Computer Lab 3

Ravinder Reddy Atla (ravat601) Hoda Fakharzadehjahromy (hodfa840)

5/5/2021

1. Gibbs Sampler for a normal model

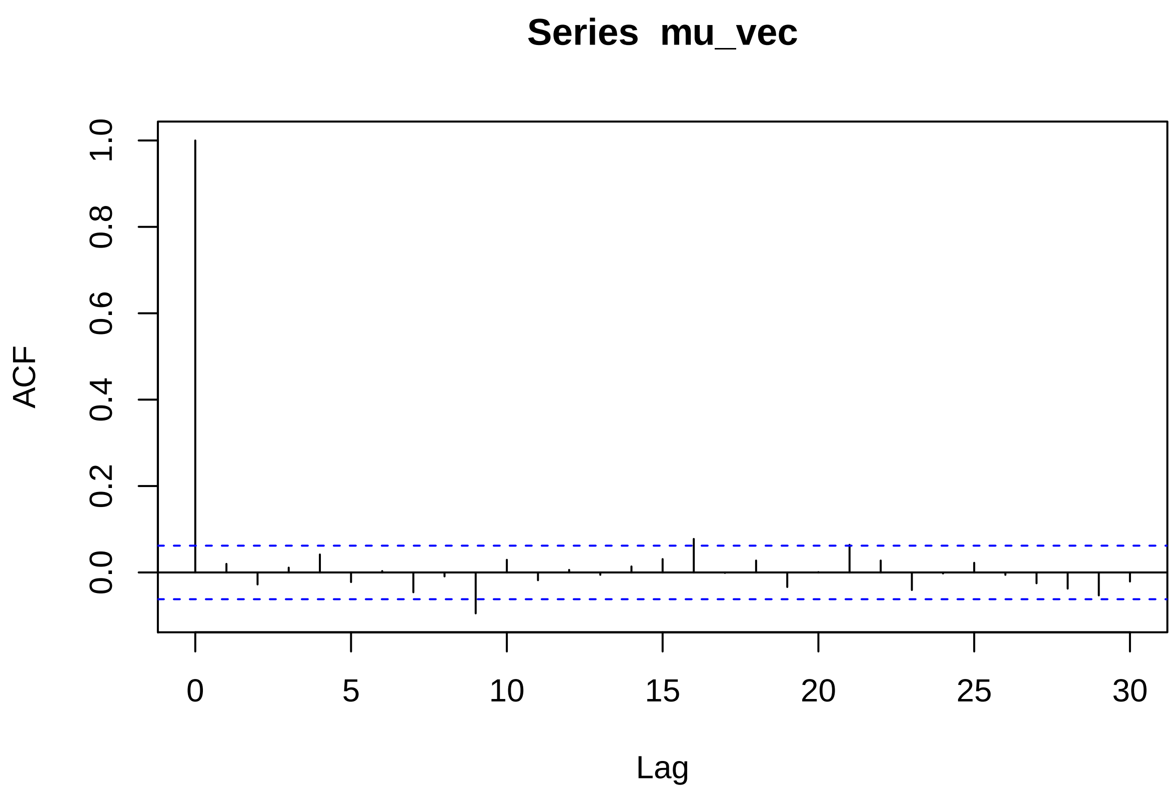
```
rainfall_data <- read.table('rainfall.dat')
```

(a) Simulation of Gibbs sampler from joint posterior distribution

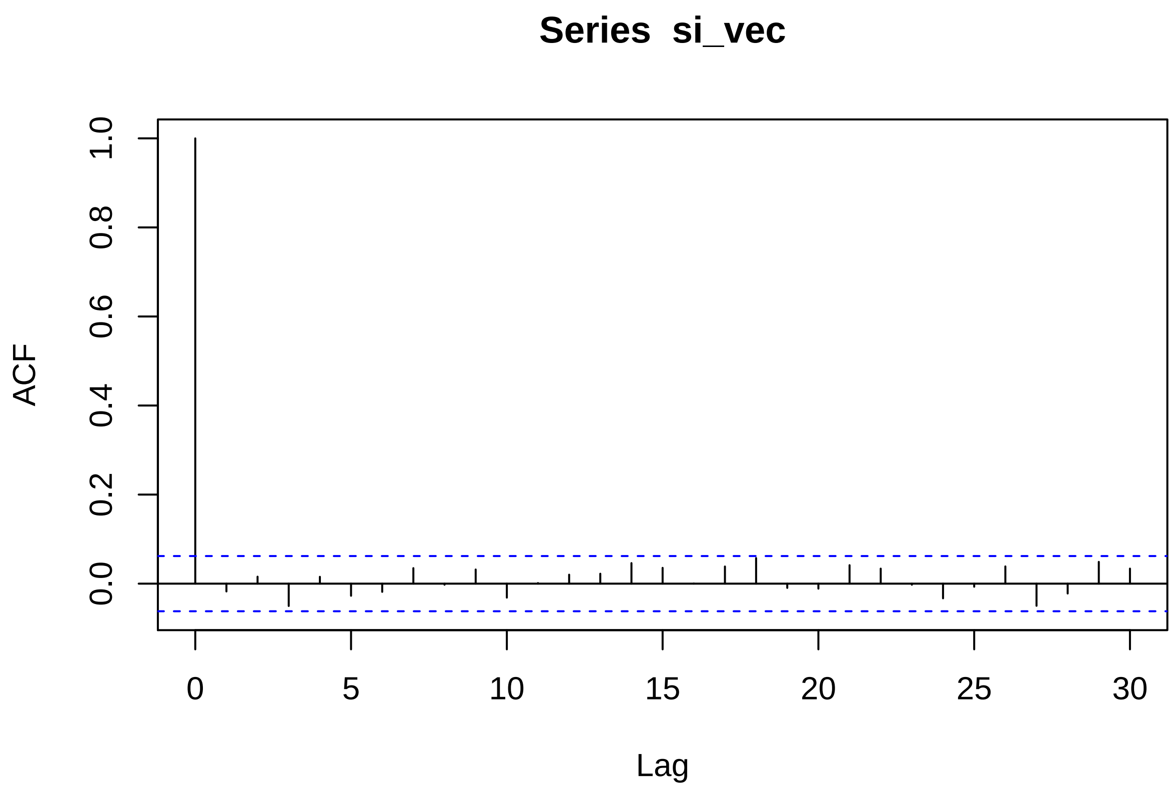
```
# Data and prior Initialization
log_y <- as.matrix(log(rainfall_data))
mu0 <- 0
tau0_2 <- 1
v0 <- 1
si0_2 <- 1

# Initial value
si_2 <- 1
n_iterations <- 1000
mu_vec <- rep(NA,n_iterations)
si_vec <- rep(NA,n_iterations)
logy_post <- rep(NA,n_iterations)
n <- length(log_y)

# Pre calculation
log_y_mean <- mean(log_y)

# Function for getting samples from inverse chisquare distributino
rinvchisquare <- function(num_draws, n, tau_sq){
  #set.seed(1234)
  x <- rchisq(num_draws,df = n)
  x_inv <- ((n)*tau_sq)/x
  return(x_inv)
}

# Gibbs sampling iterations
for(k in 1:n_iterations){
  # Sampling mu
  w <- (n/si_2)/((n/si_2) + (1/tau0_2))
  taun_2 <- w/(n/si_2)
  mun <- (w * log_y_mean) + ((1 - w) * mu0)
  mu <- rnorm(1, mun, sqrt(taun_2))
```

```
  # Sampling si_2
  vn <- v0+n
  sin_2 <- (v0*si0_2 + sum((log_y - mu)^2))/vn
  si_2 <- rinvchisquare(1, vn, sin_2)

  # sample from posterior predictive
  logy_post[k] <- rnorm(1, mu, sqrt(si_2))

  mu_vec[k] <- mu
  si_vec[k] <- si_2
}
```

```
library(MASS)

rho_mu <- acf(mu_vec)
```




```
rho_si <- acf(si_vec)
```



