Lab 3, Bayesian learning

mansj125, Isabe723

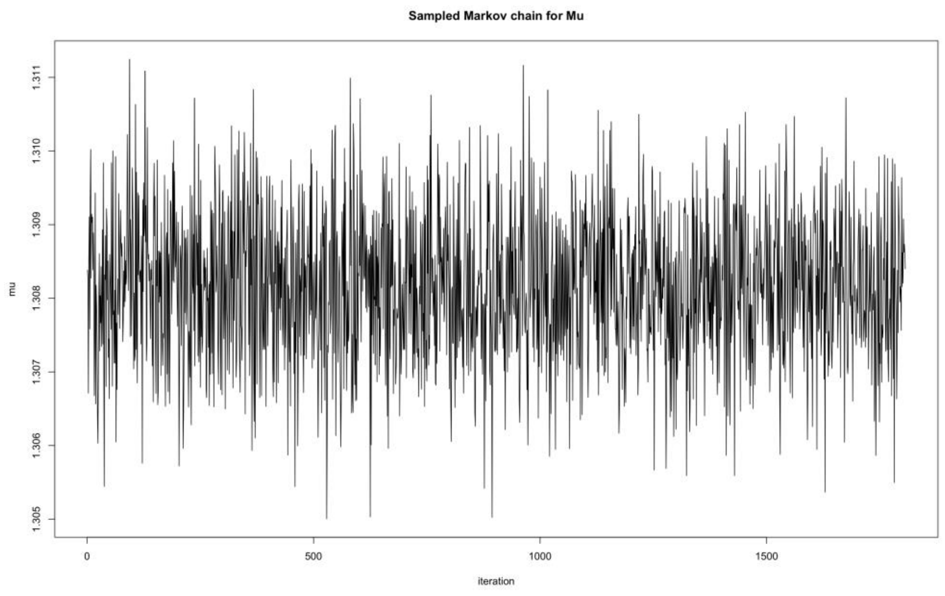
Question 1

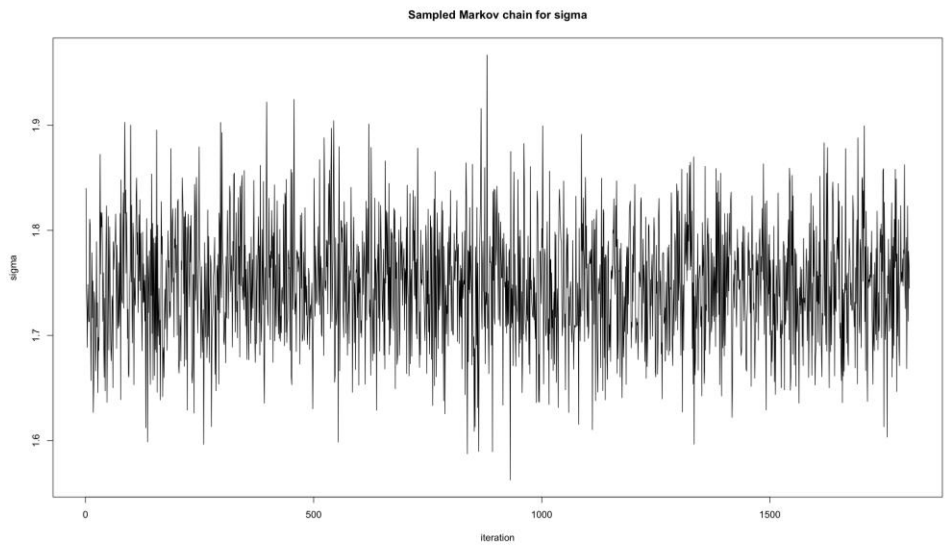
(a)

Inefficiency Factor for μ = 1.26403

Inefficiency Factor for σ^2 = 1.065704

The trajectories of the sampled Markov Chains for μ and σ^2 are given below,





It can be seen that the sample generated from does not seem to completely strongly converge to a single value, but seems to move around a value with some variance.

CODE:

