Bayesian Learning Computer Lab 3

bisku859 and gowku593

5/12/2021

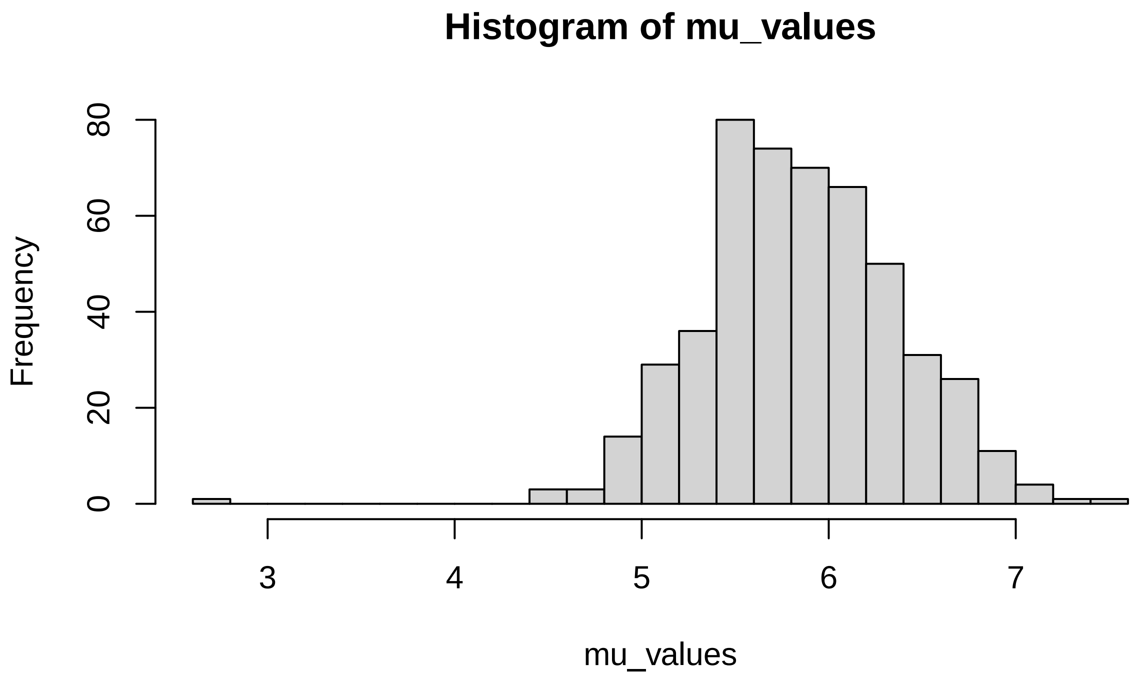
Assignment 1. Gibbs sampler for a normal model

The dataset rainfall.dat consists of daily records, from the beginning of 1948 to the end of 1983, of precipitation (rain or snow in units of 1/100 inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Assume the natural log of the daily precipitation $\{y_1, ...y_n\}$ are independent normally distributed . $lny_1, ...lny_n \mid \mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2)$, where both μ and σ^2 are unknown.

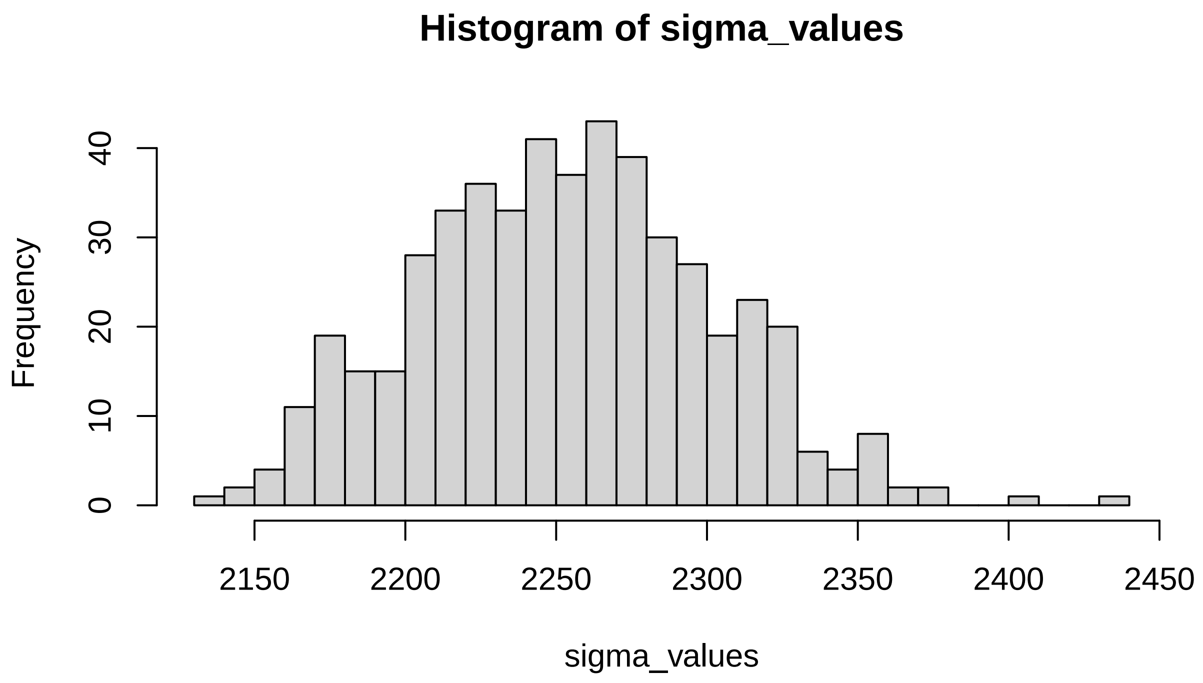
Let $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv - \chi^2(v_0, \tau_0)$.

a). Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2 \mid lny_1, ..., y_n)$. The full conditional posteriors are given on the slides from Lecture 7. Evaluate the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs) and plotting the trajectories of the sampled Markov chains.

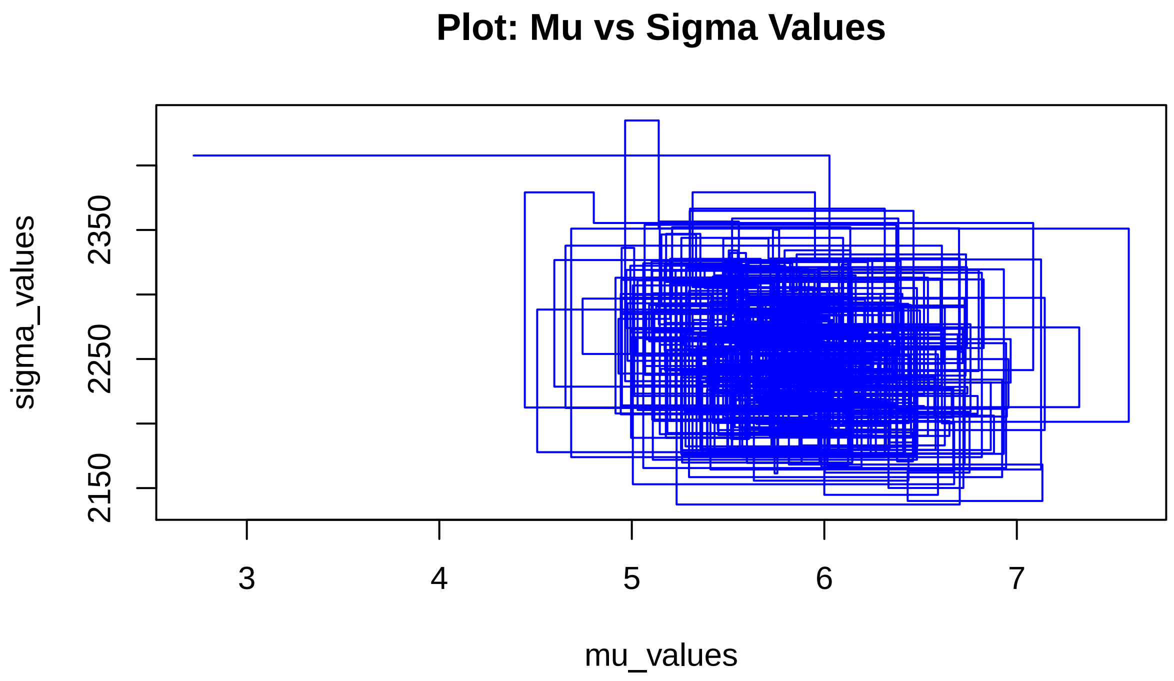
```
data <-as.vector( read.table("rainfall.dat", sep = "", dec = ".")[,1])

# set up   from lecture slide,
# chossing initial values
rho <- 0.9
mu <- mean(log(data))
sigma2 <-  var(log(data))
nu0 <- 1
tau02 <- 2
mu0 <- 25
sigma02 <- 5

n <- length(data)
xbar <- mean(log(data)) #or mu

# nu_n is given by nu_0+n

nu_n <- nu0 +n


# to draw  mu from full conditional posterior sigma^2, we draw from the scaled inverse chi square distr
# we  need to draw mu for  sigma^2, where taun^2 is given with a loop

mu_values <-c() #
sigma_values <-c()

sigmasq <-1
mu_draw<-NULL
#prior-to-posterior mapping  (Lecture 1):
for (i in 1:500){
  w= (n/sigmasq)/((n/sigmasq)+(1/tau02))
  mun = w*xbar +(1-w)*mu0

  taun2 = 1/((n/sigmasq)+(1/tau02))    # reciprocal of 1/taun^2
  mu_draw = rnorm(n=1, mean = mun, sd= sqrt(taun2))
  mu_values <- append(mu_values, mu_draw)
  x = rchisq(n=1, df=nu_n )  # pick x values
  sigmasq = nu_n*(nu0*sigma02+sum((data-mu_draw)^2)/n+nu0)/x
  sigma_values <- append(sigma_values,sigmasq)

}


hist(mu_values,breaks = 30)
```



