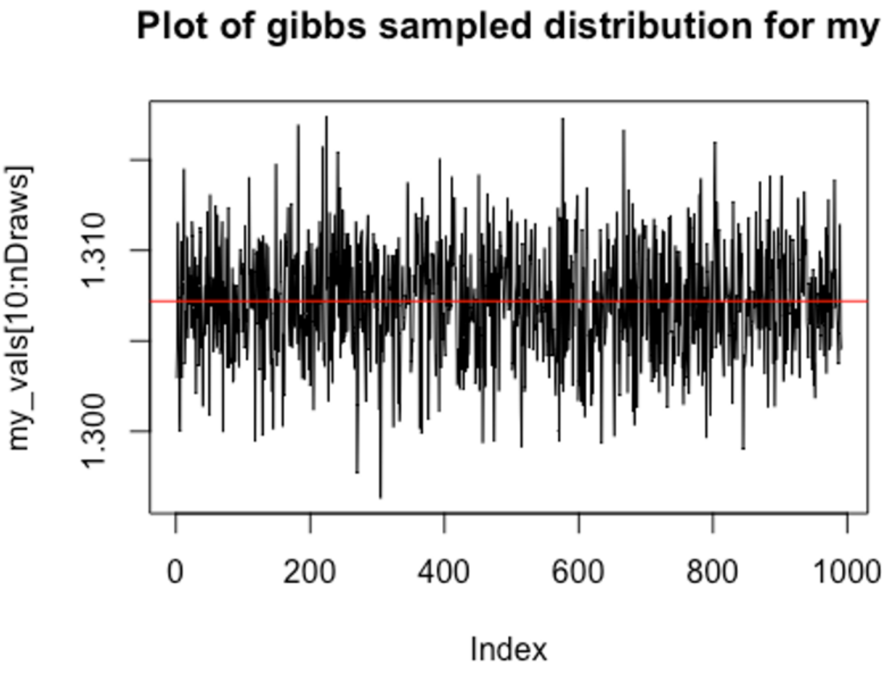
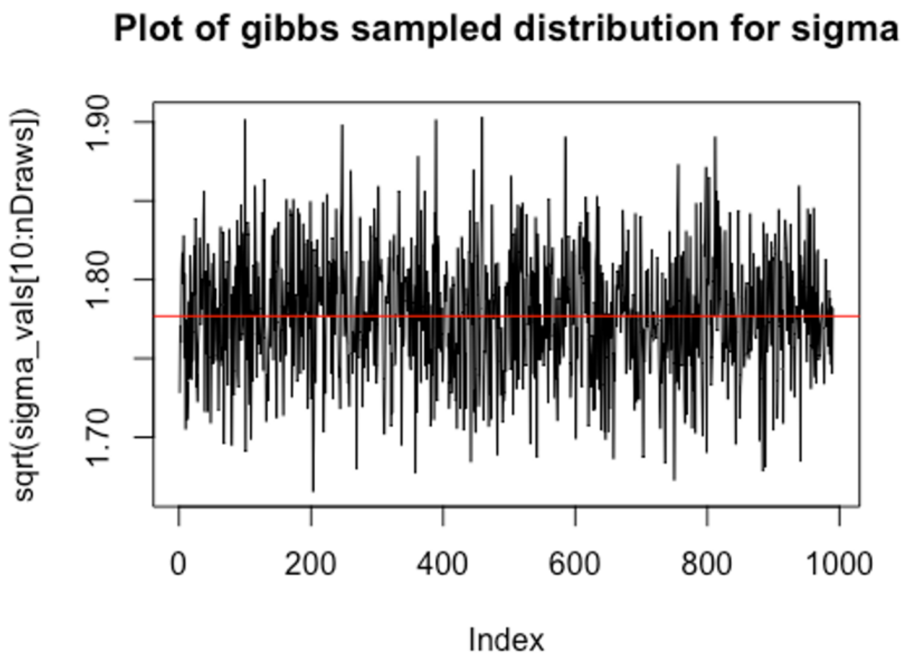
1. a)





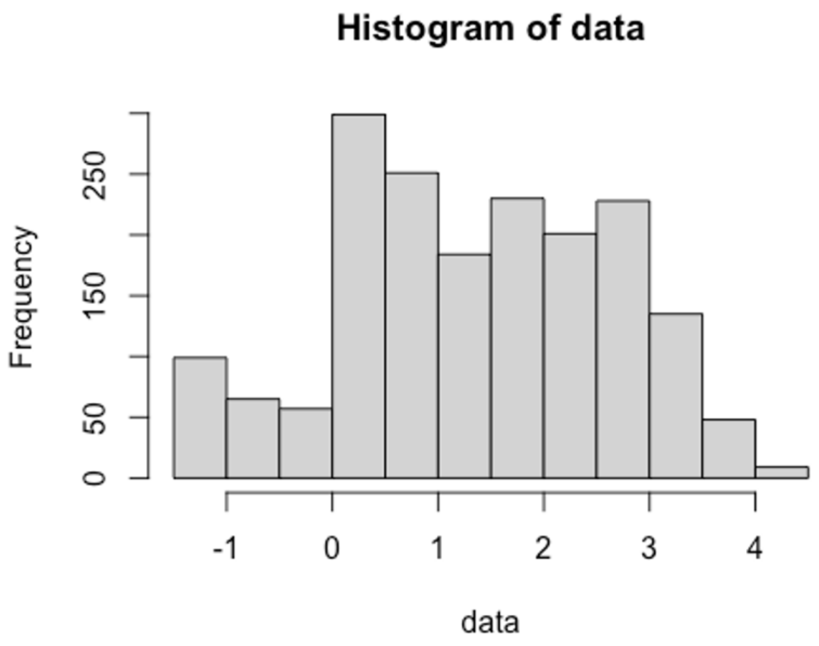
Inefficiency factor my: 0.52
Inefficiency factor sigma: 1.05

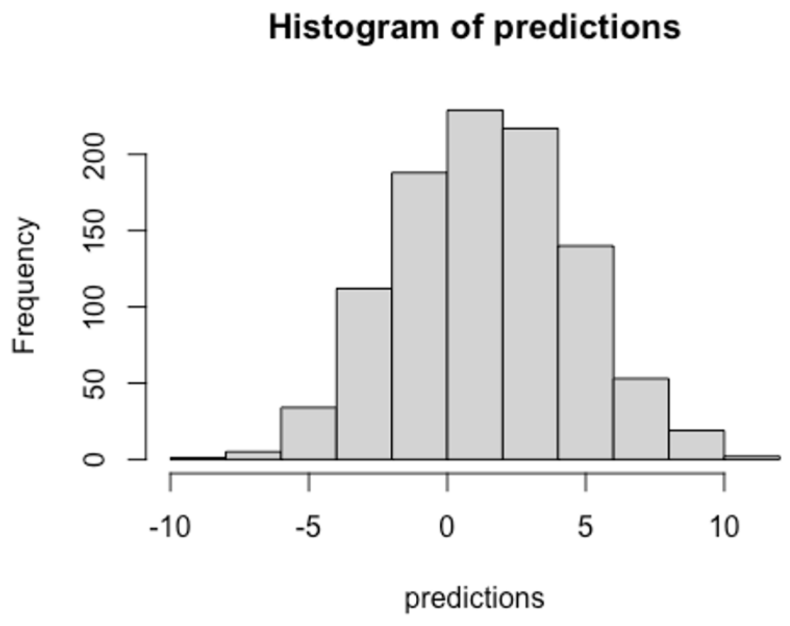
Conclusion: As the plots suggest, we can't distinguish any convergence for the mean and standard deviation as we clearly can see that the values oscillate around the means (red line). Moreover, the inefficiency factors shows how correlated the draws are

