TDDE07 - Lab 3

Johannes Hägerlind - johha451
Martin Friberg - marfr370

2021/05/14

1 Assignment 1.

1.0.1 Code Assignment 1a

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# Contains 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 excluded) at Snoqualmie Falls, WSH
data <- read.table('rainfall.dat', header = FALSE, sep="\t")
# Assume the natural log of the daily precipitation {y1, ..., yn} are
# Independent normally distributed ln y1,...ln yn | mu, sigma^2~N(mu, sigma^2),
# Where both mu and sigma^2 are unknown. Let mu ~ N(mu_o, tao_0^2) independently
# of sigma^2 ~ Inv-chi^2(ny_0, sigma_0^2).

################ SubAss a) ################
# Implement a Gibbs sampler that simulates from the joint posterior
mu_rain <- mean(data[,1])
logdata <- log(data)
mu0 <- 1
sigmaSq0 <- 1
v0 <- 1
taoSq0 <- 1
nDraws <- 500
realMu <- mean(logdata[,1])
# mu_n and tao_n defined the same as when sigma^2 is known, i.e
# mu_n = w*x  +(1-w)*mu_0 (lec2, slide 4)
# 1/tao_n^2 = n/sigma^2 + 1/tao_0^2 <=> tao_n = 1/(n/sigma^2 + 1/tao_0^2) (lec2 slide 4)
# w = (n/sigma^2)/(n/simga^2 + 1/tao_0^2) lec2, slide 4
# v_n = v_0 + n (lec3, slide 7)
# n = nr of data points

dim <- as.numeric(length(logdata[,1]))
vn <- v0 + dim

# Function of calculating tao_n^2
calcTaoSq_n <- function (sigmaSquared, n, taoSq_0){
  return(1/(n/sigmaSquared + 1/taoSq_0))
}

calcMu_n <- function (w, x, mu_0){
```

```
  return(w*x + (1-w)*mu_0)
}

calcW <- function(n, sigmaSq, taoSq_0){
  return((n/sigmaSq)/(n/sigmaSq + 1/taoSq_0))
}

invChiSq <- function(vn, sigmaSq_0, v0, n, logdata, mu){
  xDraw=rchisq(1, vn) #rchisq(nr of draws, degrees of freedom)
  return((vn*(v0*sigmaSq_0 + sum((logdata-mu)^2))/(n+v0))/xDraw) # sigma^2 = degrees of freedom * scali
}

# Gibbs sampling
gibbsDraws <- matrix(0,nDraws,2)
gibbsDraws[1,2] <- 1
gibbsDraws[1,1] <- 1

for (i in 2:nDraws){
  # Update mu given sigma^2
    w <- calcW(dim, gibbsDraws[i-1,2], taoSq0)
    mu <- rnorm(n=1, mean=calcMu_n(w, realMu, mu0), sd=calcTaoSq_n(gibbsDraws[i-1,2], dim, taoSq0))
  gibbsDraws[i,1] <- mu

  # Update sigma^2 given mu
    sigmaSq <- invChiSq(vn, sigmaSq0, v0, dim, logdata, gibbsDraws[i,1])

  gibbsDraws[i,2] <- sigmaSq
}

gibbsDraws <- gibbsDraws[-1,]
rhoMu <- acf(gibbsDraws[,1], plot=FALSE)
rhoSig<- acf(gibbsDraws[,2], plot=FALSE)

IFMu <- 1 + 2*sum(rhoMu$acf[-1])
IFSig<- 1+ 2*sum(rhoSig$acf[-1])

# plot(gibbsDraws[,1],
#      gibbsDraws[,2],
#      type='l',
#      xlab="mu",
#      ylab="sigmaSq",
#      main="Iterations"
#
#
# ) #
# points(x=gibbsDraws[1,1],
#        y=gibbsDraws[1,2],
#        pch=16,
#        col="green")
# points(x=gibbsDraws[length(as.numeric(gibbsDraws[,1])),1],
#        y=gibbsDraws[length(as.numeric(gibbsDraws[,1])),2],
#        pch=16,
#        col="red")
```

```
###### Should we use this??? ########################################
nrIter=seq(1, length(as.numeric(gibbsDraws[,1])),1)
plot(nrIter, gibbsDraws[1:nrow(gibbsDraws),1], type="l", xlab=" Iteration",
     ylab="Mu", main="Marginal posterior for mu")
plot(nrIter, gibbsDraws[1:nrow(gibbsDraws),2], type="l", xlab="
Iteration",
     ylab="Sigma", main="Marginal posterior for sigmaSq")
#####################################################################



print(IFMu)
print(IFSig)

print(mean(logdata[,1]))
print(var(logdata[,1]))
```

1.1 Question a)

Implemented a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2|lny_1, ..., lny_n)$. The full conditional posteriors are

$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$ and

$\sigma^2|\mu, x \sim Inv - \chi^2(v_n, \frac{v_0\sigma_0^2+\sum_{i=1}^{n}(x_i-\mu)^2}{n+v_0}$.

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

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

$v_n = v_0 + n$.

To get the inefficiency factors the method acf was used. This method describes how well draws correlate with each other with different lags. The inefficiency factors were then calculated by taking $1 + 2 * \sum_{k=1}^{n} \rho_k$. The inefficiency factors were calculated with the default max.lag of 30.

```
print(IFMu)
```

```
## [1] 1.323064
```

```
print(IFSig)
```

```
## [1] 1.251917
```

The plots below shows us the trajectories of the sampled markov chains. The logarithm of the data gives us a real mean and real variance of:

```
print(mean(logdata[,1]))
```

```
## [1] 2.726208
```

```
print(var(logdata[,1]))
```

```
## [1] 1.882013
```

The convergence of the gibbs sampler is good since our inefficiency factors for both μ and σ are around 1 which is low. Around the time it would take for direct draws. We can also see from the plotted trajectories that the values converge quickly towards the true values.

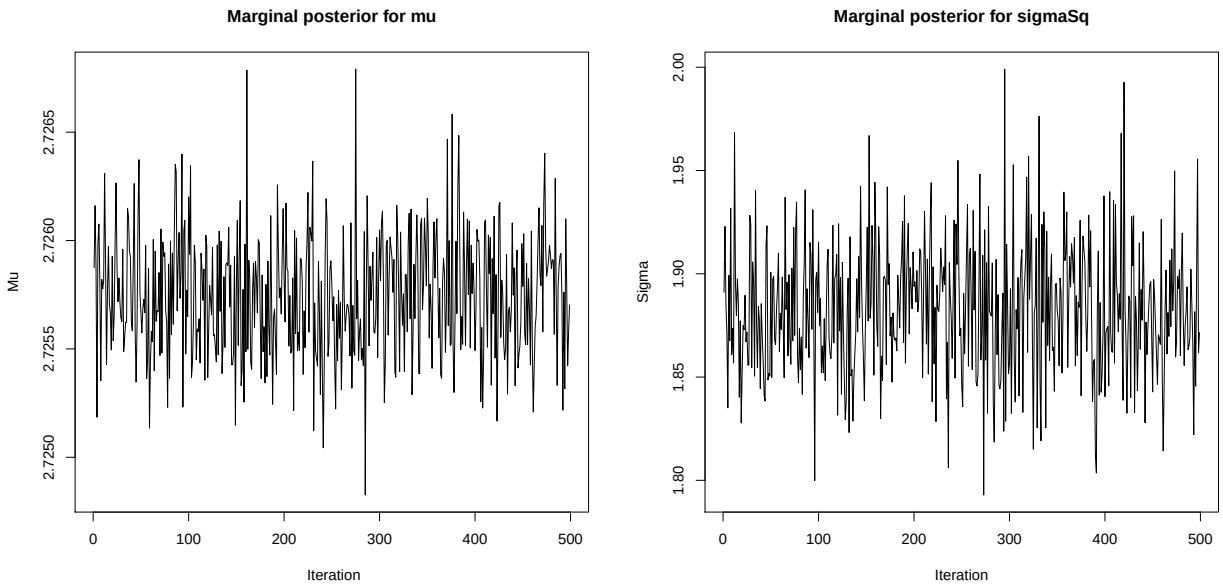


Figure 1: Marginal posteriors for mu and sigma

1.1.1 Code assignment 1b

