"Group Report Lab 3"

Andreas Stasinakis(andst745) & Mim Kemal Tekin(mimte666)

May 13, 2019

Contents

Question 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 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, Gibbs Sampler.

Assume the daily precipitation $y_1, y_2, ..., y_n$ *are independent normally distributed,* $y_1, ..y_n|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ *where both* μ *and* σ^2 *are unknown. Let* $\mu \sim N(\mu_0, \tau_0^2)$ *independently of* $\sigma^2 \sim Inv - X^2(\nu_0, \sigma_0^2)$.

i) Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2|y_1, y_2, ..., y_n)$. *The full conditional posteriors are given on the slides from Lecture 7.*

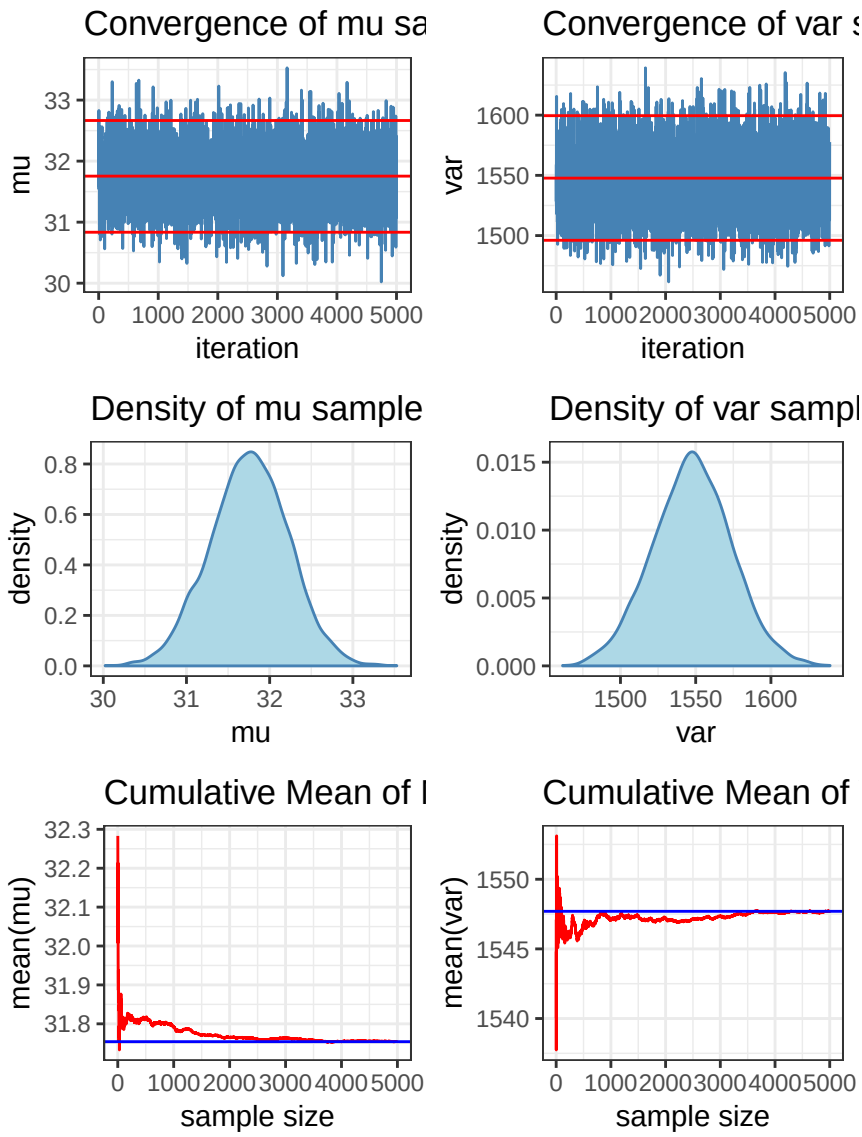
ii) Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

```
r_inv_chisq = function(n=1, df, tau_sq){
  return(df * tau_sq / rchisq(n, df))
}


# Full conditional posterior for the mean(Normal distribution)
# the var in the input is the variance calculated
# from the full conditional posterior below
mu_posterior = function(y, var, mu_0, tausq_0){
  n = length(y)
  # calculate the tau_n from the formulas in lecture 2(normal data, known var)
  # the formula is for the inverse tau
  temp = n/var + 1/tausq_0
  tausq_n = 1/temp

  # For the mean, we also need W(which is also given from the slides)
  w = (n/var) / temp

  # calculate mu_n
  # compute the mu_n using the formulas and the above values
  mu_n = w*mean(y) + (1-w)*mu_0

  # now we have the parameters and we know that our variable is normally distr.
  # draw and return mu parameter
  return(rnorm(1, mu_n, sqrt(tausq_n)))
}

# full conitional posterior for the variance (inv- X distribution)
# The mu in the input is the mu using the above conditional posterior
var_posterior = function(y, mu, v_0, var_0){
  n = length(y)
  # calculate v_n (degrees of freedom)
  v_n = v_0 + n
  # calculate variance in two steps
  num = v_0*var_0 + t(y-mu)%*%(y-mu)
  var = num / v_n
  return(r_inv_chisq(1, v_n, var))
```