Note: The code includes Gibbs sampling as well as direct sampling which was done for verification purposes

```
library(scales)
 data = readRDS("Precipitation.rds")

# 3.1.a)
 y = log(data)
 n = length(y)
 y_mean = mean(y)

# initializing hyper parameters
 mu_0 = mean(y)
 t_sq_0 = var(y)
 sigma_sq_0 = 1.8
 v0 = 100



# Plotting prior to check if it is similar to the data given
 prior_sigmasq = c()
 N_Draws = 2000
 prior_mu = rnorm(N_Draws,mu_0,t_sq_0)
 for(i in 1:N_Draws){
   X = rchisq(1,v0)
   prior_sigmasq = c(prior_sigmasq, v0*sigma_sq_0 / X)
 }
```

```
# Drawing form the simulated prior mean and sigma^2
 prior_datapoints = c()
 for(i in 1:N_Draws){
   prior_datapoints = c(prior_datapoints, rnorm(1, prior_mu[i],prior_sigmasq[i]))
 }


# Plotting histogram of data points
 hist(prior_datapoints)
```

```
mean(prior_datapoints)
```

```
## [1] 1.370565
```

```
var(prior_datapoints)
```

```
## [1] 6.122464
```

```
# Initial Setting
N_Draws = n
 mu = rnorm(1,mu_0,t_sq_0)
 X = rchisq(1,v0)
 sigmasq = v0*sigma_sq_0 / X


Gibbs_posterior_mu = c()
 Gibbs_posterior_sigma = c()


# Posterior form Gibbs Sampling
 for( i in 1:N_Draws){


  w = (n/sigmasq) / ( (n/sigmasq) + (1 / t_sq_0))
   mu_n = w*y_mean + (1-w)*mu_0
   t_sq_n = 1/((n/sigmasq) + (1/t_sq_0))


  mu = rnorm(1,mu_n,t_sq_n)
   Gibbs_posterior_mu = c(Gibbs_posterior_mu, mu)


  v_n = v0 + n
   X = rchisq(1,v_n)
   thing = (v0*sigma_sq_0 + sum((mu - y)^2))/(n + v0)
   sigmasq = v_n*thing/X
   Gibbs_posterior_sigma = c(Gibbs_posterior_sigma, sigmasq)
 }


acf_mu = acf(Gibbs_posterior_mu)
```

```
acf_sigma = acf(Gibbs_posterior_sigma)



# Calculating Inefficiency Factors for mu and sigma^2
 if_mu = 1+2*sum(acf_mu$acf[-1])
 if_sigma = 1+2*sum(acf_sigma$acf[-1])


hist(Gibbs_posterior_mu)
```

```
hist(Gibbs_posterior_sigma)
```

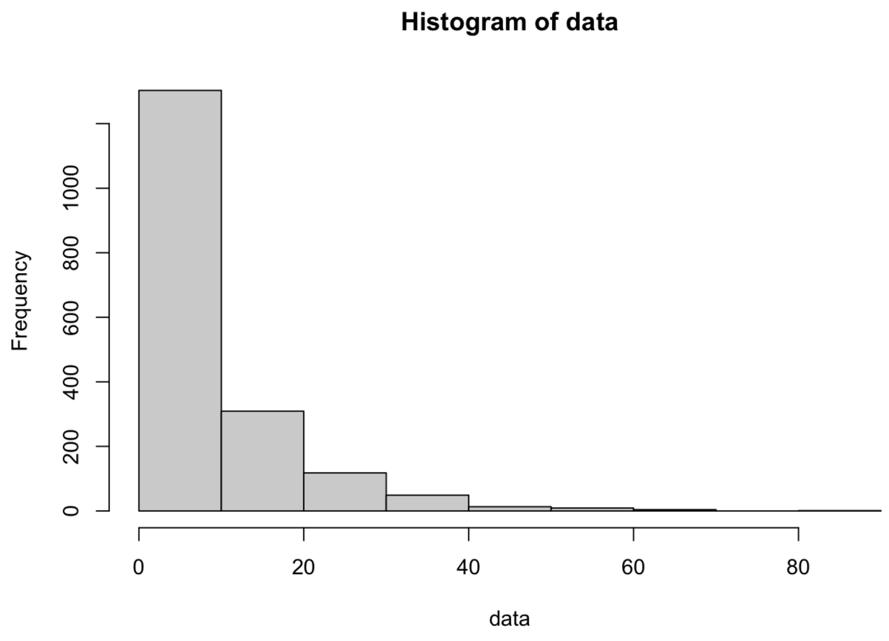
```
plot(Gibbs_posterior_mu,type="l", xlab = "iteration", ylab = "mu", main="Sampled Markov chain for
Mu")
```

```
plot(Gibbs_posterior_sigma,type="l", xlab = "iteration", ylab = "sigma", main="Sampled Markov
chain for sigma")
```