```
#posteriorData <- exp(gi)
max(as.numeric(data[,1]))
plot(density(as.numeric(data[,1])), xlab="x", main="Density of precipitation", col="blue")
mean((gibbsDraws[,1]))
ranGibbsSampleLn<-rnorm(gibbsDraws[,1],
      mean = gibbsDraws[,1],
      sd = sqrt(gibbsDraws[,2]))
ranGibbsSample<- exp(ranGibbsSampleLn)
ranGibbsSampleDense <- density(ranGibbsSample)
lines(ranGibbsSampleDense, col="red")
legend("topright",
       c("real precipitation", "gibbs sampling"),
       fill=c("blue", "red"),
       box.lwd = 2)
```

1.2 Question b)

As seen in the plot below, the posterior predictive density agrees well with the real data of the precipitation.

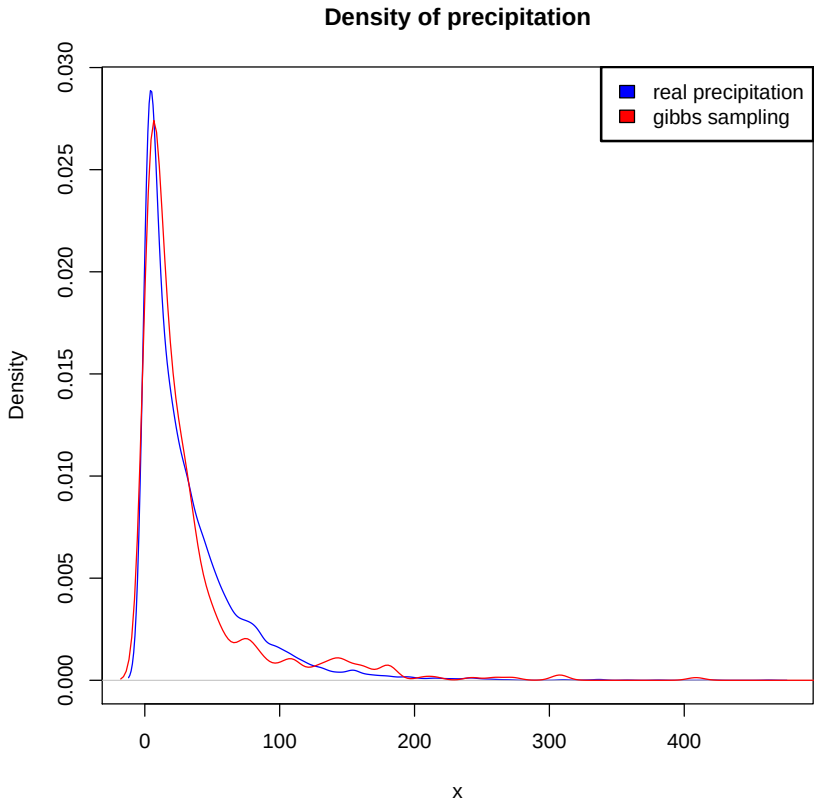


Figure 2: Real Density of precipitation and estimated density of precipitation by gibbs sampling

2 Assignment 2.

2.1 Question a)

```
library("mvtnorm")

eBayData = read.table("./eBayNumberOfBidderData.dat", header=TRUE)

mleRes = glm(nBids ~ .-Const, data=eBayData, family = "poisson")

print(summary(mleRes))
```

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

Significant covariates are (Intercept), VerifyID, Sealed, MajBlem, LogBook and MinBidShare.

2.2 Question b)

```
LogPostPoisson = function(betas,X, y, mu,Sigma){
   linPred = X%*%betas;
   #see book for original formula
   logLik = sum( y*linPred - exp(linPred));
   logPrior = dmvnorm(x=betas, mean=mu, sigma=Sigma, log=TRUE);
   logPost = logLik + logPrior; # propotional to
   return(logPost);
}
```

```r
X = as.matrix(eBayData[, -1])

y = as.matrix(eBayData[, 1])

nCovariates = length(names(eBayData))-1

priorMu = rep(0, nCovariates)

priorSigma = 100 * solve(t(X) %*% X)

initValues <- rep(0, nCovariates)

optimRes <- optim(initValues, LogPostPoisson, gr=NULL, X, y, priorMu, priorSigma,
                  method=c("BFGS"), control=list(fnscale=-1), hessian=TRUE)

betaMode = optimRes$par

print(betaMode)
```

```
## [1]  1.06984118 -0.02051246 -0.39300599  0.44355549 -0.05246627 -0.22123840
## [7]  0.07069683 -0.12021767 -1.89198501
```

```r
negInvHessian = -solve(optimRes$hessian)

print(negInvHessian)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  9.454625e-04 -7.138972e-04 -2.741517e-04 -2.709016e-04 -4.454554e-04
##  [2,] -7.138972e-04  1.353076e-03  4.024623e-05 -2.948968e-04  1.142960e-04
##  [3,] -2.741517e-04  4.024623e-05  8.515360e-03 -7.824886e-04 -1.013613e-04
##  [4,] -2.709016e-04 -2.948968e-04 -7.824886e-04  2.557778e-03  3.577158e-04
##  [5,] -4.454554e-04  1.142960e-04 -1.013613e-04  3.577158e-04  3.624606e-03
##  [6,] -2.772239e-04 -2.082668e-04  2.282539e-04  4.532308e-04  3.492353e-04
##  [7,] -5.128351e-04  2.801777e-04  3.313568e-04  3.376467e-04  5.844006e-05
##  [8,]  6.436765e-05  1.181852e-04 -3.191869e-04 -1.311025e-04  5.854011e-05
##  [9,]  1.109935e-03 -5.685706e-04 -4.292827e-04 -5.759169e-05 -6.437066e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.772239e-04 -5.128351e-04  6.436765e-05  1.109935e-03
##  [2,] -2.082668e-04  2.801777e-04  1.181852e-04 -5.685706e-04
##  [3,]  2.282539e-04  3.313568e-04 -3.191869e-04 -4.292827e-04
##  [4,]  4.532308e-04  3.376467e-04 -1.311025e-04 -5.759169e-05
##  [5,]  3.492353e-04  5.844006e-05  5.854011e-05 -6.437066e-05
##  [6,]  8.365059e-03  4.048644e-04 -8.975843e-05  2.622264e-04
##  [7,]  4.048644e-04  3.175060e-03 -2.541751e-04 -1.063169e-04
##  [8,] -8.975843e-05 -2.541751e-04  8.384703e-04  1.037428e-03
##  [9,]  2.622264e-04 -1.063169e-04  1.037428e-03  5.054757e-03
```

2.3 Question c)

```r
# Task 2c)
Metropolis = function(nSamples, nBurnIns, c, logPostFunction, theta, ...){

   nParameters = length(theta)
```

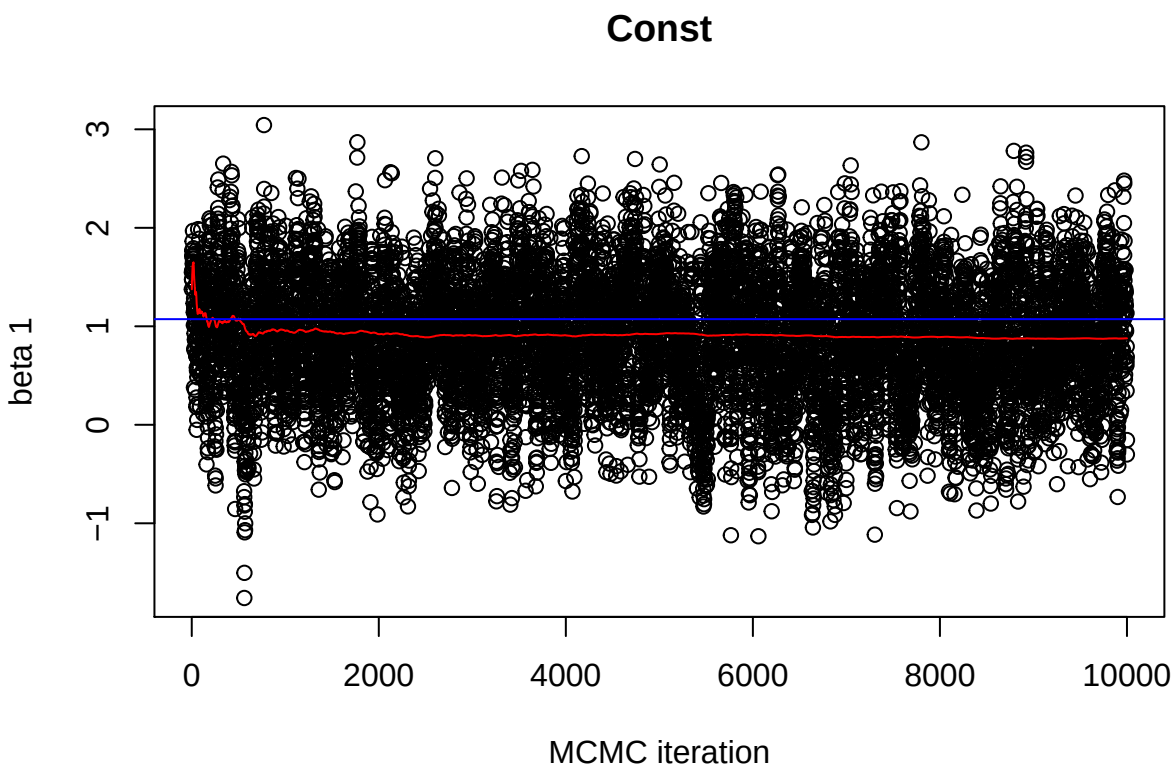
```
   resThetas = matrix(0, nSamples, nParameters)