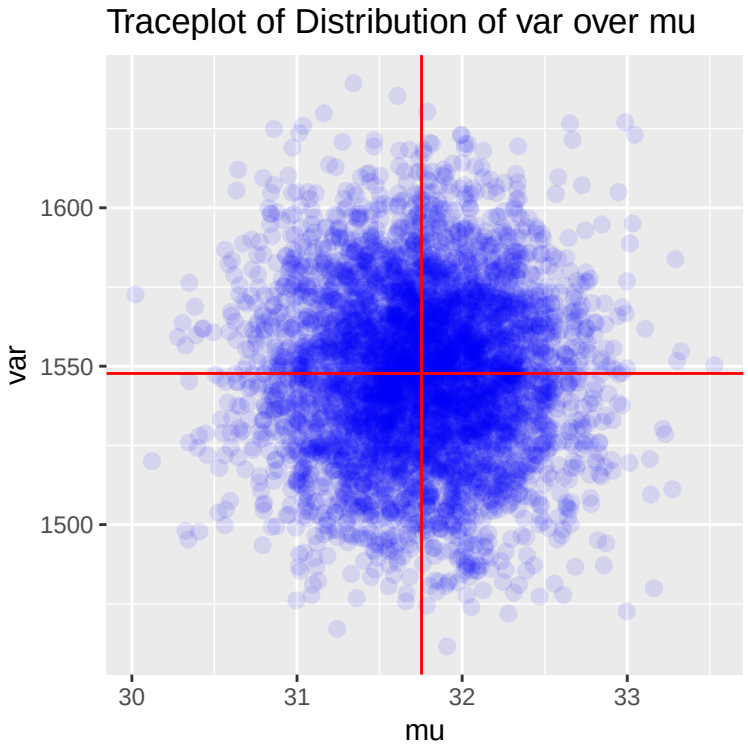
```
}

data = read.table("datasets/rainfall.dat")$V1

# First we have to choose initial values for the parameters
# Users input
# the prior of the mean parameter is normal distributed
mu_0 = 20          # mean of the prior
var_0 = 1          # variance of the prior

# the prior of the variance parameter is inv-chi distributed
tausq_0 = 5        # variance of this prior
v_0 = 1            # df for this prior

nDraw = 5000       #sample size


# gibbs to generate the parameters
# vectors to store the gibbs samples
mu_sample = numeric(nDraw)
var_sample = numeric(nDraw)
# initialize first sample observations randomly

mu_sample[1] = mean(data)
var_sample[1] = var(data)


set.seed(12345)
for(i in 2:nDraw){
  # for the mean, we need the variance from the previous iteration
  # for the first value, we just use the initial
  mu_sample[i] = mu_posterior(y = data, var = var_sample[i-1],
                              mu_0 = mu_0, tausq_0 = tausq_0)
  # for the variance, we use the mean for this iteration
  var_sample[i] = var_posterior(data, mu_sample[i-1],
                             v_0, var_0)
}

library(ggplot2)

df_sample = data.frame(x = 1:nDraw,
                       mu = mu_sample,
                       var = var_sample)

# find the quantiles in order to show the 95% confidence interval of the samples.
interval_mu = quantile(mu_sample, c(0.025, 0.975))
interval_var = quantile(var_sample, c(0.025, 0.975))

# the mean of the sample will give us the expected value of the parameter mean
# store posterior means for parameters
mu_task1a = mean(mu_sample)
var_task1a = mean(var_sample)
```

```r
# we also want the cumsum plots, so we need the cumsum
# we calculate for each i, the mean of the cumsum for all previous iterations
cumu_mean_mu = sapply(1:nDraw, function(x) return(mean(mu_sample[1:x])))
cumu_mean_var = sapply(1:nDraw, function(x) return(mean(var_sample[1:x])))
# cumu_mean_mu = cumsum(mean_sample[1:Ndraws])/c(1:Ndraws)
# cumu_mean_var = cumsum(var_sample[1:Ndraws])/c(1:Ndraws)


###
### Convergence Plots
###
# convergence plot of mu
pl_converge_mu = ggplot(df_sample, aes(x=x)) +
  geom_line(aes(y = mu), color="steelblue") +
  geom_hline(yintercept = mean(mu_sample), color = "red") +
  geom_hline(yintercept = interval_mu[1],  color = "red") +
  geom_hline(yintercept = interval_mu[2], color = "red") +
  labs(title = "Convergence of mu sample",
       x = "iteration", y = "mu") +
  theme_bw()

# convergence plot of var
pl_converge_var = ggplot(df_sample, aes(x=x)) +
  geom_line(aes(y = var), color="steelblue") +
  geom_hline(yintercept = mean(var_sample), color = "red") +
  geom_hline(yintercept = interval_var[1],  color = "red") +
  geom_hline(yintercept = interval_var[2], color = "red") +
  labs(title = "Convergence of var sample",
       x = "iteration", y = "var") +
  theme_bw()


###
### Density Plots
###
# density plot of mu
density_mu = ggplot(df_sample) +
  geom_density(aes(x = mu, y=..density..), fill="lightblue", col="steelblue")+
  labs(title = "Density of mu sample",
       x = "mu", y = "density") +
  theme_bw()

# density plot of var
density_var = ggplot() +
  geom_density(aes(x = var_sample, y=..density..), fill="lightblue", col="steelblue")+
  labs(title = "Density of var sample",
       x = "var", y = "density") +
  theme_bw()

###
## Cumulative Mean Plots
###
# Cumulative Mean plot of mu
```

```r
cumsum_mean = ggplot() +
  geom_line(aes(x=1:nDraw, y = cumu_mean_mu), color = "red") +
  geom_hline(yintercept = mean(mu_sample), color = "blue") +
  labs(title = "Cumulative Mean of Mu Sample",
       x = "sample size", y = "mean(mu)") +
  theme_bw()


# Cumulative Mean plot of  Variance
cumsum_var = ggplot() +
  geom_line(aes(x=1:nDraw, y = cumu_mean_var), color = "red") +
  geom_hline(yintercept = mean(var_sample), color = "blue") +
  labs(title = "Cumulative Mean of Var Sample",
       x = "sample size", y = "mean(var)") +
  theme_bw()

library(gridExtra)

grid.arrange(grobs = list(pl_converge_mu,
                          pl_converge_var,
                          density_mu,
                          density_var,
                          cumsum_mean,
                          cumsum_var), ncol=2)
```




```
# trace plot in order to see the convergence var over mu
pl_trace = ggplot(df_sample) +
  geom_point(aes(x = mu, y = var), color="blue", size=2.5, alpha = 0.1) +
  geom_vline(xintercept = mean(mu_sample), color = "red") +
  geom_hline(yintercept = mean(var_sample), color = "red") +
  labs(title = "Traceplot of Distribution of var over mu")

pl_trace
```



