Lab-3

Naveen Gabriel ,Sridhar Adhikarla

2019-05-19

Contents

1 Normal model, mixture of normal model with semi-conjugate prior. 2
1.1 Normal model. 2
1.2 Mixture Model 9
1.2.1 Convergence of sampler 9
1.2.2 Parameter Trajectories 10
1.3 Graphical Comparison 11

2 Poisson regression - the MCMC way. 11
2.1 A -> Fitting a regression model on the data (GLM) 11
2.2 B -> Bayesian analysis of the Poisson regression 11
2.3 C -> Simulate Posterior using Metropolis algorithm 12

3 Appendix 16

1 Normal model, mixture of normal model with semi-conjugate prior.

1.1 Normal model.

Assuming the daily precipitaion follows normal distribution whre both μ and σ^2 are unknown.

$$y_1, y_2 ... y_n \sim \mathcal{N}(\mu, \sigma^2)$$

Assuming our semi-conjugate priors μ and σ follows below distribution :

$$\mu \sim \mathcal{N}(\mu_0, tau_0^2)$$
$$\sigma^2 \sim Inv - \chi^2(v_0, \sigma_0^2)$$

So our full conditional posterior is :

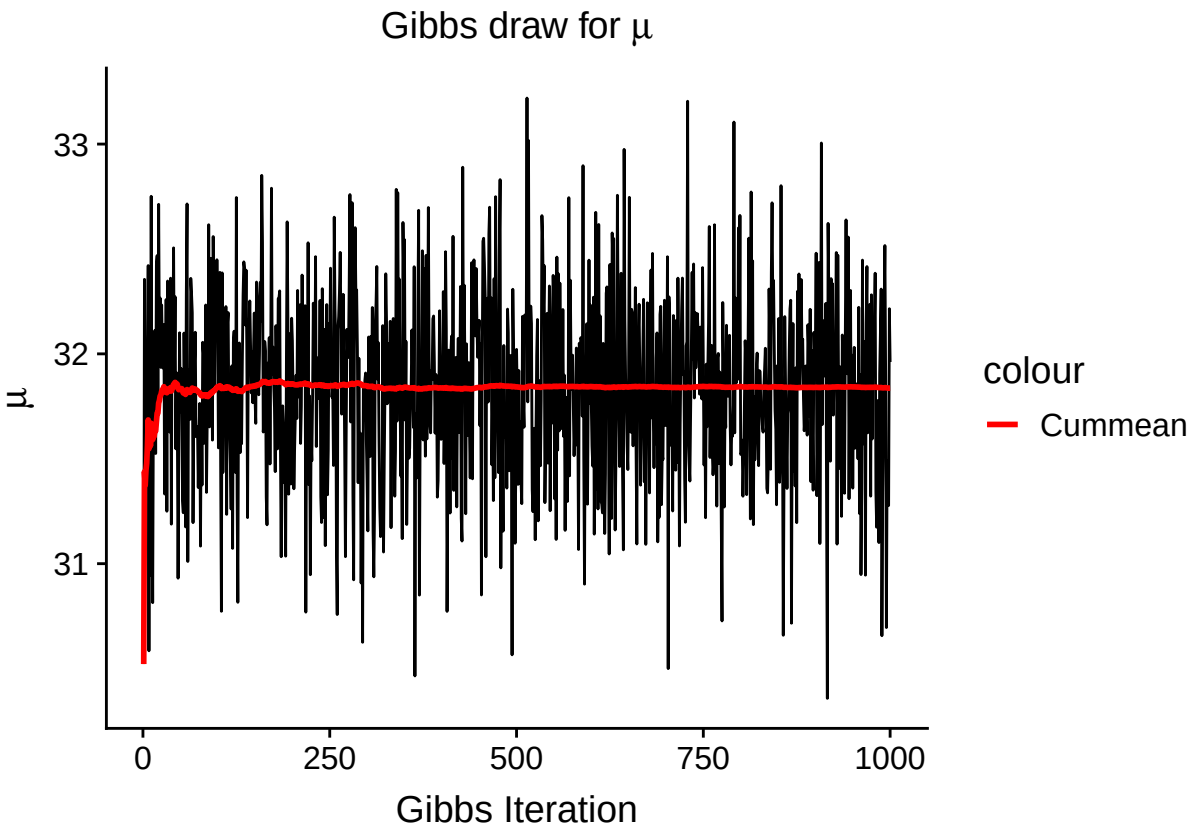
$$\mu|\sigma^2, x \sim \mathcal{N}(\mu_n, tau_n^2)$$
$$\sigma^2|\mu, x \sim Inv - \chi^2(v_n, \frac{v_o\sigma^2 + \sum_{i=0}^{n}(x_i - \mu)^2}{n + v_0})$$