Code:

```
data=readRDS("Precipitation.rds") #Reading data
data = log(data)

#a)

# Initial setting (prior settings)
my0 = 1
tao0=1
v0=1
sigma0sq=1
nDraws = 1000
n = nDraws
true_mean = mean(data)
vn = v0 + n

#Lists for storing draws
my_vals = rep(0,nDraws)
sigma_vals = rep(0,nDraws)
sigma_vals[1] = 1 #Initial sigmasq value to be used

#Drawing posteriors
for(i in 1:nDraws) {

 w = (n/sigma_vals[i])/((n/sigma_vals[i])+(1/tao0**2)) #Calculating weights
 myn = w*true_mean+(1-w)*my0 #Calculating myn
 taonsq = 1/((n/sigma_vals[i])+(1/tao0)) #Calculating taonsq
 my_vals[i] = rnorm(1,mean=myn, taonsq) #Calculating posterior my_val

 if (i <= (nDraws-1)) {
 chi_val = rchisq(1,vn) #Drawing random value from chi distribution
 sigmansq = (v0*sigma0sq+sum((data-my_vals[i])**2))/(n+v0) #Calculating approximation
sigmasq
 sigma_vals[i+1] = vn*sigmansq/chi_val #Calculating posterior sigma_val
 }
}

#Plotting my convergence
plot(my_vals[10:nDraws], ,main = "Plot of gibbs sampled distribution for my",type = "l")
abline(h=mean(my_vals), col="red") #Expected value

#Plotting sigma convergence
plot(sqrt(sigma_vals[10:nDraws]), ,main = "Plot of gibbs sampled distribution for sigma",type
= "l")
abline(h=mean(sqrt(sigma_vals)), col="red") #Expected value

#Calculating inefficienty factor (IF)


#Autocorrelation for my
a_my = acf(my_vals)

inefficiency_factor_my = 1+2*sum(a_my$acf[-1])
inefficiency_factor_my

## [1] 0.5193648

#Autocorrelation for my
a_sigma = acf(sqrt(sigma_vals))

inefficiency_factor_sigma = 1+2*sum(a_sigma$acf[-1])
inefficiency_factor_sigma

## [1] 1.04877

#Conclusion: As the plots suggest, we can't distinguish any convergence for the mean and standard deviation
#as we clearly can see that the values oscillate around the means (red line)
#Moreover, the inefficiency factors shows how correlated the draws are, as we can see, the results are
#relatively close to 1 which indicates low correlation
```

b)



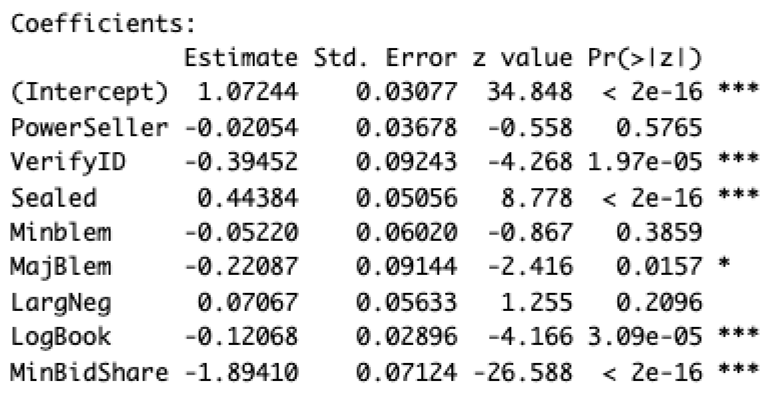


Conclusion: As we can see, the draws from the predictions differ slightly from the true distribution. The main difference is that the normal distribution have a bit longer tails on each end, however the distributions seem to be centred around the same values in both plots

Code:
#b)
#1) Histogram or kernel density estimate of the daily preception y1,...,yn

```r
hist(data) #Histogram with true data
#2) The resulting posterior predictive density p(ỹ|y1,...,yn) using the simulated posterior
draws from (a)
predictions = rep(0,nDraws)
for (i in 1:nDraws) {
predictions[i] = rnorm(1,my_vals[i],sigma_vals[i])
}

hist(predictions)

#Conclusion: As we can see, the draws from the predictions differ slightly from the true
distribution.
#The main difference is that the normal distribution have a bit longer tails on each end,
however
#the distributions seem to be centered around the same values in both plots
```

2. a)



```
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 ***
```

the z-value shows that VerifyID, Sealed, MajBlem, LogBook and MinBidShare are significant

Code:

```
# (a) 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?

library(mvtnorm)

ebay_data <-  (read.table("eBayNumberOfBidderData.dat", header=TRUE))
Xnames <- names(ebay_data[,2:ncol(ebay_data)]) #Names of columns
n_obs <- dim(ebay_data)[1]  #number of observations
data <- ebay_data[,-2] #training and test data without Const
#y <- as.numeric(data[,1]) #target output

# maximum likelihood estimator of beta
model = glm(formula=nBids~., data=data, family=poisson())
summary(model)

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

#the z-value shows that VerifyID, Sealed, MajBlem, LogBook and MinBidShare are significant

b)

The posterior mode:

```
 [1,]  1.06984118 "Const"
 [2,] -0.02051246 "PowerSeller"
 [3,] -0.39300599 "VerifyID"
 [4,]  0.44355549 "Sealed"
 [5,] -0.05246627 "Minblem"
 [6,] -0.22123840 "MajBlem"
 [7,]  0.07069683 "LargNeg"
 [8,] -0.12021767 "LogBook"
 [9,] -1.89198501 "MinBidShare"
```

Posterior standard deviation:

