbs Sampler Implementation

Comment from submission: 100 samples are quite few. Consider using e.g. 1000 samples.

Response: We did 100 samples so it's easier to see something in the plot.

First we load the data and set the parameters for our prior.

```r
################################################################################
# Exercise 1.a i)
################################################################################

rainfall_data = read.table("data/rainfall.dat", header=FALSE)

# Prior Parameters (slides page 15)
# These are basically random, guessed or from the data
# mu
mu_0 = 0
tau_sq_0 = 1

# sigma_sq
nu_0 = 1
sigma_sq_0 = 1 # sigma is 1/nu_0
```

This is a custom function to sample from the inverse chi squared distribution.

```r
crinvchisq = function(n, df, s_sq) {
  samples = rchisq(n, df)
  # These are draws from the inverse chi squared
  sigma_sq = (df - 1) * s_sq / samples
  return(sigma_sq)
}
```

This is the Gibbs Sampler, which takes the number of draws, a default σ (as both σ and μ depend on each other we need to start somewhere) and some more parameters to calculate the posterior parameters.

```r
gibbs_sample = function(nDraws, data, default_sigma, tau_sq_0, mu_0, nu_0,
                        sigma_sq_0) {

  # Posterior Parameters (Taken from lecture 2 slide 4)
  n = length(data)
  mu_n = mean(data) + mu_0
  nu_n = nu_0 + n
  default_sigma_sq = default_sigma^2

  # To store all iterative results
  values_df = data.frame(matrix(NA, nrow = nDraws, ncol = 2))

  # To save current iterative results
  values = list(mu = NaN, sigma_sq = default_sigma_sq)

  # As mu depends on sigma and sigma depends on mu, we need one initial value to start.
  # In this implementation the default value to start with is sigma_sq.

  for (i in 1:nDraws) {
    tau_sq_n = 1 / ((n/values$sigma) + (1/tau_sq_0))
    values$mu = rnorm(1, mu_n, sqrt(tau_sq_n))
    values$sigma_sq =
      crinvchisq(1, nu_n,(nu_0*sigma_sq_0 + sum((data - values$mu)^2))/(n + nu_0))
    values_df[i,] = values
  }

  colnames(values_df) = c("mu", "sigma_sq")
  return(values_df)
}
```

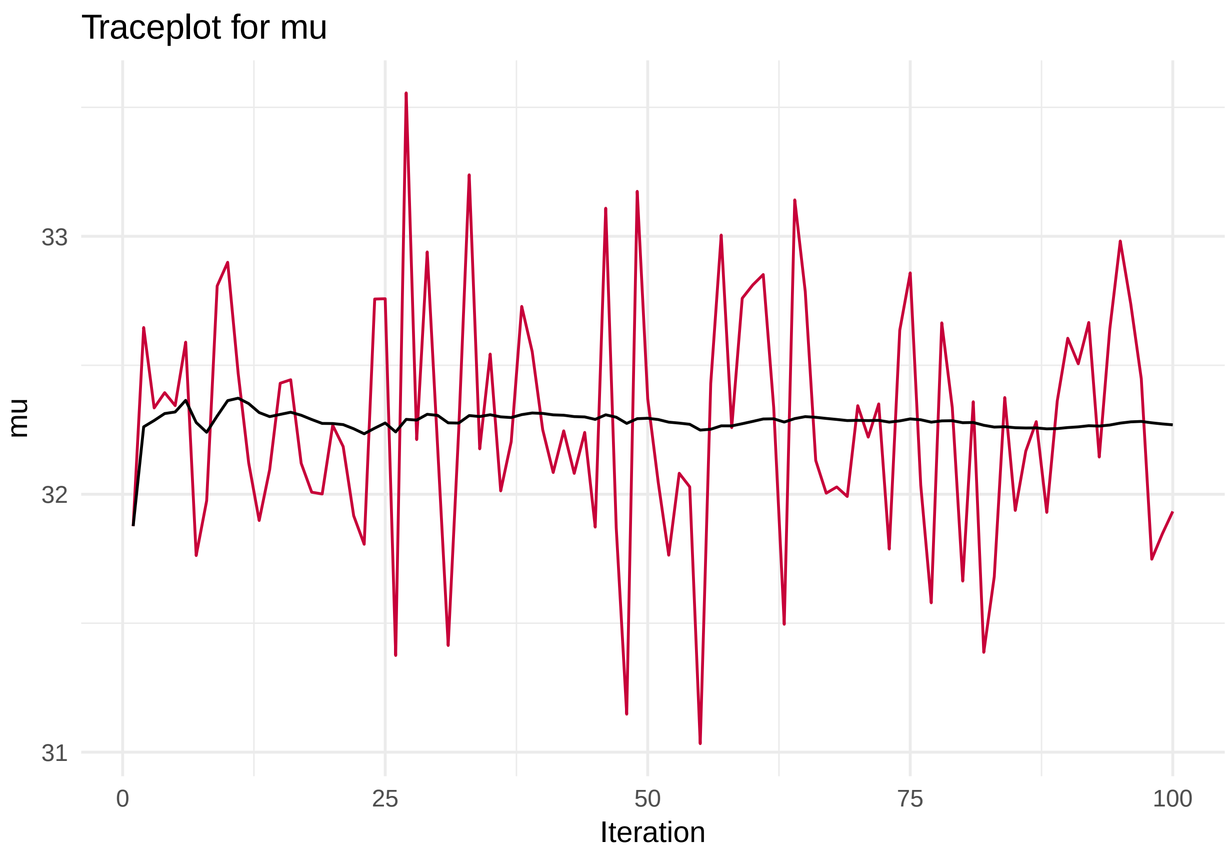
Now we call our Gibbs Sampler and print the draws.

```r
res = gibbs_sample(nDraws = 100,
                   data = rainfall_data$V1,
                   default_sigma = 40,
                   tau_sq_0 = tau_sq_0,
                   mu_0 = mu_0,
                   nu_0 = nu_0,
                   sigma_sq_0 = sigma_sq_0)

kable(head(res))
```

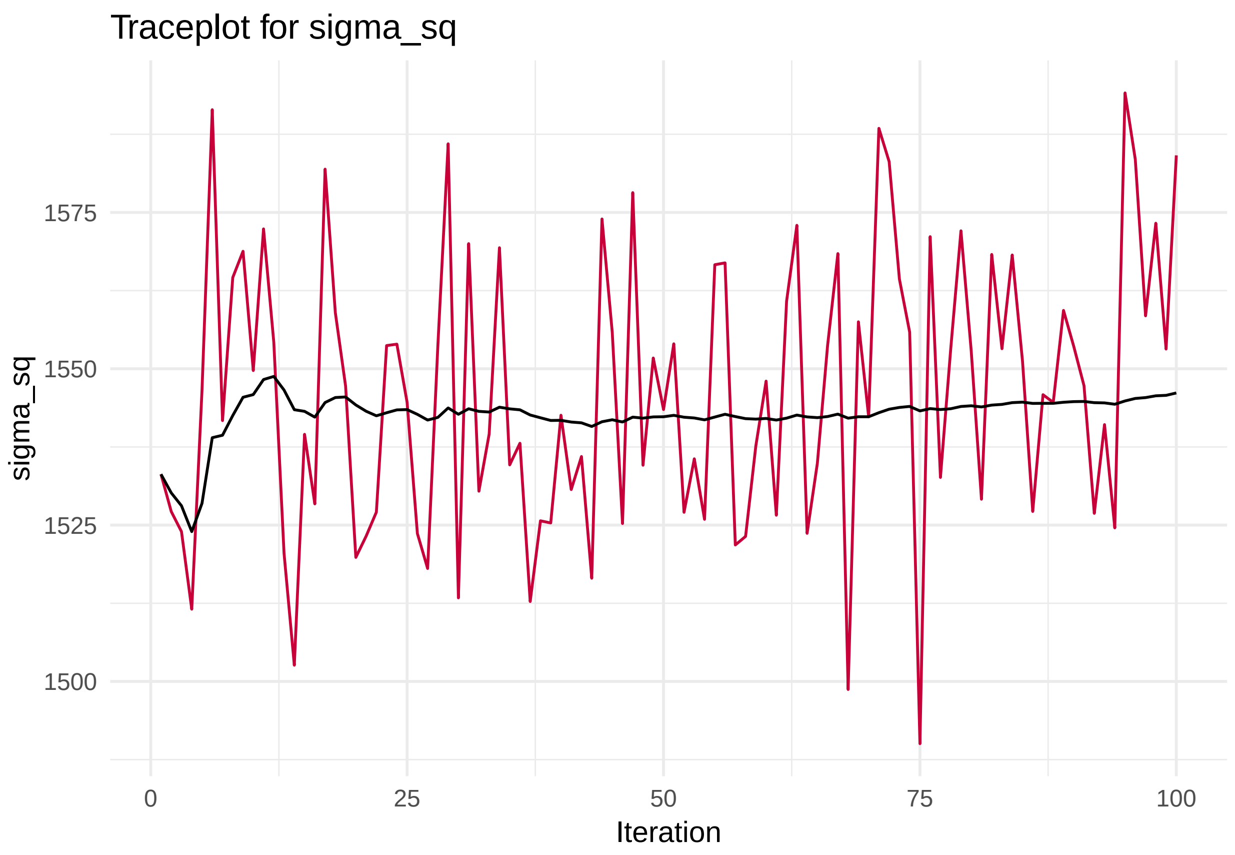
mu	sigma_sq
31.87645	1533.137
32.64637	1527.174
32.33453	1523.956
32.39383	1511.552
32.34361	1546.613
32.58967	1591.431

