Lab3

Wuhao Wang

5/4/2022

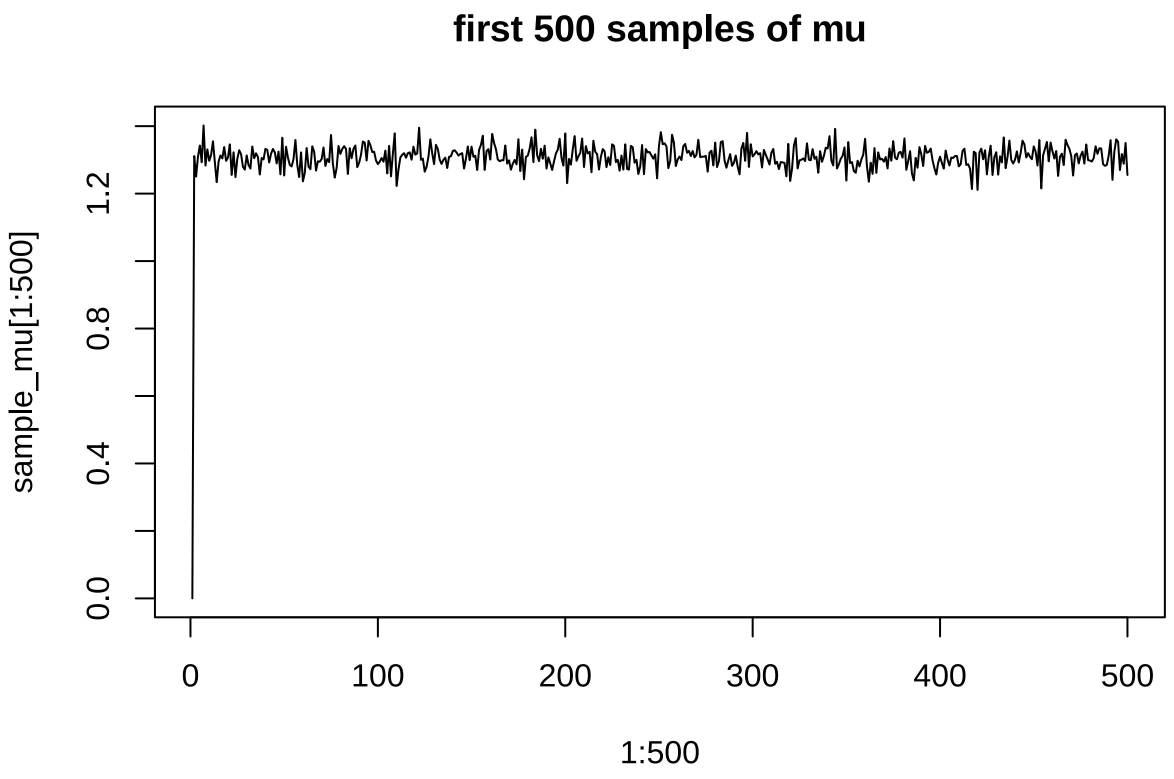
1 Gibbs sampler for a normal model

a

```
## IF of mu is  1.06454
```

```
## IF of sigma is  0.8580785
```

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



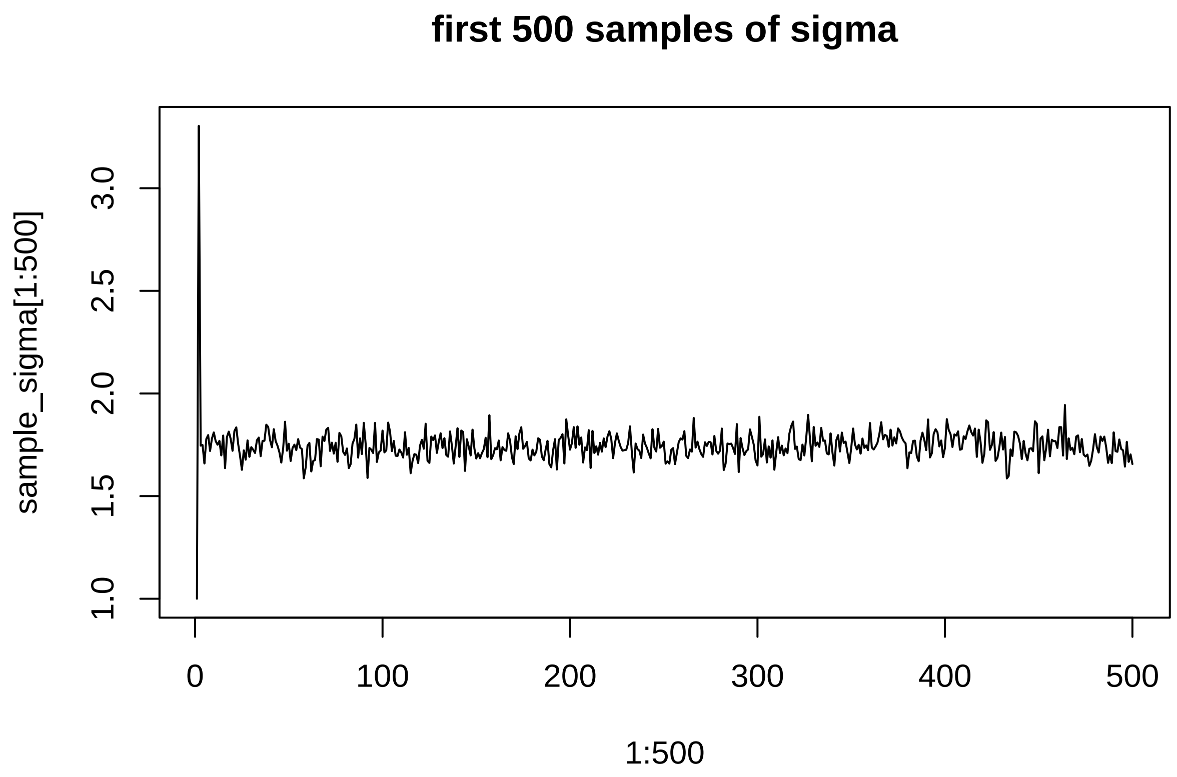
```
## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
## character
```

~~~
```{r , include=FALSE}
library(mvtnorm)
library(ggplot2)
library(coda)
library(rstan)
library(LaplacesDemon)
#setwd('E:\\courseMaterial\\732A73 bayesian\\Module 3')

```
~~~

