Computer Lab 3

Phillip H??lscher

3 5 2019

Contents

1. Normal model, mixture of normal model with semi-conjugate prior. **2**
(a) Normal model. 2
(b) Mixture normal model. 5
(c) Graphical comparison. 6

2. Metropolis Random Walk for Poisson regression. **9**
(a) Obtain the maximum likelihood estimator 9
(b) Bayesian analysis of the Poisson regression 10
(c) Simulate from the actual posterior 10

Example code **14**

Lecture code **14**

Appendix **18**

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

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 $\frac{1}{100}$ inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

(a) Normal model.

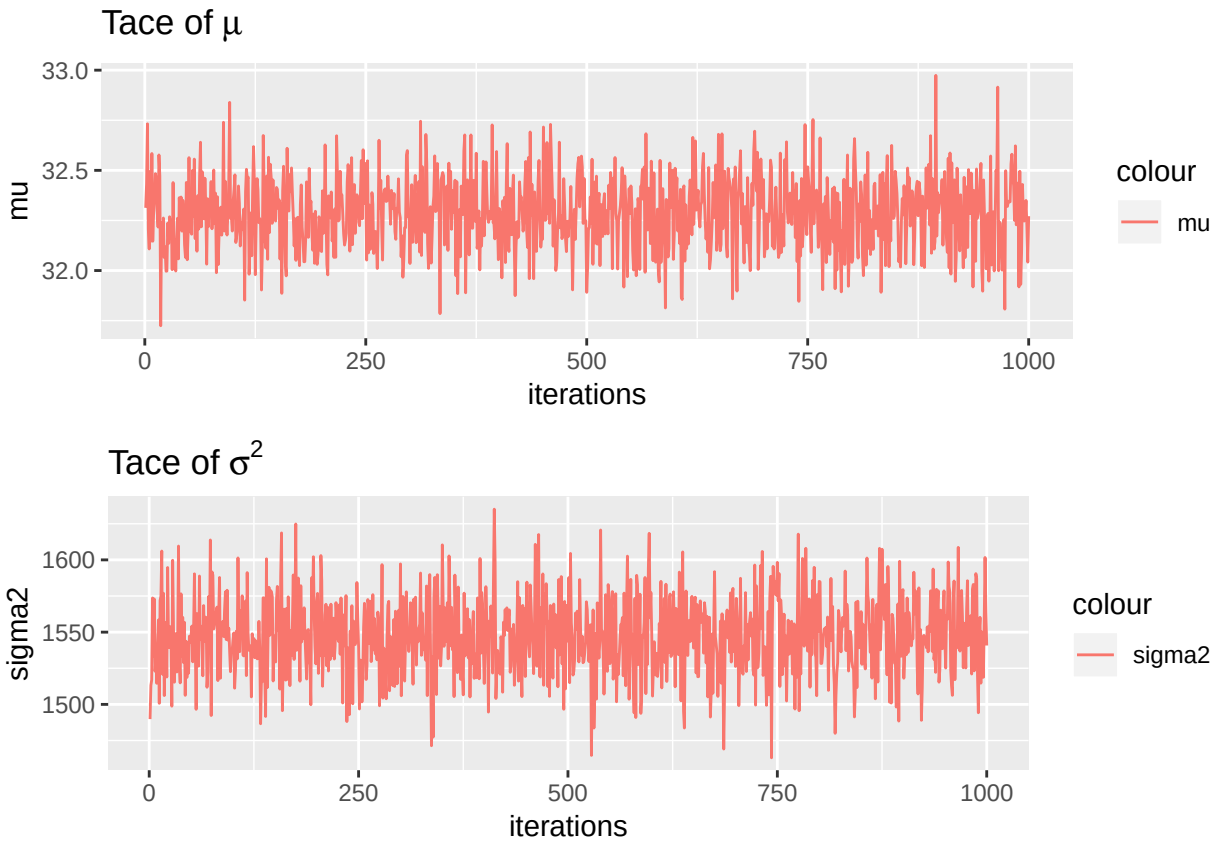
Assume the daily precipitation $\{y_1, ..., y_n\}$ are independent normally distributed, $y_1, ..., y_n \mid \mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim (v_0, \sigma_0^2)$.

- i. Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2 \mid y_1, ..., y_n)$ The full conditional posteriors are given on the slides from Lecture 7.

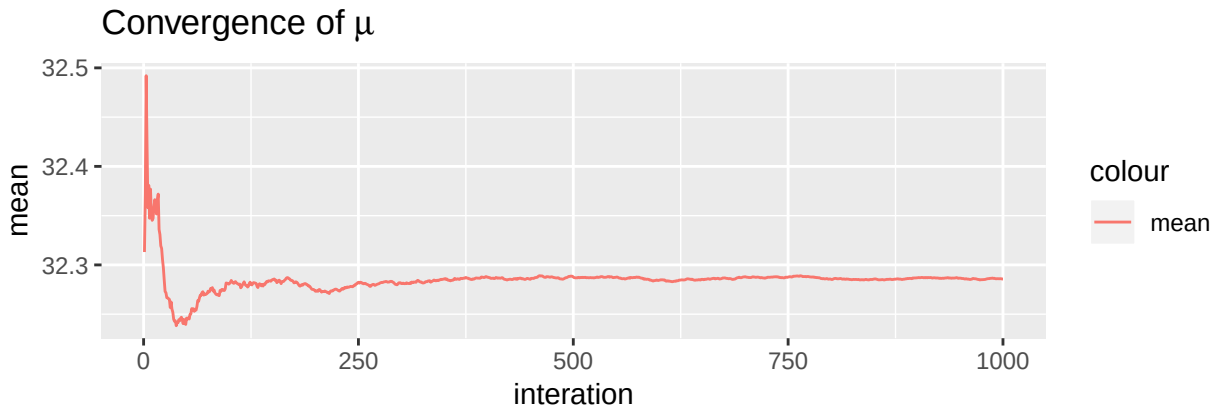
```
# implement Gibbs sampling
# set seed
set.seed(123456)

# Gibbs sampling for normal model with non-conjugate prior - L7S15

# prior parameter for sigma
# initialized values - decided by my own
# v_0 tells us how much we trust the prior
v_0 <- 1

sigma2_0 <- var(rainfall$precipitation) # the variacne of the data
# sigma
sigma2 <- rinvchisq(n = 1, df = v_0, scale = sigma2_0) # need this value for the first simulation

# prior parameter for mu
# initialized values - decided by my own
tau2_0 <- 1
#tau2_0 <- sigma2_0
mu_0 <- mean(rainfall$precipitation)
# mu
mu <- rnorm(n = 1, mean = mu_0, tau2_0) # need this value for the first simulation

# the sampling
nDraws <- 1000
Gibbs_sampling <- data.frame("mu" = numeric(),
                             "sigma2" = numeric())

# for exercise 1 a) ii
# create vairables for mean and varaicne
mean_vec <- vector(mode = "numeric", length = nDraws)
variance_vec <- vector(mode = "numeric", length = nDraws)

for (i in 1:nDraws) {
  # full conditional posterior
  # generate mu - need mu_n (need w to generate mu_n) & tau2_n
  # mu_n - L2S3 - Normal data, known variance - normal prior
```

