Lab3 Report

aswma317,varsi146

2023-05-13

Question 1

1 a).

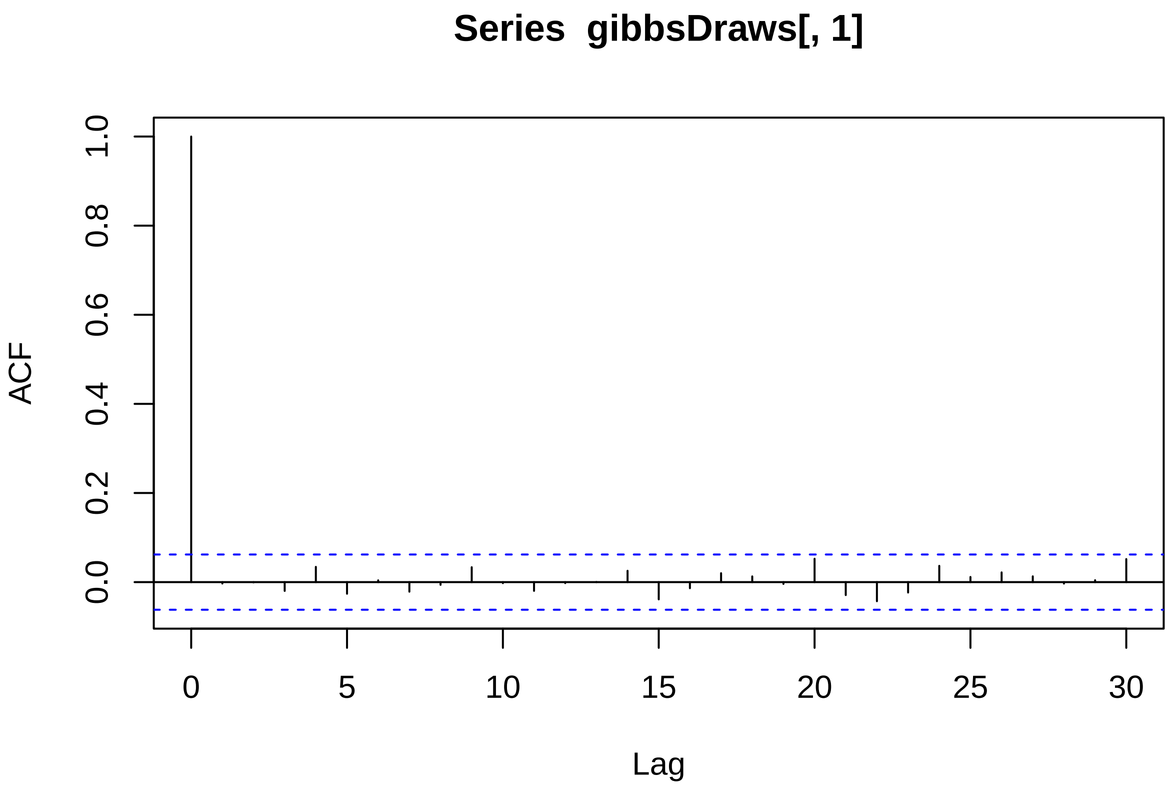
```
####Question 1: Gibbs sampler for Normal model####

#Get data
data <- readRDS('Precipitation.rds')
#ln(data) is Normally distributed
ln_data <- log(data)
n <- length(data)

#Prior mu is N(mu0, tau0_sq)
#Prior sigma_sq is N(nu0, sigma0_sq)
#Initializing the prior parameters
pr_mu0 <- 1; pr_tau0_sq <- 1; pr_nu0 <- 1; pr_sigma0_sq <- 1

#Question a.part 1: simulate from joint posterior from full conditional L7,S16

#Gibbs sampler - only when the full conditional distr is known
#Initial values for posterior pos_sigma_sq:
#set it from fitdistr(ln_data,'normal'), can check if pos_mu matches the output
#pos_sigma_sq <- 1.32087052
pos_mu <- 1.30820257
nDraws <- 1000
gibbsDraws <- matrix(0,nDraws,2)#1-Mean, 2-Var

for (i in 1:nDraws) {
  #Get the nu_n and pos_sigma_sq
  nu_n <- pr_nu0 + n
  #L3,S5 - Draw from scaled inverse chi-square same as in Lab2
  X <- rchisq(n = 1, df = nu_n)
  pos_sigma_sq <- (nu_n *
                     ((pr_nu0*pr_sigma0_sq) + sum((ln_data-pos_mu)^2))/(nu_n))/X
  # Alternative Implementation for posterior of sigma as per L7 i.e, full
  # conditional posterior of sigma in Gibbs Sampling.
  # pos_sigma_sq <- rinvchisq(n = 1, df = nu_n,
  #                           scale = ((pr_nu0*pr_sigma0_sq) +
  #                           sum((ln_data-pos_mu)^2))/(nu_n))
  gibbsDraws[i,2] <- pos_sigma_sq

  #Get the mu_n(mean) and tau_n_sq(var) for pos_mu which is N(mu_n, tau_n_sq)
  # Part of Full Conditional posterior for mean in Gibbs Sampling
  # As part of derivation for Normal Model - Known variance - Normal Prior
  mu_n <- mean(ln_data) * (pr_tau0_sq/(pr_tau0_sq + (pos_sigma_sq/n))) +
    pr_mu0 * (pos_sigma_sq/((n*pr_tau0_sq) + pos_sigma_sq))
  tau_n_sq <- (pr_tau0_sq * pos_sigma_sq)/
    ((n*pr_tau0_sq) + pos_sigma_sq)
  # Full conditional Posterior - L7
  pos_mu <- rnorm(n = 1, mean = mu_n, sd = sqrt(tau_n_sq))
  gibbsDraws[i,1] <- pos_mu
}
gibbs_posterior <- apply(gibbsDraws, 2, mean)

print('The mean of the fully conditional posterior parameters are:\n')
```

```
## [1] "The mean of the fully conditional posterior parameters are:\n"
```

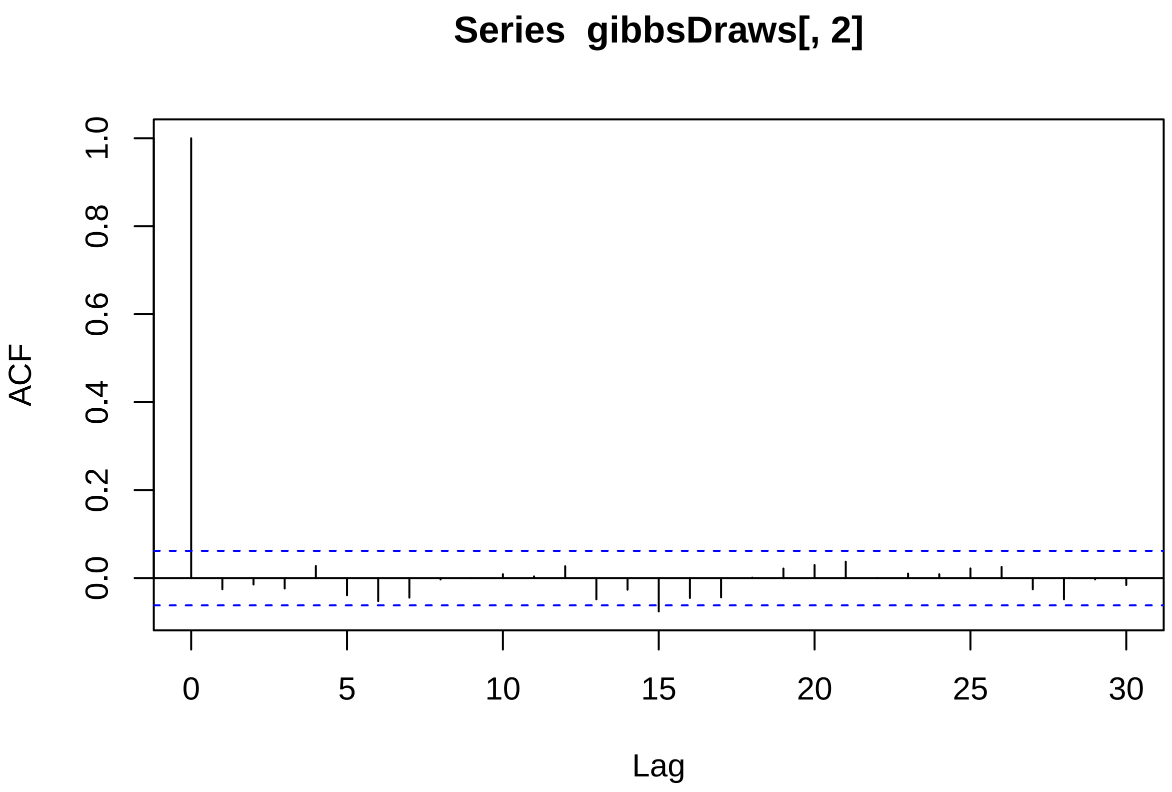
```
print(gibbs_posterior)
```

```
## [1] 1.307764 1.747503
```

```
#Question a.part 2: Evaluate convergence by calculating the Inefficiency Factor
#and by plotting the trajectories of the sampled Markov chains.

# acf calculates the auto-correlation between draws for given lag
a_mu_Gibbs <- acf(gibbsDraws[,1])
```



```
a_var_Gibbs <- acf(gibbsDraws[,2])
```



```r
IF_mu_Gibbs <- 1+2*sum(a_mu_Gibbs$acf[-1])
IF_var_Gibbs <- 1+2*sum(a_var_Gibbs$acf[-1])

print(paste('The inefficiency factor of Gibbs draws with respect to posterior mean is:',
            round(IF_mu_Gibbs,2)))
```

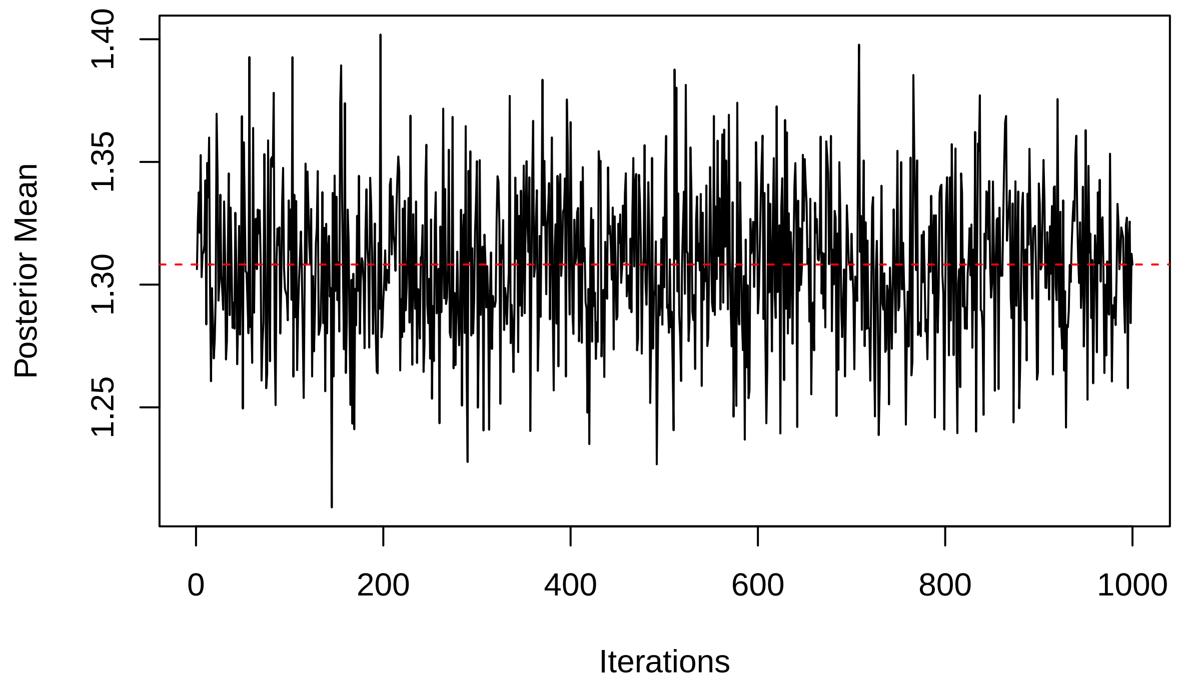
```
## [1] "The inefficiency factor of Gibbs draws with respect to posterior mean is: 1.13"
```

```r
print(paste('The inefficiency factor of Gibbs draws with respect to posterior sigma-squared is:',
            round(IF_var_Gibbs,2)))
```

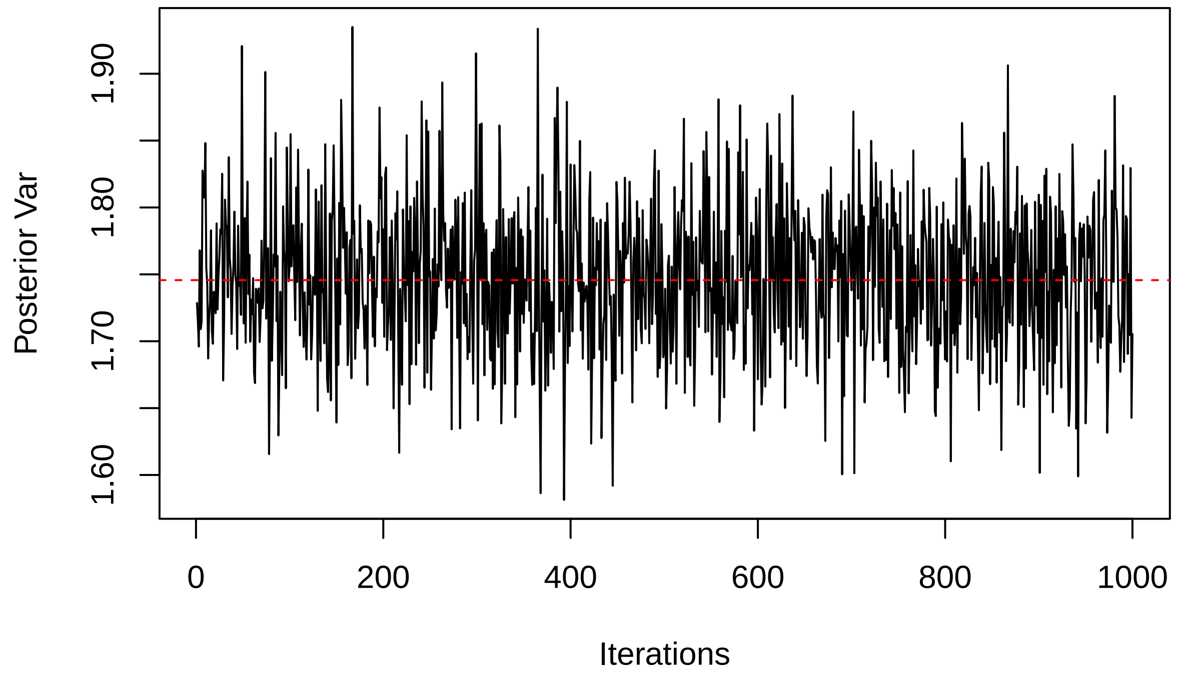
```
## [1] "The inefficiency factor of Gibbs draws with respect to posterior sigma-squared is: 0.38"
```

As seen above, there is moderate level of correlation between the draws, however we can conclude that the Gibbs samples are efficient and converging. The below plot confirm that they converge(the horizontal lines indicates the true mean and variance).