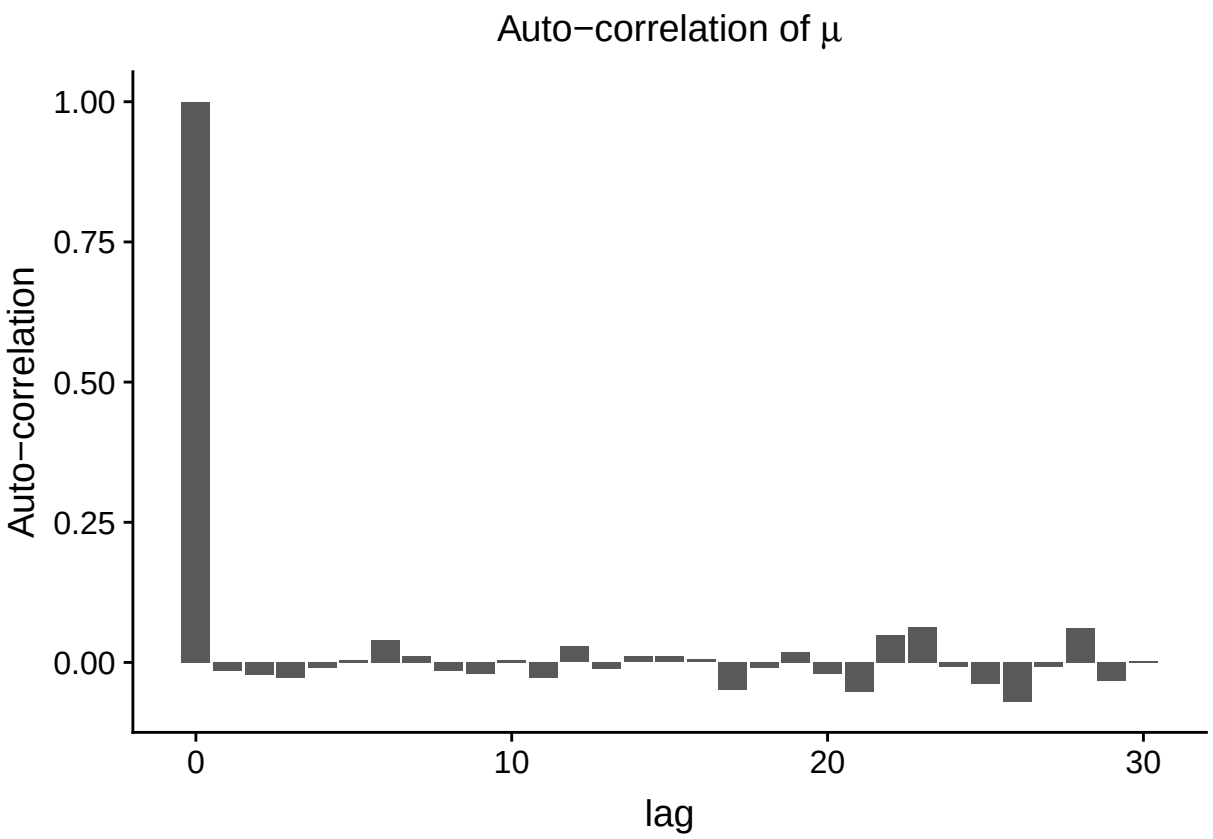
Where v_n and tau_n^2 values are

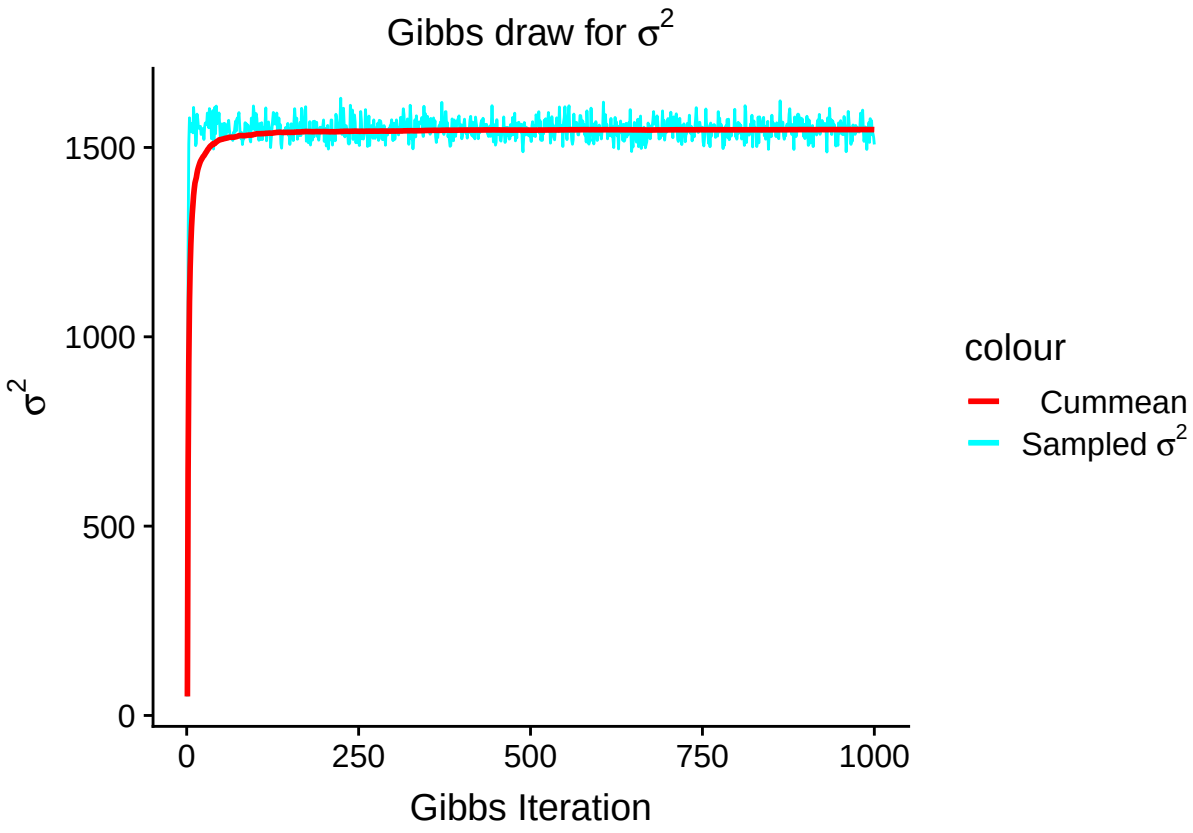
$$\tau_n^2 = \frac{\sigma^2\tau_0^2}{n\tau_0^2 + \sigma^2}$$
$$\mu_n = w\bar{x} + (1 - w)\mu_0$$

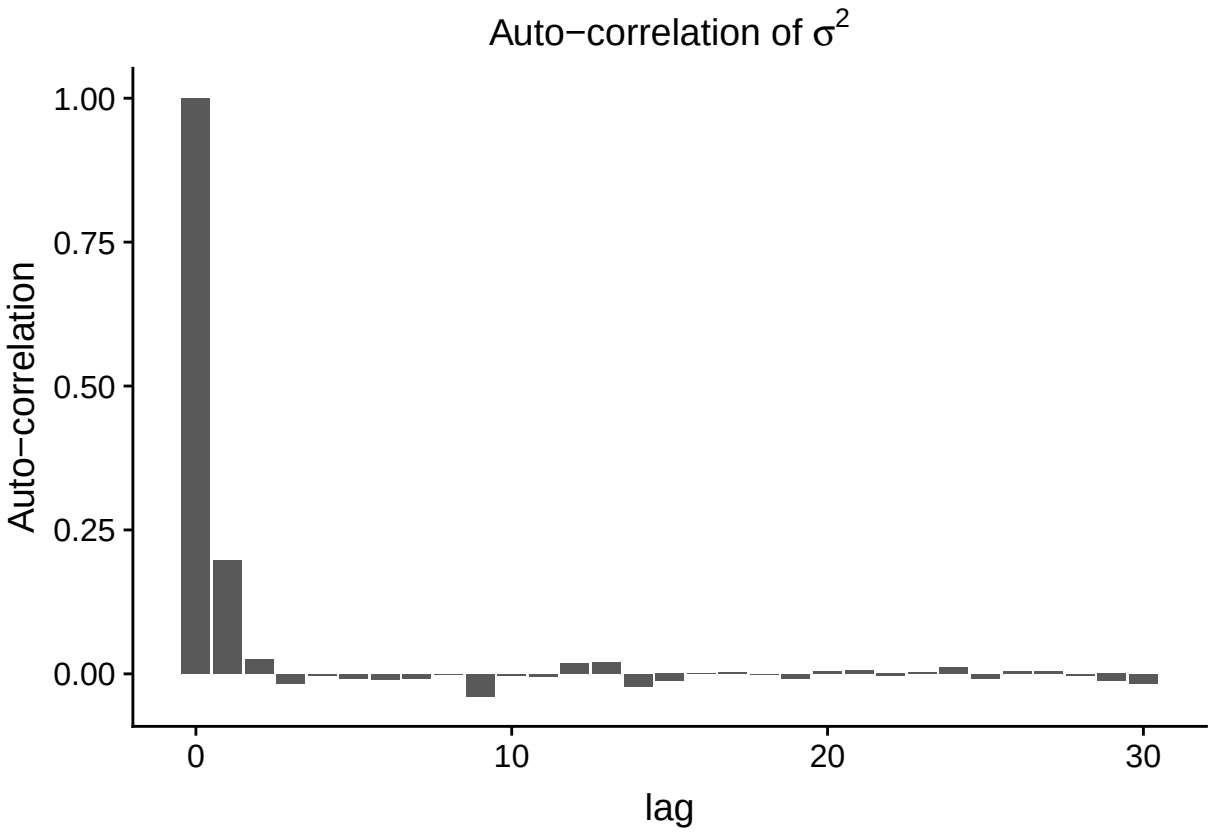
Where w is :

