TDDE07 - Lab 3

Sophie Lindberg - sopli268
Arvid Edenheim - arved490

19-06-09

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

1a) - Normal model

The code for the Gibbs implementation can be seen in appendix 1.

By plotting the trajectories of the sampled Markov chains it can be seen that σ and μ converges around 39 and 32 respectively.

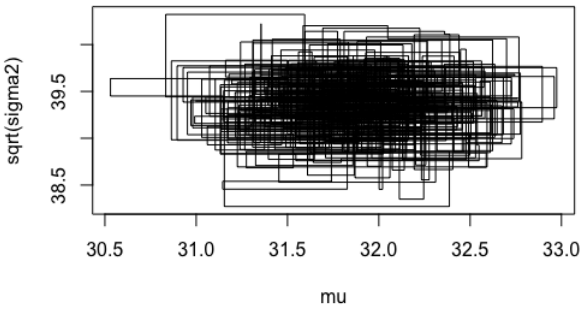


Figure 1: Analyzing the convergence

1b) - Mixture normal model

The results of the Gibbs sampling data augmentation algorithm given in NormalMixtureModel.R resulted can be seen below.

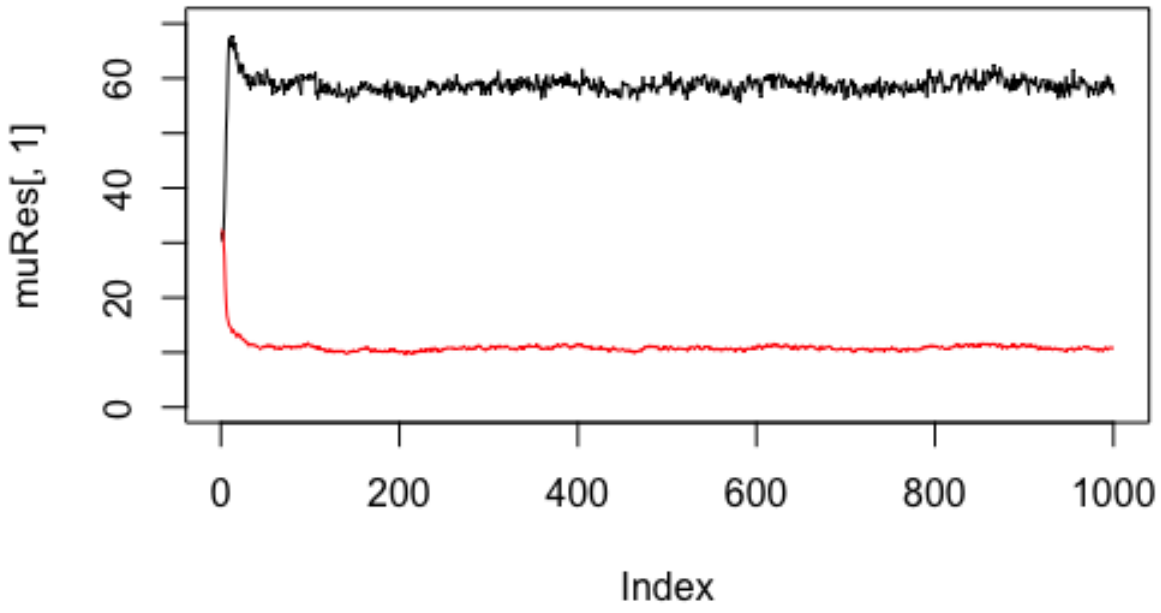


Figure 2: The convergence of mu

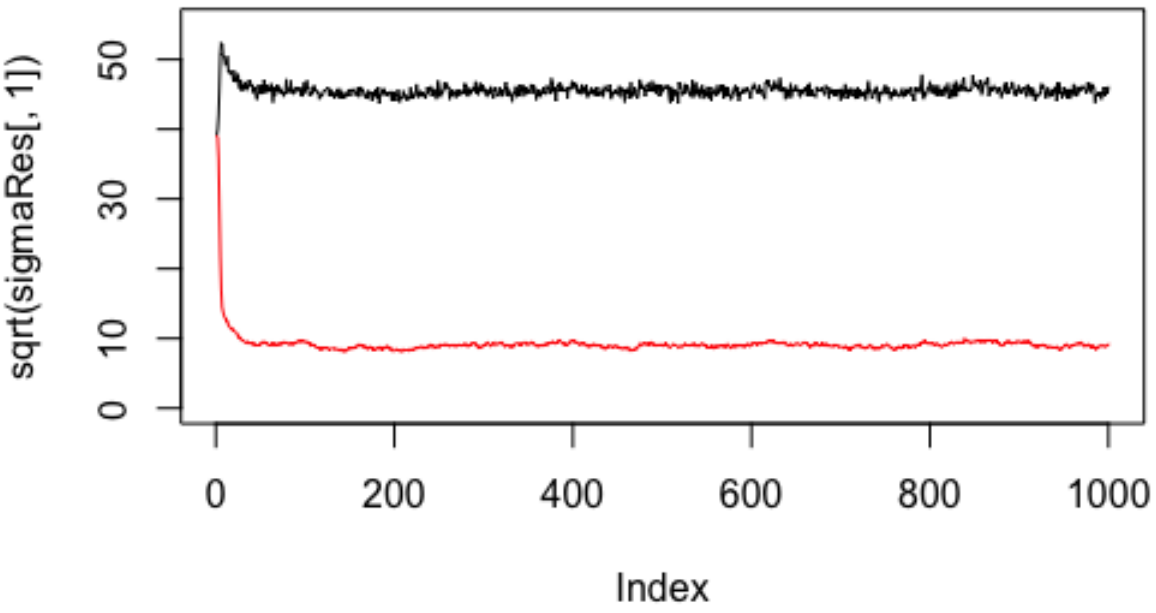


Figure 3: The convergence of sigma

Both μ and σ converged after a few samples.