```
 Const        PowerSeller  VerifyID   Sealed       Minblem    MajBlem     LargNeg
 0.03074837  0.03678418  0.09227871  0.05057448  0.06020470  0.09146070  0.05634767
 LogBook     MinBidShare
0.02895635  0.07109682
```

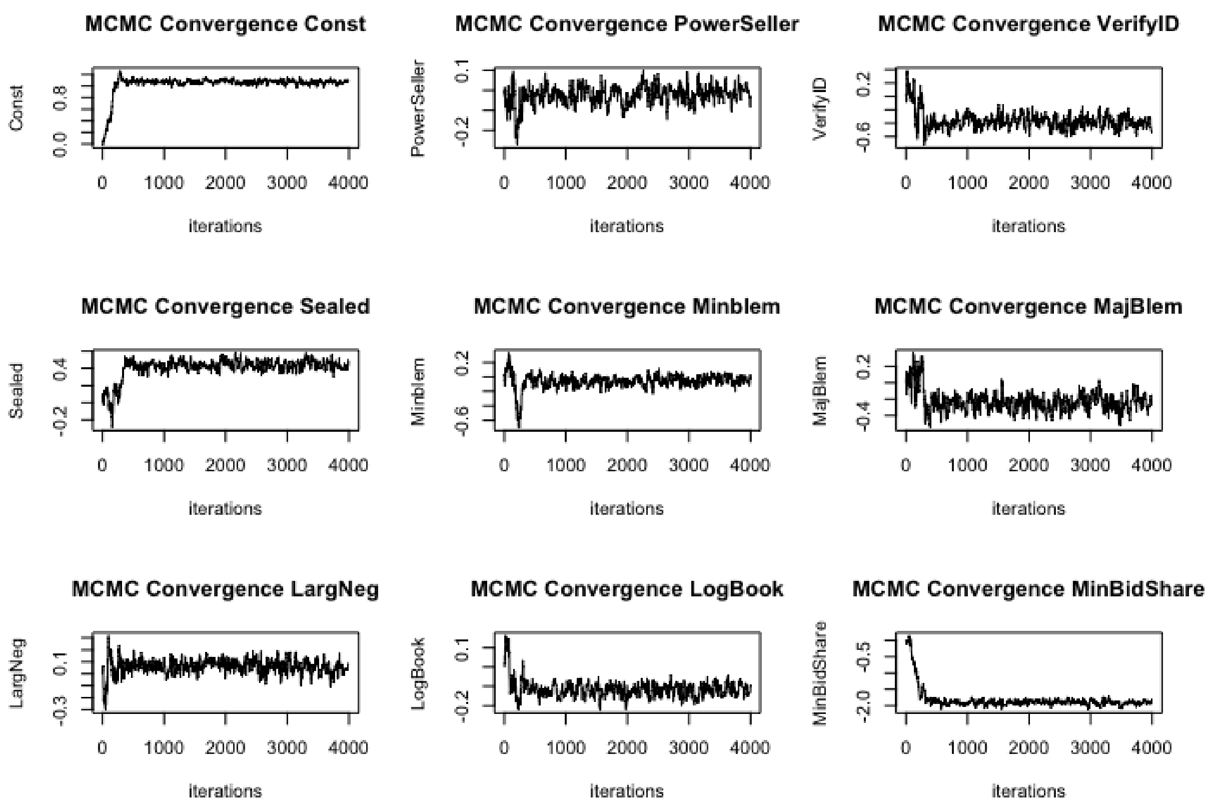
the result seems reasonable since it is similar to a)

Code:

```r
#  (b) Let's do a Bayesian analysis of the Poisson regression. Let the prior be beta ~ N[0, 
100*(X_T%*%X)]
# where X is the n × 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 ~ N(beta_tilde, Jy^-1(beta_tilde))
#where β_tilde is the posterior mode and Jy(beta_tilde) is the negative Hessian at the 
posterior mode.
#beta_tilde and Jy(beta_tilde) 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.).

#function for Poisson logPosterior
logPostPoisson = function(beta,y,X,mu,Sigma) {
  beta <- matrix(beta, nrow=1)
  linPred <- X%*%t(beta)
  logLik <- sum( - exp(linPred) + linPred*y - log(factorial(y)) ) #loglikelihood
  logPrior <- dmvnorm(beta, mu, Sigma, log=TRUE); #prior
  return(logLik + logPrior) #prior and likelihood are added since we use log
}

#initialize variables
y <- as.numeric(ebay_data[,1]) #target output
X <- as.matrix(ebay_data[,-1]) #training data
X <- apply(X, c(1, 2), as.numeric) #convert from characters to doubles
n_col <- dim(X)[2] #number of columns in training data
initVal <- rep(0,n_col) #initial values for Beta
mu <- as.matrix(rep(0,n_col)) # Prior mean vector for Beta
Sigma <- 100*solve(t(X)%*%X) # Prior covariance matrix for Beta
logPost <- logPostPoisson; #function to be minimized


set.seed(123)
OptimRes <- 
optim(initVal,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS"),control=list(fnscale=-1),hes
sian=TRUE)

# Printing the results of mean and variance of normally distributed Beta
OptimBeta <- OptimRes$par
names(OptimBeta) <- Xnames # Naming the coefficient by covariates
PostCov <- solve(-OptimRes$hessian) #get covariance matrix by inverting the negative 
hessian
approxPostStd <- sqrt(diag(PostCov)) # Computing approximate standard deviations.
names(approxPostStd) <- Xnames # Naming the coefficient by covariates
print('Posterior beta values:')
```

```
## [1] "Posterior beta values:"

print(OptimBeta)

##       Const PowerSeller    VerifyID      Sealed     Minblem      MajBlem
##  1.06984118 -0.02051246 -0.39300599  0.44355549 -0.05246627 -0.22123840
##     LargNeg     LogBook MinBidShare
##  0.07069683 -0.12021767 -1.89198501

print('Posterior standard deviation:')

## [1] "Posterior standard deviation:"

print(approxPostStd)

##       Const PowerSeller    VerifyID      Sealed     Minblem      MajBlem
##  0.03074837  0.03678418  0.09227871  0.05057448  0.06020470  0.09146070
##     LargNeg     LogBook MinBidShare
##  0.05634767  0.02895635  0.07109682

#the result seems reasonable since they are similar to a)
```

c)



Betavalues are converging as we can see in the plot.

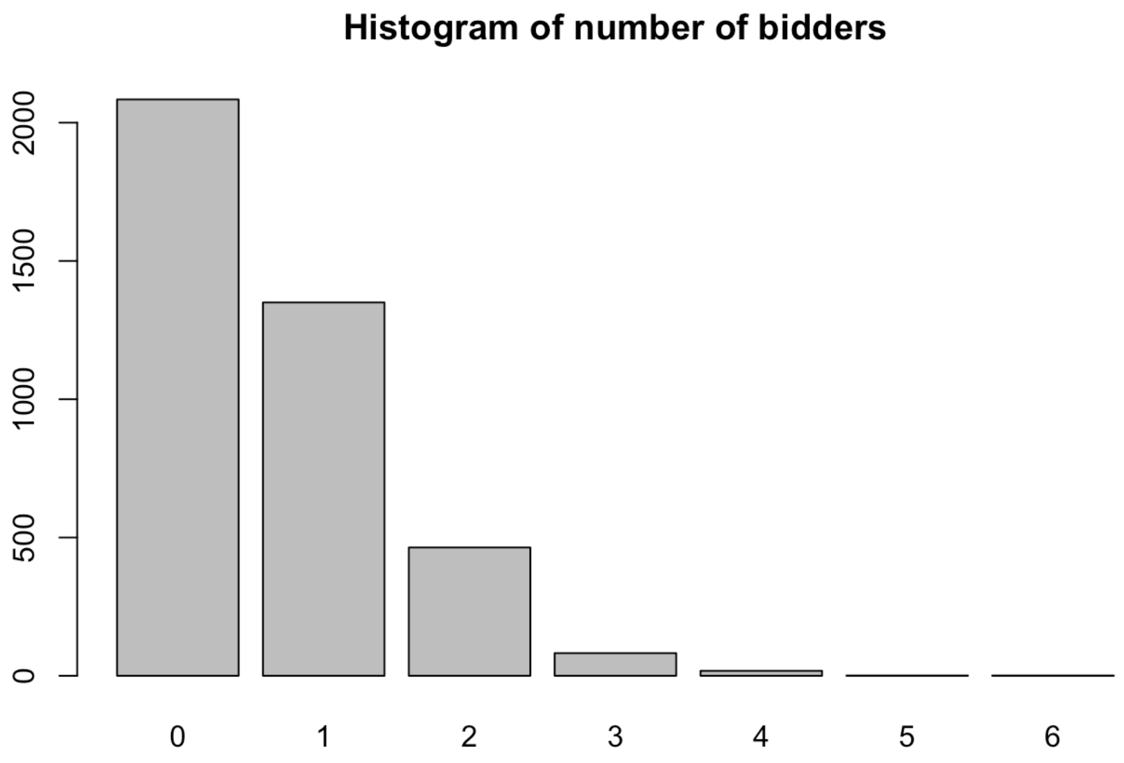
average_alpha = 0.2531, so we have average acceptance probability between 25-30% which means c=0.7 was appropriate.

Code:

```
# (c) Let's simulate from the actual posterior of beta using the Metropolis algorithm
# and compare the results 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, we denote the vector of model parameters by θ. Let the proposal density
# be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):
# θp|θ(i−1) ~ N(θ(i−1), c · Σ) where Σ = Jy^−1(beta?tilde) was 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. The note HowToCodeRWM.pdf in Lisam describes how you can 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.