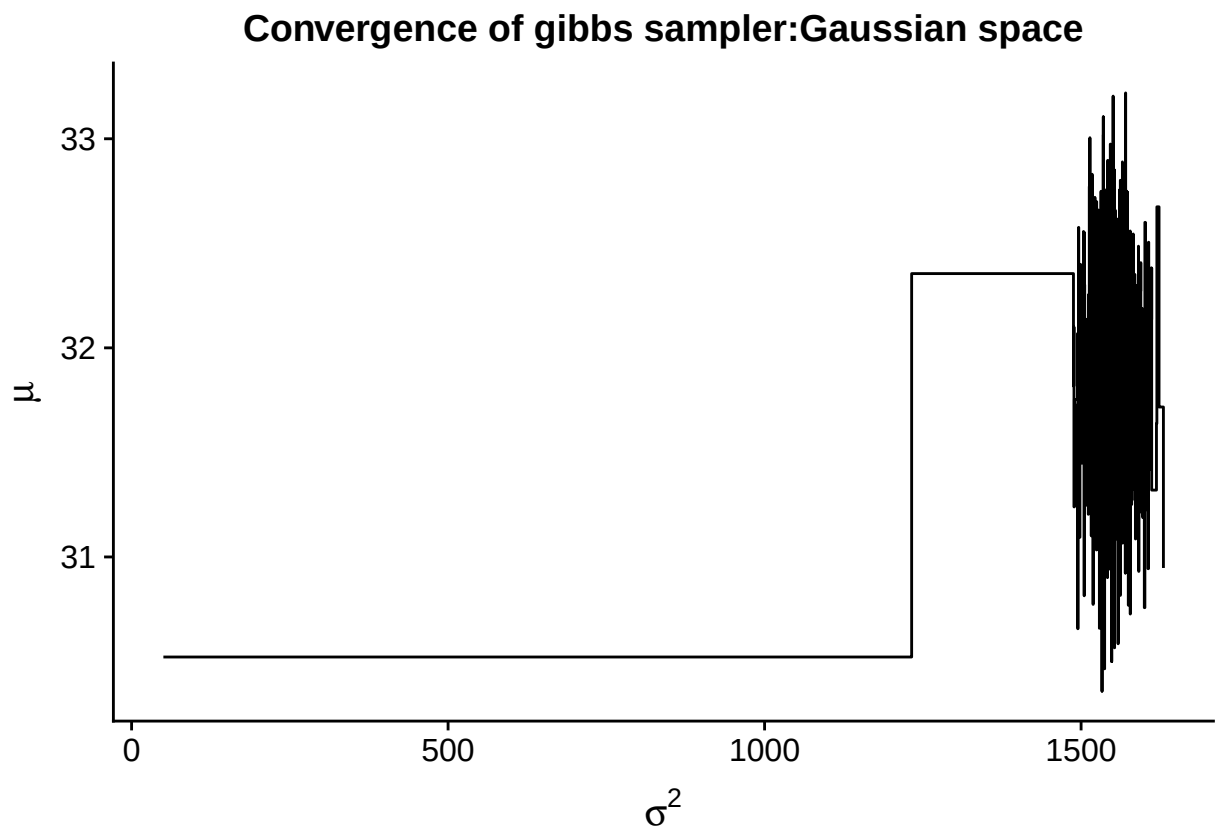
$$w = \frac{n/\sigma^2}{n/\sigma^2 + 1/\tau_0^2}$$


Gibbs draw for μ
33
32
31
μ
0
250
500
750
1000
Gibbs Iteration
colour
Cummean


Auto−correlation of μ
Auto−correlation
1.00
0.75
0.50
0.25
0.00
0
10
20
30
lag


Gibbs draw for σ2
1500
1000
500
0
σ2
0
250
500
750
1000
Gibbs Iteration
colour
Cummean
Sampled σ2