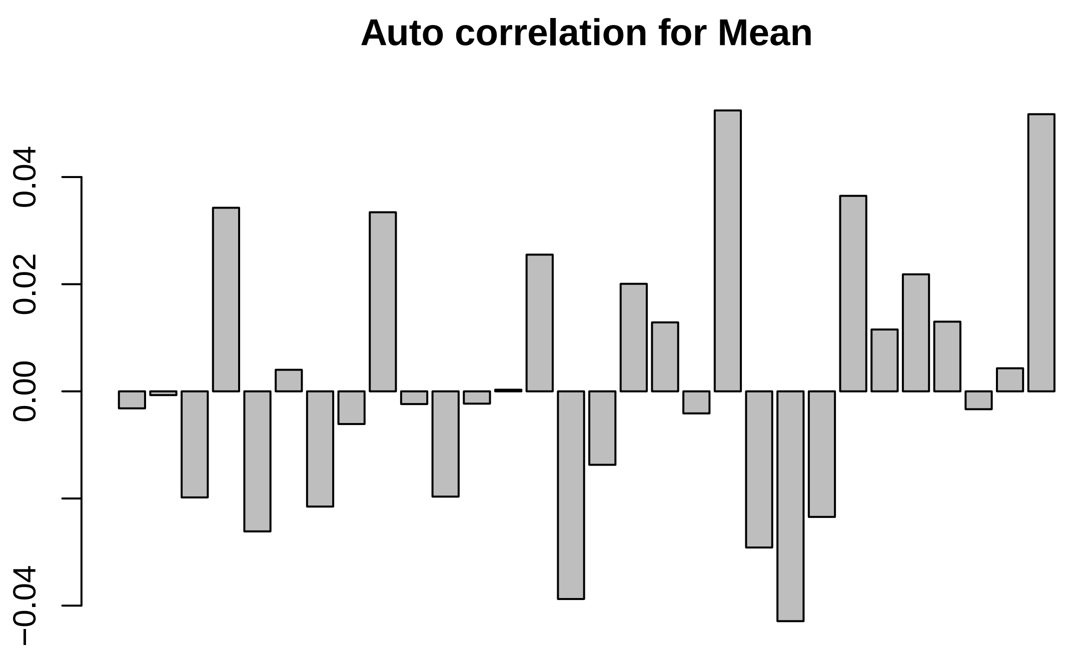
```r
# Traceplot of Gibbs draws
# par(mfrow=c(2,1))
plot(1:nDraws, gibbsDraws[,1], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Mean')
abline(h = mean(ln_data), lty = 2, col = "red")
```



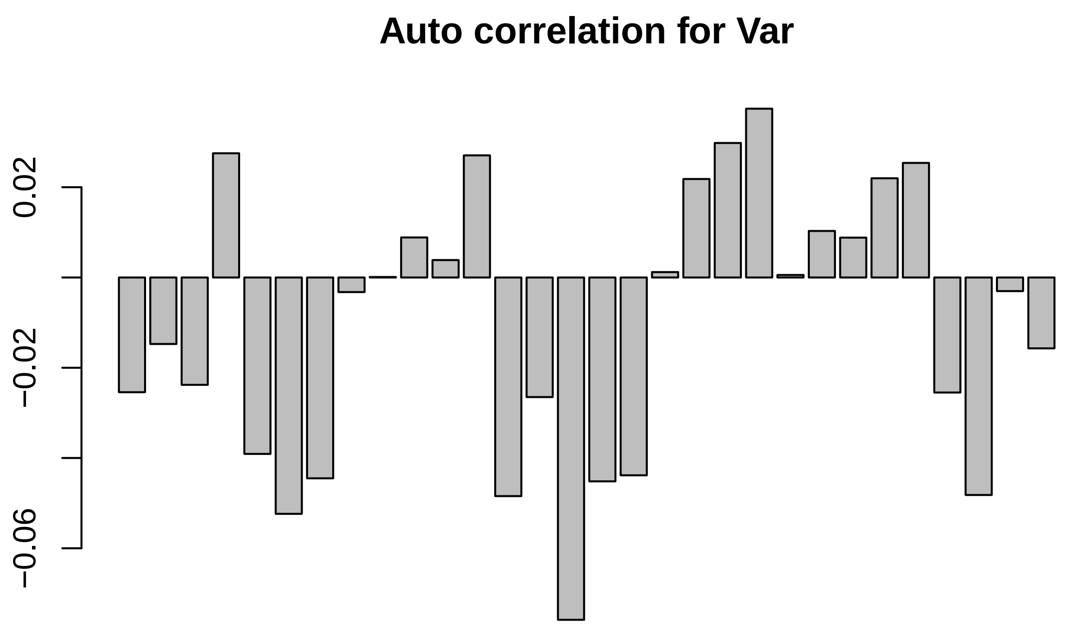
```
plot(1:nDraws, gibbsDraws[,2], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Var') # traceplot of Gibbs draws
abline(h = var(ln_data), lty = 2, col = "red")
```



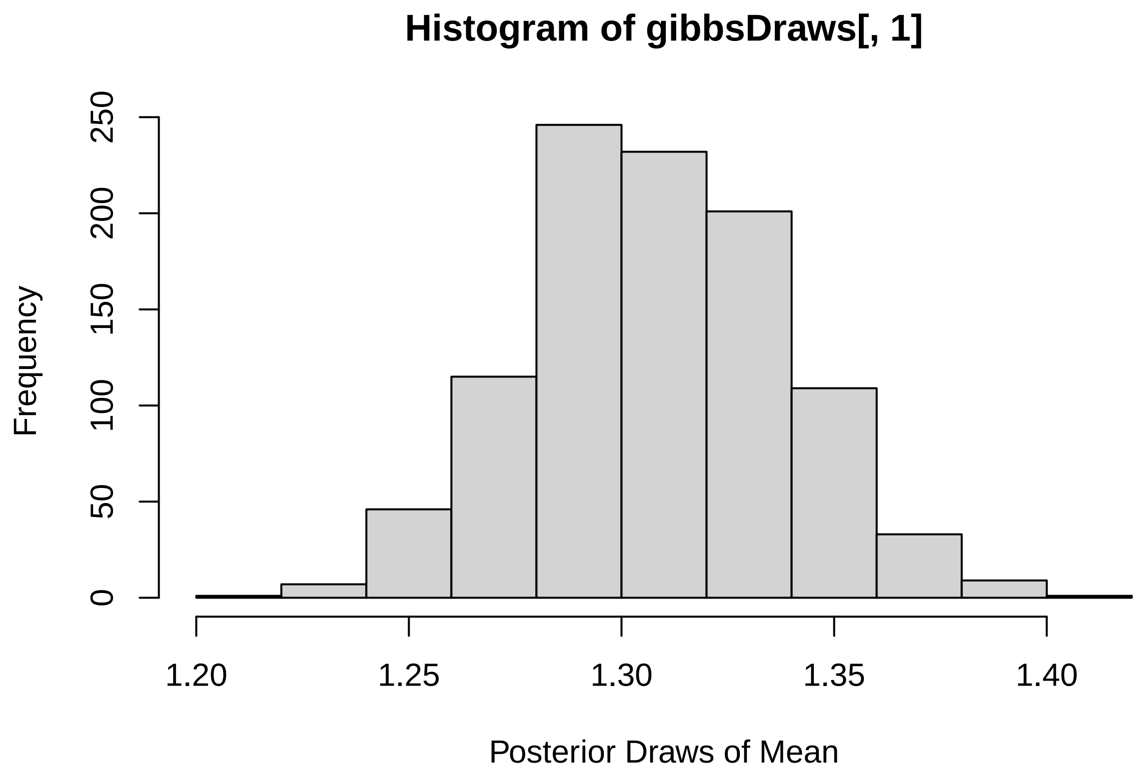
```
# par(mfrow=c(2,1))
# Acf for Gibbs draws
barplot(height = a_mu_Gibbs$acf[-1], main = 'Auto correlation for Mean')
```



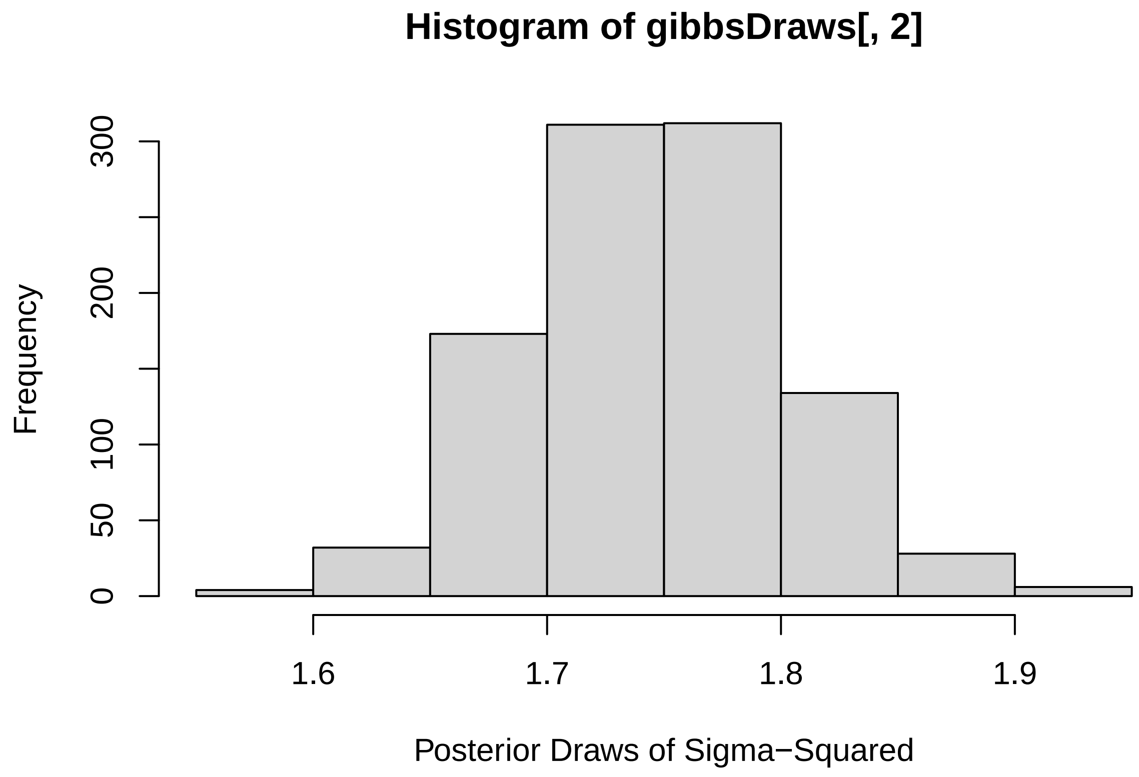
```
barplot(height = a_var_Gibbs$acf[-1], main = 'Auto correlation for Var')
```



```
# par(mfrow=c(2,1))
# Histogram of Gibbs draws
hist(gibbsDraws[,1], xlab = 'Posterior Draws of Mean')
```

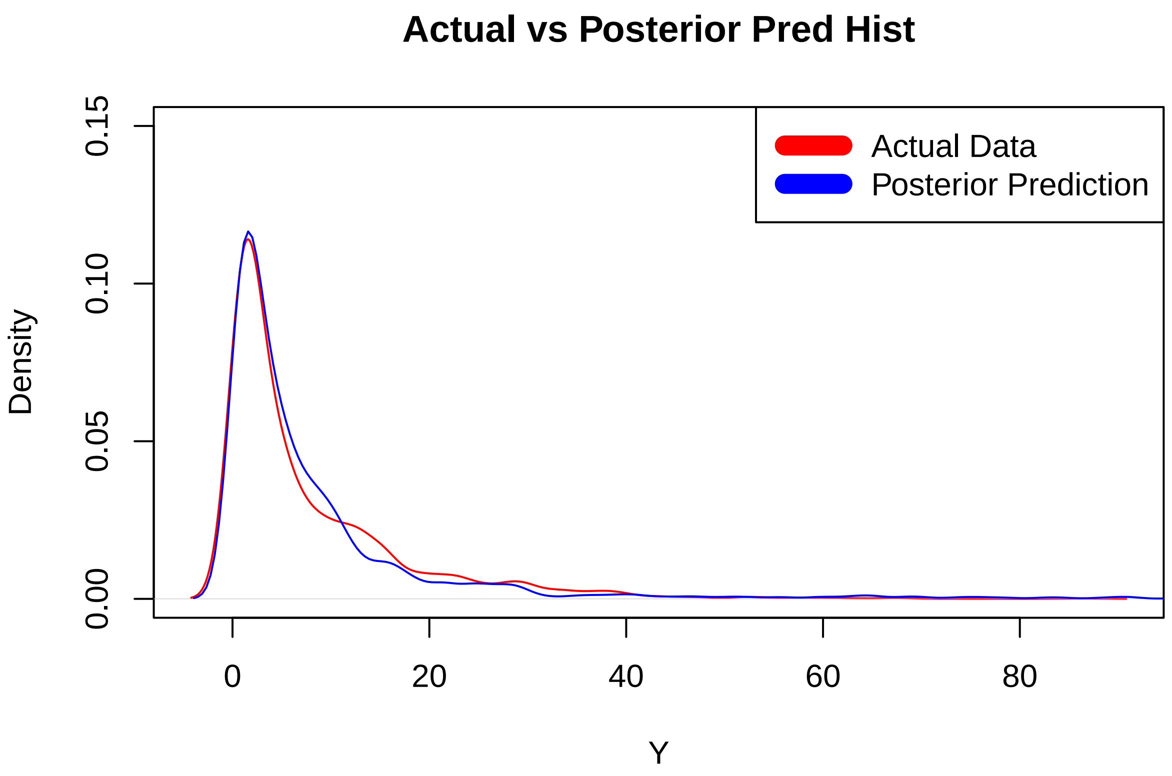


```r
hist(gibbsDraws[,2], xlab = 'Posterior Draws of Sigma-Squared')
```



1 b).

```r
#Question b: Posterior predictive density vs Actual data
#lnY is Normal
# Applying the posterior parameters to draw posterior predictions
pos_pred_ln <- apply(gibbsDraws, 1,
                     function(x) rnorm(n = 1, mean = x[1], sd = sqrt(x[2])))
plot(density(data), type = "l",col="red",
     xlab = 'Y', ylab = 'Density', main = "Actual vs Posterior Pred Hist",
     ylim = c(0, 0.15))
lines(density(exp(pos_pred_ln)), type = "l",col="blue")
legend("topright", legend=c("Actual Data", "Posterior Prediction"),
       col=c("red", "blue"), lwd=10)
```



As seen above, the posterior predictive density agrees with the actual data.