1.1.2 Analysing the Precipitation

This is the traceplot for μ including the trailing mean. We can see that the mean converges.



The same for σ^2.



1.2 Mixture Normal Model

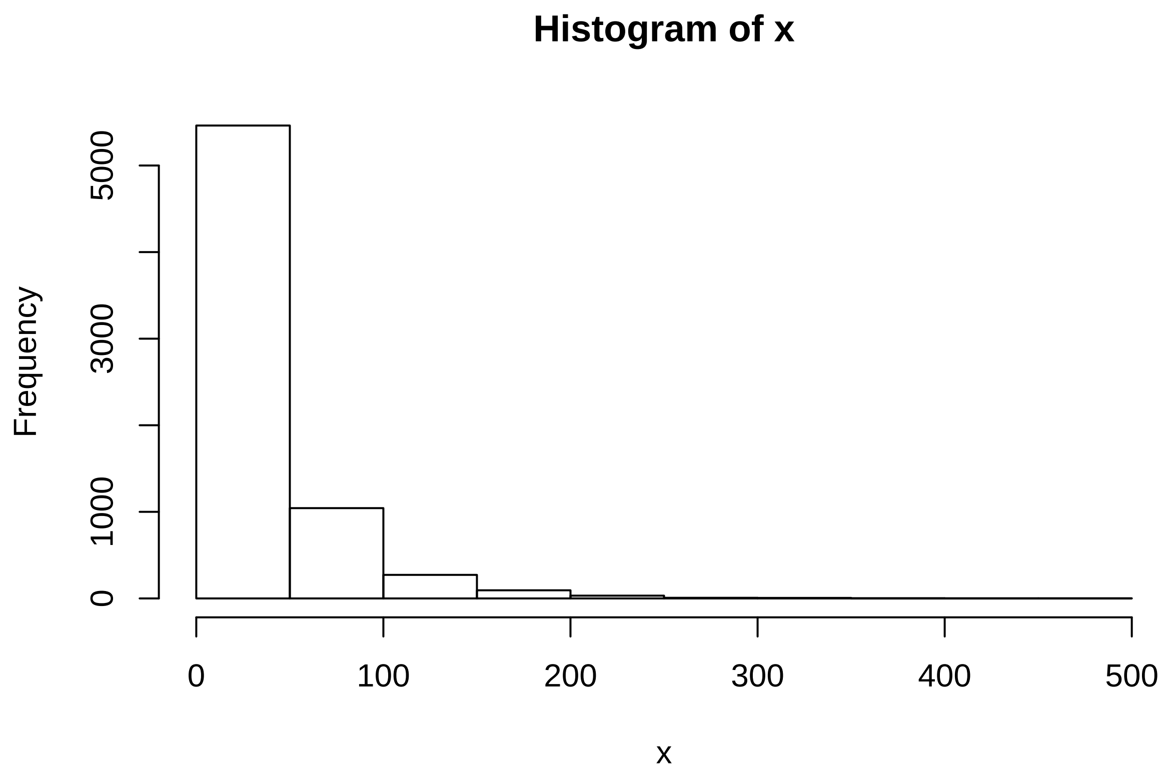
Comment from submission: Also consider plotting the posterior trajectories/means/histograms after removing the initial burnin period.

Response: The reason for keeping it was to show how it looks like.

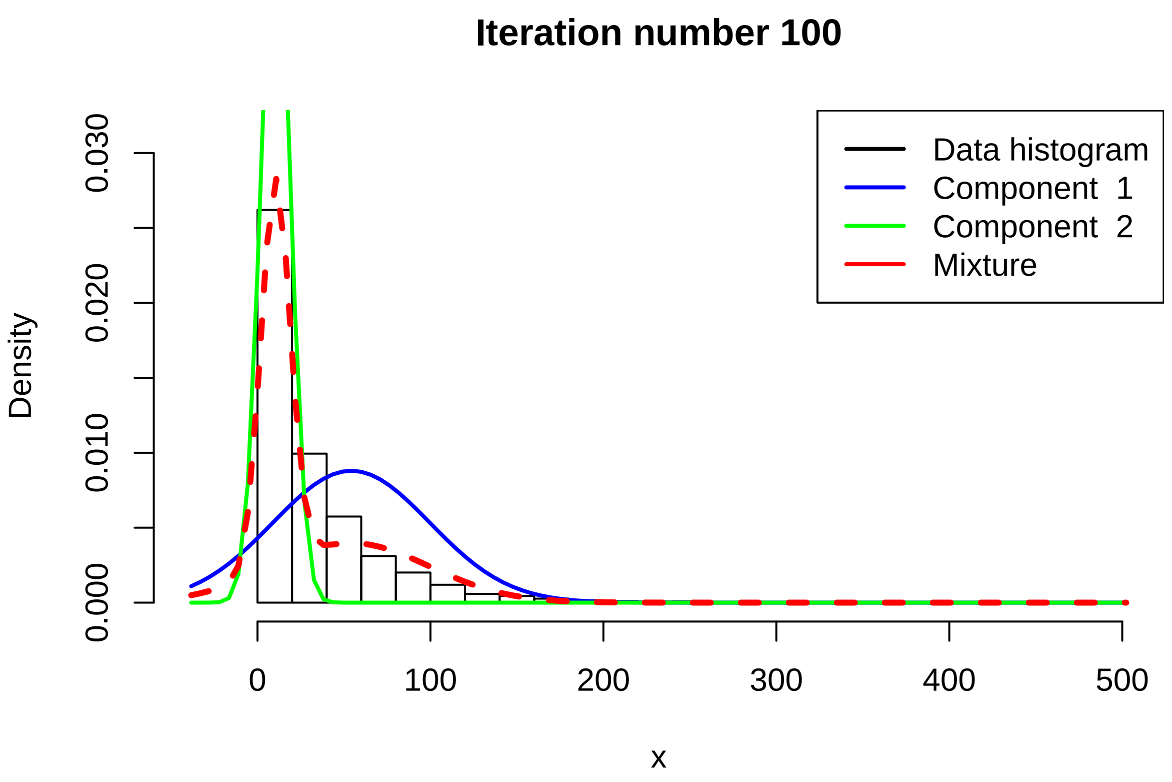
```
################################################################################
# Exercise 1.b)
################################################################################

# This is the script provided in 'NormalMixtureGibbs.R' with minor ajustments.

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

##########    BEGIN USER INPUT #################
# Data options
x <- as.matrix(rainfall_data$V1)

# Model options
## COMMENT: As the given model has two guassians we set this to two.
nComp <- 2    # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(0,nComp) # Prior mean of mu
```

```r
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 <- 100 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
# We removed the sleep time
sleepTime <- 0.1 # 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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
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)
```