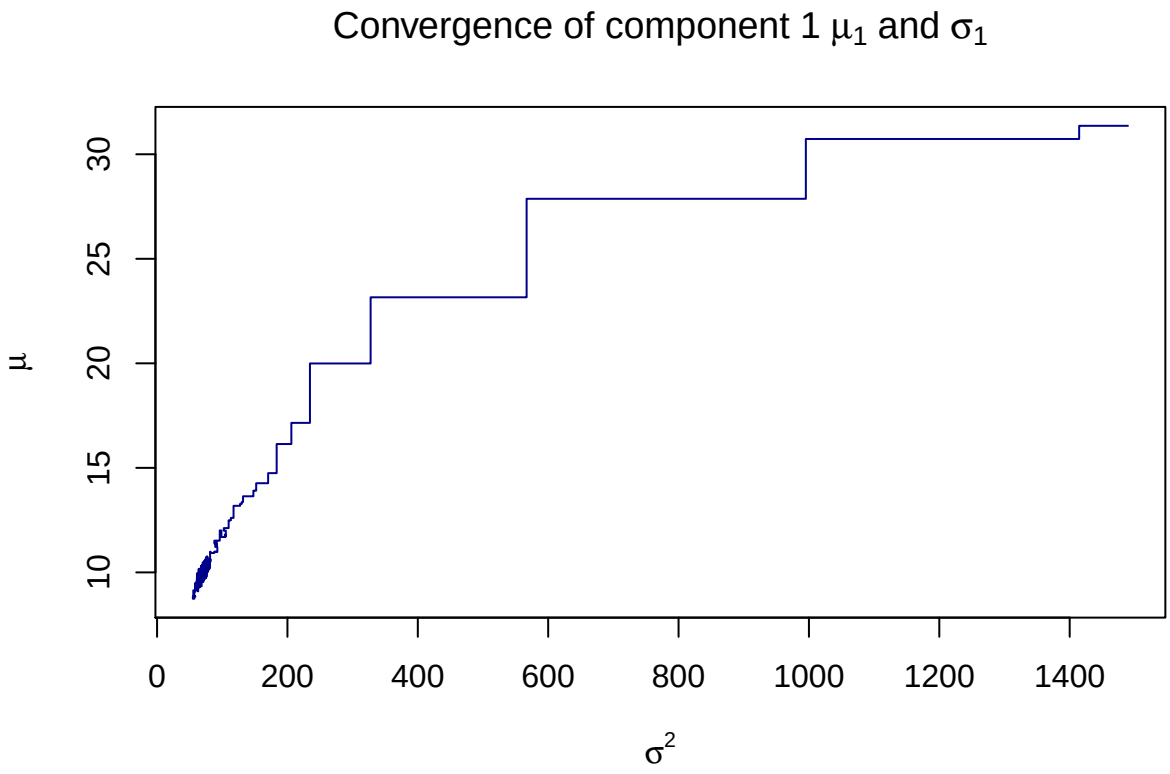

Auto−correlation of σ2
Auto−correlation
1.00
0.75
0.50
0.25
0.00
0
10
20
30
lag

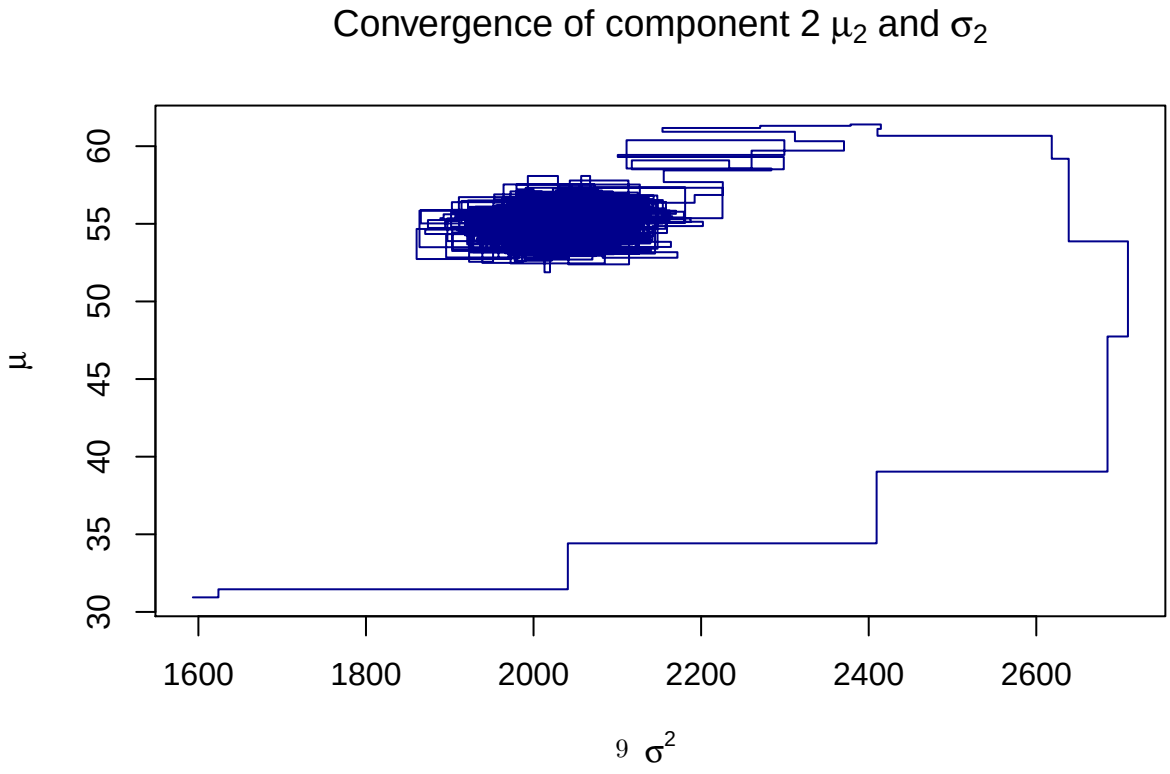


From the above plots, it is clear that using gibbs sampler converges to stable mu and sigma value