Question 2

2 a).

```r
#Get data
data <- read.table('eBayNumberOfBidderData.dat', header = TRUE)
X <- data[,2:ncol(data)]
y <- data[,'nBids']

#Question a: Get the Maximum Likelihood Estimator of beta using glm
glm_model <- glm(formula = nBids ~ ., family = 'poisson',
                 data = subset(data, select = -Const))

print('Maximum likelihood estimators for betas:')
```

```
## [1] "Maximum likelihood estimators for betas:"
```

```
glm_model$coefficients
```

```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07244206 -0.02054076 -0.39451647  0.44384257 -0.05219829 -0.22087119
##     LargNeg     LogBook MinBidShare
##  0.07067246 -0.12067761 -1.89409664
```

```
sign_coeff <- (summary(glm_model)$coefficients[,c(1,4)])
print('Significant covariates:')
```

```
## [1] "Significant covariates:"
```

```
sign_coeff[sign_coeff[,2] < 0.05 ,] #p-values less than 0.05 are significant
```

```
##               Estimate      Pr(>|z|)
## (Intercept)  1.0724421 4.567273e-266
## VerifyID    -0.3945165  1.968202e-05
## Sealed       0.4438426  1.663233e-18
## MajBlem     -0.2208712  1.571055e-02
## LogBook     -0.1206776  3.094651e-05
## MinBidShare -1.8940966 9.422877e-156
```

2 b).

```
#Question b: Get beta approximation
#Data - Y/nBids is Poisson[exp(X%*%beta)]
#Prior - beta is Normal[0, 100.solve(XX)]
#Posterior - beta is multivariate normal[posterior mode, neg Hessian at pos mode]

# Functions that returns the log posterior for the Poisson regression.
# First input argument of this function must be the parameters we optimize on,
# i.e. the regression coefficients beta.

LogPostPoisson <- function(betas,y,X,mu,Sigma){
  linPred <- X%*%betas
  logLik <- sum( linPred*y - exp(linPred) - log(factorial(y)) )
  #if (abs(logLik) == Inf) logLik = -20000; # Likelihood is not finite, stear the optimizer away from h
  logPrior <- dmvnorm(t(betas), mu, Sigma, log=TRUE)#For multivariate

  return(logLik + logPrior)
}

# Setting up the prior
X <- as.matrix(X)
Npar <- dim(X)[2]
mu <- rep(0,Npar) # Prior mean vector
Sigma <- 100 * solve(t(X) %*% X) # Prior covariance matrix

# Select the initial values for beta
initVal <- matrix(0,Npar,1)

# The argument control is a list of options to the optimizer optim,
# where fnscale=-1 means that we minimize
# the negative log posterior. Hence, we maximize the log posterior.
```

```
OptimRes <- optim(initVal,LogPostPoisson,gr=NULL,y,X,mu,
                  Sigma,method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)

beta_hat <- OptimRes$par
print('The posterior mode is:')
```

```
## [1] "The posterior mode is:"
```

```
print(beta_hat)
```

```
##              [,1]
##  [1,]  1.06984118
##  [2,] -0.02051246
##  [3,] -0.39300599
##  [4,]  0.44355549
##  [5,] -0.05246627
##  [6,] -0.22123840
##  [7,]  0.07069683
##  [8,] -0.12021767
##  [9,] -1.89198501
```

```
print('Values of  negative inverse of observed information at mode')
```

```
## [1] "Values of  negative inverse of observed information at mode"
```

```
solve(-OptimRes$hessian)
```

```
##                [,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,] -2.772238e-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 -4.292828e-04 -5.759169e-05 -6.437067e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.772238e-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 -4.292828e-04
##  [4,]  4.532308e-04  3.376467e-04 -1.311025e-04 -5.759169e-05
##  [5,]  3.492353e-04  5.844006e-05  5.854011e-05 -6.437067e-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 c).

```
# Question c: Simulate from posterior beta using Metropolis Hasting

# Simulate using N as proposal - L8,S8(Metropolis Hastings RW Algo)
RWMSampler <- function(logPostFunc, Nsamples, Npar, ...){
  # The ... is to use any arbitrary logPostFunc as a function object, provided
  # that the first argument of the function is betas.
```

```r
  #Initialize post beta matrix
  pos_betas <- matrix(data = NA, nrow = 0, ncol = Npar)
  #Step1: Initialize prev beta values
  prev_betas <- matrix(0,Npar,1)
  pos_betas <- rbind(pos_betas, t(prev_betas))

  for (i in 1:Nsamples) {
    # RWM Algorithm
    #Step2: Sample from proposal
    c <- 0.5#Step value
    cov_mat <- solve(-OptimRes$hessian)
    new_betas <- rmvnorm(n = 1, mean = prev_betas, sigma = c * cov_mat)
    #Step3: Get Acceptance Probability, alpha
    pos_prop_den <- logPostFunc(t(new_betas), ...)
    pos_prev_den <- logPostFunc(prev_betas, ...)
    #When you compute the acceptance probability, program the log posterior density
    #Here we take the exp as we have the log of the posterior density
    alpha_temp <- exp(pos_prop_den - pos_prev_den) #posterior density ratio
    alpha <- min(1, alpha_temp)
    #Step4: Accept or reject the new betas
    if (alpha > runif(1)) {
      #Accept the new beta values
      prev_betas <- t(new_betas)
      pos_betas <- rbind(pos_betas, new_betas)
    }else{
      #Beta values does not change
      prev_betas <- prev_betas
    }
  }
  return(pos_betas)
}
Nsamples = 5000
pos_beta_mat <- RWMSampler(logPostFunc = LogPostPoisson,
                           Nsamples = Nsamples,
                           Npar = Npar,
                           y,X,mu,Sigma)
print('Acceptance rate:')
```

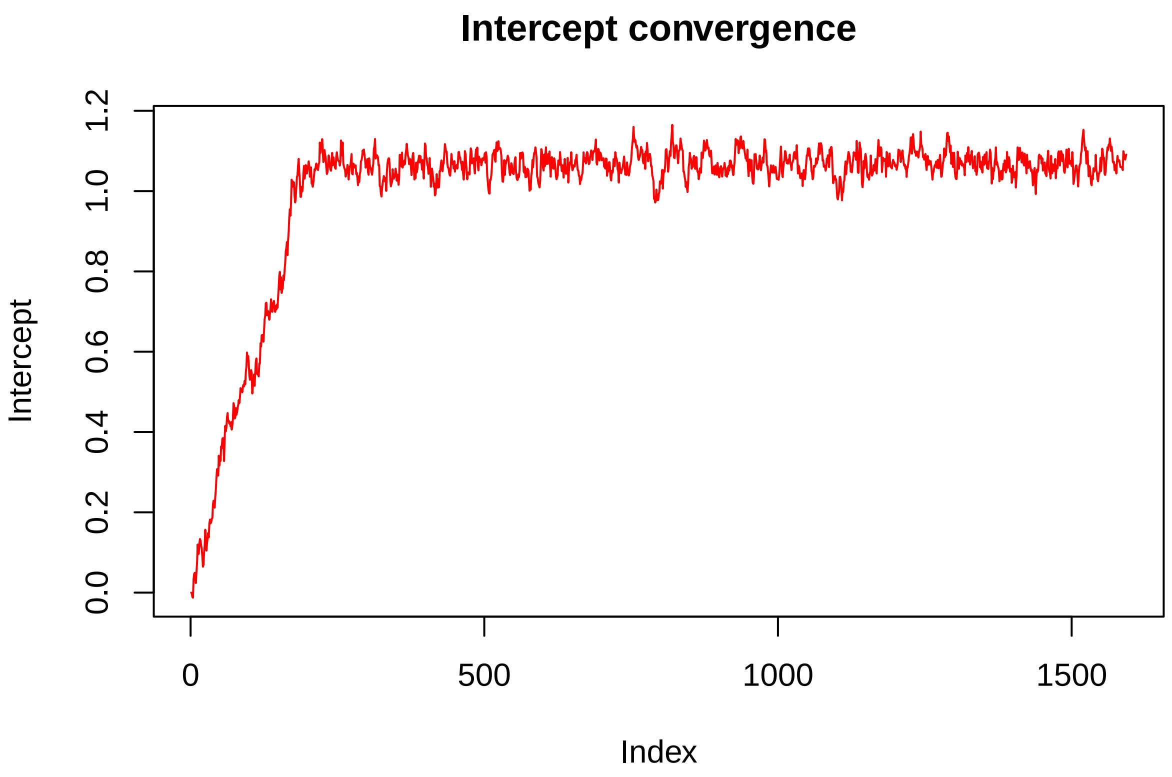
```
## [1] "Acceptance rate:"
```