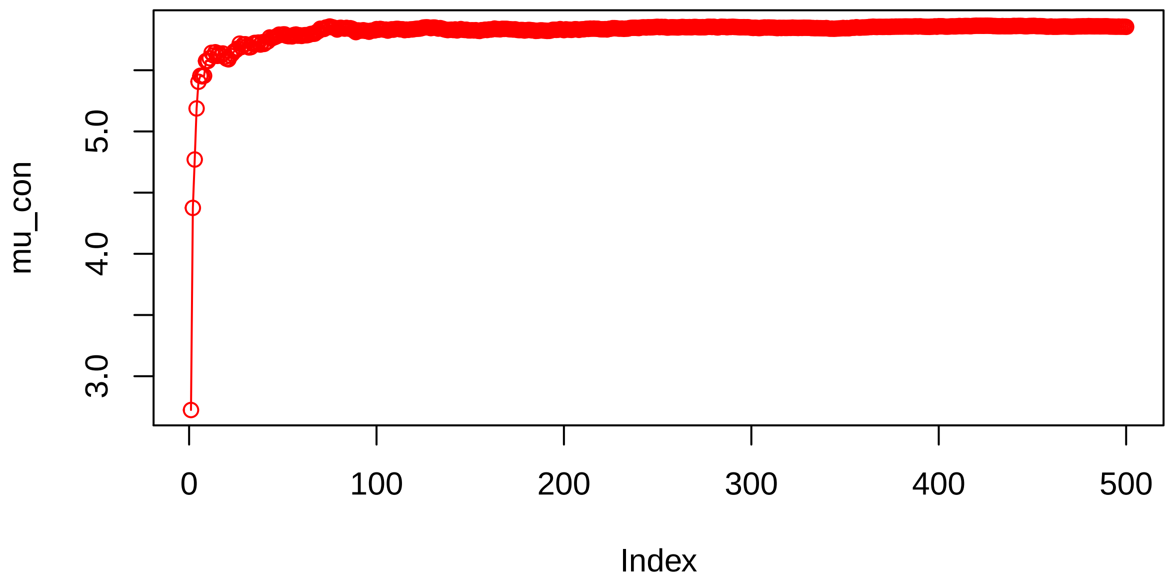
```
hist(sigma_values,breaks = 30)
```



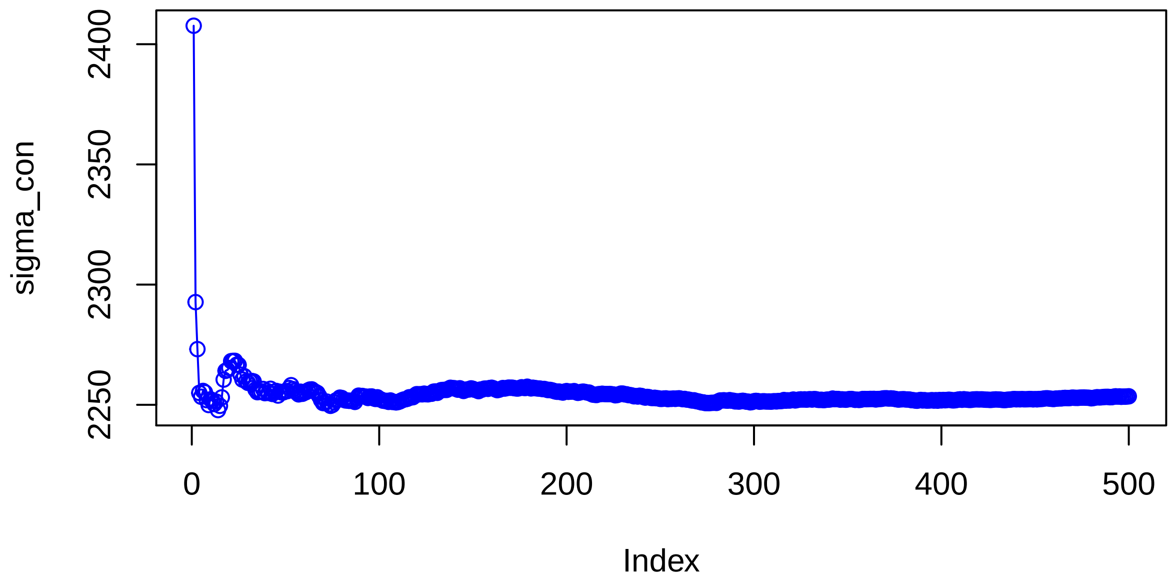
```r
plot(mu_values,sigma_values, type = "s", col="blue",main = "Plot: Mu vs Sigma Values")
```



```r
mu_con<-cumsum(mu_values)/seq_along(mu_values)
plot(mu_con,type = "o",col="red")
```



```r
sigma_con<-cumsum(sigma_values)/seq_along(sigma_values)
plot(sigma_con,type = "o",col="blue")
```



It bascically represents the average moving mean of the mu and sigma values of the sample precipitations. The sampling method starts with a lot of fluctuations and therefore an initial burn-in period is recommended for the collection of sample data corresponding to the distribution.

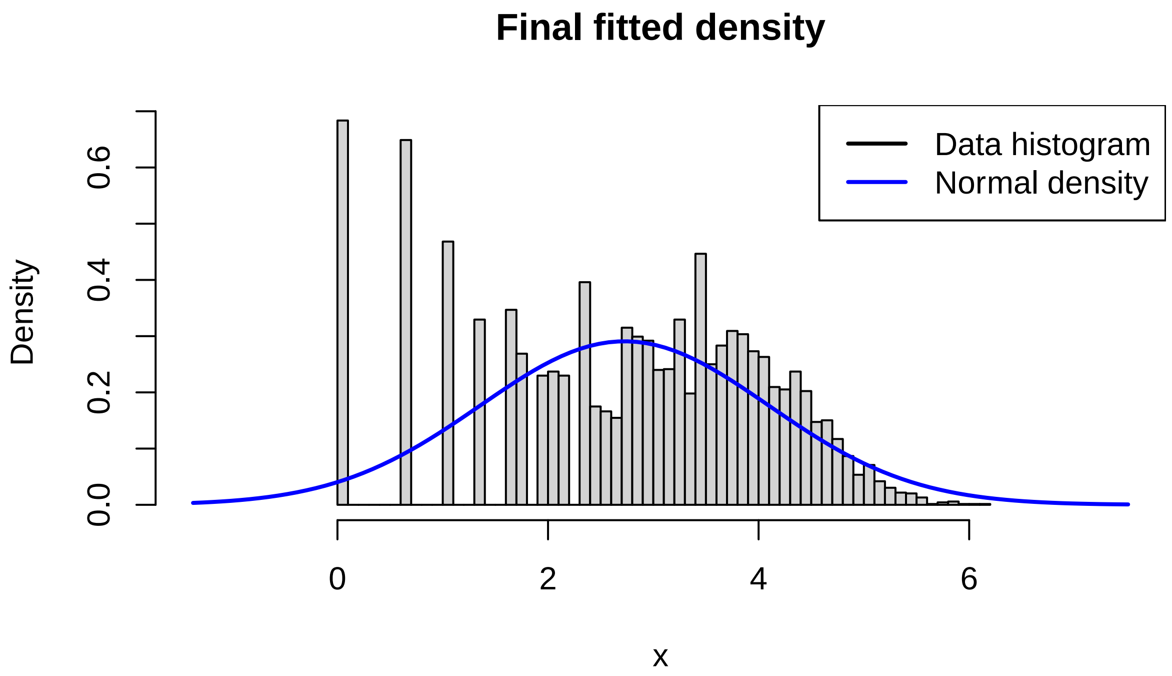
The above graphs of mu and sigma shows that the data is converging well after initial hiccups. These initial samples can be discarded as burn-in period for smooth observations/estimations.

We feel that this kind of estimations is very useful for analysing and predicting varying data and one such example is weather data.

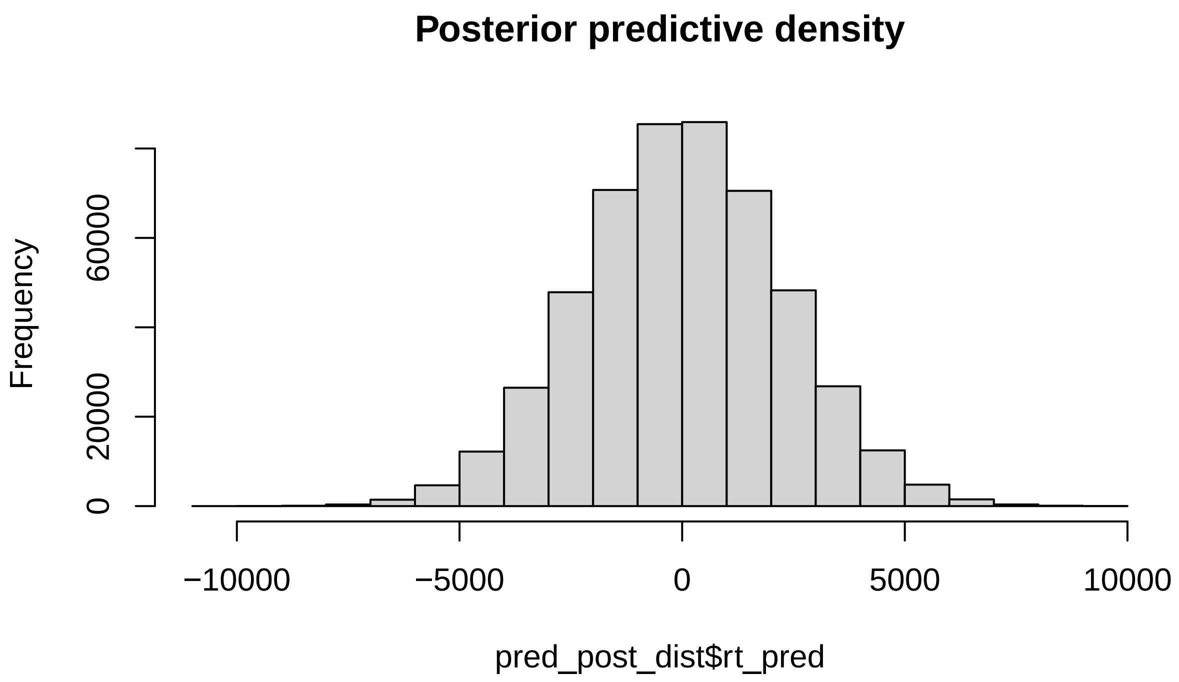
Plot the following in one figure:

1) a histogram or kernel density estimate of the daily precipitation $\{y_1, ...y_n\}$
2) The resulting posterior predictive density $p(\tilde{y} \mid y_1, ..., y_n)$ using the simulated posterior draws from (a). How well does the posterior predictive density agree with this data?

```r
#i.) histogram or kernel density estimate of the daily precipitation
data2<-read.table("rainfall.dat")
x <- as.matrix(log(data2))

# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)

hist(x, breaks = 50, freq = FALSE, xlim = c(xGridMin,xGridMax),
     main = "Final fitted density")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1,
       legend = c("Data histogram","Normal density"),
       col=c("black","blue"), lwd = 2)
```



```r
# ii) Posterior predictive density (posterior samples from a) to simulate the predictive density of y

mu_samples<-mu_values
sigma_samples<-sigma_values
No_obs<-1000

pred_dist <- function(mu_samples, sigma_samples, No_obs) {
  output <- tibble(Sno = numeric(0),
                   rt_pred = numeric(0))
  for (i in 1:length(mu_samples)) {
    mu <- mu_samples[i]
    sigma <- sigma_samples[i]
    output <- bind_rows(output ,
      tibble( Sno = seq_len(No_obs),rt_pred = rnorm(No_obs, mu, sigma))
    )
  }
  output
}

pred_post_dist<-pred_dist(mu_samples, sigma_samples, No_obs)


hist(pred_post_dist$rt_pred,main = "Posterior predictive density")
```



The histogram and the posterior density has been plotted above. It fails to capture all the data points, especially the initial ones. However, it did capture the majority of the data points as observed in the histogram.