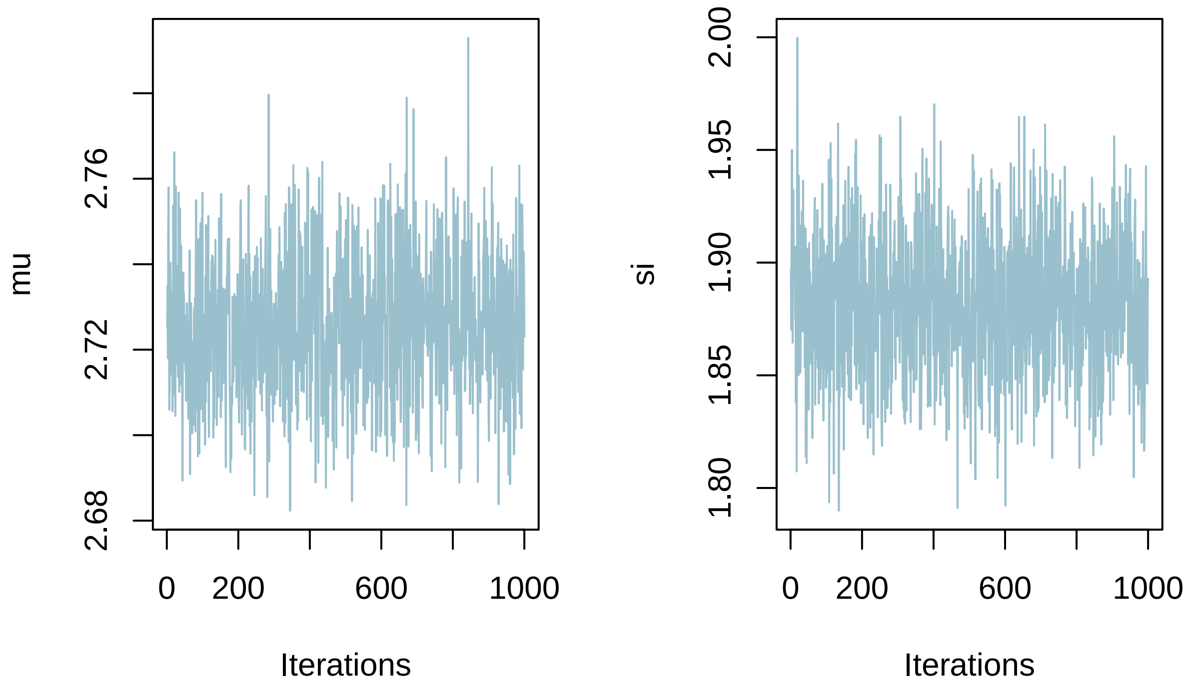

```
IF_mu <- 1 + 2*sum(rho_mu$acf)
IF_si <- 1 + 2*sum(rho_si$acf)

cat('Inefficiency factor for parameter mu : ',IF_mu,'\n')
```

```
## Inefficiency factor for parameter mu :  2.863159
```

```
cat('Inefficiency factor for parameter mu : ',IF_si)
```

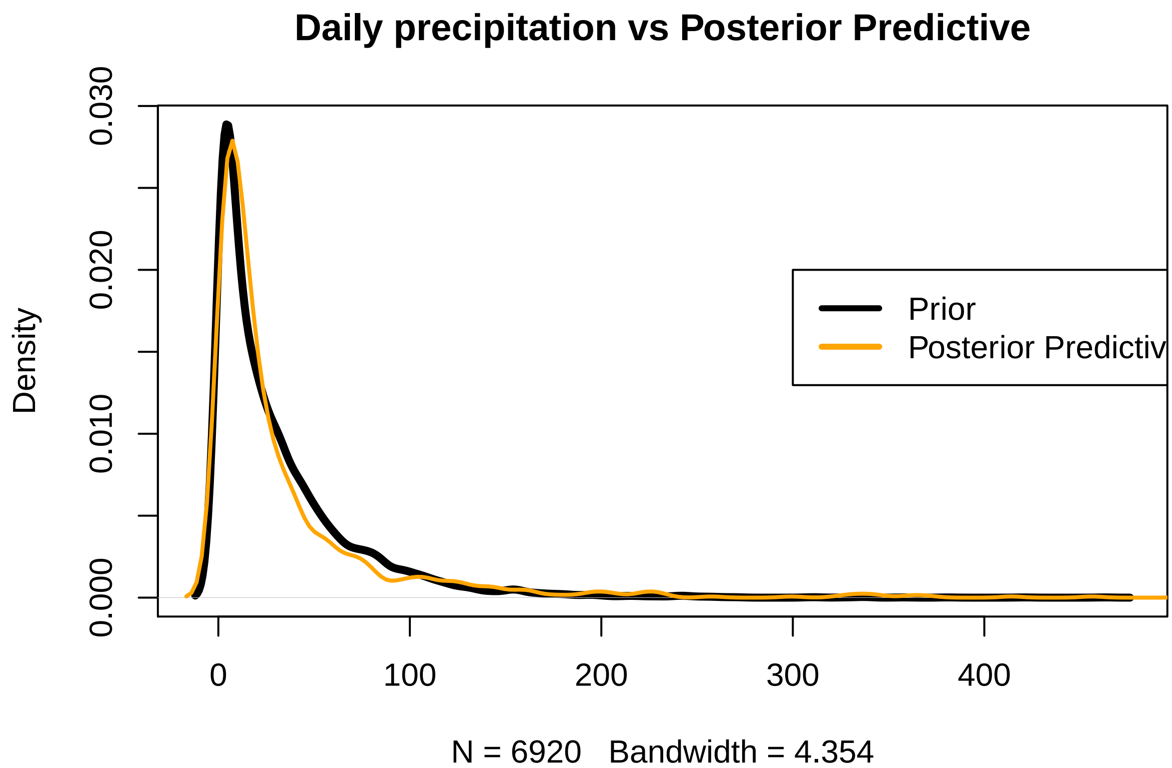
```
## Inefficiency factor for parameter mu :  3.466819
```

```
par(mfrow = c(1,2))
plot(x = 1:n_iterations, y = mu_vec,type = 'l', col = 'lightblue3',
     xlab = 'Iterations', ylab = 'mu')
plot(x = 1:n_iterations, y = si_vec,type = 'l', col = 'lightblue3',
     xlab = 'Iterations', ylab = 'si')
```



(b). Posterior Predictive density

```
y_post <- exp(logy_post)
y_prior <- as.matrix(rainfall_data)

plot(density(y_prior),lwd = 4, main = 'Daily precipitation vs Posterior Predictive')
lines(density(y_post),col = 'orange',lwd = 2)
legend(300, 0.020,
       legend = c('Prior','Posterior Predictive'),
       col = c('black','orange'),
       lty=c(1,1),lwd = 3)
```



2. Metropolis Random Walk for Poisson regression

```
ebay_data <- read.table('ebayNumberOfBidderData.dat',header = TRUE)
rows <- nrow(ebay_data)
cols <- ncol(ebay_data)
y <- as.matrix(ebay_data[1])
X <- as.matrix(ebay_data[,2:cols])
```

(a). Maximum Likelihood Estimator of beta

```
x_glm <- X[,2:ncol(X)]
model <- glm(y ~ x_glm,family = poisson())
print(model$coefficients)
```

```
##      (Intercept) x_glmPowerSeller    x_glmVerifyID      x_glmSealed
##       1.07244206      -0.02054076      -0.39451647       0.44384257
##      x_glmMinblem      x_glmMajBlem      x_glmLargNeg      x_glmLogBook
##      -0.05219829      -0.22087119       0.07067246      -0.12067761
## x_glmMinBidShare
##      -1.89409664
```

As observed from the coefficients of the covariates, MinBidShare coefficient is much lower when compared to the others hence, negatively affecting the prediction. Sealed variable covariate is the only covariate with positive coefficient which might have significant positive effect on regression.

(b). Bayesian Analysis on Poisson Regression

```
library(mvtnorm)
params <- dim(X)[2]
mu <- as.matrix(rep(0,params))
Sigma = 100*solve(t(X)%*%X)

LogPostPoisson <- function(betas,y,X,mu,Sigma){
  n <- nrow(X)
  lamda <- exp(X%*%betas)
  logLik <- (sum((X%*%betas)*y) - sum(lamda))
  Prior <- dmvnorm(betas, mu, Sigma, log=TRUE)
  return(logLik + Prior)
}

initValue <- matrix(0,params,1)

# Optimum beta(coefficient) are calculated
OptimRes <- optim(initValue,
                  LogPostPoisson, gr=NULL, y,X, mu, Sigma, method=c("BFGS"),
                  control=list(fnscale=-1), hessian=TRUE)
```

```
print('Posterior Mode: ')
```

```
## [1] "Posterior Mode: "
```

```
print(OptimRes$par)
```

```
##              [,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('Inverse of hessian matrix')
```

```
## [1] "Inverse of hessian matrix"
```

```
inversehessian <- solve(-OptimRes$hessian)
print(inversehessian)
```

```
##                [,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
```