RWMSampler <- function(oldTheta, logPostFunc, c, PostCov, ... ) {
  proposedTheta <- rmvnorm(1, mean=oldTheta, sigma=c*PostCov)
  alpha <- min(1, exp(logPostFunc(proposedTheta, ...) - logPostFunc(oldTheta, ...) ))
  t <- runif(1)
  if (alpha>t) {
        return(list(proposedTheta, alpha))
  } else {
        return(list(oldTheta, alpha))
  }
}

nDraws <- 4000
c <- 0.7
beta <- matrix(0, nDraws, n_col) #matrix to fill with samples from RWM
alpha_vector <- rep(0, nDraws) #will be used to calculate mean alpha
logPost <- logPostPoisson


for(i in 1:nDraws) {
  output <- RWMSampler(beta[i,], logPost, c, PostCov, y, X, mu, Sigma)
  alpha_vector[i] <- output[[2]]
  if(i<nDraws) {
  beta[i+1,] <- output[[1]]
  }
```

```
}

iterations=seq(1,nDraws,1)
par(mfrow=c(3,3))
for (i in 1:n_col) {
  plot(iterations, beta[,i], type="l", main=paste("MCMC Convergence", Xnames[i]),
          ylab=Xnames[i])
}

par(mfrow=c(1,1), new=FALSE)

# Calculate average alpha
average_alpha <- mean(alpha_vector)
average_alpha #0.2531, has average acceptance probability between 25-30%

## [1] 0.2621183
```

d)



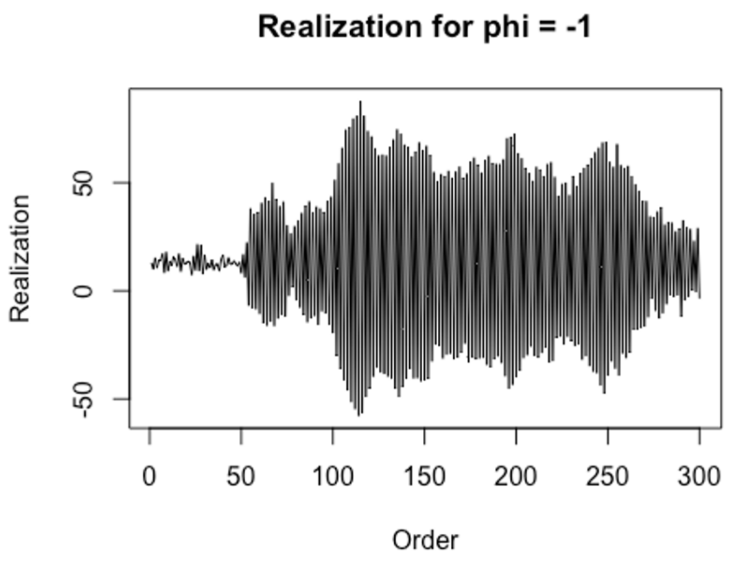
Calculations give 74% chance that we have no bidders with the given characteristics.

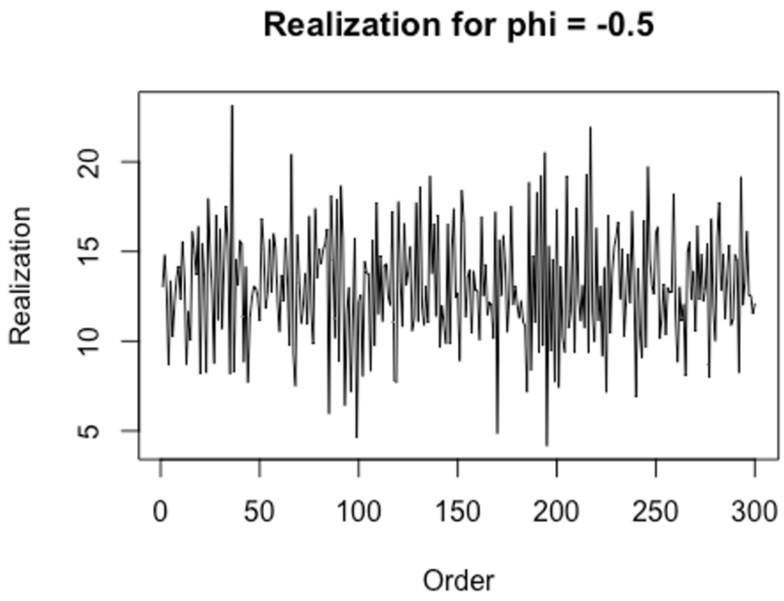
Code:

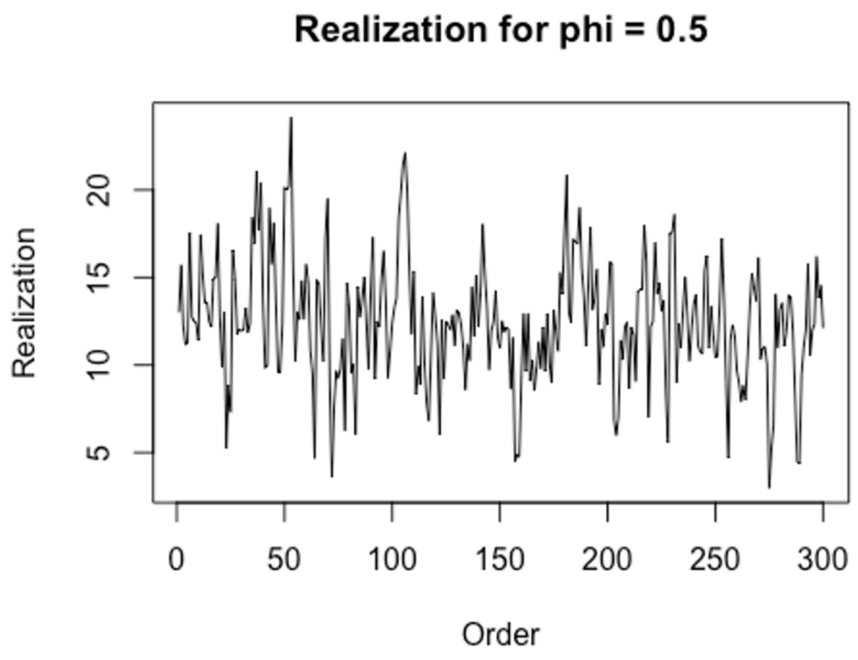
```
# (d) Use the MCMC draws from c) to simulate from the predictive distribution of
# the number of bidders in a new auction with the characteristics below.
#Plot the predictive distribution. What is the probability of no bidders in this new auction?
# PowerSeller = 1
```

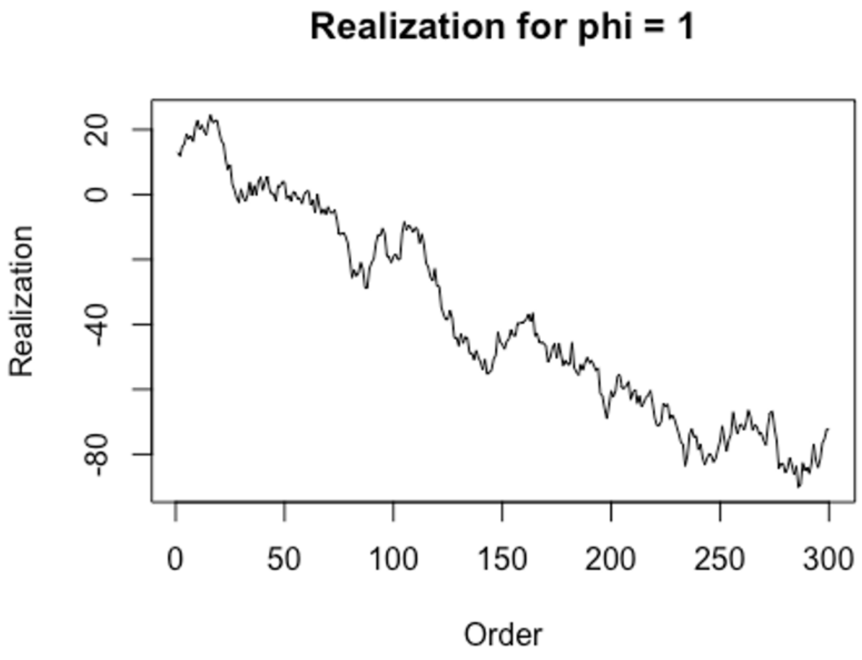
```r
# VerifyID = 0
# Sealed = 1
# MinBlem = 0
# MajBlem = 1
# LargNeg = 0
# LogBook = 1.2
# MinBidShare = 0.8