In this task we implement a gibbs sampler which simulates from a joint posterior. We have two posteriors to draw sample for mean and variance. We defined some prior parameters in the beginning and we simulate the joint posterior by using gibbs sample algorithm. In the end we have a sample set that has 5000 mean and variance values. We can see some plots about this samples above.
Firstly, we can see the convergence of mu and var samples over iteration count (to 5000). We can see clearly a quick convergence both of them. It is hard to say even something about burn-in period. The red line in the middle is the mean of the samples and other two red lines are the credible interval of the sample with a 95% confidence. We can see the convergence is stationary between those lines.

We can see the density of samples in the second row of plots. In this plot we can observe that the mean follows a really close distribution to Normal Distribution. For the variance, we know that follows inv-Chi square. It can be said that also the variance is seems like a normal distribution though. This is due to the parameters of the posterior variance.

In the last row of plots we see the cumulative mean of each sample over the sample size. The blue lines represent the mean of final samples. We can see that in both cases, we have a good convergence to the mean of final samples from the initial value. Especially we can see after 3500 iterations, cumulative mean is really close to final mean and it does not change that much. So we can comment it after 3500 iteration the parameters converged to a stationary state.
Also we can see the scatter plot of variance over mean for each drawn pairs during simulation. We can it has a good density in the middle. It looks like a normally distributed and in the middle of plot we can see they converge to real means of each sample.

b) Mixture normal model

Let us now instead assume that the daily precipitation $y_1, y_2, ..., y_n$ follow an iid two-component mixture of normals model:

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

,

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

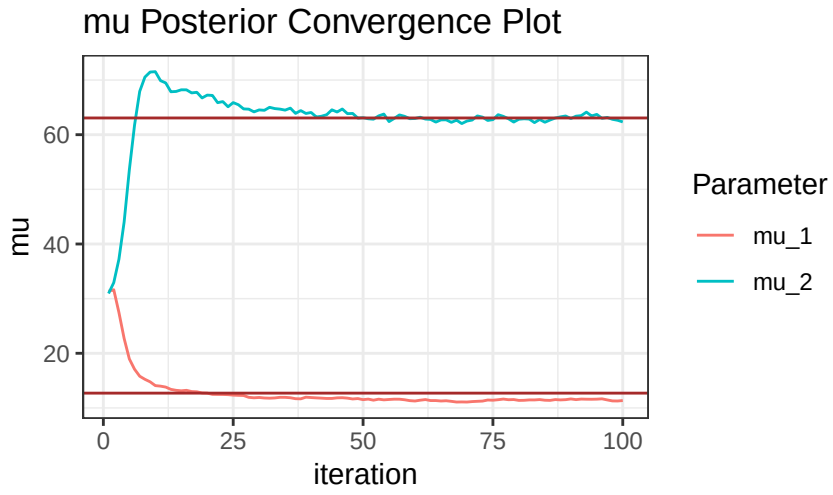
Use the Gibbs sampling data augmentation algorithm in NormalMixtureGibbs.R (available under Lecture 7 on the course page) to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