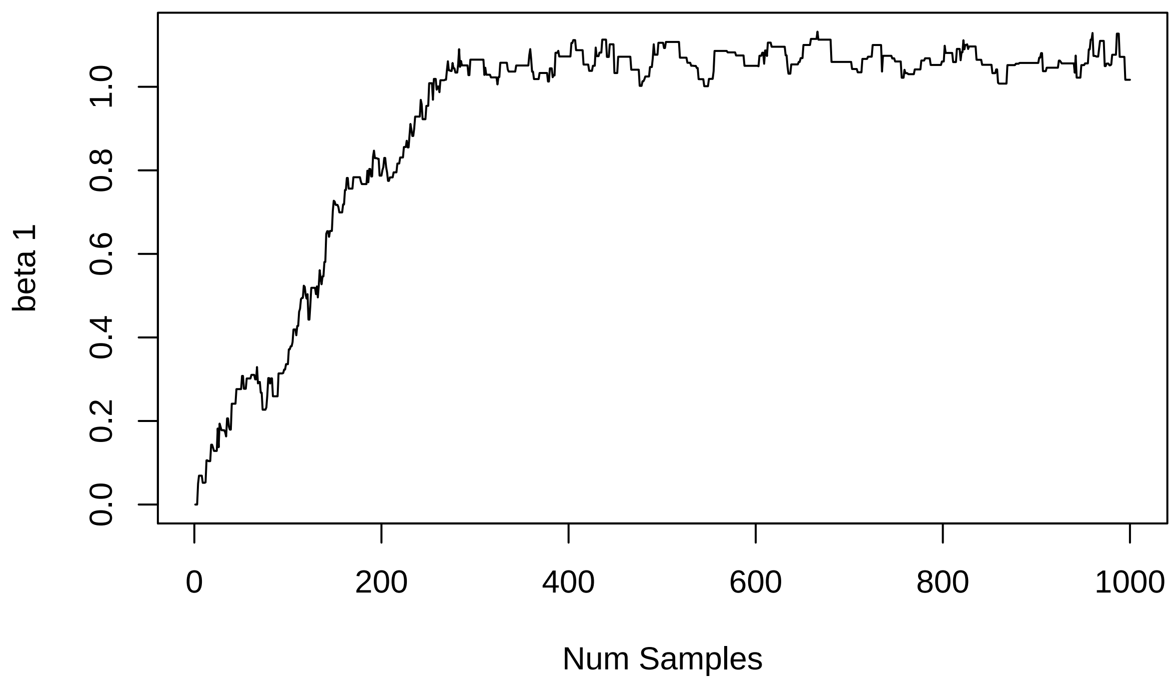
(c). Metropolis Algorithm

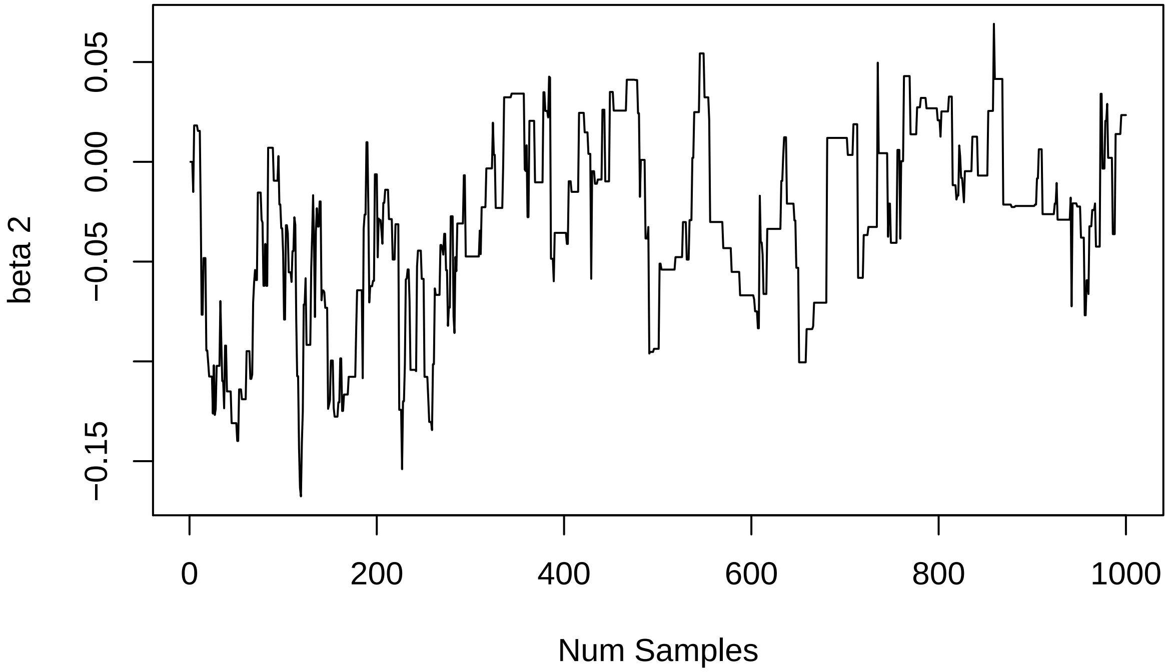
```
logPosteriorPoisson <- function(betas){
  params <- ncol(X)
  mu <- as.matrix(rep(0,params))
  Sigma = 100*solve(t(X)%*%X)

  n <- nrow(X)
  lamda <- exp(X%*%betas)
  logLik <- (sum((X%*%betas)*y) - sum(lamda))
  Prior <- dmvnorm(t(betas), mu, Sigma, log=TRUE)
  return(logLik + Prior)
}

sampleFromPosterior <- function(num_iterations, theta, c, postDensityFun){
  theta_current <- theta
  samples <- matrix(theta_current, num_iterations, nrow(theta))
  accept <- c()
  accept[1] <- 1
  for(i in 2:num_iterations){
    #shift <- as.vector(rmvnorm(1, mean = samples[i-1], sigma = c*inversehessian))
    #theta_new <- samples[i-1,] + shift
    theta_new <- as.vector(rmvnorm(1, mean = samples[i-1,], sigma = c*inversehessian))

    p_log_target_val <- postDensityFun(theta_new)
    log_target_val <- postDensityFun(samples[i-1,])

    alpha = min(1, exp(p_log_target_val - log_target_val))
    if(runif(1) <= alpha){
      #theta_current <- theta_new
      samples[i,] <- theta_new
      accept[i]<-1
    }
    else{
      samples[i,] <- samples[i-1,]
      accept[i]<-0
    }
  }
  print(sum(accept)/n_iterations)
  return(samples)
}
set.seed(1234)
```

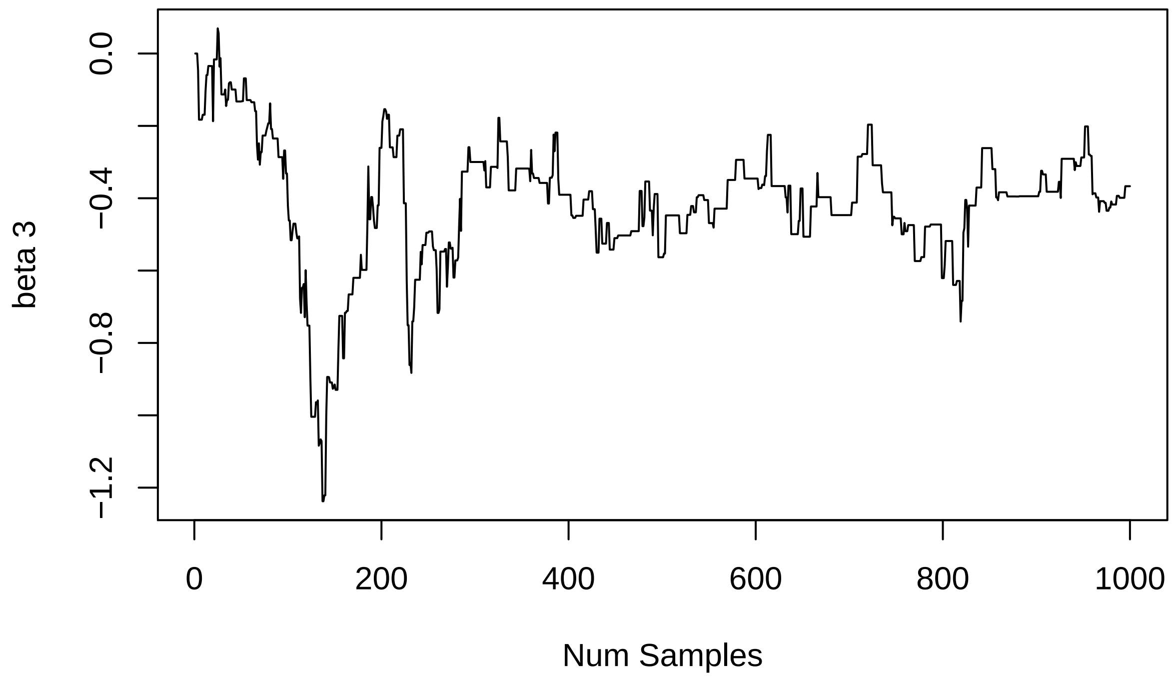
```r
beta_init <- matrix(0,dim(X)[2],1)
iter = 1000
sample_bet <- sampleFromPosterior(iter, beta_init, 0.8, logPosteriorPoisson)
```

```
## [1] 0.291
```