a

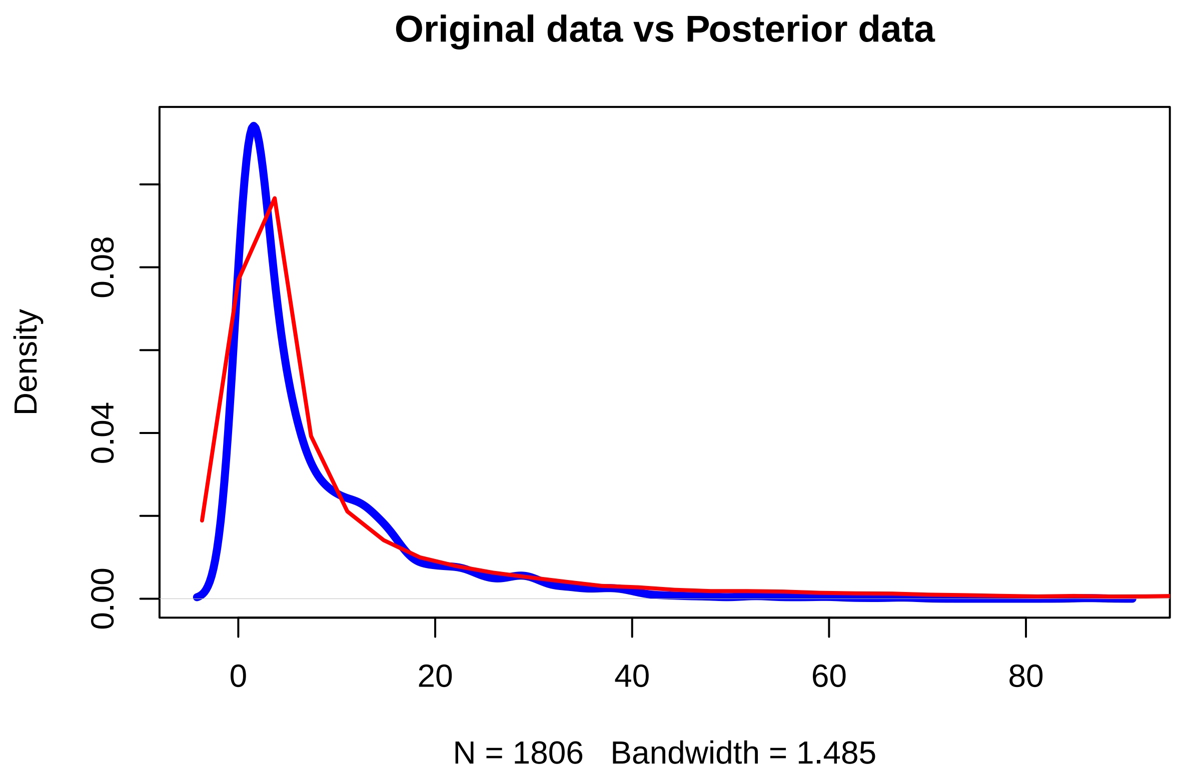
~~~
```{r echo=TRUE}
data <- readRDS('precipitation.rds')
data <- log(data)
set.seed(1234567)
mu0 <- 0
t0 <- 1
v0 <- 1
sigma0 <- 1
sample_mu <- c(mu0)
sample_sigma <- c(sigma0)
sample_data <- c()
n <- length(data)
tep <- sum(data-mean(data))**2
for (i in 2:8000)
{
  # L2 P4
  w <- (n/sample_sigma[i-1])/(n/sample_sigma[i-1]+1/t0)
  mu_n <- w*mean(data)+(1-w)*sample_mu[i-1]
  tn <- 1/(n/sample_sigma[i-1]+1/t0)
  sample_mu <- c(sample_mu,rnorm(1,mu_n,sqrt(tn)))
  # L7 P16
  vn <- v0+n
  sigma_n <- (v0*sigma0+sum((data-sample_mu[i-1])**2))/vn
  #sigma_n <- (v0*sigma0+tep+n/(n+1)*(mean(data)-sample_mu[i-1]))/vn
  sample_sigma <- c(sample_sigma,rinvchisq(n = 1, df = vn, scale = sigma_n))
  # get data draw
  sample_data <- c(sample_data,rnorm(1,sample_mu[i-1],sample_sigma[i-1]))
}
IF_mu <- 1+2*sum(acf(sample_mu,plot=FALSE)$acf[-1])
IF_sigma <- 1+2*sum(acf(sample_sigma,plot=FALSE)$acf[-1])
cat('IF of mu is ',IF_mu,'\n')
cat('IF of sigma is ',IF_sigma,'\n')
plot(1:500,sample_mu[1:500],main='first 500 samples of mu',type = 'line')
plot(1:500,sample_sigma[1:500],main='first 500 samples of sigma',type='line')
~~~



From the result above, we can see that the Inefficiency factors are quite small and both mu and sigma need only a short burn-in period to meet convergence.

b

```
sample_data <- exp(sample_data)
data <- exp(data)
plot(density(data),lwd = 4, col='blue',main = 'Original data vs Posterior data')
lines(density(sample_data),col = 'red',lwd = 2)
```



In the plot, the red line is density of sampled data, the blue line is original data, we can see that it matches quite well.

2 Metropolis Random Walk for Poisson regression

a

```
rm(list=ls())
X <- read.table('eBayNumberOfBidderData.dat',header = TRUE)
Y <- X$nBids
mle <- glm(data = X[,-2],nBids~.,family='poisson')
summary(mle)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = X[, -2])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

From the summary result above, we can find that `VeryfyID`,`Sealed`,`LogBook` and `MinBidShare` are significant.

b

```
X2 <- as.matrix(X[,-1])
n_fea <- ncol(X2)
n <- nrow(X2)
mu <- rep(0,n_fea)
beta <- as.matrix(rep(0,n_fea))
sigma <- 100*(solve(t(X2)%*%X2))
log_possion <- function(beta,X,y,mu,sigma1)
{
  lambda <- exp(X%*%beta)
  log_like <- sum(-lambda+y*log(lambda)-log(factorial(y)))
  log_prior <-dmvnorm(t(beta),mu,sigma1,log = TRUE)
  return (log_like+log_prior)
}
opt <- optim(par = beta,fn=log_possion,X=X2,y=Y,mu=mu,sigma1=sigma,hessian = TRUE,method='BFGS',control
mode <- opt$par
hessian <- -solve(opt$hessian)
cat('the posterior mode is ','\n')
```

```
## the posterior mode is
```

```
print(mode)
```

```
##             [,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
```

```
cat('the variance of posterior is','\n')
```

```
## the variance of posterior is
```

```
print(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 -4.292827e-04 -5.759169e-05 -6.437066e-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 -4.292827e-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

```
log_nor <- function(beta,X,y,mu,sigma1)
{
  lambda <- exp(X%*%beta)
  log_like <- sum(y*log(lambda)-lambda)
  log_prior <- dmvnorm(t(beta),mu,sigma1,log = TRUE)
  return (log_like+log_prior)
}
# X should not contain the target variable
# theta is a vector
n = 3500
RWM_Sampler <- function(theta,c,logPostFun,hessian,n_draws=3500,...)
{
  theta_m <- matrix(0,nrow = n_draws,ncol=length(theta))
  theta_m[1,] <- theta
  count <- 2
  total_it <- 0
  while(count <= n_draws)
  {
    total_it <- total_it+1
    sample <- t(rmvnorm(1,mean=theta_m[count-1,],sigma = c*hessian))
    up <- logPostFun(sample,...)
    down <- logPostFun(theta_m[count-1,],...)
    alpha <- min(1,exp(up-down))
    if (runif(1) <= alpha)
    {
      theta_m[count,] <- sample
      count <- count+1
    }
  }
  cat('accept rate is: ',count/total_it,'\n')
  return(theta_m)
}

samples <- RWM_Sampler(t(mu),0.55,log_nor,hessian,X=X2,y=Y,mu=mu,sigma1=sigma)
```