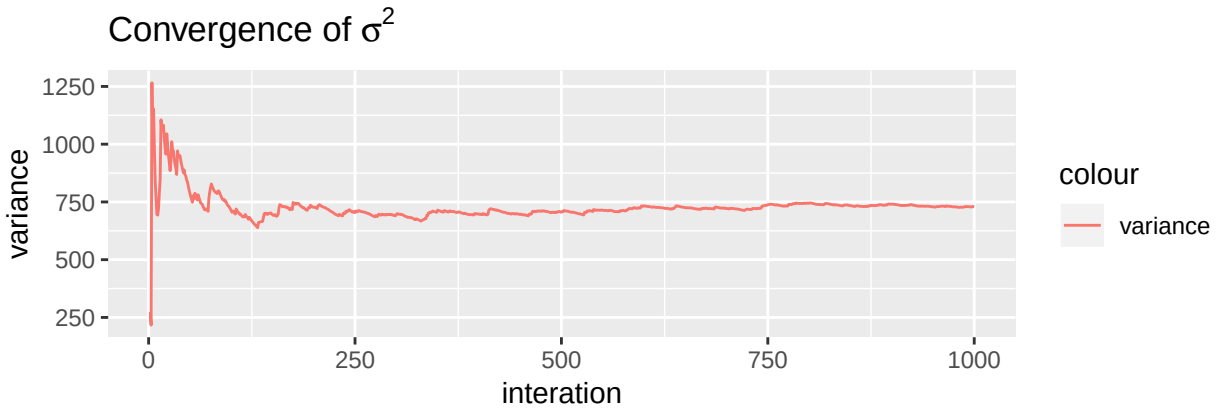
```
  #-- generate mu ------------ start
  # generate w
  w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
  # generate mu_n
  mu_n <- w * mean(rainfall$precipitation) + (1 - w) * mu_0

  # genearte tau2_n
  tau2_n <- 1 / (n/sigma2 + 1/tau2_0)

  #-- generate mu
  mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)
  #-- generate mu ------------ end


  # generate sigma2 ------------ start
  # generate v_n - L53 - Normal model with normal prior
  v_n = v_0 + n

  # generate second term - to sample sigma2
  second_term <- (v_0 * sigma2 + sum((rainfall$precipitation - mu)^2))/ (n + v_0)

  #-- generate sigma2
  sigma2 <- rinvchisq(n = 1, df = v_n, scale = second_term)
  # generate sigma2 ------------ end

  # save mu & sigma in df
  sampling <- data.frame("mu" = mu,
                         "sigma2" = sigma2)

  # variables we want to save
  Gibbs_sampling <- rbind(Gibbs_sampling, sampling)
  #save the mean of mu and variance
  mean_vec[i] <- mean(Gibbs_sampling$mu)
  variance_vec[i] <- var(Gibbs_sampling$sigma2)

}
Gibbs_sampling$iterations <- 1:nDraws
```

- ii. 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.



In the first picture you can see the course of μ, which converges very quickly. The starting value is about 22, but after a few iterations it approaches the value of 26-27. It seems that sigma does not have a start value that is extremely outside the normal range. So there is no burn-in period.





We have calculated the mean and the variance of the given iteration. This can be seen at the end of the code of the task 1a)i of the Gibbs sampler. The first picture confirms the assumption that mu converges after only a few iterations. The same is true for sigma, but here about 125 iterations are used.

(b) Mixture normal model.

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

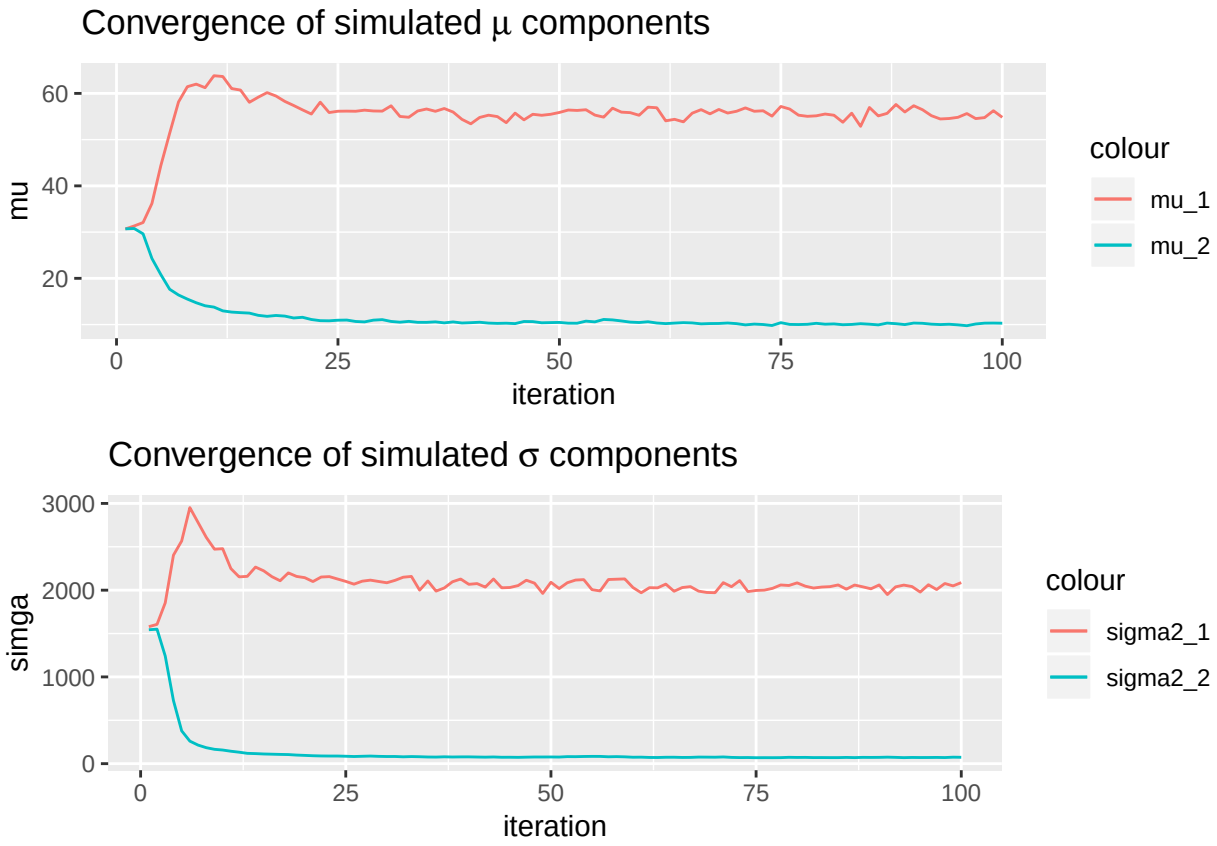
$$p(y_i \mid \mu, \sigma^2, \pi) = \pi \, \mathcal{N}(y_i \mid \mu_1, \sigma_1^2) + (1 - \pi)\mathcal{N}(y_i \mid \mu_2, \sigma_2^2),$$

where

$$\mu = (\mu_1, \mu_2) \; and \; \sigma^2 = (\sigma_1^2, \sigma_2^2)$$

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.

