Lab3 Report

aswma317,varsi146

2023-05-05

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
```

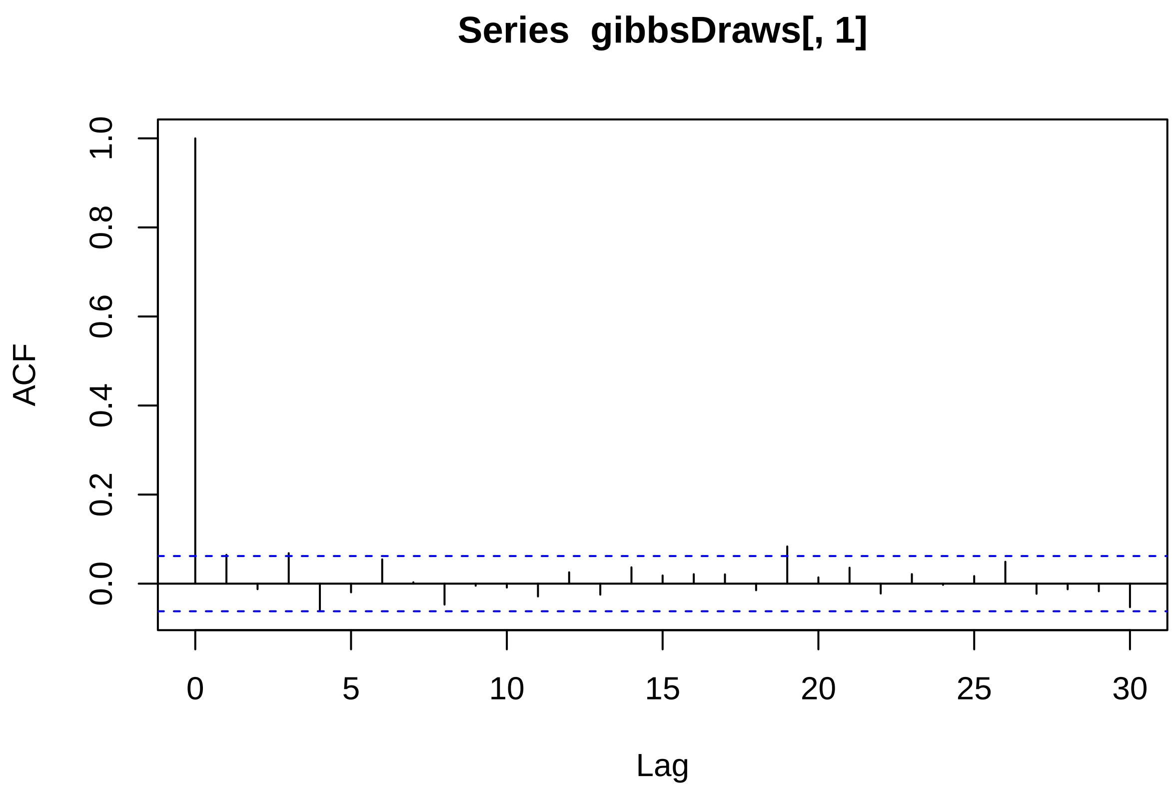
```r
  #Get the mu_n(mean) and tau_n_sq(var) for pos_mu which is N(mu_n, tau_n_sq)
  # Part of Full Conditional posterior for mean in Gibbs Sampling
  # As part of derivation for Normal Model - Known variance - Normal Prior
  mu_n <- mean(ln_data) * (pr_tau0_sq/(pr_tau0_sq + (pos_sigma_sq/n))) +
    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"
```

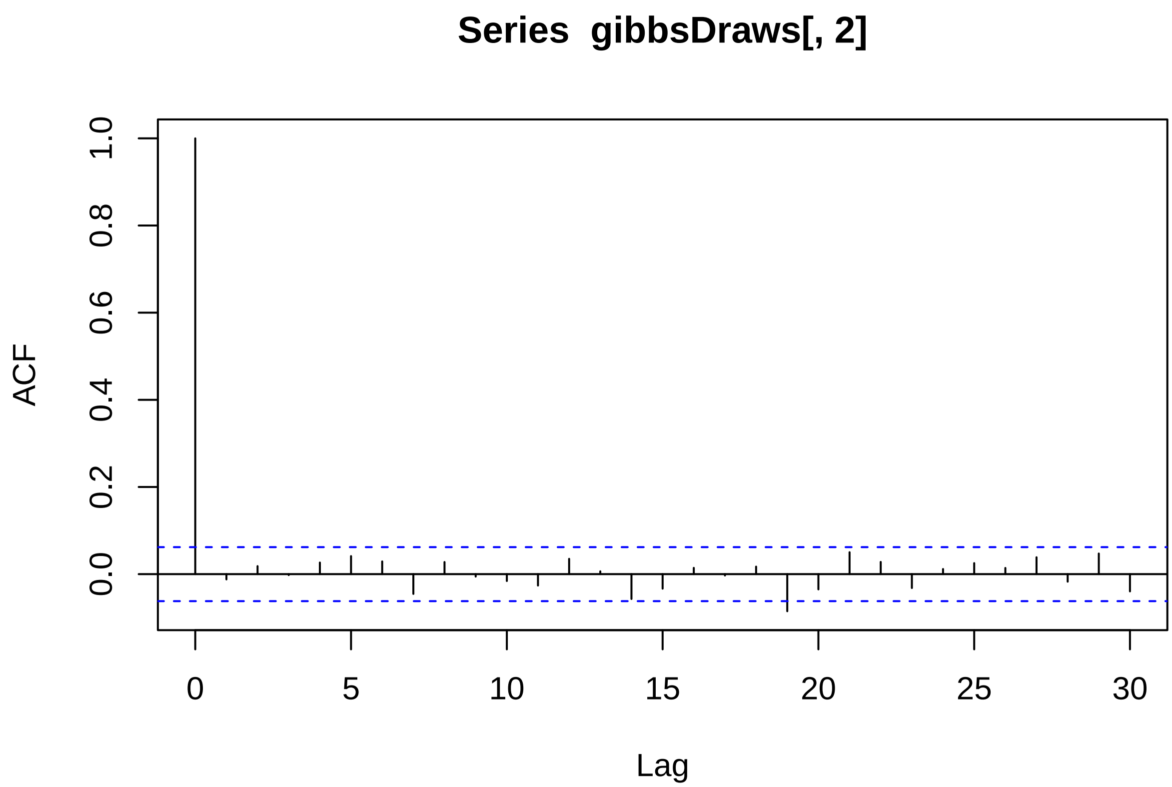
```r
print(gibbs_posterior)
```