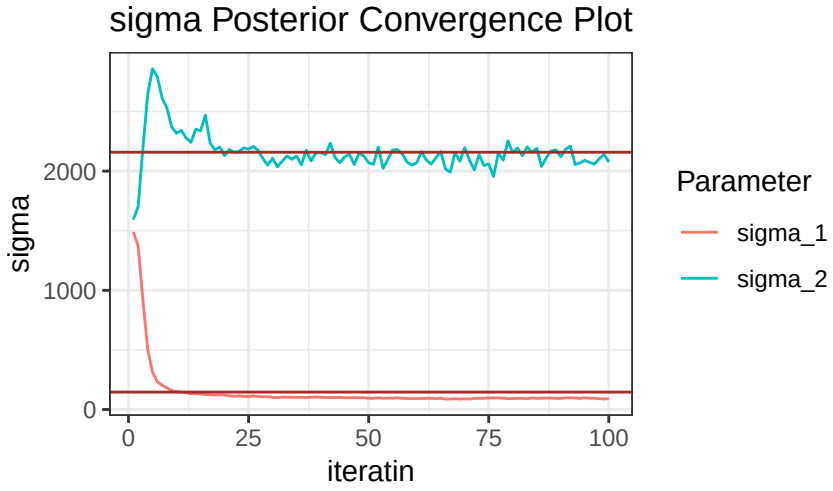
```
####### Defining a function that simulates from a Scaled Inverse Chisq distribution
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

####### 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
# S is a matrix that shows the current observation (row) belongs which component
# output of this function represents it with component numbers
S2alloc <- function(S){
  n <- dim(S)[1]
  alloc <- rep(0,n)
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

set.seed(12345)

x =as.matrix(data)

# Model options
nComp <- 2    # Number of mixture components

# Prior options BE CAREFUL TO CHANGE IT IF nComp CHANGE!!!
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- c(0,50) # 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


# Initial value for the MCMC
nObs <- length(x)
# nObs-by-nComp matrix with component allocations.
```

```r
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))

# set the mu matrix in order to store all mu's from sampling
mu = matrix(numeric(nComp*nIter),
            ncol = nComp)
mu[1,] = quantile(x, probs = seq(0,1, length = nComp))
# set the sigma matrix in order to store all sigma's from sampling
sigma2 = matrix(numeric(nComp*nIter),
                ncol = nComp)
sigma2[1,] <- rep(var(x), 2)


# set pi matrix in order to store all pi values from sampling
pi = matrix(numeric(nComp*nIter),
            ncol = nComp)

probObsInComp <- rep(NA, nComp)


# Setting up the plot
## x scale that we are going to use to plot PDF function.
# we could just use another range, not needed to be that specific one
xGrid <- seq(min(x)-2*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 300)
## minimum/maximum value of our x grid
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)

## in order to store our density function points over or xGrid
## we calculate the current density function (PDF) and we take mean of this
mixDensMean <- rep(0,length(xGrid))
## in order to count the iterations and take the mean of PDFs
effIterCount <- 0
# ylim <- c(0,2*max(hist(x)$density))

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)
  # find how many obs we have in the different components
  nAlloc <- colSums(S)
  # print(nAlloc)

  # Update components probabilities
  pi[k,] <- rDirichlet(alpha + nAlloc)

  # Update mu's
  for (j in 1:nComp){
    precPrior <- 1/tau2Prior[j]
    precData <- nAlloc[j]/sigma2[k,j]
    precPost <- precPrior + precData
    wPrior <- precPrior/precPost
    muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
```

```
    tau2Post <- 1/precPost
    mu[k, j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }

  # Update sigma2's
  for (j in 1:nComp){
    sigma2[k, j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j],
                                   scale = (nu0[j]*sigma2_0[j] +
                                              sum((x[alloc == j] -
                                               mu[k, j])^2))/(nu0[j] + nAlloc[j]))
  }

  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[k, j]*dnorm(x[i], mean = mu[k, j],
                                         sd = sqrt(sigma2[k, j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }
  ## IF we want to see the convergence of two distribution to the data we can
  ## uncomment following code and monitor the convergence
  # if (plotFit && (k%%1 ==0)){
  #   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[k, j],sd = sqrt(sigma2[k, j]))
  #     mixDens <- mixDens + pi[k,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)
  # }
}

# calculate posterior means
sigma_post_mean = colMeans(sigma2)
mu_post_mean = colMeans(mu)
pi_post_mean = colMeans(pi)


# create plot of convergence for pi, mu and sigma
plot_df_mu = as.data.frame(mu)
colnames(plot_df_mu) = c("mu_1", "mu_2")
```

```
plot_df_mu$x = 1:nIter
plot_df_mu = tidyr::gather(plot_df_mu, key = "Parameter",
                           value ="Posterior", -x)

plot_df_sigma = as.data.frame(sigma2)
colnames(plot_df_sigma) = c("sigma_1", "sigma_2")
plot_df_sigma$x = 1:nIter
plot_df_sigma = tidyr::gather(plot_df_sigma, key = "Parameter",
                              value = "Posterior", -x)

plot_df_pi = as.data.frame(pi)
colnames(plot_df_pi) = c("pi_1", "pi_2")
plot_df_pi$x = 1:nIter
plot_df_pi = tidyr::gather(plot_df_pi, key = "Parameter",
                           value = "Posterior", -x)



plot_mu = ggplot(plot_df_mu) +
  geom_line(aes(x=x, y = Posterior, color=Parameter)) +
  geom_hline(yintercept = mu_post_mean[1], color="brown") +
  geom_hline(yintercept = mu_post_mean[2], color="brown") +
  labs(title="mu Posterior Convergence Plot",
       x = "iteration", y = "mu") +
  theme_bw()


plot_sigma = ggplot(plot_df_sigma) +
  geom_line(aes(x=x, y = Posterior, color=Parameter)) +
  geom_hline(yintercept = sigma_post_mean[1], color="brown") +
  geom_hline(yintercept = sigma_post_mean[2], color="brown") +
  labs(title="sigma Posterior Convergence Plot",
       x = "iteratin", y = "sigma") +
  theme_bw()

plot_pi = ggplot(plot_df_pi) +
  geom_line(aes(x=x, y = Posterior, color=Parameter)) +
  geom_hline(yintercept = pi_post_mean[1], color="brown") +
  geom_hline(yintercept = pi_post_mean[2], color="brown") +
  labs(title="pi Posterior Convergence Plot",
       x = "iteratin", y = "pi") +
  theme_bw()

conv_plots = grid.arrange(grobs = list(plot_mu, plot_sigma, plot_pi))
```





In this task we are asked to implement a mixture normal distribution by using Gibbs Sampler again. We used the code provided and we run this method for 2 component. In previous task we had mean (mu) and variance (sigma) parameters, and in this task since we have 2 component we will draw pairs of mu and sigma in every iteration. Another thing we need is the weights of the mixture distribution, this parameter will help us to find the final distribution from components. We express this parameter as pi.

An important decision in order for the chain to converge quickly, is the choice of the hyperparameters. For the mean prior(muPrior), we use the values 0 and 50. It is seems reasonable, because if we take a look to the density of the data, we can expect two distributions with mean close to what we choose. For the variance of the two normals, we use for both the same value 10 in order to make sure that we will capture the true mean. For the prior values of variance, we use the variance of the data, which is our best guess. finally, for the degrees of freedom we choose 4 because we are not so sure that we choose the correct priors so we do not want to give so much importance in the priors. It make take more time in this way, but we definately find the correct results.