nDraws <- 10000
x <- c(1,1,0,1,0,1,0,1.2,0.8)
post_beta <- beta[(1000:nrow(beta)),] #select samples after convergence
mean <- exp(post_beta%*%x) #vector of means
samples <- rpois(nDraws, mean)
barplot(table(samples), main="Histogram of number of bidders")

prob_no_bidders <- sum(samples==0)/sum(samples) #probability of no bidders
print(prob_no_bidders)

## [1] 0.739524

#the probability of no bidders is 74%
```

3. a)


Realization for phi = -1
Realization
50
0
-50
0
50
100
150
200
250
300
Order


Realization for phi = -0.5
Realization
20
15
10
5
0
50
100
150
200
250
300
Order





Conclusion: Increased phi values reduces oscillations. We can see clear oscillations for phi=-1 and that the output becomes more and more correlated with increased phi values

Code:

```
#a)
library(rstan)

## Loading required package: StanHeaders
## Loading required package: ggplot2

## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```r
# Initial values
my=13
sigmasq=3
T=300
x1 = my
phi = seq(-1,1,0.25)

#Autoregression function
ar_function = function(my,sigmasq,T,x1,phi){
  result = rep(0,T)
  result[1] = x1
  for(i in 2:T) {
  epsilon = rnorm(1,0,sigmasq)
  result[i] = my + phi*(result[i-1]-my) + epsilon
  }
  return(result)
}

#Calculating output
result = matrix(0,T,length(phi))
for(i in 1:length(phi)) {
  result[,i] = ar_function(my,sigmasq,T,x1,phi[i])
}

#Plotting result for phi=-1 and phi=1
plot(seq(1,300,1),result[,1], xlab="Order", ylab="Realization", main="Realization for phi =
-1",type="l")

plot(seq(1,300,1),result[,3], xlab="Order", ylab="Realization", main="Realization for phi =
-0.5",type="l")

plot(seq(1,300,1),result[,7], xlab="Order", ylab="Realization", main="Realization for phi =
0.5",type="l")

plot(seq(1,300,1),result[,9], xlab="Order", ylab="Realization", main="Realization for phi =
1",type="l")

#Conclusion: Increased phi values reduces oscilations. We can see clear oscilations for
phi=-1
#and that the output becomes more and more correlated with increased phi values
```

b)

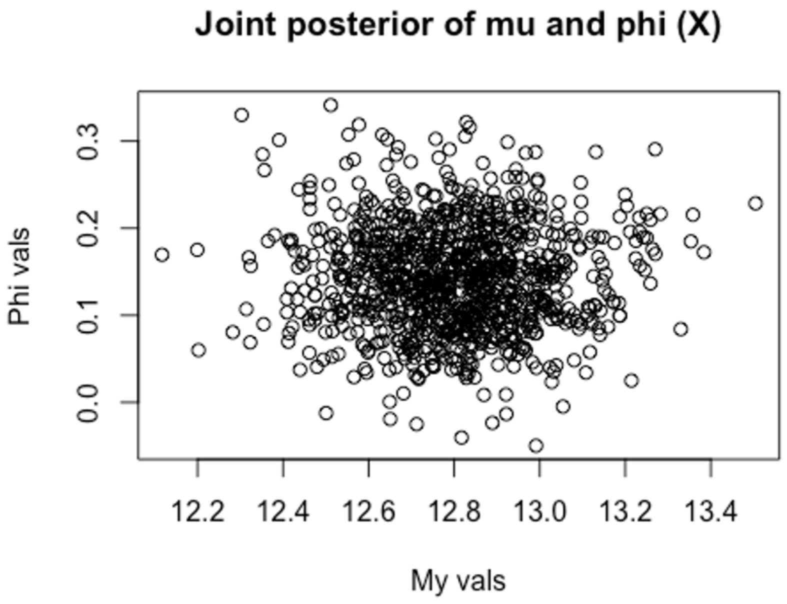
Table 1: Values for x model

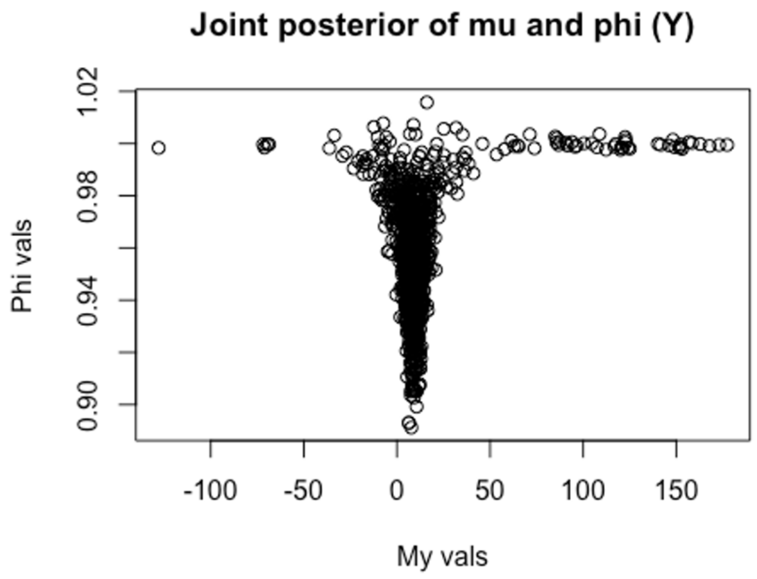
```
          mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
mu       12.84    0.00 0.20   12.43   12.71   12.84   12.97   13.24  3440    1
phi       0.12    0.00 0.06    0.01    0.08    0.12    0.16    0.24  3620    1
sigma2    9.02    0.01 0.76    7.66    8.49    8.99    9.51   10.61  3611    1
lp__   -474.33    0.03 1.30 -477.78 -474.90 -473.98 -473.39 -472.89  1970    1
```

Table 2: Values for y model