   thetaPrev = theta

   varianceProposal = c * negInvHessian

   nAccepted = 0

   for( i in 0 : nBurnIns + nSamples){

      thetaProposal = as.vector(rmvnorm(1, mean = thetaPrev, sigma = varianceProposal))

      logPostThetaPrev = logPostFunction(thetaPrev, ...)
      logPostThetaProposal = logPostFunction(thetaProposal, ...)
      alpha = min(1, logPostThetaProposal / logPostThetaPrev)

      u = runif(1, 0, 1)

      if (u < alpha){
         thetaPrev = thetaProposal
         nAccepted = nAccepted + 1
      }

      if(i > nBurnIns){
         resThetas[i-nBurnIns, ] = thetaProposal
      }
   }

   acceptance_rate = nAccepted/nSamples

   print(paste("acceptance rate = ", acceptance_rate))

   return(resThetas)
}

c = 150.0
nSamples = 10000
nBurnIns = 10000

betaSamples = Metropolis(nSamples, nBurnIns, c, LogPostPoisson, betaMode, X, y, priorMu, priorSigma)
```

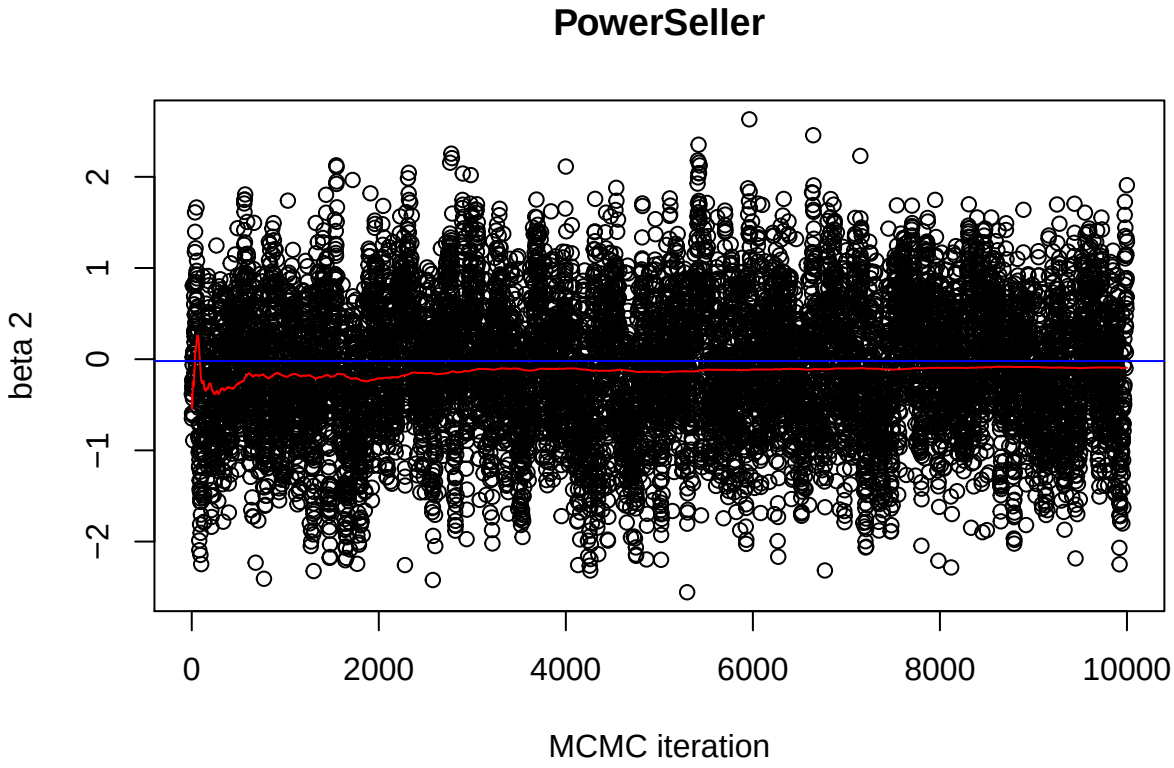
```
## [1] "acceptance rate =  0.3084"
```

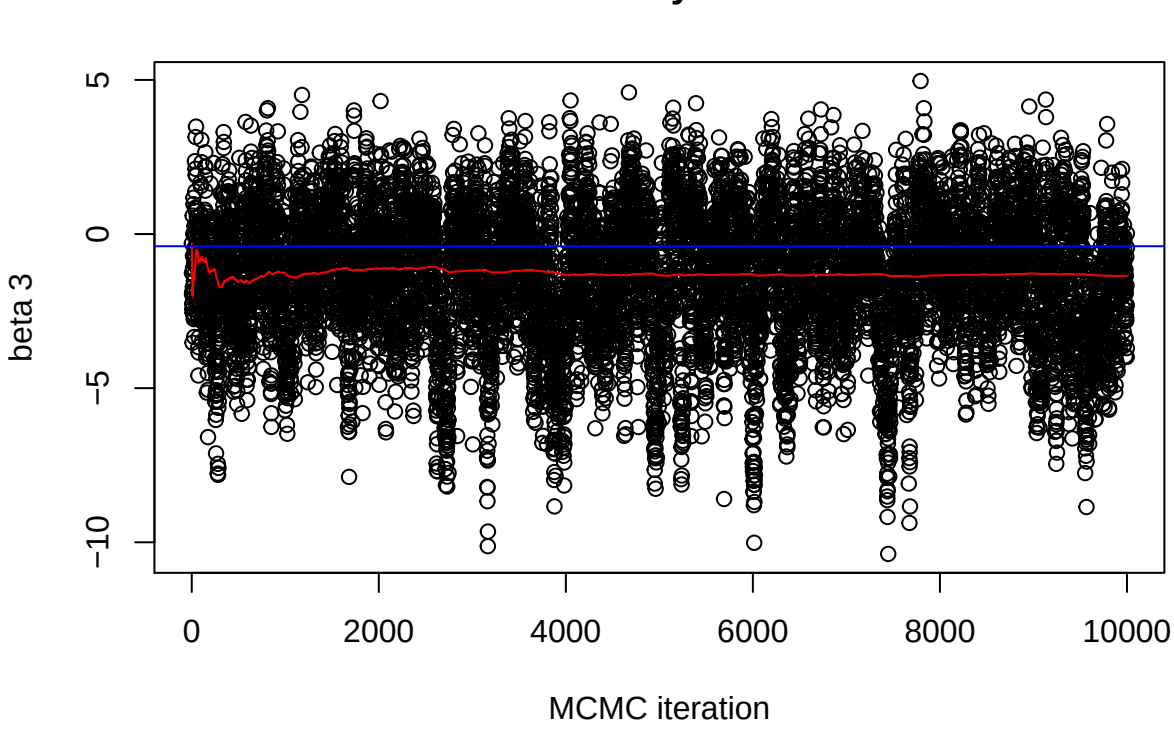
Below is some graph that shows the convergence in terms of the pattern of the samples. Ther black dots are the samples, the blue line is the actual mode of the true posterior, the red curve shows the convergence towards the mode as more and more samples are drawn. As we can see the mode of the betas converges quite well in most cases, although not perfect.

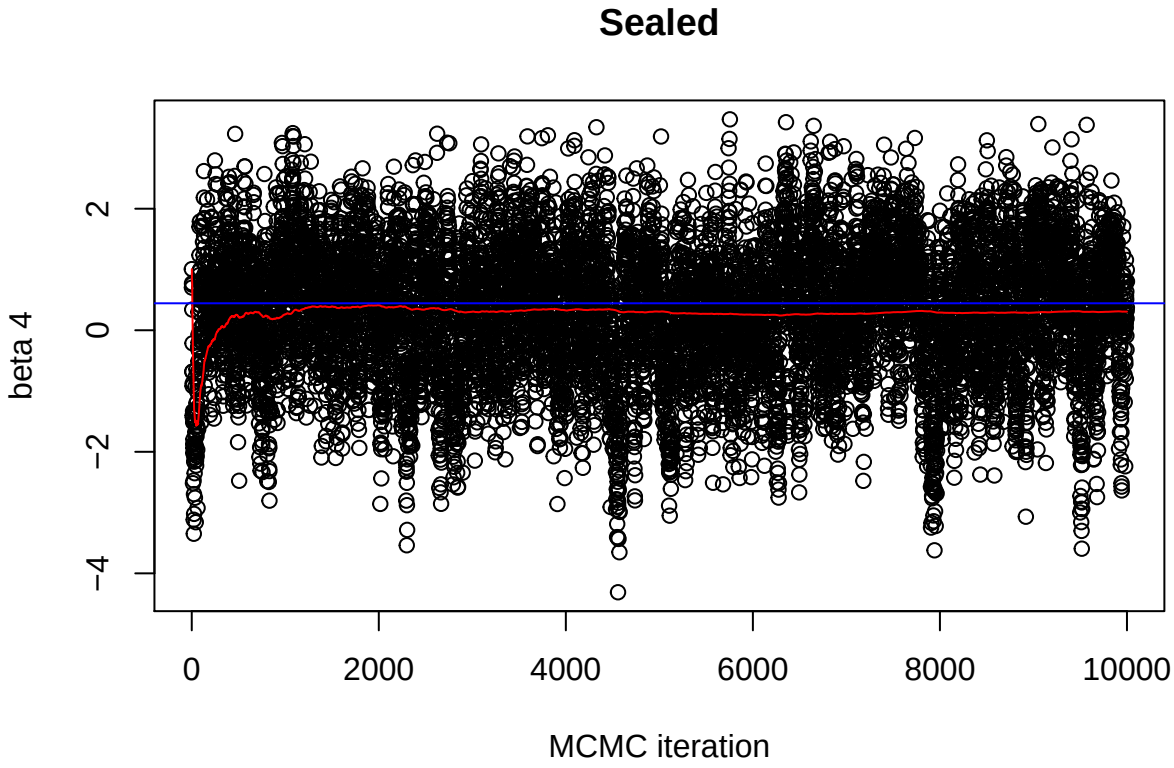
```
xs = 1:nSamples
for(i in 1:nCovariates){
   plot(x=xs, y=betaSamples[,i], main=names(eBayData)[i+1], xlab="MCMC iteration", ylab=paste("beta", i