```
## [1] 1.307682 1.748991
```

```r
#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])
```



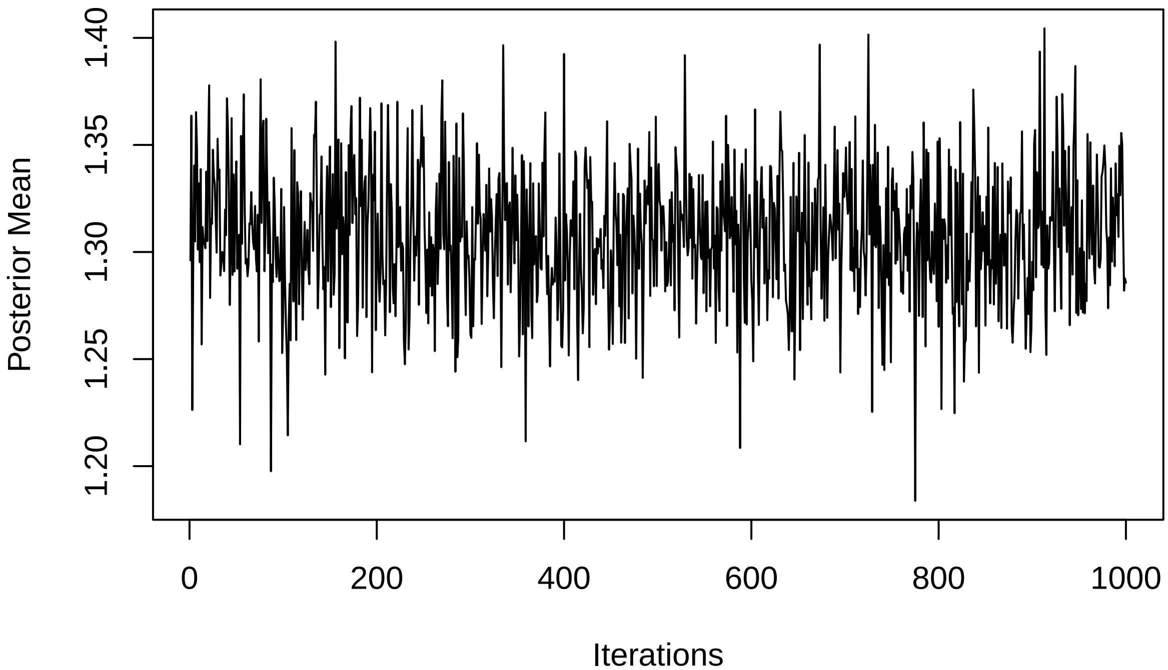

```
IF_mu_Gibbs <- 1+2*sum(a_mu_Gibbs$acf[-1])
IF_var_Gibbs <- 1+2*sum(a_var_Gibbs$acf[-1])

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: 1.37"
```

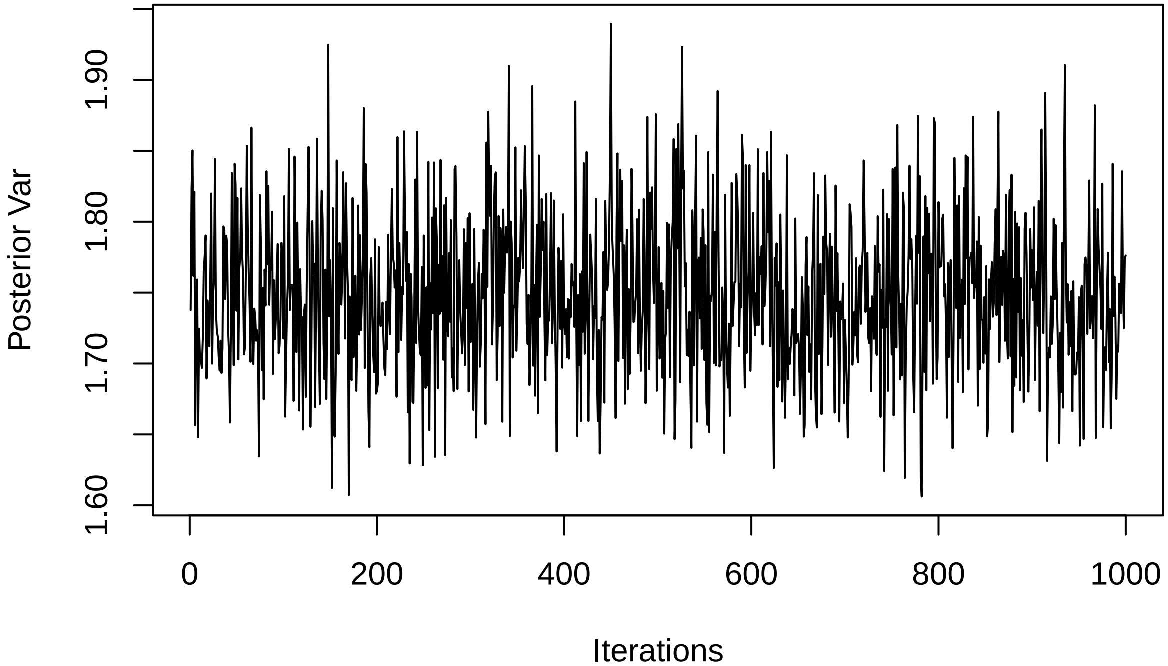
```
print(paste('The inefficiency factor of Gibbs draws with respect to posterior sigma-squared is:',
            round(IF_var_Gibbs,2)))
```