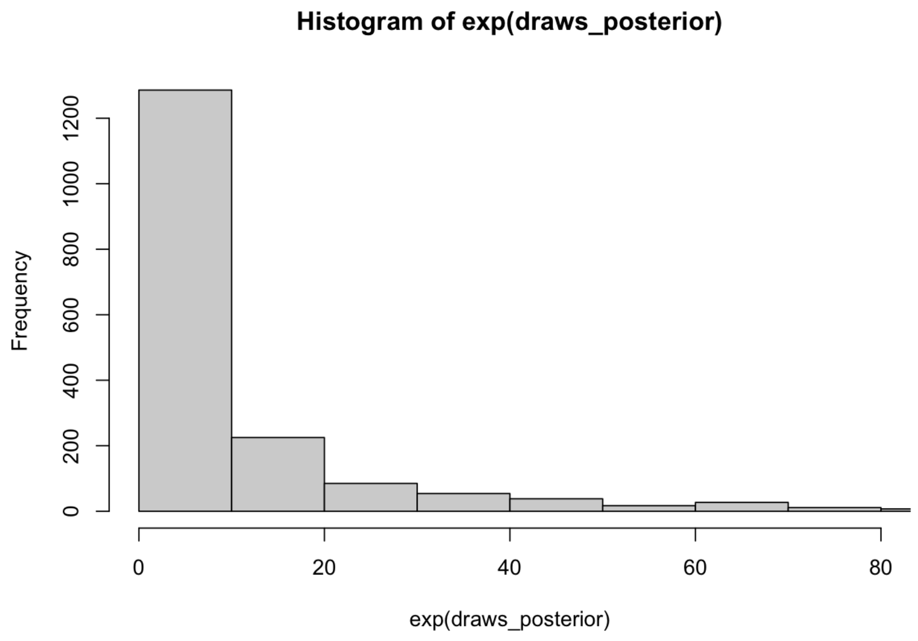
```
# Direct Draws Done for verification
 mu_0 = mean(y)
 k0 = 1
 sigma_sq_0 = 1.8
 v0 = 100


# Draws from Piror
 nDraws = 2000


X = rchisq(nDraws,v0)
 prior_direct_sigmasq = v0*sigma_sq_0 / X


prior_direct_mu = c()
 for(i in 1:nDraws){
   prior_direct_mu = c(prior_direct_mu,

rnorm(1,mu_0,prior_direct_sigmasq[i]/k0))
 }
 hist(mu)

hist(prior_direct_sigmasq)

# Posterior from Direct Sampling
 nDraws = 500
 mu_n = ((k0/(k0+n))*mu_0) + ((n/(k0+n))*y_mean)
 kn = k0 + n
 vn = v0 + n
s_sq = sum((y-y_mean)^2)/(n-1)
 sigmasq_n = (v0*sigma_sq_0 + (n-1)*s_sq + (((k0*n)/(k0+n))*(y_mean - mu_0)^2))/vn


X = rchisq(nDraws,vn)
 direct_sigmasq = vn*sigmasq_n / X


direct_mu = c()
 for(i in 1:nDraws){
   direct_mu = c(direct_mu, rnorm(1,mu_n,direct_sigmasq[i]/kn))

}


hist(direct_mu)

hist(direct_sigmasq)

plot(direct_mu,type="l")

plot(direct_sigmasq,type="l")
```

b)

The histogram of the daily precipitation from the dataset is given below,



The histogram of the posterior prediction from the simulated posterior draws from Gibbs sampling is given below



It can be seen that the posterior prediction also closely resembles the trends of the dataset that was given but the posterior predictions has some outliers.

CODE:

```
# 1.b

hist(data)

draws_posterior = c()
 for(i in 1:n){
   draws_posterior = c(draws_posterior, rnorm(1,Gibbs_posterior_mu[i], Gibbs_posterior_sigma[i]))
 }
 hist(exp(draws_posterior), breaks = 100, xlim=range(0,80))
```

Question 2

A. Significant covariates
   I. Intercept
   II. VerifyID
   III. Sealed
   IV. Logbook
   V. MinBidShare

CODE:

```
library(mvtnorm)
 data <- read.table("eBayNumberOfBidderData.dat", sep = "" , header = T , na.strings ="",
stringsAsFactors= F)
 data = as.data.frame(data)

col_names = colnames(data)

# lab 3.2.1

model <- glm(nBids ~ . , data[,-2], family = poisson(link = "log"))
 model

##
## Call:  glm(formula = nBids ~ ., family = poisson(link = "log"), data = data[,
##     -2])
 ##
## Coefficients:
 ## (Intercept)  PowerSeller     VerifyID       Sealed      Minblem      MajBlem
##     1.07244     -0.02054     -0.39452      0.44384     -0.05220     -0.22087
##     LargNeg      LogBook   MinBidShare
##     0.07067     -0.12068     -1.89410
##
```

```
## Degrees of Freedom: 999 Total (i.e. Null);  991 Residual
## Null Deviance:       2151
## Residual Deviance: 867.5     AIC: 3610
```

```
summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson(link = "log"), data = data[,
##     -2])
##
## 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
```