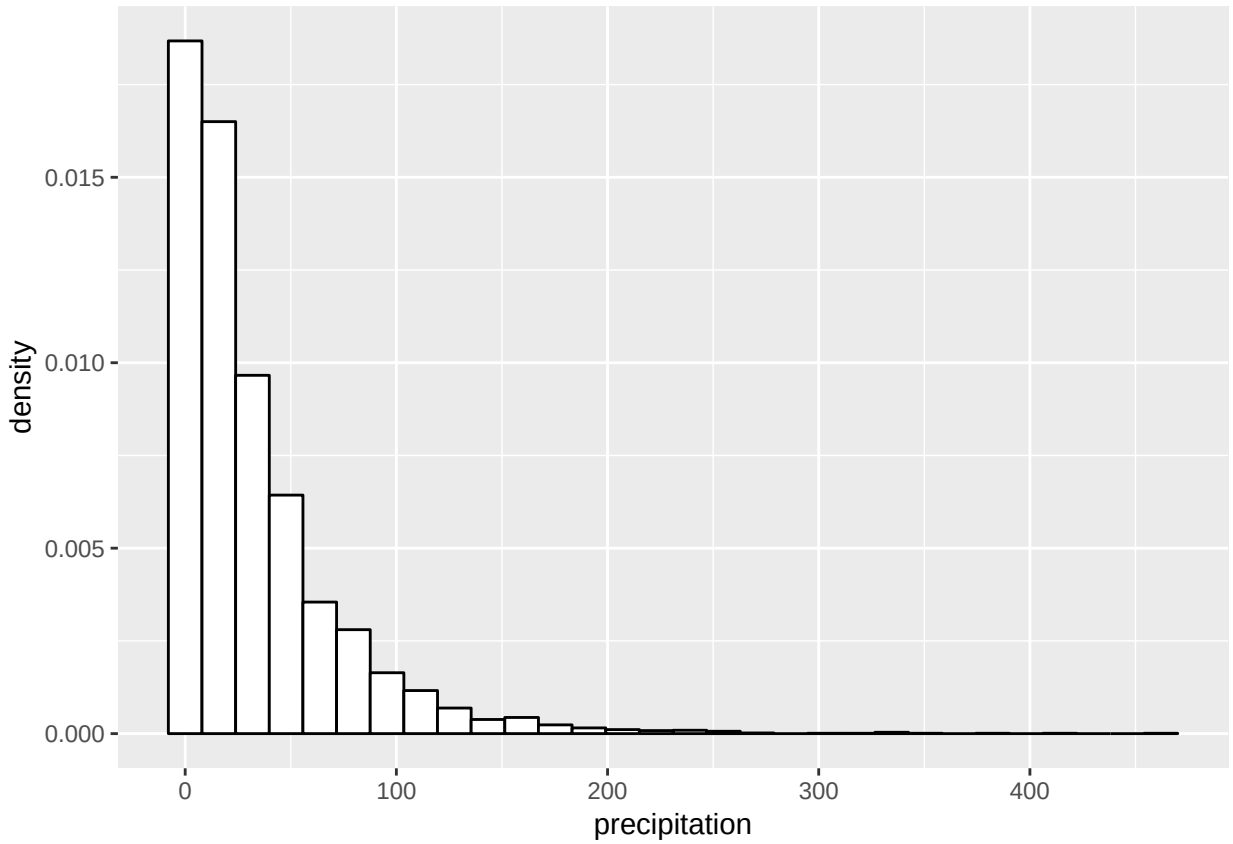
The following code from lecture 7, this one has been adapted and modified for the following tasks.



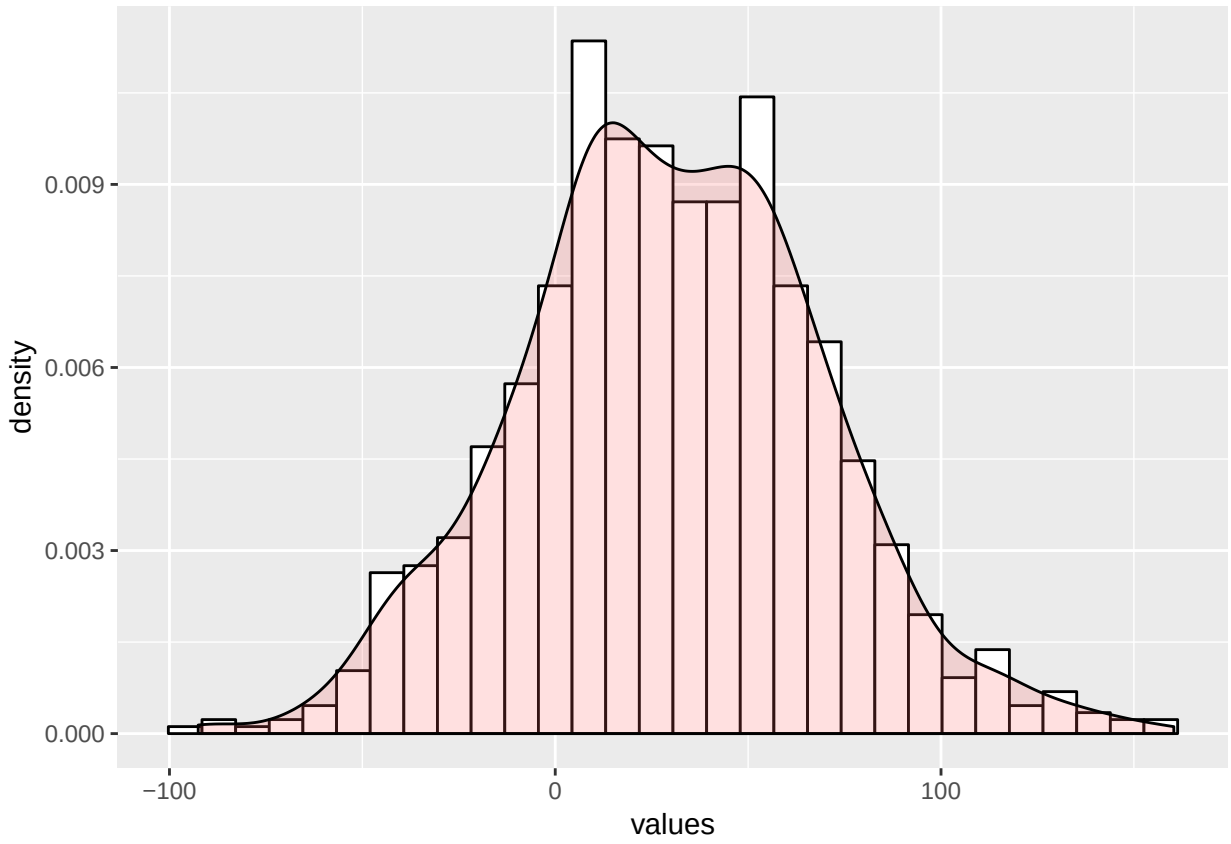
(c) Graphical comparison.