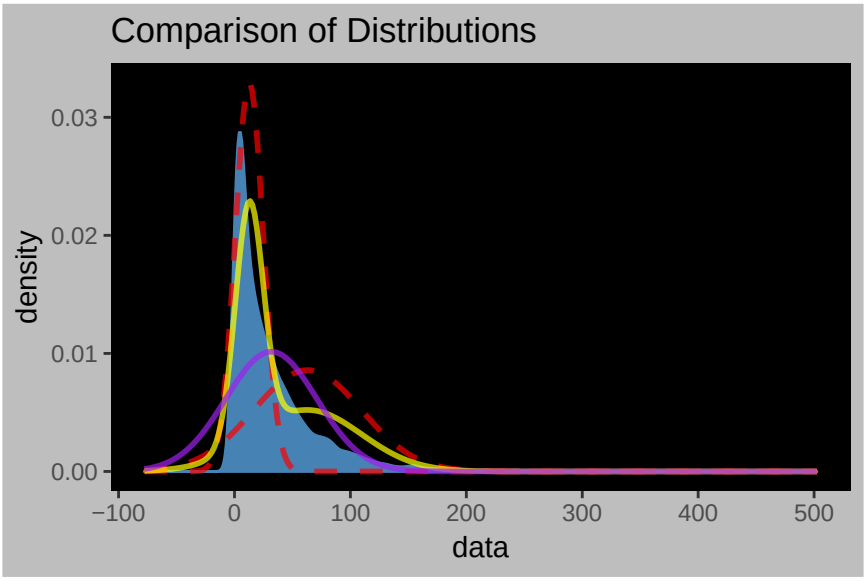
We can see the plots of these parameter pairs above. As we can observe, they converge. While pairs of mu and pi converge around 50 itearations, pair of sigmas converges in 30-40 iterations. And it is clear to see the paremeters of second component (mu_2 and sigma_2) are more stationary than first component's. Once they converge they have much less fluctuation than the first component. So we can say that the data has more density in this component because it is easier to capture for the simulation.

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

```
# caclulate single distributions for components
dens_dist1 = dnorm(xGrid,mean=mu_post_mean[1],
                                 sd = sqrt(sigma_post_mean[1]))
dens_dist2 = dnorm(xGrid,mean=mu_post_mean[2],
                           sd = sqrt(sigma_post_mean[2]))

# Calculate the PDF of mixture distribution
dens_mix = pi_post_mean[1] * dens_dist1 +
  pi_post_mean[1] * dens_dist2

# PDF function of task 1a
dens_task1a = dnorm(xGrid,mean = mu_task1a,
                    sd = sqrt(var_task1a))

len_x = length(xGrid)
plot_df = data.frame(xGrid = rep(xGrid, 4),
                     y = c(dens_task1a,
                           dens_dist1,
                           dens_dist2,
                           dens_mix),
                     type = c(rep("task1a", len_x),
                              rep("dist_1", len_x),
                              rep("dist_2", len_x),
                              rep("mix_dist", len_x))
                     )


# plot of
ggplot() +
  geom_density(aes(x=x, y=..density..), fill="steelblue", col="steelblue",
                 alpha=1) +
  geom_line(aes(x = xGrid, y = dens_dist1), lty=2, color = "red", size=1,
            alpha = 0.7) +
  geom_line(aes(x = xGrid, y = dens_dist2), lty=2, color = "red", size=1,
```

```
          alpha = 0.7) +
geom_line(aes(x = xGrid, y = dens_mix), lty=1, color = "yellow", size=1,
          alpha = 0.7) +
geom_line(aes(x = xGrid, y = dens_task1a), lty=1, color = "purple", size=1,
          alpha = 0.7) +
labs(title = "Comparison of Distributions",
     x = "data") +
theme(panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      plot.background=element_rect(fill = "gray"),
      panel.background = element_rect(fill = 'black'))
```



We can see the comparison of resulting distributions of task 1a and 1b in the plot above. First we can name the lines as following:

- Light blue distribution at the behind: Original Data distribution.
- Purple density line: Result of task 1a for data distribution.
- Red dashed lines: Individual Normal components in the task 1b.
- Yellow density line: Final density in task 1b for data distribution.

First of all, when we check the original data we have a high density observations around 0. After that there is a drop in density until 70-80 and it has a small growth around there. It can be interpreted as that if we want a good fit to original data with normal distribution, we should observe this growth separately because this data has a high peak around zero. Otherwise if we capture that peak, we will have something like the highest dashed red line component which excludes majority of the data after 50. Of course we do not want something like this and when we try to capture an avarage of this original data with only one normal distribution it will be the purple line (task 1a) which excludes completely the peak of the data. So we decide to simulate with 2 component mixed Normal Distributions. With this strategy we are able to capture the big peak around zero and the small peak around 70-80. Of course after this founding components we kind of average them by using the pi parameter that we also simulate for set the weights of components. Finally we get a mixed normal distribution which is visible in the plot as yellow density line. This density (task 1b) captured both of the peaks and it has better fit than result of task 1a.

Question 2 Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model

$$y_i|\beta \sim \text{Poisson}\big[\exp(x_i^T\beta)\big], i = 1, 2, .., n$$

,

where y_i is the count for the ith observation in the sample and xi 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 bidsin each auction. The remaining variables are features/covariates (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. Standardized) MinBidShare (a variable that measures ratio of the minimum selling price (starting price) to the book value. Standardized).

a)using GLM for obtain maximum likelihood estimator of a posson regression model

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?

```
data = read.table("datasets/eBayNumberOfBidderData.dat",
                  stringsAsFactors = F,header = T)

#fit a glm model in the data
#we exclude the the variable const,because glm function add its one intercept
glm_data = data[,-2]
glmModel = glm(formula = nBids~.,data = glm_data, family = "poisson")

glm_coef = glmModel$coefficients
```

In this task we fit a generalized linear model using glm function, in order to estimate the coeffiecients of a Poisson regression model. The glm function estimates the coefficients using the maximum likelihood estimation. We print the optimal coefficients and we also print a summary of the model in order to interpret the results.

```
print(list("MLE coefficients for the poisson Regression model" = glm_coef))
```

```
## $`MLE coefficients for the poisson Regression model`
## (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
```

```
# we can analyze the results using the summary of the model
summary(glmModel)
```