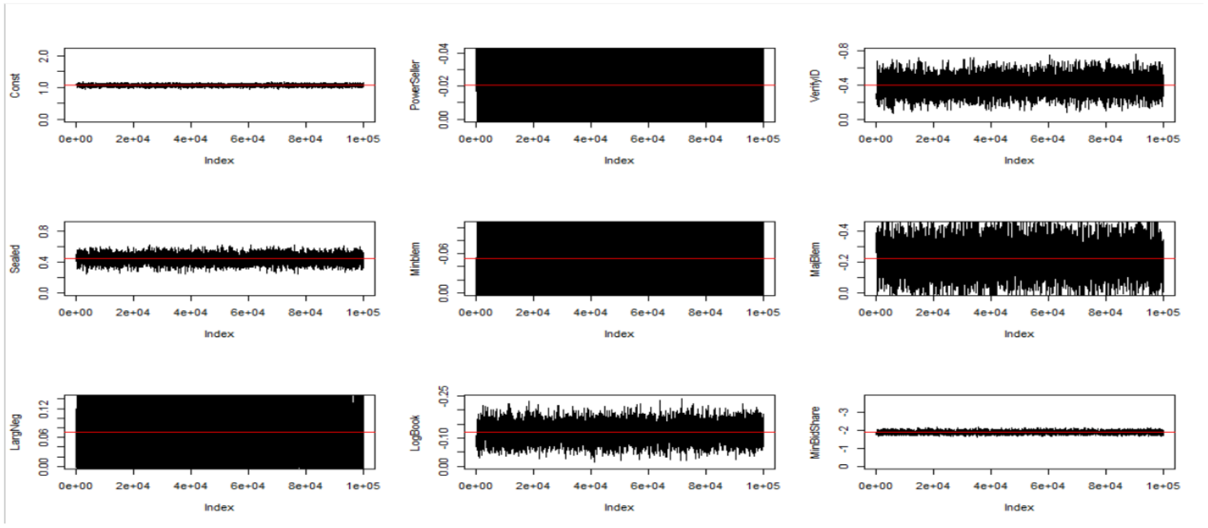
B. See code

CODE:

```
# lab 3.2.2
Y = data$nBids
X = as.matrix(data[,-1])
n = dim(data)[1]
mu = rep(0,9)



Sigma = 100*solve(t(X)%*%X)
beta_prior = rmvnorm(1,mean=mu,sigma=Sigma)
dim(beta_prior)

## [1] 1 9

LogPostPoisson <- function(betas,Y,X,mu,Sigma){
  linPred <- betas%*%t(X);
  # finding the log likelihood
 logLik <- sum(Y*linPred - exp(linPred) - log(factorial(Y)));
  #print(logLik)
  # finding the log logPrior using the parameters given
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  #print(logPrior)
  # returning the log posterior as the sum of loglikihood and logPrior
```

```
   return(logLik + logPrior)
 }
OptimRes <-
optim(beta_prior,LogPostPoisson,gr=NULL,Y,X,mu,Sigma,method=c("BFGS"),control=list(fnscale=-
1),hessian=TRUE)

# Getting the hessian matrix from the results returned from the optim function
 beta_mode_hessian = OptimRes$hessian
 # Getting the mode β values as the parameters returned from the optim function
 beta_mode = OptimRes$par
 colnames(beta_mode) = col_names[-1]
 # Getting the J(beta_mode) as the negative of the hessian returned
J = -beta_mode_hessian
 # Calculating inverse of J
 J_inverse = solve(-beta_mode_hessian)

posterior_beta_draw = rmvnorm(1,beta_mode, J_inverse)
```

C. The beta values do not converge. Neither do they diverge. They oscillate around the values suggested by our glm model from question 1. It seems like the significant(according to glm) features stay closer to the red line than the insignificant ones. Suggesting they influence model performance more.