```
          mean se_mean    sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
mu       14.76    1.33 24.52  -40.56    9.38   14.82   19.58   76.75   342 1.01
phi       0.98    0.00  0.02    0.94    0.96    0.98    0.99    1.00   779 1.01
sigma2    8.64    0.02  0.72    7.37    8.14    8.58    9.11   10.20  1273 1.00
lp__   -467.64    0.05  1.38 -470.83 -468.53 -467.35 -466.51 -465.89   764 1.01
```

Conclusion: As the values in the credible interval are more narrow for the one where phi=0.2 were used, we get a better understanding of the true values for this one compared to the one where phi=0.95 were used. For this one, the values are really wide in the credible intervals





Conclusion: We can distinguish a convergence for the first sampler. However it is more difficult to distinguish a convergence for the second sampler as shown in the figure. This follows the previous reasoning, that the second sampler have a wider span for its values compared to the first sampler

Code:
```
#b)
#Drawing x1:T values using phi=0.2
x=rep(0,T)
x=ar_function(my,sigmasq,T,x1,0.2)

#Drawing y1:T values using phi=0.95
y=rep(0,T)
y=ar_function(my,sigmasq,T,x1,0.95)

#Implementing Stan model

StanModel = '
data {
  int<lower=0> N;
  vector[N] z;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma2;
}
model {
  for(i in 2:N){
    z[i] ~ normal(mu+phi*(z[i-1]-mu),sqrt(sigma2));
  }
}'

#For X
data <- list(N=T, z=x)
fit_x <- stan(model_code=StanModel,data=data)

#For Y
data <- list(N=T, z=y)
fit_y <- stan(model_code=StanModel,data=data)

#Extract posterior draws
post_x = extract(fit_x)
post_y = extract(fit_y)


#i)
#AR-process for X
#Calculating mean and credible interval for all parameters
#Can be achived with print(fit_x) too
mean_mu = mean(post_x$mu)
cred_int_mu = quantile(post_x$mu,probs=c(0.025,0.975))
print(mean_mu)

## [1] 12.79963

print(cred_int_mu)

##     2.5%    97.5%
## 12.43112 13.17352

mean_phi= mean(post_x$phi)
cred_int_phi = quantile(post_x$phi,probs=c(0.025,0.975))
print(mean_phi)

## [1] 0.1432679

print(cred_int_phi)

##       2.5%      97.5%
## 0.03507495 0.25712395

mean_sigma2 = mean(post_x$sigma2)
cred_int_sigma2 = quantile(post_x$sigma2,probs=c(0.025,0.975))
print(mean_sigma2)

## [1] 7.918592

print(cred_int_sigma2)
```

```
##     2.5%    97.5%
## 6.726024 9.330297

#AR-process for Y
#Calculating mean and credible interval for all parameters
mean_mu = mean(post_y$mu)
cred_int_mu = quantile(post_y$mu,probs=c(0.025,0.975))
print(mean_mu)

## [1] 10.04247

print(cred_int_mu)

##      2.5%     97.5%
## -16.08935  50.19349

mean_phi= mean(post_y$phi)
cred_int_phi = quantile(post_y$phi,probs=c(0.025,0.975))
print(mean_phi)

## [1] 0.957765

print(cred_int_phi)

##      2.5%     97.5%
## 0.9116279 1.0007403

mean_sigma2 = mean(post_y$sigma2)
cred_int_sigma2 = quantile(post_y$sigma2,probs=c(0.025,0.975))
print(mean_sigma2)

## [1] 9.138192

print(cred_int_sigma2)

##      2.5%     97.5%
##  7.744661 10.745055

#Plotting using print
print(fit_x)

## Inference for Stan model: d85eaf939f1fe064d5bdde49c5af5b23.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu       12.80    0.00 0.19   12.43   12.68   12.80   12.92   13.17  3586    1
## phi       0.14    0.00 0.06    0.04    0.10    0.14    0.18    0.26  3594    1
## sigma2    7.92    0.01 0.66    6.73    7.46    7.88    8.33    9.33  4058    1
## lp__   -455.35    0.03 1.22 -458.63 -455.90 -455.02 -454.47 -453.96  2033    1
##
## Samples were drawn using NUTS(diag_e) at Tue May  9 10:30:51 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).

print(fit_y)

## Inference for Stan model: d85eaf939f1fe064d5bdde49c5af5b23.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu        10.04    1.21 17.77  -16.09    5.62    8.79   11.80   50.19   215 1.02
## phi        0.96    0.00  0.02    0.91    0.94    0.96    0.97    1.00   584 1.01
## sigma2     9.14    0.02  0.76    7.74    8.61    9.07    9.64   10.75  1608 1.00
## lp__    -476.53    0.06  1.48 -480.00 -477.38 -476.14 -475.37 -474.75   614 1.01
##
## Samples were drawn using NUTS(diag_e) at Tue May  9 10:30:56 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).

#Conclusion: As the values in the credible interval are more narrow for the one where phi=0.2
#were used, we get a better understanding of the true values for this one compared to the one where
#phi=0.95 were used. For this one, the values are really wide in the credible intervals

#ii)
#Joint posterior for X
plot(post_x$mu[1000:2000],post_x$phi[1000:2000],xlab="My vals",ylab="Phi vals",
main="Joint posterior of mu and phi (X)")

#Joint posterior for X
plot(post_y$mu[1000:2000],post_y$phi[1000:2000],xlab="My vals",ylab="Phi vals",
main="Joint posterior of mu and phi (Y)")

#Conclusion: We can distinguish a convergence for the first sampler. However it is more difficult to distinguish a
#convergence for the second sampler as shown in the figure. This follows the previous reasoning, that the
#second sampler have a wider span for its values compared to the first sampler
```

Appendix

Exercise 1

```
data=readRDS("Precipitation.rds") #Reading data
data = log(data)

#a)

# Initial setting (prior settings)
my0 = 1
tao0=1
v0=1
sigma0sq=1
nDraws = 1000
n = nDraws
true_mean = mean(data)
vn = v0 + n

#Lists for storing draws
my_vals = rep(0,nDraws)
sigma_vals = rep(0,nDraws)
sigma_vals[1] = 1 #Initial sigmasq value to be used

#Drawing posteriors
for(i in 1:nDraws) {

 w = (n/sigma_vals[i])/((n/sigma_vals[i])+(1/tao0**2)) #Calculating weights
 myn = w*true_mean+(1-w)*my0 #Calculating myn
 taonsq = 1/((n/sigma_vals[i])+(1/tao0)) #Calculating taonsq
 my_vals[i] = rnorm(1,mean=myn, taonsq) #Calculating posterior my_val

 if (i <= (nDraws-1)) {
 chi_val = rchisq(1,vn) #Drawing random value from chi distribution
 sigmansq = (v0*sigma0sq+sum((data-my_vals[i])**2))/(n+v0) #Calculating approximation
sigmasq
 sigma_vals[i+1] = vn*sigmansq/chi_val #Calculating posterior sigma_val
 }
}

#Plotting my convergence
plot(my_vals[10:nDraws], ,main = "Plot of gibbs sampled distribution for my",type = "l")
abline(h=mean(my_vals), col="red") #Expected value

#Plotting sigma convergence
plot(sqrt(sigma_vals[10:nDraws]), ,main = "Plot of gibbs sampled distribution for sigma",type
= "l")
abline(h=mean(sqrt(sigma_vals)), col="red") #Expected value
```