```r
nrow(pos_beta_mat)/Nsamples#Tune c values based on acceptance rate: 25-30%
```

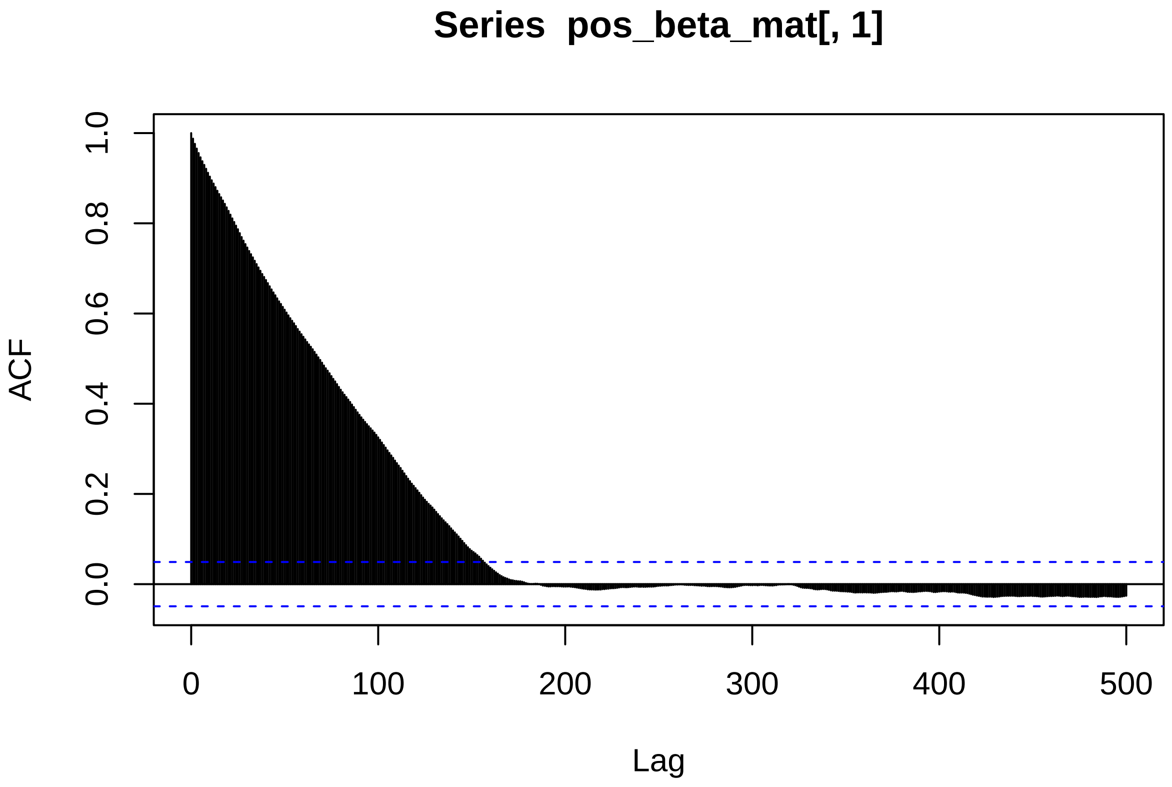
```
## [1] 0.3186
```

As seen below, all the covariates are converging to the stationary distribution. Additionally, we can observe from the acf plots that there is a rapid decay in the autocorrelation of the samples as the lag increase which suggests that samples are nearly independent to each other indicating that the chain has converged to a stationary distribution.

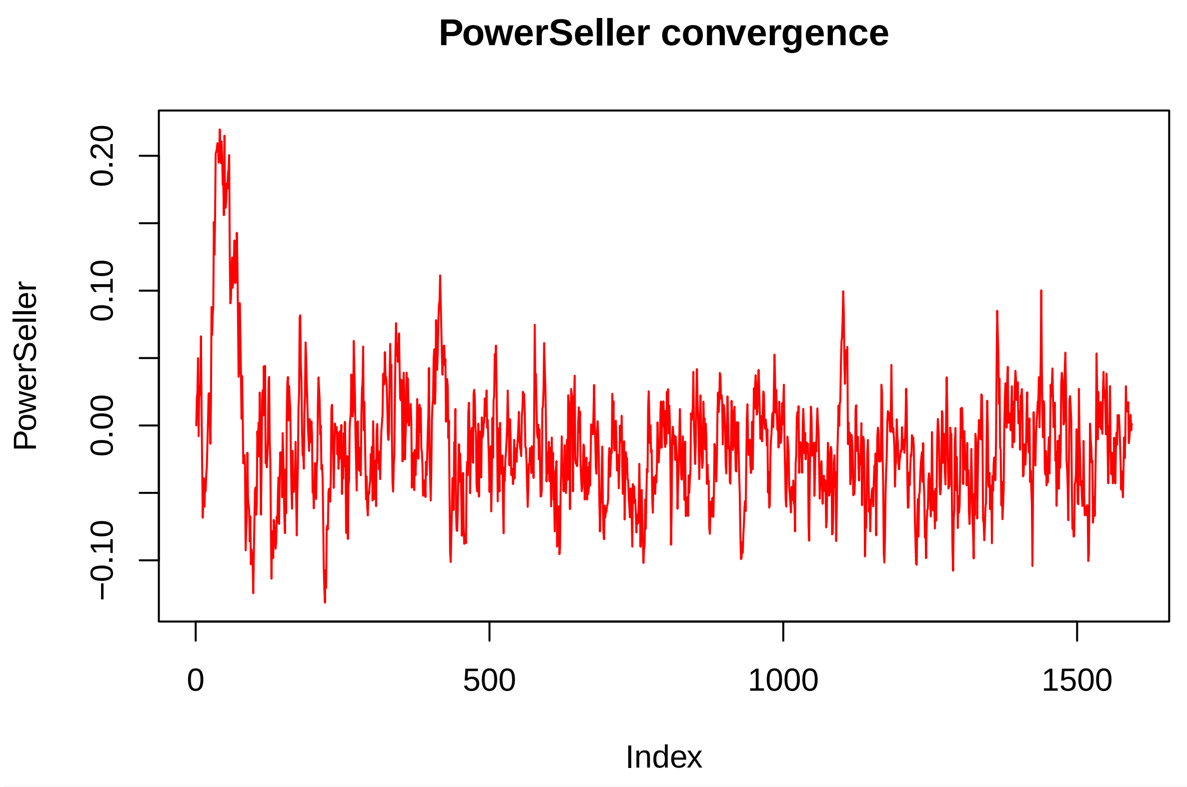
```r
plot(pos_beta_mat[,1], type = "l",col= "red",
     ylab = 'Intercept', main = "Intercept convergence")
```



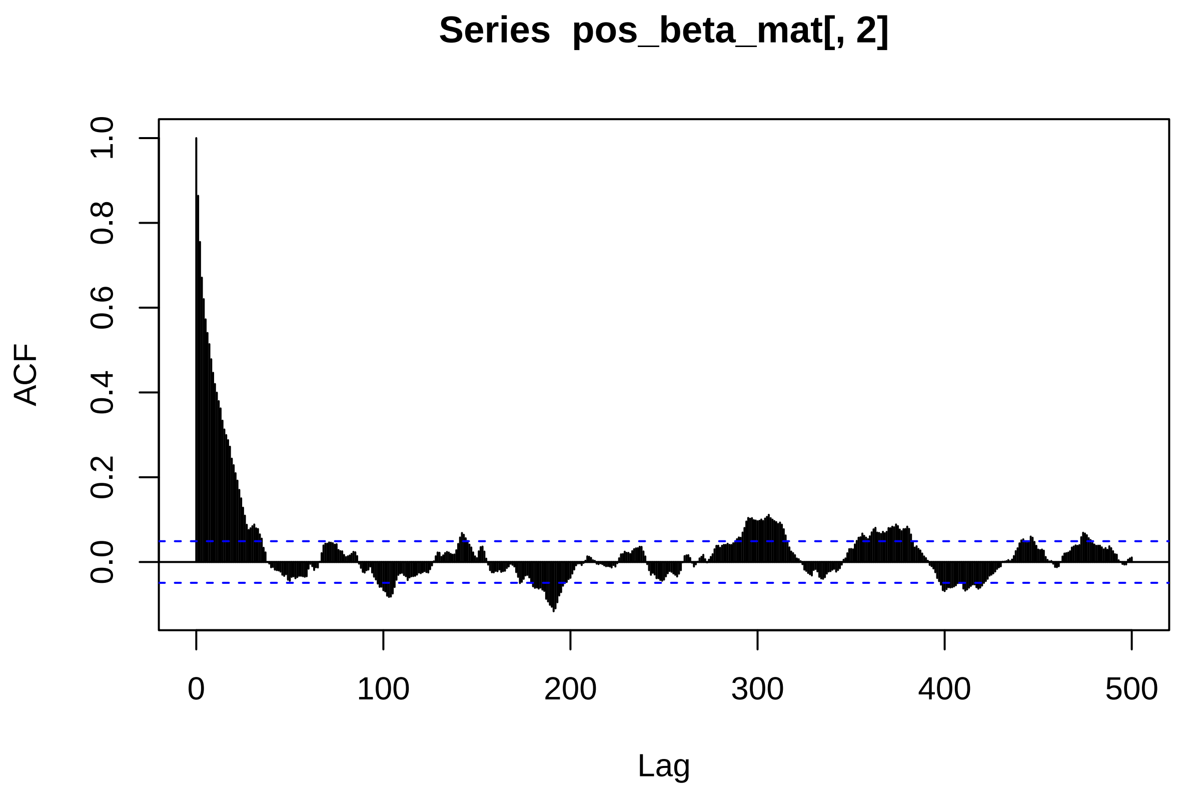
```r
acf(pos_beta_mat[,1], lag.max = 500)
```



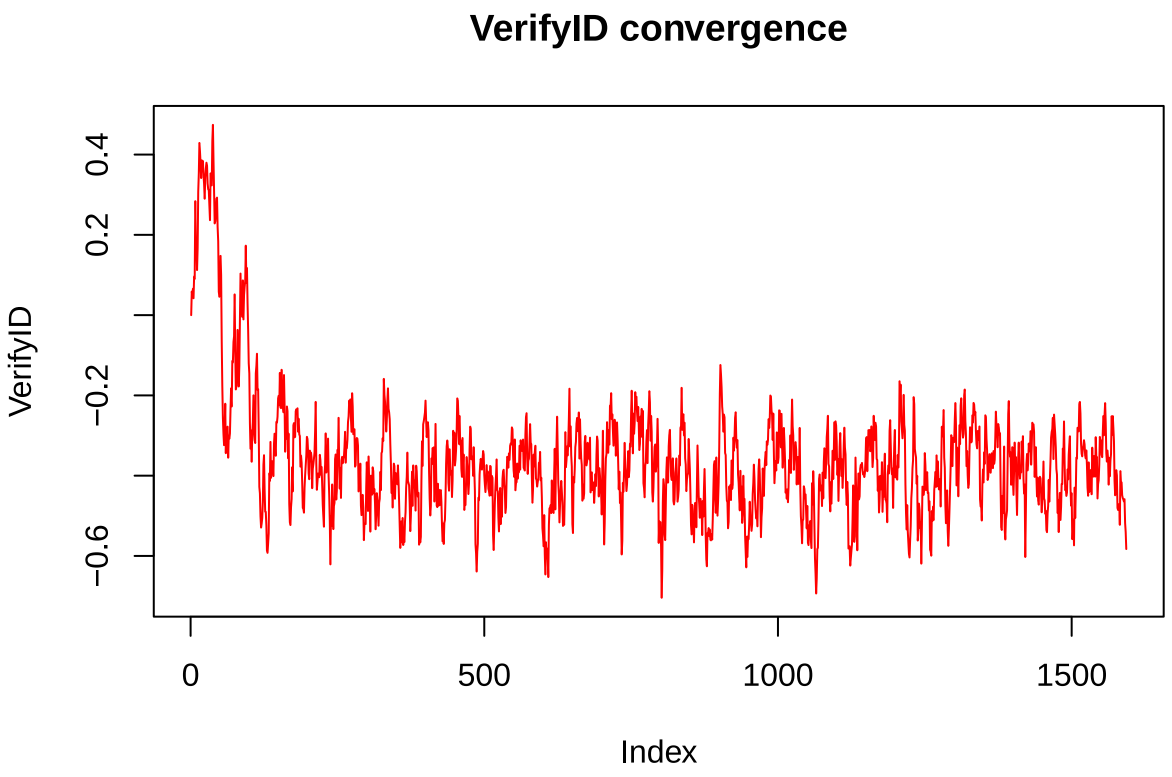
```r