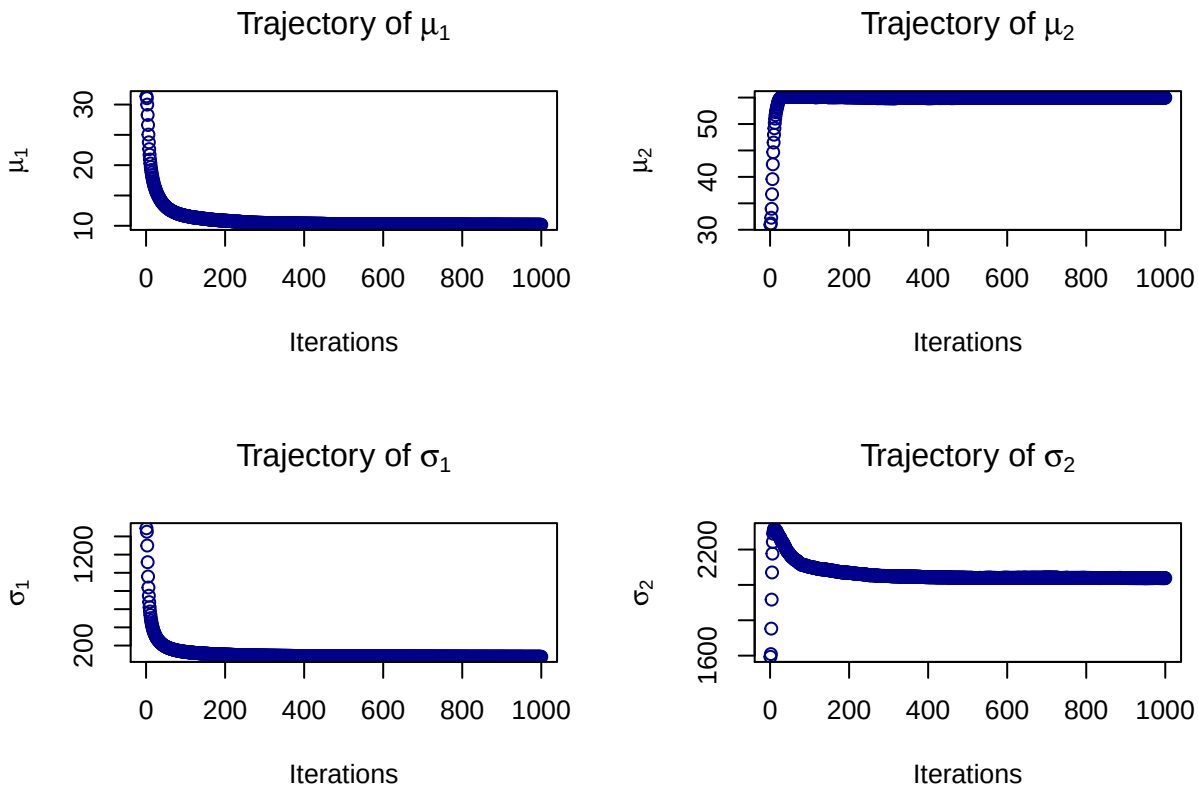
1.2 Mixture Model

1.2.1 Convergence of sampler



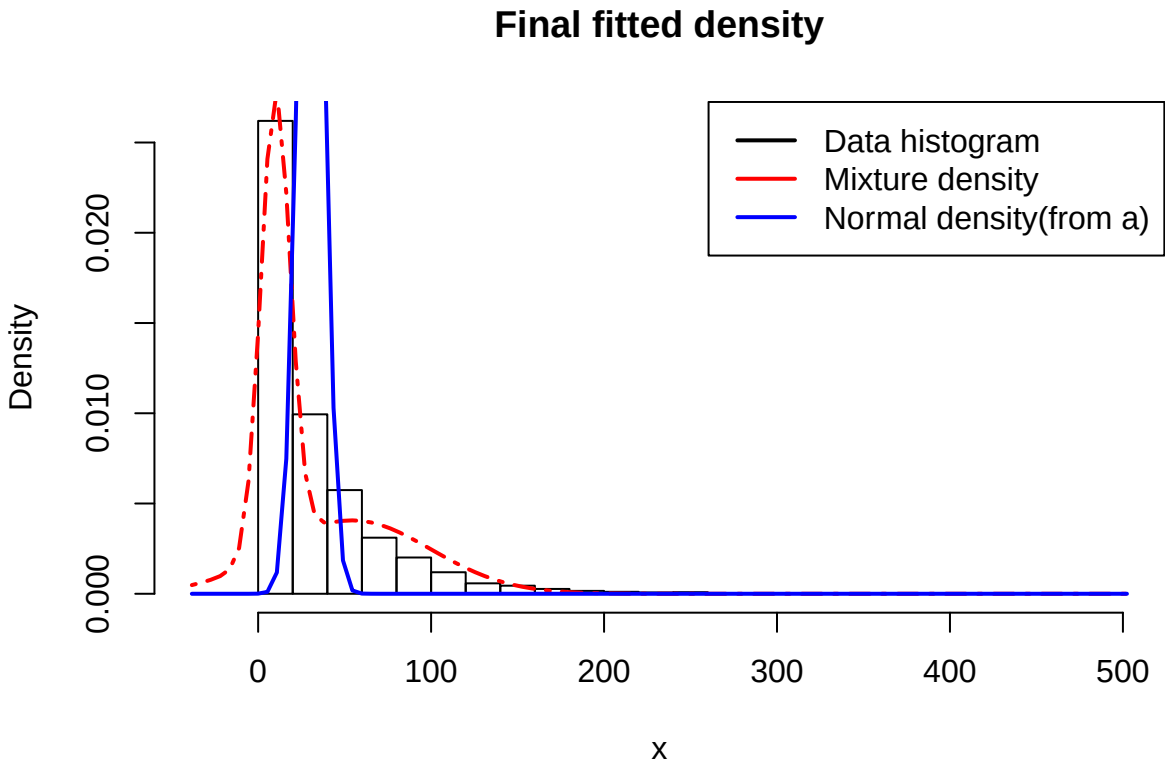


1.2.2 Parameter Trajectories



When we consider 2 component mixture of normel model for iid distribution of precipitation, the convergance of sampler and component parameters are visualized using the plots above.

1.3 Graphical Comparison



For plotting the normal density from part a, we have considered the mode of sampled values from mu and sigma and using that the normal distribution was plotted for the range of values.

2 Poisson regression - the MCMC way.

2.1 A -> Fitting a regression model on the data (GLM)

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

We have sorted the covariates according to their absolute significance in the output above. Some of the most significant covariates were - “MinBidShare”, “Sealed”, “VarifyID”, “LogBook”.

2.2 B -> Bayesian analysis of the Poisson regression

```
## MinBidShare       Const      Sealed    VerifyID     MajBlem     LogBook
## -1.89196574  1.06984694  0.44356304 -0.39301311 -0.22122262 -0.12021722
##     LargNeg     Minblem PowerSeller
##  0.07072379 -0.05248375 -0.02051922
```

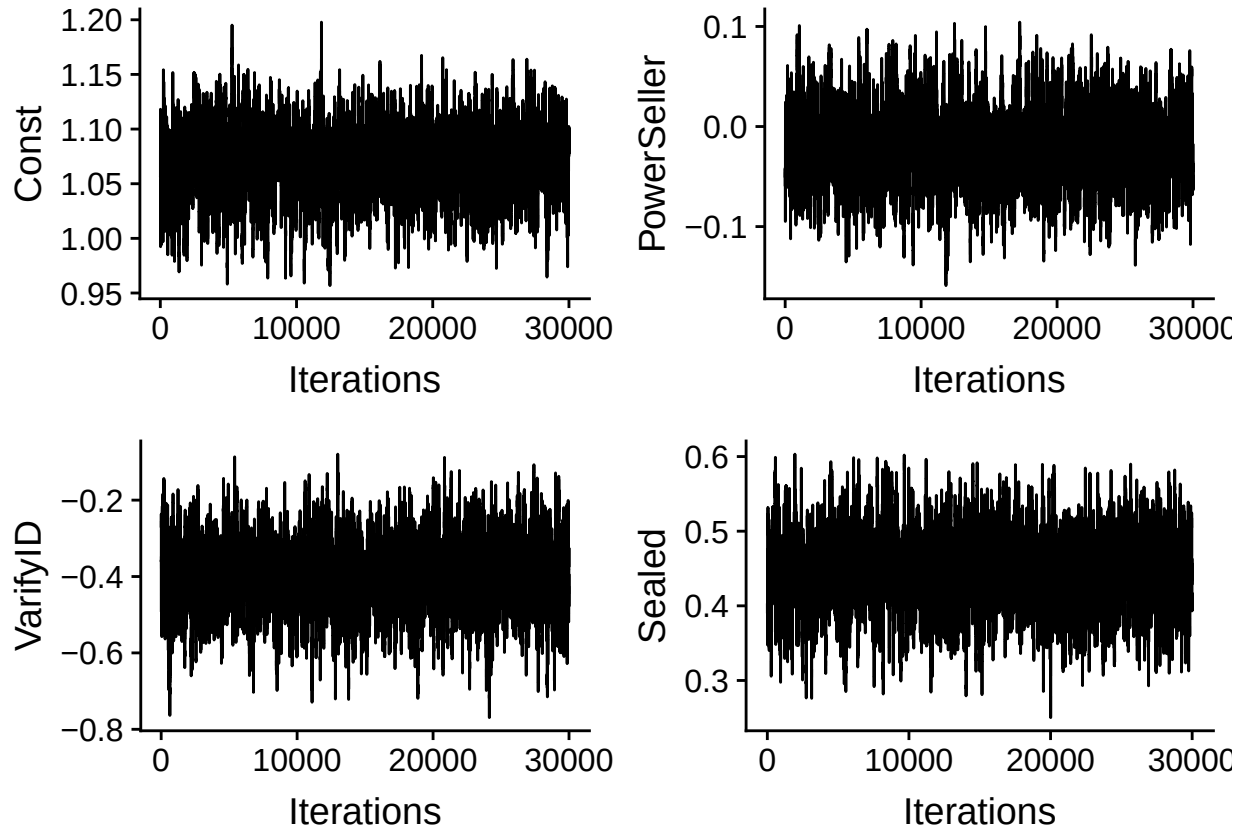
Using numerical optimization(optim function in R) for approximating the posterior distribution of beta as a multivariate normal, we get similar values for the covariates. The order of significance remains the same as we got in the previous question. The covariates "MinBidShare", "Sealed", "VarifyID", "LogBook" are still significant.