```
## accept rate is:  0.3053908
```

```
cat('The mean vector from random walk',colMeans(samples),'\n')
```

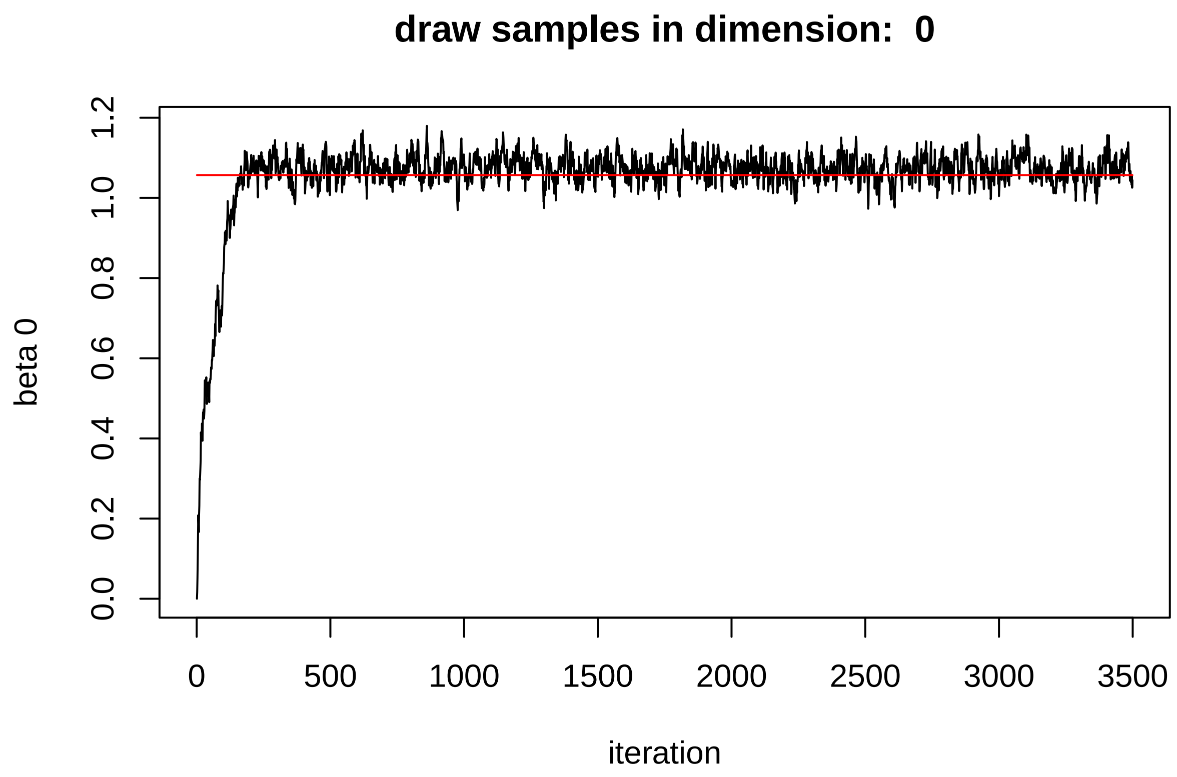
```
## The mean vector from random walk 1.057014 -0.04091288 -0.4013724 0.4346486 -0.06003627 -0.2234405 0.0
```

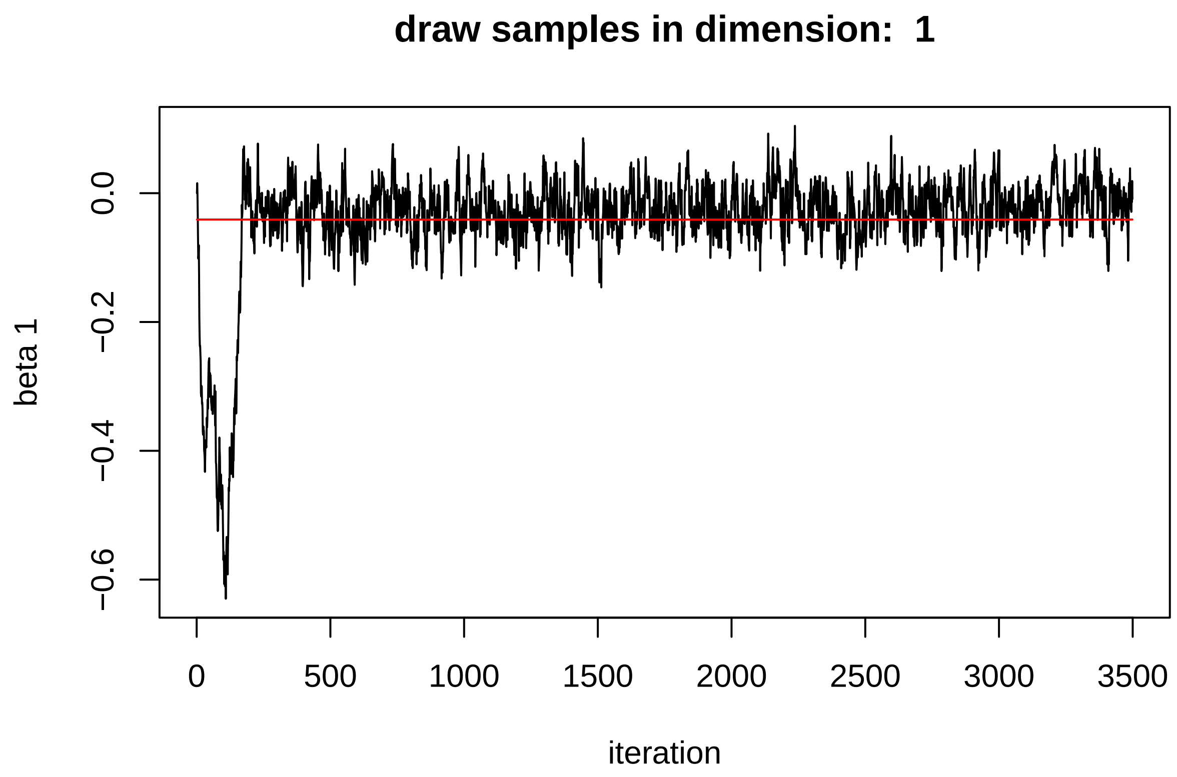
```
cat('The result from (b)',t(mode))
```

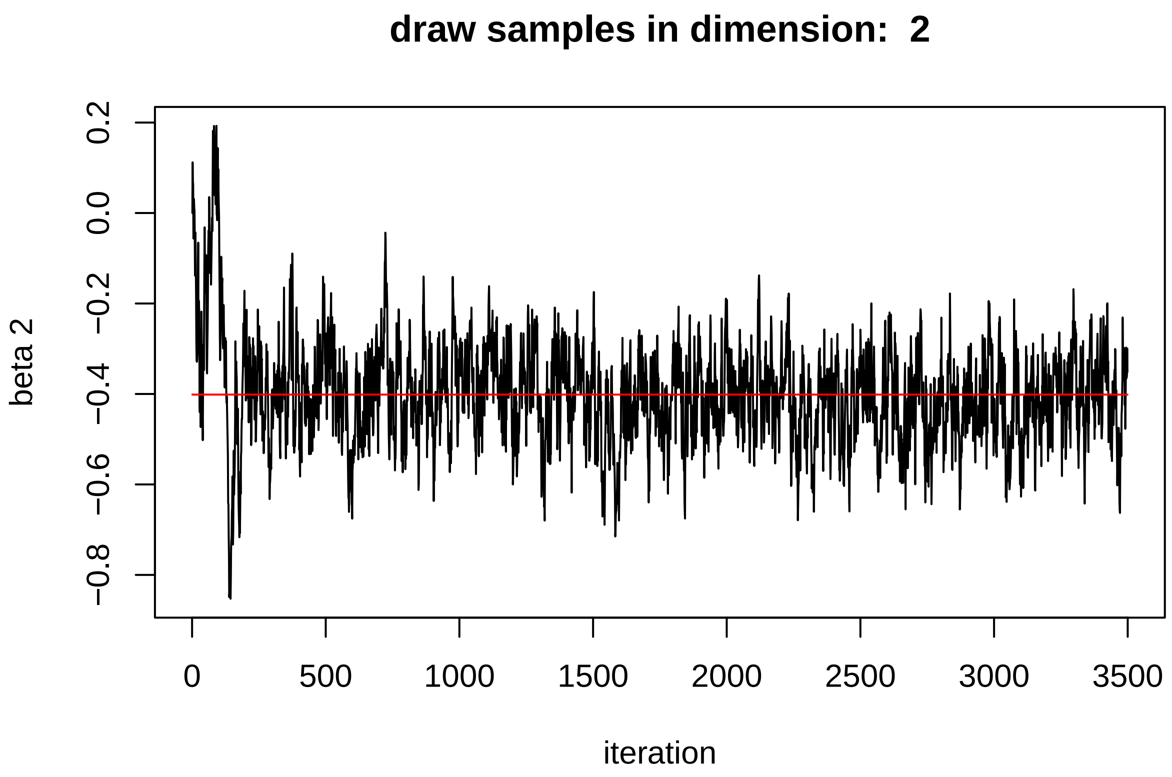
```
## The result from (b) 1.069841 -0.02051246 -0.393006 0.4435555 -0.05246627 -0.2212384 0.07069683 -0.12
```

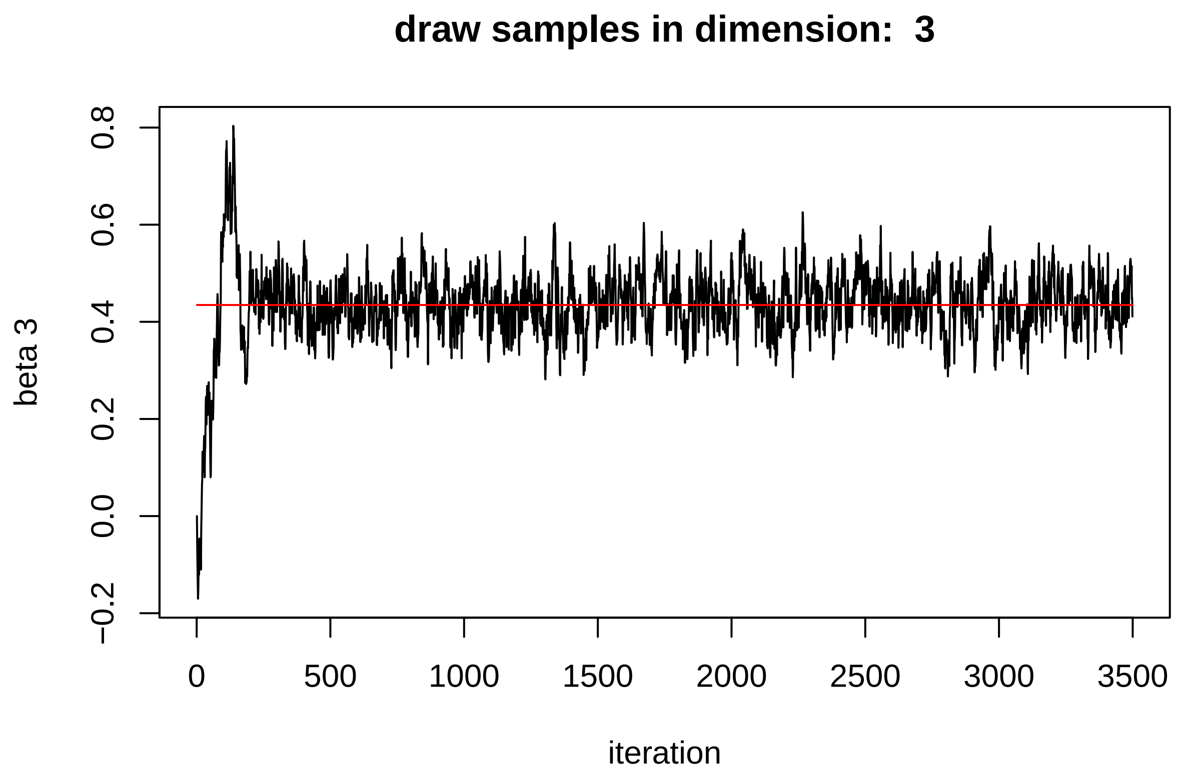