plot(pos_beta_mat[,2], type = "l",col="red",
     ylab = 'PowerSeller', main = "PowerSeller convergence")
```



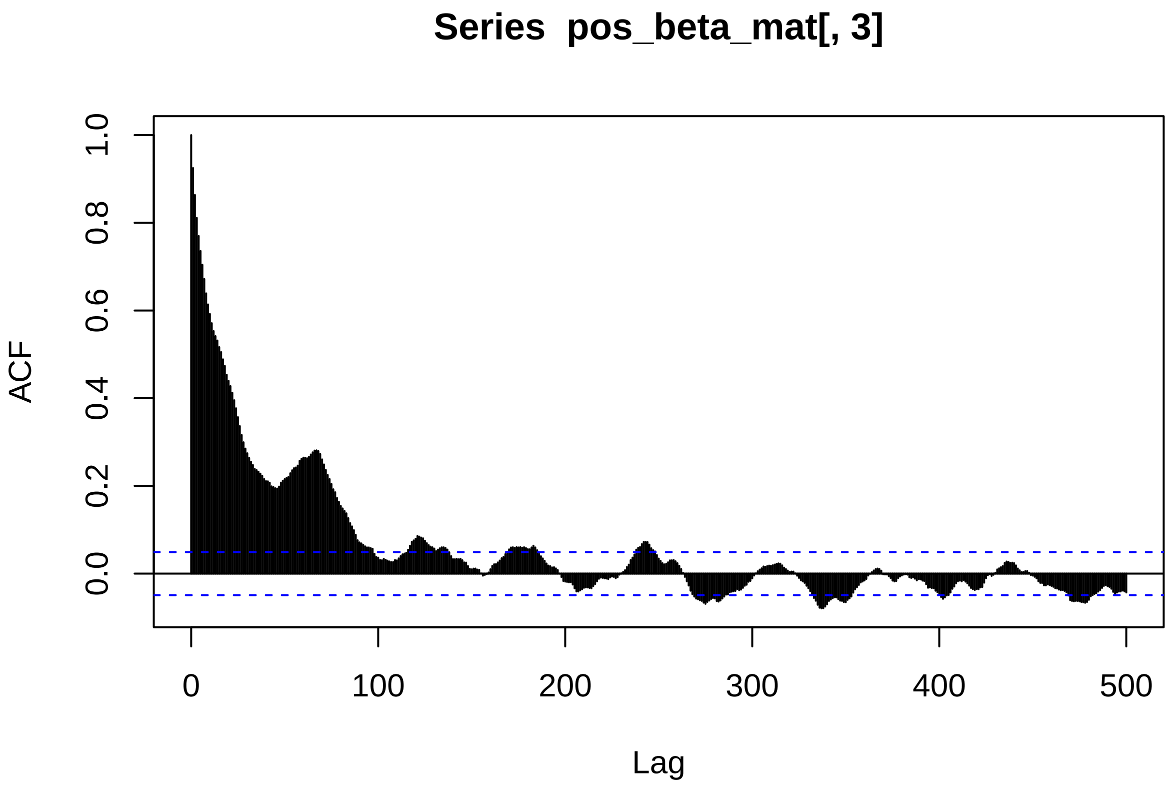
```
acf(pos_beta_mat[,2], lag.max = 500)
```



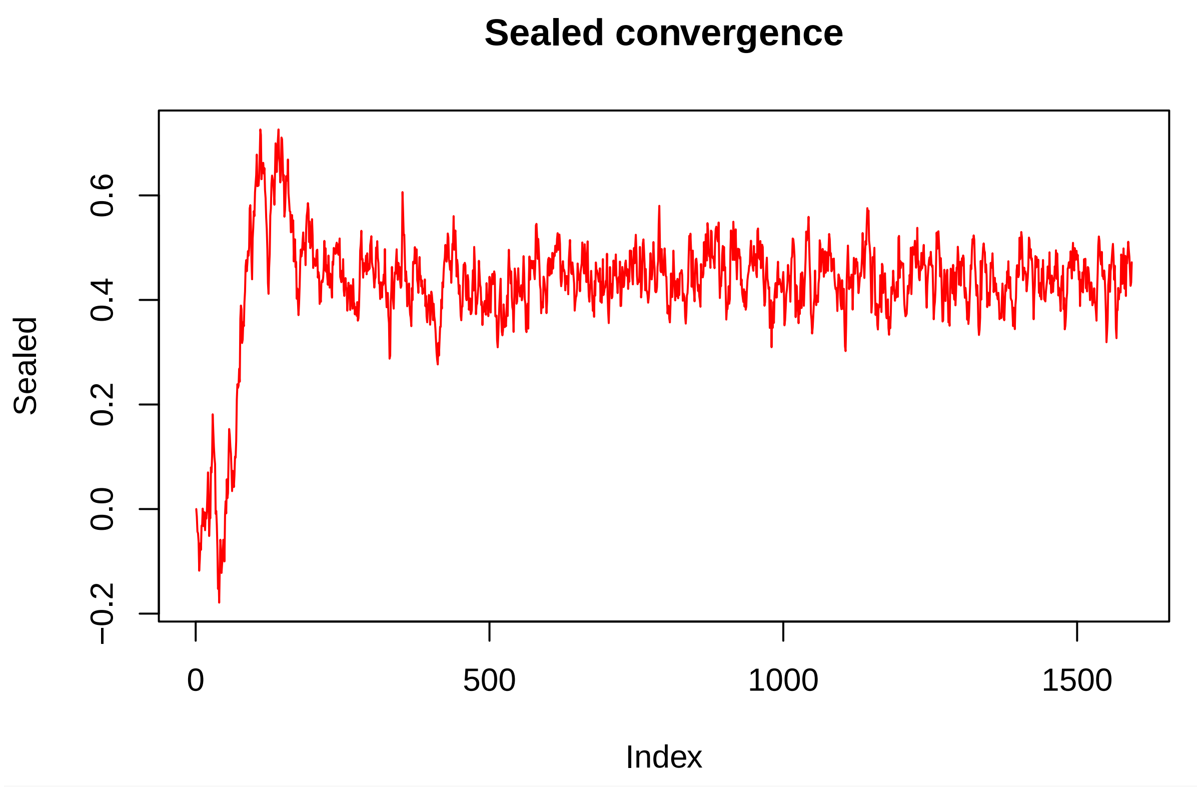
```
plot(pos_beta_mat[,3], type = "l",col="red",
     ylab = 'VerifyID', main = "VerifyID convergence")
```



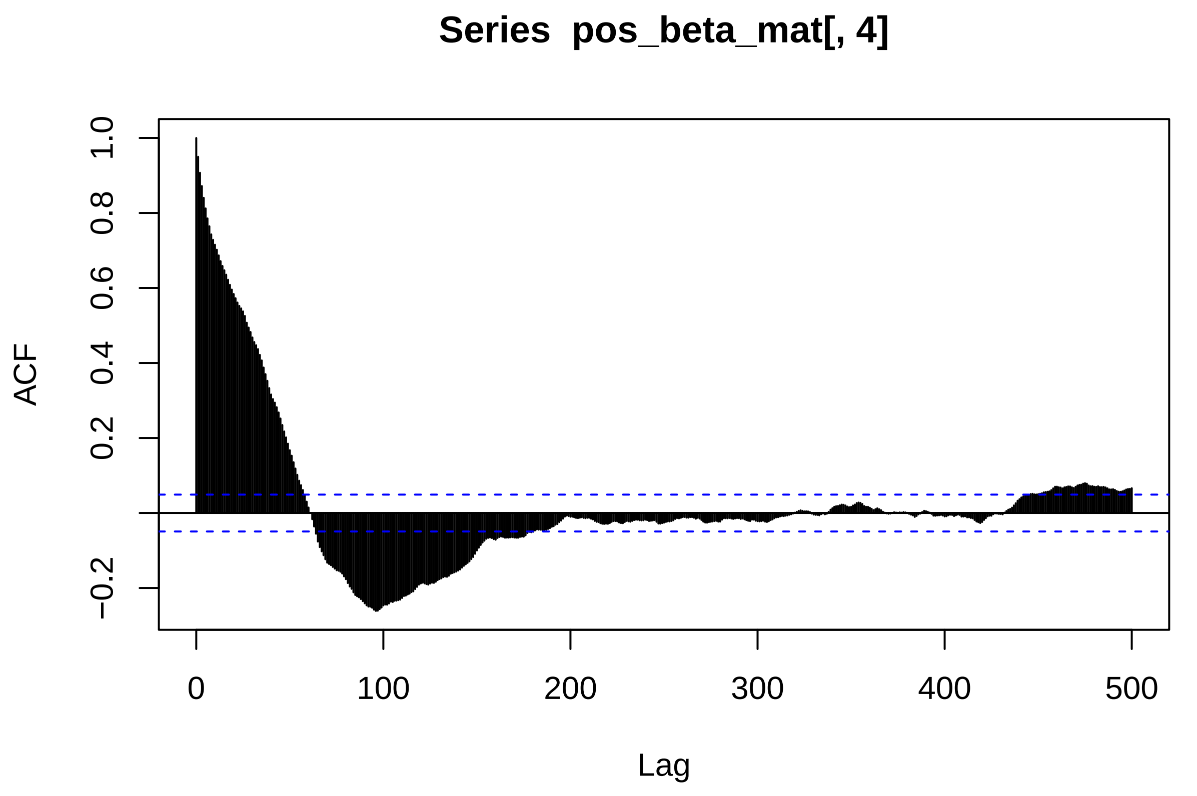
```r
acf(pos_beta_mat[,3], lag.max = 500)
```



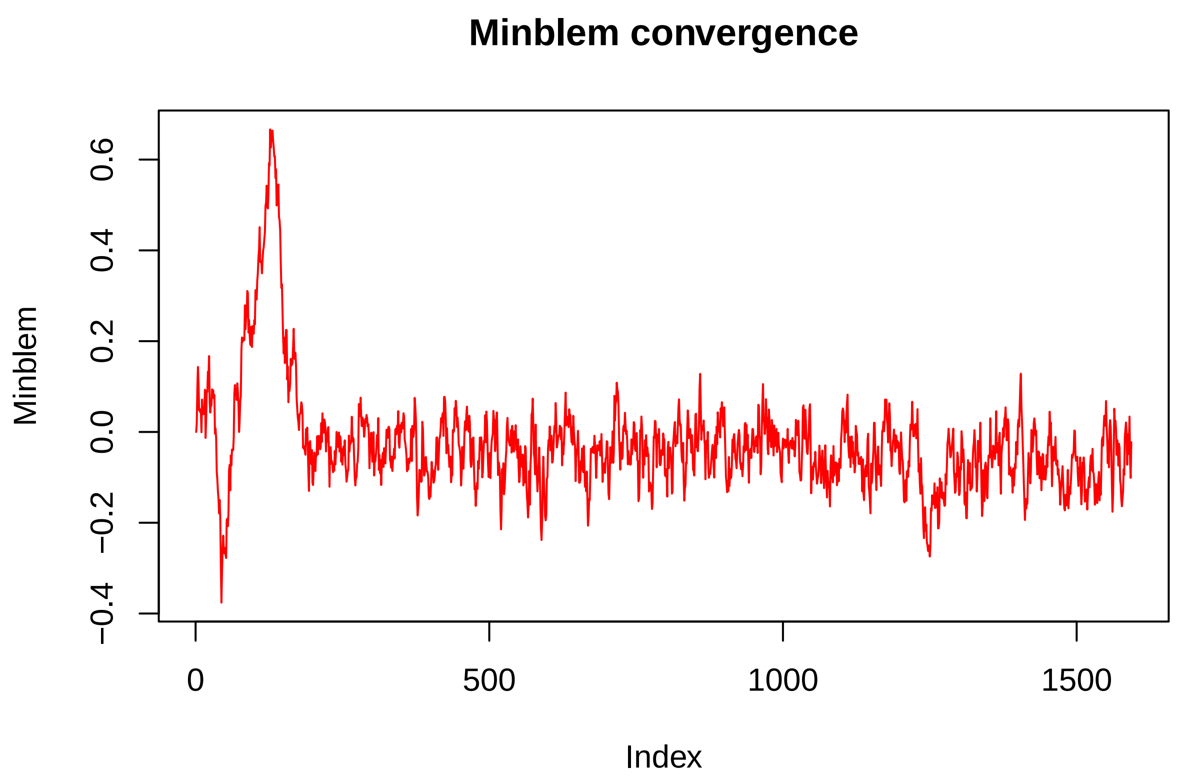
```r
plot(pos_beta_mat[,4], type = "l",col="red",
     ylab = 'Sealed', main = "Sealed convergence")
```



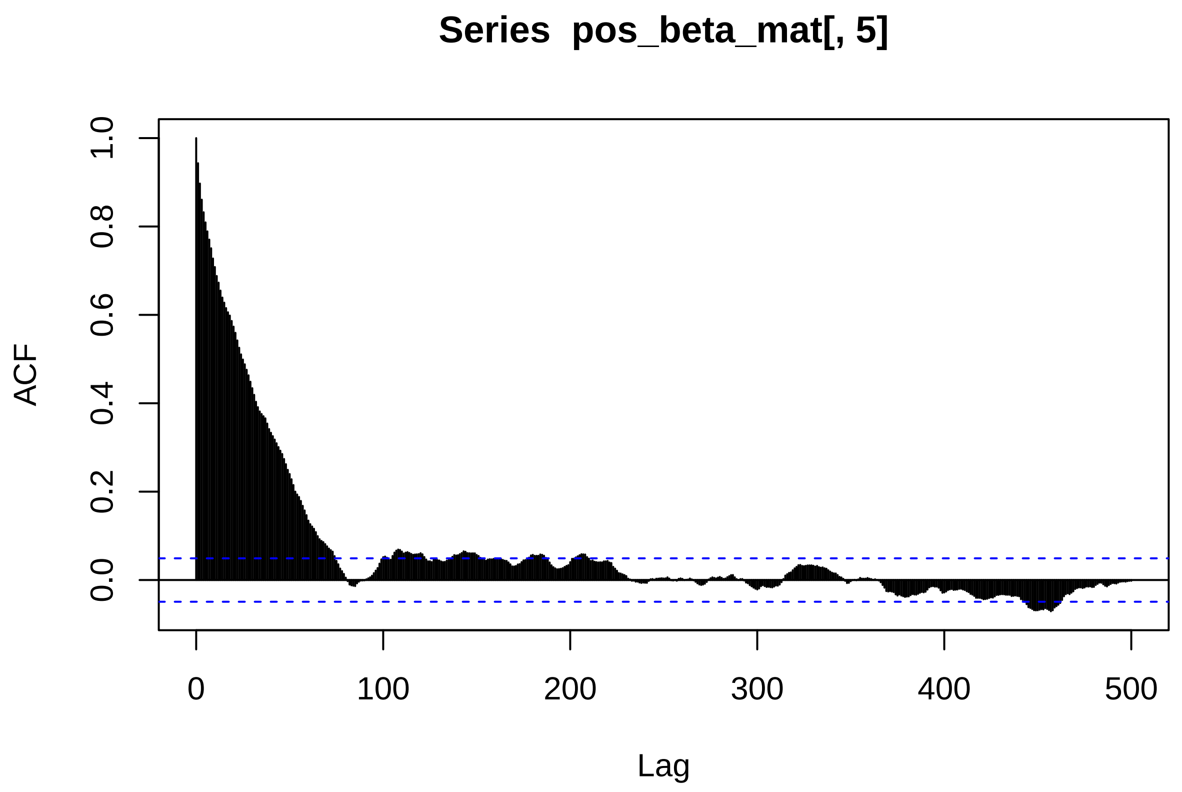
```
acf(pos_beta_mat[,4], lag.max = 500)
```



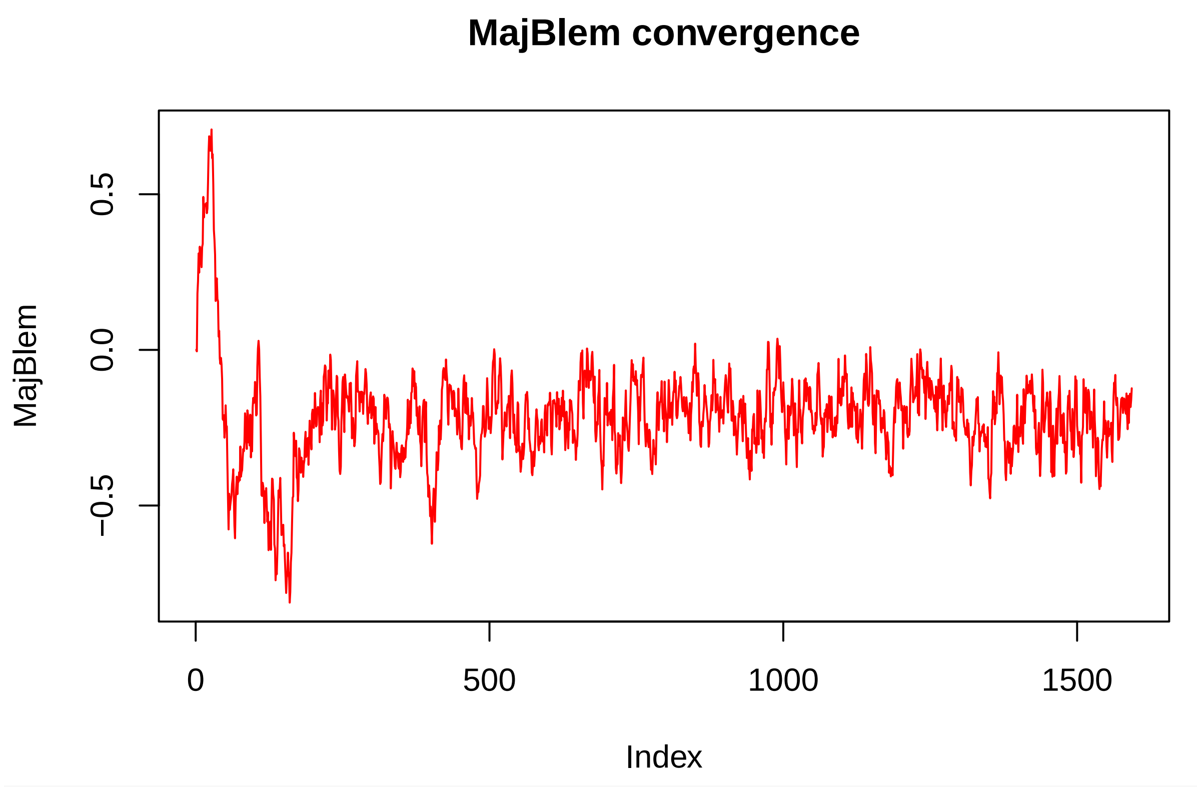
```
plot(pos_beta_mat[,5], type = "l",col="red",
     ylab = 'Minblem', main = "Minblem convergence")