2. Metropolis Random Walk for Poisson regression

a)

```
Data<- read.table("eBayNumberOfBidderData.dat",header=TRUE)
#head(data)
#length(Data)
# as per lecture codes
y <- as.vector(Data[,1]); # Data from the read.table function is a data frame.
#Let's convert y and X to vector and matrix respectivelky.
X <- as.matrix(Data[,2:10])
covNames <- names(Data)[2:length(names(Data))]

poisson_model <-glm(nBids~PowerSeller+VerifyID+Sealed+Minblem+MajBlem+LargNeg+LogBook+MinBidShare,
                    data=Data, family = poisson)
coefficients(poisson_model)
```

```
## (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
# Summary to check for most significant coefficients
summary(poisson_model)
```

```
##
## Call:
## glm(formula = nBids ~ PowerSeller + VerifyID + Sealed + Minblem +
##     MajBlem + LargNeg + LogBook + MinBidShare, 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
```

```r
chooseCov <- c(1,3,4,6,8,9)
sign_X <- X[,chooseCov] # choosen the most significant covariates
cat("The covariates that are significant are : \n",colnames(sign_X),sep = "\n")
```

```
## The covariates that are significant are :
##
## Const
## VerifyID
## Sealed
## MajBlem
## LogBook
## MinBidShare
```

```r
#const is intercept
```

b)

The PDF for the Poisson distribution is given by $\frac{\lambda^x e^{-\lambda}}{x!}$

We estimate for λ from the joint PDF $\Pi_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}$

Taking log , the value of λ maximizes $\log(\Pi_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}) = \sum_{i=1}^{n} \log(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}) = \sum_{i=1}^{n} x_i \log(\lambda) - \sum_{i=1}^{n} \lambda - \sum_{i=1}^{n} \log(x_i!)$

or, $= \log(\lambda) \sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n} \log(x_i!)$

Now, comparing with given equation, we can say that $\lambda = exp(x_i'\beta)$

or, $= x_i'\beta \sum_{i=1}^{n} x_i - nx_i'\beta - \sum_{i=1}^{n} \log(x_i!)$

```
# Priors are below :
Beta_prior = rep(0, 9) # mean prior
Sigma_prior = 100 * solve(t(X) %*% X) # Sigma prior

logPostPoisson = function(Beta, mu, Sigma_prior, X, y) {
  p = length(Beta)

  # log of the likelihood
  log.likelihood = sum(y * X %*% Beta- exp(X %*% Beta))

  # if likelihood is very large or very small, stear optim away
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

  # log of the prior
  log.prior = dmvnorm(Beta, mean = mu, sigma = Sigma_prior, log = TRUE)

  return(log.likelihood + log.prior)
}


# Optimize posterior of beta given priors and data
OptimResults = optim(Beta_prior, logPostPoisson, gr=NULL,
                   Beta_prior, Sigma_prior, X, y,
                   method="BFGS", control=list(fnscale=-1), hessian=TRUE)

postMode<-OptimResults$par # Beta hat
cat("\n  The Posterior Mode or Beta_hat : \n",postMode,"\n")
```

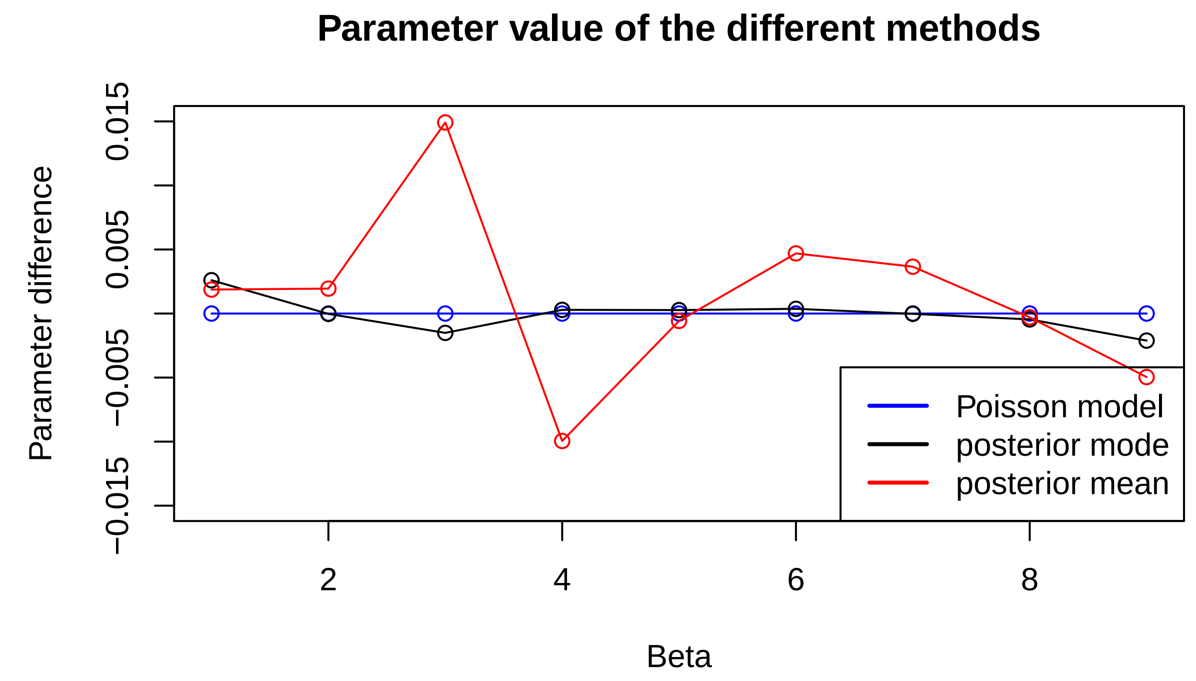
```
##
##   The Posterior Mode or Beta_hat :
##  1.069841 -0.02051246 -0.393006 0.4435555 -0.05246627 -0.2212384 0.07069683 -0.1202177 -1.891985
```

```
postCov<--solve(OptimResults$hessian) # posterior Covariance Matrix , (posterior variance)
cat("\n The Posterior Covariance Matrix  : \n",postCov,"\n")
```

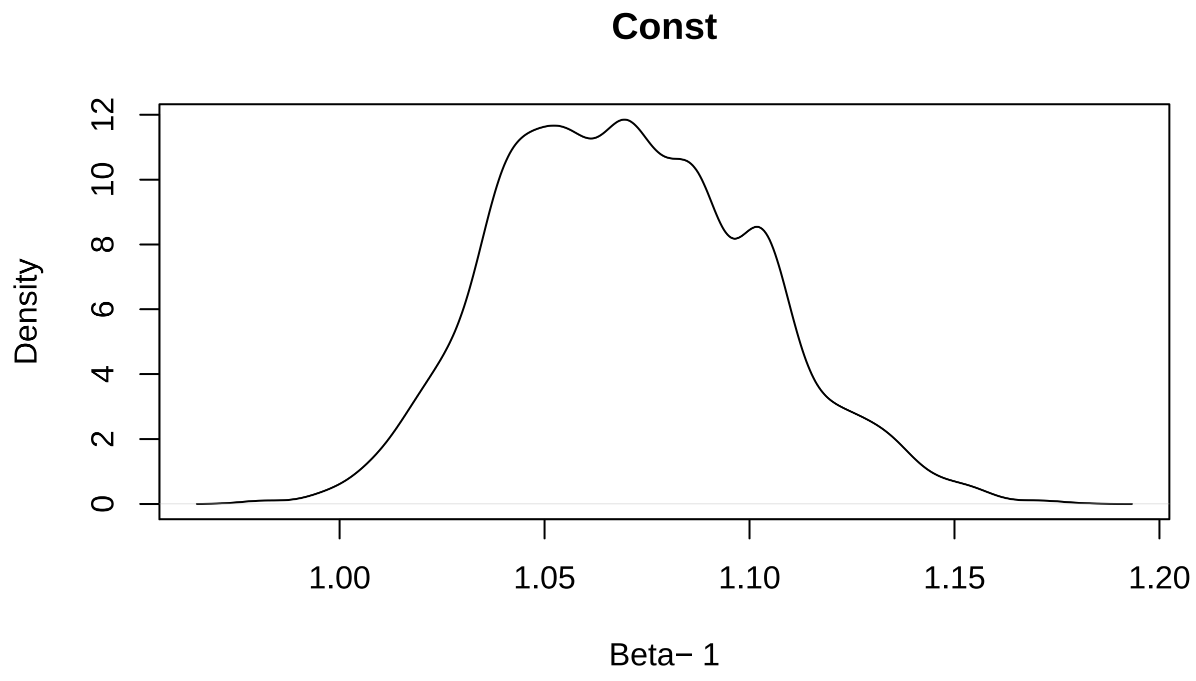
```
##
##  The Posterior Covariance Matrix  :
##  0.0009454625 -0.0007138972 -0.0002741517 -0.0002709016 -0.0004454554 -0.0002772239 -0.0005128351 6.
```

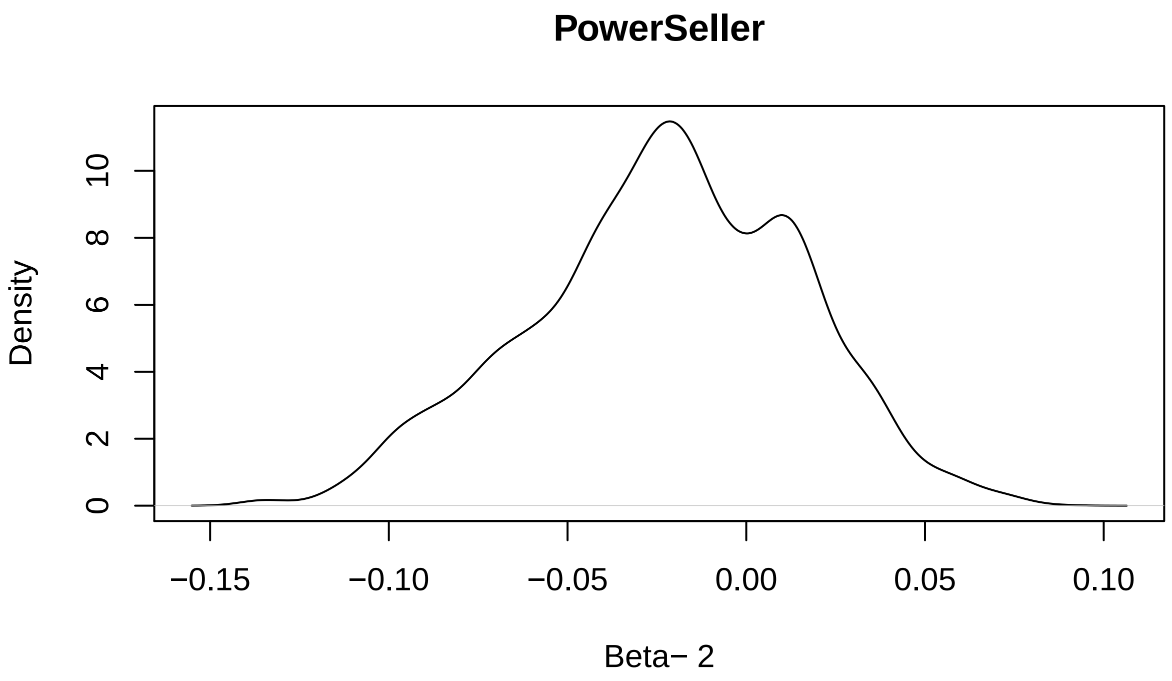
c)