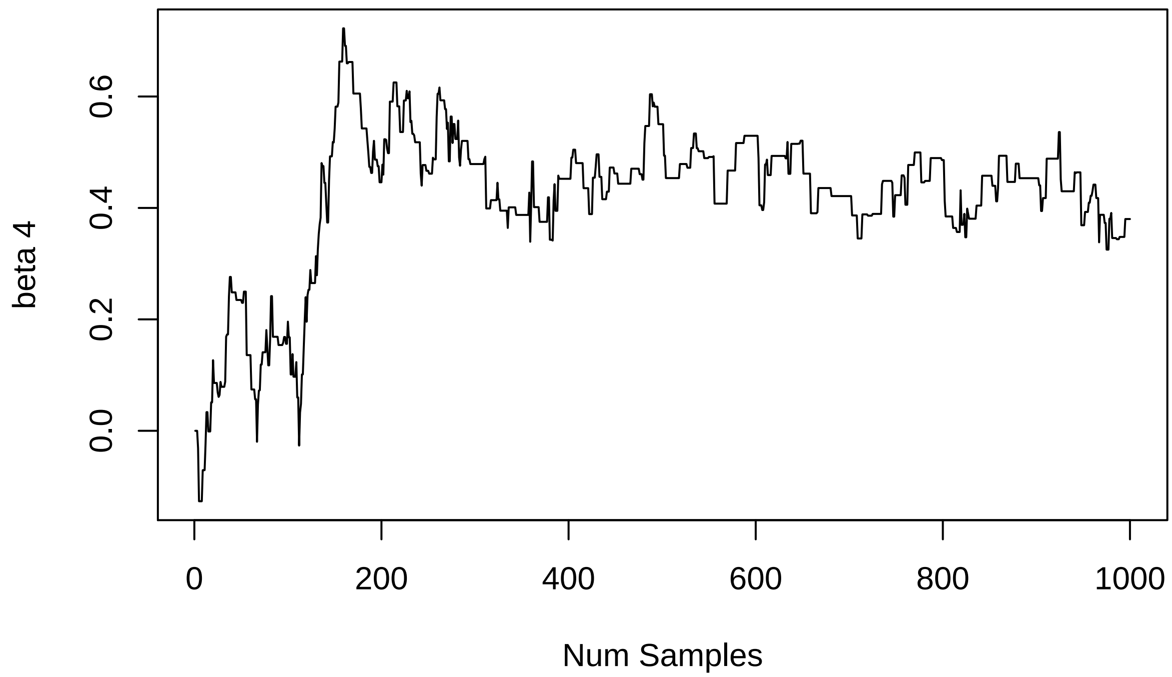
```r
for(i in 1:ncol(sample_bet)){
  plot(c(1:iter),sample_bet[,i],'l',
       ylab = paste('beta',i), xlab = 'Num Samples')
}
```

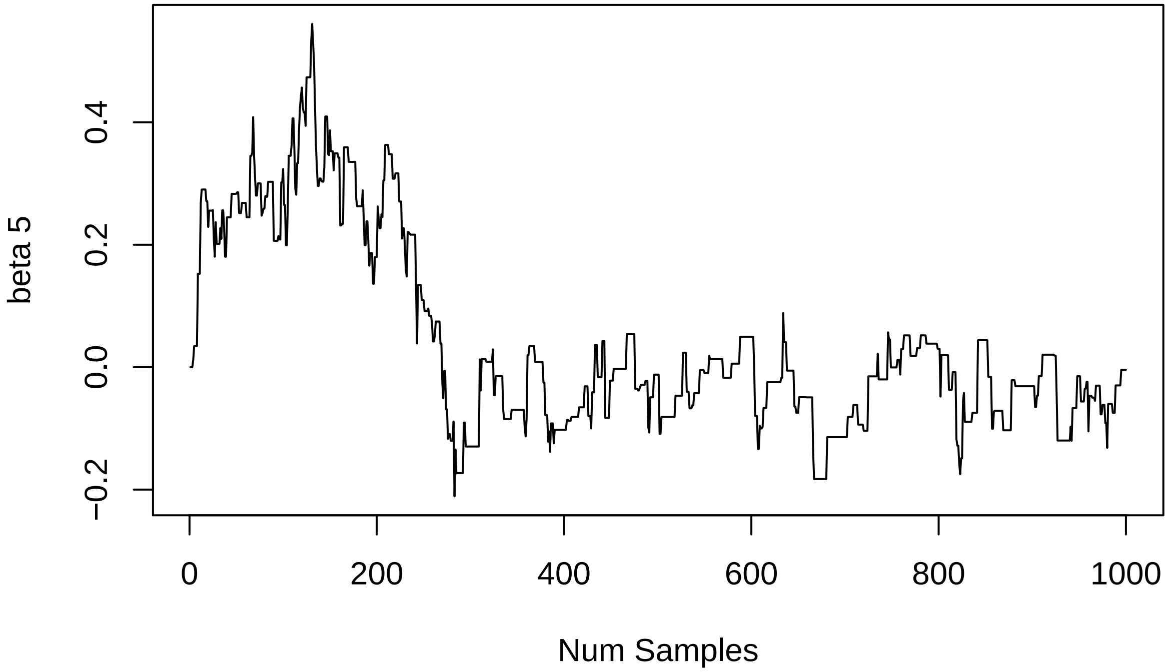


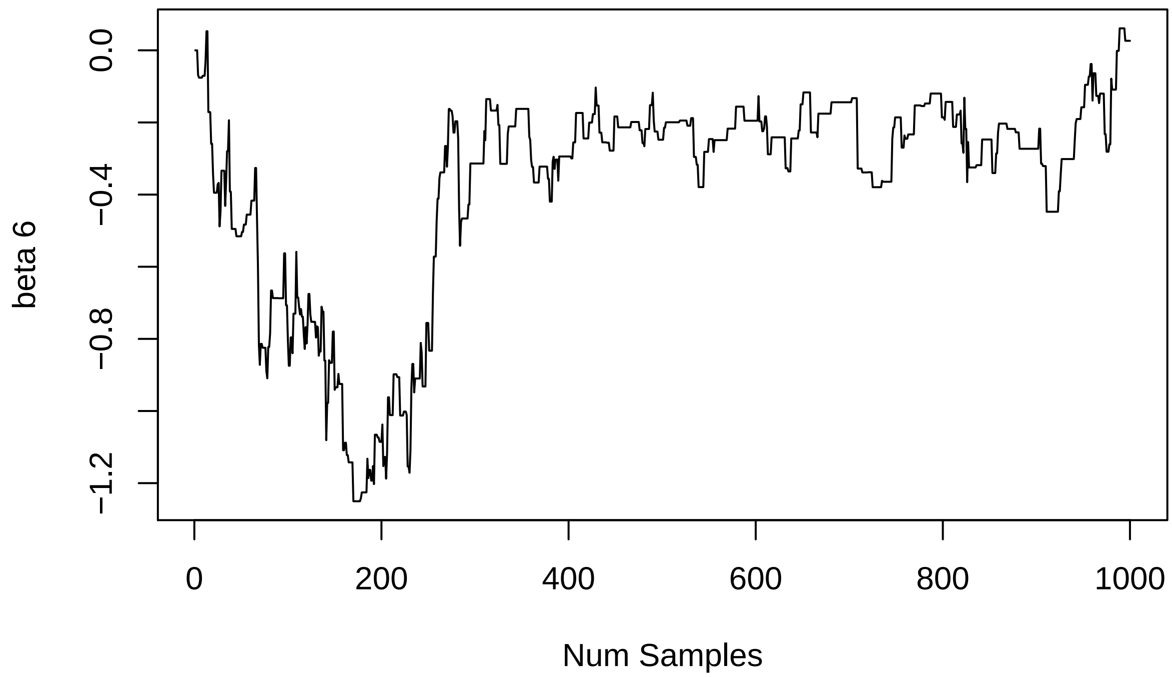

beta 2
0.05
0.00
−0.05
−0.15
0
200
400
600
800
1000
Num Samples



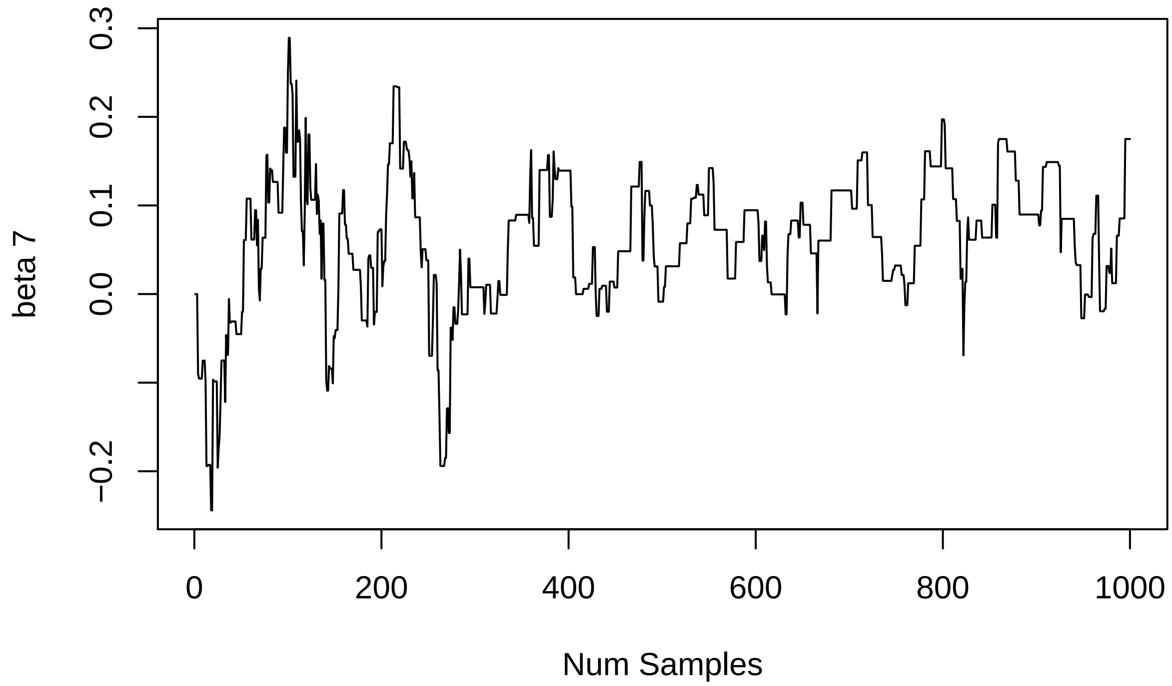
beta 3
0.0
−0.4
−0.8
−1.2
0
200
400
600
800
1000
Num Samples