   betaMean = mleRes$coefficients[i]

   cumsumMean = rep(NA, nSamples)
```

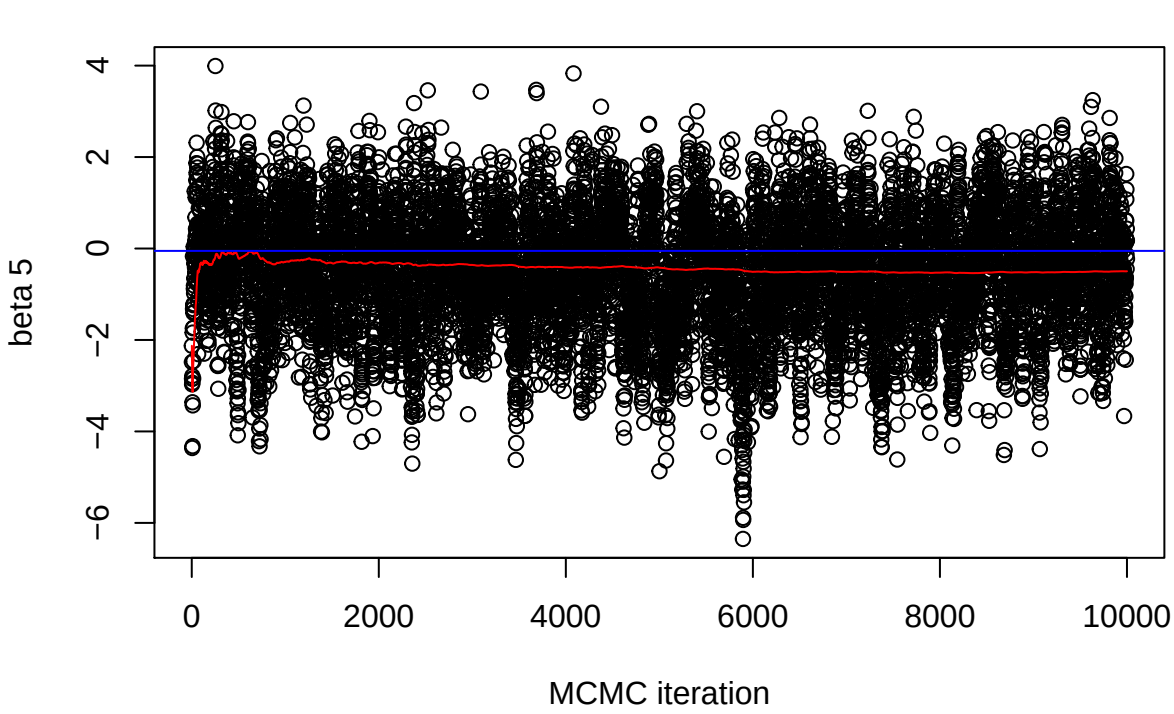
```
   myCumsum = cumsum(betaSamples[,i])