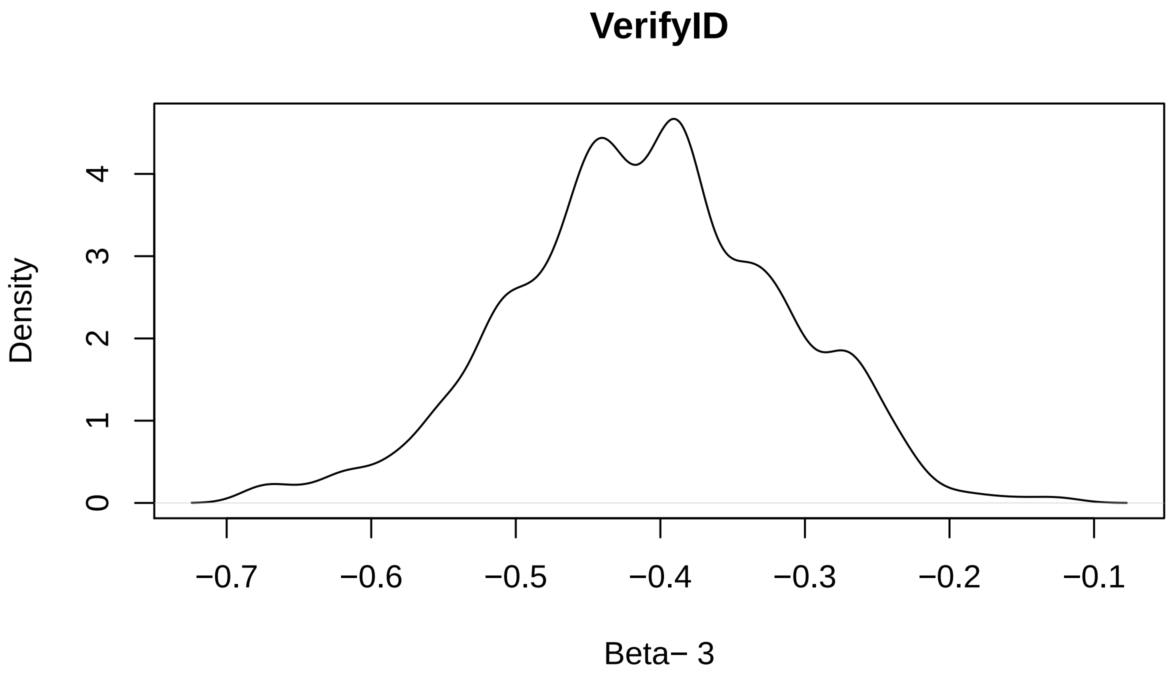
```
General_fun = function(nSamples, theta, c, logPostFunc, ...) {
  # Setup
  theta1 = theta
  Sigma1 = c * postCov # obtained in b
  counter = 0
  nPara = length(theta)
  samples = matrix(NA, size, nPara)

  # loop
  for(i in 1: size) {
    # Sample from proposal distribution as theta2
    theta2 = as.vector(rmvnorm(1, mean = theta1, sigma = Sigma1))

    # log posteriors
    log_post_2 = logPostFunc(theta2, ...)
    log_post_1= logPostFunc(theta1, ...)

    #  alpha
    alpha = min(1, exp(log_post_2 - log_post_1))

    # selection of samples wrt alpha
    u = runif(1)
    if (u <= alpha){
      theta1 = theta2
      counter = counter + 1
    }

    # Saving the samples
    if (i>0) samples[i,] = theta1
  }
  return(samples)
}

# Setup
c = 1
size = 4000


# Samples from posterior using metropolis
Beta_samples = General_fun( size,postMode, c, logPostPoisson, Beta_prior, Sigma_prior, X, y)

# Estimation of  parameters utilizing Beta_samples or posterior mean
postMode_mean = apply(Beta_samples, 2, mean)

# Plot parameters
plot(rep(0, 9), ylim = c(-0.015, 0.015), col="blue",
     xlab="Beta", ylab="Parameter difference", main="Parameter value of the different methods",type="o"
points(poisson_model$coefficients-postMode, col="black",type="o")
points(poisson_model$coefficients-postMode_mean, col="red",type="o")
legend("bottomright", legend=c("Poisson model", "posterior mode", "posterior mean"), lwd=2,
       col=c("blue", "black", "red"))
```

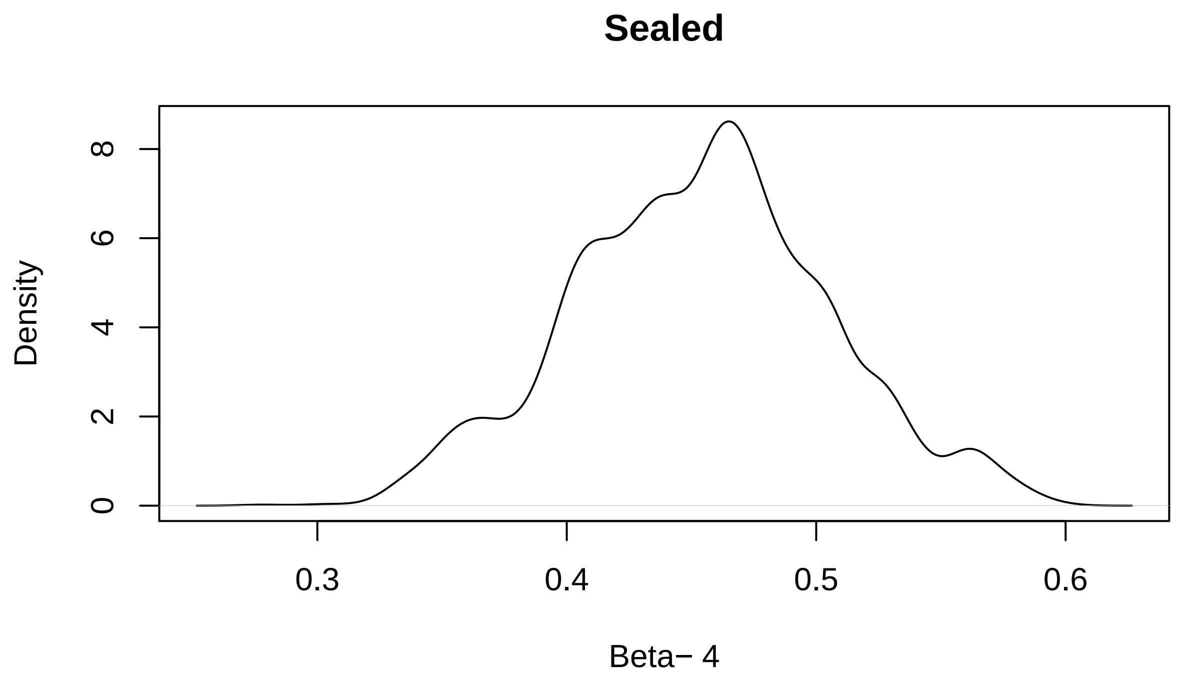


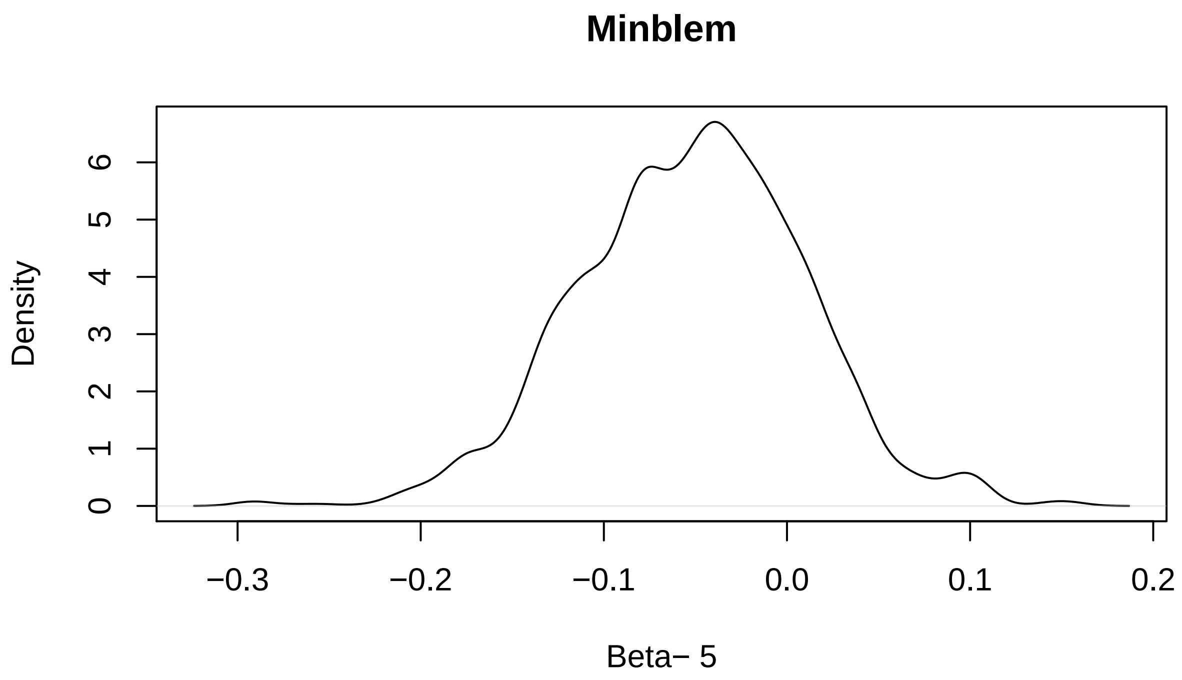
```r
# Plot posterior distribution of all Beta
for(i in 1:9){
  plot(density(Beta_samples[,i]), main=covNames[i], xlab=paste("Beta-",i), ylab="Density")
}
```