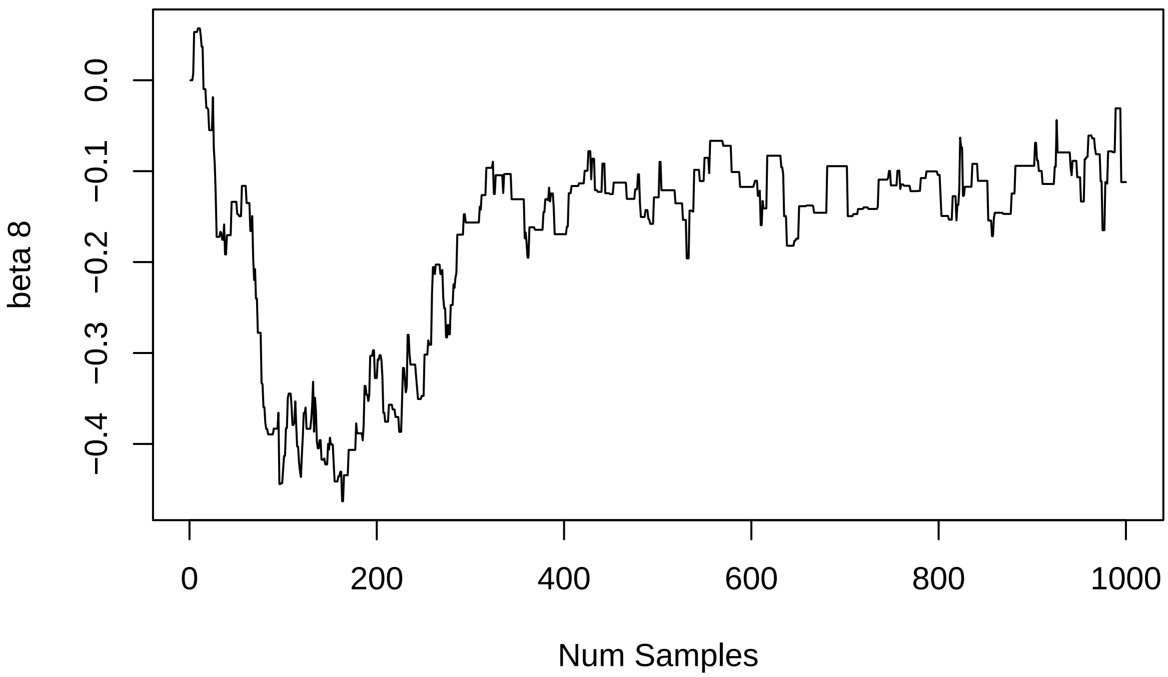

beta 4
0.0
0.2
0.4
0.6
0
200
400
600
800
1000
Num Samples

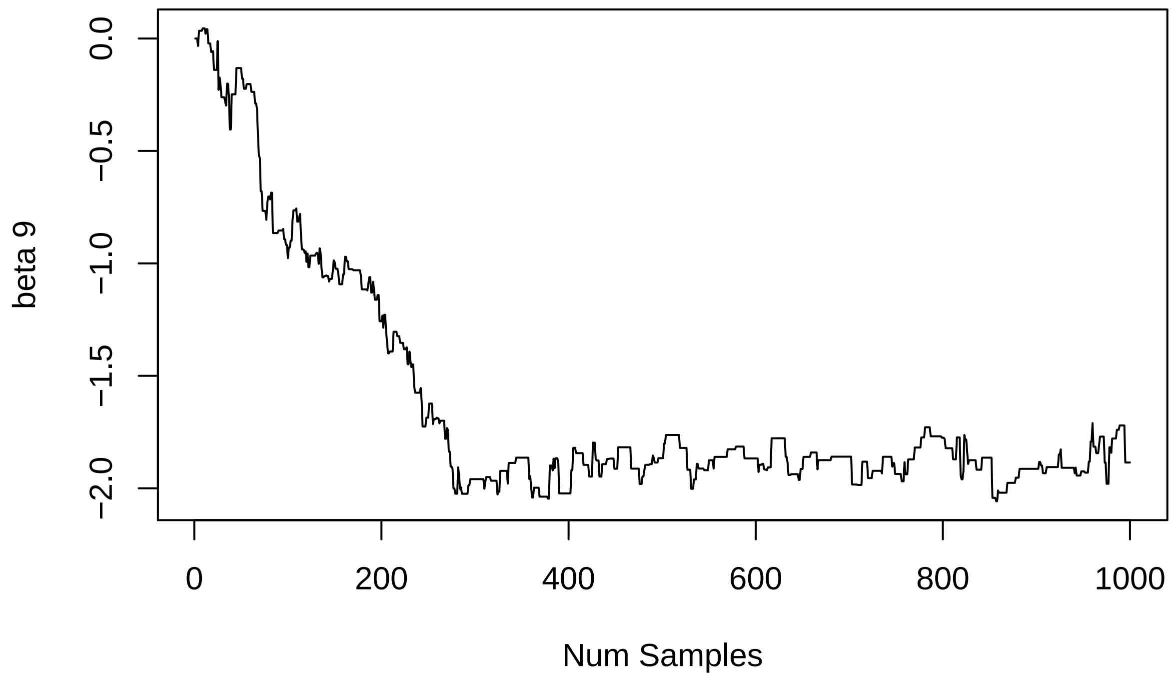

beta 5
−0.2
0.0
0.2
0.4
0
200
400
600
800
1000
Num Samples



0.0
−0.4
−0.8
−1.2
beta 6
0
200
400
600
800
1000
Num Samples

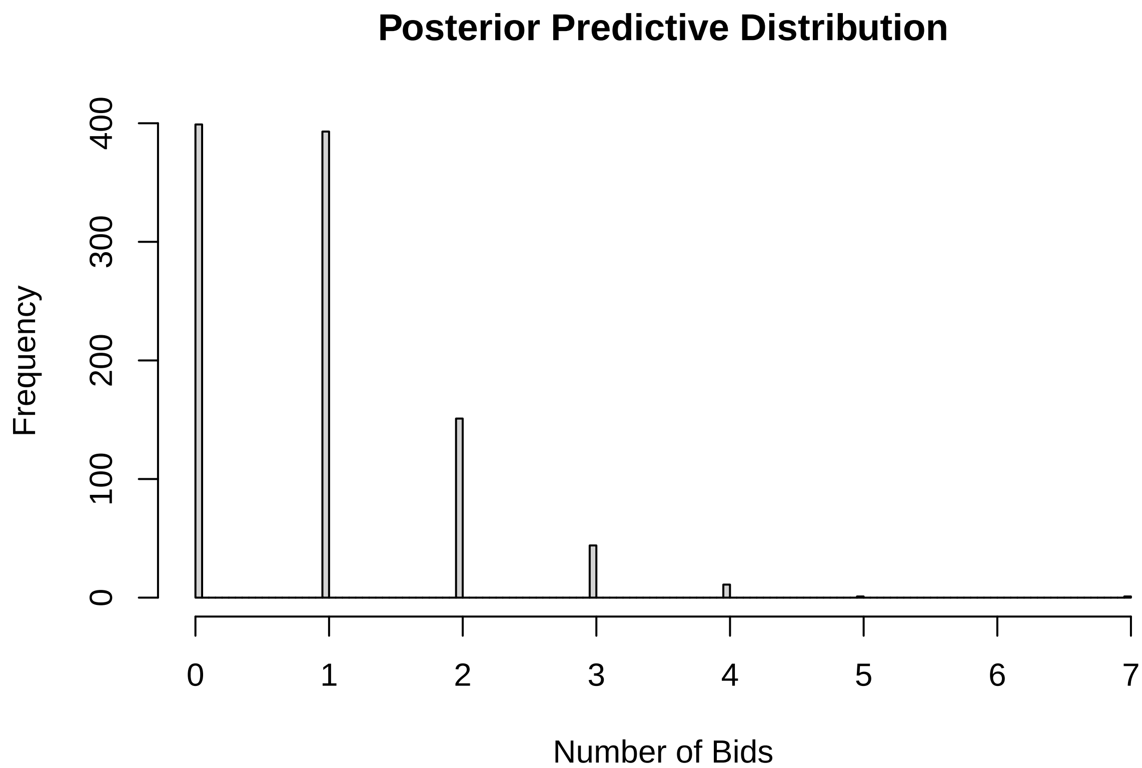

beta 7
0.3
0.2
0.1
0.0
−0.2
0
200
400
600
800
1000
Num Samples


beta 8
0.0
−0.1
−0.2
−0.3
−0.4
0
200
400
600
800
1000
Num Samples



(d)

```
input <- c(1,1,1,1,0,1,0,1,0.7)
pred <- c()
n_samp <- 1000
for(i in 1:n_samp){
  pred[i] <- rpois(1,lambda = input*colMeans(sample_bet))
}
hist(pred, breaks = 100, xlab = 'Number of Bids',
     main = 'Posterior Predictive Distribution')
```




```
zero_bid <- length(pred[pred==0])
zero_bid_prob <- zero_bid/n_samp
print(paste('Probability of no bidders in the new auction is:',zero_bid_prob))
```

```
## [1] "Probability of no bidders in the new auction is: 0.399"
```