```
#Calculating inefficienty factor (IF)

#Autocorrelation for my
a_my = acf(my_vals)

inefficiency_factor_my = 1+2*sum(a_my$acf[-1])
inefficiency_factor_my

## [1] 0.5193648

#Autocorrelation for my
a_sigma = acf(sqrt(sigma_vals))

inefficiency_factor_sigma = 1+2*sum(a_sigma$acf[-1])
inefficiency_factor_sigma

## [1] 1.04877

#Conclusion: As the plots suggest, we can't distinguish any convergence for the mean and standard deviation
#as we clearly can see that the values oscillate around the means (red line)
#Moreover, the inefficiency factors shows how correlated the draws are, as we can see, the results are
#relatively close to 1 which indicates low correlation

#b)
#1) Histogram or kernel density estimate of the daily preception y1,...,yn

hist(data) #Histogram with true data

#2) The resulting posterior predictive density p(ỹ|y1,...,yn) using the simulated posterior draws from (a)
predictions = rep(0,nDraws)
for (i in 1:nDraws) {
predictions[i] = rnorm(1,my_vals[i],sigma_vals[i])
}

hist(predictions)

#Conclusion: As we can see, the draws from the predictions differ slightly from the true distribution.
#The main difference is that the normal distribution have a bit longer tails on each end, however
#the distributions seem to be centered around the same values in both plots
```

Exercise 2

```
# Metropolis Random Walk for Poisson regression.
#Consider the following Poisson regression model y_i|beta(iid) ~
Poisson[exp(x_i_transpose%*%β)], i=1,...,n
# where y_i is the count for the ith observation in the sample
# and xi is the p-dimensionalvector 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 (x):
# Const (for the intercept)
# PowerSeller (equal to 1 if the seller is selling large volumes on eBay)
# VerifyID (equal to 1 if the seller is a verified seller by eBay)
# Sealed (equal to 1 if the coin was sold in an unopened envelope)
# MinBlem (equal to 1 if the coin has a minor defect)
# MajBlem (equal to 1 if the coin has a major defect)
# LargNeg (equal to 1 if the seller received a lot of negative feedback from customers)
# LogBook (logarithm of the book value of the auctioned coin according to expert sellers.
Standardized)
# MinBidShare (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, don't forget that glm() adds its own intercept so don't input the covariate
Const].
# Which covariates are significant?

library(mvtnorm)

ebay_data <-  (read.table("eBayNumberOfBidderData.dat", header=TRUE))
Xnames <- names(ebay_data[,2:ncol(ebay_data)]) #Names of columns
n_obs <- dim(ebay_data)[1]  #number of observations
data <- ebay_data[,-2] #training and test data without Const
#y <- as.numeric(data[,1]) #target output

# maximum likelihood estimator of beta
model = glm(formula=nBids~., data=data, family=poisson())
summary(model)

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

#the z-value shows that VerifyID, Sealed, MajBlem, LogBook and MinBidShare are significant

#  (b) Let's do a Bayesian analysis of the Poisson regression. Let the prior be beta ~ N[0, 100*(X_T%*%X)]
# where X is the n × 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 ~ N(beta_tilde, Jy^-1(beta_tilde))
#where β_tilde is the posterior mode and Jy(beta_tilde) is the negative Hessian at the posterior mode.
#beta_tilde and Jy(beta_tilde) 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.).

#function for Poisson logPosterior
logPostPoisson = function(beta,y,X,mu,Sigma) {
  beta <- matrix(beta, nrow=1)
  linPred <- X%*%t(beta)
  logLik <- sum( - exp(linPred) + linPred*y - log(factorial(y)) ) #loglikelihood
  logPrior <- dmvnorm(beta, mu, Sigma, log=TRUE); #prior
  return(logLik + logPrior) #prior and likelihood are added since we use log
}
```

```
#initialize variables
y <- as.numeric(ebay_data[,1]) #target output
X <- as.matrix(ebay_data[,-1]) #training data
X <- apply(X, c(1, 2), as.numeric) #convert from characters to doubles
n_col <- dim(X)[2] #number of columns in training data
initVal <- rep(0,n_col) #initial values for Beta
mu <- as.matrix(rep(0,n_col)) # Prior mean vector for Beta
Sigma <- 100*solve(t(X)%*%X) # Prior covariance matrix for Beta
logPost <- logPostPoisson; #function to be minimized


set.seed(123)
OptimRes <-
optim(initVal,logPost,gr=NULL,y,X,mu,Sigma,method=c("BFGS"),control=list(fnscale=-1),hes
sian=TRUE)

# Printing the results of mean and variance of normally distributed Beta
OptimBeta <- OptimRes$par
names(OptimBeta) <- Xnames # Naming the coefficient by covariates
PostCov <- solve(-OptimRes$hessian) #get covariance matrix by inverting the negative
hessian
approxPostStd <- sqrt(diag(PostCov)) # Computing approximate standard deviations.
names(approxPostStd) <- Xnames # Naming the coefficient by covariates
print('Posterior beta values:')

## [1] "Posterior beta values:"

print(OptimBeta)

##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.06984118 -0.02051246 -0.39300599  0.44355549 -0.05246627 -0.22123840
##     LargNeg     LogBook MinBidShare
##  0.07069683 -0.12021767 -1.89198501

print('Posterior standard deviation:')

## [1] "Posterior standard deviation:"

print(approxPostStd)

##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03074837  0.03678418  0.09227871  0.05057448  0.06020470  0.09146070
##     LargNeg     LogBook MinBidShare
##  0.05634767  0.02895635  0.07109682

#the result seems reasonable since they are similar to a)

# (c) Let's simulate from the actual posterior of beta using the Metropolis algorithm
# and compare the results 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, we denote the vector of model parameters by θ. Let the proposal density
# be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):
# θp|θ(i−1) ~ N(θ(i−1), c · Σ) where Σ = Jy^−1(beta?tilde) was 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. The note HowToCodeRWM.pdf in Lisam describes how you can 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.

RWMSampler <- function(oldTheta, logPostFunc, c, PostCov, ... ) {
 proposedTheta <- rmvnorm(1, mean=oldTheta, sigma=c*PostCov)
 alpha <- min(1, exp(logPostFunc(proposedTheta, ...) - logPostFunc(oldTheta, ...) ))
 t <- runif(1)
 if (alpha>t) {
       return(list(proposedTheta, alpha))
 } else {
       return(list(oldTheta, alpha))
 }
}

nDraws <- 4000
c <- 0.7
beta <- matrix(0, nDraws, n_col) #matrix to fill with samples from RWM
alpha_vector <- rep(0, nDraws) #will be used to calculate mean alpha
logPost <- logPostPoisson


for(i in 1:nDraws) {
 output <- RWMSampler(beta[i,], logPost, c, PostCov, y, X, mu, Sigma)
 alpha_vector[i] <- output[[2]]
 if(i<nDraws) {
 beta[i+1,] <- output[[1]]
 }
}

iterations=seq(1,nDraws,1)
par(mfrow=c(3,3))
for (i in 1:n_col) {
 plot(iterations, beta[,i], type="l", main=paste("MCMC Convergence", Xnames[i]),
       ylab=Xnames[i])
}

par(mfrow=c(1,1), new=FALSE)

# Calculate average alpha
average_alpha <- mean(alpha_vector)
average_alpha #0.2531, has average acceptance probability between 25-30%