From the result above, we can see the draws can match the result from b) very well.

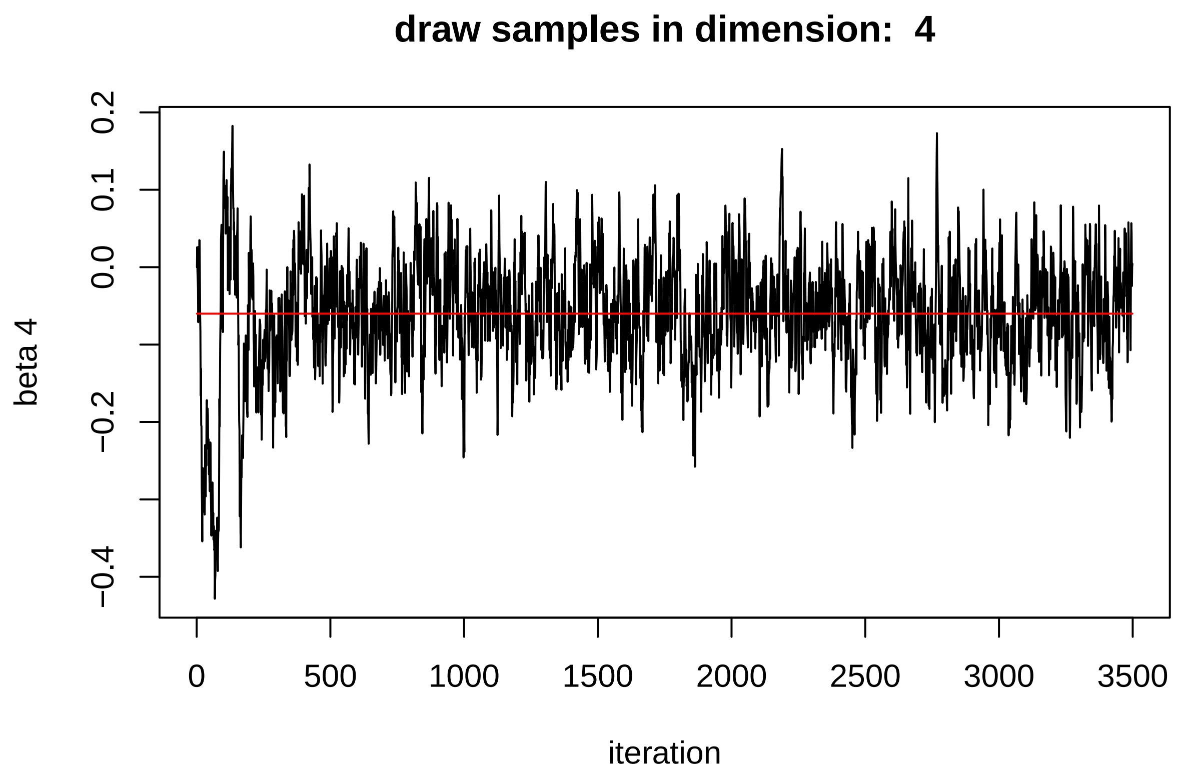
```
n_draw <- n-1
for (i in 1:9)
{
title <- paste('draw samples in dimension: ',i-1)
ylabel <- paste('beta',i-1)
plot(1:n_draw,samples[1:n_draw,i],main = title,xlab = 'iteration',ylab = ylabel,type = 'l')
lines(1:n_draw,rep(mean(samples[1:n_draw,i]),n_draw),type = 'l',col='red')
}
```

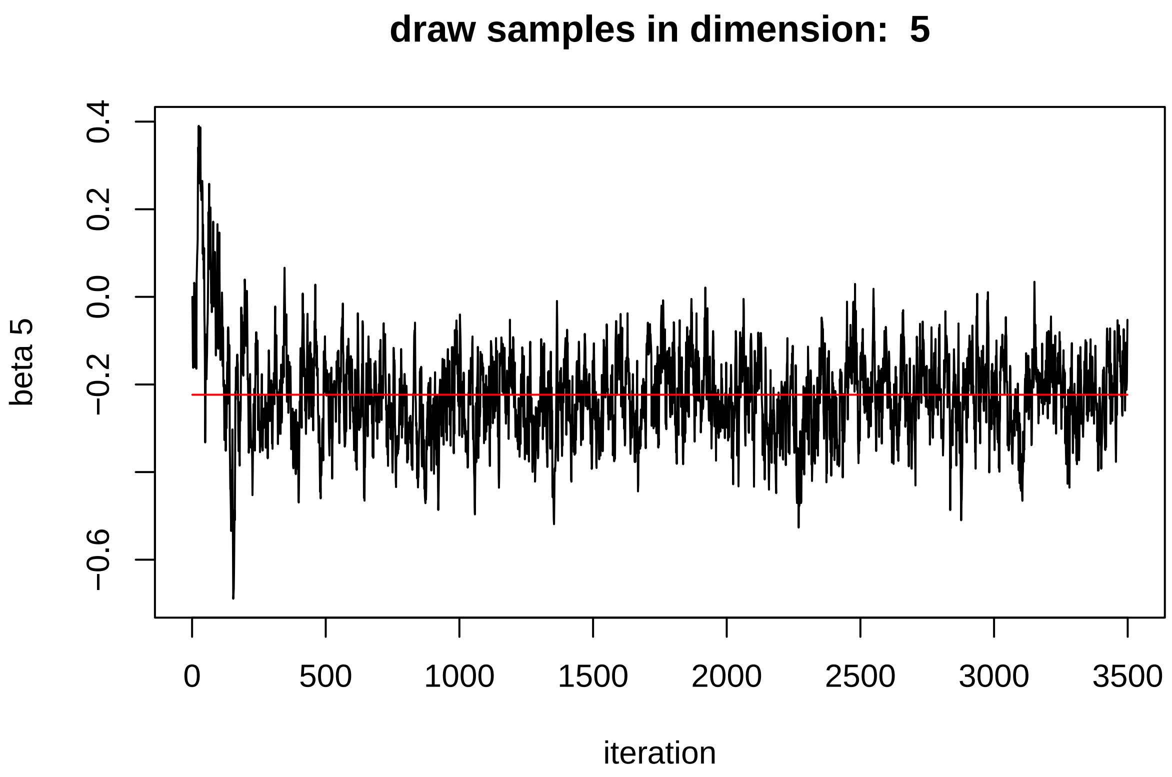

draw samples in dimension: 0
beta 0
iteration
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