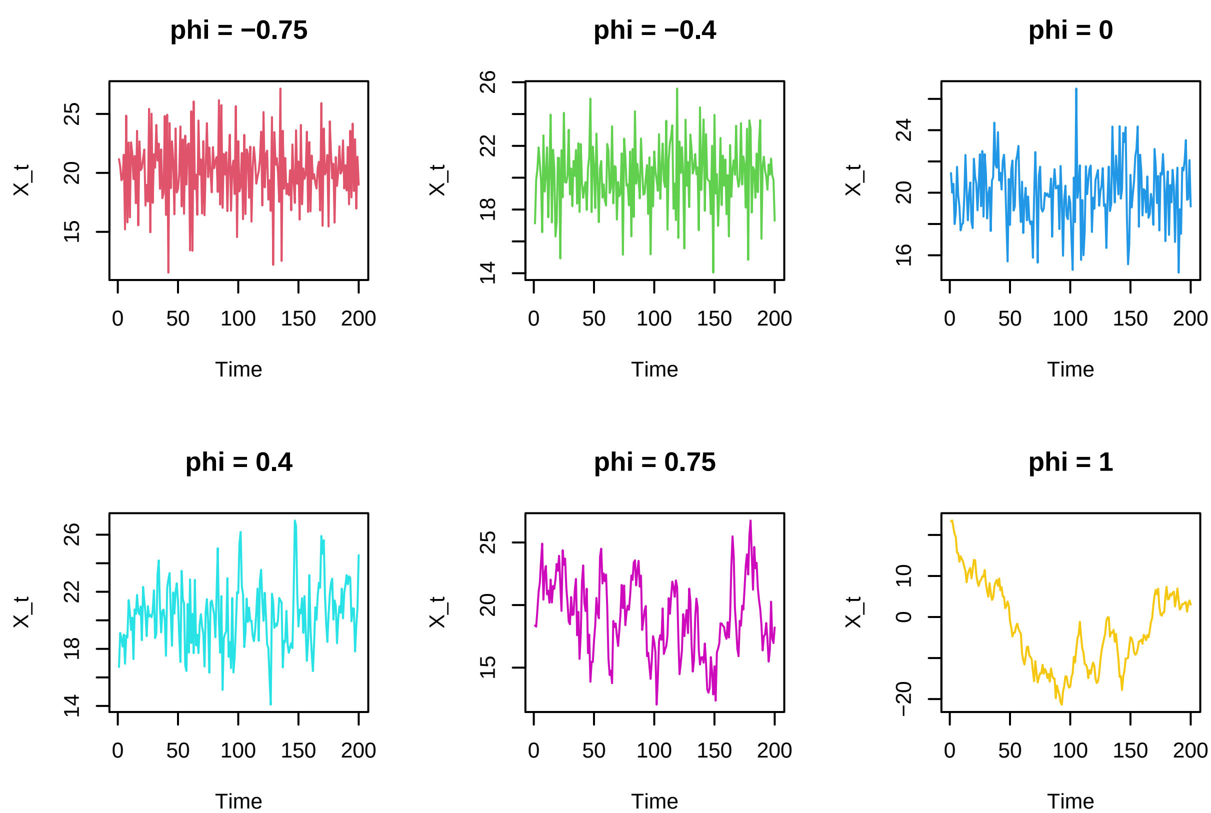
3. Time Series Models in Stan

(a). Simulate from AR process (R):

We assume $\mu \sim \mathcal{N}(0, 1000)$ and $\sigma^2 \sim Inv\ \chi^2(1000, 2)$ prior for μ and σ

```
library(rstan)

mu = 20
sigma2 = 4
nT = 200

AR.1 <- function(phi,mu=20,sigma2=4,nT=200) {

  x=numeric(nT)
  x[1]=mu + rnorm(n = 1,sd = sqrt(sigma2),mean = 0)
  for (i in 2:nT) {
    x[i]=mu+phi*(x[i-1]-mu)+ rnorm(1,0,sqrt(sigma2))
  }
  return(x)
```

```
}

phi <- c(-0.75, -.4,0, 0.4, 0.75,1)
set.seed(12345)
simulation = data.frame(sapply(phi, AR.1))

par(mfrow=c(2,3))

for(i in 1:length(simulation)){

  plot(simulation[,i],x=1:nT,type='l',col=i+1,
       xlab='Time',ylab='X_t',
       main=paste(expression(phi),"=",phi[i]))
}
```



Based on the plot ϕ is the parameter that detrmine the amount of dependence to the previous iterations. when $\phi = 0$ all the terms seem to be independent $\phi = 1$ means that every point of the time series will be heavily dependent on each other.In fact we can almost observe a linear trend for $phi = 1$.

(b). Simulate two AR(1)-processe:

```
#b

x <- AR.1(phi=0.3)
y <- AR.1(phi=0.9)
```

```
StanModel='
data {
  int<lower=0> nT;
  vector[nT] x;
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real<lower=-1, upper=1> phi;
}
model {
  mu ~ normal(0,1000);
  sigma2 ~ scaled_inv_chi_square(1000,2); //sigma2 initial value is 4
  x[2:nT] ~ normal( mu + phi*(x[1:(nT-1)]-mu), sqrt(sigma2)); //
}'

fit.x <- stan(
  model_code = StanModel,
  data = list(N = length(x), d = x),    # named list of data
  chains = 1,
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  refresh = 0
)
fit.y <- stan(

  model_code = StanModel,
  data = list(N = length(y), d = y),    # named list of data
  chains = 1,
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  refresh = 0
)

fit.x
```

```
## Inference for Stan model: 6eace9813df20597ead8597c14e4ada5.
## 1 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##            mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff
## mu        19.65    0.01 0.20    19.27    19.52    19.65    19.78    20.03   950
## sigma2     4.01    0.01 0.16     3.71     3.90     4.00     4.12     4.33   924
## phi        0.29    0.00 0.07     0.15     0.24     0.30     0.34     0.44  1022
## lp__   -1432.33    0.06 1.20 -1435.07 -1432.98 -1432.03 -1431.44 -1430.91   474
##        Rhat
## mu     1.00
## sigma2 1.01
## phi    1.00
## lp__   1.00
##
## Samples were drawn using NUTS(diag_e) at Wed May 19 22:08:36 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
```

```
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

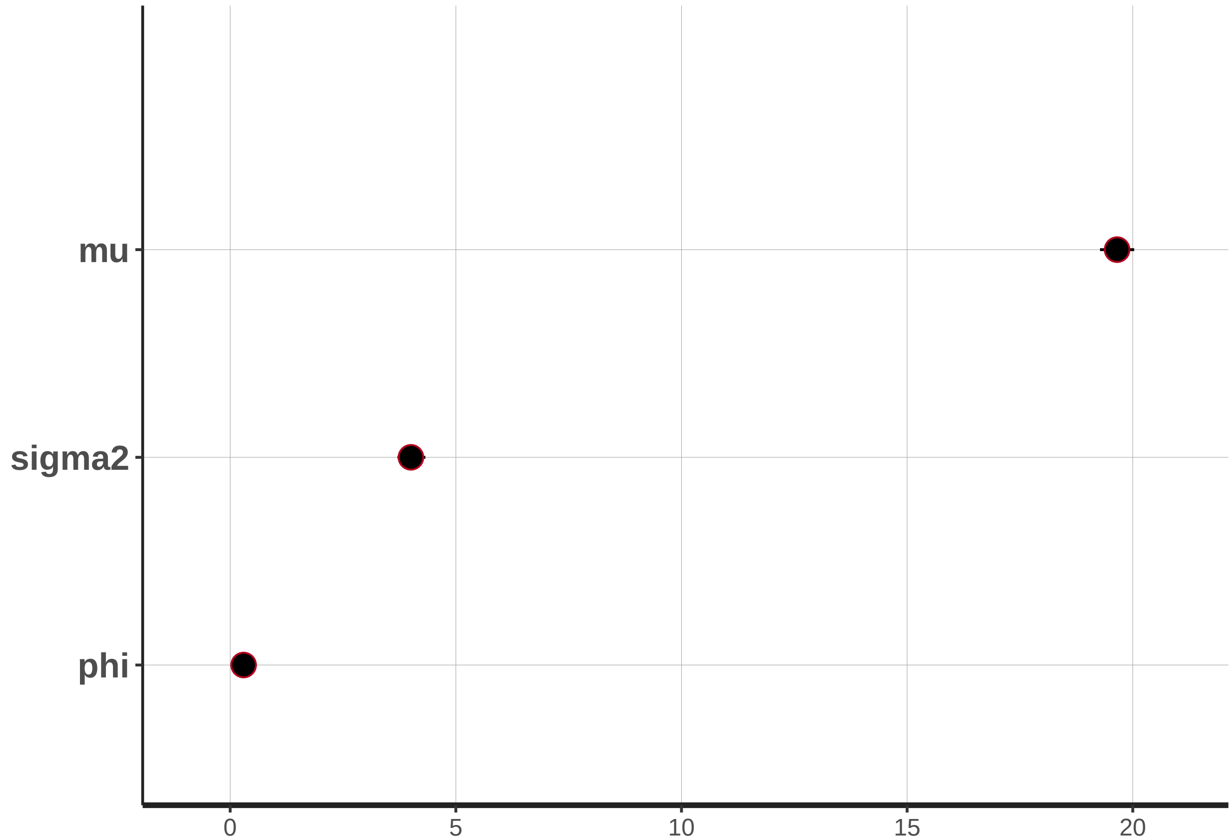
```
fit.y
```