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

```
## 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
```

As one can observe from the summary, the features with 3 stars in the coefficinets table are the features which are significant. In order to be significant, their p value should be less that 0.001. The features that satisfy this condition are the follow: "intercept", "VerifyID", "Sealed", "LogBook", "MindBidShare". Also the feaure "MajBlem" is important for the model but not in the same level of significance as the feautres above. Is it obvious that the feautres that are really close to zero are the ones that do not play important role in the model.

b)Normal approximation with poisson model

Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$ *where* X *is the n x 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:*

$$\beta|y \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

,

where where $\widetilde{\beta}$ *is the posterior mode and* $J(\widetilde{\beta})$ *is the observed Hessian evaluated at the posterior mode.* $\widetilde{\beta}$ *and* $J(\widetilde{\beta})$ *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.).*

```
library(mvtnorm)

#For the normal approximation the first thing we need is the log posterior
#function for log posterior of the parameter
log_posterior = function(betas,mu,X,y,Sigma){

  #we use it many times so better to store it
```

```
  a = X%*%betas

  # for the likelihood we calculate it analytically
  logLike = sum(-log(factorial(y)) + a*y - exp(a))

  #The prior is a multivariate normal distribution with all means equal to 0
  # and variance equal to 100(X^TX)^-1.
  logPrior = dmvnorm(betas, mu, Sigma, log=TRUE)

  #we need the log post so the product of prior and likelihood is now sum
  return(logLike + logPrior)
}

y = as.vector(data$nBids) # response variable
X = as.matrix(data[,-1]) #  all features
nPara = ncol(X) #number of parameters to estimate
mu = rep(0,nPara) # mean of the prior (it is given)
Sigma = (100*(solve(t(X)%*%(X)))) #cov matrix of the prior using the formula

#We have to define the initial values.
#We can choose also other values, but because we want to estimate coef
#it is better to set them all zero
init <- rep(0,nPara)

#We use the optim function in order to estimate the posterior mode
#and the Hessian matrix
OptimResults = optim(init,log_posterior,gr=NULL,mu,X,y,Sigma,
                     method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)


#The output will give us the parameter we want for the approximation
optim_coef = as.vector(OptimResults$par) #theta hat = optimal coef
info_matrix = -solve(OptimResults$hessian) #The covariance matrix for multi. Nor
names(optim_coef) = names(glmModel$coefficients)
colnames(info_matrix) = names(glmModel$coefficients)
rownames(info_matrix) = names(glmModel$coefficients)


# we know that the aproximate posterior dist is a multivariate normal
# with parameters the optimal coef and the - inverse of the hessian matrix
#we also have from optimResults
betas_approx_sample = as.matrix(rmvnorm(n = 1000,
                                  mean = optim_coef,sigma = info_matrix))

# Now we have a Ndraws sample of the vector of coefficients
#In order to compare them with the glm output
#we compute the mean of each column(each parameter).
betas_normal_approx = apply(betas_approx_sample, 2, mean)
```

It is really useful some times, istead of just just estimating the optimal coefficients(for instance using MLE), to "search" for the distribution which those coefficients follow. In order though to do that, we need a known posterior distribution such as Normal or Gamma for example. In this case though, the posterior is not that clear so we use the Normal approximation. The details for this procedure have been discussed in the previous lab. The result can be seen bellow. It seems that the Normal approximation does a really decent job. If we

compare the results with the glm model, we can see that they are really close each other.

```
print(list("MLE coefficients for the poisson Regression model" = glm_coef))
```

```
## $`MLE coefficients for the poisson Regression model`
## (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
```

```
print(
list("coefficients estimated by Normal approximation for the poisson  model" =
        betas_normal_approx))
```

```
## $`coefficients estimated by Normal approximation for the poisson  model`
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.06954130 -0.02203444 -0.39295472  0.44537038 -0.05335492 -0.22420771
##     LargNeg     LogBook MinBidShare
##  0.06897487 -0.11974450 -1.89155541
```

c)Simulate from the actual distribution using RW Metropolis hastings.

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):

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

,

where $\Sigma = J_y^{-1}(\widetilde{\beta})$ 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 this in R.

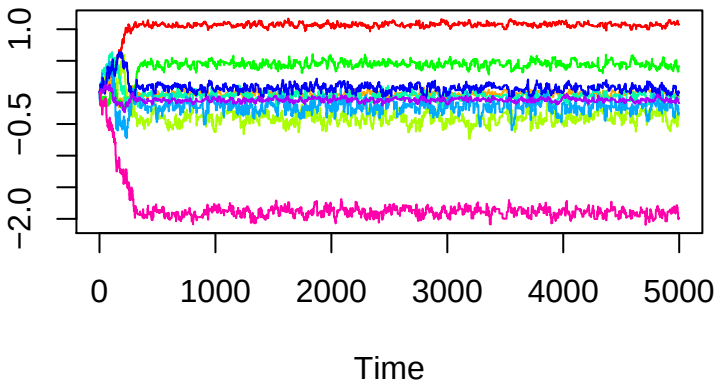
Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

```
set.seed(12345)
library(mvtnorm)
library(tidyr)
library(coda)
#func tion in order to sample from a distribution using the metropolis-hast algo.
#we use the properties of  function objects in R and the ... arquement
# first input arqument is the log - Posterior(log of the target pdf)
#So the user can actual can sample from any posterior wants
#Be careful is the log-posterior because it is more handy for overflow

#For the input we have to different cases
#One is for the standard input(parameters that every case needs)
#Those par are : c : tuning parameter, Ndraws : sample size,
#init: the initial values of pars, Sigma_post: The cov mat of the proposal
#If i want a different proposal i have to change some of the inputs above
```