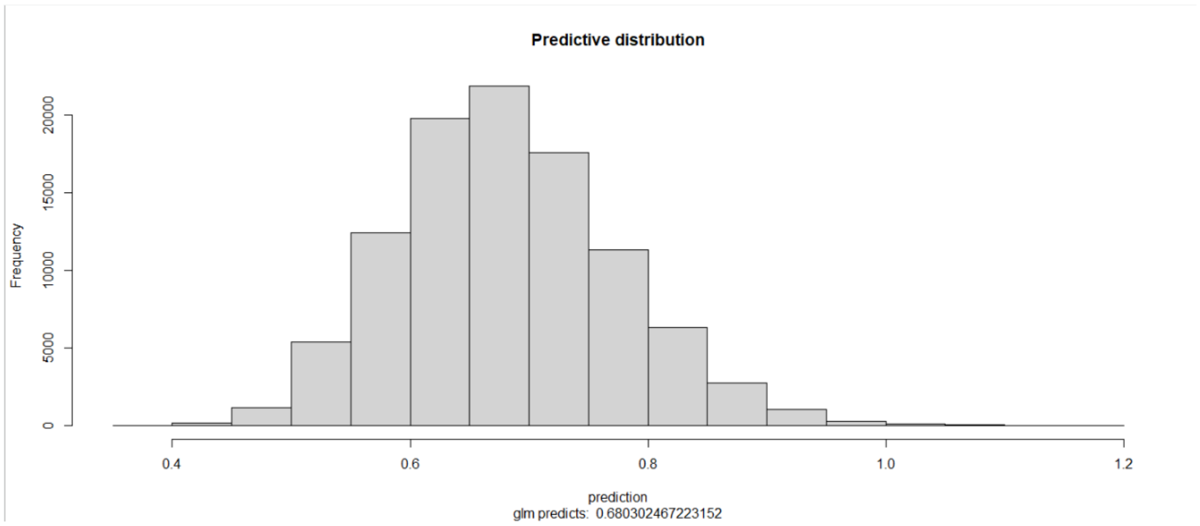


CODE:

```
# lab 3.2.3
 num_random_walk = 1E4
 # Function for metropolis random walk
 MetropolisRandomWalk <- function(beta_initial, c, J_inverse, log_density_function, mu, Sigma, Y,
X){
   metropolis_betas = matrix(data = NA, nrow = num_random_walk, ncol = dim(X)[2])
   # Initializing previous betas as the initial value
   prev_beta = beta_initial
   for(i in 1:num_random_walk){
     # Sampling betas from the multivariate normal distribution
    sample = rmvnorm(1, mean = prev_beta, sigma = c * J_inverse)
     # Calculating the log of the density values for the sampled betas
     log_density_sample = log_density_function(sample, Y, X, mu, Sigma)
```

```
      # Calculating the log of the density values for the previous samples of accepted betas
      log_density_prev_sample = log_density_function(prev_beta, Y, X, mu, Sigma)
     # Calculating alpha values
      alpha = min(1, exp((log_density_sample - log_density_prev_sample)))
      # Generating a random number between 0 and 1
      rand = runif(n=1, min = 0, max = 1)
      # Accepting sample based on the probability alpha
      if(rand<alpha){
        metropolis_betas[i,] = sample
        prev_beta = sample
      }
      else{
        metropolis_betas[i,] = prev_beta
      }

  }
  return(metropolis_betas)
 }
 # Sampling betas based using Metropolis Random Walk
 metropolis_posterior_betas = MetropolisRandomWalk(posterior_beta_draw, .6, J_inverse,
LogPostPoisson, mu, Sigma, Y, X)
 colnames(metropolis_posterior_betas) = col_names[-1]


# plotting Sampled value to access the convergence of betas
 par(mfrow=c(3,3))
 for(i in 1:dim(X)[2]){
   plot(metropolis_posterior_betas[,i], type = "l", ylim = c(0,2*model$coefficients[i]))
   abline(h=model$coefficients[i], col="red")
 }
```

D. Our predictive distribution. The model from question 1 predicted 0.68 on the same test case. Once again it seems they agree.
   We estimate the probability of no bidders to be 50.871%



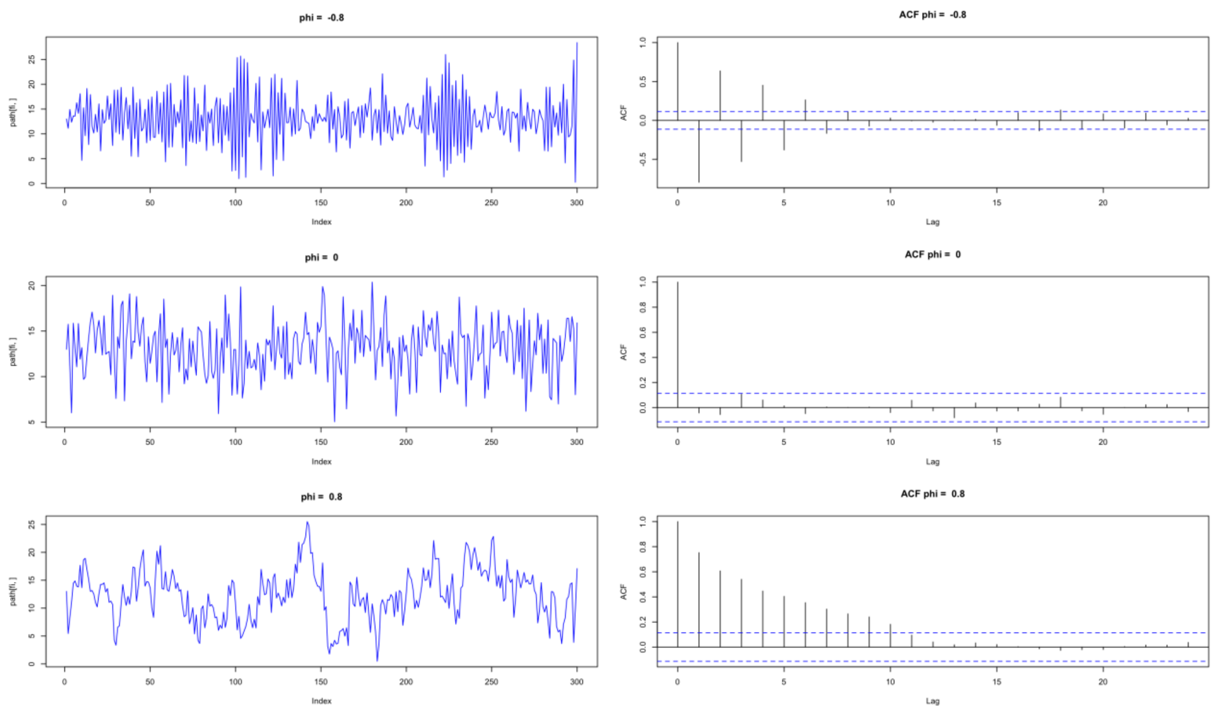
CODE:

```
# lab 3.2.4
 par(mfrow=c(1,1))
 # Creation data for test case
sample = matrix(c(1, 1, 0, 1, 0, 1,0,1.2,0.8))
 # Getting predictions for the test data based on sampled betas as the mean of the Poisson
Distribution is parameters passed it
 # The plot of the parameters passed to it (exp(betas*X)) would be good plot of the predictive
distribution
 prediction = exp(metropolis_posterior_betas %*% sample)


#print(exp(model$coefficients%*%sample))
 hist(prediction, main = "Predictive distribution", sub =paste("glm predicts:
",exp(model$coefficients%*%sample)))



# Calculating probability of no bidders
 prob_no_bids = mean(rpois(num_random_walk,prediction)==0)

prob_no_bids
```

```
[1] 0.5184
```