500
1000
1500
2000
2500
3000
3500

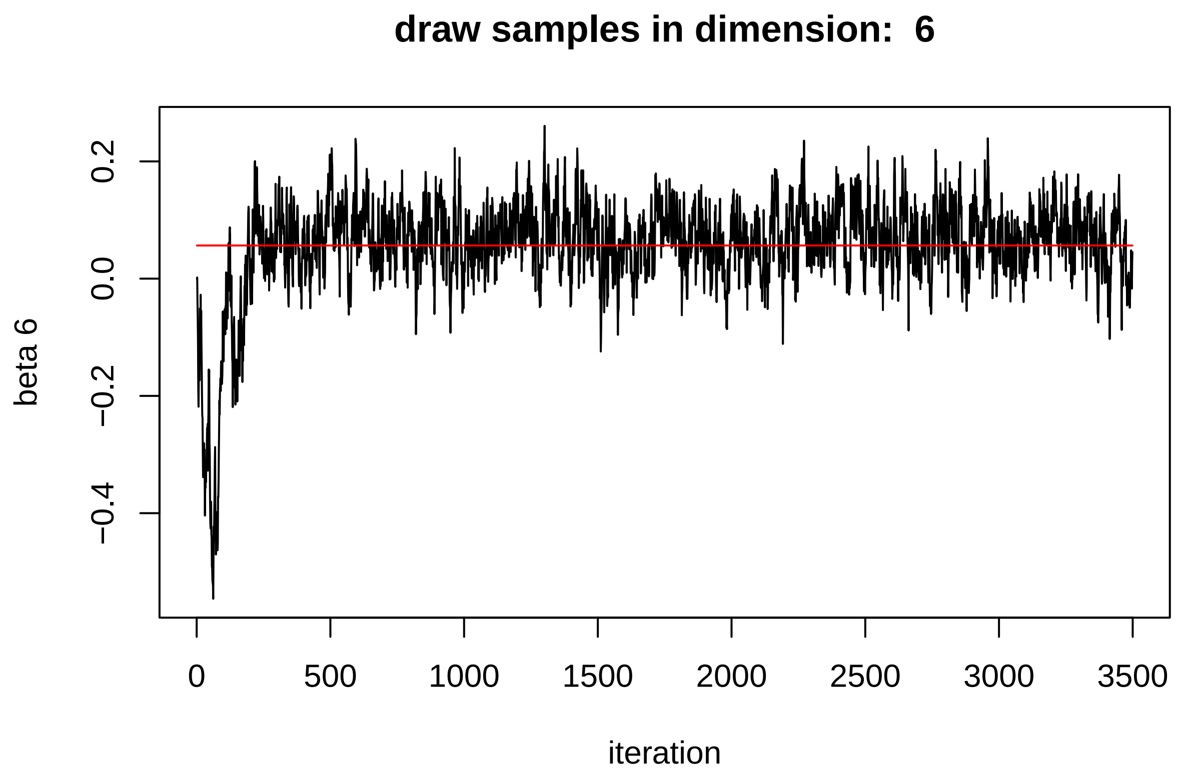

draw samples in dimension: 1
beta 1
0.0
−0.2
−0.4
−0.6
0
500
1000
1500
2000
2500
3000
3500
iteration

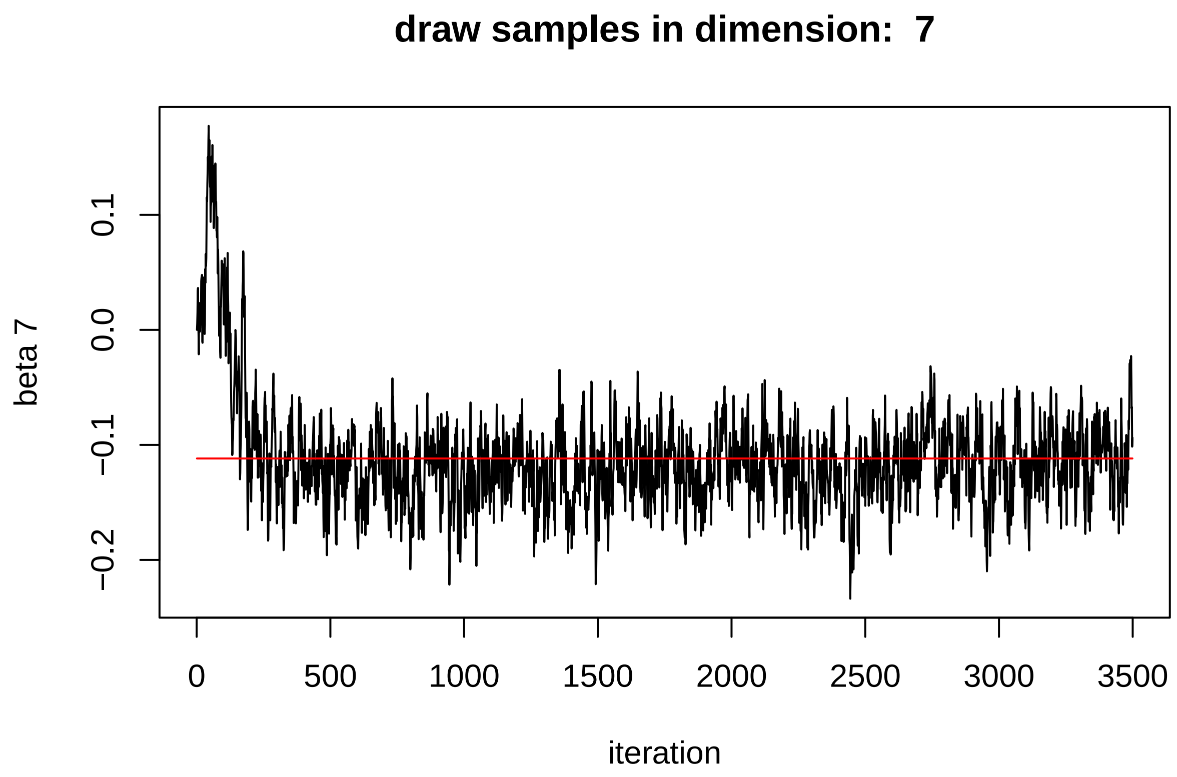

draw samples in dimension: 2
beta 2
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
500
1000
1500
2000
2500
3000
3500
iteration

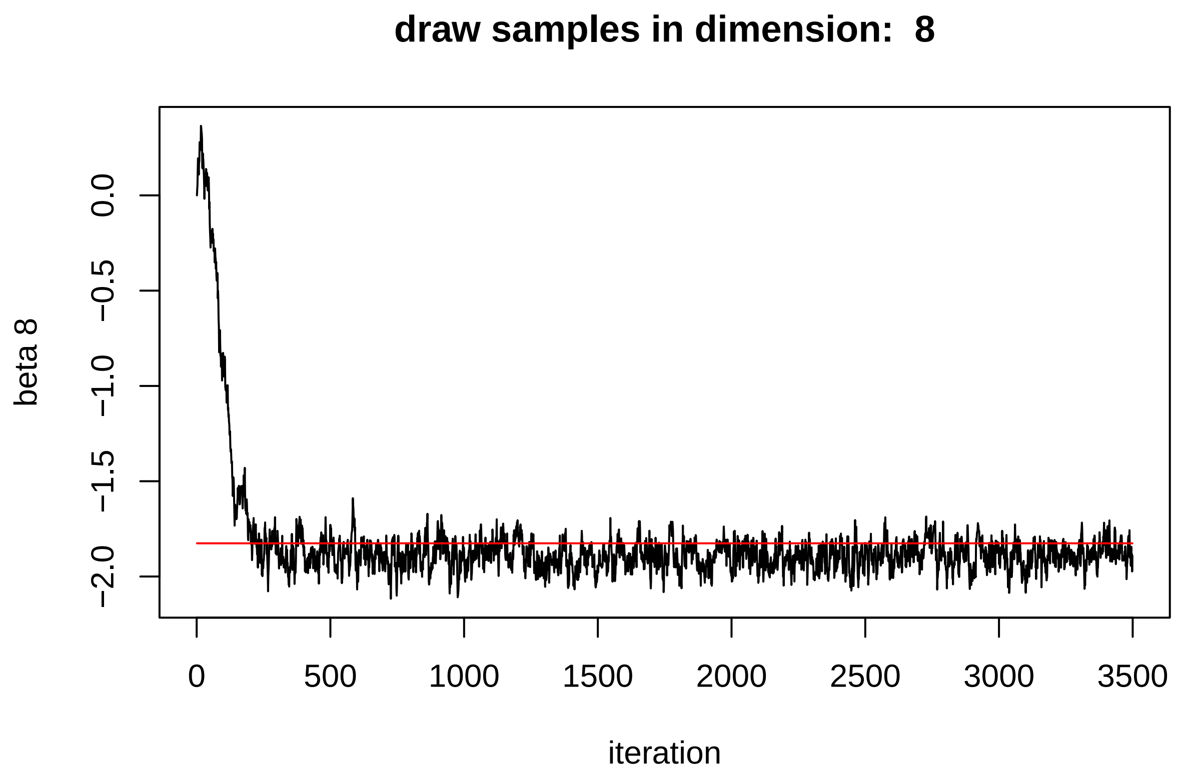

draw samples in dimension: 3
beta 3
iteration
−0.2
0.0
0.2
0.4
0.6
0.8
0
500
1000
1500
2000
2500
3000
3500


draw samples in dimension: 4