1c) - Graphical comparison

The figure below shows a comparison between the samplers. The blue line represents the normal density from excercise a, and the green line represents the mixture of normals density from excercise b. Both of the models have limitations when it comes to fit the data.

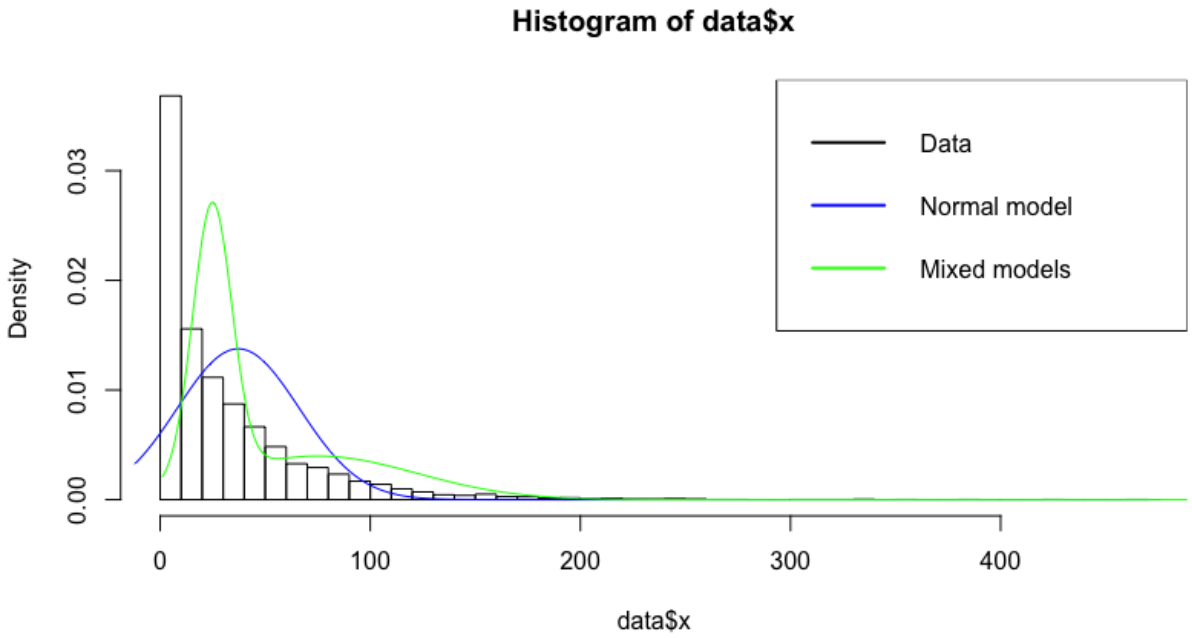


Figure 4: Graphical comparison

2 - Metropolis Random Walk for Poisson regression

2a)

The table below shows the p-values for each parameter. The p-value given for the Z-statistic can be interpreted as how likely it is that a result as extreme or more extreme than that observed would have occured under the null hypothesis. Thus, the covariates MinBidShare, Sealed, and VerifyID seams to be the most significant ones, whereas MinBlem is the least significant one.

Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
4.56e-266	5.76e-01	1.96e-05	1.66e-18	3.85e-01	1.57e-02	2.09e-01	3.09e-05	9.42e-156

2b)

The logistic Poisson regression resulted in the following values for βs and σ^2s.