## [1] 0.2621183

# (d) Use the MCMC draws from c) to simulate from the predictive distribution of
# the number of bidders in a new auction with the characteristics below.
#Plot the predictive distribution. What is the probability of no bidders in this new auction?
# PowerSeller = 1
# VerifyID = 0
# Sealed = 1
# MinBlem = 0
# MajBlem = 1
# LargNeg = 0
# LogBook = 1.2
# MinBidShare = 0.8

nDraws <- 10000
x <- c(1,1,0,1,0,1,0,1.2,0.8)
post_beta <- beta[(1000:nrow(beta)),] #select samples after convergence
mean <- exp(post_beta%*%x) #vector of means
samples <- rpois(nDraws, mean)
barplot(table(samples), main="Histogram of number of bidders")

prob_no_bidders <- sum(samples==0)/sum(samples) #probability of no bidders
print(prob_no_bidders)

## [1] 0.739524

#the probability of no bidders is 74%
```

Exercise 3

```
#a)
library(rstan)

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

# Initial values
my=13
sigmasq=3
T=300
x1 = my
phi = seq(-1,1,0.25)

#Autoregression function
ar_function = function(my,sigmasq,T,x1,phi){
  result = rep(0,T)
  result[1] = x1
  for(i in 2:T) {
  epsilon = rnorm(1,0,sigmasq)
  result[i] = my + phi*(result[i-1]-my) + epsilon
  }
  return(result)
}

#Calculating output
result = matrix(0,T,length(phi))
for(i in 1:length(phi)) {
  result[,i] = ar_function(my,sigmasq,T,x1,phi[i])
}

#Plotting result for phi=-1 and phi=1
plot(seq(1,300,1),result[,1], xlab="Order", ylab="Realization", main="Realization for phi =
-1",type="l")

plot(seq(1,300,1),result[,3], xlab="Order", ylab="Realization", main="Realization for phi =
-0.5",type="l")

plot(seq(1,300,1),result[,7], xlab="Order", ylab="Realization", main="Realization for phi =
0.5",type="l")
```

```
plot(seq(1,300,1),result[,9], xlab="Order", ylab="Realization", main="Realization for phi =
1",type="l")

#Conclusion: Increased phi values reduces oscilations. We can see clear oscilations for
phi=-1
#and that the output becomes more and more correlated with increased phi values

#b)
#Drawing x1:T values using phi=0.2
x=rep(0,T)
x=ar_function(my,sigmasq,T,x1,0.2)

#Drawing y1:T values using phi=0.95
y=rep(0,T)
y=ar_function(my,sigmasq,T,x1,0.95)

#Implementing Stan model

StanModel = '
data {
  int<lower=0> N;
  vector[N] z;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma2;
}
model {
  for(i in 2:N){
    z[i] ~ normal(mu+phi*(z[i-1]-mu),sqrt(sigma2));
  }
}'

#For X
data <- list(N=T, z=x)
fit_x <- stan(model_code=StanModel,data=data)

#For Y
data <- list(N=T, z=y)
fit_y <- stan(model_code=StanModel,data=data)

#Extract posterior draws
post_x = extract(fit_x)
post_y = extract(fit_y)

#i)
#AR-process for X
#Calculating mean and credible interval for all parameters
```

```
#Can be achived with print(fit_x) too
mean_mu = mean(post_x$mu)
cred_int_mu = quantile(post_x$mu,probs=c(0.025,0.975))
print(mean_mu)

## [1] 12.79963

print(cred_int_mu)

##     2.5%    97.5%
## 12.43112 13.17352

mean_phi= mean(post_x$phi)
cred_int_phi = quantile(post_x$phi,probs=c(0.025,0.975))
print(mean_phi)

## [1] 0.1432679

print(cred_int_phi)

##       2.5%      97.5%
## 0.03507495 0.25712395

mean_sigma2 = mean(post_x$sigma2)
cred_int_sigma2 = quantile(post_x$sigma2,probs=c(0.025,0.975))
print(mean_sigma2)

## [1] 7.918592

print(cred_int_sigma2)

##     2.5%    97.5%
## 6.726024 9.330297

#AR-process for Y
#Calculating mean and credible interval for all parameters
mean_mu = mean(post_y$mu)
cred_int_mu = quantile(post_y$mu,probs=c(0.025,0.975))
print(mean_mu)

## [1] 10.04247

print(cred_int_mu)

##      2.5%     97.5%
## -16.08935  50.19349

mean_phi= mean(post_y$phi)
cred_int_phi = quantile(post_y$phi,probs=c(0.025,0.975))
print(mean_phi)

## [1] 0.957765
```

```
print(cred_int_phi)

##      2.5%     97.5% 
## 0.9116279 1.0007403

mean_sigma2 = mean(post_y$sigma2)
cred_int_sigma2 = quantile(post_y$sigma2,probs=c(0.025,0.975))
print(mean_sigma2)

## [1] 9.138192

print(cred_int_sigma2)

##      2.5%     97.5% 
##  7.744661 10.745055

#Plotting using print
print(fit_x)

## Inference for Stan model: d85eaf939f1fe064d5bdde49c5af5b23.
## 4 chains, each with iter=2000; warmup=1000; thin=1; 
## post-warmup draws per chain=1000, total post-warmup draws=4000.
## 
##           mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu       12.80    0.00 0.19   12.43   12.68   12.80   12.92   13.17  3586    1
## phi       0.14    0.00 0.06    0.04    0.10    0.14    0.18    0.26  3594    1
## sigma2    7.92    0.01 0.66    6.73    7.46    7.88    8.33    9.33  4058    1
## lp__   -455.35    0.03 1.22 -458.63 -455.90 -455.02 -454.47 -453.96  2033    1
## 
## Samples were drawn using NUTS(diag_e) at Tue May  9 10:30:51 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at 
## convergence, Rhat=1).

print(fit_y)

## Inference for Stan model: d85eaf939f1fe064d5bdde49c5af5b23.
## 4 chains, each with iter=2000; warmup=1000; thin=1; 
## post-warmup draws per chain=1000, total post-warmup draws=4000.
## 
##           mean se_mean    sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu       10.04    1.21 17.77  -16.09    5.62    8.79   11.80   50.19   215 1.02
## phi       0.96    0.00  0.02    0.91    0.94    0.96    0.97    1.00   584 1.01
## sigma2    9.14    0.02  0.76    7.74    8.61    9.07    9.64   10.75  1608 1.00
## lp__   -476.53    0.06  1.48 -480.00 -477.38 -476.14 -475.37 -474.75   614 1.01
## 
## Samples were drawn using NUTS(diag_e) at Tue May  9 10:30:56 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at 
## convergence, Rhat=1).
```

```
#Conclusion: As the values in the credible interval are more narrow for the one where
phi=0.2
#were used, we get a better understanding of the true values for this one compared to the
one where
#phi=0.95 were used. For this one, the values are really wide in the credible intervals

#ii)
#Joint posterior for X
plot(post_x$mu[1000:2000],post_x$phi[1000:2000],xlab="My vals",ylab="Phi vals",
main="Joint posterior of mu and phi (X)")

#Joint posterior for X
plot(post_y$mu[1000:2000],post_y$phi[1000:2000],xlab="My vals",ylab="Phi vals",
main="Joint posterior of mu and phi (Y)")

#Conclusion: We can distinguish a convergence for the first sampler. However it is more
difficult to distinguish a
#convergence for the second sampler as shown in the figure. This follows the previous
reasoning, that the
#second sampler have a wider span for its values compared to the first sampler
```