Question 3

A. See realizations using phi = [-.8, 0, .8] below, including autcorrelation



Using phi < 0 makes the series jump back and forth around the mean in a very erratic way. The autocorrelation is changing sign with each lag-step and approaching zero as the number of lag steps increases

Using phi == 0 gives a trajectory where the only things affecting the values are the mean and the error/random term. There is no significant autocorrelation.

phi > 0 will give us a smoother series of values, where each value will tend to be close to the previous. The autocorrelation is positive but decreases exponentially with the number of lag steps.

CODE:

```
mu = 13
 sigma_sq = 3
 t_max = 300
 fi = 1

# Function to simulate Auto Regressive Function
 AR_Process <- function(mu, FI, sigma_sq)
 {
   path = matrix(data = NA, nrow = length(FI), ncol = t_max)
   i = 1
   for (fi in FI){
     path[i,1] = mu
     for (t in 2:t_max){
       path[i,t] = mu + fi*(path[i,t-1]-mu) + rnorm(1,0,sigma_sq)
     }
    i = i+1
   }
   return(path)
 }
```

```
# Simulating Draws from the AR1 process with phi value -0.8, 0, 0.8
 FI = seq(-.8,.8,.8)
 path = AR_Process(mu, FI, sigma_sq)
 par(mfrow=c(3,2))
 for (fi in 1:length(FI)){
   plot(path[fi,],main=paste("phi = ",FI[fi]),type="line",col="blue")
  acf(path[fi,],main=paste("ACF phi = ",FI[fi]))
 }
```

```
## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
 ## character

## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
 ## character

## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
 ## character
```

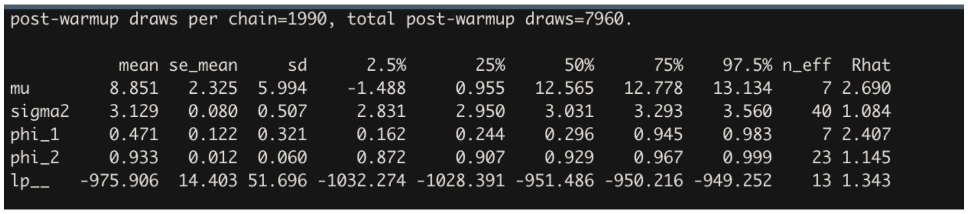
B.

i) The 95% credible intervals for each of the parameters that has been obtained from the model are given below,

   i. Mu - (-1.488, 13.134)
   ii. Sigma - (2.831, 3.560)
   iii. Phi_1 - (0.162, 0.983)
   iv. Phi_2 - (0.872, .999)

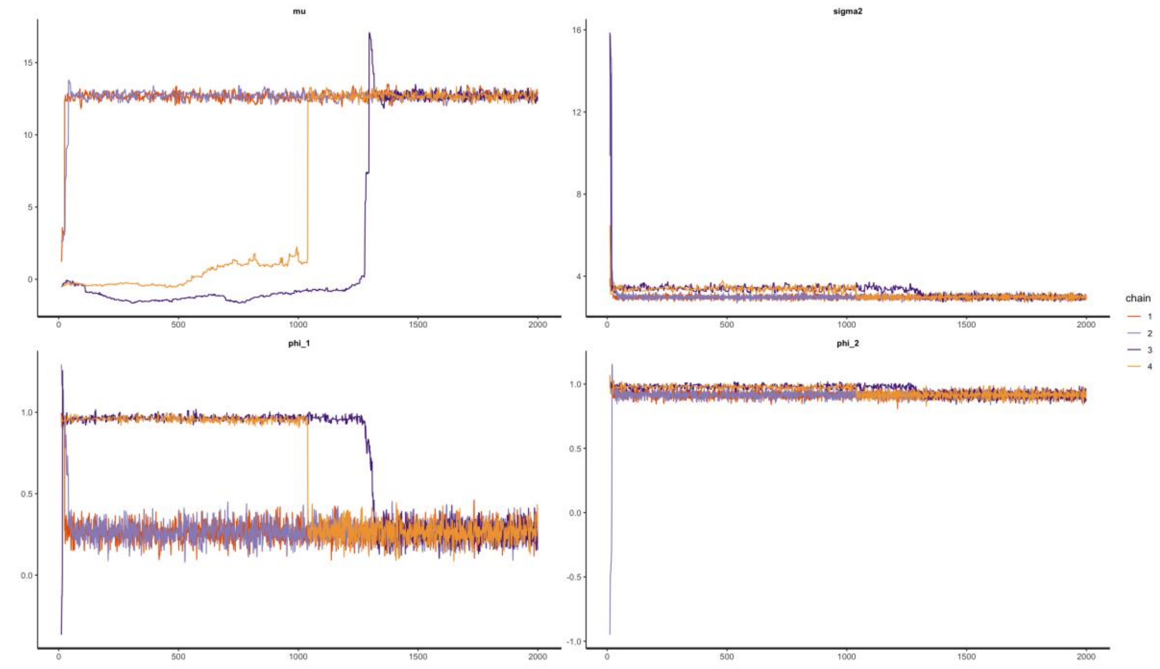
Yes, it can be seen that from the model the true values can be estimated. The means for the parameters from the model is very close to the true values for the parameters and the credible intervals also include the true values of the parameters.