```
## [1] "The inefficiency factor of Gibbs draws with respect to posterior sigma-squared is: 1.05"
```

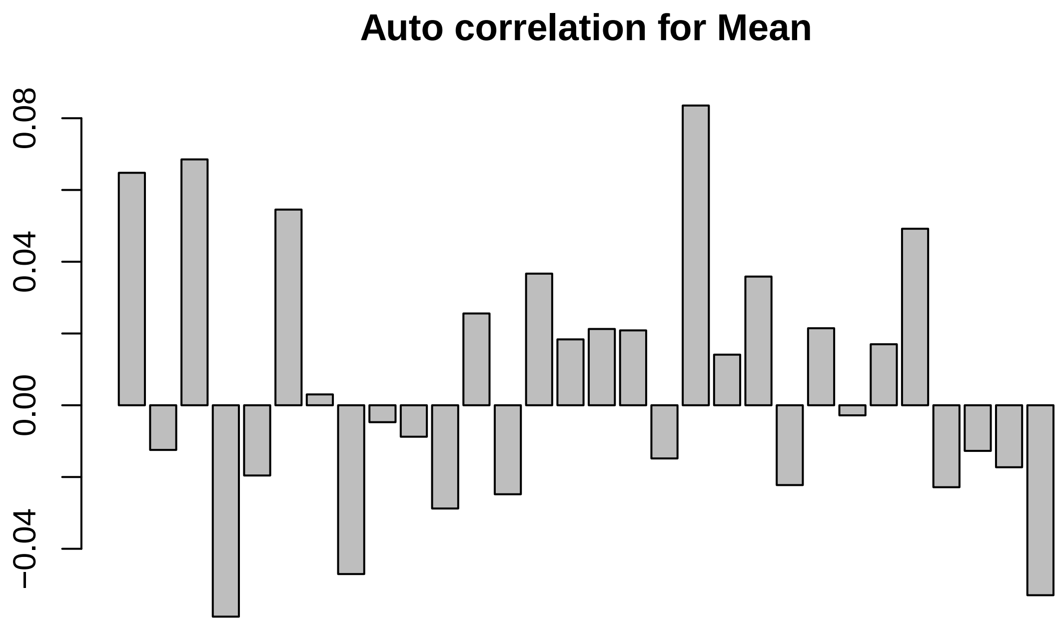
```
# Traceplot of Gibbs draws
# par(mfrow=c(2,1))
plot(1:nDraws, gibbsDraws[,1], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Mean')
```



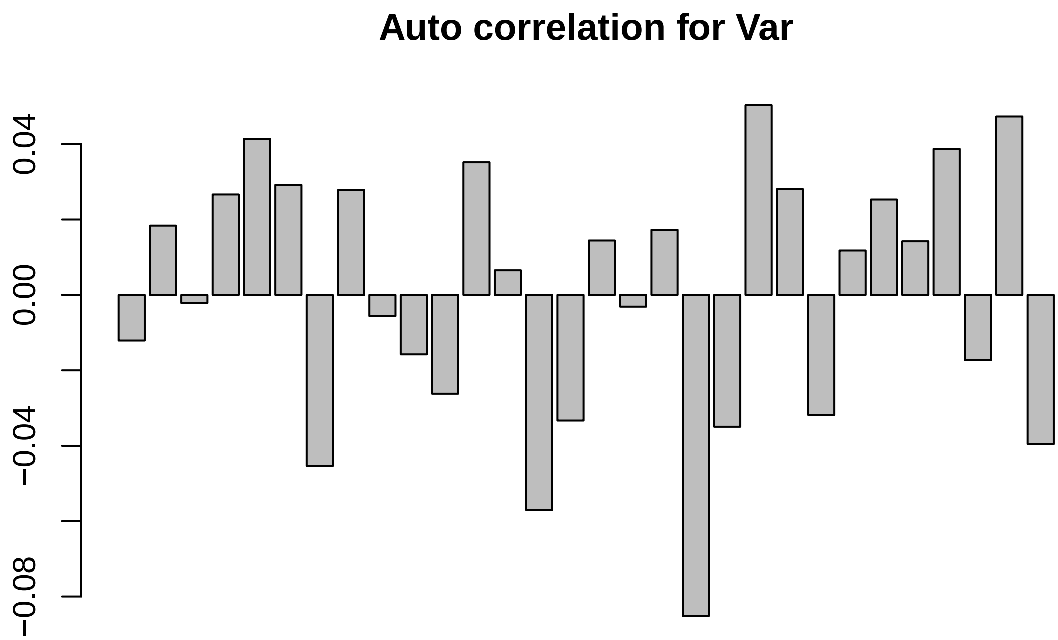