   for(j in 1:nSamples){
      cumsumMean[j] = myCumsum[j]/j

   }

   lines(x=xs, y=cumsumMean, col="red")
   abline(h=betaMean, col="blue")
}
```

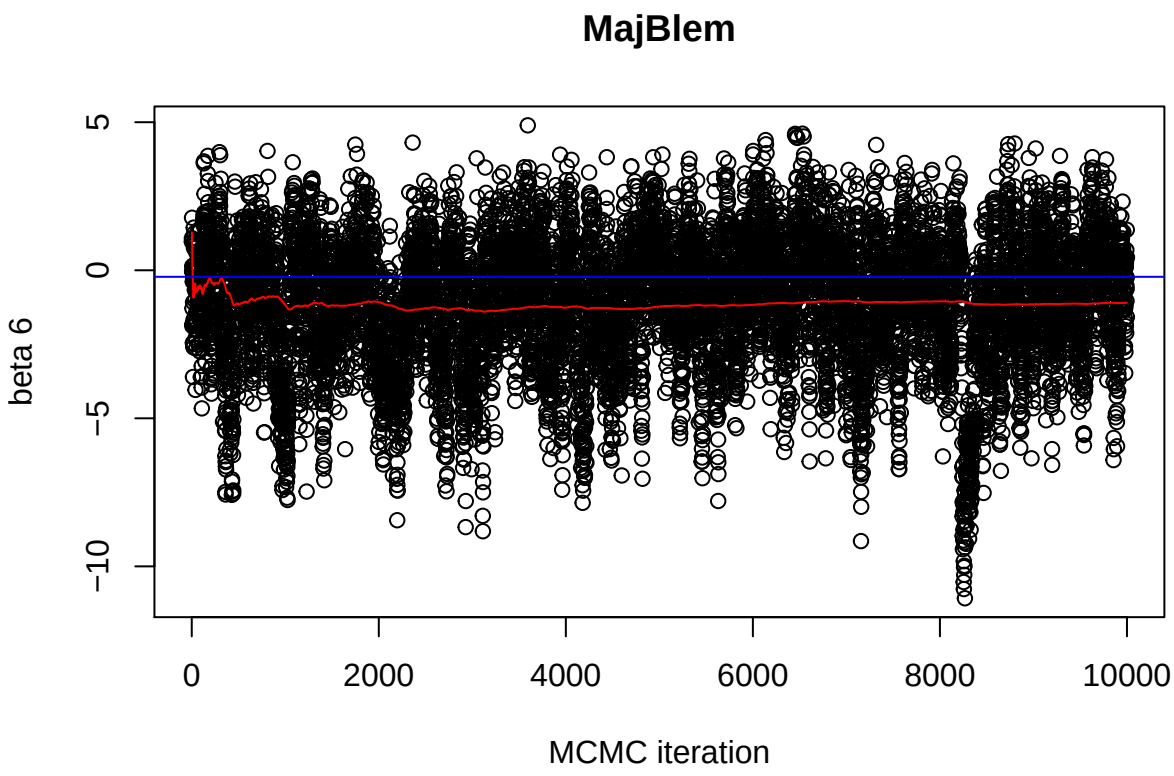


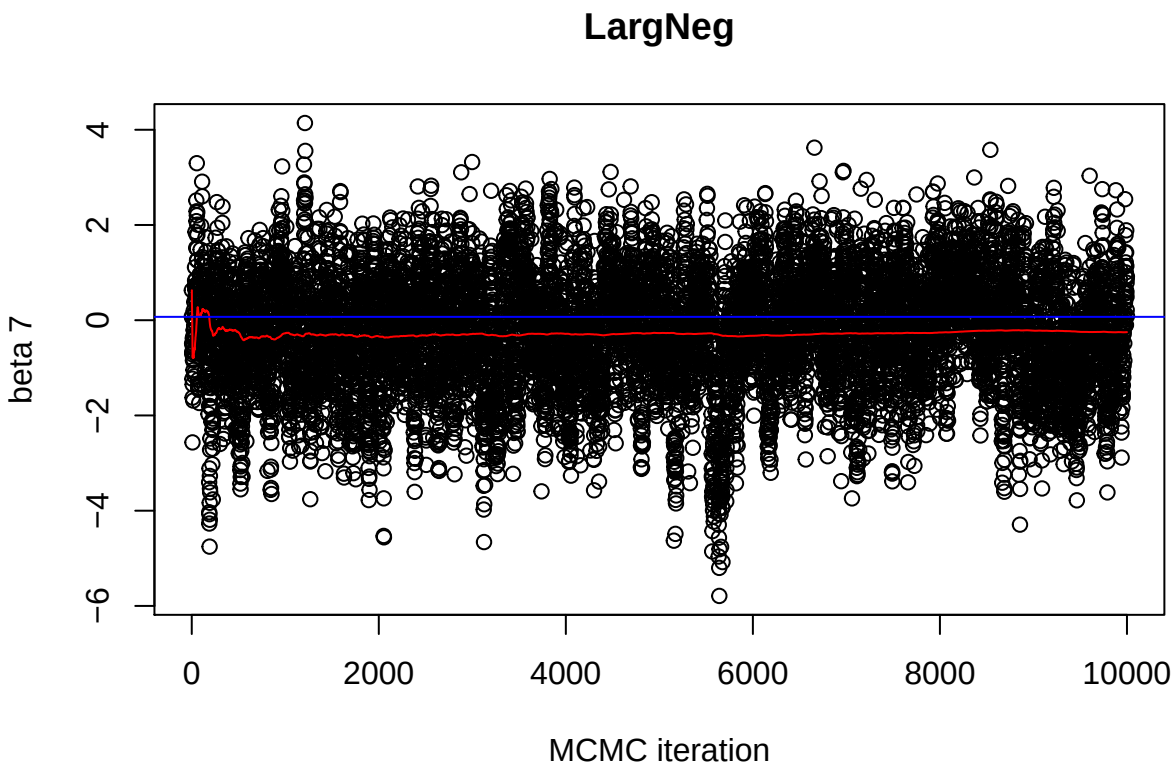

PowerSeller
beta 2
2
1
0
−1
−2
0
2000
4000
6000
8000
10000
MCMC iteration

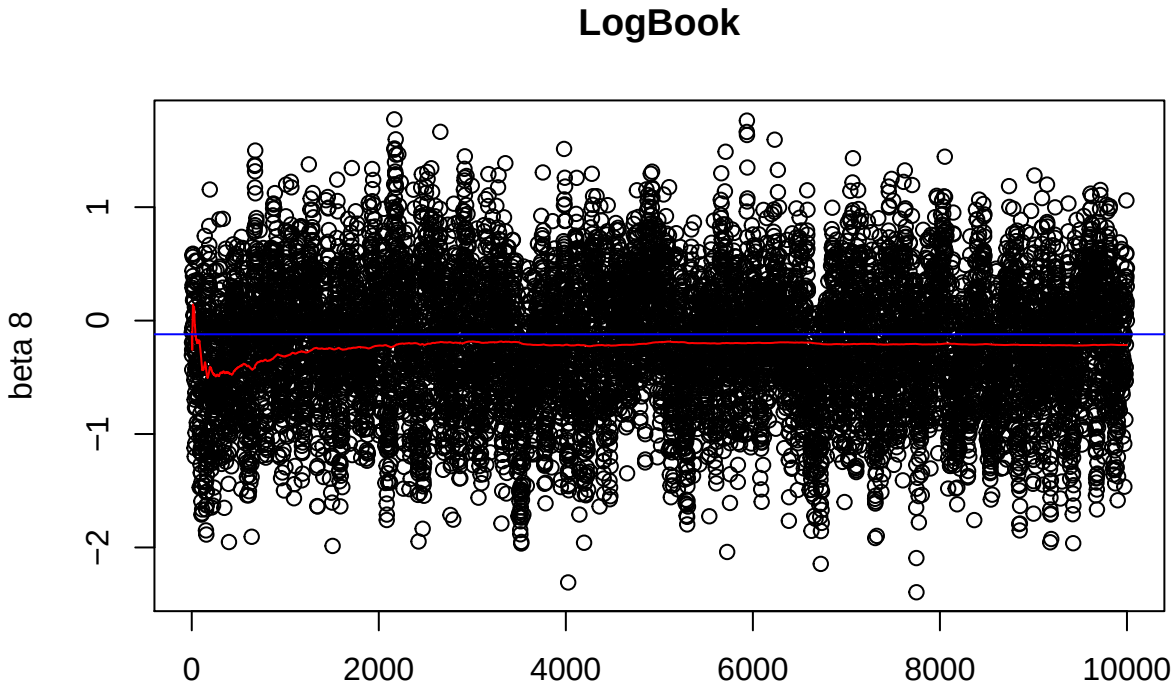

VerifyID
beta 3
5
0
−5
−10
0
2000
4000
6000
8000
10000
MCMC iteration

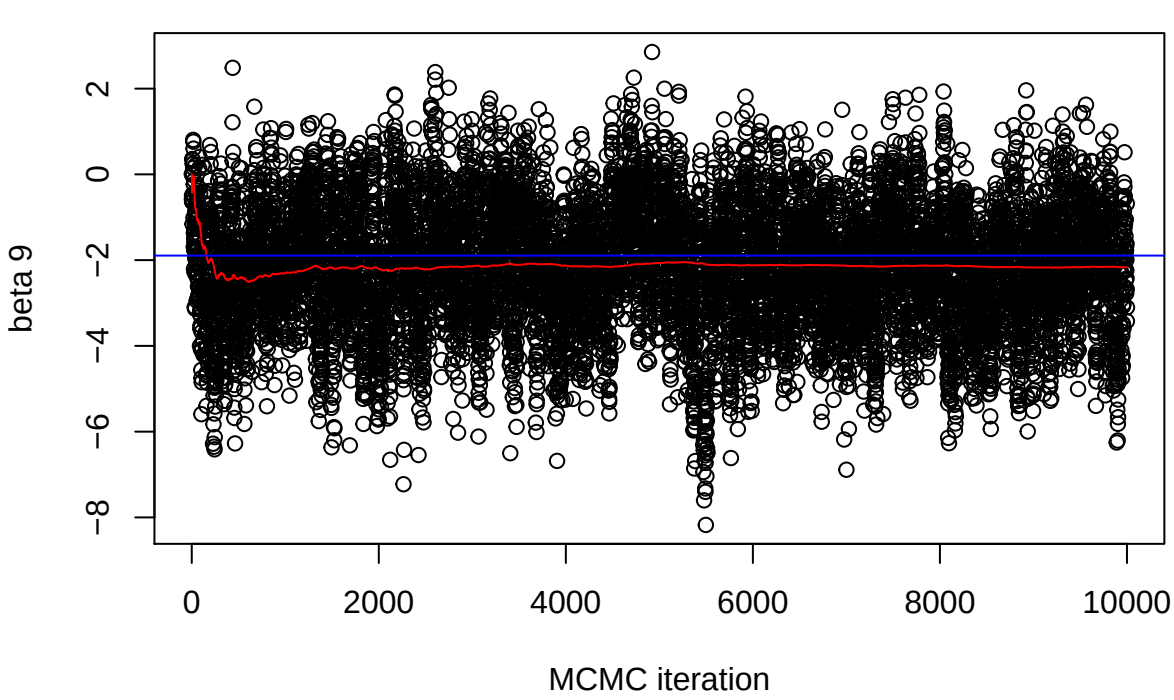

Sealed
beta 4
2
0
−2
−4
0
2000
4000
6000
8000
10000
MCMC iteration


Minblem
beta 5
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000
MCMC iteration


MajBlem
beta 6
5
0
−5
−10
0
2000
4000
6000
8000
10000
MCMC iteration


LargNeg
beta 7
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000
MCMC iteration

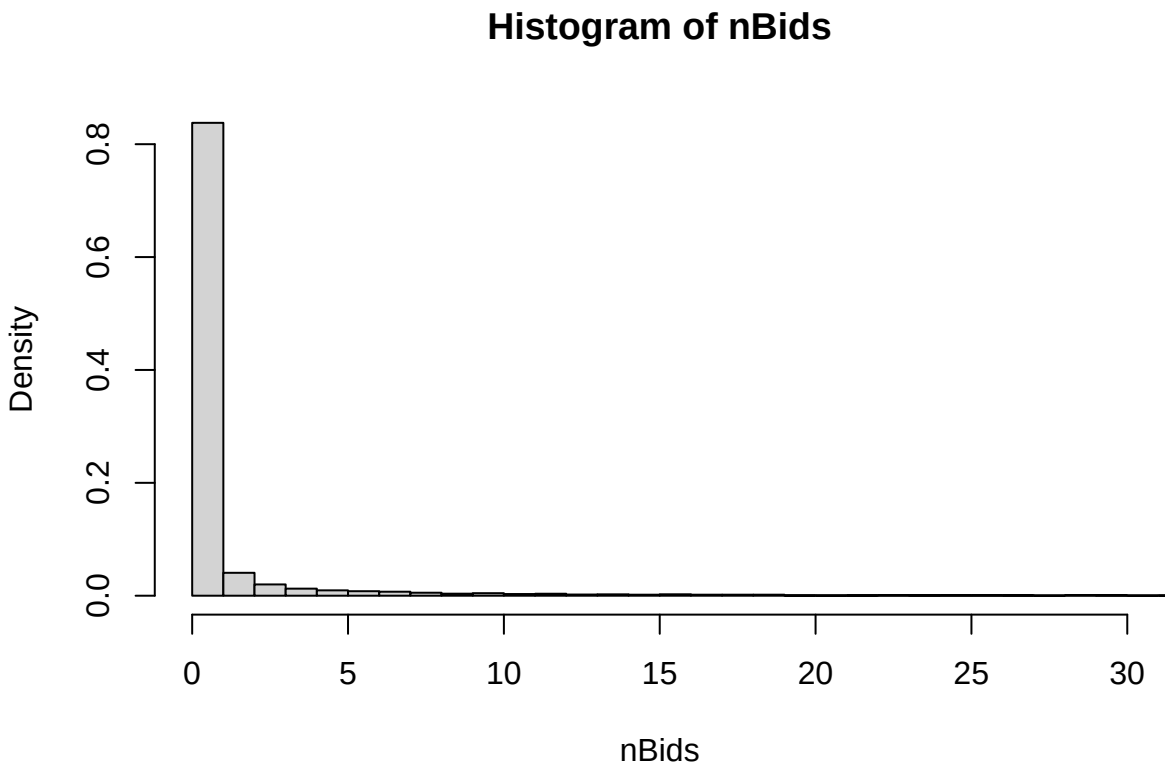




2.4 Question d)