Let $\hat{\mu}$ denote the posterior mean of the parameter μ and correspondingly for the other parameters. Plot the following densities in one figure:

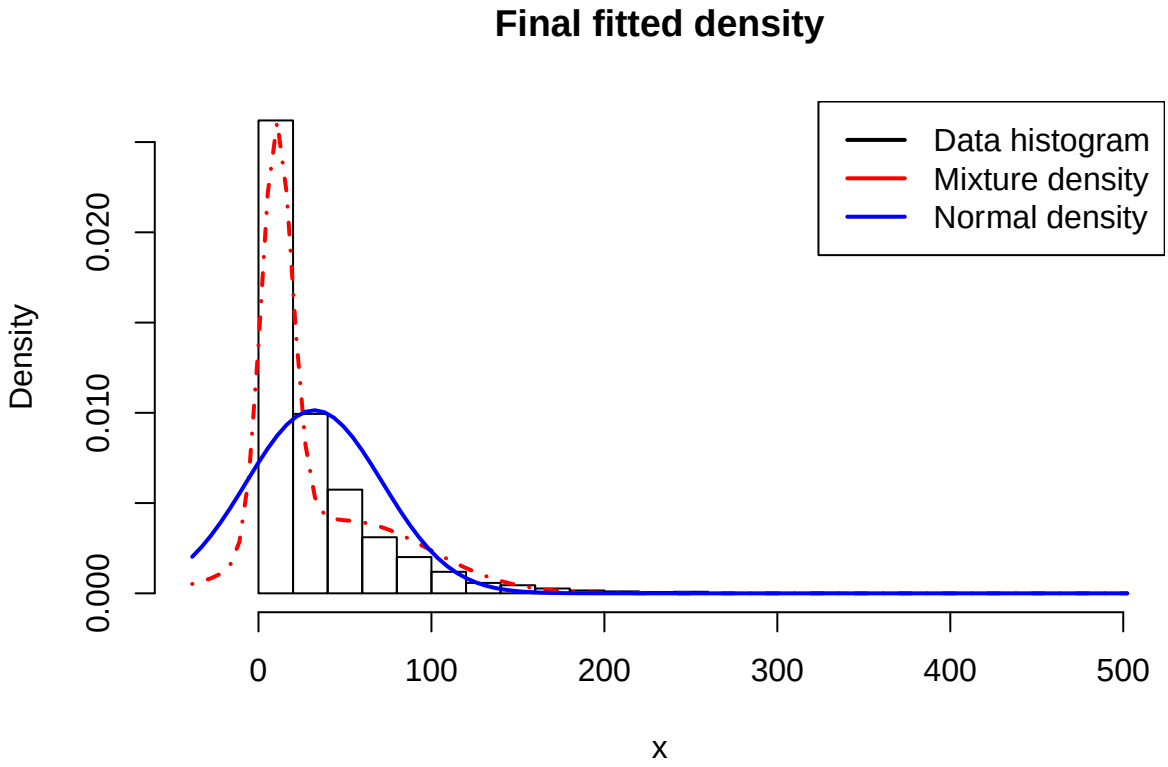
1) a histogram or kernel density estimate of the data.



2) Normal density $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ in (a).



3) Mixture of normals density $p(y_i \mid \hat{\mu}, \hat{\sigma}^2, \hat{\pi})$ in (b).



2. Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model

$$y_i \mid \beta \sim \ Poisson \ \big[\ exp(\mathbf{x}_i^T \beta) \big], \ i = 1, ..., n,$$

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**, do nott forget that **glm()** adds its own intercept so do nott input the covariate Const]. Which covariates are significant?

The output of the poisson regression model:

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

Above do we see the output of the fitted Poisson regression model. The exercises are to find covariates which are significant. This doe we observe at the Coefficients table. The p-values which do have three starts next to it are statistically significant. This is the case for **VerifyID, Sealed, LogBook, MinBidShare**. This variables have the highest influence of the number of bets (response variable).

Coefficients values:

	(Intercept)	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
Poisson_rm$coefficients	1.072442	-0.0205408	-0.3945165	0.4438426	-0.0521983	-0.2208712	0.0706725	-0.1206776	-1.894097

(b) Bayesian analysis of the Poisson regression

Lets now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}\big[\mathbf{0}, 100 \cdot (\mathbf{X}^T\mathbf{X})^{-1}\big]$ 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:

$$\beta \mid y \sim \mathcal{N}\big(\tilde{\beta}, J_{\mathbf{y}}^{-1}(\tilde{\beta})\big),$$

where $\tilde{\beta}$ is the posterior mode and $J_{\mathbf{y}}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_{\mathbf{y}}(\tilde{\beta})$ can 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.).

Posterior mode - Coefficients values:

	(Intercept)	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
postMode	1.069841	-0.0205125	-0.393006	0.4435555	-0.0524663	-0.2212384	0.0706968	-0.1202177	-1.891985

As we can see does the result of the coefficient estimation looks like in the exercise of 2a).

(c) Simulate from the actual posterior

Now, lets 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):

$$\theta_p \mid \theta^{i-1} \sim \mathcal{N}\big(\theta^{i-1}, c \cdot \sum\big),$$

where $\sum = J_{\mathbf{y}}^{-1}(\tilde{\beta})$ obtained in b). The value c is a tuning parameter and y should be an input to your Metropolis function. The user of your Metropo- lis 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 this in 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.

```
### Version 2

# Metropolis-Hastings


# Random walk Metropolis algorithm - L8S7
RWMSampler = function(logPostFunc, theta0, c, Sigma, n_iter, ...){
  # Input:
  # LogPostFunc - any posterior distribution
  # Sigma - negative Hessian at the posterior mode. (calculated in b))
  # c - tunning parameter

  # init vlaues
  accept <- 0
  reject <- 0


  # Initialize theta^(0)
  # Initialize matrix to save the result
  theta_output <- matrix(data = 0, nrow = n_iter, ncol = length(postMode))
  thetha_i_1 <- theta0

  # given proposal density
  theta_c = mvrnorm(n =1, mu = thetha_i_1, Sigma = c * Sigma) # current theta

  # setp 1 - sample proposal distribution

  for (i in 1:n_iter) {
    # given proposal density
    theta_p <- mvrnorm(n = 1, mu = theta_c, Sigma = c * Sigma)

    # setp 2 - compute acceptance probability

    # calculate the ratio
    # p(thetha_p|y) / p(theta^(i-1)|y)
    p_theta_p <- logPostFunc(theta_p, ...)
    p_theta_i_1 <- logPostFunc(theta_c, ...)
    ratio <- exp(p_theta_p - p_theta_i_1)

    # find alpha
    alpha <- min(c(1, ratio))

    # setp 3 - set theta^(i) = theta_p and theta^(i) = theta^(i-1)

    U = runif(1)
    if(U < alpha){
      theta_c <- theta_p
      accept = accept + 1
    } else{
      reject = reject + 1
    }

    # save vakue
```