```
plot(1:nDraws, gibbsDraws[,2], type = "l",#col="red",
     xlab = 'Iterations', ylab = 'Posterior Var') # traceplot of Gibbs draws
```



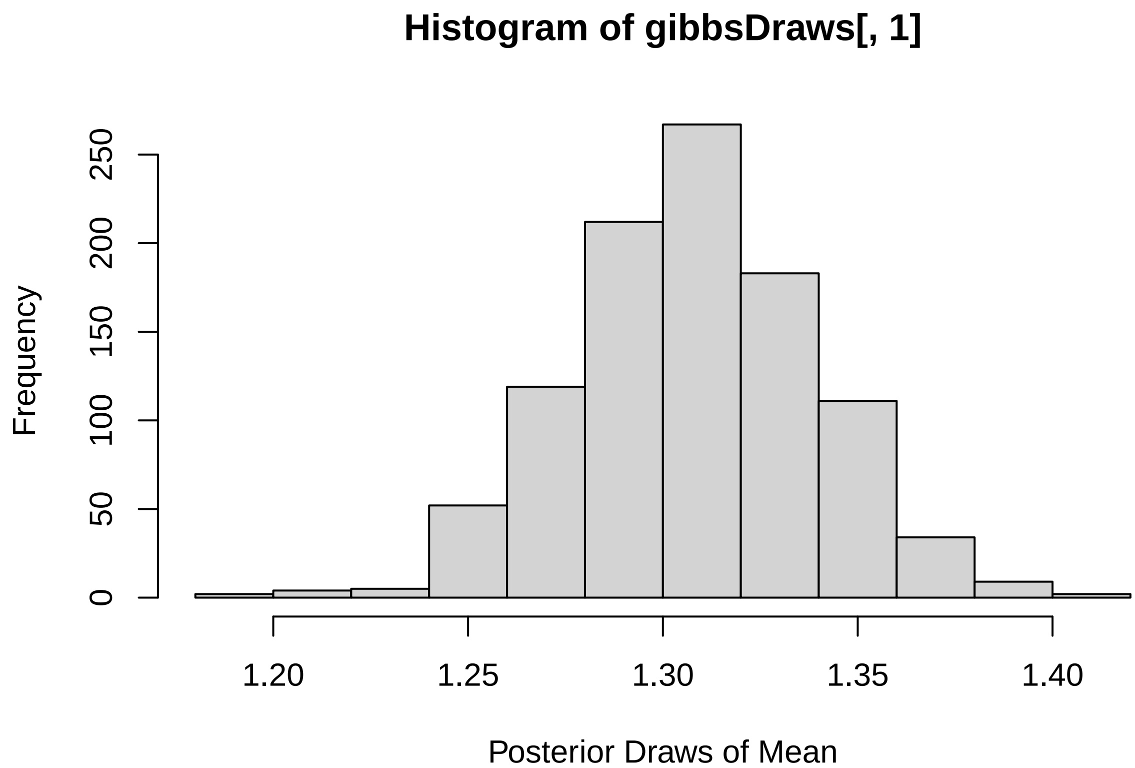

```
# par(mfrow=c(2,1))
# Acf for Gibbs draws
barplot(height = a_mu_Gibbs$acf[-1], main = 'Auto correlation for Mean')
```



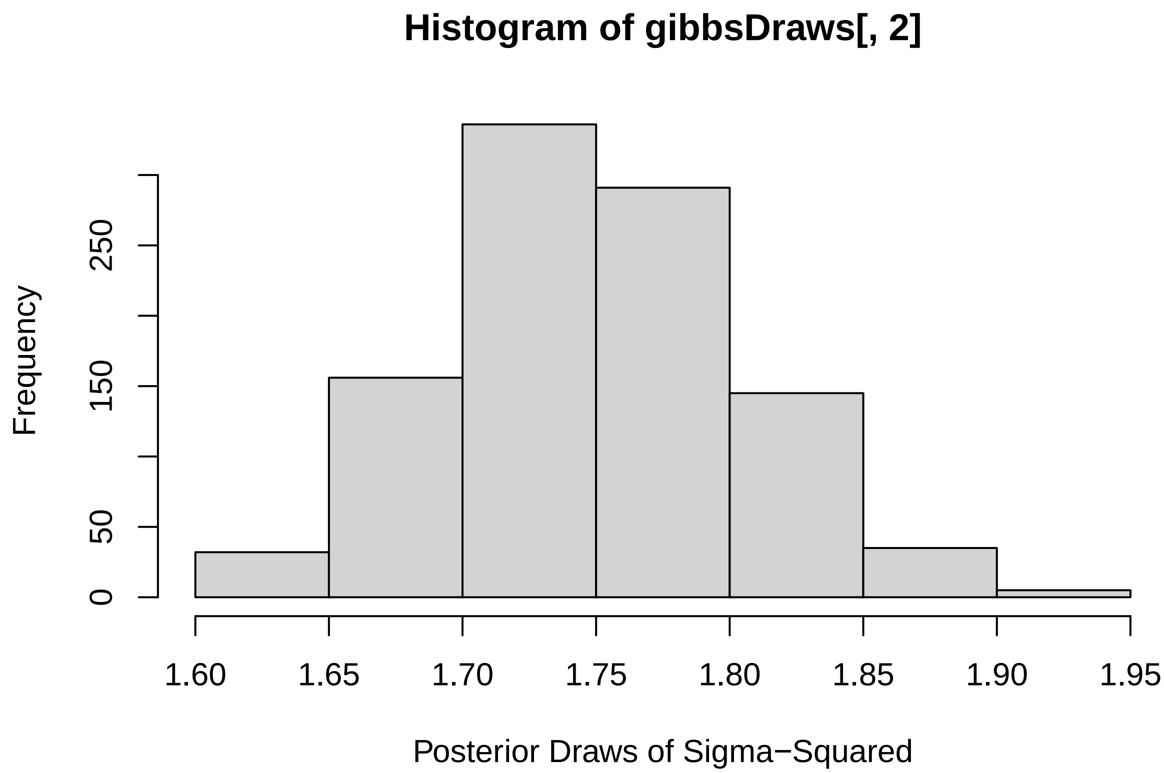