```



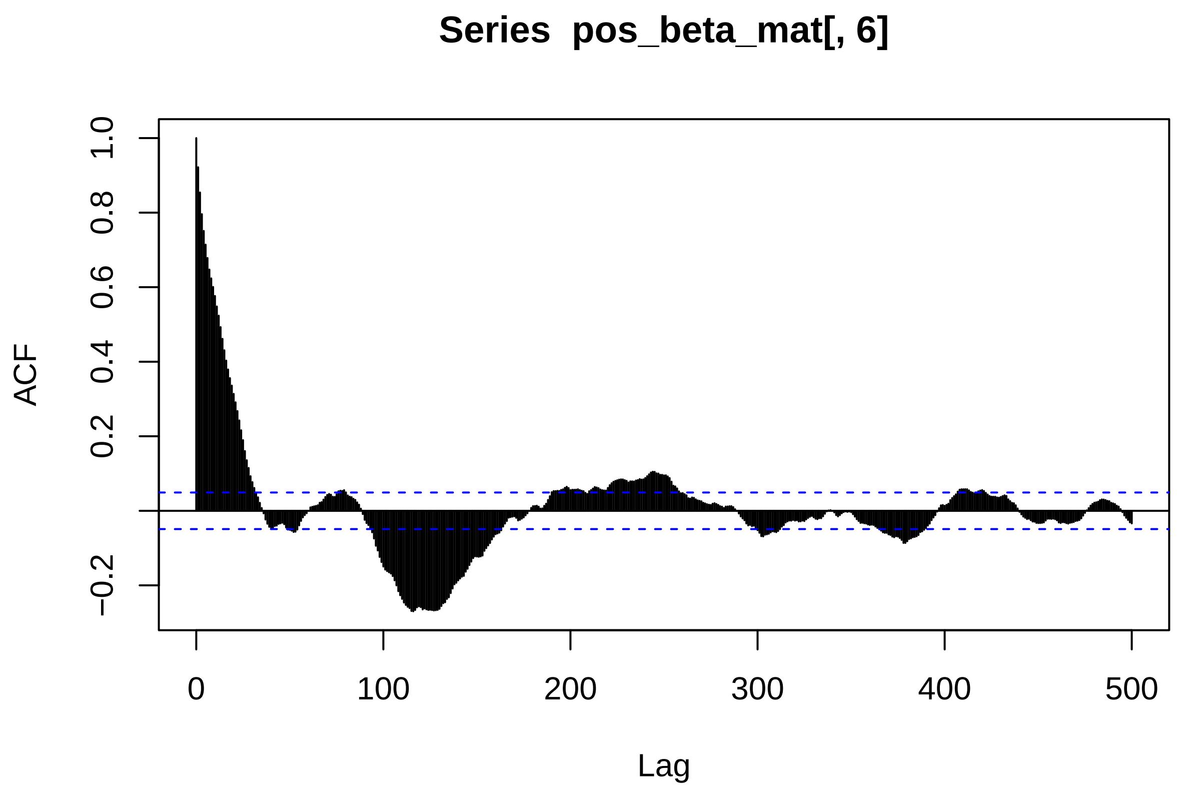
```
acf(pos_beta_mat[,5], lag.max = 500)
```



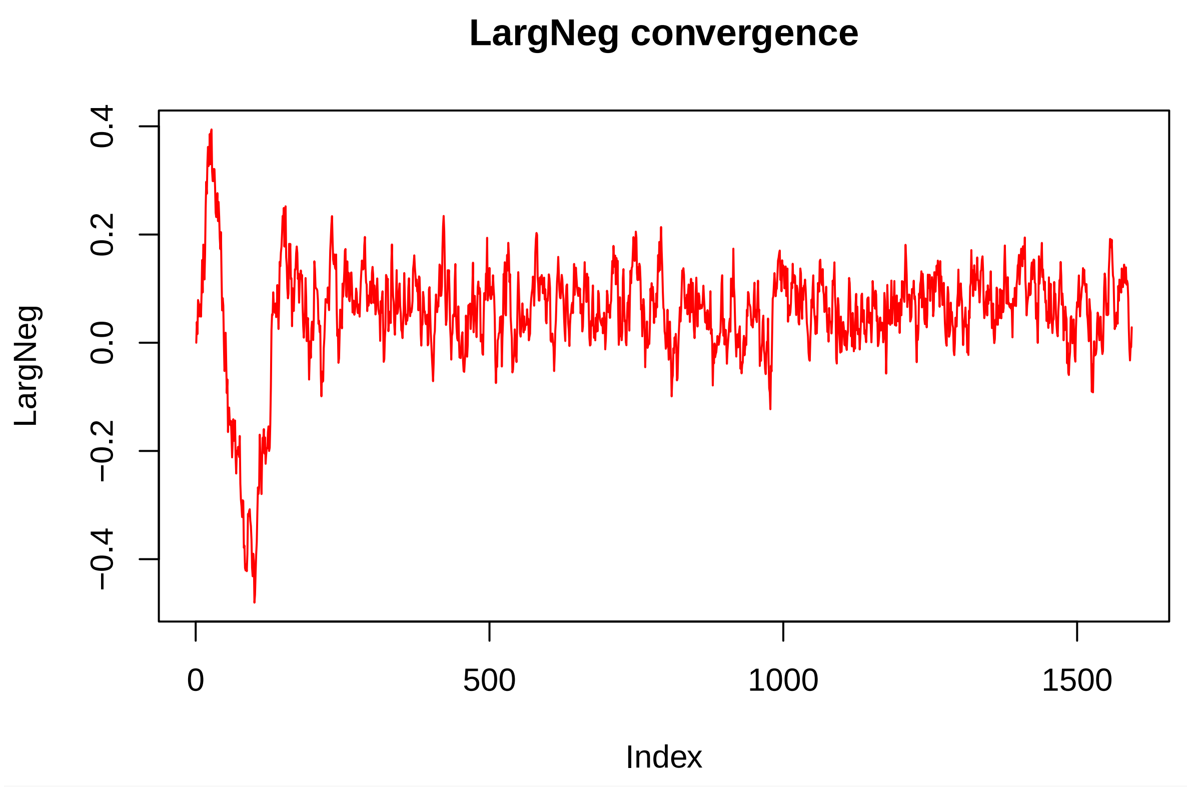
```
plot(pos_beta_mat[,6], type = "l",col="red",
     ylab = 'MajBlem', main = "MajBlem convergence")
```



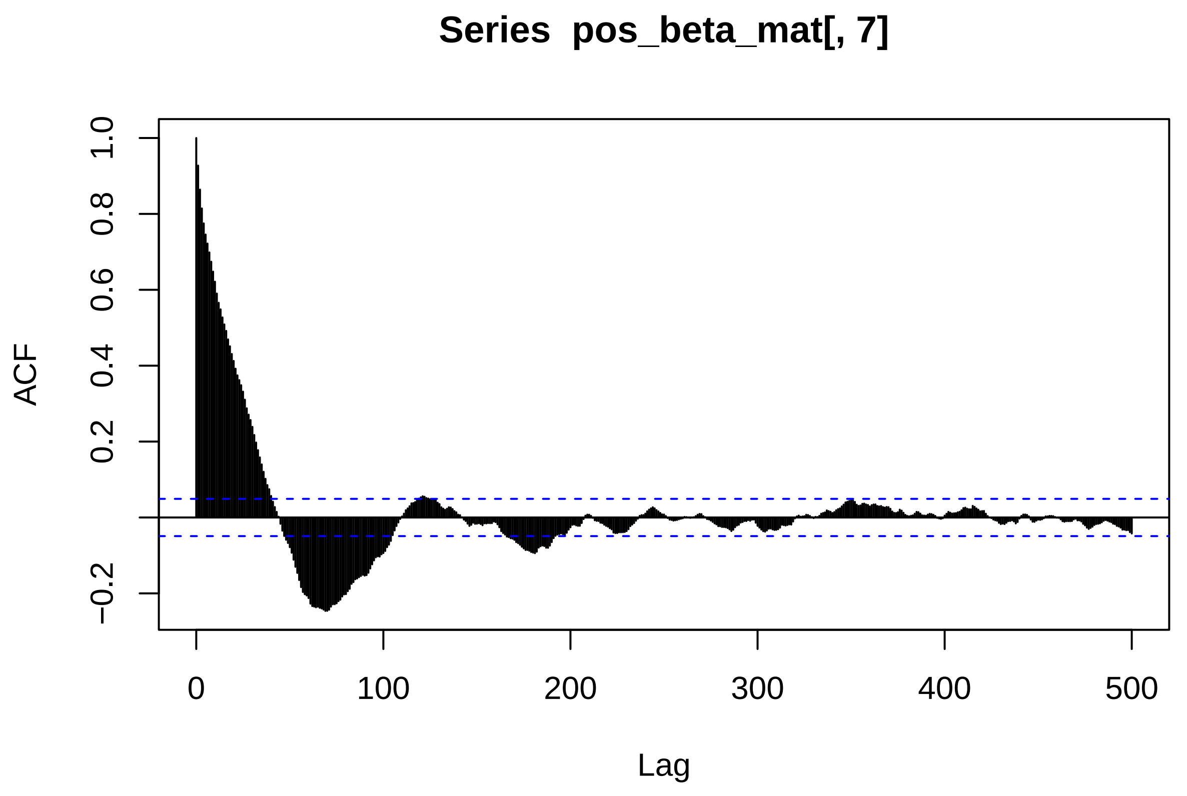
```
acf(pos_beta_mat[,6], lag.max = 500)
```



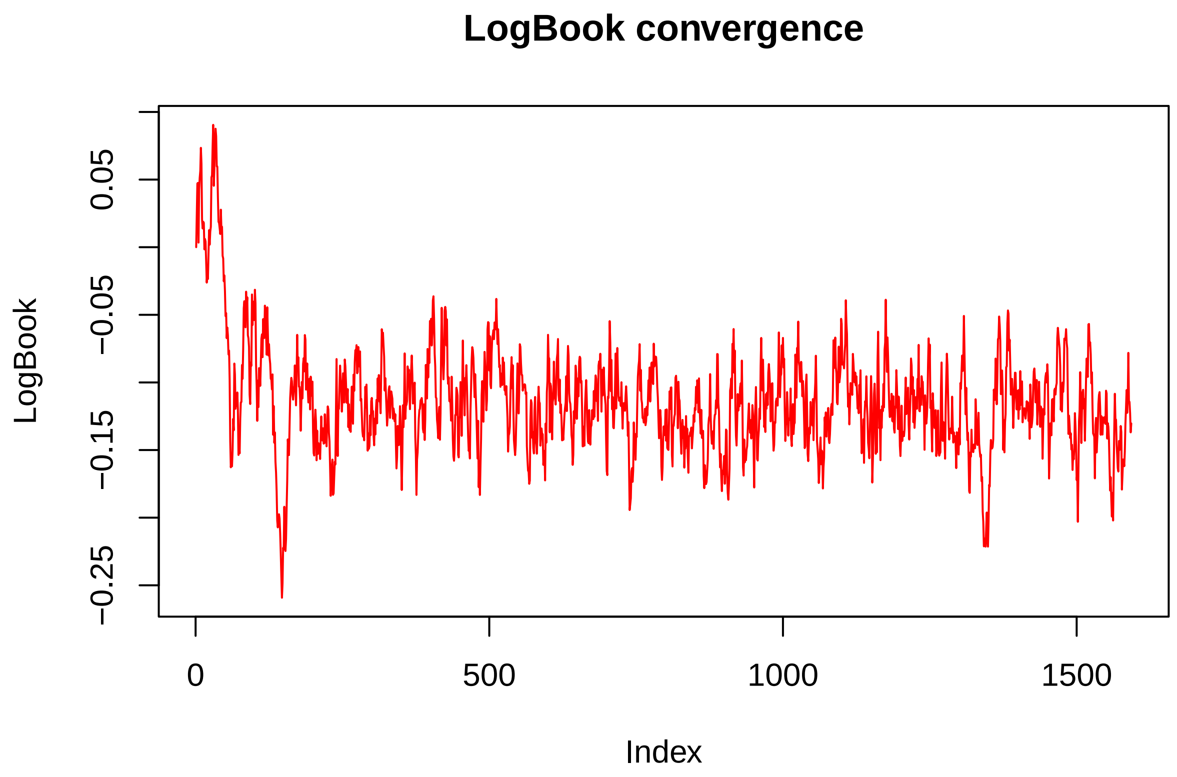
```
plot(pos_beta_mat[,7], type = "l",col="red",
     ylab = 'LargNeg', main = "LargNeg convergence")
```



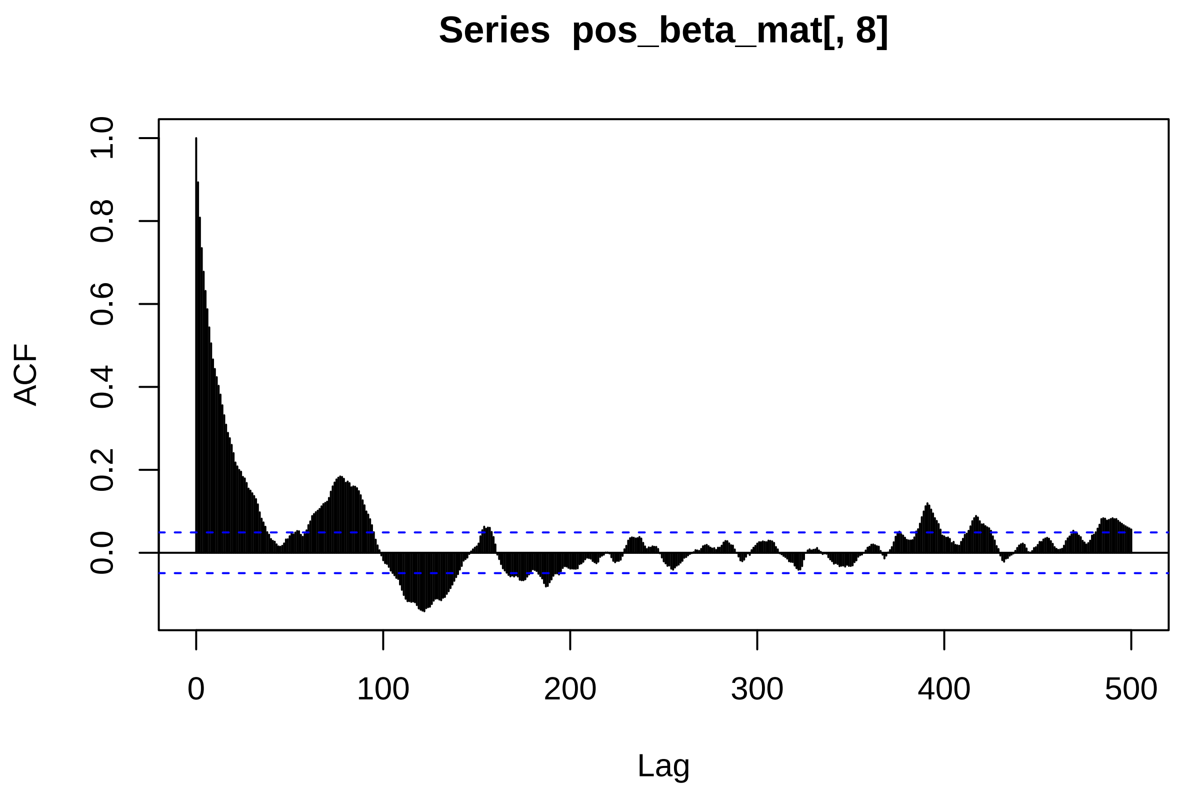
```
acf(pos_beta_mat[,7], lag.max = 500)
```



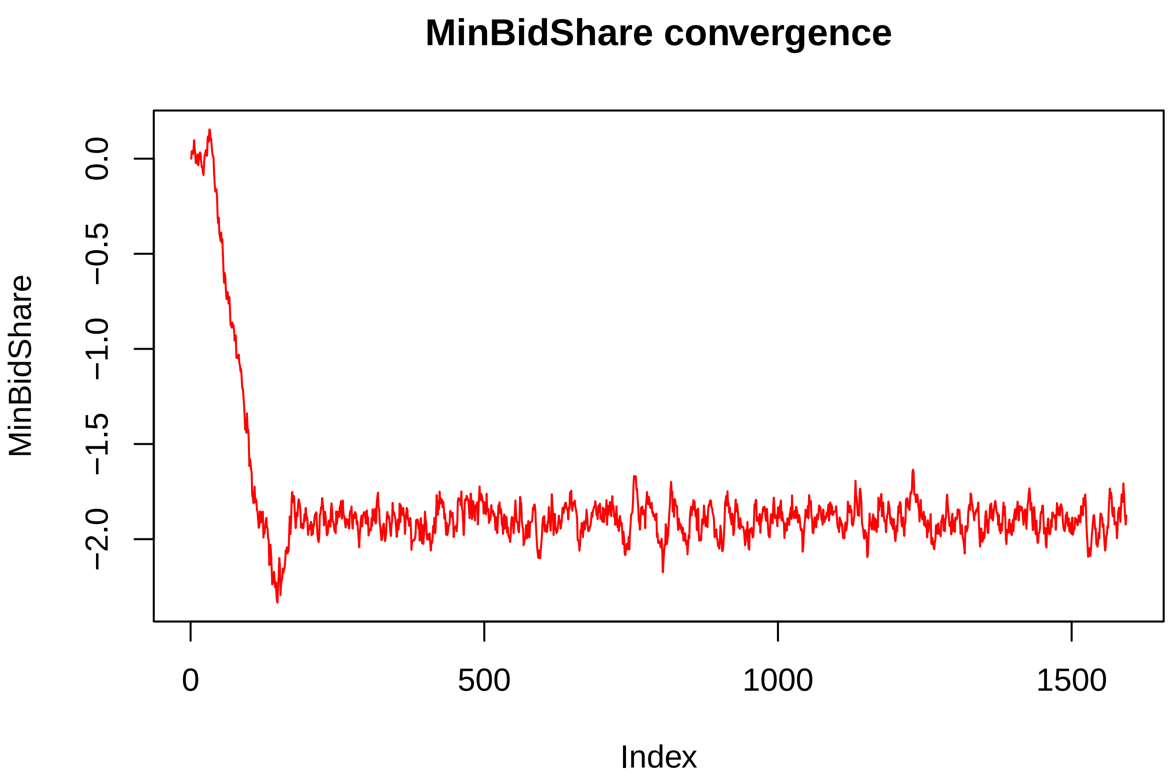
```
plot(pos_beta_mat[,8], type = "l",col="red",
     ylab = 'LogBook', main = "LogBook convergence")
```



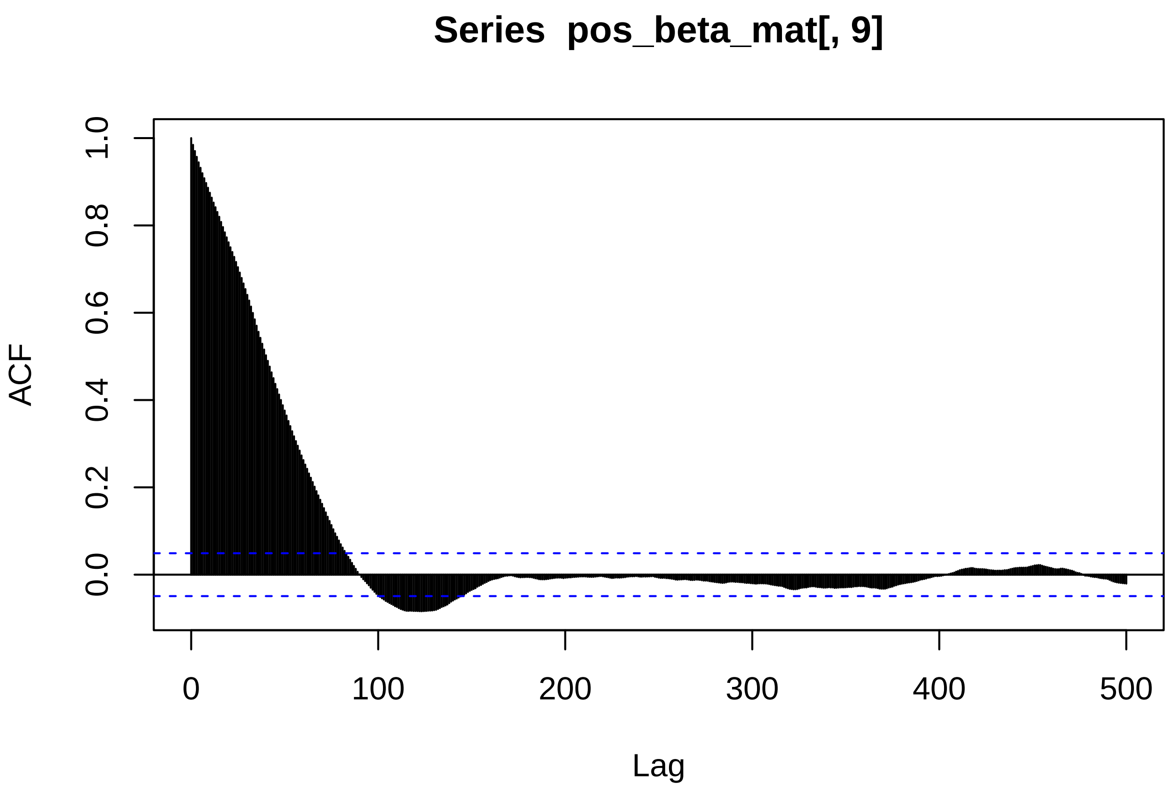
```
acf(pos_beta_mat[,8], lag.max = 500)
```