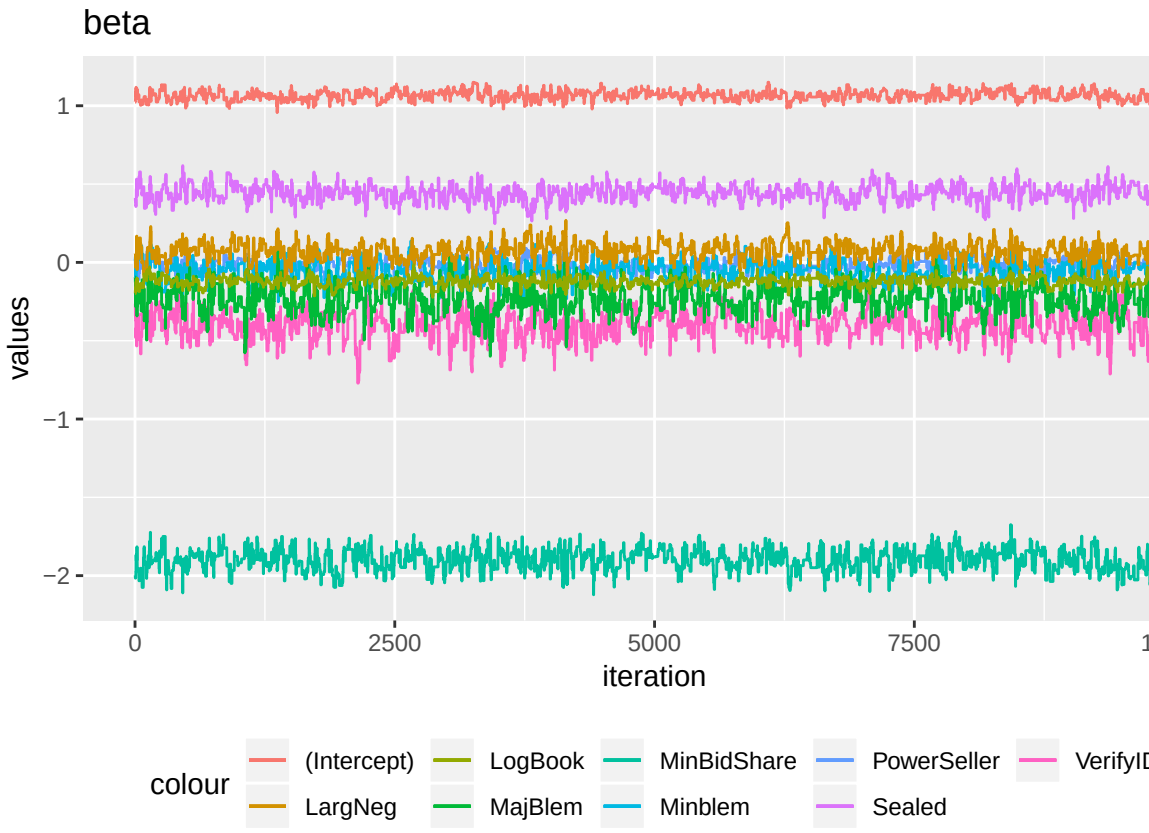
```
    theta_output[i,] <- theta_c

    }

    # return results
    result_list <- list("theta" = theta_output,
                        "accept" = accept,
                        "reject" = reject,
                        "accep_prob" = accept/reject)

}
```

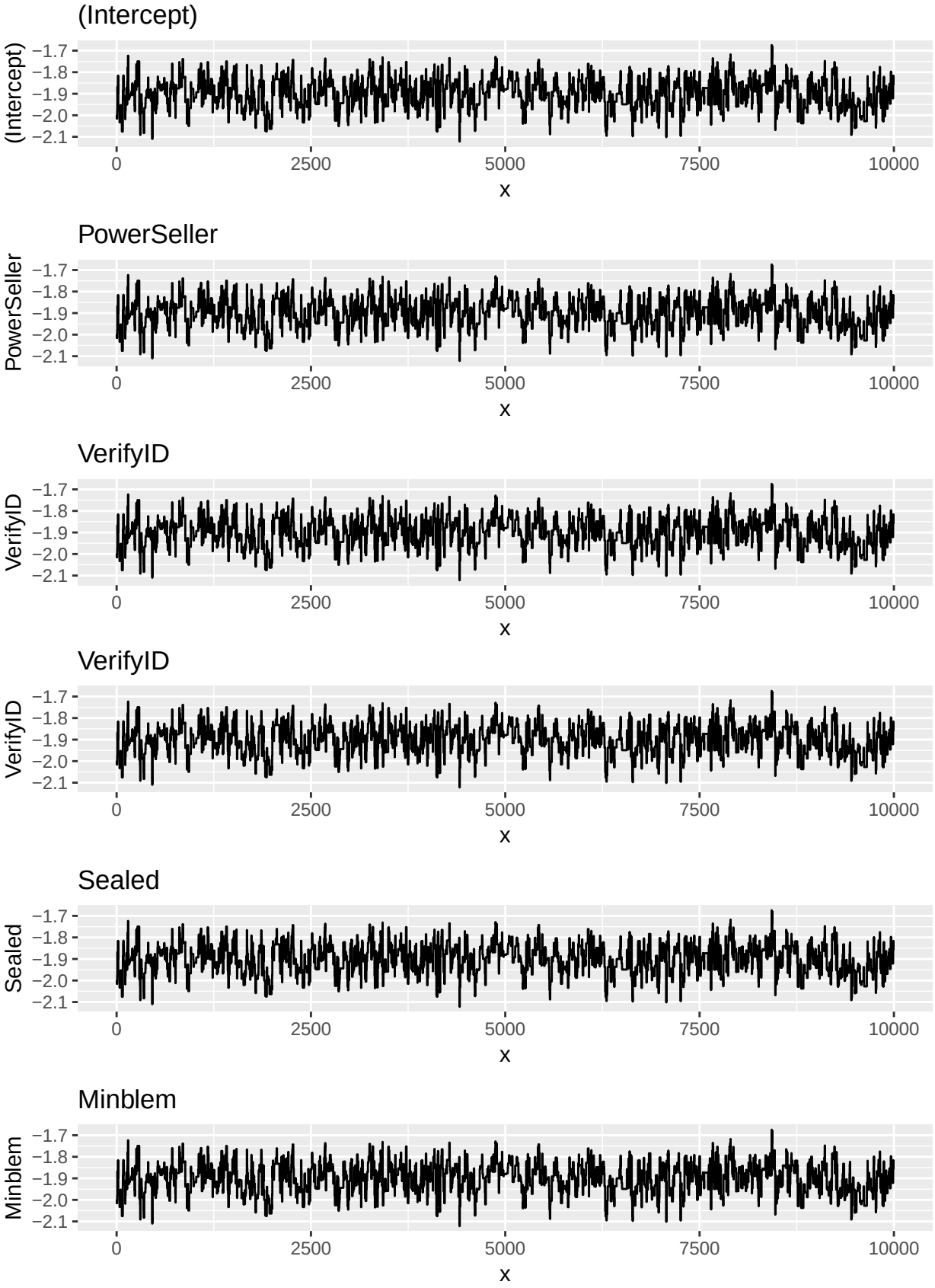
```
# run random walk
RandomWalk_Metropolis <- RWMSampler(logPostFunc = LogPostPoisson,
                                    theta = postMode,
                                    c = 1,
                                    Sigma = postCov,
                                    n_iter=10000,
                                    y=y, X=X)
```