```
barplot(height = a_var_Gibbs$acf[-1], main = 'Auto correlation for Var')
```




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

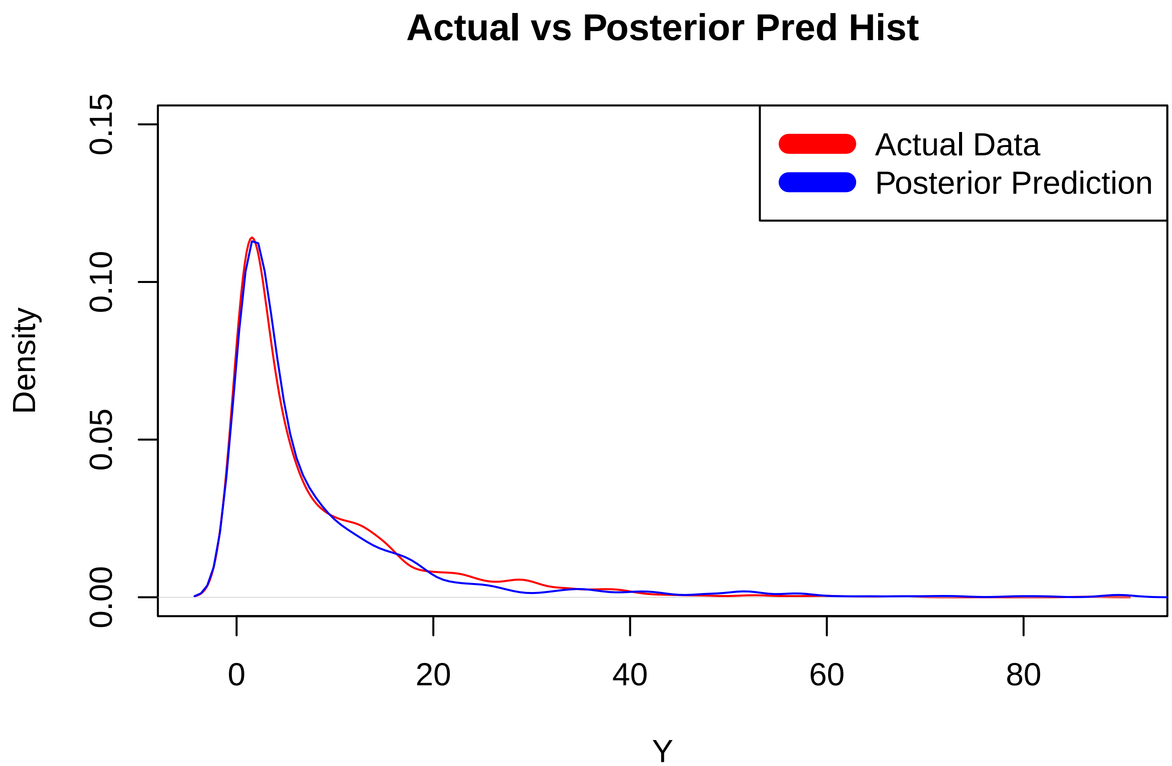



```
hist(gibbsDraws[,2], xlab = 'Posterior Draws of Sigma-Squared')
```



1 b).

```
#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)
```



Question 2

2 a).

```
#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
```

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

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

```r
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).

```r
#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)
```

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

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

```r
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
```

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

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

```r
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,] -2.772239e-04 -2.082668e-04  2.282539e-04  4.532308e-04  3.492353e-04
##  [7,] -5.128351e-04  2.801777e-04  3.313568e-04  3.376467e-04  5.844006e-05
##  [8,]  6.436765e-05  1.181852e-04 -3.191869e-04 -1.311025e-04  5.854011e-05
##  [9,]  1.109935e-03 -5.685706e-04 -4.292828e-04 -5.759169e-05 -6.437066e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.772239e-04 -5.128351e-04  6.436765e-05  1.109935e-03
##  [2,] -2.082668e-04  2.801777e-04  1.181852e-04 -5.685706e-04
##  [3,]  2.282539e-04  3.313568e-04 -3.191869e-04 -4.292828e-04
##  [4,]  4.532308e-04  3.376467e-04 -1.311025e-04 -5.759169e-05
##  [5,]  3.492353e-04  5.844006e-05  5.854011e-05 -6.437066e-05
##  [6,]  8.365059e-03  4.048644e-04 -8.975843e-05  2.622264e-04
##  [7,]  4.048644e-04  3.175060e-03 -2.541751e-04 -1.063169e-04
##  [8,] -8.975843e-05 -2.541751e-04  8.384703e-04  1.037428e-03
##  [9,]  2.622264e-04 -1.063169e-04  1.037428e-03  5.054757e-03
```

2 c).