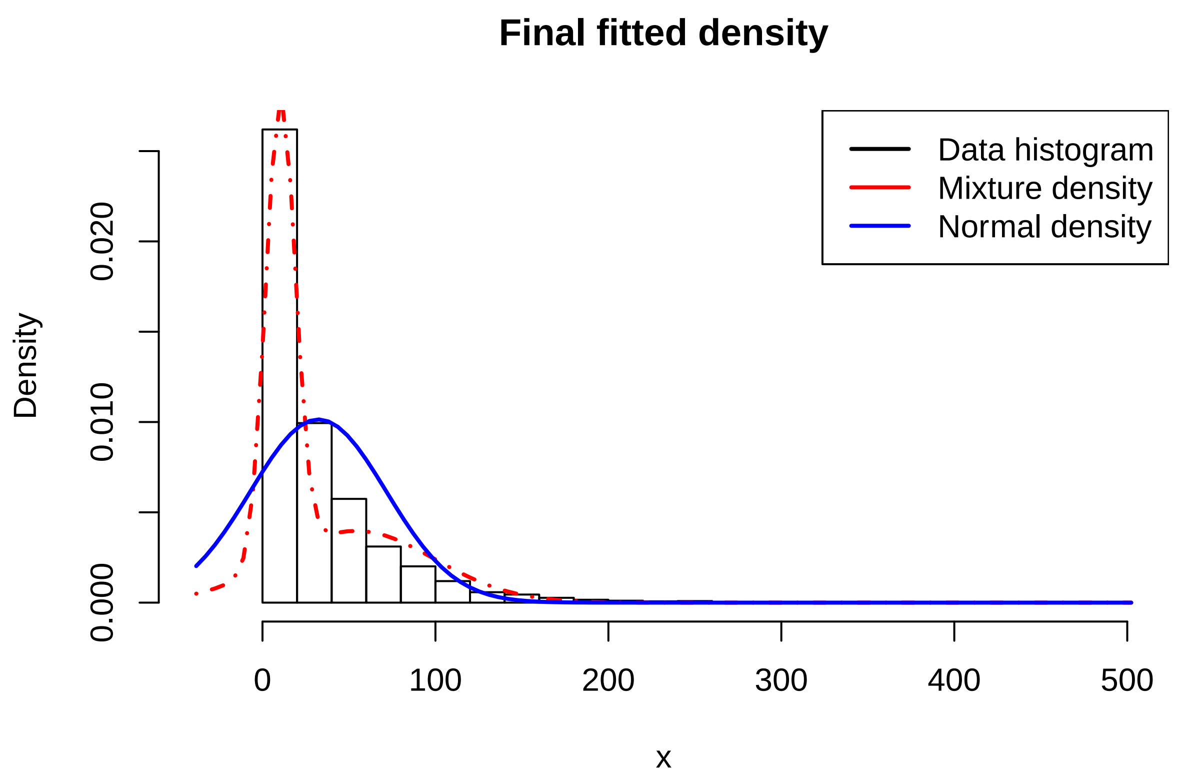
```r
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x)$density))
```



```r
# Added for stroring the draws and print the plots
all_mu = matrix(rep(NaN, nIter*nComp), nrow = nIter)
all_sigma_sq = matrix(rep(NaN, nIter*nComp), nrow = nIter)

for (k in 1:nIter){
  #message(paste('Iteration number:',k))
  # Just a function that converts between different representations of the
  # group allocations
  alloc <- S2alloc(S)
  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 == j] -
                                        mu[j])^2))/(nu0[j] +
                                        nAlloc[j]))
  }

    all_mu[k,] = mu
    all_sigma_sq[k,] = sigma2

  # 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 == nIter)){
    effIterCount <- effIterCount + 1
    hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
         main = paste("Iteration number",k), ylim = ylim)
    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)
  }
}
```

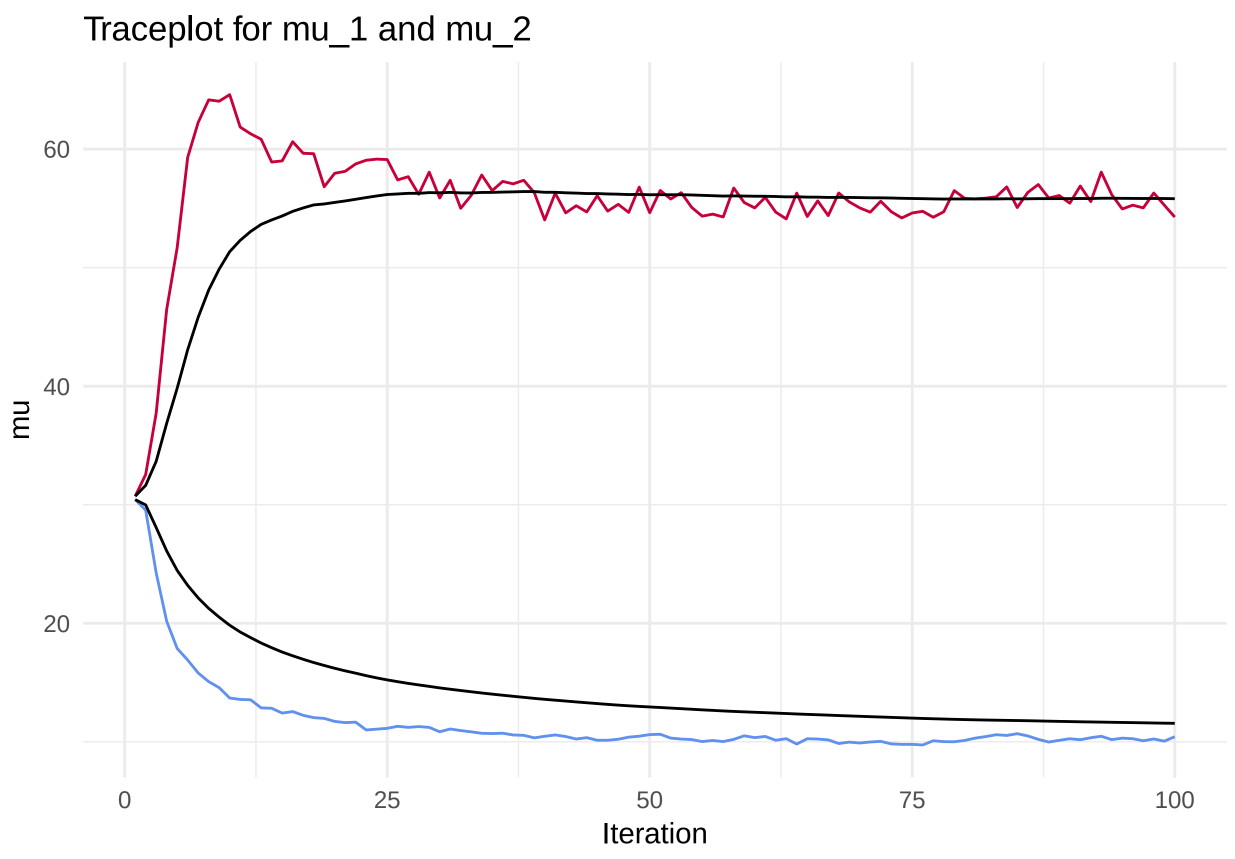


```r
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
     main = "Final fitted density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l",
      lwd = 2, col = "blue")
legend("topright", box.lty = 1,
       legend = c("Data histogram","Mixture density", "Normal density"),
       col = c("black","red","blue"), lwd = 2)
```