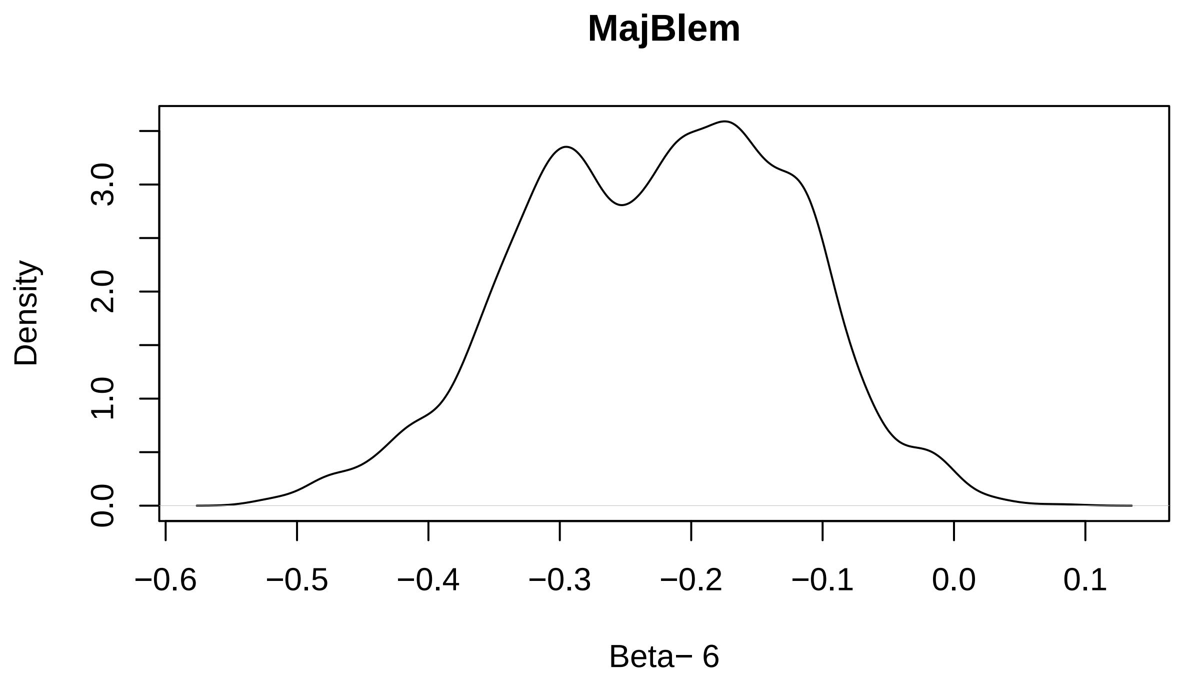


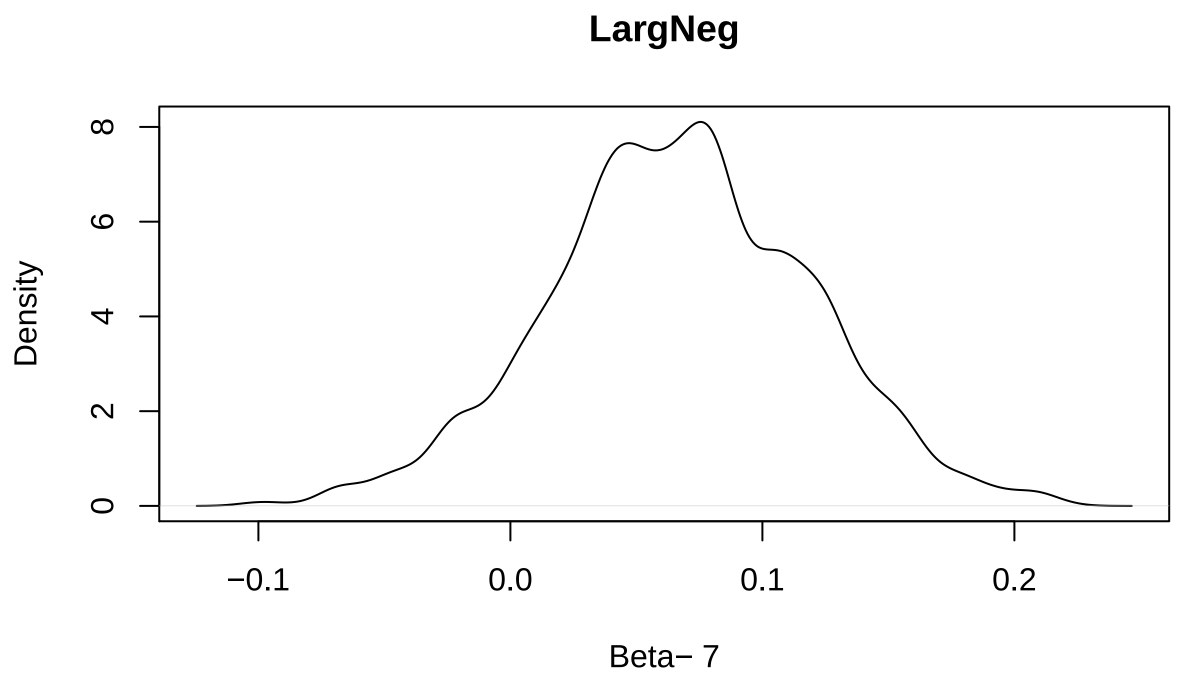

PowerSeller
Density
Beta− 2
0
2
4
6
8
10
−0.15
−0.10
−0.05
0.00
0.05
0.10

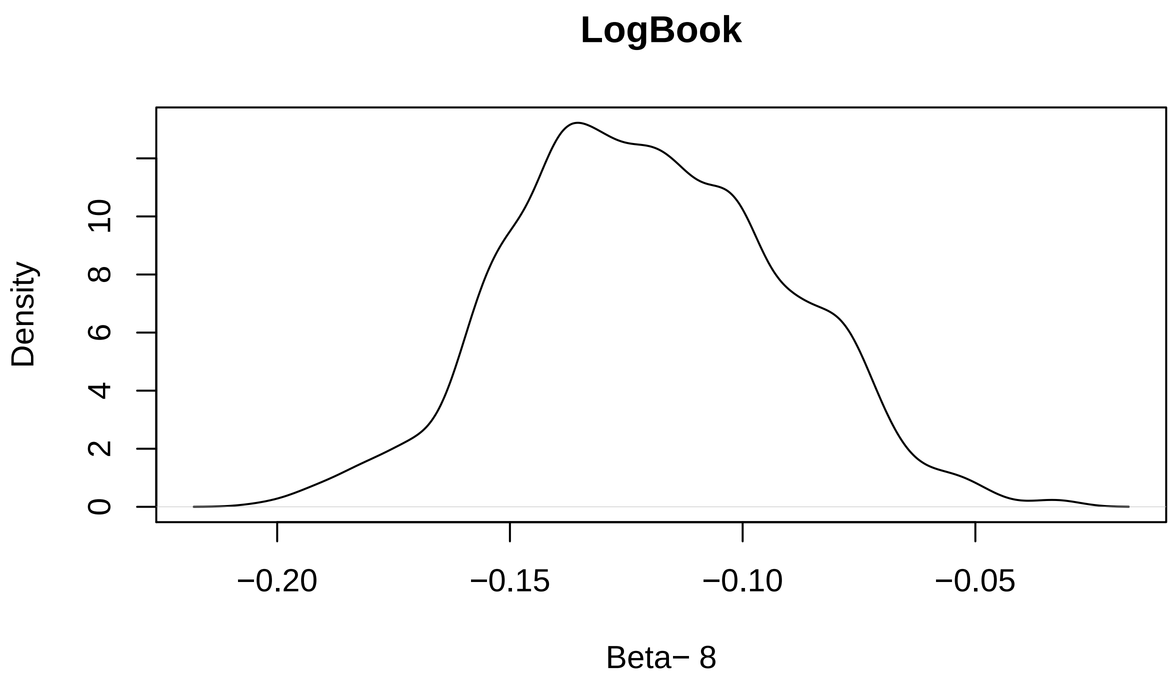

VerifyID
Density
Beta− 3
0
1
2
3
4
−0.7
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1

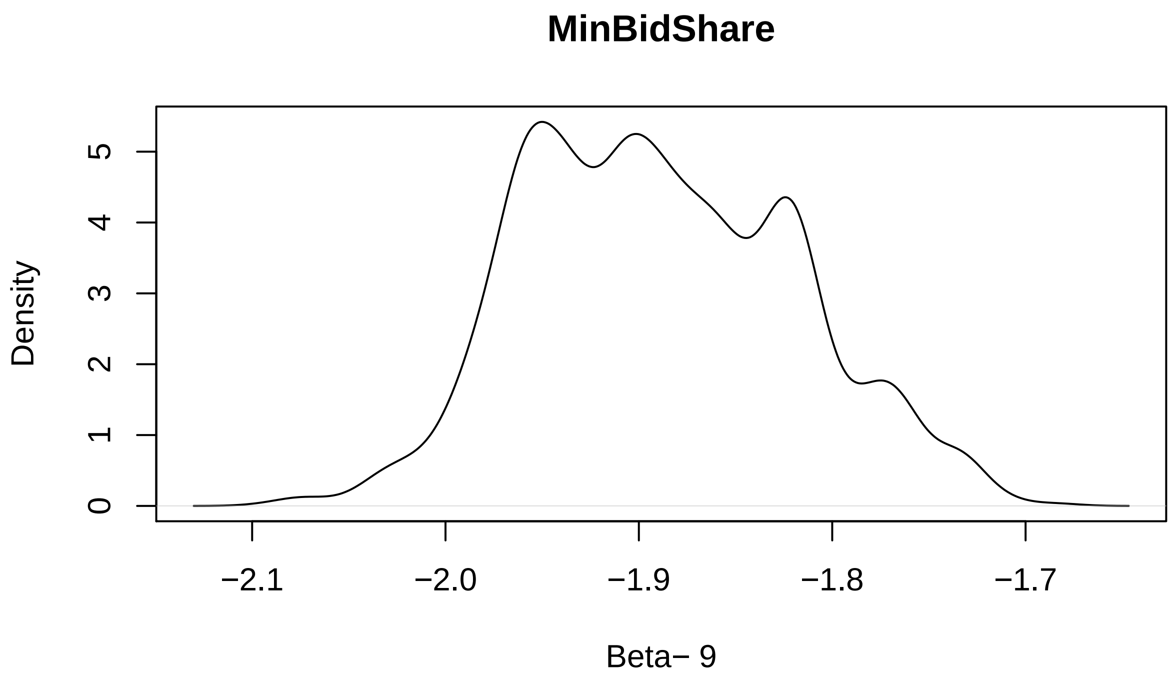

Sealed
Density
0
2
4
6
8
0.3
0.4
0.5
0.6
Beta− 4


Minblem
Density
0
1
2
3
4
5
6
−0.3
−0.2
−0.1
0.0
0.1
0.2
Beta− 5

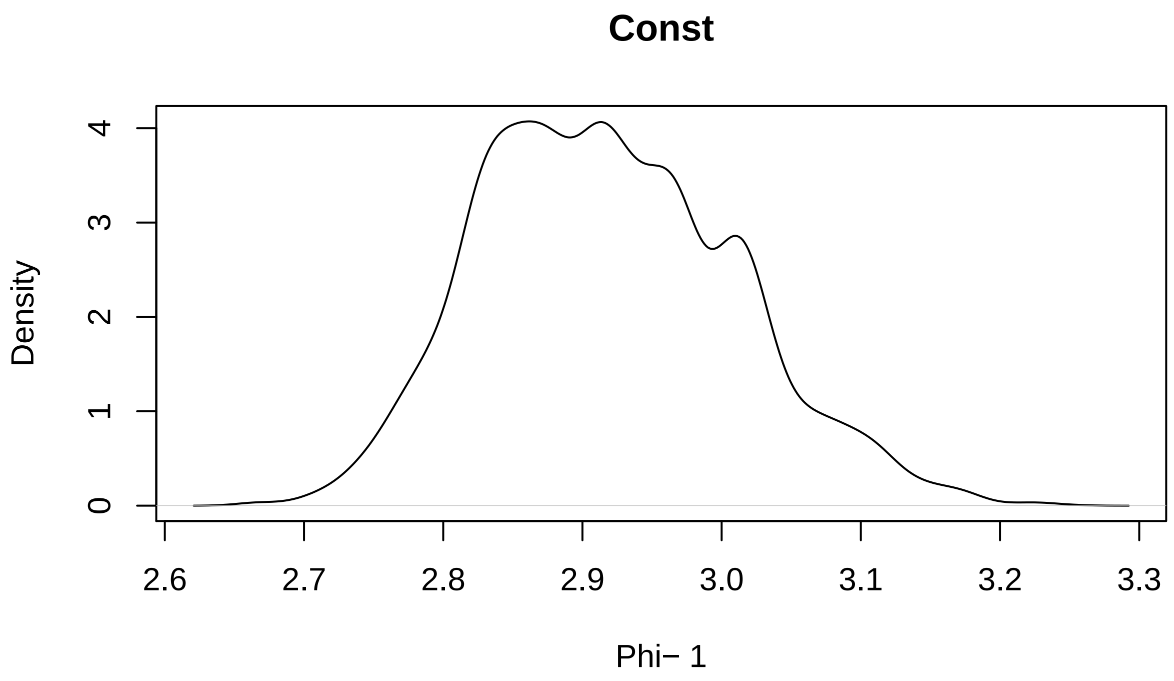

MajBlem
Density
0.0
1.0
2.0
3.0
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
0.1
Beta− 6