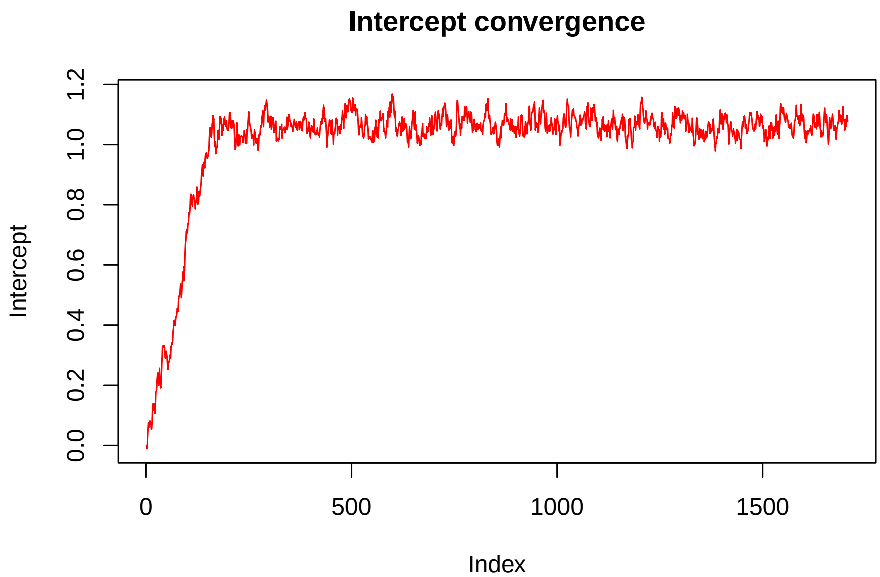
```r
# Question c: Simulate from posterior beta using Metropolis Hasting

# 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.
  #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:"
```

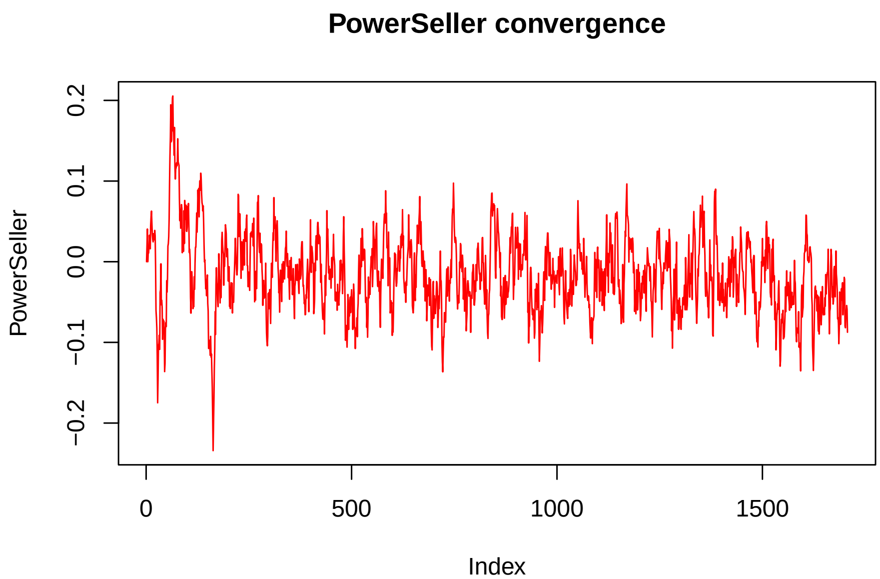
```r
nrow(pos_beta_mat)/Nsamples#Tune c values based on acceptance rate: 25-30%
```

```
## [1] 0.3414
```

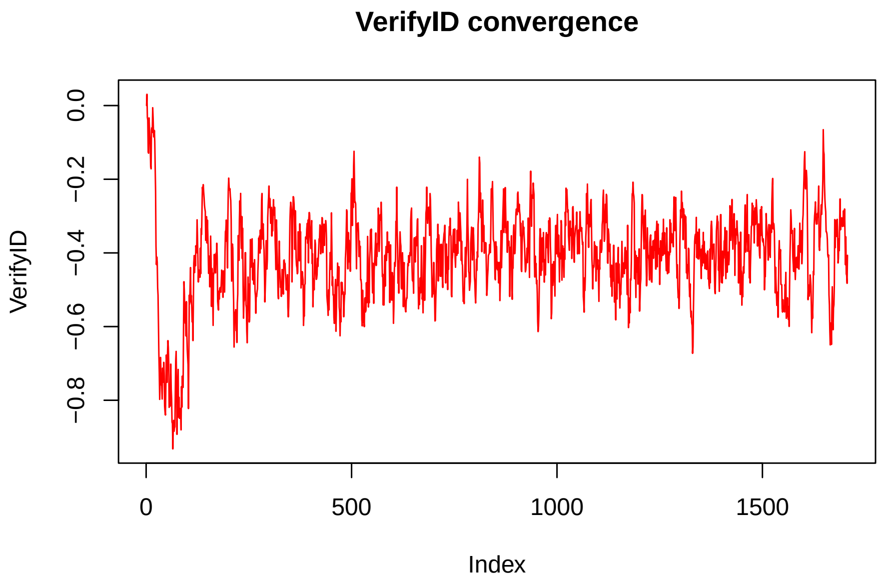
```r
plot(pos_beta_mat[,1], type = "l",col= "red",
     ylab = 'Intercept', main = "Intercept convergence")
```