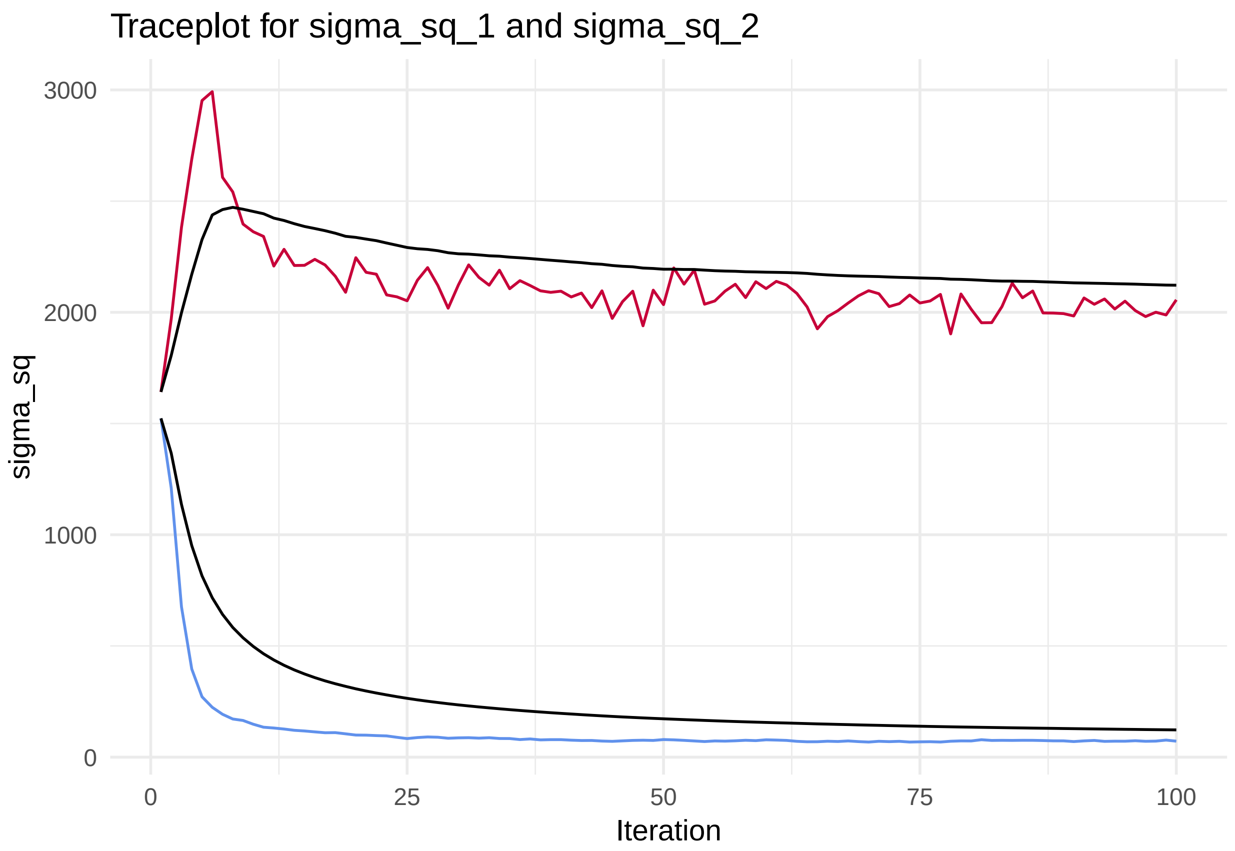


```
########################    Helper functions    ##############################
```

This is the traceplot for μ.



This is the traceplot for σ.



Parameters of the Prior: Without looking at the data we have no clear idea of the priors should look like. We can assume that probably the mean is greater than 0, but about the variance or α we do not really have an idea. That's why we leave the parameters as they are. The default implementation actially takes the variance of the data for the prior, where we think that's not 100 percent appropriate but it depends on the problems one tackles in the real world of course. In addition it shouldn't matter that much because after the burn-in period the prior shouldn't matter any more to discover the posterior density function.

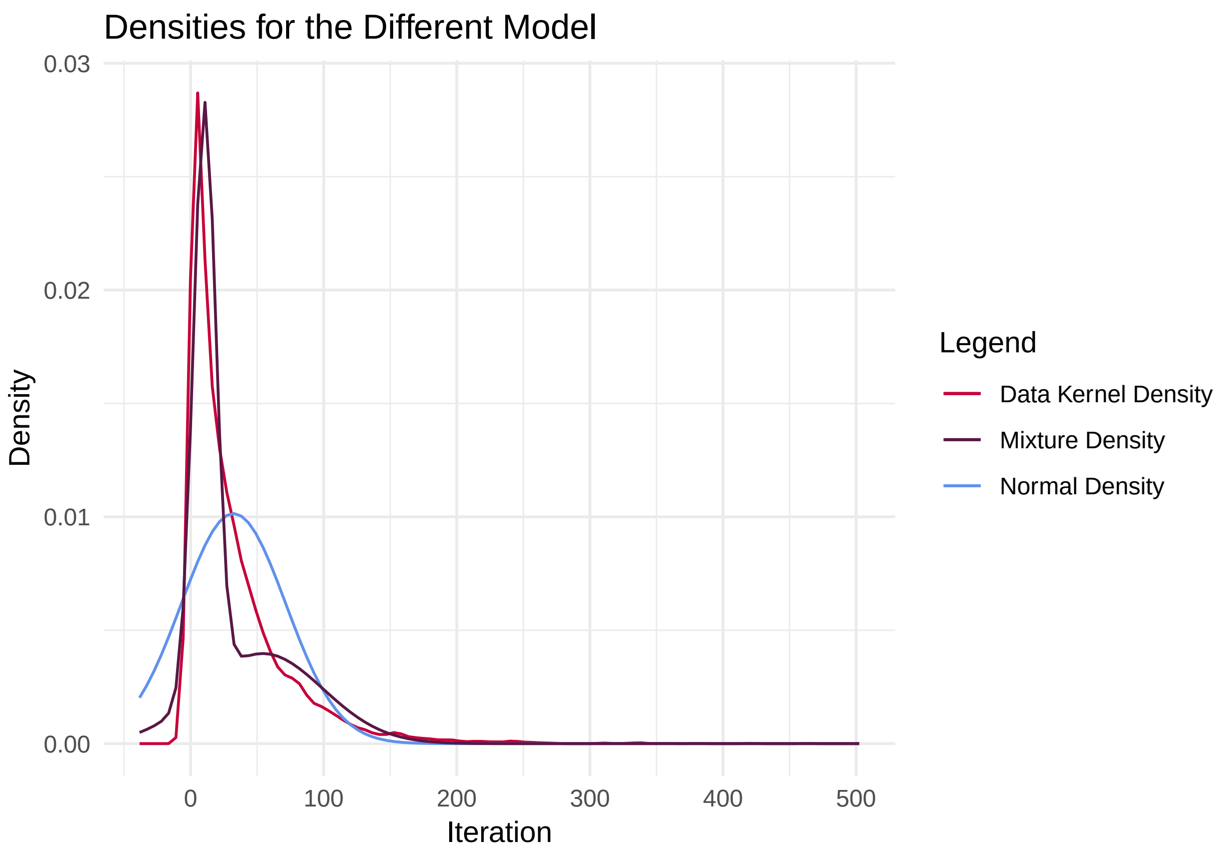
Convergence: As we can see from the different trace plots the parameters actually converge the same as in the previous exercise. More iterations will make the result more accurate but we set it to a low amount of iterations to have a better zoom on the start of the process to observe the burn-in period.

1.3 Graphical Comparison

Comment from submission: Your mixture density is plotted using the default output from Mattias' code which computes the mean of the mixture densities computed at each Gibbs iteration. This is ok, but not exactly the same as the density requested in the problem, which is based on the posterior mean of the parameters.