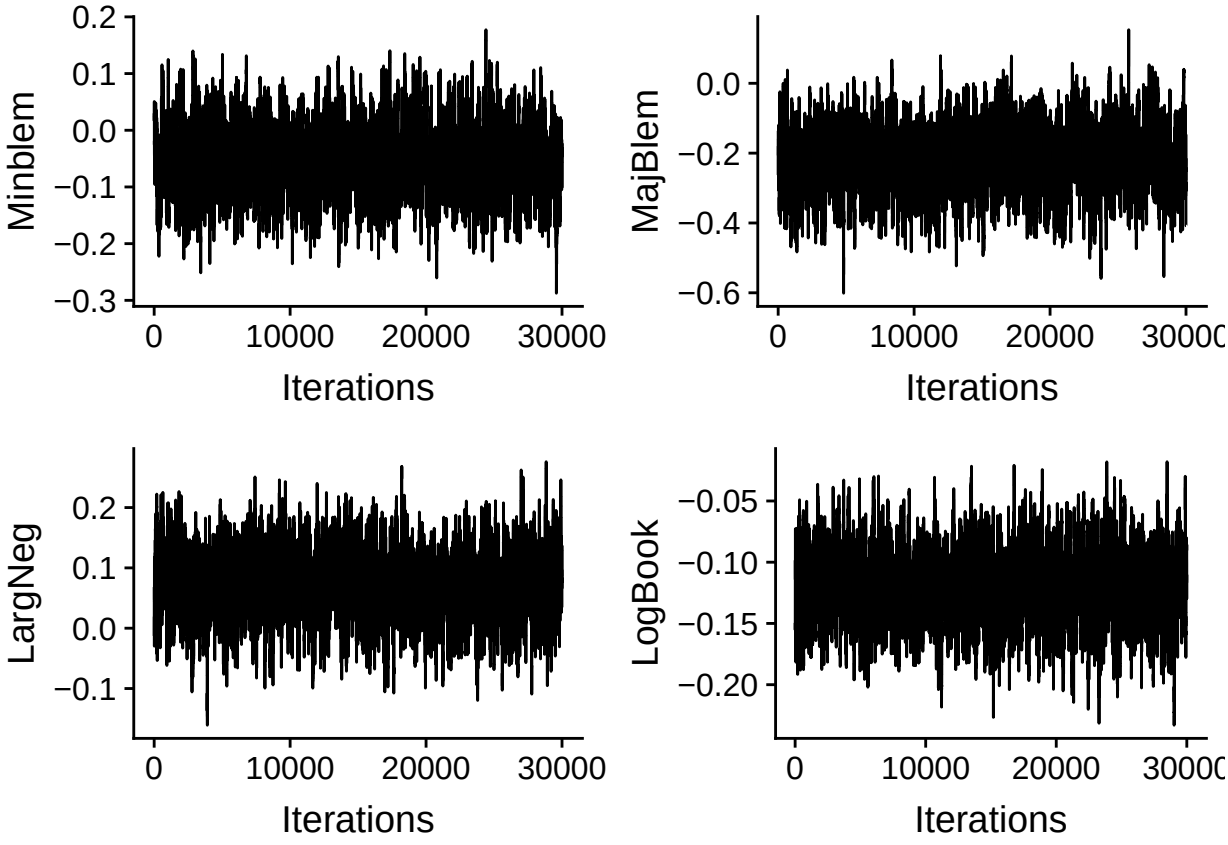
We implemented the logPostProp function for this question, which calculates the sum of log prior and the log likelihood given the initial beta values and the data.

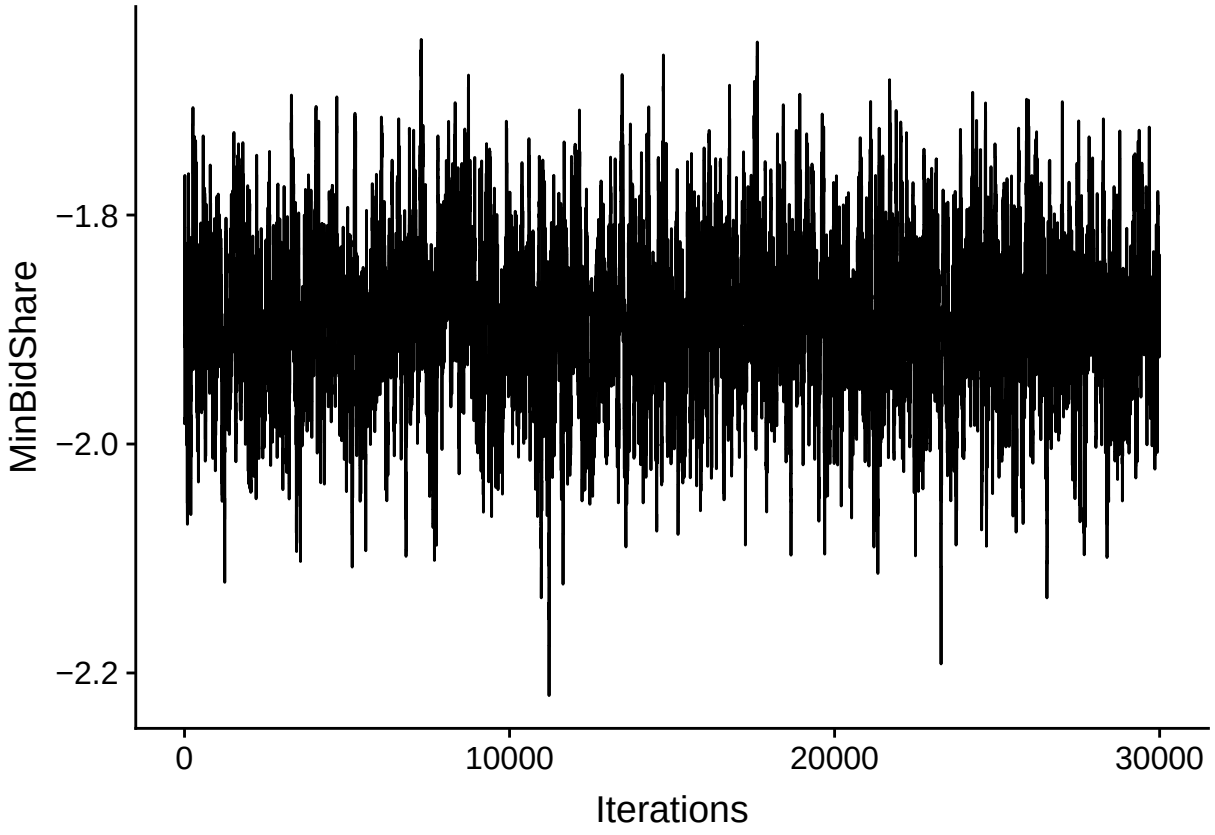
Likelihood values can get very small because we multiply so many small values, and there is a possible loss of numerical precision. This is the reason we use the log of the posterior to prevent this problem.