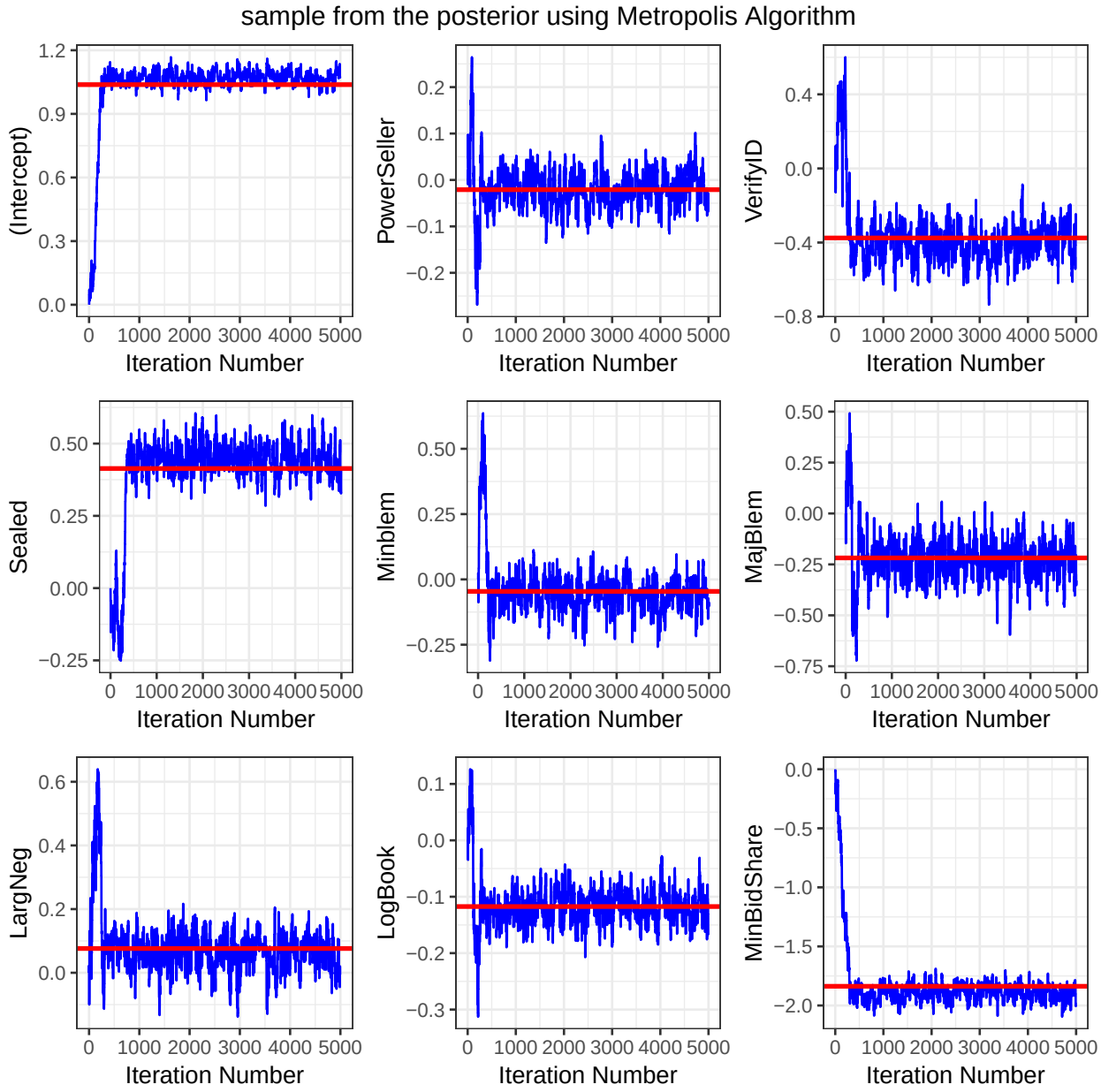
```r
#The second part of inputs are a function and the parameters that this f needs.
#The function is the log posterior of the target distribution
#So the user, as mentioned before, can pick any target dis they want
#Input: LogPostFunct is the log posterior from the target distribution
#This log posterior is a function with parameters that we do not know now
#for that reason we use ..., so the user can add the needed parameters

RWMSampler = function(log_posterior,c,Ndraws,init,Sigma_post,...){

  nParam = length(init)

  coefs = matrix(0,nrow = Ndraws, ncol = nParam)
  coefs[1,] = init

  #for loop depends on the sample size the user wants
  for (i in 2:Ndraws) {

    #FIRST step is to generate from the proposal distribution
    #we use the parameters from the previous iteration
    #in this example the proposal is multivariate Normal with par:
    #mean : vector of coefficients from the previous iteration
    #cov matrix : c * sigma

    temp_coef = as.vector(rmvnorm(n = 1,mean = as.vector(coefs[i-1,]),
                                  c*as.matrix(Sigma_post)))

    #SECOND step is to compute the acceptance probability
    #we have two different formulas here depends on the proposal
    #Here the proposal is symmetri(normal) so the formula is simplier.
    #If we have another proposal, we have also to compute the ratio of it.

    #ratio of the log posterior using the temp parameters
    #The formula for the ratio is p(theta_p/y)/p(theta(iter i-1)/y)
    #but it is always better to work with the log -posterior
    #For that reason we just transform that ratio in to a log-posterior ratio
    #we have to define the log - Posterior then but
    # The log posterior is the same as before
    #Remember the first arquement SHOULD be the parameter we want to evaluate.

    log_ratio =
      exp(log_posterior(betas = temp_coef,...)
          -log_posterior(betas = as.vector(coefs[i-1,]),...))

    #acceptance probability
    a =  min(1, log_ratio)

    #now we need to generate from uniform distribution
    u = runif(1)

    #accept the new theta
    if(u<=a){
      coefs[i,] = temp_coef
      accepted <<-accepted + 1
```

```
    }else{
      coefs[i,] = coefs[i-1,]
    }
  }
  return(coefs)
}

#User input
init = rep(0,nPara) #initial values for the beta coeficients(same as before)
#The proposal is multivariate Normal distribution with parameters
#mean  : the coefficinets from the previous iteration
#covariance matrix
Sigma_post = info_matrix #The covariance matrix for the proposal(it is given)
#tuning parameter REALLY IMPORTANT
c = 0.65

# The prior of the coefficinets we want to estimate
#It is given that it is multivariate normal with parameters(also given)
mu = rep(0,nPara) # mean of the prior
Sigma = (100*(solve(t(X)%*%(X)))) #cov matrix of the prior using the formula
#acceptance probability in order to select parameter c
accepted = 0
Ndraws = 5000
#now the user can ask for every posterior distribution wants
#instead of the ..., they can put as input all the parameters of the logPost.
betas = RWMSampler(log_posterior  = log_posterior,c = c,Ndraws = Ndraws,
                   init = init , Sigma = Sigma,mu = mu,X = X,y = y,
                   Sigma_post = Sigma_post)

#create a df with all the samples and one column for the iterations
df_conv = data.frame(betas)
colnames(df_conv) = names(glm_coef)


#acceptance probability
accept_prob = accepted/Ndraws

#We can also use the coda package for some analysis.
#first we need to convert the sample as a MCMC
MCMC = as.mcmc(df_conv)

#We can calculate the effective sample size
ESS = effectiveSize(MCMC)

#We can also calculate the inefficient factor wich is Ndraws/ESS
IF = Ndraws/ESS

#we take as burn in period 500 iterations
gen_betas = apply(betas[500:Ndraws,],2,mean)


#----------- Assessing MCMC convergence by graphical methods
#we can also create a function in order to use it for different analysis
```

```r
#So we can plot each coefficient in a different plot

my_plots <- function(col, data){
  x = nrow(data)
  ggplot(data = data, mapping = aes(x = 1:x)) +
  geom_line(mapping = aes(y = data[,col]), col = "blue") +
  geom_hline(yintercept = mean(data[,col]), size = 1, col = "red") +
  labs(y = col, x = "Iteration Number") +
  theme_bw()
}


names <- colnames(df_conv)
grobs <- lapply(X = names, FUN = my_plots, data = df_conv)


#using ordinary plot we can plot all the parameters
conv_plot = ts.plot(df_conv,gpars= list(col=rainbow(9)))
```




```r
library(gridExtra)
grid.arrange(grobs = grobs, ncol=3
             , top="sample from the posterior using Metropolis Algorithm")
```