Response: Okay, thanks for the clarification!

The following plot shows the densities of the data and the two models.



2 Metropolis Random Walk for Poisson Regression

Exercise: Consider the following Poisson regression model

$$y_i|\beta \sim \text{Poisson}\left[\exp(x_i^T\beta)\right], i = 1, ..., n$$

where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set `eBayNumberOfBidderData.dat`. This dataset contains observations from 1000 eBay auctions of coins. The response variable is **nBids** and records the number of bids in each auction. The remaining variables are features/covariates ($\mathbf{x}$):

- **Const** (for the intercept)
- **PowerSeller** (is the seller selling large volumes on eBay?)
- **VerifyID** (is the seller verified by eBay?)
- **Sealed** (was the coin sold sealed in never opened envelope?)
- **MinBlem** (did the coin have a minor defect?)
- **MajBlem** (a major defect?)
- **LargNeg** (did the seller get a lot of negative feedback from customers?)
- **LogBook** (logarithm of the coins book value according to expert sellers. Stan- dardized)
- **MinBidShare** (a variable that measures ratio of the minimum selling price (starting price) to the book value. Standardized).

a) Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: `glm.R`, don't forget that `glm()` adds its own intercept so don't input the covariate Const]. Which covariates are significant?

b) Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}[\mathbf{0}, 100 \cdot (X^T X)^{-1}]$ where $\mathbf{X}$ is the $n \times p$ covariate matrix. This is a commonly used prior which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:

$$\beta|y \sim \mathcal{N}[\tilde{\beta}, J_y^{-1}(\tilde{\beta})],$$

where $\tilde{\beta}$ is the posterior mode and $J_y(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_y(\tilde{\beta})$ an be obtained by numerical optimization (`optim.R`) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

c) Now, let's simulate from the actual posterior of β using the Metropolis algorithm and compare with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general function for any model, I will denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta^{(i-1)} \sim \mathcal{N}(\theta^{(i-1)}, c \cdot \Sigma),$$

where $\Sigma = J_y^{-1}(\tilde{\beta})$ obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across *function objects* in R and the triple dot (...) wildcard argument. I have posted a note (`HowToCodeRWM.pdf`) on the course web page that describes how to do thisin R.

Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

2.1 Maximum Likelihood Estimator

First we load the data and have a look at it.

```r
################################################################################
# Exercise 2.a)
################################################################################

ebay_data = read.table("data/eBayNumberOfBidderData.dat", header=TRUE)
kable(head(ebay_data))
```

nBids	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
2	1	0	0	0	0	0	0	-0.2237	-0.2088
6	1	1	0	0	0	0	0	0.6073	-0.3478
1	1	1	0	0	0	0	0	0.0332	0.4423
1	1	0	0	0	1	0	0	0.3755	0.1441
4	1	0	0	0	0	0	1	1.4347	-0.4104
2	1	0	0	0	0	0	0	-0.9142	0.6318

Now we fit the glm model and look at the obtained coefficients.

```r
glm_model = glm(formula = nBids ~ ., data = ebay_data[,-2], family = poisson)
summary(glm_model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = ebay_data[,
##     -2])
##
## 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
```

```
glm_model$coefficients
```

```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07244206 -0.02054076 -0.39451647  0.44384257 -0.05219829 -0.22087119
##     LargNeg     LogBook MinBidShare
##  0.07067246 -0.12067761 -1.89409664
```

The covariates *Intercept*, *VerifyID*, *Sealed*, *LogBook* and *MinBidShare* are significant.

2.2 Bayesian Analysis of the Poisson Regression

The prior is given as:

$$P(\theta) = \mathcal{N}(\mathbf{0}, 100 \cdot (X^T X)^{-1})$$

The likelihood is derived by taking the product of the probabilities of the observations. For calculating the probability for each observation the probability mass function is used, which is in it's general form:

$$p(y|x;\theta) = \frac{\lambda^y}{y!}e^{-\lambda} = \frac{e^{y\theta}e^{-e^{\theta x}}}{y!}$$

Using this, the likelihood is given by:

$$P(X,y|\theta) = L(\theta|X,y) = \prod_{i=1}^{N} \frac{e^{y_i\theta} e^{-e^{\theta x_i}}}{y_i!}$$

Taking the log gives:

$$\mathcal{L}(\theta|X,y) = \sum_{i=1}^{N} y_i\theta - e^{\theta x_i} - ln(y_i!)$$

As we don't care about factors not depending on θ we can rewrite as:

$$\mathcal{L}(\theta|X,y) \propto \sum_{i=1}^{N} y_i\theta - e^{\theta x_i}$$

And finally we vectorize the left part (dot product):

$$\mathcal{L}(\theta|X,y) \propto y \cdot \theta - \sum_{i=1}^{N} e^{\theta x_i}$$

The posterior log-likelihood also contains the log normal part, which can be found in the code.

```r
################################################################################
# Exercise 2.b)
################################################################################

# Parameters
Y = as.matrix(ebay_data[,1])
# We take all covariates
X = as.matrix(ebay_data[,-1])
# Feature names
feature_names =  colnames(ebay_data[,2:ncol(ebay_data)])
colnames(X) = feature_names

# Defining the prior parameters
mu_prior = rep(0, ncol(ebay_data) - 1)
covariate_prior = 100 * solve(t(X) %*% X)
# An we need initial beta parameters
beta = rmvnorm(1, mu_prior, covariate_prior)

# Cost-Function
## We need cost function which will be optimized (e.g. the likelihood).
## The first parameters has to be the regression coefficient
## The function will calculate the log-likelihood and the log-prior for getting
## the posterior-log-likelihood which is to be optimized
## We use two sub-functions for the two logs

# https://nptel.ac.in/courses/111104074/Module14/Lecture41.pdf
log_likelihood = function(beta, X ,Y) {
  lambda = t(X %*% beta) # log-link function
  llik = t(Y) %*% t(lambda) - sum(exp(lambda))# - sum(log(factorial(Y)))
  return(llik)
}
```

```r
posterior_log_likelihood = function(beta, X, Y, sigma, mu) {
  return(dmvnorm(beta, mu, sigma, log = TRUE) + log_likelihood(beta, X, Y))
}

res = optim(beta, posterior_log_likelihood, method = "BFGS",
            control = list(fnscale = -1), hessian = TRUE,
            X = X, Y = Y, sigma = covariate_prior, mu = mu_prior)

# Now we use the results to extract the desired values and same them to
# variables with a speaking name
posterior_mode = as.vector(res$par)
names(posterior_mode) = feature_names
posterior_covariance = - solve(res$hessian)
posterior_sd = sqrt(diag(posterior_covariance))
names(posterior_sd) = feature_names
```

The posterior mode is given as:

```r
posterior_mode
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem 
##  1.06984046 -0.02050368 -0.39303987  0.44355373 -0.05252398 -0.22125486 
##     LargNeg     LogBook MinBidShare 
##  0.07069804 -0.12021443 -1.89200096
```