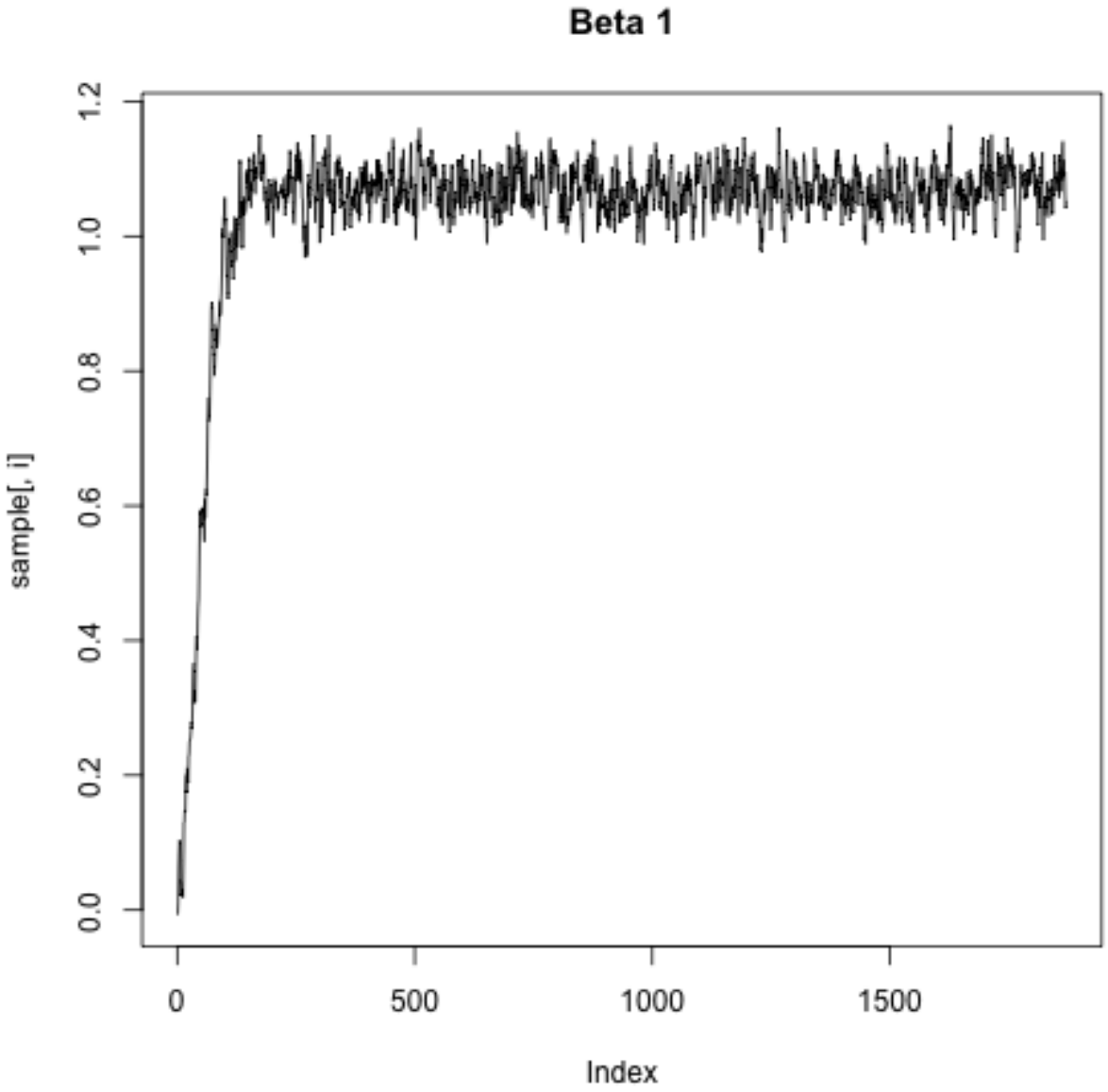
:

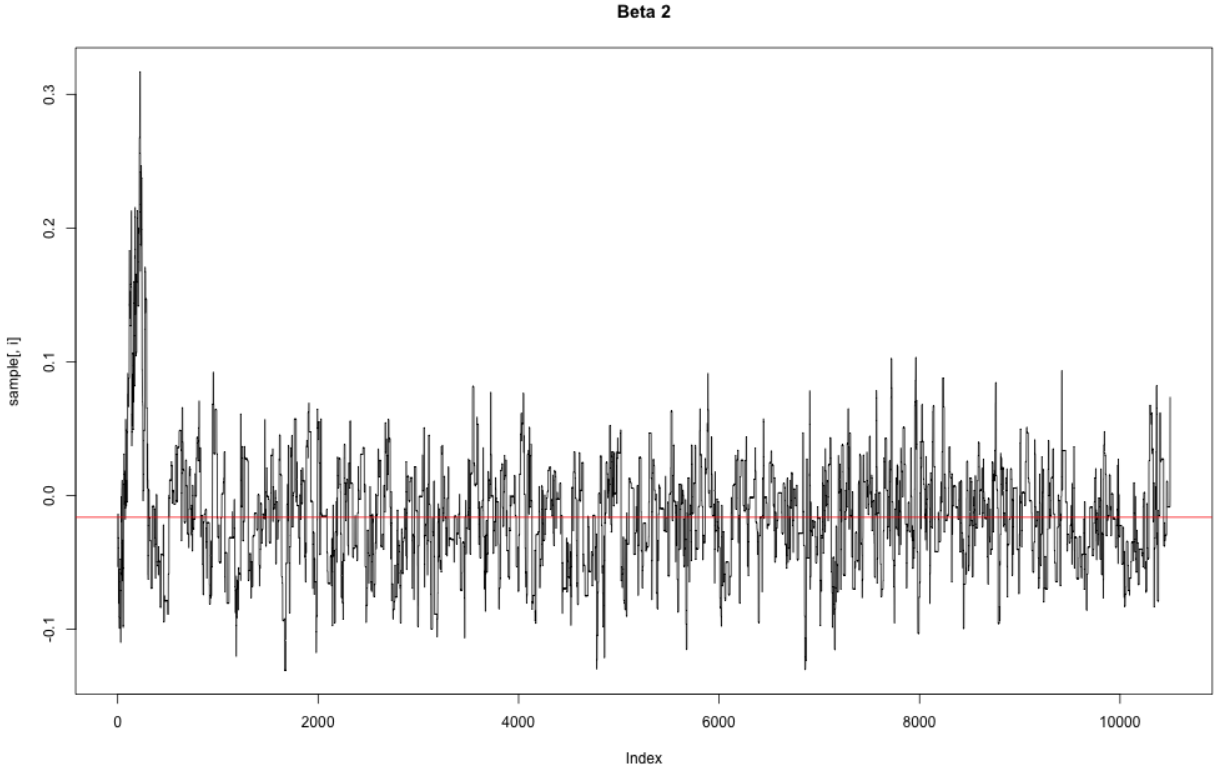
	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
σ	0.0307	0.0367	0.0922	0.0505	0.0602	0.0914	0.0563	0.0289	0.0711
$\hat{\beta}$	1.0698	-0.0205	-0.3930	0.4435	-0.0524	-0.2212	0.0707	-0.1202	-1.8919