beta 4
0.2
0.1
0.0
−0.2
−0.4
0
500
1000
1500
2000
2500
3000
3500
iteration


draw samples in dimension: 5
beta 5
iteration
0.4
0.2
0.0
−0.2
−0.6
0
500
1000
1500
2000
2500
3000
3500


draw samples in dimension: 6
beta 6
iteration
0.2
0.0
−0.2
−0.4
0
500
1000
1500
2000
2500
3000
3500

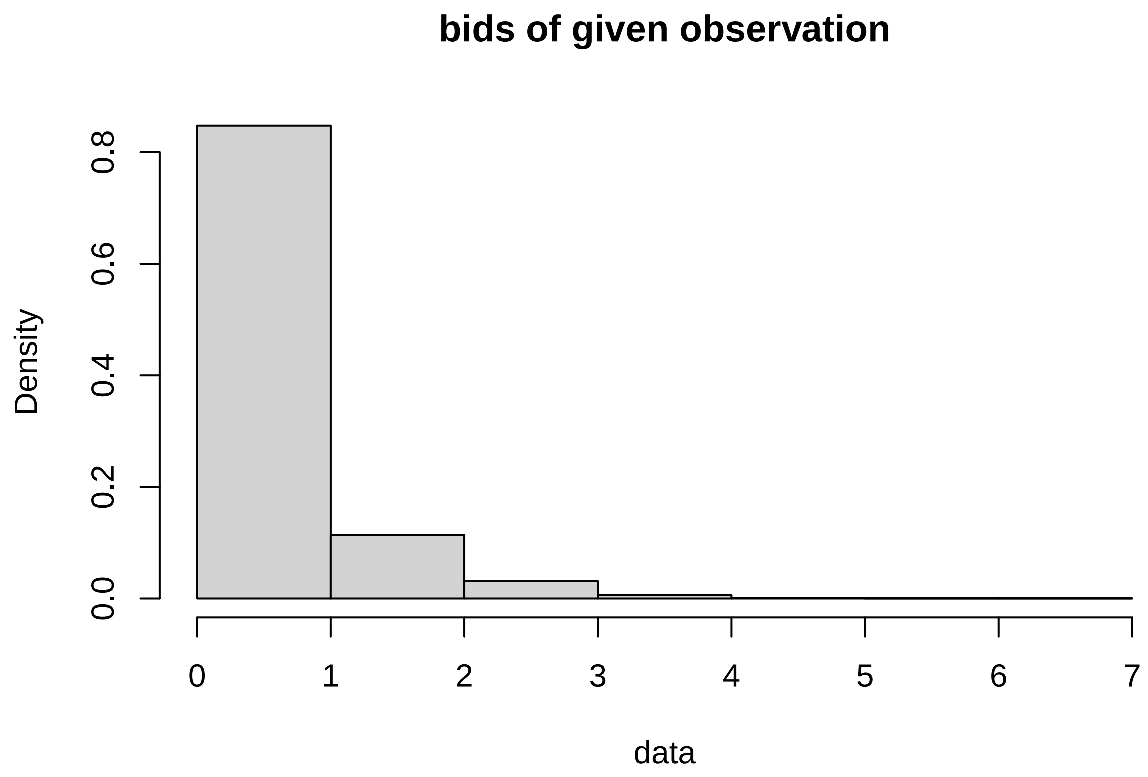

draw samples in dimension: 7
beta 7
iteration
0.1
0.0
−0.1
−0.2
0
500
1000
1500
2000
2500
3000
3500



The plots above also show that all dimensions only need a short burn-in period to reach convergence.

d