The summary of the model is given below,

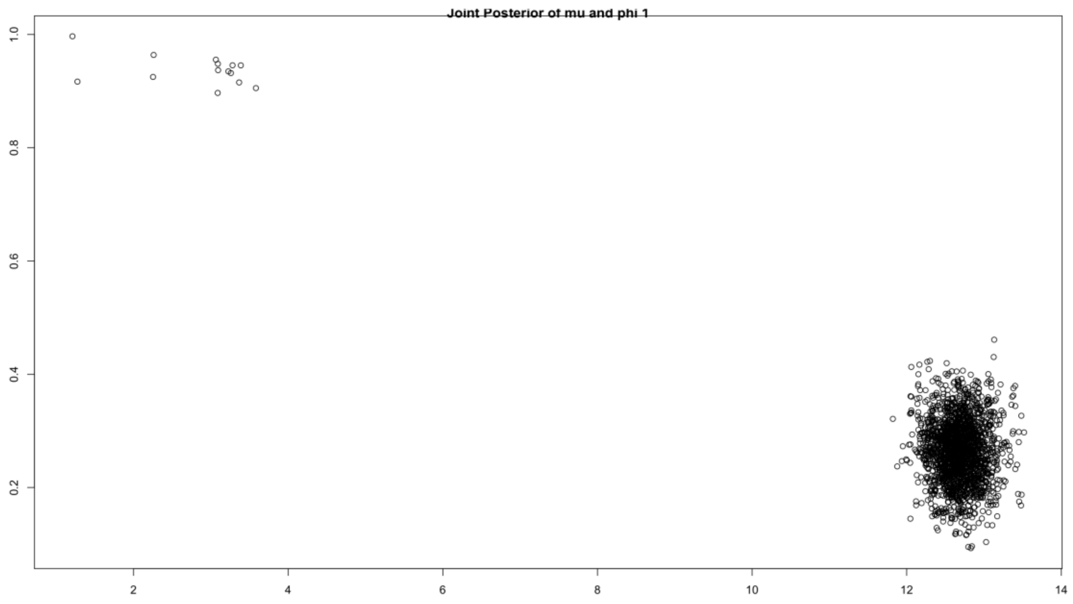


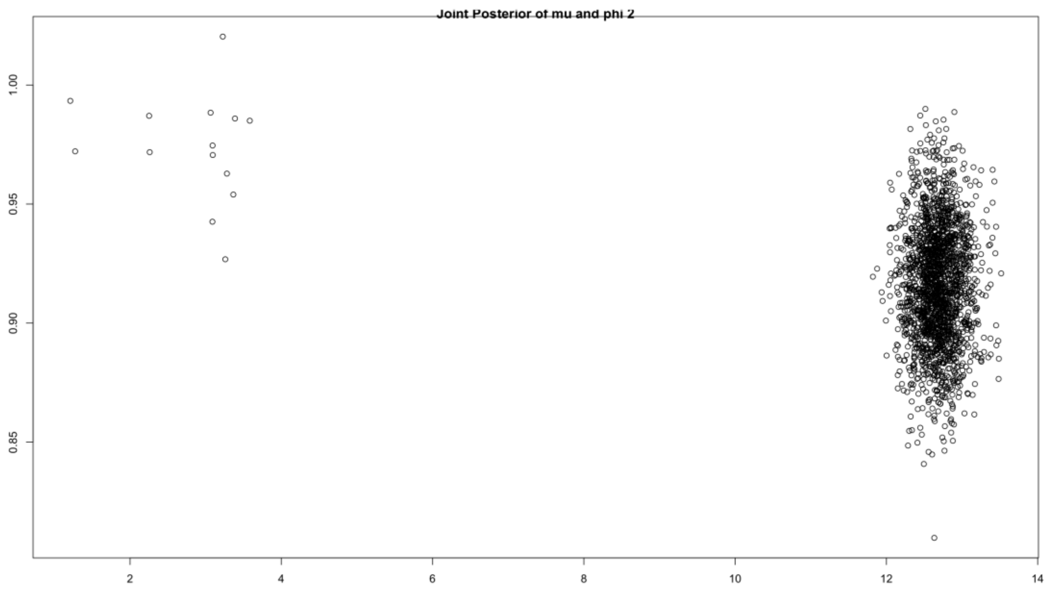
```
post-warmup draws per chain=1990, total post-warmup draws=7960.

            mean se_mean     sd      2.5%       25%      50%      75%    97.5% n_eff  Rhat
mu         8.851   2.325  5.994    -1.488     0.955   12.565   12.778   13.134     7 2.690
sigma2     3.129   0.080  0.507     2.831     2.950    3.031    3.293    3.560    40 1.084
phi_1      0.471   0.122  0.321     0.162     0.244    0.296    0.945    0.983     7 2.407
phi_2      0.933   0.012  0.060     0.872     0.907    0.929    0.967    0.999    23 1.145
lp__    -975.906  14.403 51.696 -1032.274 -1028.391 -951.486 -950.216 -949.252    13 1.343
```