2.3 C -> Simulate Posterior using Metropolis algorithm

```
## Acceptance Rate for Metropolis : 0.4374848
```




Minblem
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
10000
20000
30000
Iterations
MajBlem
0.0
−0.2
−0.4
−0.6
0
10000
20000
30000
Iterations
LargNeg
0.2
0.1
0.0
−0.1
0
10000
20000
30000
Iterations
LogBook
−0.05
−0.10
−0.15
−0.20
0
10000
20000
30000
Iterations



We implemented a universal function for Metropolis haistings simulation algorithm. This function takes in as arguments the logPosterior function, the initial parameters for the posterior, and a parameter to control the variability of the simulations being generated. Along with this you can provide any number of additional arguments required for your logPosterior function. In our scenario we are sending in the data X, Y as the additional arguments for the function.

We are making 30,000 draws from the posterior, and we add in the additional burn-in period to this as 10 percent of the number of draws we are makin. So Total draws we are making is 33,000 off which we discard the first 3000 draws.

The parameter "c" that controls the variance of the simulation, was adjusted so that we get an acceptance rate close to 40%. We are getting an acceptance rate of : 0.4351212 for this chain, that is 43% accept rate, which is reasonable.

From the above trace plots for the parameters it looks like the Markov chain has reached the stationary distribution.

```
## Effective sample size for Const  :  882.8752
## Effective sample size for PowerSeller  :  874.1789
## Effective sample size for VerifyID  :  862.3082
## Effective sample size for Sealed  :  962.5294
## Effective sample size for Minblem  :  921.9693
## Effective sample size for MajBlem  :  909.1043
## Effective sample size for LargNeg  :  935.1068
## Effective sample size for LogBook  :  899.4706
## Effective sample size for MinBidShare  :  894.6875
```

On further analysis for the convergence of the markov chain, we can see that the effective sample size is also good, approx 850 for all the parameters. So we can get a lot of information from this posterior to make

interval estimates about the parameter values.

```
## [1] "Mean parameter values from the simulated posterior:"
```

```
##         Const PowerSeller   VerifyID    Sealed    Minblem    MajBlem
## [1,] 1.068269 -0.02048009 -0.3990935 0.4457779 -0.0521104 -0.2204971
##         LargNeg    LogBook MinBidShare
## [1,] 0.07148741 -0.1206097   -1.890744
```