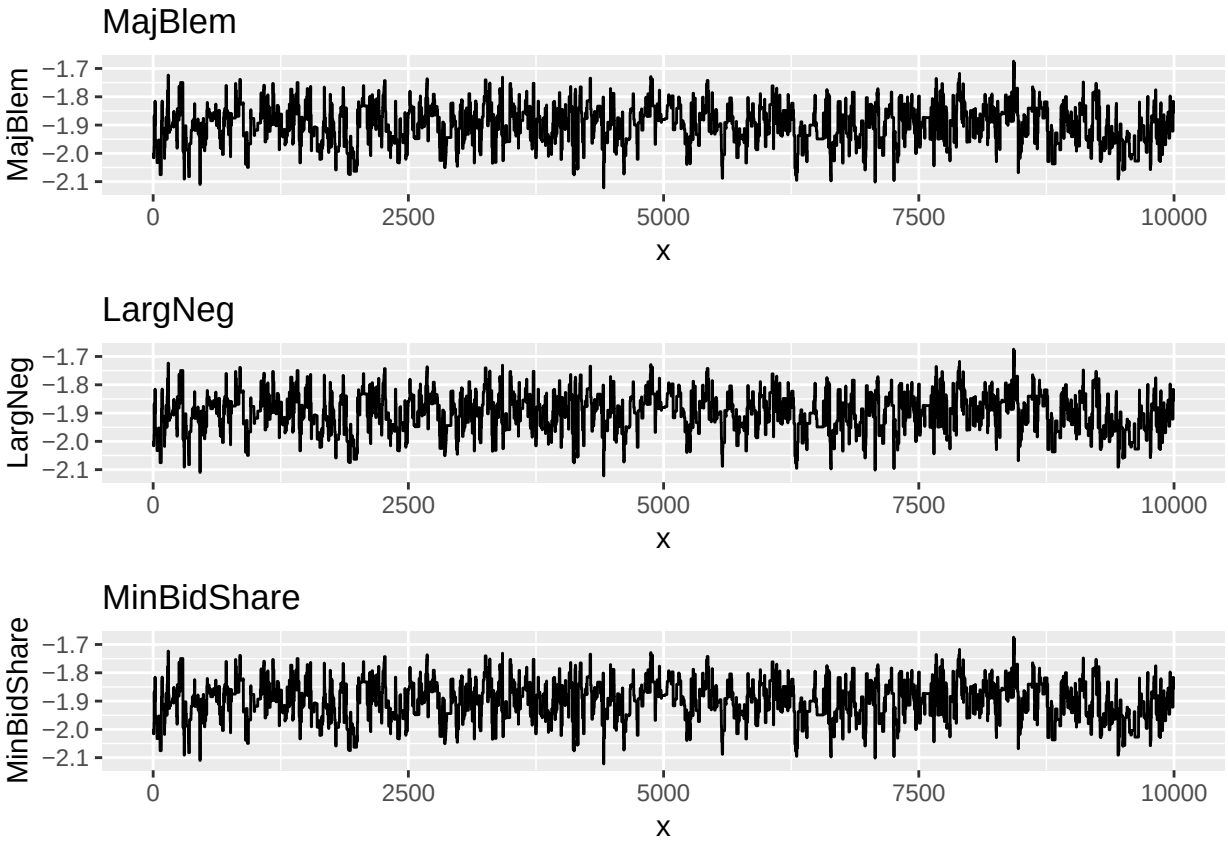
Coefficients values - Random Walk Metropolis:

(Intercept)	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
1.066068	-0.0187374	-0.3989406	0.4422477	-0.053574	-0.2312308	0.0681343	-0.1211083	-1.899356

Plot of all the coefficients:




(Intercept)
(Intercept)
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x
PowerSeller
PowerSeller
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x
VerifyID
VerifyID
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x
VerifyID
VerifyID
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x
Sealed
Sealed
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x
Minblem
Minblem
−1.7
−1.8
−1.9
−2.0
−2.1
0
2500
5000
7500
10000
x