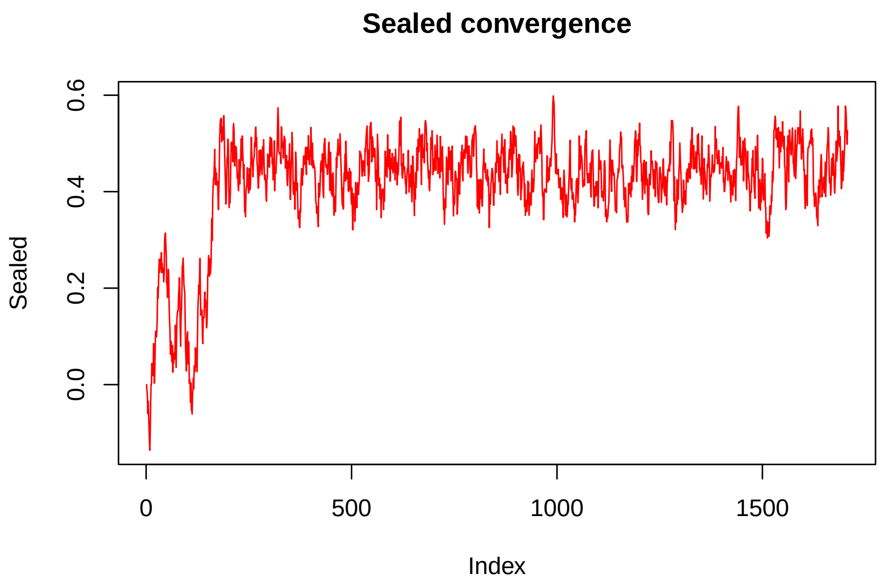

```r
plot(pos_beta_mat[,2], type = "l",col="red",
     ylab = 'PowerSeller', main = "PowerSeller convergence")
```



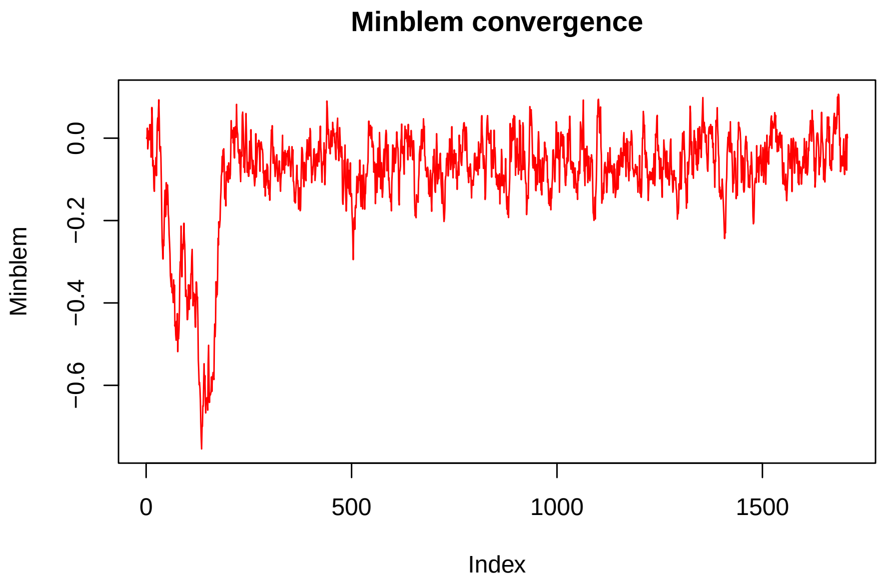

```r
plot(pos_beta_mat[,3], type = "l",col="red",
     ylab = 'VerifyID', main = "VerifyID convergence")
```



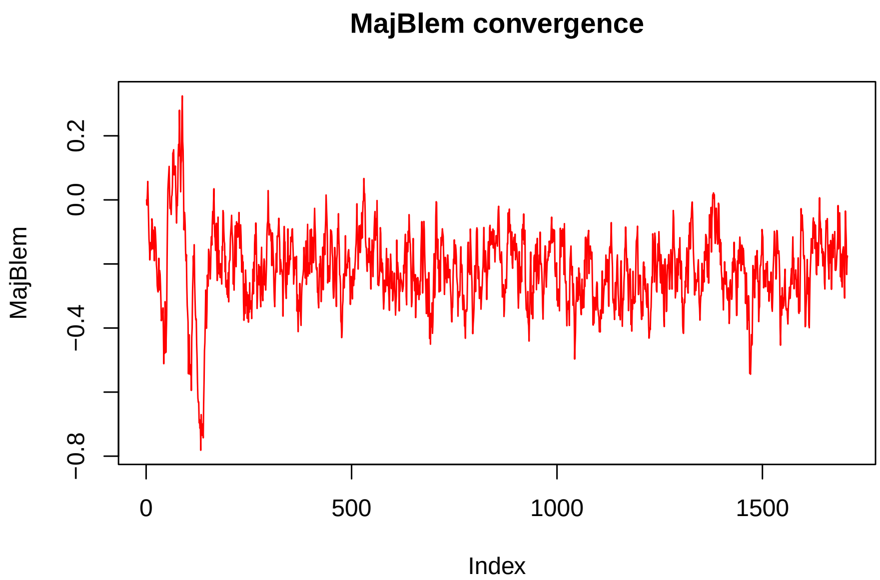

```
plot(pos_beta_mat[,4], type = "l",col="red",
     ylab = 'Sealed', main = "Sealed convergence")
```