```
plot(pos_beta_mat[,9], type = "l",col="red",
     ylab = 'MinBidShare', main = "MinBidShare convergence")
```



```
acf(pos_beta_mat[,9], lag.max = 500)
```



```
apply(pos_beta_mat, 2, mean)
```

```
## [1]  1.00233503 -0.01418931 -0.36063793  0.42881075 -0.02576741 -0.21603606
## [7]  0.05591239 -0.11401252 -1.81350996
```

2 d).

```r
# Question d: Simulate predictive distribution using MCMC draws of betas
#Given: new X, intercept added as 1
NewX <- matrix(c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8),
               nrow = 1, ncol = 9)

temp <- exp(pos_beta_mat %*% t(NewX))

# Posterior Predictions
y_pred <- apply(temp, 1, rpois, n = 1)
print('Probability of no bidders in new auction:')
```

```
## [1] "Probability of no bidders in new auction:"
```

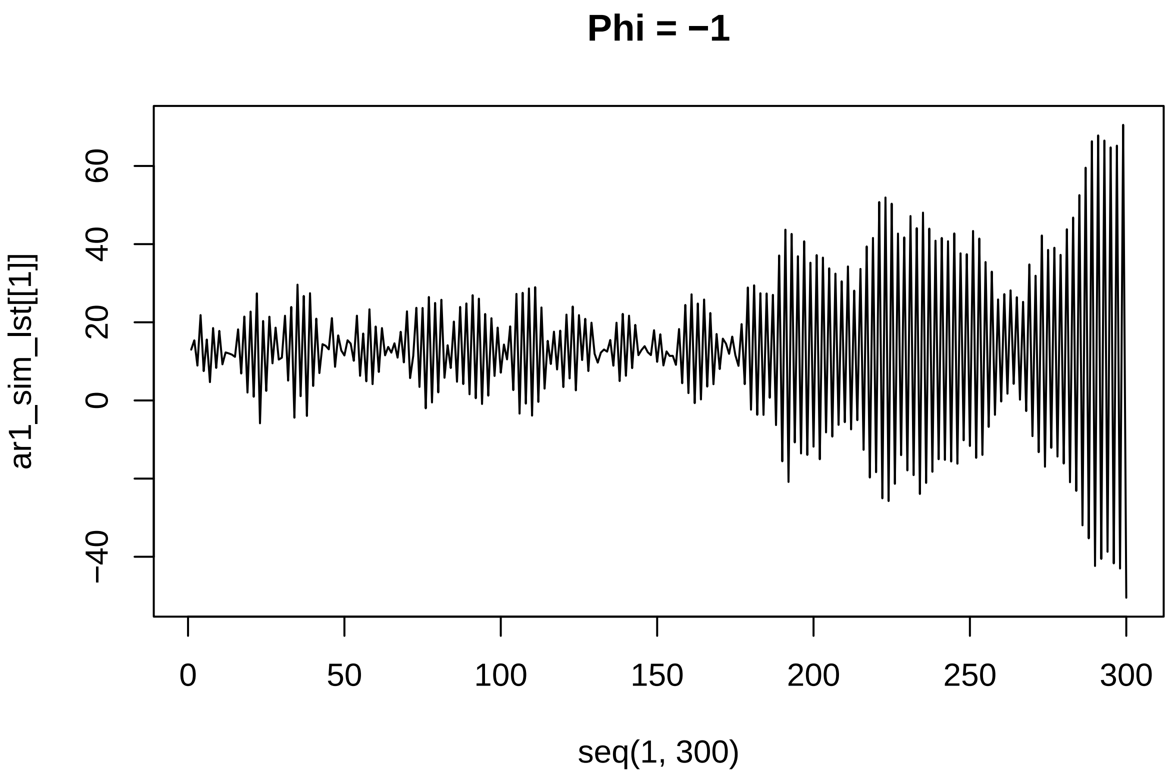
```r
total_bids <-  nrow(pos_beta_mat)
sum(y_pred == 0)/total_bids
```

```
## [1] 0.5015694
```

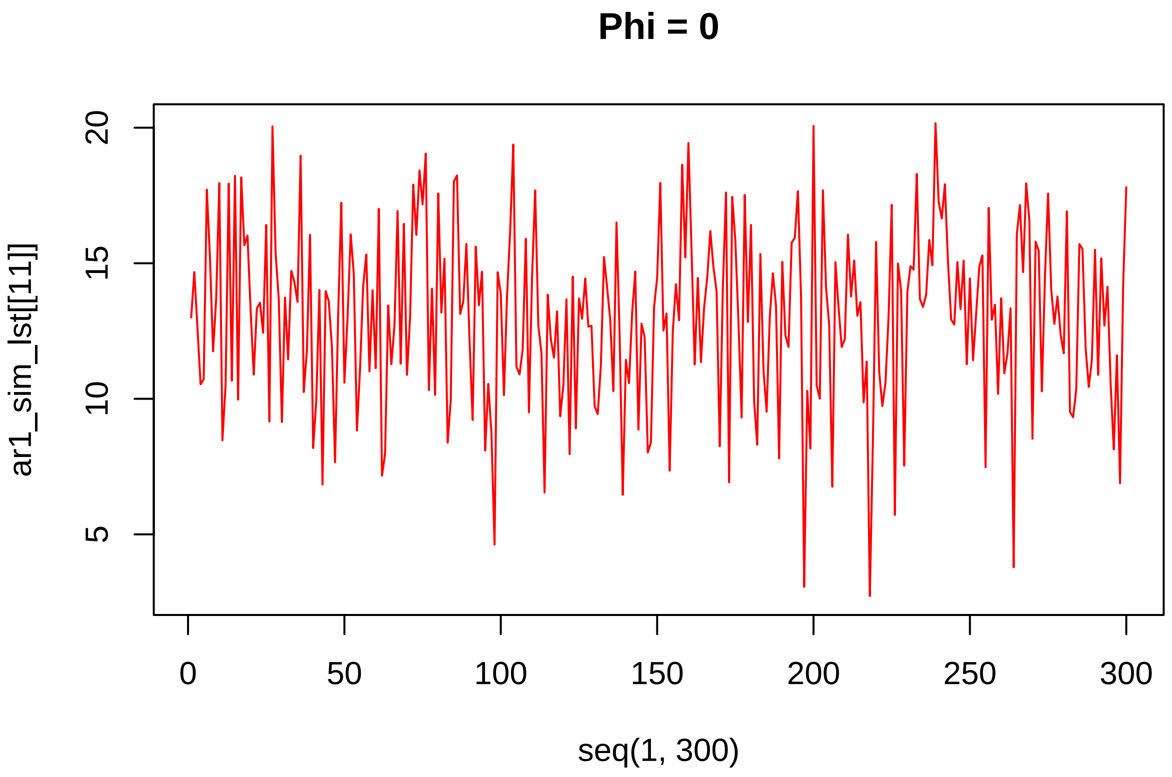
Question 3

3 a).

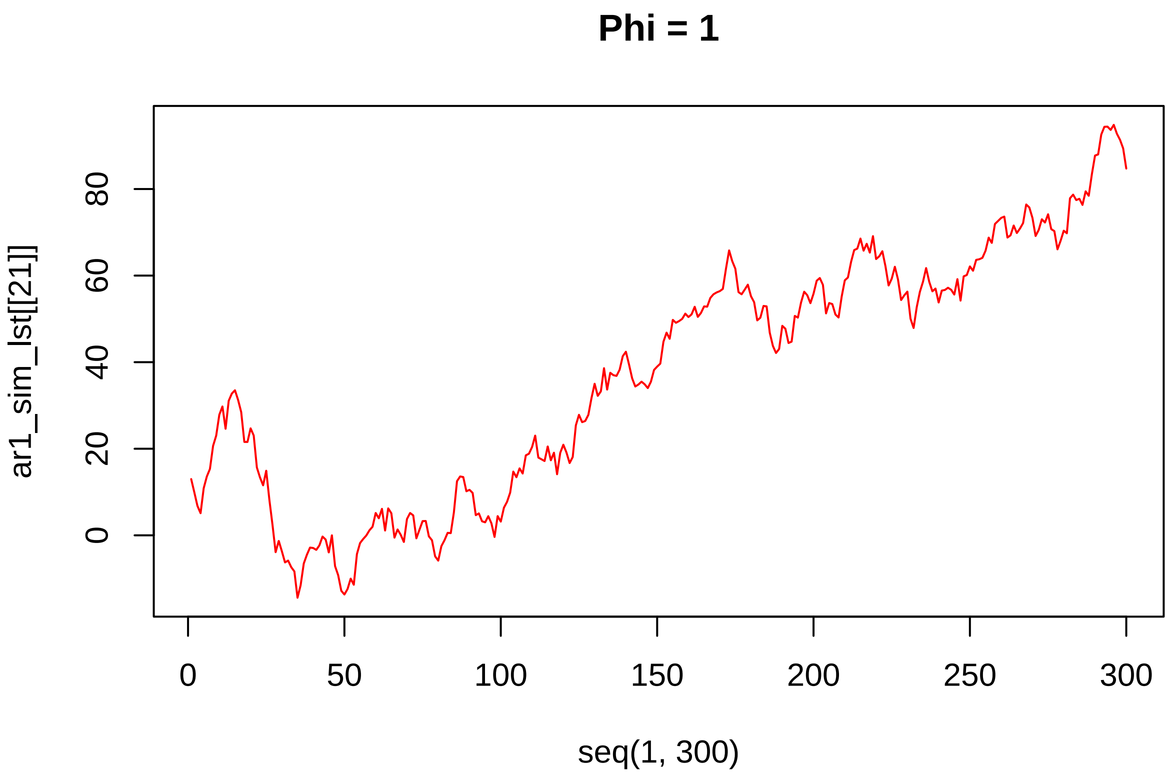
```r
####Question 3: Time series model in Stan####
# Question a: Simulate from AR(1)
ar1_sim <- function(phi){
  T <- 300
  mu <- 13
  err_sigma_sq <- 3

  x <- vector(length = T)
  x[1] <- mu
  for (i in 2:T) {
    # The AR(1) Model:
    x[i] <- mu + (phi * (x[i-1] - mu)) +
      rnorm(n = 1, mean = 0, sd = err_sigma_sq)
  }
  return(matrix(data = x, nrow = 1))
}

phi_values <- seq(-1, 1, 0.1)
ar1_sim_lst <- lapply(phi_values, function(phi){
  ar1_sim(phi)
})
#par(mfrow=c(2,1))
plot(seq(1,300),ar1_sim_lst[[1]], type = "l", main = "Phi = -1")
```



```
plot(seq(1,300),ar1_sim_lst[[11]], type = "l", col = "red", main = "Phi = 0")
```



```
plot(seq(1,300),ar1_sim_lst[[21]], type = "l", col = "red", main = "Phi = 1")
```



As seen above, when phi is -1 and 1, the current draw is heavily correlated to the previous draws, however, when phi is 0, the current value is not influenced/ correlated to the previous value.

3 b).

```
# Question b: Use STAN to sample from posterior
library(rstan)
# Data
phi_values <- c(0.2,0.95)
ar1_sim_lst <- lapply(phi_values, function(phi){
  ar1_sim(phi)
})
library(abind)#To combine a list of matrix into a single matrix
ar1_sim_lmat <- abind(ar1_sim_lst, along = 1)
N <- dim(ar1_sim_lmat)[2]


# Model
StanModel <- '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  //Flat priors are non-informative and need not be defined
  for (n in 2:N)
    y[n] ~ normal(mu + phi * (y[n-1]-mu), sigma);//AR(1) process simulated above
}
'
```

```
print("The following is for phi=0.2")
```

```
## [1] "The following is for phi=0.2"
```

```
data <- list(N=N, y=ar1_sim_lmat[1,])#For phi=0.2
#data <- list(N=N, y=ar1_sim_lmat[2,])#For phi=0.95
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, verbose = FALSE)
```

```
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.91 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.227725 seconds (Warm-up)
## Chain 1:                0.270178 seconds (Sampling)
## Chain 1:                0.497903 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5.7e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.57 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.2406 seconds (Warm-up)
## Chain 2:                0.192239 seconds (Sampling)
## Chain 2:                0.432839 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.53 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.231473 seconds (Warm-up)
## Chain 3:                0.174565 seconds (Sampling)
## Chain 3:                0.406038 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.9e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.49 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.198725 seconds (Warm-up)
## Chain 4:                0.175073 seconds (Sampling)
## Chain 4:                0.373798 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
#print(fit,digits_summary=3)
print(paste('Posterior mean of mu:',
            summary(fit)$summary[1, "mean"]))
```

```
## [1] "Posterior mean of mu: 13.0611365289355"
```

```
print(paste('Posterior mean of phi:',
            summary(fit)$summary[2, "mean"]))
```

```
## [1] "Posterior mean of phi: 0.199656162420391"
```

```
print(paste('Posterior mean of sigma-sq:',
            summary(fit)$summary[3, "mean"]))
```

```
## [1] "Posterior mean of sigma-sq: 3.1693502896783"
```

As seen above, the mean values of the parameters matches with the true data, i.e. $\mu = 13, \phi = 0.2, \sigma^2 = 3$

```
print(paste('95% CI of mu:',
            summary(fit)$summary[1, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of mu: 12.5962920373031" "95% CI of mu: 13.5172798839802"
```

```
print(paste('95% CI of phi:',
            summary(fit)$summary[2, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of phi: 0.0841743649248458" "95% CI of phi: 0.313329505597738"
```

```
print(paste('95% CI of sigma-sq:',
            summary(fit)$summary[3, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of sigma-sq: 2.91675731177918"
## [2] "95% CI of sigma-sq: 3.43480256478067"
```

As shown below, an Rhat value of 1 signifies the convergence of the samplers. The same is confirmed by the trace plots which converges to a stationary distribution.

```
print(paste('Rhat value for mu:',
            summary(fit)$summary[1, "Rhat"]))
```

```
## [1] "Rhat value for mu: 1.00051807206082"
```

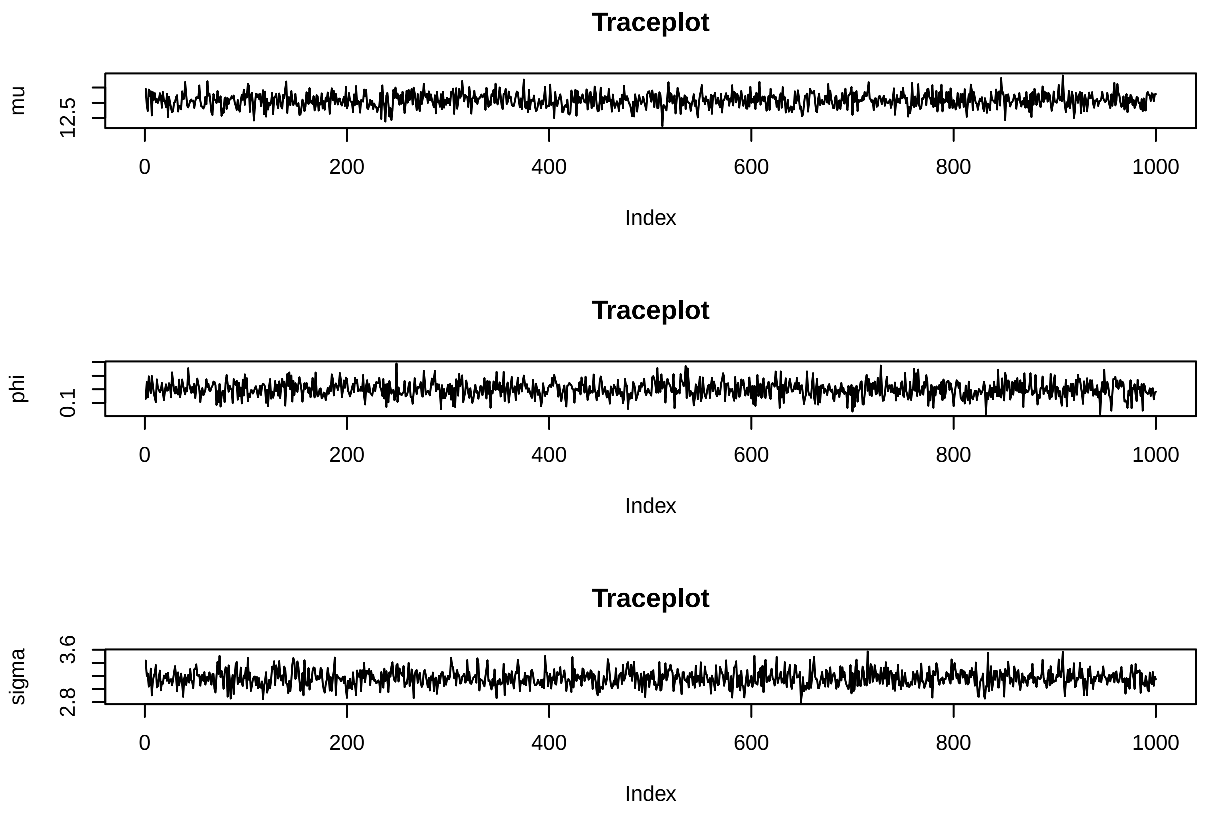
```
print(paste('Rhat value for phi:',
            summary(fit)$summary[2, "Rhat"]))
```