Example code

Lecture code

```
# the given R file - NormalMixtureGibbs.R

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

##########    BEGIN USER INPUT #################
# Data options
data(faithful)
rawData <- faithful
x <- as.matrix(rawData['eruptions'])

# Model options
nComp <- 4    # 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
```

```r
# MCMC options
nIter <- 1000 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
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)
  }
  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)
  }

}

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(

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

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# library used
library(ggplot2)
library(MASS) # To access the mvrnorm() function
library(LaplacesDemon) # for rinvchisq
library(gridExtra) # put plot together
library(mvtnorm)
library(knitr)
#install.packages("kableExtra")
library(kableExtra)
#install.packages('latex2exp')
library(latex2exp)
library(dplyr)
# read data
rainfall <- read.delim("rainfall.dat", header = FALSE)
colnames(rainfall) <- c("precipitation")
n = nrow(rainfall)
# implement Gibbs sampling
# set seed
set.seed(123456)

# Gibbs sampling for normal model with non-conjugate prior - L7S15

# prior parameter for sigma
# initialized values - decided by my own
# v_0 tells us how much we trust the prior
v_0 <- 1

sigma2_0 <- var(rainfall$precipitation) # the variacne of the data
# sigma
sigma2 <- rinvchisq(n = 1, df = v_0, scale = sigma2_0) # need this value for the first simulation

# prior parameter for mu
# initialized values - decided by my own
tau2_0 <- 1
#tau2_0 <- sigma2_0
mu_0 <- mean(rainfall$precipitation)
# mu
mu <- rnorm(n = 1, mean = mu_0, tau2_0) # need this value for the first simulation

# the sampling
nDraws <- 1000
Gibbs_sampling <- data.frame("mu" = numeric(),
                             "sigma2" = numeric())

# for exercise 1 a) ii
# create vairables for mean and varaicne
mean_vec <- vector(mode = "numeric", length = nDraws)
variance_vec <- vector(mode = "numeric", length = nDraws)

for (i in 1:nDraws) {
```

```r
  # full conditional posterior
  # generate mu - need mu_n (need w to generate mu_n) & tau2_n
  # mu_n - L2S3 - Normal data, known variance - normal prior

  #-- generate mu ------------ start
  # generate w
  w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
  # generate mu_n
  mu_n <- w * mean(rainfall$precipitation) + (1 - w) * mu_0

  # genearte tau2_n
  tau2_n <- 1 / (n/sigma2 + 1/tau2_0)

  #-- generate mu
  mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)
  #-- generate mu ------------ end


  # generate sigma2 ------------ start
  # generate v_n - L53 - Normal model with normal prior
  v_n = v_0 + n

  # generate second term - to sample sigma2
  second_term <- (v_0 * sigma2 + sum((rainfall$precipitation - mu)^2))/ (n + v_0)

  #-- generate sigma2
  sigma2 <- rinvchisq(n = 1, df = v_n, scale = second_term)
  # generate sigma2 ------------ end

  # save mu & sigma in df
  sampling <- data.frame("mu" = mu,
                         "sigma2" = sigma2)

  # variables we want to save
  Gibbs_sampling <- rbind(Gibbs_sampling, sampling)
  #save the mean of mu and variance
  mean_vec[i] <- mean(Gibbs_sampling$mu)
  variance_vec[i] <- var(Gibbs_sampling$sigma2)

}
Gibbs_sampling$iterations <- 1:nDraws

# plot mu and sigma2

# plot mu
plot_tace_mu <- ggplot(data = Gibbs_sampling) +
  geom_line(aes(x = iterations, y = mu, color = "mu")) +
  ggtitle(TeX("Tace of $\\mu$"))
  # plot sigma2
plot_tace_sigma2 <-ggplot(data = Gibbs_sampling) +
  geom_line(aes(x = iterations, y = sigma2, color = "sigma2")) +
  ggtitle(TeX("Tace of $\\sigma^2$"))
```

```r
grid.arrange(plot_tace_mu, plot_tace_sigma2, nrow = 2)

# converge of mean
plot_converge_mu <- ggplot() +
  geom_line(aes(x = Gibbs_sampling$iterations, y = mean_vec, color = "mean")) +
  labs(title = TeX("Convergence of $\\mu$"), x = "interation", y = "mean")

# converge of sigma
plot_converge_sigma <-ggplot() +
  geom_line(aes(x = Gibbs_sampling$iterations, y = variance_vec, color = "variance")) +
  labs(title = TeX("Convergence of $\\sigma^2$"), x = "interation", y = "variance")

grid.arrange(plot_converge_mu, plot_converge_sigma, nrow = 2)

### Don??t plot this part!

# plotting the trajectories
# plot data
plot_data_q1aii <- Gibbs_sampling[,1:2]
plot_data_q1aii <- plot_data_q1aii[1:100,] # test- just take the first x rows

# ggplot(plot_data_q1aii) +
#   geom_line(x = plot_data_q1aii$mu, y = plot_data_q1aii$sigma2)

plot_mu_vs_sigma_dot <- ggplot(plot_data_q1aii, aes(x = plot_data_q1aii$mu, y = plot_data_q1aii$sigma2)
  geom_point(aes(x = plot_data_q1aii$mu[1], y = plot_data_q1aii$sigma2[1], color = "start"), size = 3) +
  geom_point(aes(x = plot_data_q1aii$mu[length(plot_data_q1aii$mu)],
                 y = plot_data_q1aii$sigma2[length(plot_data_q1aii$sigma2)], color = "end"), size = 3) +
  ggtitle("mu vs sigma2 - dots") + ylab("sigma2") + xlab("mu") +
  theme(legend.position = "bottom")
plot_mu_vs_sigma_dot


### Don??t plot this part!

plot_mu_vs_sigma_line <- ggplot(plot_data_q1aii, aes(x = plot_data_q1aii$mu, y = plot_data_q1aii$sigma2
  geom_point(aes(x = plot_data_q1aii$mu[1], y = plot_data_q1aii$sigma2[1], color = "start"), size = 3) +
  geom_point(aes(x = plot_data_q1aii$mu[length(plot_data_q1aii$mu)],
                 y = plot_data_q1aii$sigma2[length(plot_data_q1aii$sigma2)], color = "end"), size = 3) +
  ggtitle("mu vs sigma2 - line") + ylab("sigma2") + xlab("mu") +
  theme(legend.position = "bottom")
plot_mu_vs_sigma_line
# Set the prior hyperparameters suitably
# Evaluate the convergence of the sampler -> need to save mu??s and sigma??s

# I m

##########    BEGIN USER INPUT #################
# Data options
rawData <- rainfall
x <- as.matrix(rainfall$precipitation)

# Model options
```

```r
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')
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)
  }
  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)
# 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
```

```r
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))

# -------------------- Update - start -------------------
# we want to save mu??s and sigma??s for every interation
# we have two misxtured model - so two coloums & nIter (100) as rows
simulated_mu <- matrix(data = NA, nrow = nIter, ncol = 2)
simulated_sigma <- matrix(data = NA, nrow = nIter, ncol = 2)
# -------------------- Update - end -------------------

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 - start -------------------
    simulated_mu[k,] <- mu
  # -------------------- Update - end -------------------

  # 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]))
  }

  # -------------------- Update - start -------------------
    simulated_sigma[k,] <- sigma2
  # -------------------- Update - end -------------------



  # 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)
  }

}

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

# make the plot above with ggplot
ggplot(data = as.data.frame(x)) +
  geom_histogram(aes(x = x))


ggplot(data = as.data.frame(x), aes(x = x)) +
    geom_histogram(aes(y=..density..),
                   colour="black",
                   fill="white",
                   bins=30)

+
  geom_line(aes(x = xGrid, y = mixDensMean))

plot_simulated_mu <- ggplot() +
  geom_line(aes(x = 1:nIter, y = simulated_mu[,1], color = "mu_1")) +
  geom_line(aes(x = 1:nIter, y = simulated_mu[,2], color = "mu_2")) +
  labs(title = TeX("Convergence of simulated $\\mu$ components"), x = "iteration", y = "mu")

plot_simulated_sigma <- ggplot() +
  geom_line(aes(x = 1:nIter, y = simulated_sigma[,1], color = "sigma2_1")) +
  geom_line(aes(x = 1:nIter, y = simulated_sigma[,2], color = "sigma2_2"))+
```

```
  labs(title = TeX("Convergence of simulated $\\sigma$ components"), x = "iteration", y = "simga")


grid.arrange(plot_simulated_mu, plot_simulated_sigma, nrow = 2)
ggplot(data = rainfall, aes(x = precipitation)) +
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)

# \hat{mu} & \hat{sigma}^2 - simulated with rnomal with this values

# ggplot(data = Gibbs_sampling, aes(x = mu)) +
#     geom_histogram(aes(y=..density..),
#                    colour="black",
#                    fill="white",
#                    bins=30)
#
# ggplot(data = Gibbs_sampling, aes(x = sigma2)) +
#     geom_histogram(aes(y=..density..),
#                    colour="black",
#                    fill="white",
#                    bins=30)

# plot data
plot_data_q1c2 <- c()