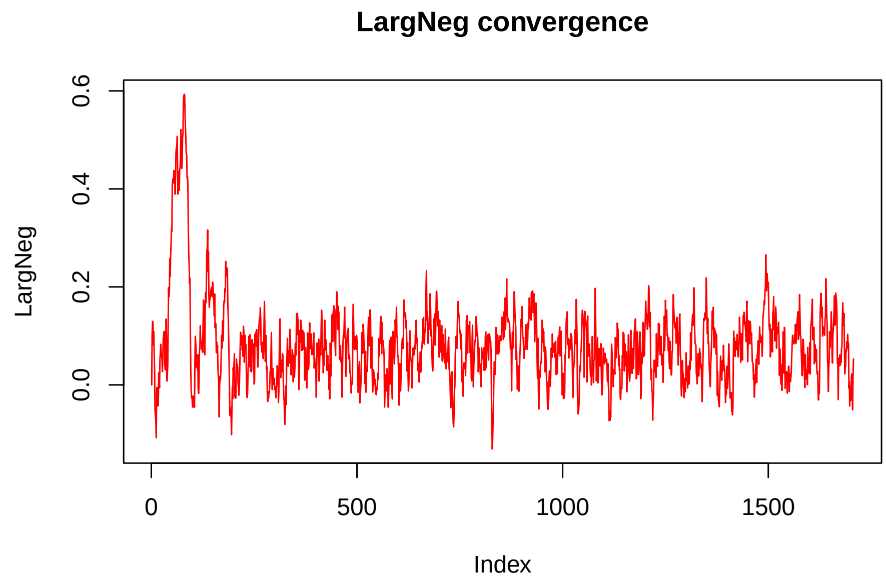

```
plot(pos_beta_mat[,5], type = "l",col="red",
     ylab = 'Minblem', main = "Minblem convergence")
```



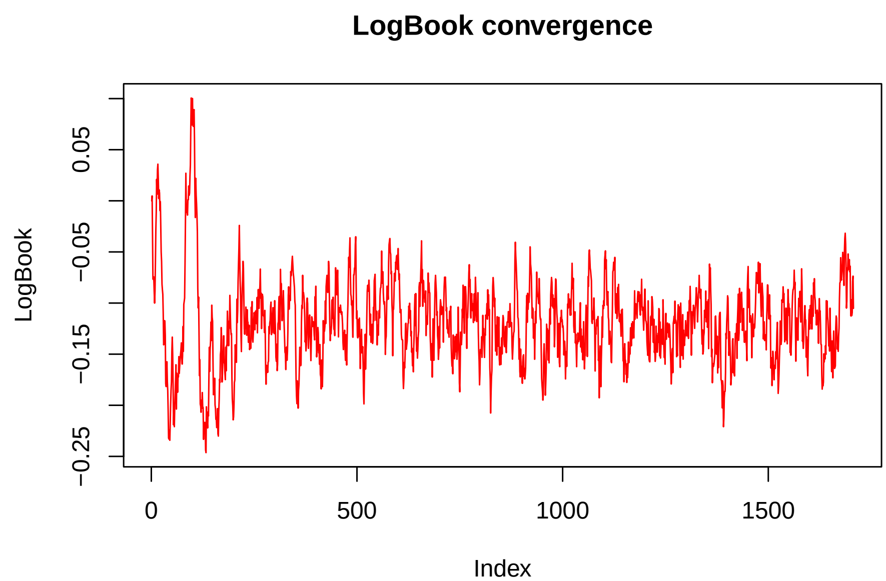

```
plot(pos_beta_mat[,6], type = "l",col="red",
     ylab = 'MajBlem', main = "MajBlem convergence")
```



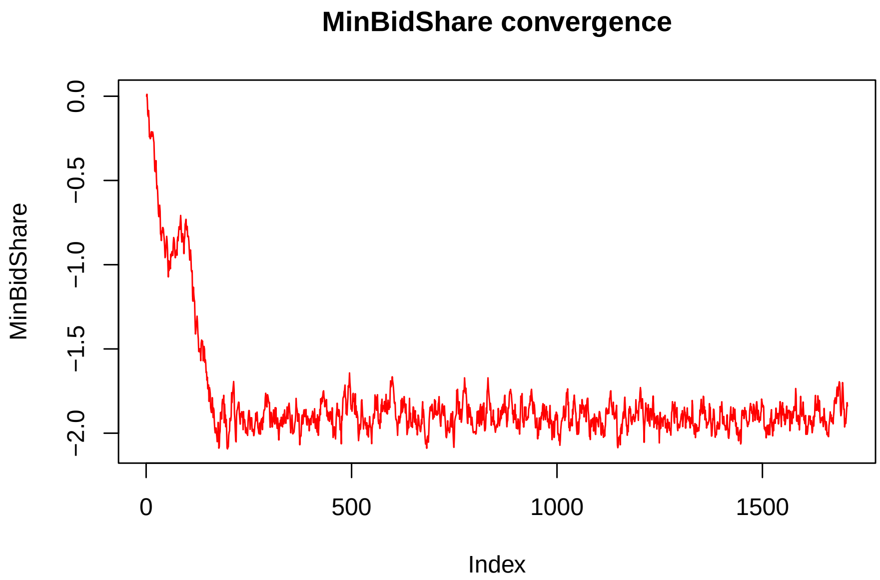

```
plot(pos_beta_mat[,7], type = "l",col="red",
     ylab = 'LargNeg', main = "LargNeg convergence")
```




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




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




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

```
## [1]  1.01428871 -0.01615839 -0.41168552  0.41481405 -0.08554989 -0.21660927
## [7]  0.08008683 -0.11908405 -1.81326592
```