```
auctionExample = c(1, 1, 1, 1, 0, 1, 0, 1, 0.7)

nBids = c()
```

```r
for(i in 1:nSamples){
   nBids[i] = rpois(1, exp(betaSamples[i, ] %*% auctionExample))
}

hist(x=nBids, xlim=c(0, 30), breaks=max(nBids), freq = FALSE)
```




```r
noBiddersProb = sum(nBids == 0)/(nSamples)

print(noBiddersProb)
```

```
## [1] 0.7463
```

3 Assignment 3.

3.0.1 Code for assignment 3a

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# initial settings
mu <- 20
sigmaSq <- 4
T <- 200
xt <- 20

AR1 <- function(mu, phi, prevXt, sigmaSquared){
  epsilon <- rnorm(1, mean=0, sd=sqrt(sigmaSquared))
  xt <- mu + phi*(prevXt-mu) + epsilon
  return(xt)
}
phiMatrix <- matrix(0, nrow = T, ncol=21)
phiMatrix[1,] <- 20

for (i in 2:T){
  col <-1
  for (phi in seq(-1, 1, 0.1)){

    AR <- AR1(mu, phi, xt, sigmaSq)
    xt <- AR
    phiMatrix[i, col] <- xt
    col <- col + 1
  }

}

plot(phiMatrix[,3], type="l", col="green", ylab="x(t)", xlab="t")
lines(phiMatrix[,7], col="black")
lines(phiMatrix[,11], col="blue")
lines(phiMatrix[,15], col="grey")
lines(phiMatrix[,19], col="red")
legend("topright",
       c("phi = -0.8", "phi = -0.4", "phi = 0", "phi = 0.4", "phi = 0.8"),
       fill=c("green","black", "blue", "grey", "red"),
       box.lwd = 2)

hist(phiMatrix[,3], xlab="x(t)", main="Histogram of x(t), phi=-0.8")
hist(phiMatrix[,7], xlab="x(t)", main="Histogram of x(t), phi=-0.4")
hist(phiMatrix[,11], xlab="x(t)", main="Histogram of x(t), phi=0")
hist(phiMatrix[,15], xlab="x(t)", main="Histogram of x(t), phi=0.4")
hist(phiMatrix[,19], xlab="x(t)", main="Histogram of x(t), phi=0.8")

# subass b)
phi1 <- 0.3
phi2 <- 0.9
```

```r
phivec1 <- vector()
phivec2 <- vector()
phivec1[1] <- 20
phivec2[1] <- 20

for (i in 2:T){

  phivec1[i] <- AR1(mu, phi1, phivec1[i-1], sigmaSq)

    phivec2[i] <- AR1(mu, phi2, phivec2[i-1], sigmaSq)
  }


library(rstan)
stanFunc <- "data {
  int<lower=0> T;
  vector[T] x;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  for (t in 2:T)
    x[t] ~ normal(mu + phi * (x[t-1]-mu), sigma);
}
"
my_data1 <- list(T=200, x=phivec1)
my_data2 <- list(T=200, x=phivec2)

fit_AR1 <- stan(model_code=stanFunc, data = my_data1)
fit_AR2 <- stan(model_code=stanFunc, data = my_data2)

extract1 = summary(fit_AR1, pars = c("mu","phi","sigma"),
                   probs = c(0.025, 0.975))$summary

extract2 = summary(fit_AR2, pars = c("mu","phi","sigma"),
                    probs = c(0.025, 0.975))$summary


# Given dependent samples, the number of independent samples is replaced with the effective sample size
# N_eff, which is the number of independent samples with the same estimation power as the
# N autocorrelated samples.

mu_phi_sigma_quants1 <- extract1[, c("mean", "2.5%", "97.5%", "n_eff")]
mu_phi_sigma_quants2 <- extract2[, c("mean", "2.5%", "97.5%", "n_eff")]

print(mu_phi_sigma_quants1)
print(mu_phi_sigma_quants2)

mean(phivec1) # from original model
mean(phivec2) # from original model
```

3.1 Question 3a)

The value of ϕ affects the time series by making it more or less stable. For lower values of ϕ e.g ϕ=0, the AR 1 process oscillates less since the only thing affecting the next time step is the initial value of μ aswell as ϵ. If the value of ϕ is set to 1 or above or to -1 and below, the AR1-process destabilizes.