The posterior covariance is given as:

```r
posterior_covariance
```

```
##                [,1]          [,2]          [,3]          [,4]
##  [1,]  9.454651e-04 -7.138993e-04 -2.741521e-04 -2.708986e-04
##  [2,] -7.138993e-04  1.353077e-03  4.024742e-05 -2.949012e-04
##  [3,] -2.741521e-04  4.024742e-05  8.515599e-03 -7.824866e-04
##  [4,] -2.708986e-04 -2.949012e-04 -7.824866e-04  2.557770e-03
##  [5,] -4.454535e-04  1.142950e-04 -1.013616e-04  3.577135e-04
##  [6,] -2.772198e-04 -2.082692e-04  2.282530e-04  4.532305e-04
##  [7,] -5.128355e-04  2.801782e-04  3.313562e-04  3.376468e-04
##  [8,]  6.436677e-05  1.181878e-04 -3.191854e-04 -1.311053e-04
##  [9,]  1.109945e-03 -5.685687e-04 -4.292841e-04 -5.759669e-05
##                [,5]          [,6]          [,7]          [,8]
##  [1,] -4.454535e-04 -2.772198e-04 -5.128355e-04  6.436677e-05
##  [2,]  1.142950e-04 -2.082692e-04  2.801782e-04  1.181878e-04
##  [3,] -1.013616e-04  2.282530e-04  3.313562e-04 -3.191854e-04
##  [4,]  3.577135e-04  4.532305e-04  3.376468e-04 -1.311053e-04
##  [5,]  3.624775e-03  3.492332e-04  5.844054e-05  5.853857e-05
##  [6,]  3.492332e-04  8.365114e-03  4.048636e-04 -8.975945e-05
##  [7,]  5.844054e-05  4.048636e-04  3.175052e-03 -2.541763e-04
##  [8,]  5.853857e-05 -8.975945e-05 -2.541763e-04  8.384714e-04
##  [9,] -6.437321e-05  2.622256e-04 -1.063208e-04  1.037432e-03
##                [,9]
##  [1,]  1.109945e-03
##  [2,] -5.685687e-04
##  [3,] -4.292841e-04
##  [4,] -5.759669e-05
##  [5,] -6.437321e-05
##  [6,]  2.622256e-04
##  [7,] -1.063208e-04
##  [8,]  1.037432e-03
##  [9,]  5.054785e-03
```

The posterior standard deviation is given as:

```
posterior_sd
```

```
##      Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
## 0.03074842  0.03678419  0.09228000  0.05057440  0.06020611  0.09146100
##    LargNeg     LogBook MinBidShare
## 0.05634760  0.02895637  0.07109701
```

2.3 Simulate from the Actual Posterior Using the Metropolis Algorithm

First we define the function that is performing the Metropolis Random Walk. It takes the posterior function as an argument, the arguments for this function are taken with the `...`. `n` specifies the number of samples to be obtained. θ_0 are the start values for the parameters. We used 0 for all of them but they could also be set to our prior believes as mentioned during the lecture. As we want to see the burn-in period, we went for the 0's.

```
################################################################################
# Exercise 2.c)
################################################################################

RMWSampler = function(dposterior, n = 1, theta_0, c, Sigma, ...) {

  # Variables
  samples = data.frame()
  Sigma = c * Sigma
  i = 0

  # Current and next step for random walk
  theta_1 = theta_0
  theta = theta_0

  while (nrow(samples) < n) {
    # Get proposal
    theta_star = rmvnorm(n = 1, mean = theta_1, sigma = Sigma)

    # According to the help file we go for this version of alpha
    alpha = min(1, exp(dposterior(theta_star, ...) - dposterior(theta_1, ...)))

    # Decide if to accept or reject the proposal
    u = runif(n = 1, min = 0, max = 1)

    if (u <= alpha) {
      # Accept
      theta_1 = theta
      theta = theta_star
      samples = rbind(samples, theta_star)
    }
    else {
      # Reject
      theta = theta_1
    }

    # To keep track of the rejection rate
    i = i + 1
  }

  return(list(samples = samples, rejection_rate = (1 - (n / i))))
}
```

The proportional log posterior is given as prior plus likelihood, as previously defined:

$$log(P(\beta|X, y)) \propto log(\mathcal{N}(\mathbf{0}, 100 \cdot (X^T X)^{-1})) + y \cdot \theta - \sum_{i=1}^{N} e^{\theta x_i}$$

θ is the first argument to our function. Also needed are the prior parameters and the data for the likelihood.

```
dposterior = function(theta, mu_prior, sigma_prior, X, Y) {

  # Prior
  prior = dmvnorm(theta, mu_prior, sigma_prior, log = TRUE)

  # Likelihood
  likelihood = log_likelihood(t(theta), X, Y)

  return(prior + likelihood)
}
```

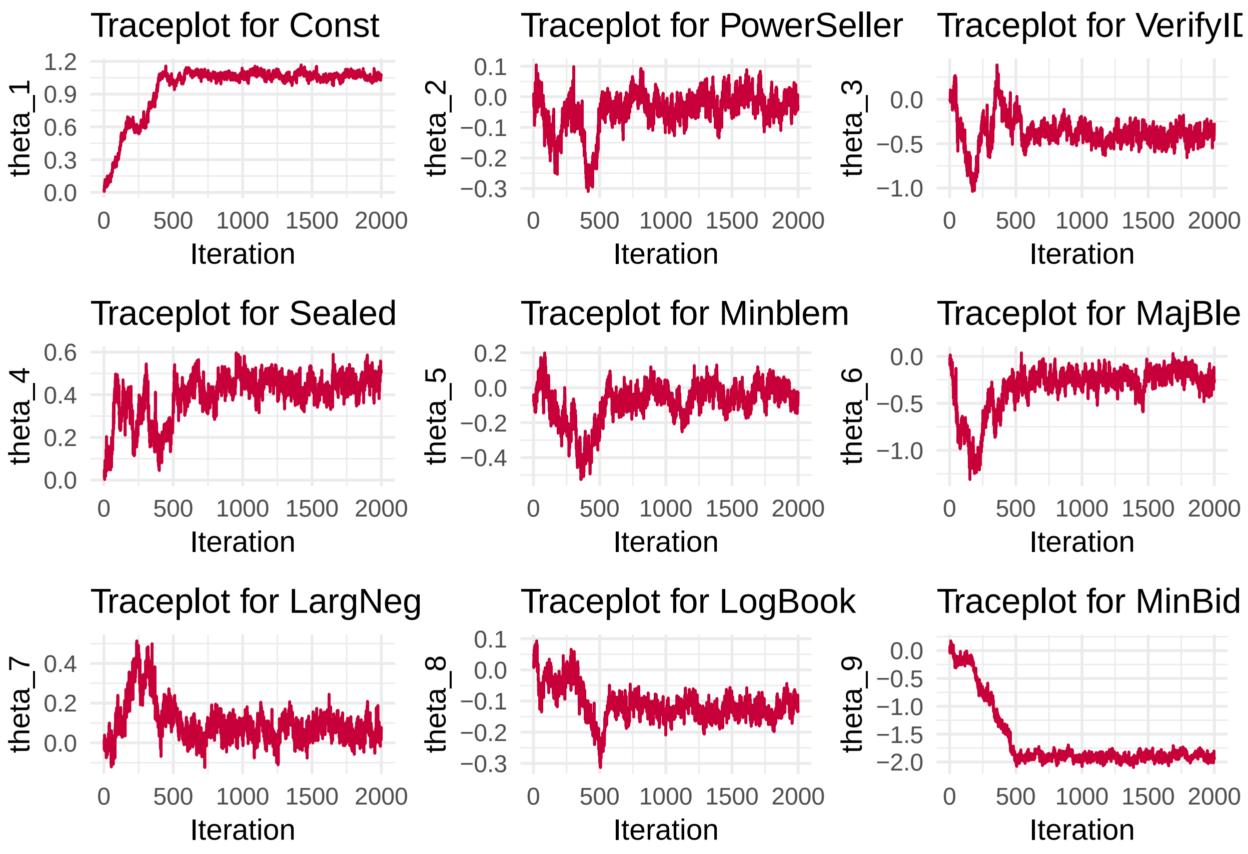
It's time to call the function. We set $c = 0.5$ to have a usable tradeoff between a small step size and the rejection rate to converge not to slowly and to cover the posterior density as good as possible. We take 1000 drawns and start with 0 for θ as mentioned.

```
# Our parameters for the function
c = 0.5
n = 2000
theta = t(as.matrix(rep(0, 9)))

# Call the Random Metropolis Walk
ms_res = RMWSampler(dposterior = dposterior,
                    n = n,
                    theta_0 = theta,
                    c = c,
                    Sigma = posterior_covariance,
                    # Now ...
                    mu_prior = posterior_mode,
                    sigma_prior = covariate_prior,
                    X = X,
                    Y = Y)

names(ms_res$samples) = feature_names

tail(ms_res$samples)
```

```
##         Const  PowerSeller   VerifyID    Sealed      Minblem    MajBlem
## 1995 1.026156 -0.006456219 -0.3862594 0.5242803 -0.02288692 -0.1555171
## 1996 1.038351  0.006998802 -0.4175616 0.5133134 -0.10629339 -0.2661668
## 1997 1.080442 -0.037053374 -0.4591248 0.5016081 -0.06745647 -0.1162047
## 1998 1.072311 -0.042728838 -0.4155356 0.5030842 -0.09373062 -0.2370597
## 1999 1.072056 -0.028208909 -0.4175453 0.5155377 -0.08201554 -0.2799492
## 2000 1.063228 -0.026593576 -0.2644220 0.5190157 -0.02384339 -0.2473272
##         LargNeg     LogBook MinBidShare
## 1995 0.06320879 -0.11309808   -1.928504
## 1996 0.07245428 -0.13442670   -1.938801
## 1997 0.03187880 -0.08032020   -1.801094
## 1998 0.07896667 -0.09071818   -1.880484
## 1999 0.01494335 -0.12878635   -1.949880
## 2000 0.07740431 -0.13605328   -1.913331
```