```
## Inference for Stan model: 6eace9813df20597ead8597c14e4ada5.
## 1 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=1000.
##
##            mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff
## mu        19.65    0.01 0.21    19.24    19.51    19.65    19.79    20.03   752
## sigma2     4.01    0.01 0.17     3.70     3.89     4.00     4.12     4.37  1069
## phi        0.30    0.00 0.07     0.16     0.25     0.30     0.35     0.44   956
## lp__   -1432.40    0.06 1.31 -1436.01 -1432.96 -1432.06 -1431.45 -1430.94   479
##        Rhat
## mu        1
## sigma2    1
## phi       1
## lp__      1
##
## Samples were drawn using NUTS(diag_e) at Wed May 19 22:08:38 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

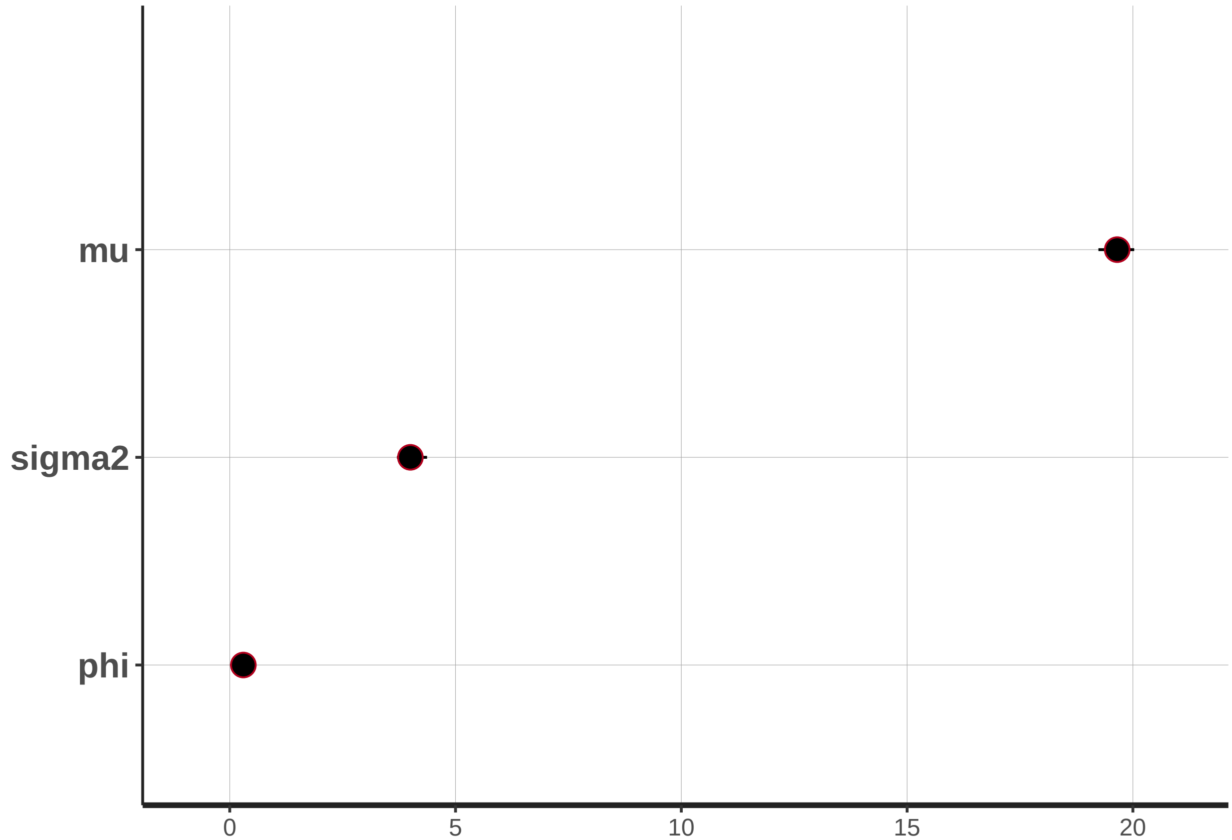
```
xDraws = extract(fit.x)
yDraws = extract(fit.y)
```

From the model summary we can see that all the parameters are fairly close to our set values.

```
plot(fit.x)
```