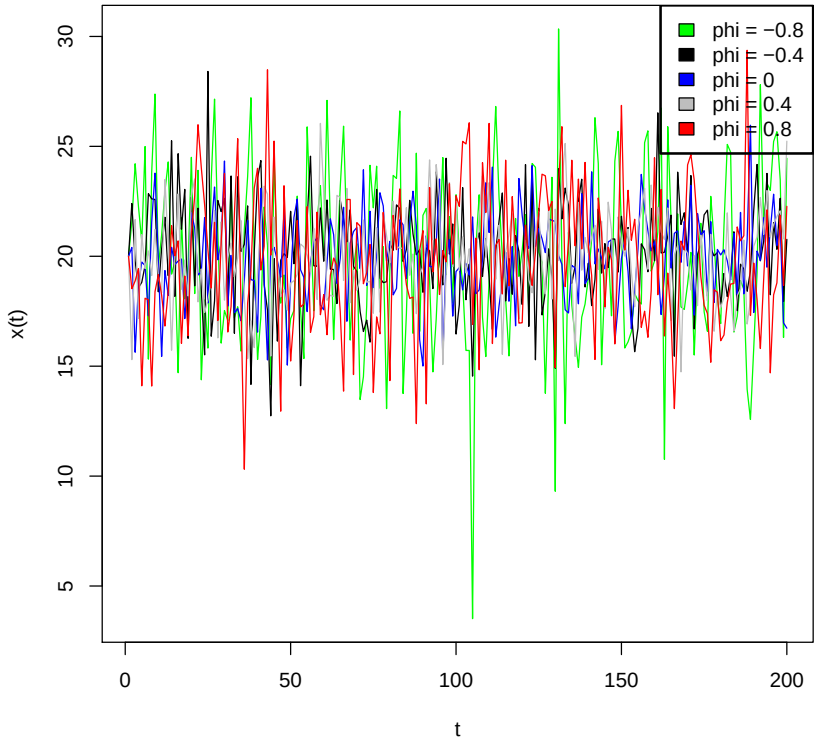


Figure 3: Realizations of x1:T for phi=-0.8, -0.4, 0, 0.4, 0.8

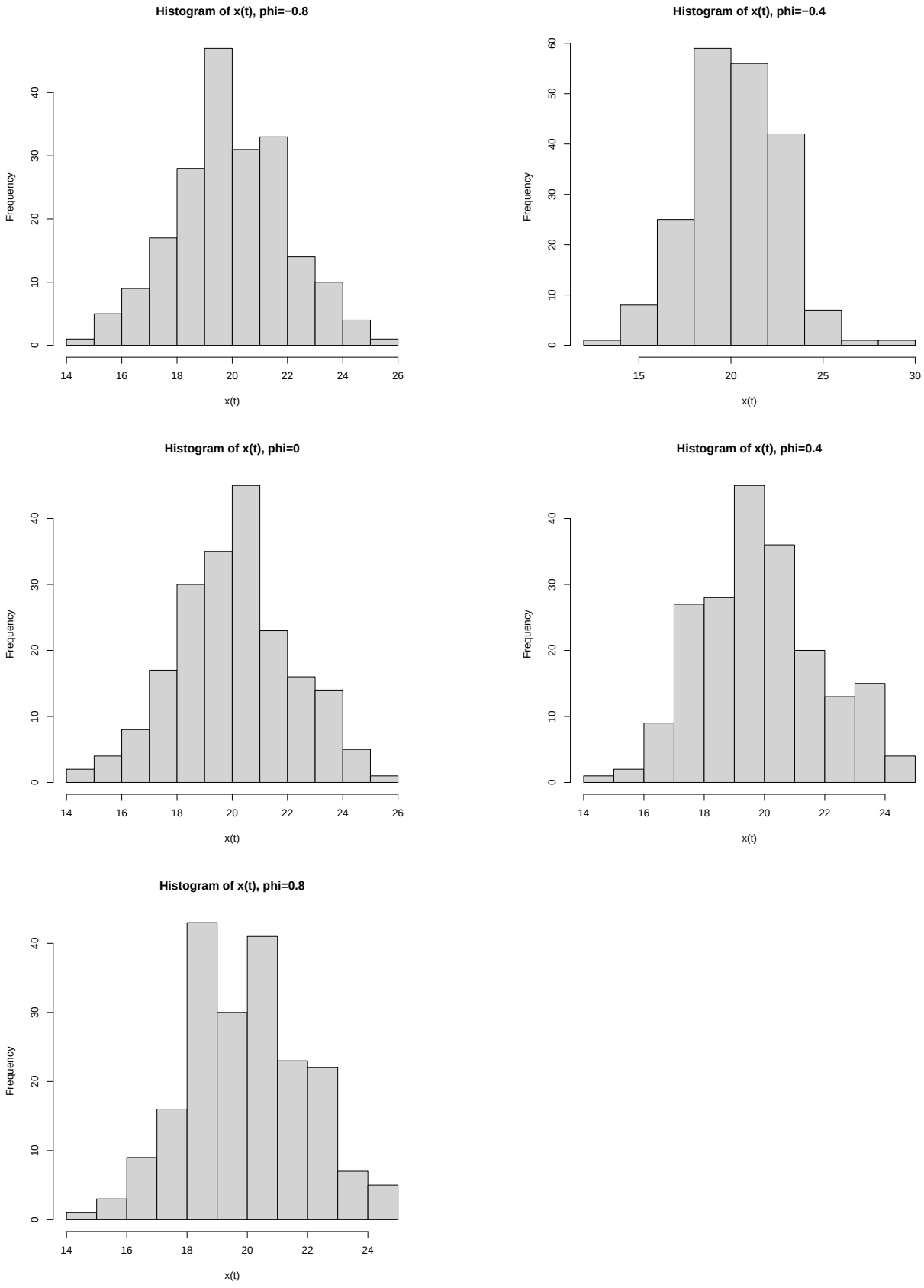


Figure 4: Histograms for phi = -0.8, -0.4, 0, 0.4 and 0.8

Given dependent samples, the number of independent samples is replaced with the effective sample size N_eff. Since our samples are dependent, we use the effective sample size N_eff.

For $\phi = 0.3$ we get the following mean, 95% credible intervals and effective posterior samples of μ, ϕ and σ:

```
print(mu_phi_sigma_quants1)
```

```
##             mean        2.5%      97.5%    n_eff
## mu    19.9801344 19.59300388 20.3590211 3391.128
## phi    0.2230197  0.08497081  0.3631718 3688.917
## sigma  2.1168585  1.92010582  2.3401669 3814.877
```

For $\phi = 0.9$ we get the following mean, 95% credible intervals and effective posterior samples of μ, ϕ and σ:

```
print(mu_phi_sigma_quants2)
```

```
##             mean      2.5%     97.5%     n_eff
## mu    17.2284996 8.2678048 22.606757  587.4006
## phi    0.9177426 0.8503291  0.988663 1354.7987
## sigma  2.2159346 2.0064075  2.443740 2955.4575
```

```
print(mean(phivec1)) # from original model
```

```
## [1] 19.98934
```

```
print(mean(phivec2)) # from original model
```

```
## [1] 17.75282
```

When ϕ is 0.3 it is possible to estimate the true values of the parameters in a good way since the mean obtained is close to the real mean, and the estimated values of ϕ and σ are also close to the real values. The 95% credible interval for μ is very narrow which shows that we can estimate it in a good way. Furthermore the effective sample size is larger for $\phi = 0.3$ than for $\phi = 0.9$ which shows us that the standard deviation of the estimated parameters are lower.

When ϕ is 0.9 it is harder to estimate the true values which can be seen since the mean is divergent from the true mean. The credible intervals are also wider for μ.

```
## Convergence
draws1 = extract(fit_AR1)
draws2 = extract(fit_AR2)

plot(draws1$mu)
plot(draws1$phi)
plot(draws1$sigma)
plot(draws1$mu, draws1$phi)

plot(draws2$mu)
plot(draws2$phi)
plot(draws2$sigma)
plot(draws2$mu, draws2$phi)
```

3.2 3b)

Values for μ also diverges very much when $\phi = 0.9$ since the phi values in the stan process will vary a bit and go above 1. This destabalizes the AR1 process and gives values for μ that is way off. This can be seen in the plot below for the joint posterior of μ and ϕ when $\phi = 0.9$.