```
# situation include constant 1
situation <- c(1,1,0,1,0,1,0,1.2,0.8)
beta <- samples
a <- t(situation)%*%t(beta)
data <- c()
for (i in 1:length(a))
{
  data <- c(data,rpois(1,exp(a[1,i])))
}
hist(data,freq=FALSE,breaks = length(unique(data)),main = 'bids of given observation')
```



```
cat('the probability of no bid is:',sum(data==0)/length(data))
```

```
## the probability of no bid is: 0.502
```

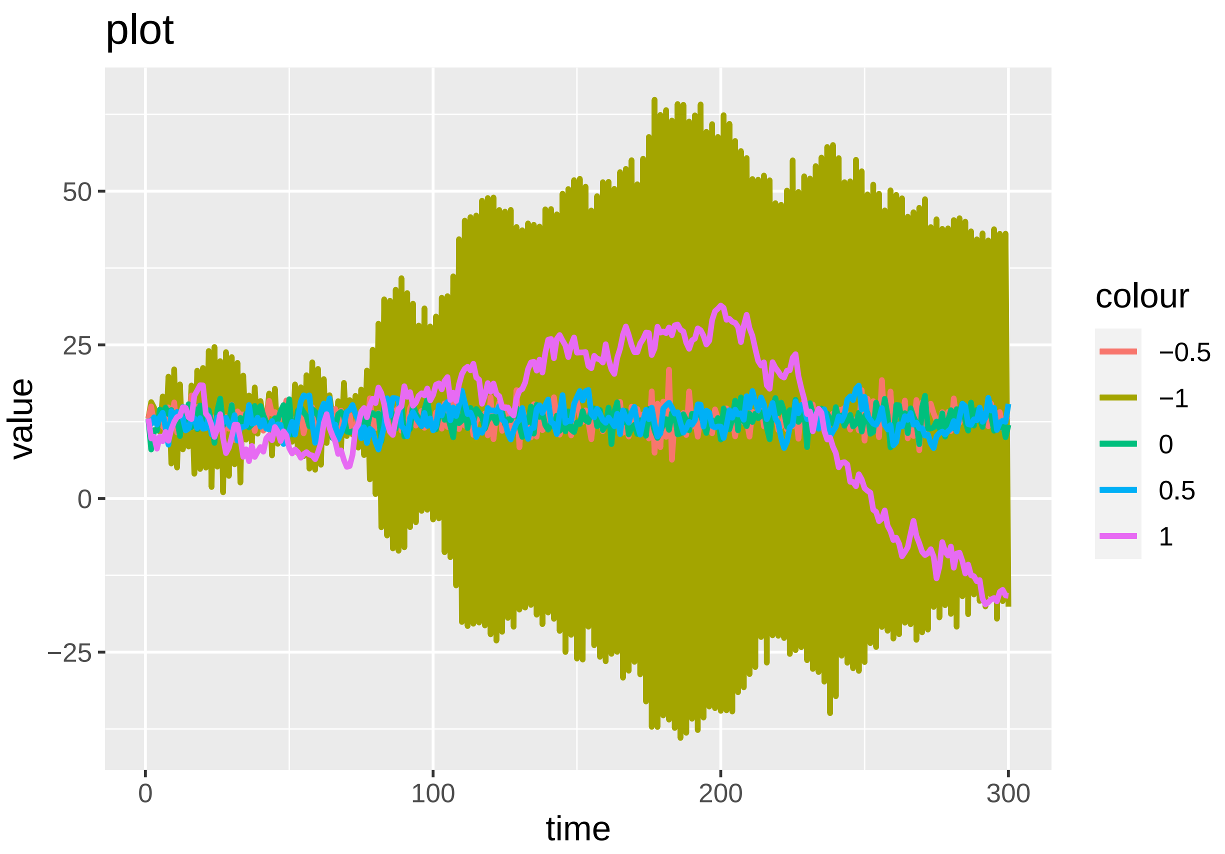
3 Time series models in Stan

a

```
rm(list=ls())
ar1_a <- function(mu,f,sigma,t)
{
  res <- c(mu)
  for (i in 2:t)
  {
    res <- c(res,mu+f*(res[i-1]-mu)+rnorm(1,0,sqrt(sigma)))
  }
  return(res)
}
df <- data.frame(index=1:300)
for (i in 1:5)
{
  f <- i/2 - 1.5
  df <-cbind(df,ar1_a(13,f,3,300))
}
```

```r
colnames(df) <- c('index','f_minus_1','f_minus_0.5','f_0','f_0.5','f_1')

ggplot()+geom_line(data = df,aes(x = index,y = f_minus_1,colour = "-1"),size=1)+
  geom_line(data = df,aes(x = index,y = f_minus_0.5,colour ='-0.5'),size=1) +
  geom_line(data = df,aes(x = index,y = f_0,colour ='0'),size=1) +
  geom_line(data = df,aes(x = index,y = f_0.5,colour ='0.5'),size=1) +
  geom_line(data = df,aes(x = index,y = f_1,colour ='1'),size=1) +
  xlab("time")+ylab("value")+
  ggtitle("plot")+
  theme(text=element_text(size=13))
```



When the absolute value of phi increase, the variance of draws will also increase.

b.i