In the previous task, we simulate from an approximation of the posterior distribution. But we can also simulate from the actual posterior distribution using the Metropolis Hastings algorith. In order to do that we need a proposal distribution, which in this case is Normal. The covariance of the proposal depends on the parameter c, which is called tuning parameter. This parameter is *really* important for the model. In order to find a good value for the parameter, we can run the algorithm for different c values and compare. We can see that a value between 0.6-0.7 will be enough. We also calculate the acceptance probability, which is equal to 0.2738. The average acceptance probability should be betwwen 25% and 30%, so c equal to 0.65 seems reasonable. We can mention here that as we increase the number of iterations, the acceptance probability reduces.

As we know, Metropolis Hastings algorithm is a MCMC algorithm. Therefore, we need to take a proportion of the sampling as a burn in period. A chain needs some time in order to converge to the stationery distribution. As a result we do not want to include inaccurate values of the parameters. For that reason, we exclude the n-th first iteration from the final sample. In order to decide how many iterations we should exclude we can use graphical methods. As we can see from the plot, most of the parameters need more than 500 iterations in

order to converge. For tha reason we take as a burn in period 500 iterations.

Our target in this task was to estimate the coefficients for the poisson Regression model. In order to do that, we take the mean for each parameter(excluding the burn in period). As easily can be seen, the obtained coefficients are really close to the previous tasks. We also use the package coda in order to compute the efficient sample size and the inefficiency factor.

```
print(list("acceptance probability"= accept_prob))
```

```
## $`acceptance probability`
## [1] 0.2738
```

```
print(list("Efficient sample size"= ESS))
```

```
## $`Efficient sample size`
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##    15.57012   108.85284    38.20488    23.24692    57.13528   106.40165
##     LargNeg     LogBook MinBidShare
##    60.56681    95.79246    18.65448
```

```
print(list("Inefficiency factor" = IF))
```

```
## $`Inefficiency factor`
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##   321.12780    45.93357   130.87335   215.08226    87.51160    46.99175
##     LargNeg     LogBook MinBidShare
##    82.55347    52.19618   268.03210
```

```
#create a data frame to compare all the coefficients between the 3 methods
df = t(data.frame(glm_coef,betas_normal_approx,gen_betas))
rownames(df) = c("GLM", "Normal Appr", "MH")

knitr::kable(x = t(df),caption = "Estimated Coefficients for all 3 methods")
```

Table 1: Estimated Coefficients for all 3 methods

	GLM	Normal Appr	MH
(Intercept)	1.0724421	1.0695413	1.0707241
PowerSeller	-0.0205408	-0.0220344	-0.0205452
VerifyID	-0.3945165	-0.3929547	-0.4080217
Sealed	0.4438426	0.4453704	0.4486491
Minblem	-0.0521983	-0.0533549	-0.0607300
MajBlem	-0.2208712	-0.2242077	-0.2247789
LargNeg	0.0706725	0.0689749	0.0625589
LogBook	-0.1206776	-0.1197445	-0.1194903
MinBidShare	-1.8940966	-1.8915554	-1.8946779