The rejection rate during the sampling was 0.6495532.

The following plots show the traces of all θ.



As we can see we obtained convergence for all of the parameters and their mean, after convergence, fits with the results obtained in a) and b).

Let's also have a look at the Inefficiency Factor and the Effective Sample Size.

```
effective_sample_size = effectiveSize(ms_res$samples)
names(effective_sample_size) = feature_names

effective_sample_size
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##    4.600366   12.975606   18.397631   16.177648   11.038625    8.087805
##     LargNeg     LogBook MinBidShare
##   16.052075   14.936053    2.858587
```

The Inefficiency Factor is calculated below.

```
inefficiency_factor = n / effective_sample_size

inefficiency_factor
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##    434.7480    154.1354    108.7096    123.6274    181.1820    247.2859
##     LargNeg     LogBook MinBidShare
##    124.5945    133.9042    699.6463
```

3 Source Code

```
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(ggplot2)
library(mvtnorm)
library(gridExtra)
library(coda)

################################################################################
# Exercise 1.a i)
################################################################################

rainfall_data = read.table("data/rainfall.dat", header=FALSE)

# Prior Parameters (slides page 15)
# These are basically random, guessed or from the data
# mu
mu_0 = 0
tau_sq_0 = 1

# sigma_sq
nu_0 = 1
sigma_sq_0 = 1 # sigma is 1/nu_0


crinvchisq = function(n, df, s_sq) {
  samples = rchisq(n, df)
  # These are draws from the inverse chi squared
  sigma_sq = (df - 1) * s_sq / samples
  return(sigma_sq)
}


gibbs_sample = function(nDraws, data, default_sigma, tau_sq_0, mu_0, nu_0,
                        sigma_sq_0) {

  # Posterior Parameters (Taken from lecture 2 slide 4)
  n = length(data)
  mu_n = mean(data) + mu_0
```

```r
  nu_n = nu_0 + n
  default_sigma_sq = default_sigma^2

  # To store all iterative results
  values_df = data.frame(matrix(NA, nrow = nDraws, ncol = 2))

  # To save current iterative results
  values = list(mu = NaN, sigma_sq = default_sigma_sq)

  # As mu depends on sigma and sigma depends on mu, we need one initial value to start.
  # In this implementation the default value to start with is sigma_sq.

  for (i in 1:nDraws) {
    tau_sq_n = 1 / ((n/values$sigma) + (1/tau_sq_0))
    values$mu = rnorm(1, mu_n, sqrt(tau_sq_n))
    values$sigma_sq =
      crinvchisq(1, nu_n,(nu_0*sigma_sq_0 + sum((data - values$mu)^2))/(n + nu_0))
    values_df[i,] = values
  }

  colnames(values_df) = c("mu", "sigma_sq")
  return(values_df)
}


res = gibbs_sample(nDraws = 100,
                   data = rainfall_data$V1,
                   default_sigma = 40,
                   tau_sq_0 = tau_sq_0,
                   mu_0 = mu_0,
                   nu_0 = nu_0,
                   sigma_sq_0 = sigma_sq_0)

kable(head(res))


################################################################################
# Exercise 1.a ii)
################################################################################

plotdf = data.frame(1:nrow(res), res, cumsum(res$mu)/(1:nrow(res)),
                    cumsum(res$sigma_sq)/(1:nrow(res)))
colnames(plotdf) = c("index", "mu", "sigma_sq", "mu_trailing_mean",
                     "sigma_sq_trailing_mean")

ggplot(plotdf) +
  geom_line(aes(x = index, y = mu), color = "#C70039") +
  geom_line(aes(x = index, y = mu_trailing_mean), color = "#000000") +
  labs(title = "Traceplot for mu", y = "mu",
  x = "Iteration", color = "Legend") +
  theme_minimal()
```

```r
ggplot(plotdf) +
  geom_line(aes(x = index, y = sigma_sq), color = "#C70039") +
  geom_line(aes(x = index, y = sigma_sq_trailing_mean), color = "#000000") +
  labs(title = "Traceplot for sigma_sq", y = "sigma_sq",
  x = "Iteration", color = "Legend") +
  theme_minimal()


################################################################################
# Exercise 1.b)
################################################################################

# This is the script provided in 'NormalMixtureGibbs.R' with minor ajustments.

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

##########    BEGIN USER INPUT #################
# Data options
x <- as.matrix(rainfall_data$V1)

# Model options
## COMMENT: As the given model has two guassians we set this to two.
nComp <- 2    # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(0,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 <- 100 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
# We removed the sleep time
sleepTime <- 0.1 # 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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
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))

# Added for stroring the draws and print the plots
all_mu = matrix(rep(NaN, nIter*nComp), nrow = nIter)
all_sigma_sq = matrix(rep(NaN, nIter*nComp), nrow = nIter)

for (k in 1:nIter){
  #message(paste('Iteration number:',k))
  # Just a function that converts between different representations of the
  # group allocations
  alloc <- S2alloc(S)
  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 == j] -
                                           mu[j])^2))/(nu0[j] +
                                           nAlloc[j]))
  }

    all_mu[k,] = mu
    all_sigma_sq[k,] = sigma2

  # 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 == nIter)){
    effIterCount <- effIterCount + 1
    hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
         main = paste("Iteration number",k), ylim = ylim)
    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)
  }
}

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")
```

```r
legend("topright", box.lty = 1,
       legend = c("Data histogram","Mixture density", "Normal density"),
       col = c("black","red","blue"), lwd = 2)

#########################    Helper functions    ###############################



################################################################################
# Exercise 1.c)
################################################################################

mu_plot_dataframe = data.frame(1:nIter, all_mu, cumsum(all_mu[,1])/(1:nrow(all_mu)),
                               cumsum(all_mu[,2])/(1:nrow(all_mu)))
colnames(mu_plot_dataframe) = c("index", "mu_1", "mu_2",
                                "mu_1_trailing_mean", "mu_2_trailing_mean")

ggplot(mu_plot_dataframe) +
  geom_line(aes(x = index, y = mu_1), color = "#C70039") +
  geom_line(aes(x = index, y = mu_1_trailing_mean), color = "#000000") +
  geom_line(aes(x = index, y = mu_2), color = "#6091EC") +
  geom_line(aes(x = index, y = mu_2_trailing_mean), color = "#000000") +
  labs(title = "Traceplot for mu_1 and mu_2", y = "mu",
  x = "Iteration", color = "Legend") +
  theme_minimal()


sigma_sq_plot_dataframe = data.frame(1:nIter, all_sigma_sq,
                                     cumsum(all_sigma_sq[,1])/
                                       (1:nrow(all_sigma_sq)),
                               cumsum(all_sigma_sq[,2])/(1:nrow(all_sigma_sq)))
colnames(sigma_sq_plot_dataframe) = c("index", "sigma_sq_1", "sigma_sq_2",
                                "sigma_sq_1_trailing_mean",
                                "sigma_sq_2_trailing_mean")

ggplot(sigma_sq_plot_dataframe) +
  geom_line(aes(x = index, y = sigma_sq_1), color = "#C70039") +