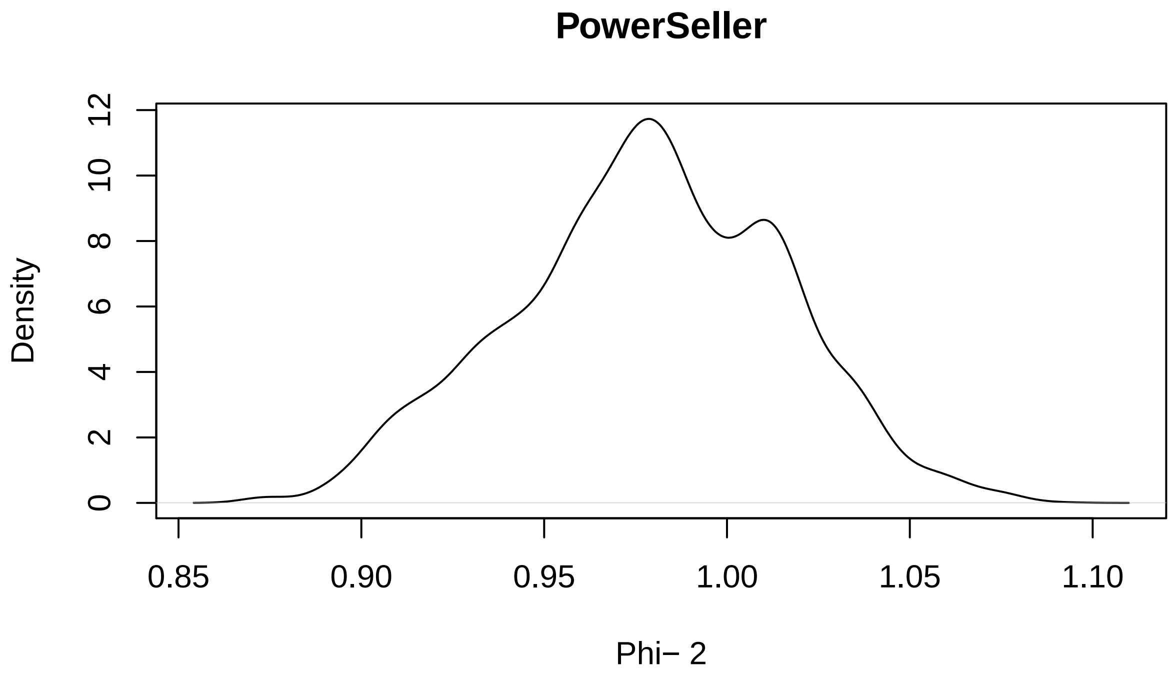

LargNeg
Density
0
2
4
6
8
−0.1
0.0
0.1
0.2
Beta− 7

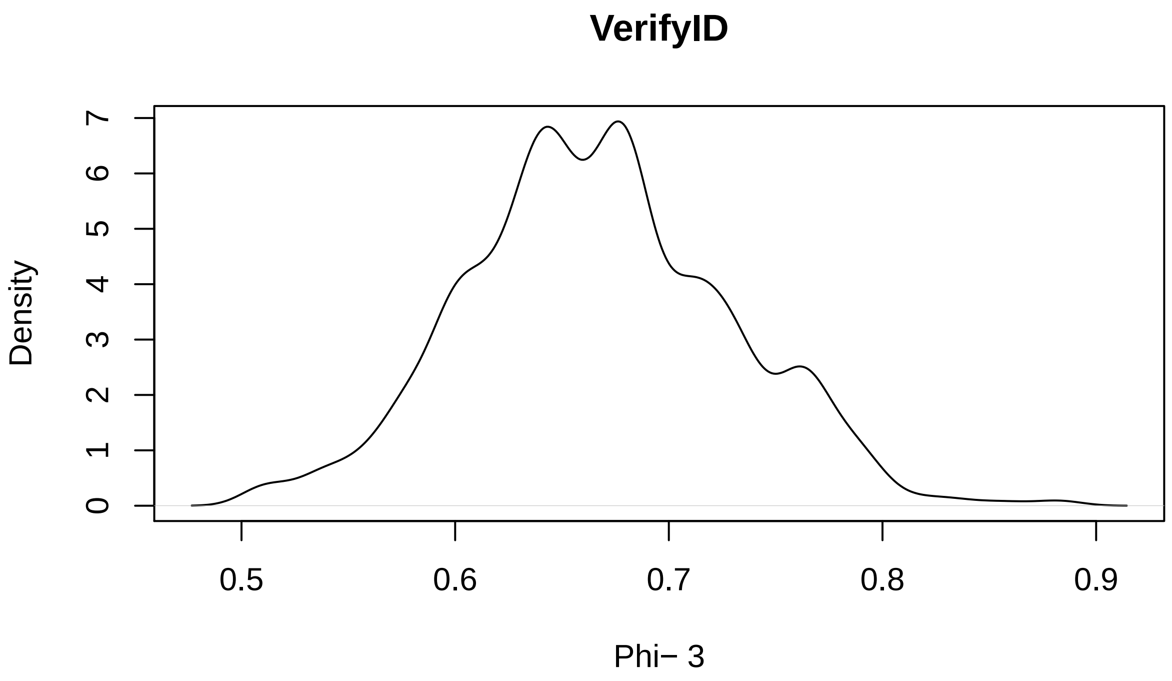


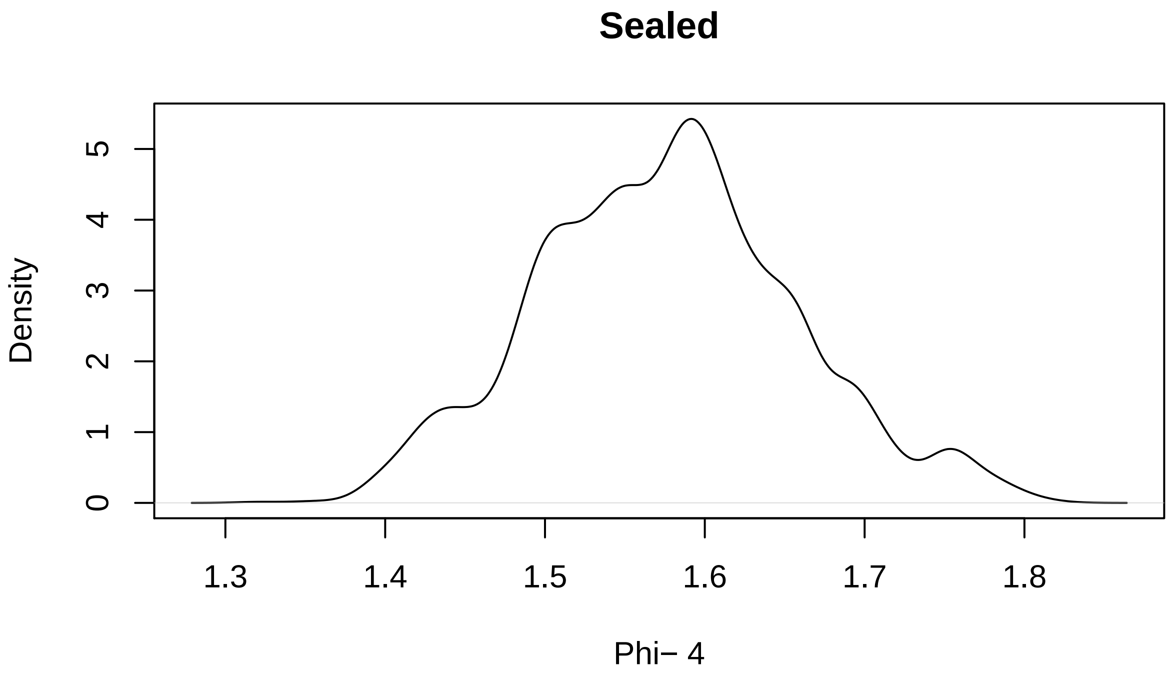


```r
# Plot posterior distribution of all phi
phis = exp(Beta_samples)
for(i in 1:9){
  plot(density(phis[,i]), main=covNames[i], xlab=paste("Phi-",i), ylab="Density")
}
```

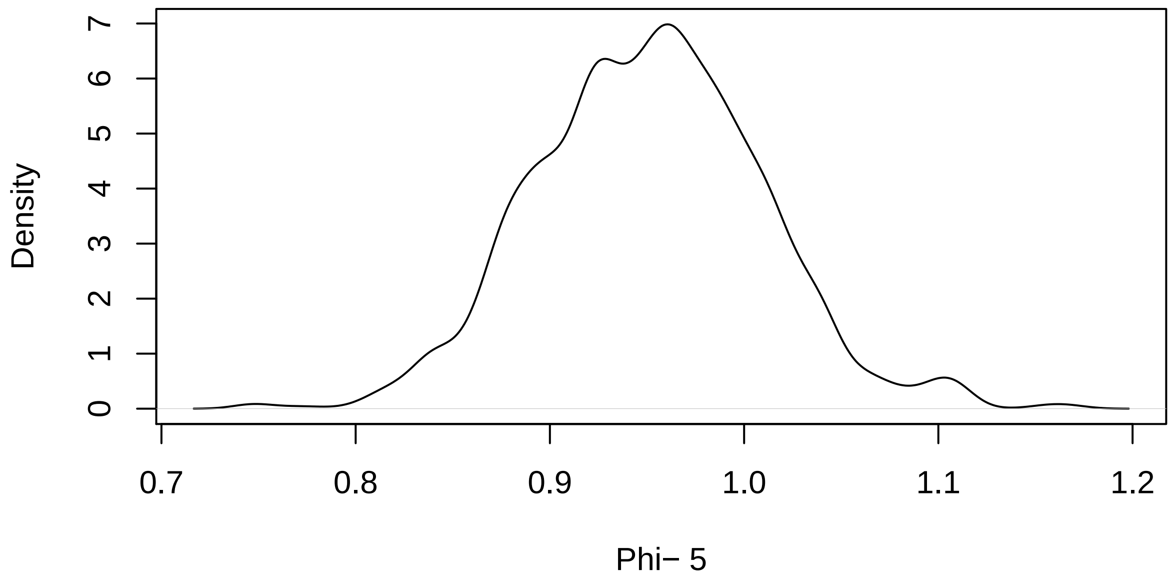

Const
Density
Phi− 1
0
1
2
3
4
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3