for (i in 1:nDraws) {
  plot_data_q1c2[i] <-  rnorm(n = 1, mean = Gibbs_sampling$mu[i],
                           sd = sqrt(Gibbs_sampling$sigma2[i]))
}
# the plot
ggplot(data = as.data.frame(plot_data_q1c2), aes(x = plot_data_q1c2)) +
    geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)+
  geom_density(alpha=.2, fill="#FF6666") +
  xlab("values")

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(
# clean environment
rm(list = ls())
### -------------- Lab 3 ### --------------  ###

### --------------  a)

# load and prep data
data <- read.table("eBayNumberOfBidderData.dat", header = TRUE)

y <- data$nBids
```

```r
n <- nrow(data)
# for the R glm function, we do not need a intercept in the data - glm does create it??s own
data_noIntercept <- data[,-2]
colnames_vec <- colnames(data_noIntercept)
# Poisson regression model

Poisson_rm <- glm(formula = nBids ~ .,
                  data = data_noIntercept,
                  family = poisson)
summary(Poisson_rm)


# transform data into data frame
coefficnets_2a <- t(as.data.frame(Poisson_rm$coefficients))

kable(coefficnets_2a) %>%
  kable_styling(latex_options="scale_down")

#### ----------------- b)
# Setting up the prior
# Zellner's g-prior
X = as.matrix(data[,-1])
mu = rep(0, times = ncol(X)) # need to be a vector
Sigma = 100 * (solve(t(X) %*% X)) # Covariance matrix

# Zellner's g-prior for \beta is a multivariate normal distribution with covariance matrix proportional
#?mvrnorm() #sigma => covariance matrix

beta <- mvrnorm(n =1, mu = mu, Sigma = Sigma)

# generate \hat{\beta} and J_y(\hat{\beta}) - optim

# Defining the functions that returns the log posterior (Logistic and Probit models). Note that the fir

# this function must be the one that we optimize on, i.e. the regression coefficients.

# Poisson regression model
LogPostPoisson = function(beta,y,X){
  nPara <- length(beta)
  #linPred <- X%*%betaVect;

  # evaluating the log-likelihood
  logLik <- sum(y * (X %*% beta)  - exp(X %*% beta))

  # evaluating the prior
  logPrior <- dmvnorm(beta, matrix(0,nPara,1), Sigma, log=TRUE)
  if (abs(logLik) == Inf) logLik = -20000
  # Likelihood is not finite, stear the optimizer away from here!


  # add the log prior and log-likelihood together to get log posterior
  return(logLik + logPrior)
}
```

```r
# Different starting values. Ideally, any random starting value gives you the same optimum (i.e. optimu
initVal <- as.vector(rep(0,dim(X)[2]))


OptimResults<-optim(initVal,
                    LogPostPoisson,
                    gr=NULL,
                    y,X,
                    method=c("BFGS"),
                    control=list(fnscale=-1),
                    hessian=TRUE)
# hessian=TRUE -> output includes hessian matrix

# Posterior mode & the negative Hessian at the posterior mode
postMode <- OptimResults$par
postCov <- -solve(OptimResults$hessian) # Posterior covariance matrix is -inv(Hessian)

postMode_df <- t(as.data.frame(postMode, row.names = c("(Intercept)", colnames_vec[-1])))
#rownames(postMode) <- "Values"
kable(postMode_df) %>%
  kable_styling(latex_options="scale_down")

### Version 2

# Metropolis-Hastings


# Random walk Metropolis algorithm - L8S7
RWMSampler = function(logPostFunc, theta0, c, Sigma, n_iter, ...){
  # Input:
  # LogPostFunc - any posterior distribution
  # Sigma - negative Hessian at the posterior mode. (calculated in b))
  # c - tunning parameter

  # init vlaues
  accept <- 0
  reject <- 0


  # Initialize theta^(0)
  # Initialize matrix to save the result
  theta_output <- matrix(data = 0, nrow = n_iter, ncol = length(postMode))
  thetha_i_1 <- theta0

  # given proposal density
  theta_c = mvrnorm(n =1, mu = thetha_i_1, Sigma = c * Sigma) # current theta

  # setp 1 - sample proposal distribution

  for (i in 1:n_iter) {
    # given proposal density
    theta_p <- mvrnorm(n = 1, mu = theta_c, Sigma = c * Sigma)
```

```
    # setp 2 - compute acceptance probability

    # calculate the ratio
    # p(thetha_p|y) / p(theta^(i-1)|y)
    p_theta_p <- logPostFunc(theta_p, ...)
    p_theta_i_1 <- logPostFunc(theta_c, ...)
    ratio <- exp(p_theta_p - p_theta_i_1)

    # find alpha
    alpha <- min(c(1, ratio))

    # setp 3 - set theta^(i) = theta_p and theta^(i) = theta^(i-1)

    U = runif(1)
    if(U < alpha){
      theta_c <- theta_p
      accept = accept + 1
    } else{
      reject = reject + 1
    }

    # save vakue
    theta_output[i,] <- theta_c

    }

    # return results
    result_list <- list("theta" = theta_output,
                        "accept" = accept,
                        "reject" = reject,
                        "accep_prob" = accept/reject)

}


# run random walk
RandomWalk_Metropolis <- RWMSampler(logPostFunc = LogPostPoisson,
                                    theta = postMode,
                                    c = 1,
                                    Sigma = postCov,
                                    n_iter=10000,
                                    y=y, X=X)

# create data frame output
rw_Coefficients <-as.data.frame(RandomWalk_Metropolis$theta)
colnames(rw_Coefficients) <- c("(Intercept)", colnames_vec[-1])
rw_Coefficients$iter <- 1:nrow(rw_Coefficients)
rw_Coefficients_mean <- colMeans(rw_Coefficients[-10])
#rownames(rw_Coefficients_mean) <- "Values"
#kable(rw_Coefficients_mean) %>%
#  kable_styling(latex_options="scale_down")
kable(t(rw_Coefficients_mean)) %>%
  kable_styling(latex_options="scale_down")
```

```r
#plot of the coefficients
plot_list <- list()
plot_data2c <- rw_Coefficients[-10]
for (i in 1:ncol(rw_Coefficients[-10])) { # -10 because we want to remove the iter coloumn
  colname <- colnames(plot_data2c)[i]

  plot <- ggplot(plot_data2c, aes(x = 1:nrow(plot_data2c),y = plot_data2c[,i])) +
    geom_line() +
    ggtitle(paste(colname)) + ylab(paste(colname)) + xlab("x")

  plot_list[[i]] = plot
}

  plot <- ggplot(plot_data2c, aes(x = 1:nrow(plot_data2c))) +
  ggtitle("beta") + ylab("values") + xlab("iteration") +
  geom_line(aes(y = plot_data2c[,1], color = "(Intercept)")) +
  geom_line(aes(y = plot_data2c[,2], color = "PowerSeller")) +
  geom_line(aes(y = plot_data2c[,3], color = "VerifyID")) +
  geom_line(aes(y = plot_data2c[,4], color = "Sealed" )) +
  geom_line(aes(y = plot_data2c[,5], color = "Minblem" )) +
  geom_line(aes(y = plot_data2c[,6], color = "MajBlem")) +
  geom_line(aes(y = plot_data2c[,7], color = "LargNeg")) +
  geom_line(aes(y = plot_data2c[,8], color = "LogBook")) +
  geom_line(aes(y = plot_data2c[,9], color = "MinBidShare")) +
  theme(legend.position = "bottom") +
  scale_fill_discrete(name = "Coefficients")

plot
grid.arrange(plot_list[[1]],plot_list[[2]],plot_list[[3]],
             plot_list[[3]],plot_list[[4]],plot_list[[5]],
             plot_list[[6]],plot_list[[7]],plot_list[[9]],
             ncol = 3, nrow = 3)
grid.arrange(plot_list[[1]],plot_list[[2]],plot_list[[3]],
             ncol = 1, nrow = 3)

grid.arrange(plot_list[[3]],plot_list[[4]],plot_list[[5]],
             ncol = 1, nrow = 3)
grid.arrange(plot_list[[6]],plot_list[[7]],plot_list[[9]],
             ncol = 1, nrow = 3)
# ------------- illustrate
# How code up the Random Walk Metropolis Algorithm in R

# This is the log posterior density of the beta(s+a,f+b) density
LogPostBernBeta <- function(theta, s, f, a, b){
logPost <- (s+a-1)*log(theta) + (f+b-1)*log(1-theta)
return(logPost)
}
# Testing if the log posterior function works
s <- 8;f <- 2;a <- 1;b <- 1
logPost <- LogPostBernBeta(theta = 0.1, s, f, a, b)
print(logPost)
# This is a rather useless function that takes the function myFunction,
# evaluates it at x = 0.3, and then returns two times the function value.
```

```r
MultiplyByTwo <- function(myFunction, ...){
x <- 0.3
y <- myFunction(x,...)
return(2*y)
}
#Let???s try if the MultiplyByTwo function works:
MultiplyByTwo(LogPostBernBeta,s,f,a,b)

# the given R file - NormalMixtureGibbs.R

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

##########    BEGIN USER INPUT #################
# Data options
data(faithful)
rawData <- faithful
x <- as.matrix(rawData['eruptions'])

# Model options
nComp <- 4    # 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')
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)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum of the elements in that co
  return(piDraws)
}
```

```r
# 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)
  }

}

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(

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