The parameter values are similar to the ones we got for the first two questions

3 Appendix

```
knitr::opts_chunk$set(echo = FALSE,warning=NA,eval=TRUE,message=FALSE)

library(ggplot2)
library(gridExtra)
library(dplyr)
#to calculate inverse chi sqaure
invchisq <- function(v,sq) {
  sg_sq <- (v*sq)/rchisq(1,v)
  return(sg_sq)
}
rainfall <- read.table("rainfall.txt")
colnames(rainfall) <- "Precipitation"

n <- nrow(rainfall)

#Initial parameters
mu0 <- 30
sg0_sq <- 50
tau0_sq <- 1
nDraws = 1000  #number of draws for parameters
v0 <- 1500


gibbs_bayes <- function(x,mu,sg_sq,tau0_sq,v0)
{ mn <- mean(x)
  n <- length(x)
  vn <- v0 +n
  gibbsDraws <- matrix(0,nDraws,2)
  gibbsDraws[1,] <- c(mu,sg_sq)
  for (i in 2:nDraws) {

    numertr <- (n/sg_sq)
    denomtr <- (n/sg_sq) + (1/tau0_sq)
    w <- (numertr)/(denomtr)
    mu_n <- w*mn + (1-w)*mu0

    taun_sq <-  (sg_sq*tau0_sq)/((n*tau0_sq)+ sg_sq)
    mu <- rnorm(1,mu_n,sqrt(taun_sq))

    scn_arg <- ((v0*sg_sq)+sum((x-mu)^2))/vn
    sg_sq <- invchisq(vn,scn_arg)
    gibbsDraws[i,] <- c(mu,sg_sq)

  }

  return(gibbsDraws)

}

mu <- rnorm(1,mu0,sqrt(tau0_sq))
sg_sq <- invchisq(v0,sg0_sq)
```

```
data <- gibbs_bayes(rainfall$Precipitation,mu,sg_sq,tau0_sq,v0)
data <- as.data.frame(data)
colnames(data) <- c("mu","sig")
data$cummean_mu <- cummean(data$mu)
data$cummean_sig <- cummean(data$sig)

#calculating autocorrelation

mu_ac <- acf(data$mu,plot=FALSE,type="correlation")
sig_ac <- acf(data$sig,plot=FALSE,type="correlation")
autocor_data <- data.frame(mu_ac$acf,sig_ac$acf,lag=mu_ac$lag)
colnames(autocor_data) <- c("mu_ac","sig_ac","lag")

    ggplot(data, aes(x = 1:nDraws,y =mu, mucolor = "mu")) + geom_line() + xlab("Gibbs Iteration") +
    ylab(expression(mu)) + geom_line(aes(x = 1:nDraws,y =cummean_mu,color = "cummean_mu"), size = 1) +
    ggtitle(expression(paste("Gibbs draw for ", mu))) +
    scale_color_manual(labels = c("Cummean", expression(paste("Sampled ", mu))),
    values = c("red", "Cyan"))

    ggplot(data = autocor_data, mapping = aes(x = lag, y = mu_ac)) +
    geom_bar(stat = "identity", position = "identity") +
    ggtitle(expression(paste("Auto-correlation of ", mu))) + ylab("Auto-correlation")

    ggplot(data, aes(x = 1:nDraws,y = sig,color = "sig")) + geom_line() + xlab("Gibbs Iteration") +
    ylab(expression(sigma ^ 2)) + geom_line(aes(x = 1:nDraws,y = cummean_sig,
                                                color = "cummean_mu"), size = 1) +
    ggtitle(expression(paste("Gibbs draw for ", sigma^2))) +
    scale_color_manual(labels = c("Cummean", expression(paste("Sampled ", sigma ^
    2))),values = c("red", "cyan"))

    ggplot(data = autocor_data, mapping = aes(x = lag, y = sig_ac)) +
    geom_bar(stat = "identity", position = "identity") +
    ggtitle(expression(paste("Auto-correlation of ", sigma ^
    2))) +
    ylab("Auto-correlation")


ggplot(data,aes(x=sig,y=mu)) + geom_step() + xlab(expression(sigma^2)) + ylab(expression(mu)) + ggtitle
#Code credits :  Estimating a simple mixture of normals Author: Mattias Villani
#this uses semi conjugate prior
# Data options
set.seed(12345)
x <- as.matrix(rainfall$Precipitation)

# Model options
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 <- 1000 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
################   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){
    # from all components it return which position has 1
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
#S is a matrix which says from which component the
#particular observation is sampled'
# 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(density(x)$y))
#ylim = c(0,2*max(hist(x)$density))
par_mat = matrix(NA, nrow = nIter, ncol = nComp*2)
for (k in 1:nIter){
  alloc <- S2alloc(S) # Just a function that converts between different
                      #representations of the group allocations
  nAlloc <- colSums(S)
  # 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))
  }
   par_mat[k,1:nComp] = mu

  # 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]))

  }
  par_mat[k, -c(1:nComp)] = 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%%1 ==0)){
    effIterCount <- effIterCount + 1

    mixDens <- rep(0,length(xGrid))

    for (j in 1:nComp){
      compDens <- dnorm(xGrid,mu[j],sd = sqrt(sigma2[j]))
      mixDens <- mixDens + pi[j]*compDens
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount

  }
```

```
}


plot(x = par_mat[,3], y = par_mat[,1], type = 's',
     main=expression(paste("Convergence of component 1 ",mu[1]," and ", sigma[1])),
     col = "DarkBlue", xlab=expression(paste("",sigma^2)),
     ylab=expression(paste("",mu)))

plot(x = par_mat[,4], y = par_mat[,2], type = 's',
     main=expression(paste("Convergence of component 2 ",mu[2]," and ", sigma[2])),
     col = "DarkBlue",xlab=expression(paste("",sigma^2)),
     ylab=expression(paste("",mu)))

par_mat_cumsum = apply(par_mat, 2, cumsum)
par_mat_cumsum = apply(par_mat_cumsum,2,"/",seq(1:nIter))

par(mfrow =c(2,2))
plot(x = seq(1:nIter) , y = par_mat_cumsum[,1], col = "DarkBlue" ,
     main = expression(paste("Trajectory of ",mu[1])), xlab = "Iterations",
     ylab = expression(paste(mu[1])))

plot(x = seq(1:nIter) , y = par_mat_cumsum[,2], col = "DarkBlue" ,
     main = expression(paste("Trajectory of ",mu[2])), xlab = "Iterations",
     ylab = expression(paste(mu[2])))

plot(x = seq(1:nIter) , y = par_mat_cumsum[,3], col = "DarkBlue" ,
     main = expression(paste("Trajectory of ",sigma[1])), xlab = "Iterations",
     ylab = expression(paste(sigma[1])))

plot(x = seq(1:nIter) , y = par_mat_cumsum[,4], col = "DarkBlue" ,
    main = expression(paste("Trajectory of ",sigma[2])), xlab = "Iterations",
    ylab = expression(paste(sigma[2])))

Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
     main = "Final fitted density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 6, col = "red")
lines(xGrid, dnorm(xGrid, mean = Mode(data$mu), sd = sqrt(Mode(data$sig))),
      type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1,
       legend = c("Data histogram","Mixture density","Normal density(from a)"), col=c("black","red","blu
knitr::opts_chunk$set(echo = FALSE, message = FALSE,warning = FALSE)
library(MASS)
library(coda)
library(mvtnorm)
library(LaplacesDemon)
library(cowplot)
#Q1
data = read.table("eBayNumberOfBidderData.dat", header=TRUE)
n = length(data)
```

```
n_features = ncol(data) - 1 # Except y and const
feature_labels = colnames(data[,2:ncol(data)])

y = data$nBids
X = as.matrix(data[,2:ncol(data)])
X_X = t(X)%*%X
glm_model = glm(nBids ~ 0 + ., data = data, family = poisson)

glm_model$coefficients[order(abs(glm_model$coefficients), decreasing = TRUE)]
#summary(glm_model)
# Beta prior (Zellner's g-prior)
mu0 = numeric(n_features)
covar0 = 100 * ginv(X_X)
init_beta = mvrnorm(n=1, mu0, covar0)

# Log Posterior Poisson model
logPostProp <- function(betas, X, y){
  log_prior = dmvnorm(betas, mu0, covar0, log=TRUE)
  lambda = exp(X%*%betas)

  log_lik = sum(dpois(y, lambda, log=TRUE))

  return (log_lik + log_prior)
}

opt_results = optim(init_beta, logPostProp, gr=NULL, X, y,method=c("BFGS"),
                    control=list(fnscale=-1), hessian=TRUE)

# MLE beta
post_mode = opt_results$par
names(post_mode) = names(glm_model$coefficients)
post_cov = -solve(opt_results$hessian)
post_mode[order(abs(post_mode), decreasing = TRUE)]
Sigma = post_cov
c = .3
n_draws = 30000
burn_in = floor(n_draws / 10)
n_draws = n_draws + burn_in

metropolisHastings = function(logPostFunc, theta, c_param, ...){
  theta_draws = matrix(0, n_draws, length(theta))

  # Set initial
  theta_c = mvrnorm(n=1, theta, c_param)
  accpt = 0
  for(i in 1:n_draws){
    # 1: Draw new candidate theta
    theta_p = mvrnorm(n=1, theta_c, c_param)

    # 2: Determine the acceptance probability alpha
    alpha = min(c(1, exp(logPostFunc(theta_p, ...) - logPostFunc(theta_c, ...))))

    # 3: Set new value with prob = alpha
```

```
    if(rbern(n=1, p=alpha)==1){
      theta_c = theta_p
      accpt = accpt + 1
    }
    theta_draws[i,] = theta_c
  }
  cat("Acceptance Rate for Metropolis :", accpt/n_draws, "\n")

  return (theta_draws)
}
init_beta = mvrnorm(n=1, mu0, covar0)
beta_draws = metropolisHastings(logPostProp, init_beta, c*Sigma, X, y)

beta_draws = beta_draws[(burn_in+1):nrow(beta_draws),]
beta_means = t(as.matrix(colMeans(beta_draws)))
colnames(beta_means) = feature_labels

tot = dim(beta_draws)[1]
tp = ggplot() + xlab("Iterations")
plot_grid(tp + geom_line(aes(x = 1:tot, y = beta_draws[,1])) + ylab("Const"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,2])) + ylab("PowerSeller"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,3])) + ylab("VarifyID"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,4])) + ylab("Sealed"))
plot_grid(tp + geom_line(aes(x = 1:tot, y = beta_draws[,5])) + ylab("Minblem"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,6])) + ylab("MajBlem"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,7])) + ylab("LargNeg"),
          tp + geom_line(aes(x = 1:tot, y = beta_draws[,8])) + ylab("LogBook"))
tp + geom_line(aes(x = 1:tot, y = beta_draws[,9])) + ylab("MinBidShare")
for(i in 1:9){
  cat("Effective sample size for", feature_labels[i]," : ",
      effectiveSize(as.mcmc(beta_draws[,i])), "\n")
}
print("Mean parameter values from the simulated posterior:")
beta_means
```