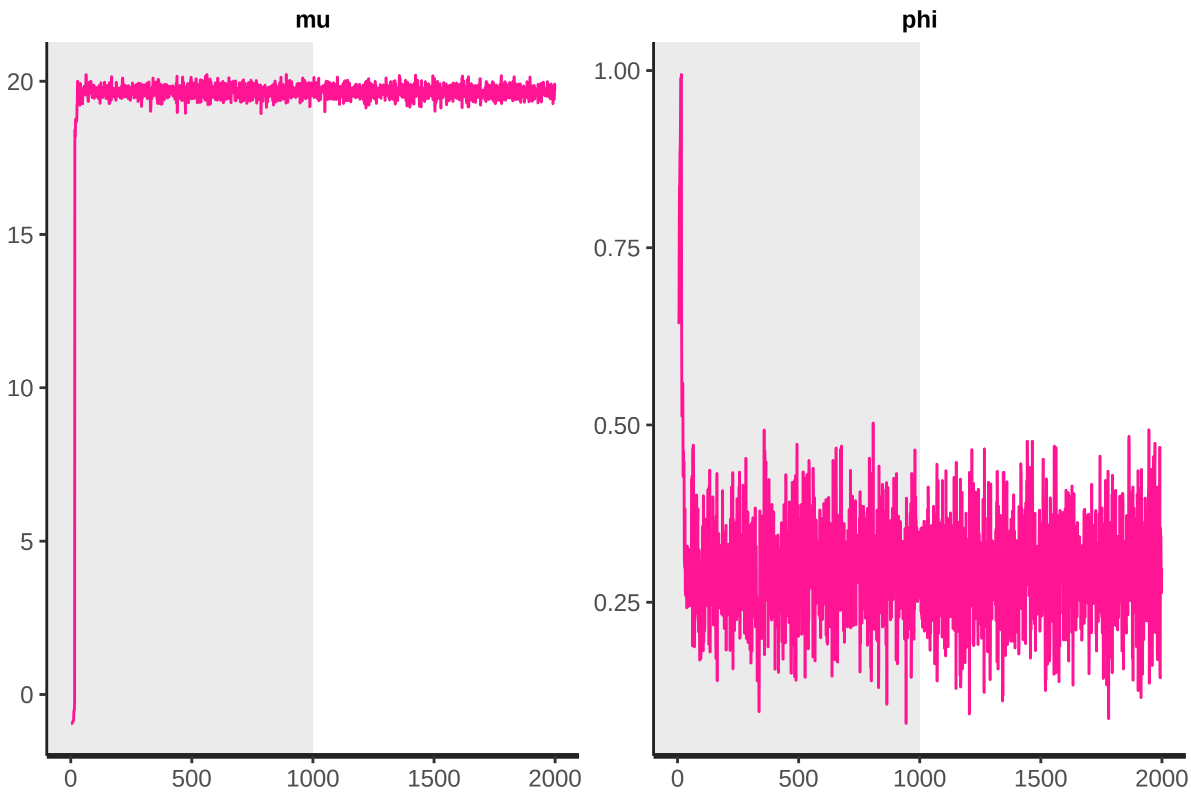

```
plot(fit.y)
```



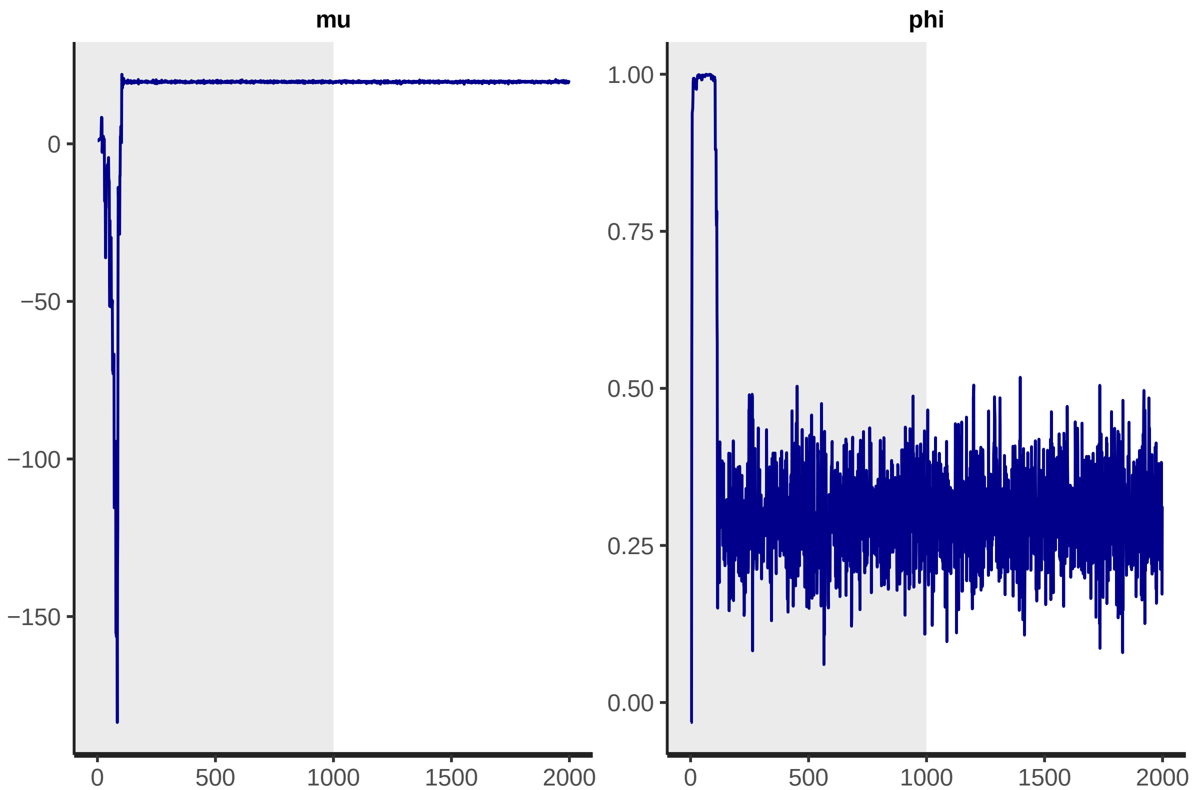
all parameters converges to the true value.

(ii) For each of the two data sets, evaluate the convergence of the samplers

```
trace1 <- traceplot(fit.x,  pars = c("mu", "phi"),
                    inc_warmup = TRUE,col='deeppink',include = TRUE)

trace2 <- traceplot(fit.y, pars = c("mu", "phi"),
                    inc_warmup = TRUE,col='darkblue',include = TRUE)
trace1
```


mu
phi
20
15
10
5
0
1.00
0.75
0.50
0.25
0
500
1000
1500
2000
0
500
1000
1500
2000

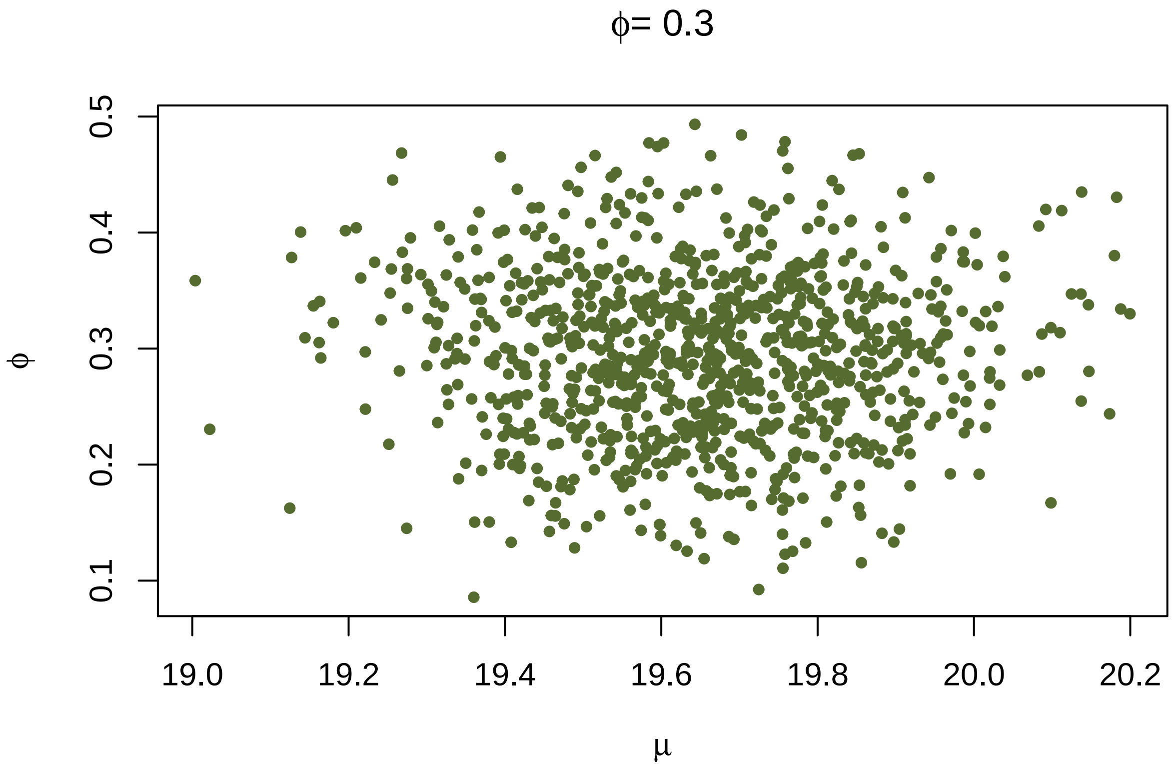
```
trace2
```



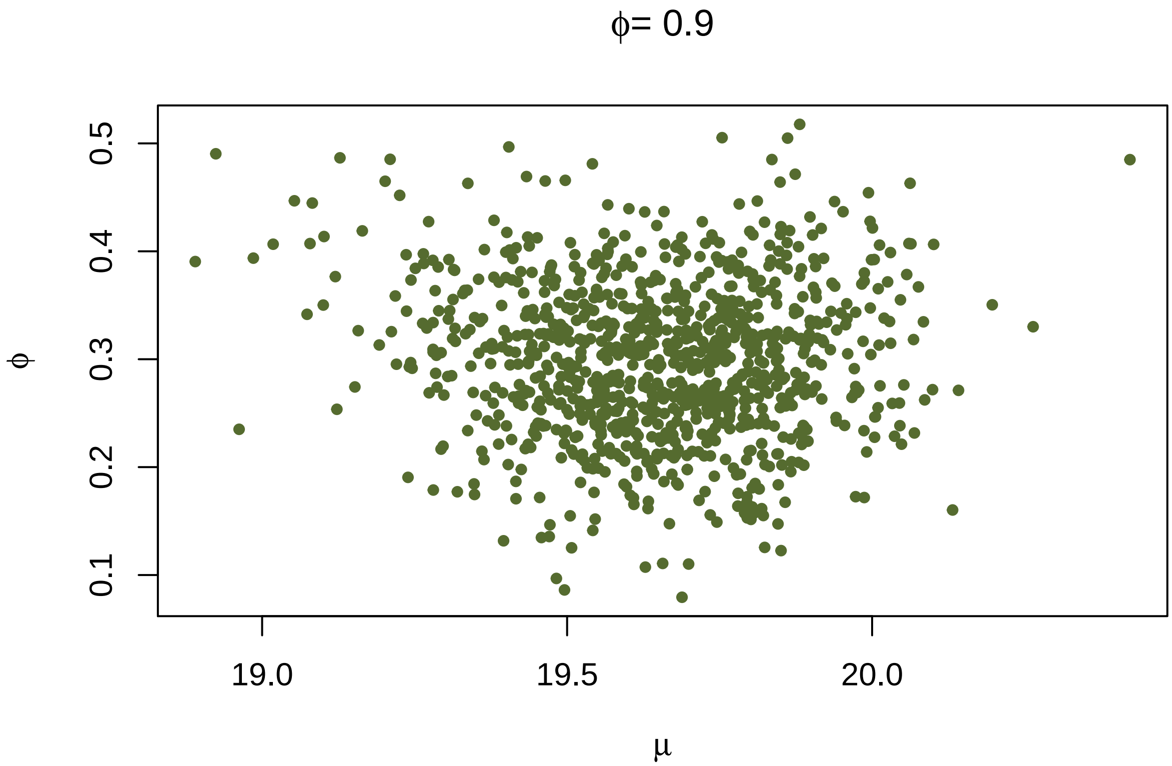
All paramaters converges to true value before 500 iteration.

(ii) plot the joint posterior μ and σ

```
par(mfrow=c(1,1))
plot(x=xDraws$mu, y=xDraws$phi,
     xlab=expression(mu),
     ylab=expression(phi),
     main=expression(paste(phi,"="," 0.3")),
     col='darkolivegreen',pch=20)
```




```
plot(x=yDraws$mu, y=yDraws$phi,
     xlab=expression(mu), ylab=expression(phi),
     main=expression(paste(phi,"="," 0.9")),
     col='darkolivegreen',pch=20)
```



In the the joint posterior for μ and ϕ when $\phi = 0.3$ we can see more variation on our draws .