PowerSeller
Density
Phi− 2
0
2
4
6
8
10
12
0.85
0.90
0.95
1.00
1.05
1.10

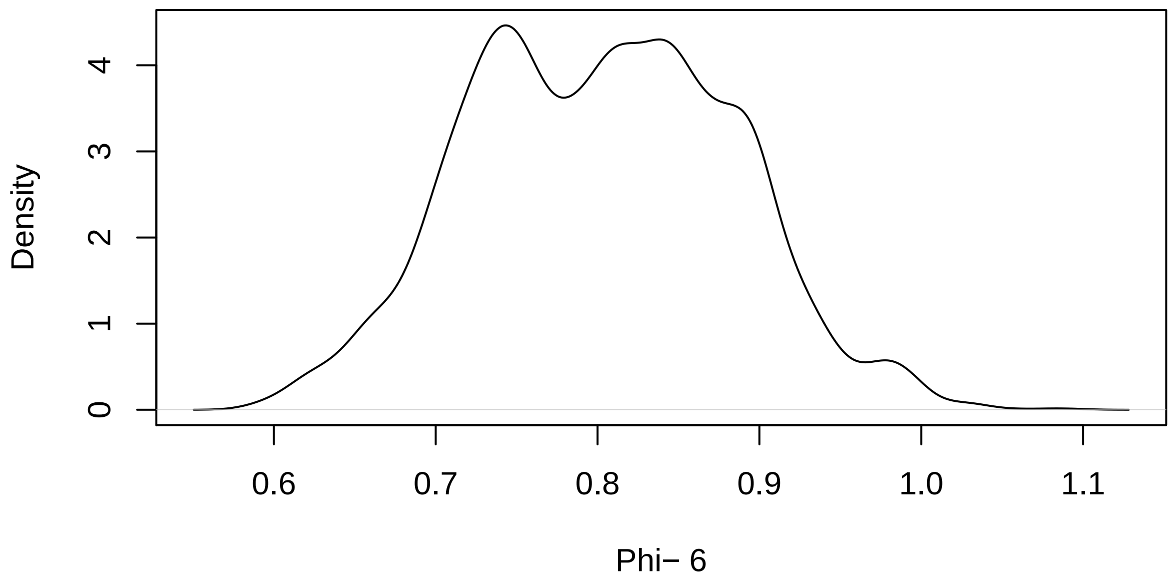

VerifyID
Density
Phi− 3
0
1
2
3
4
5
6
7
0.5
0.6
0.7
0.8
0.9

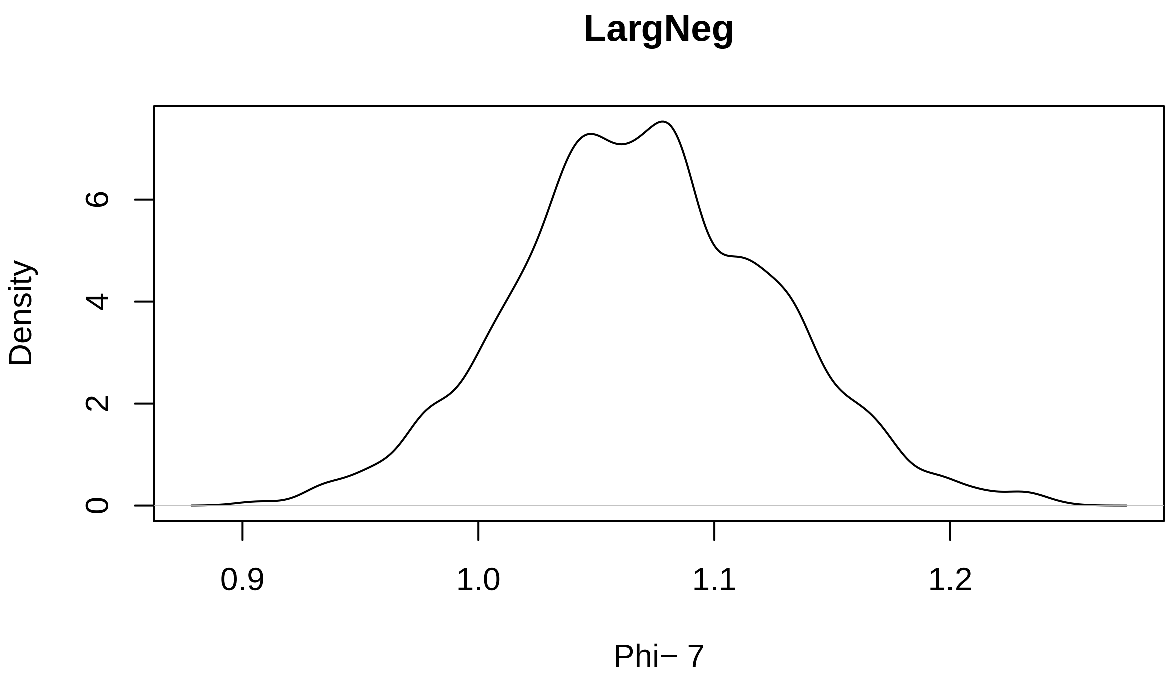

Sealed
Density
Phi− 4
0
1
2
3
4
5
1.3
1.4
1.5
1.6
1.7
1.8

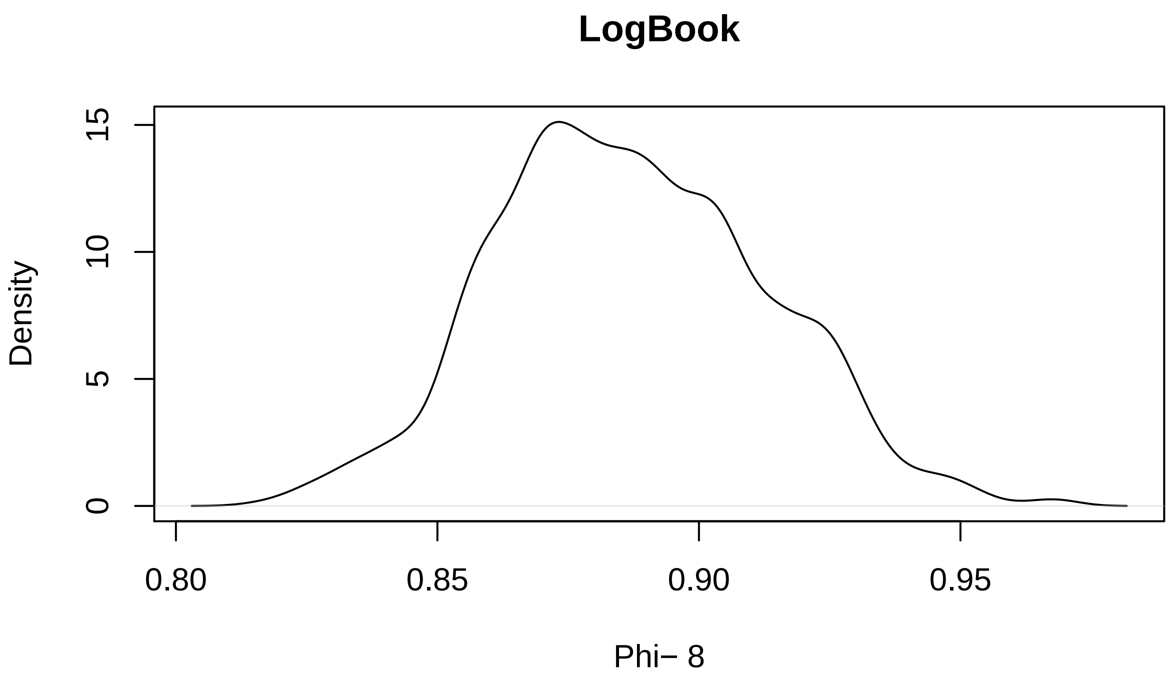
Minblem

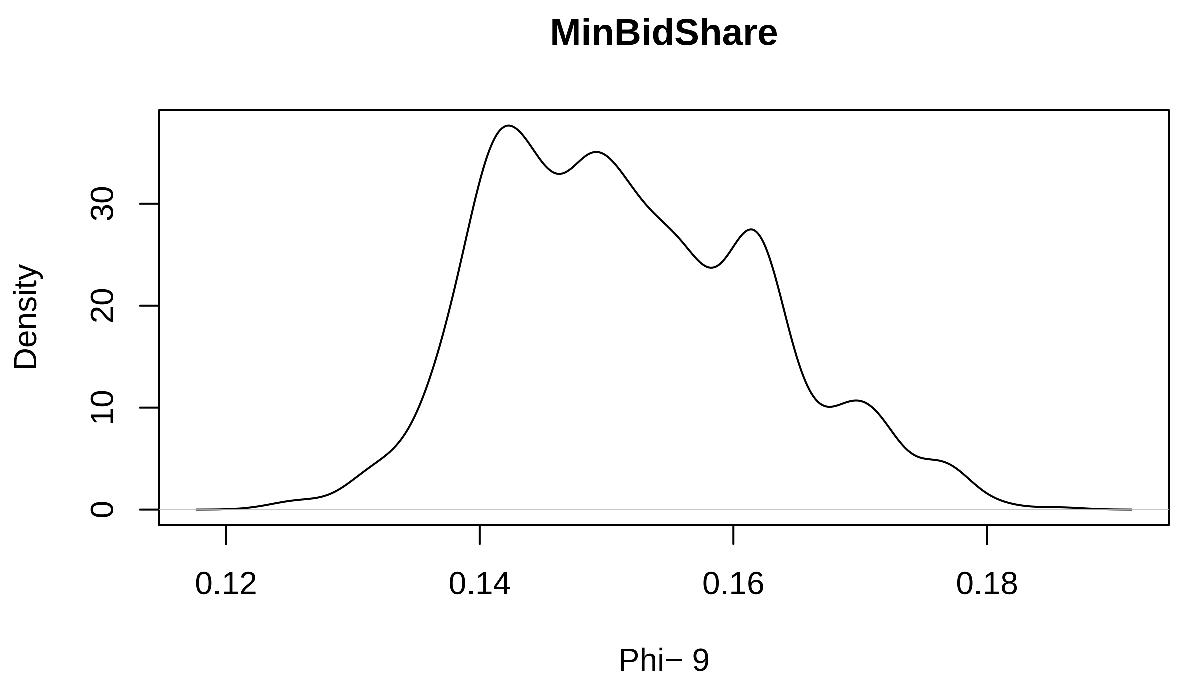


MajBlem




LargNeg
Density
0
2
4
6
0.9
1.0
1.1
1.2
Phi− 7


LogBook
Density
0
5
10
15
0.80
0.85
0.90
0.95
Phi− 8

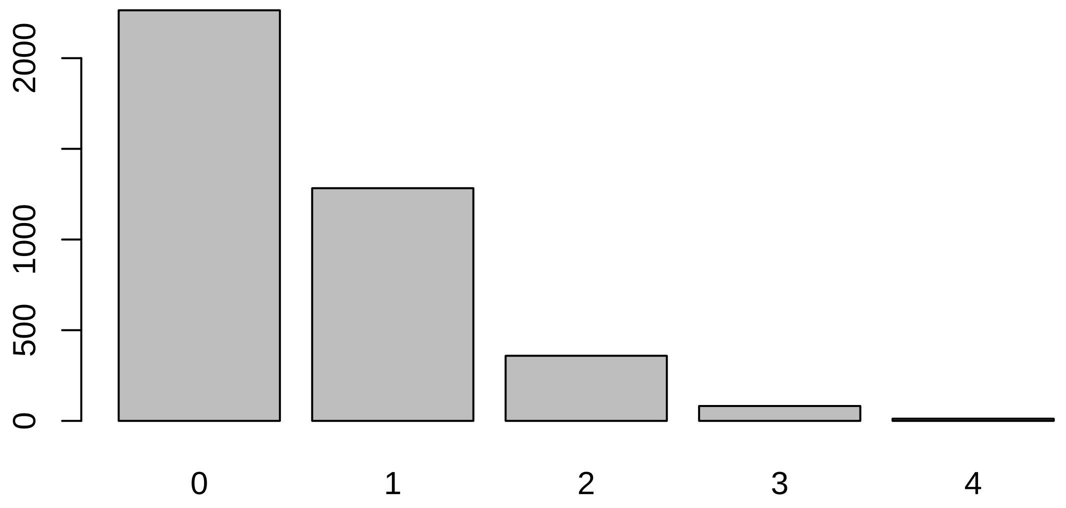


Looking at the above plot for parameters values of different methods, we can see that Poisson Model (or Glm model) and the posterior mode are very much in line and provides similar results as parameter values. On the other hand the posterior mean by MCMC method gives very much different output(or parameters value) when compared with poisson and posterior mode.