  geom_line(aes(x = index, y = sigma_sq_1_trailing_mean), color = "#000000") +
  geom_line(aes(x = index, y = sigma_sq_2), color = "#6091EC") +
  geom_line(aes(x = index, y = sigma_sq_2_trailing_mean), color = "#000000") +
  labs(title = "Traceplot for sigma_sq_1 and sigma_sq_2", y = "sigma_sq",
  x = "Iteration", color = "Legend") +
  theme_minimal()


x = xGrid
density_data = density(rainfall_data$V1, from = x[1], to = x[length(x)],
                       n = length(x))$y
density_normal = dnorm(x, mean = mean(res$mu), sd = sqrt(mean(res$sigma_sq)))
density_mixture = mixDensMean

density_comparison_df = data.frame(x,
                                   density_data,
                                    density_normal,
                                    density_mixture)

ggplot(density_comparison_df) +
  geom_line(aes(x = x, y = density_data, colour = "Data Kernel Density")) +
  geom_line(aes(x = x, y = density_normal, colour = "Normal Density")) +
  geom_line(aes(x = x, y = density_mixture, colour = "Mixture Density")) +
  labs(title = "Densities for the Different Model", y = "Density",
  x = "Iteration", color = "Legend") +
  scale_color_manual("Legend", values = c("#C70039", "#581845", "#6091EC")) +
  theme_minimal()


################################################################################
# Exercise 2.a)
################################################################################

ebay_data = read.table("data/eBayNumberOfBidderData.dat", header=TRUE)
kable(head(ebay_data))


glm_model = glm(formula = nBids ~ ., data = ebay_data[,-2], family = poisson)
summary(glm_model)

glm_model$coefficients

################################################################################
# Exercise 2.b)
################################################################################

# Parameters
Y = as.matrix(ebay_data[,1])
# We take all covariates
X = as.matrix(ebay_data[,-1])
# Feature names
feature_names =  colnames(ebay_data[,2:ncol(ebay_data)])
colnames(X) = feature_names

# Defining the prior parameters
mu_prior = rep(0, ncol(ebay_data) - 1)
covariate_prior = 100 * solve(t(X) %*% X)
# An we need initial beta parameters
beta = rmvnorm(1, mu_prior, covariate_prior)

# Cost-Function
## We need cost function which will be optimized (e.g. the likelihood).
## The first parameters has to be the regression coefficient
## The function will calculate the log-likelihood and the log-prior for getting
## the posterior-log-likelihood which is to be optimized
## We use two sub-functions for the two logs

# https://nptel.ac.in/courses/111104074/Module14/Lecture41.pdf
log_likelihood = function(beta, X ,Y) {
  lambda = t(X %*% beta) # log-link function
  llik = t(Y) %*% t(lambda) - sum(exp(lambda))# - sum(log(factorial(Y)))
  return(llik)
}

posterior_log_likelihood = function(beta, X, Y, sigma, mu) {
  return(dmvnorm(beta, mu, sigma, log = TRUE) + log_likelihood(beta, X, Y))
}

res = optim(beta, posterior_log_likelihood, method = "BFGS",
            control = list(fnscale = -1), hessian = TRUE,
            X = X, Y = Y, sigma = covariate_prior, mu = mu_prior)

# Now we use the results to extract the desired values and same them to
# variables with a speaking name
posterior_mode = as.vector(res$par)
names(posterior_mode) = feature_names
posterior_covariance = - solve(res$hessian)
posterior_sd = sqrt(diag(posterior_covariance))
names(posterior_sd) = feature_names

posterior_mode
posterior_covariance
posterior_sd

################################################################################
# Exercise 2.c)
################################################################################

RMWSampler = function(dposterior, n = 1, theta_0, c, Sigma, ...) {

  # Variables
  samples = data.frame()
  Sigma = c * Sigma
  i = 0

  # Current and next step for random walk
  theta_1 = theta_0
  theta = theta_0

  while (nrow(samples) < n) {
    # Get proposal
    theta_star = rmvnorm(n = 1, mean = theta_1, sigma = Sigma)

    # According to the help file we go for this version of alpha
    alpha = min(1, exp(dposterior(theta_star, ...) - dposterior(theta_1, ...)))

    # Decide if to accept or reject the proposal
    u = runif(n = 1, min = 0, max = 1)

    if (u <= alpha) {
      # Accept
      theta_1 = theta
      theta = theta_star
      samples = rbind(samples, theta_star)
    }
    else {
      # Reject
      theta = theta_1
    }

    # To keep track of the rejection rate
    i = i + 1
  }

  return(list(samples = samples, rejection_rate = (1 - (n / i))))
}


dposterior = function(theta, mu_prior, sigma_prior, X, Y) {

  # Prior
  prior = dmvnorm(theta, mu_prior, sigma_prior, log = TRUE)

  # Likelihood
  likelihood = log_likelihood(t(theta), X, Y)

  return(prior + likelihood)
}


# Our parameters for the function
c = 0.5
n = 2000
theta = t(as.matrix(rep(0, 9)))

# Call the Random Metropolis Walk
ms_res = RMWSampler(dposterior = dposterior,
                    n = n,
                    theta_0 = theta,
                    c = c,
                    Sigma = posterior_covariance,
                    # Now ...
                    mu_prior = posterior_mode,
                    sigma_prior = covariate_prior,
                    X = X,
                    Y = Y)

names(ms_res$samples) = feature_names

tail(ms_res$samples)

plotdf = data.frame(1:n, ms_res$samples)
colnames(plotdf)[1] = "index"

df_1 = ggplot(plotdf) +
  geom_line(aes(x = index, y = Const), color = "#C70039") +
  labs(title = "Traceplot for Const", y = "theta_1",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_2 = ggplot(plotdf) +
  geom_line(aes(x = index, y = PowerSeller), color = "#C70039") +
  labs(title = "Traceplot for PowerSeller", y = "theta_2",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_3 = ggplot(plotdf) +
  geom_line(aes(x = index, y = VerifyID), color = "#C70039") +
  labs(title = "Traceplot for VerifyID", y = "theta_3",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_4 = ggplot(plotdf) +
  geom_line(aes(x = index, y = Sealed), color = "#C70039") +
  labs(title = "Traceplot for Sealed", y = "theta_4",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_5 = ggplot(plotdf) +
  geom_line(aes(x = index, y = Minblem), color = "#C70039") +
  labs(title = "Traceplot for Minblem", y = "theta_5",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_6 = ggplot(plotdf) +
  geom_line(aes(x = index, y = MajBlem), color = "#C70039") +
  labs(title = "Traceplot for MajBlem", y = "theta_6",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_7 = ggplot(plotdf) +
  geom_line(aes(x = index, y = LargNeg), color = "#C70039") +
  labs(title = "Traceplot for LargNeg", y = "theta_7",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_8 = ggplot(plotdf) +
  geom_line(aes(x = index, y = LogBook), color = "#C70039") +
  labs(title = "Traceplot for LogBook", y = "theta_8",
  x = "Iteration", color = "Legend") +
  theme_minimal()

df_9 = ggplot(plotdf) +
  geom_line(aes(x = index, y = MinBidShare), color = "#C70039") +
  labs(title = "Traceplot for MinBidShare", y = "theta_9",
  x = "Iteration", color = "Legend") +
  theme_minimal()
```

```r
grid.arrange(df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, nrow = 3)


effective_sample_size = effectiveSize(ms_res$samples)
names(effective_sample_size) = feature_names

effective_sample_size


inefficiency_factor = n / effective_sample_size

inefficiency_factor
```