C. Like in question 2.c the values do not converge but rather oscillate around some value and stay there. It seems that no benefit would be gained by performing more iterations.



The joint posterior of mu and phi_1 as well as mu and phi_2 are shown below.
We see that the draws are clustered around the true values in both cases. There are some outliers which were drawn before the chains converged. Increasing the number of warmup-rounds remove them.
The true values are (13, .2) and (13, 0.95) respectively.





CODE:

```
# Creatind Data to pass to stan model
 fi_1 = .2
 fi_2 = .95


FI = c(fi_1, fi_2)
 path = AR_Process(mu, FI, sigma_sq)
 par(mfrow=c(2,2))
 for (fi in 1:length(FI)){
   plot(path[fi,],main=paste("phi = ",FI[fi]),type="line",col="blue")

  acf(path[fi,],main=paste("ACF phi = ",FI[fi]))
 }
```

```
## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
 ## character


## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
 ## character
```

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.5, 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)
```

```
# Creating Stan Model to simulate the two AR(1)-processes


StanModel = '
 data {
   matrix[300,2] M; // synthetic data


}
 parameters {
   real mu;
   real<lower=0> sigma2;
   real phi_1;
   real phi_2;
 }
 model {
   mu ~ uniform(-100,200); // Normal with mean 0, st.dev. 100


sigma2 ~ uniform(0,100); // Scaled-inv-chi2 with nu 1,sigma 2
   phi_1 ~ uniform(-10,10);
   phi_2 ~ uniform(-10,10);


  for(i in 2:300){
     M[i,1] ~ normal(mu + phi_1*(M[i-1,1]-mu),sigma2);
     M[i,2] ~ normal(mu + phi_2*(M[i-1,2]-mu),sigma2);
  }
 }'


data <- list(M=t(path))
 warmup <- 10
 niter <- 2000
 fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)

print(fit,digits_summary=3)
```

```
## Inference for Stan model: 5f654d47b517b96731618cb2a4ab776f.
 ## 4 chains, each with iter=2000; warmup=10; thin=1;
## post-warmup draws per chain=1990, total post-warmup draws=7960.
 ##
##             mean se_mean    sd     2.5%      25%      50%      75%    97.5%
 ## mu        7.855   3.584  5.778   -0.180    0.410   12.337   12.648   13.086
 ## sigma2    3.198   0.103  0.602     2.867     3.000    3.105    3.362    3.556
 ## phi_1     0.575   0.188  0.319     0.217     0.307    0.373    0.948    0.982
 ## phi_2     0.946   0.018  0.048     0.886     0.920    0.947    0.978    1.002
 ## lp__   -988.203  19.685 58.930 -1029.260 -1025.162 -959.295 -957.325 -956.176
 ##        n_eff  Rhat
 ## mu         3 3.916
 ## sigma2    34 1.062
 ## phi_1      3 2.720
 ## phi_2      7 1.220
 ## lp__       9 1.220
 ##
# Extract posterior samples
 postDraws <- extract(fit)

# Do traceplots of the first chain
 par(mfrow = c(1,1))

# Do automatic traceplots of all chains
 traceplot(fit)

# Plotting joint posterior for mu and phi1
 plot(postDraws$mu[1:(niter-warmup)], postDraws$phi_1[1:(niter-

 warmup)], xlab = "mu", ylab = "phi_1", main = "Joint Posterior of mu and phi 1")

# Plotting joint posterior for mu and phi2
 plot(postDraws$mu[1:(niter-warmup)], postDraws$phi_2[1:(niter-warmup)], xlab = "mu", ylab =
"phi_1", main = "Joint Posterior of mu and phi 2")

# Bivariate posterior plots
 pairs(fit)
```