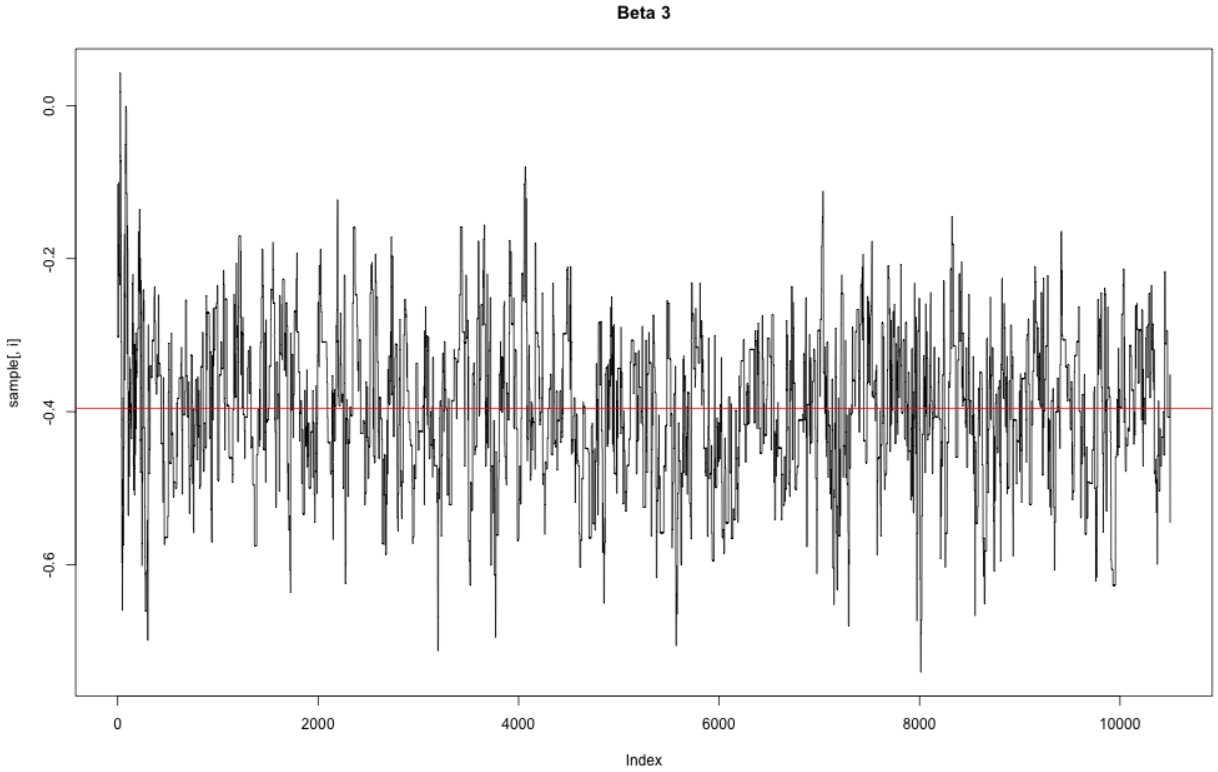
Table 1: Posterior mode and standard diviation