The plots tells us the same story as before where the values obtained when $\phi = 0.9$ diverges more.

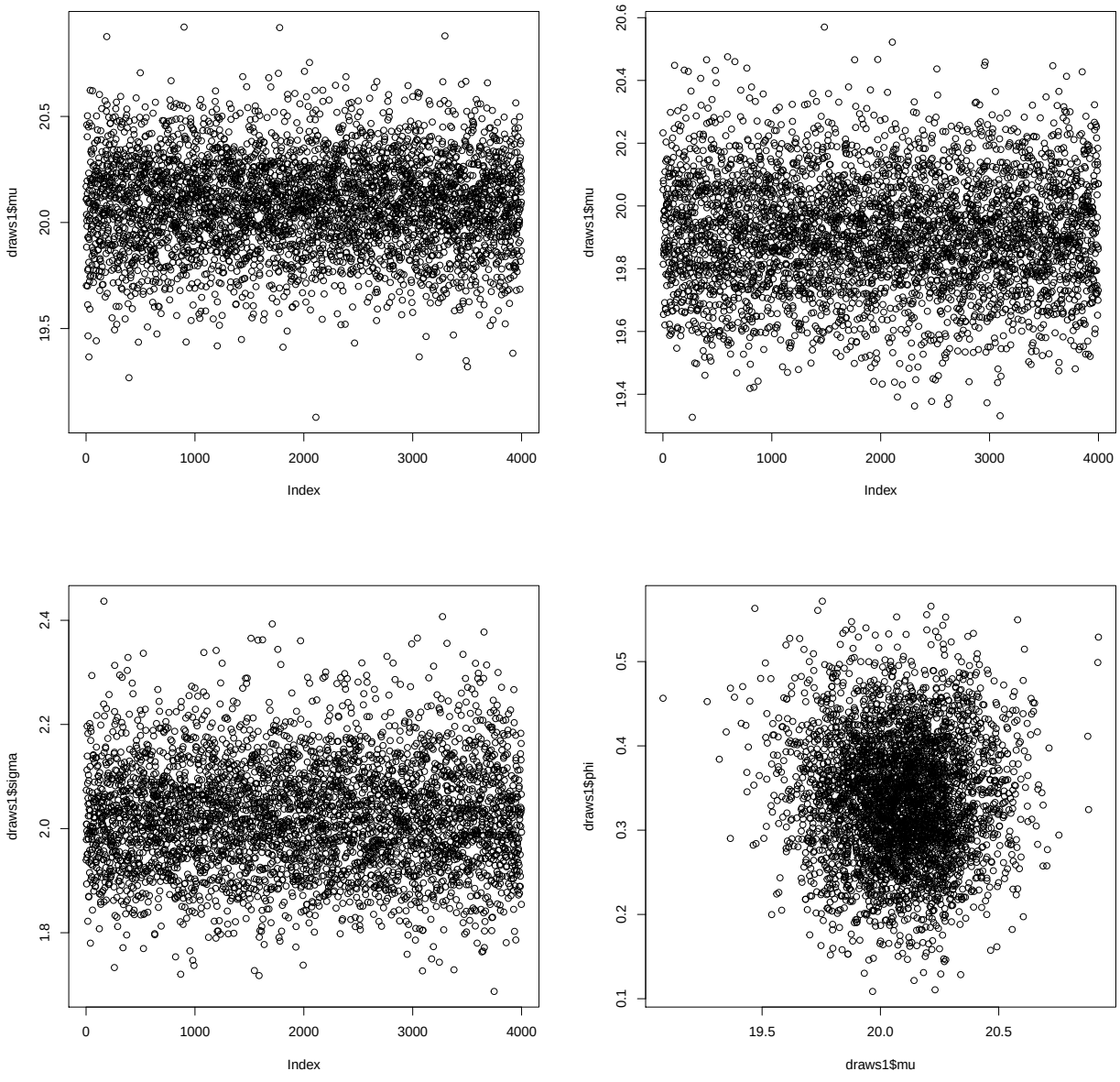


Figure 5: Marginal posteriors for mu and sigma

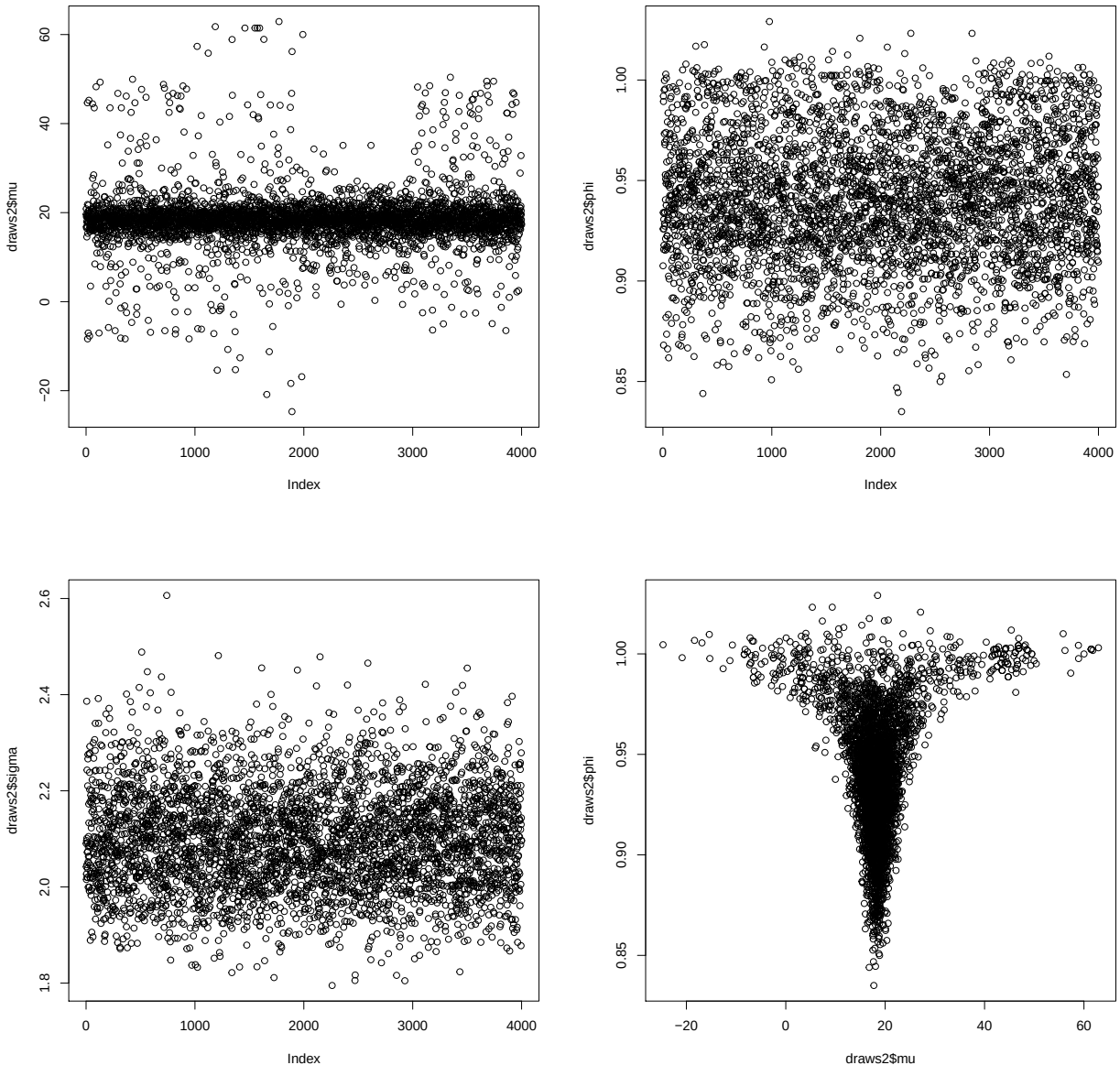


Figure 6: Marginal posteriors for mu and sigma