```
## [1] "Rhat value for phi: 0.999728477900266"
```

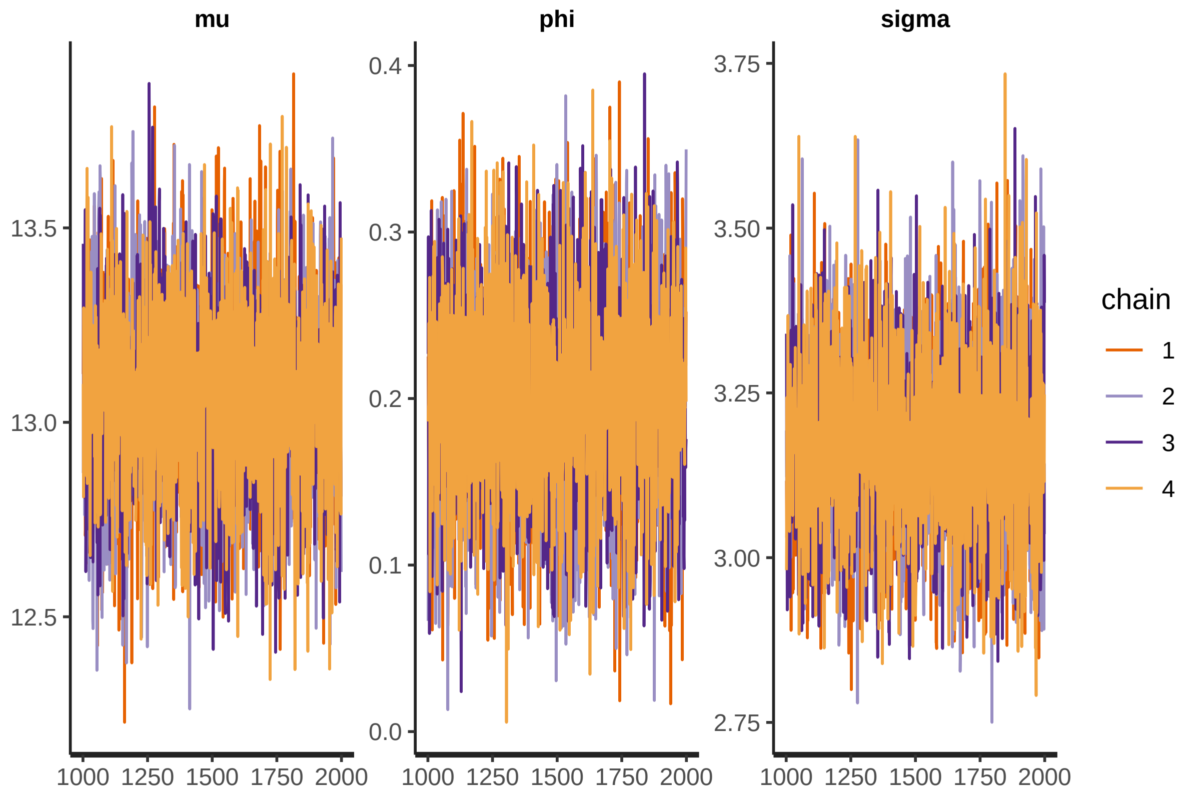
```
print(paste('Rhat value for sigma-sq:',
            summary(fit)$summary[3, "Rhat"]))
```

```
## [1] "Rhat value for sigma-sq: 0.999421008468595"
```

```
# Extract posterior samples
postDraws <- extract(fit)
# Do traceplots of the first chain
par(mfrow = c(3,1))
plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
plot(postDraws$phi[1:(niter-warmup)],type="l",ylab="phi",main="Traceplot")
plot(postDraws$sigma[1:(niter-warmup)],type="l",ylab="sigma",main="Traceplot")
```



```
# Do automatic traceplots of all chains
traceplot(fit)
```



```
# Bivariate posterior plots
#pairs(fit)
```

```
print("The following is for phi=0.95")
```

```
## [1] "The following is for phi=0.95"
```

```
# data <- list(N=N, y=ar1_sim_lmat[1,])#For phi=0.2
data <- list(N=N, y=ar1_sim_lmat[2,])#For phi=0.95
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, verbose = FALSE)
```

```
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.48 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.28204 seconds (Warm-up)
## Chain 1:                0.323496 seconds (Sampling)
## Chain 1:                1.60553 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.41 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.898761 seconds (Warm-up)
## Chain 2:                0.650129 seconds (Sampling)
## Chain 2:                1.54889 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.43 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
```

```
## Chain 3:  Elapsed Time: 1.31591 seconds (Warm-up)
## Chain 3:                0.270463 seconds (Sampling)
## Chain 3:                1.58638 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'f62a875dcafbdefa26c267b7becd02a9' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.4 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.74177 seconds (Warm-up)
## Chain 4:                0.645564 seconds (Sampling)
## Chain 4:                3.38734 seconds (Total)
## Chain 4:

## Warning: There were 489 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.07, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```r
# Print the fitted model
#print(fit,digits_summary=3)
print(paste('Posterior mean of mu:',
            summary(fit)$summary[1, "mean"]))
```

```
## [1] "Posterior mean of mu: -11.7464770043454"
```

```r
print(paste('Posterior mean of phi:',
            summary(fit)$summary[2, "mean"]))
```

```
## [1] "Posterior mean of phi: 0.993658304508059"
```

```
print(paste('Posterior mean of sigma-sq:',
            summary(fit)$summary[3, "mean"]))
```

```
## [1] "Posterior mean of sigma-sq: 3.10289099323941"
```

As seen above, the mean values of the parameters matches with the true data, i.e. $\mu = 13, \phi = 0.2, \sigma^2 = 3$

```
print(paste('95% CI of mu:',
            summary(fit)$summary[1, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of mu: -283.07255367996" "95% CI of mu: 257.926904090753"
```

```
print(paste('95% CI of phi:',
            summary(fit)$summary[2, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of phi: 0.970950666904057" "95% CI of phi: 1.00577306049968"
```

```
print(paste('95% CI of sigma-sq:',
            summary(fit)$summary[3, c("2.5%", "97.5%")]))
```

```
## [1] "95% CI of sigma-sq: 2.86305801978377"
## [2] "95% CI of sigma-sq: 3.37007591292082"
```

As shown below, an Rhat value of 1 signifies the convergence of the samplers. The same is confirmed by the trace plots which converges to a stationary distribution.

```
print(paste('Rhat value for mu:',
            summary(fit)$summary[1, "Rhat"]))
```

```
## [1] "Rhat value for mu: 1.07597616101502"
```

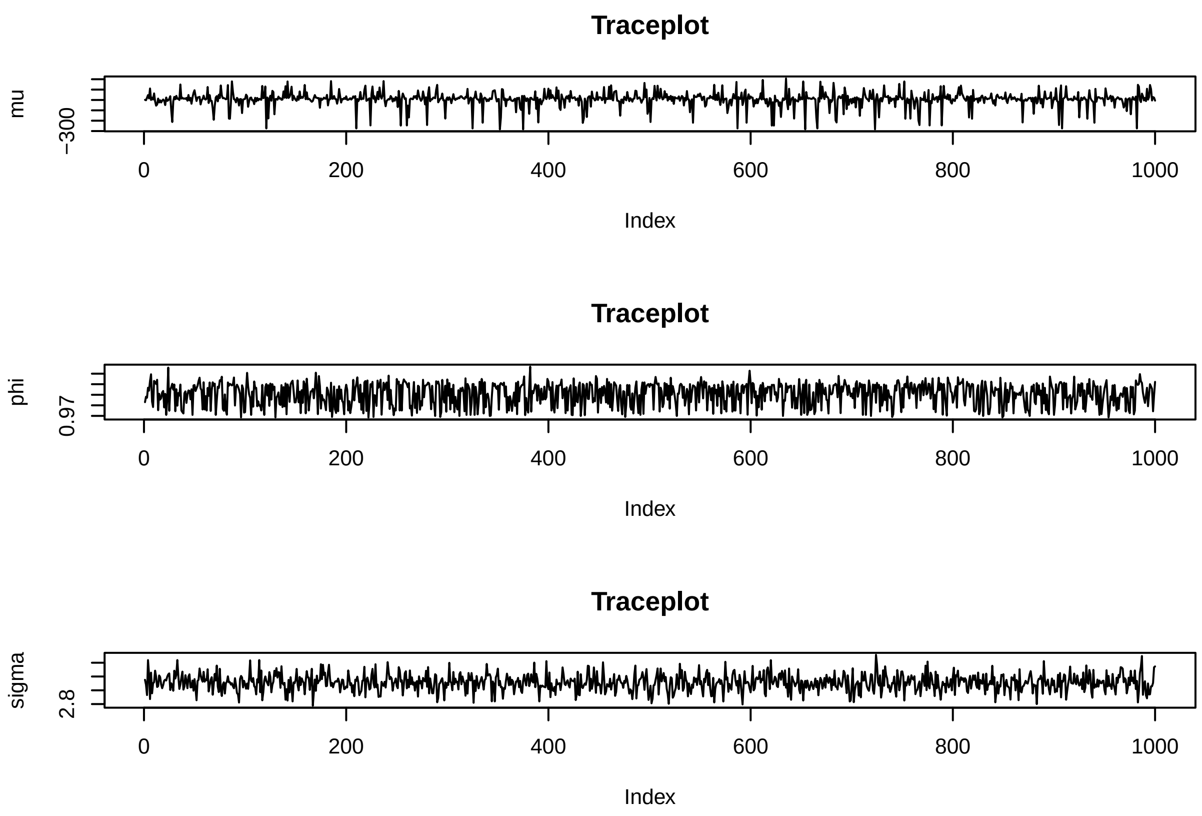
```
print(paste('Rhat value for phi:',
            summary(fit)$summary[2, "Rhat"]))
```

```
## [1] "Rhat value for phi: 1.05419090375798"
```

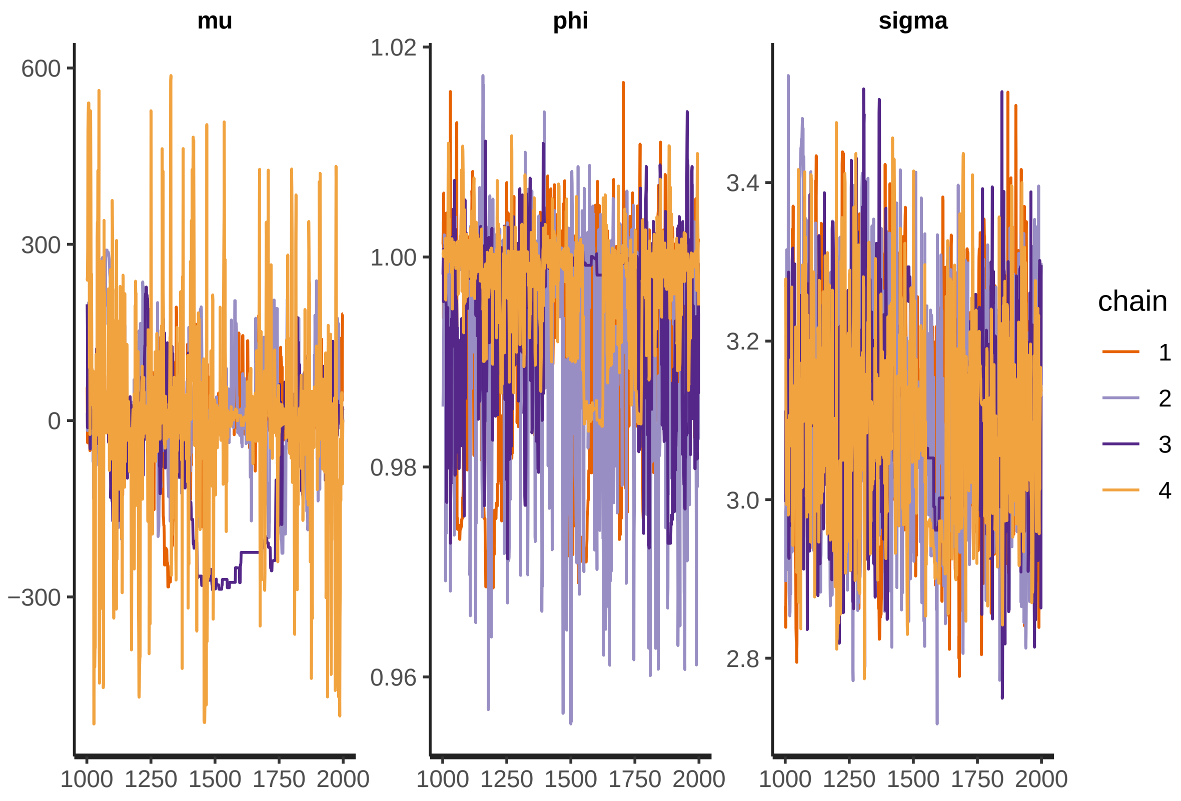
```
print(paste('Rhat value for sigma-sq:',
            summary(fit)$summary[3, "Rhat"]))
```

```
## [1] "Rhat value for sigma-sq: 1.00849874456931"
```

```
# Extract posterior samples
postDraws <- extract(fit)
# Do traceplots of the first chain
par(mfrow = c(3,1))
plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
plot(postDraws$phi[1:(niter-warmup)],type="l",ylab="phi",main="Traceplot")
plot(postDraws$sigma[1:(niter-warmup)],type="l",ylab="sigma",main="Traceplot")
```



```r
# Do automatic traceplots of all chains
traceplot(fit)
```

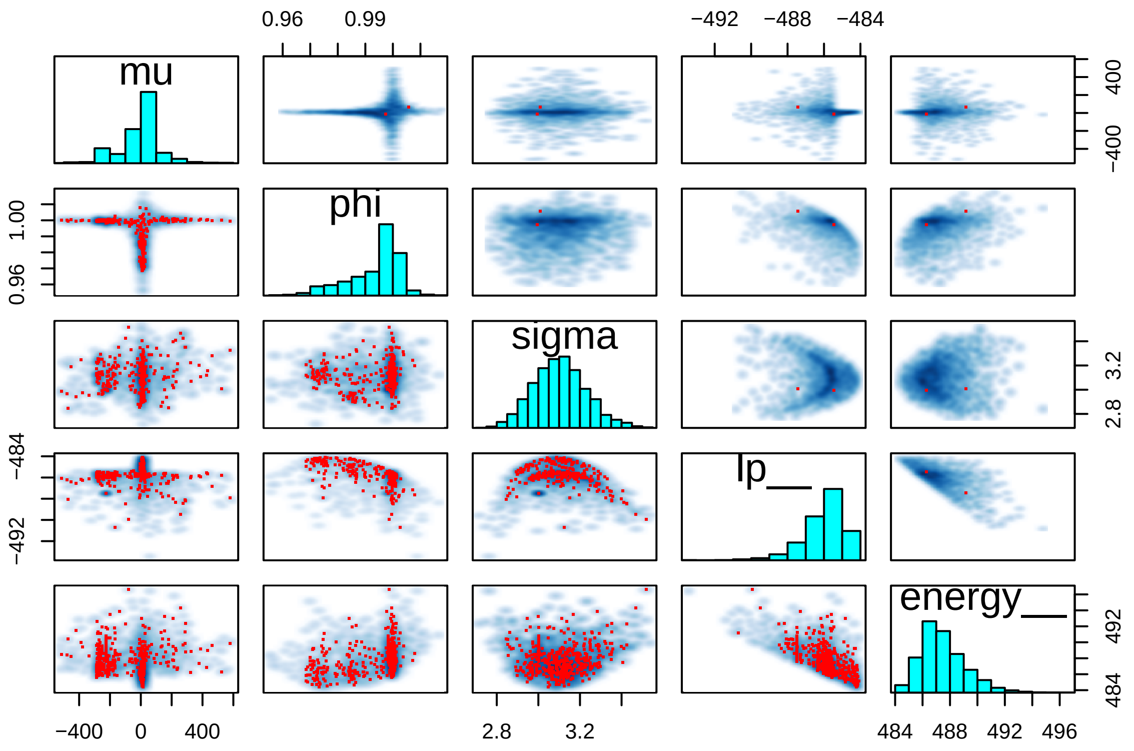


As the correlation has increased, i.e. phi value increased, we observe that the estimation of the posterior mean of the parameters are relatively bad. Also, the 95% confidence intervals has relatively more concentration around the mean when phi is 0.2, i.e. when there is less correlation.

The join posterior of μ and ϕ is given below:

```
pairs(fit)
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```


mu
phi
sigma
lp__
energy__