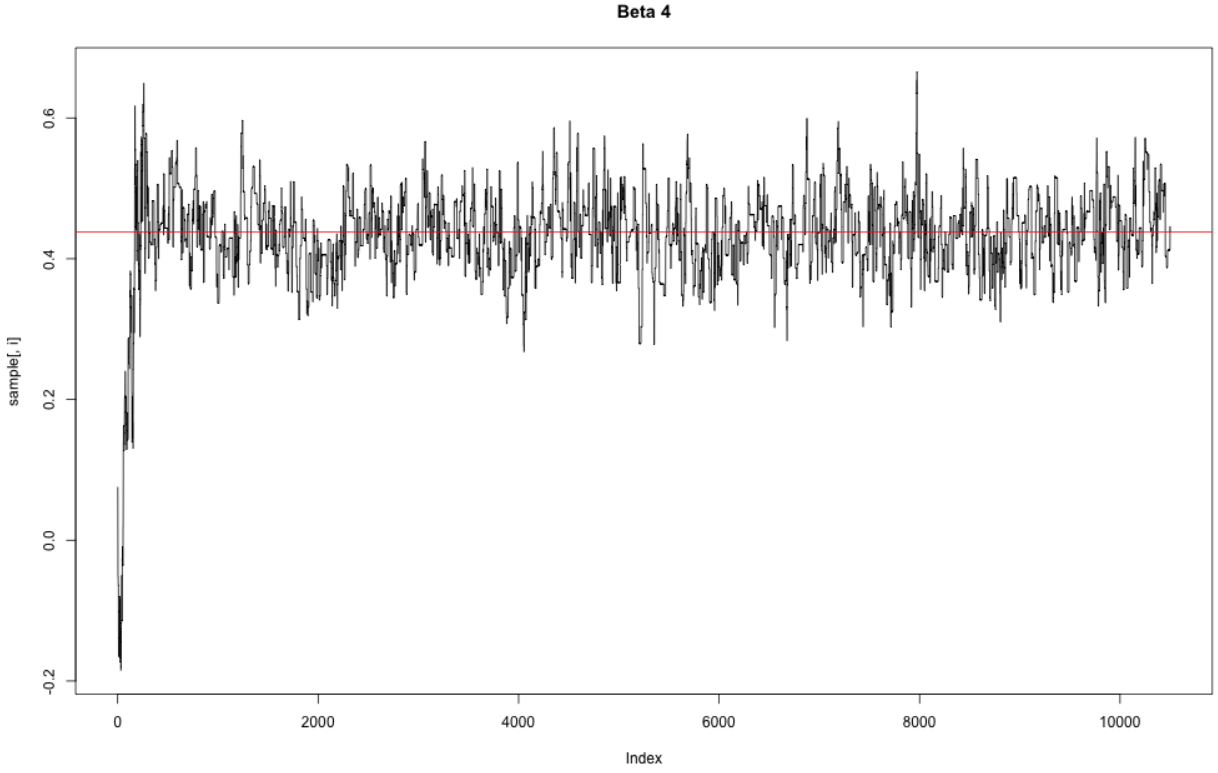
2c)

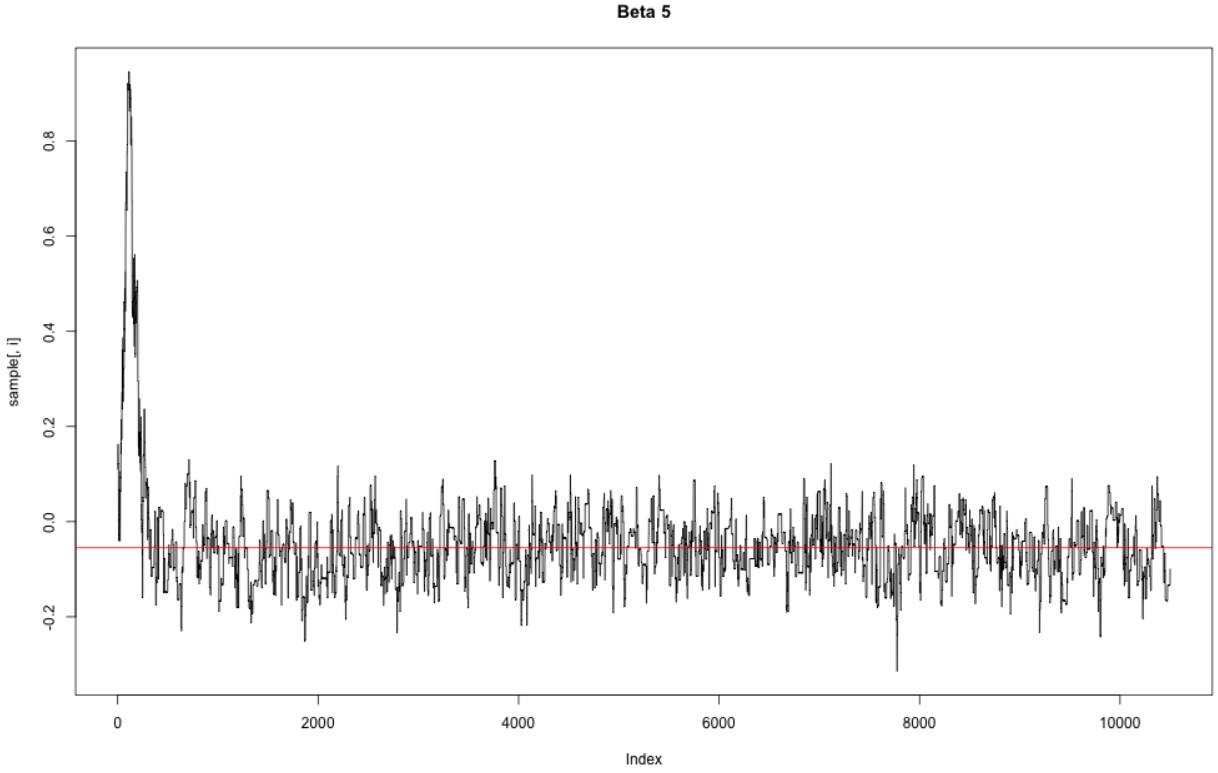
The result below is from 1000 sample draws from the posterior with 500 iterations as burnin. In order to analyze the convergence, the trajectories of the different βs are plotted in the figures below. The red line depicts the posterior mean of each parameter computed with the samples after the burnin phase. As can be seen, the posterior mean is similar to the estimated values obtained by the optim function. The samples are converging after roughly 500 iterations.