Also, we plotted the density of all the phi values to check on the convergence of MCMC .All of them seems to follow normal distributions. We could say that the posterior distributions looks good.

d)

```
# Given
values = c(1, 1, 1, 1, 0, 1, 0, 1, 0.7)

# sampling
Bid_samples = numeric()
for(i in 1:size) {
  Bid_samples[i] = rpois(1, exp(values %*% Beta_samples[i,]))
}

# Plot of prdictive distribution :
barplot(table(Bid_samples))
```



```
# probabily of no bidders in this new auction is given by :
probab = sum(Bid_samples == 0) / size # 0.56

cat("\n The Probability of no (zero) bidders in this new auction  : \n",probab,"\n")
```

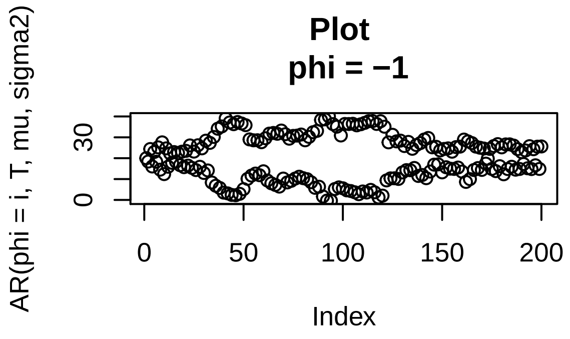
```
##
##  The Probability of no (zero) bidders in this new auction  :
##  0.566
```

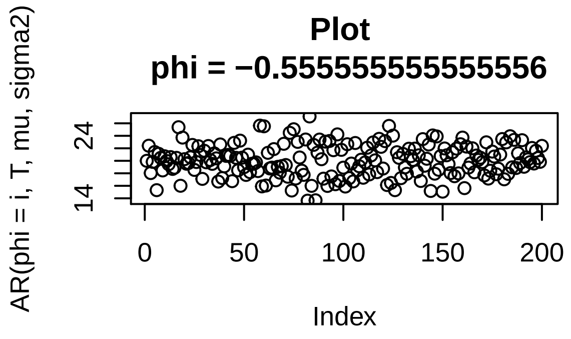
3. Time series models in Stan

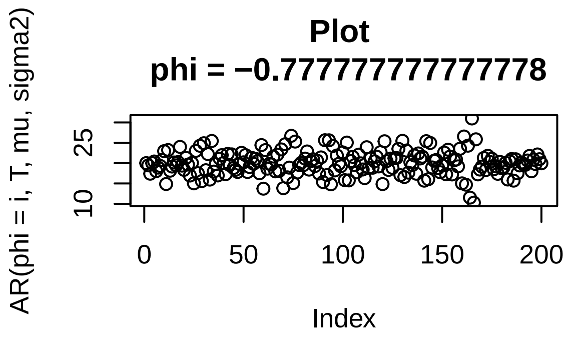
a)

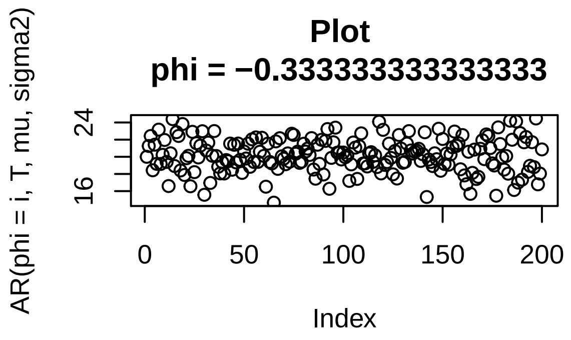
```
set.seed(12345)
# Given
mu<-20
sigma2<-4
T <-200

# Phi
phi_seq<-seq(from=-1,to=1,length.out =10)

# function
AR<-function(phi,T,mu,sigma2){
x<-c()
x[1]<-mu
for(t in 2:T){
x[t]<-mu+phi*(x[t-1]-mu)+rnorm(1,0,sqrt(sigma2))
}
return(x)}

# Data
result<-matrix(nrow = 200,ncol=10)
for(k in 1:10){
for (i in phi_seq){
  result[,k]<-AR(phi=i,T,mu,sigma2)
}}
```

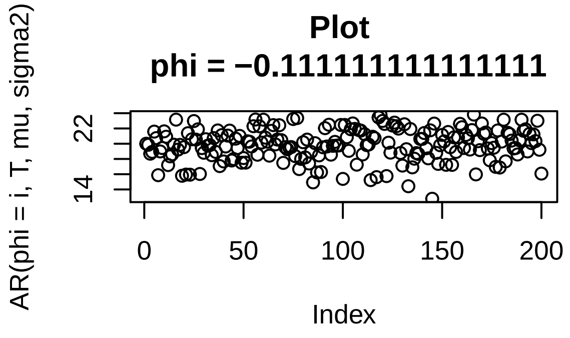
```r
#Plot realizations
par(mfcol =c(2,2))
for (i in phi_seq){
  plot(AR(phi=i,T,mu,sigma2),type="p",main= paste("Plot \n phi =", i) )
}
```

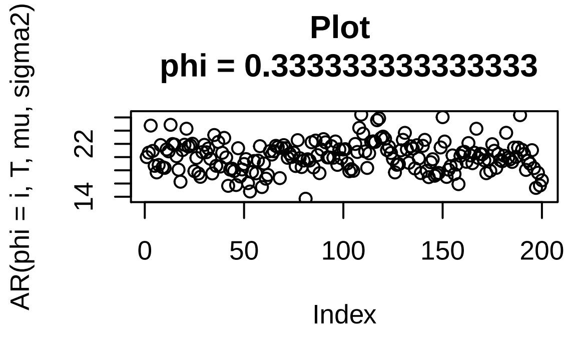


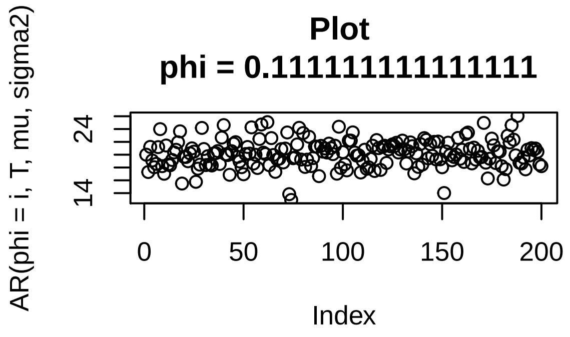


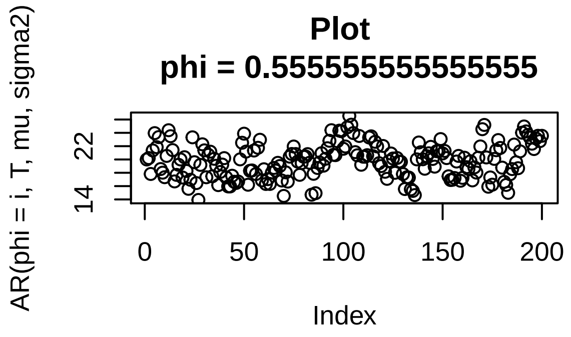


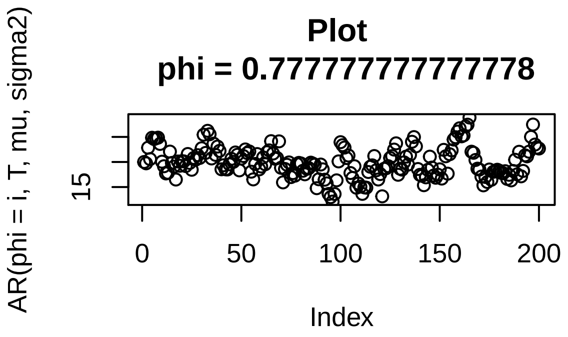


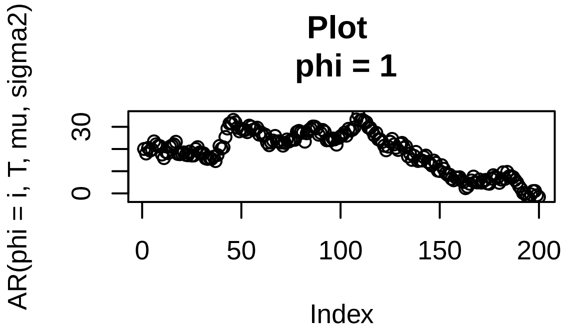












Looking at the plots above for various values of phi, we can say that when phi is extreme negative i.e -1, it gives us a inverted trend wherein the plot starts around the mean region and expands away from the mean and then it again alters towards the mean. For rest of the negative to positive values , the x(t) values are scattered and alernating more around the mean. For extreme positive value of phi ie +1, x(t) values follows a local upward and then downward trend, which is clear from the plot.

b)

```
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000139 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.39 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: 0.469177 seconds (Warm-up)
## Chain 1:                0.109162 seconds (Sampling)
## Chain 1:                0.578339 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.23 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: 0.209542 seconds (Warm-up)
## Chain 2:                0.114654 seconds (Sampling)
## Chain 2:                0.324196 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.22 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: 0.613674 seconds (Warm-up)
## Chain 3:                0.116637 seconds (Sampling)
## Chain 3:                0.730311 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.22 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: 0.138477 seconds (Warm-up)
## Chain 4:                0.093629 seconds (Sampling)
## Chain 4:                0.232106 seconds (Total)
## Chain 4:

##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.25 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: 0.277553 seconds (Warm-up)
## Chain 1:                0.090714 seconds (Sampling)
## Chain 1:                0.368267 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.21 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: 0.20995 seconds (Warm-up)
## Chain 2:                0.098833 seconds (Sampling)
## Chain 2:                0.308783 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.22 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: 0.281804 seconds (Warm-up)
## Chain 3:                0.122433 seconds (Sampling)
## Chain 3:                0.404237 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '09a37f652adfb58b425710d822b812d2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.2 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: 0.291593 seconds (Warm-up)
## Chain 4:                0.108212 seconds (Sampling)
## Chain 4:                0.399805 seconds (Total)
## Chain 4:

## Warning: There were 3 divergent transitions after warmup. See
## http://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

##
##  The True value estimations for phi =0.3 :
##  mean : 19.85389
##  phi : 0.3045181
##  sigma : 2.145861

##
##  The True value estimations for phi =0.9 :
##  mean : 20.09558
##  phi : 0.8662347
##  sigma : 2.009574
```

ii) Joint Posterior density of of mu and phi for both data sets

The data set gives reasonable probability when calculating alpha with phi values close to 0.9 to 0.95 as we can see from the plots of sample data set that there are many data points below mu=19 (estimated mu1=19.8 and estimated mu2= 20.1) and hence the estimated mu seems to be reasonable values for both the phi values.

Further,looking at the posterior density plots, we can say that when phi is 0.3 the posterior density is closely packed and follows the normal distribution. Whereas when phi is 0.9 it looks like the posterior density curve has the normal distribution with skewed ends (when phi value is around 0.95). This is because of the high likelihood of samples selection irrespective of the sampled mu values.

Contribution :

Biswas contributed majorly with Assignment # 1 and #2, report writing and overall trouble-shooting. Gowtham contributed majorly with Assignment #3. Both team members discussed on solution approach, expected outcomes, and revision for the submitted comments of all the assignments.

Note : We have referred lecture notes, our group's previous submission, received comments analysis and R -documentation

Code Appendix

```r
plot(rep(0, 9), ylim = c(-0.015, 0.015), col="blue",
```