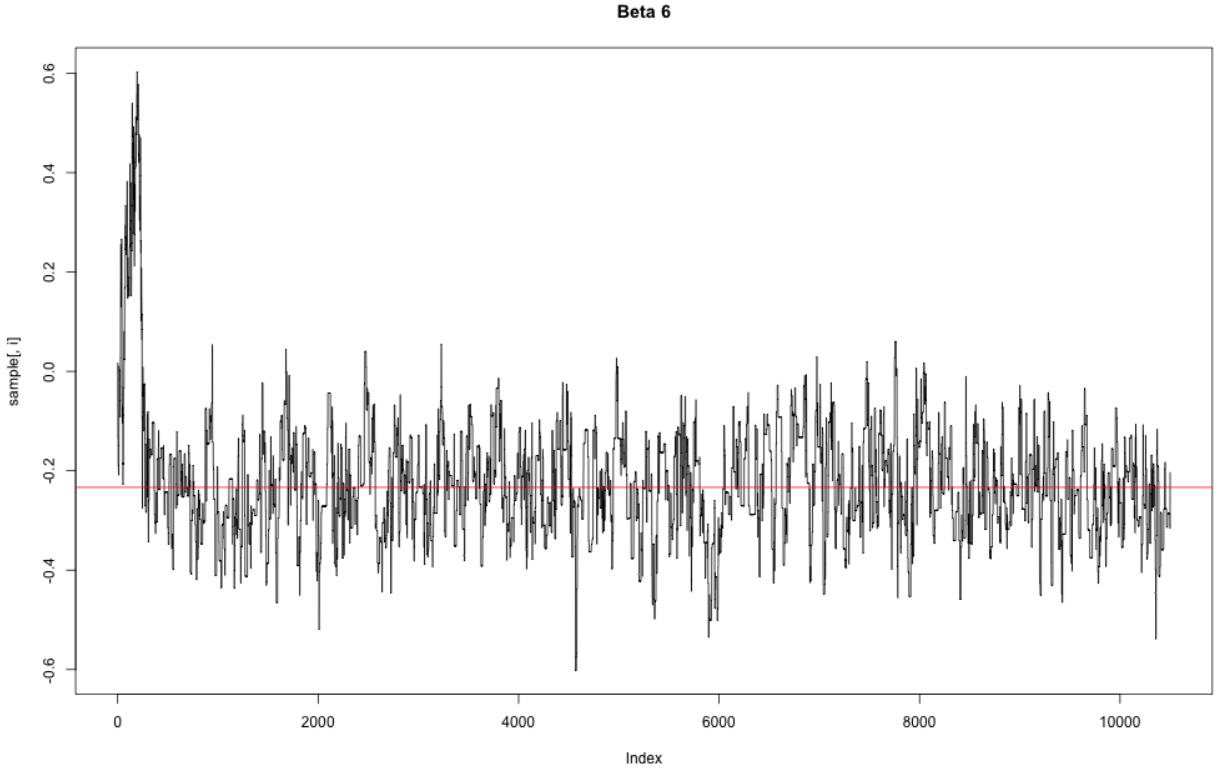


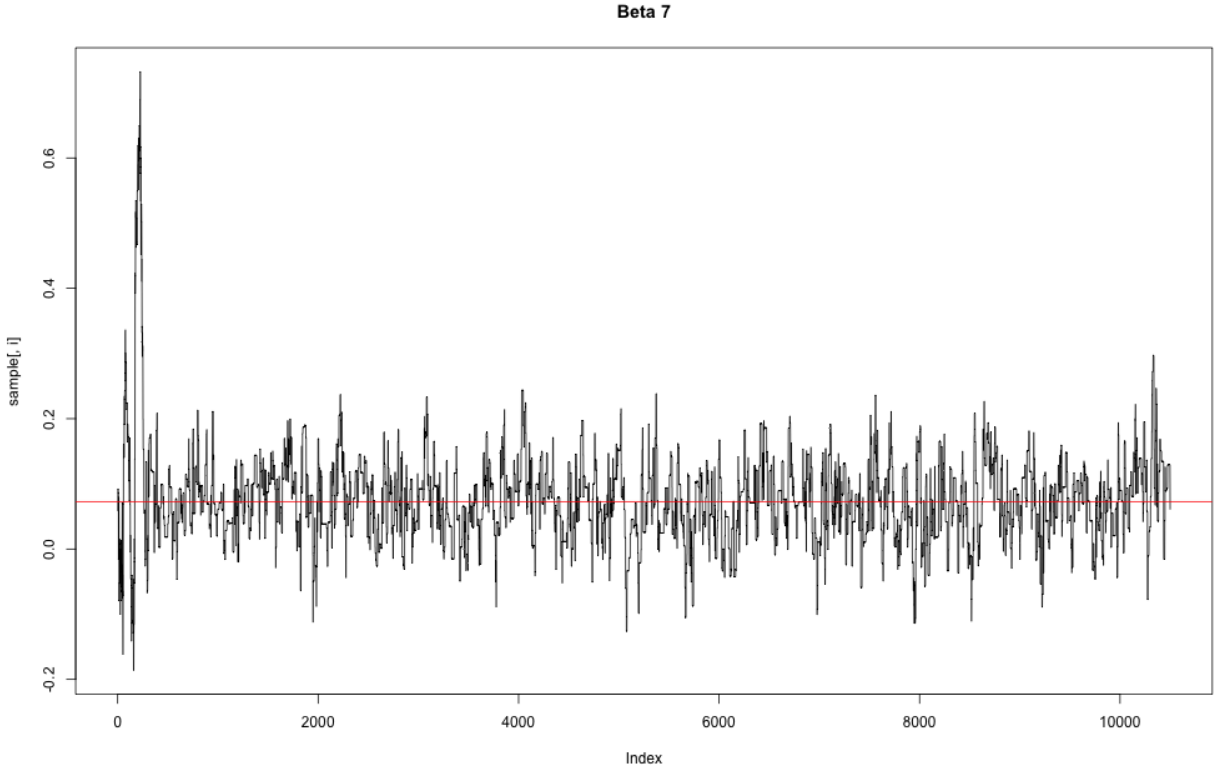

Beta 2
sample[, i]
0.3
0.2
0.1
0.0
-0.1
0
2000
4000
6000
8000
10000
Index

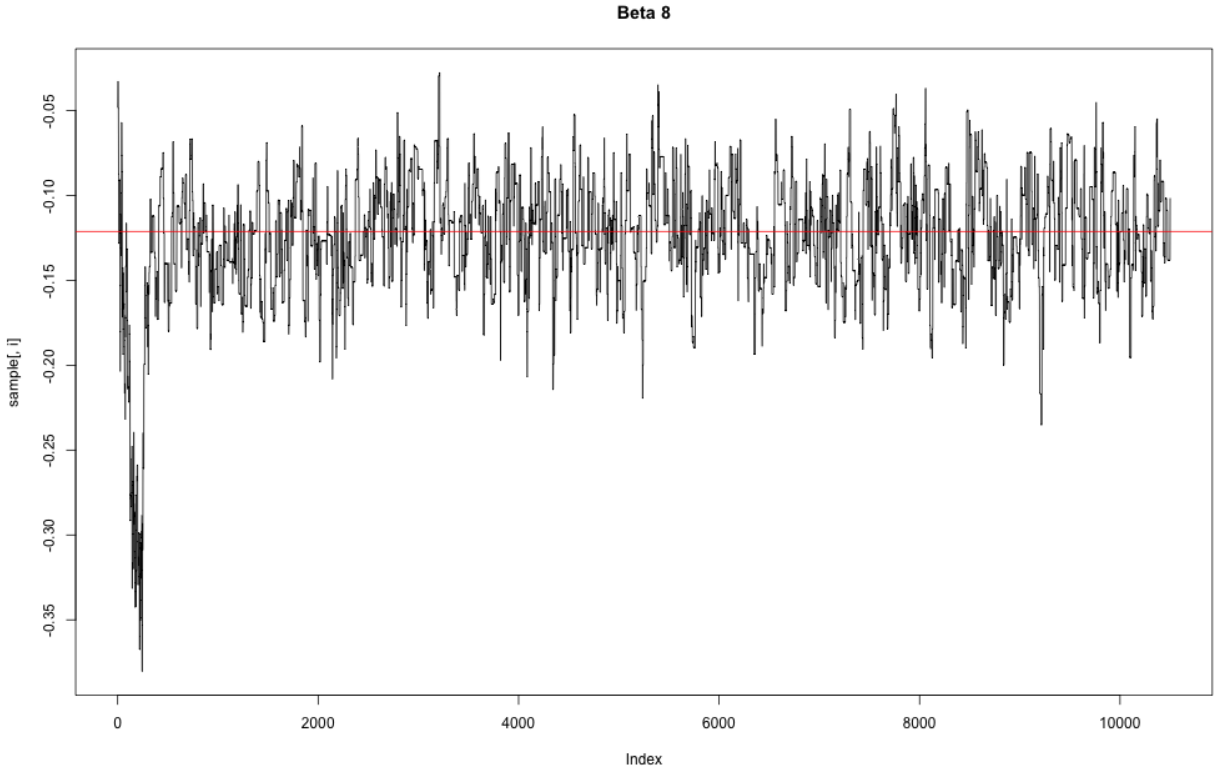

Beta 3
sample[, i]
0.0
-0.2
-0.4
-0.6
0
2000
4000
6000
8000
10000
Index


Beta 4
sample[, i]
-0.2
0.0
0.2
0.4
0.6
0
2000
4000
6000
8000
10000
Index


Beta 5
sample[, i]
-0.2
0.0
0.2
0.4
0.6
0.8
0
2000
4000
6000
8000
10000
Index


Beta 6
sample[, i]
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
0
2000
4000
6000
8000
10000
Index


Beta 7
sample[, i]
0.6
0.4
0.2
0.0
-0.2
0
2000
4000
6000
8000
10000
Index


Beta 8
sample[, i]
-0.05
-0.10
-0.15
-0.20
-0.25
-0.30
-0.35
0
2000
4000
6000
8000
10000
Index

Code - 1

```
data = read.table("rainfall.txt", col.names = "x")

########### A ###########
# parameter init values
mu0 <- 30
tau0 <- 1
v0 <- 0
sigma0 <- 10

n <- nrow(data)
dataMean <- mean(data$x)
vn <- v0 + n

iter <- 1000

# draw mu
drawMu <- function(prevMu, prevSigma) {
  tauSq <- 1/( (n/prevSigma) + (1/tau0^2) )
  w <- (n/prevSigma)/((n/prevSigma) + (1/tau0^2))
  mu <- w*dataMean + (1-w)*mu0
  draw <- rnorm(1, mu, sqrt(tauSq))
  return (draw)
}
#inv chi square
invChiSquare <- function(v, s) {
  return(v*s / rchisq(1,v))
}

# draw sigma
drawSigma <- function(mu) {
  sum <- 0
  for (i in 1:n) {
    sum <- sum + (data[i,1] - mu)^2
  }
  s <- (v0*sigma0 + sum)/(n+v0)
  return(invChiSquare(vn, s))
}

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

currMu <- 32
currSigma <- sigma0
for (i in 1:iter) {
  if(i %% 2 == 0) {
    currMu <- drawMu(currMu, currSigma)
  } else {
    currSigma <- drawSigma(currMu)
  }
  mu <- c(mu, currMu)
  sigma2 <- c(sigma2, currSigma)
```

```
}

## plot trajectories of sampled mu and sigma
plot(mu, sqrt(sigma2),type='l')

# Also consider plotting the trajectories (the sampled values of mu and sigma2) over the iterations.
plot(mu, type = 'l')
plot(sqrt(sigma2), type = 'l')

########## C #########
# Your plot is very small and hard to see.
# The green density looks incorrect, maybe just using the wrong grid.
# Your mixture density is plotted using the default output from Mattias'
# code which computes the mean of the mixture densities computed at each Gibbs iteration.
# This is ok, but not exactly the same as the density requested in the problem,
# which is based on the posterior mean of the parameters.

densityData = density(data$x)

xGrid = seq(min(densityData$x),max(densityData$x),length = length(densityData$x))
ndens = dnorm(xGrid, mean(mu), mean(sqrt(sigma2)))

hist(data$x, 50, freq = FALSE)

lines(xGrid,
      ndens,
      col='blue')

lines(mixDens,
      col = 'green')

legend("topright",
       box.lty = 1,
       legend = c("Data",'Normal model', 'Mixed models'),
       col = c("black",'blue', 'green'),
       lwd = 2)
```

Code - 2

```
library(mvtnorm)

data <- read.table("eBayNumberOfBidderData.txt", header = TRUE)


########## A ##########

#The significance of a parameter is not governed by its estimated
#absolute value, but by e.g. the p-value. Use e.g. summary(fit).

fit <- glm(nBids ~ 0 + ., data, family = poisson)
```

```r
#Obtain p-values
p_values <- coef(summary(fit))[,4]

coeff <- t(fit$coefficients)
plot(abs(coeff), type='h',
     lwd=2,
     xlab = "coefficient index",
     main='Significance of covariates',
     ylab='absolute value of coefficient')

X <- as.matrix(data[,2:10])
## The most significant covariate is minBidShare.

########## B ##########

# Your estimated beta values should be close to the ones from glm() if
# you did it right. Double check this line:

sigmaPrior <- 100 * solve(t(X)%*%X)

logPois <- function(beta, y, X, ...) {
  # log likelihood of poisson model
  if(!is.null(dim(beta))) {
    beta <- beta[1,]
  }

  lambda <- exp(X%*%beta)
  logLik <- sum(dpois(y, lambda, log=TRUE))

  if (logLik == -Inf) {
    logLik <- -2000
  }
  # logLik <- logLik + y[i] * t(beta)%*%x[i,] - exp( t(beta)%*%x[i,] - log(factorial(y[i])))
  # log of prior
  logPrior <- dmvnorm(beta, mean = rep(0, 9), sigma = sigmaPrior, log=TRUE)
  # add
  return(logLik + logPrior)
}

OptimResults<-optim(coeff,logPois,gr=NULL, y = data$nBids,X = X, method=c("BFGS"),control=list(fnscale=

postCov <- -solve(OptimResults$hessian)
st_div <- sqrt(diag(postCov))
betaMode <- OptimResults$par

########## C ##########
# Also plot the MCMC trajectories for the different betas.
# Your results will be different when you fix 2b) also.

gaussianSample <- function(theta, sigma, c) {
  val <- rmvnorm(1, theta, c*sigma)
  return (val)
}
```

```
RWMSampler <- function(c, it, initBeta, fn, ...) {
  accRate <- 0
  sample <- c()
  prev <- gaussianSample(initBeta, postCov, c)
  for (i in 1:it) {
    candidate <- gaussianSample(prev, postCov, c)

    alpha <- min(1, exp( fn(candidate, ...) - fn(prev, ...) ))
    u <- runif(1, 0, 1)
    if (alpha >= u) {
      # accept candidate
      prev <- candidate
      accRate <- accRate + 1
      # as matrix
    }
    sample <- rbind(sample, prev)
  }
  return (sample)
}

# sample with random walk metropolis
burnin <- 500
it <- 10000
nIter <- burnin + it
sample <- RWMSampler(1,nIter, rep(0, 9), logPois, as.vector(data$nBids), X)
hist(sample[,9])
plot(sample[,1],
     sample[,2],
     type='l',
     xlab = expression(beta[1]),
     ylab = expression(beta[2]),
     main = expression("Samples of" ~ beta[1] ~ "and" ~ beta[2])
     )

# plot trajectoroies for betas
for (i in 1:dim(sample)[2]) {
  file <- paste("lab3_2c_",i,".png", sep="")
  png(file)
  plot(sample[,i], type = 'l', main=paste("Beta", i))
  abline(h=mean(sample[burnin+1:it,i]), col="red")
  dev.off()
}
```