```r
mu <- 13
sigma <- 3
t <- 300
sample1 <- ar1_a(mu,0.2,sigma,t)
sample2 <- ar1_a(mu,0.95,sigma,t)
```

```r
StanModel <- '
data {
int<lower=0> N;
vector[N] y;
}
```

```
parameters {
real mu;
real phi;
real<lower=0> sigma2;
}
model {
mu ~ normal(0,100);// Normal with mean 0, st.dev. 100
phi ~ normal(0,1);
sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2;
for (n in 2:N)
y[n] ~ normal(mu + phi * (y[n-1]-mu), sqrt(sigma2));
}'
```

```
data1 <- list(N=300,y=sample1)
data2 <- list(N=300,y=sample2)
warmup <- 1000
niter <- 4000
fit1 <- stan(model_code = StanModel,data = data1,
warmup = warmup,iter=niter,chains= 4)
```

```
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 1: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 1: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 1: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 1: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 1: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 1: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 1: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.178 seconds (Warm-up)
## Chain 1:                0.455 seconds (Sampling)
## Chain 1:                0.633 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.001 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 10 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
```

```
## Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 2: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 2: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 2: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 2: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 2: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 2: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 2: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.473 seconds (Warm-up)
## Chain 2:                0.422 seconds (Sampling)
## Chain 2:                0.895 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 3: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 3: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 3: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 3: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 3: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 3: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 3: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 3: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 3: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.239 seconds (Warm-up)
## Chain 3:                0.457 seconds (Sampling)
## Chain 3:                0.696 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 4: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 4: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 4: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 4: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 4: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 4: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 4: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 4: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 4: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 4: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.37 seconds (Warm-up)
## Chain 4:                0.388 seconds (Sampling)
## Chain 4:                0.758 seconds (Total)
## Chain 4:
```

```
fit2 <- stan(model_code = StanModel,data = data2,
warmup = warmup,iter=niter,chains= 4)
```

```
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 1: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 1: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 1: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 1: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 1: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 1: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 1: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.42 seconds (Warm-up)
## Chain 1:                0.529 seconds (Sampling)
## Chain 1:                0.949 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 2: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 2: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 2: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 2: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 2: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 2: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 2: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.257 seconds (Warm-up)
## Chain 2:                0.479 seconds (Sampling)
## Chain 2:                0.736 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 3: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 3: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 3: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 3: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 3: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 3: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 3: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 3: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 3: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.279 seconds (Warm-up)
## Chain 3:                0.479 seconds (Sampling)
## Chain 3:                0.758 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'fd60267ea33813e4e391f690528a308b' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 4: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 4: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1001 / 4000 [ 25%]  (Sampling)
## Chain 4: Iteration: 1400 / 4000 [ 35%]  (Sampling)
## Chain 4: Iteration: 1800 / 4000 [ 45%]  (Sampling)
## Chain 4: Iteration: 2200 / 4000 [ 55%]  (Sampling)
## Chain 4: Iteration: 2600 / 4000 [ 65%]  (Sampling)
## Chain 4: Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 4: Iteration: 3400 / 4000 [ 85%]  (Sampling)
## Chain 4: Iteration: 3800 / 4000 [ 95%]  (Sampling)
## Chain 4: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.262 seconds (Warm-up)
## Chain 4:                0.485 seconds (Sampling)
## Chain 4:                0.747 seconds (Total)
## Chain 4:
```

```
## Warning: There were 2 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
```

```
print('The summary of phi = 0.2')
```

```
## [1] "The summary of phi = 0.2"
```

```
print(summary(fit1, pars=c("mu","phi","sigma2"))$summary[,c(1,4,8)],digits_summary=3)
```

```
##              mean       2.5%      97.5%
## mu     12.9117459 12.6547564 13.1727994
## phi     0.2459477  0.1350083  0.3581123
## sigma2  2.8607935  2.4415748  3.3489745
```

```
print('The summary of phi = 0.95')
```

```
## [1] "The summary of phi = 0.95"
```

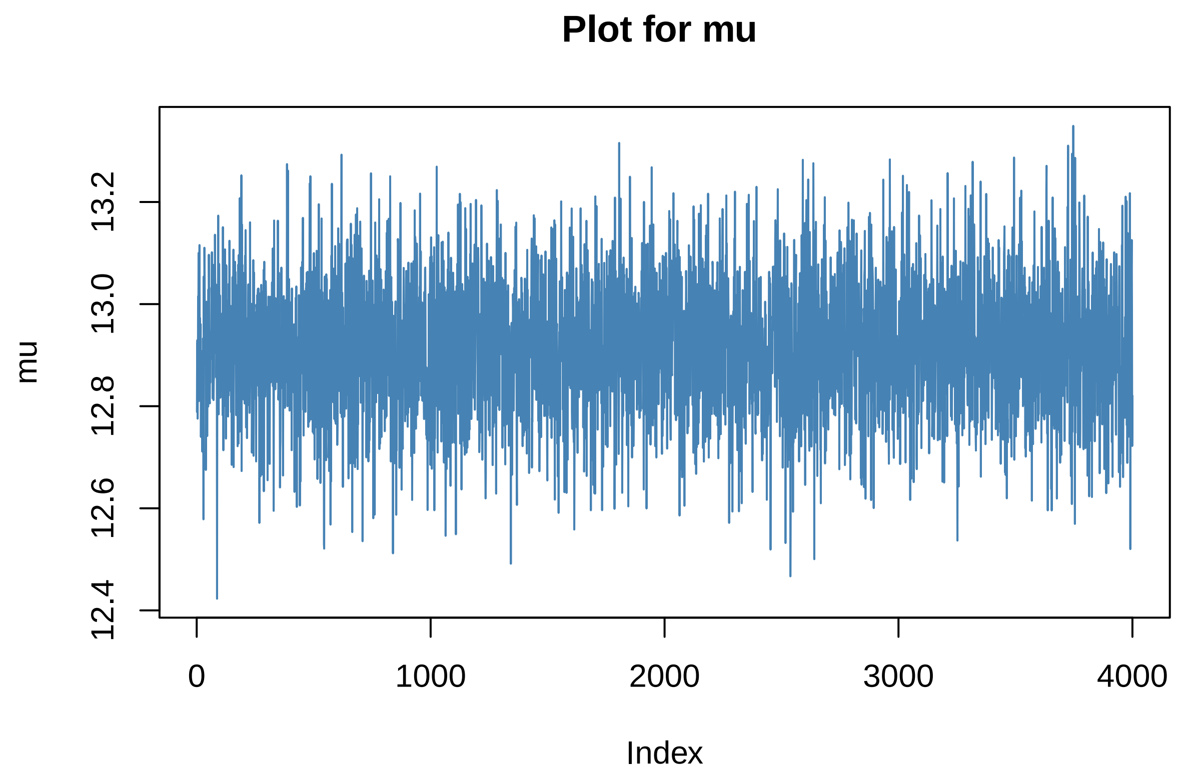
```
print(summary(fit2, pars=c("mu","phi","sigma2"))$summary[,c(1,4,8)],digits_summary=3)
```

```
##             mean      2.5%      97.5%
## mu     12.373529 9.5759261 15.1034256
## phi     0.915759 0.8665762  0.9694889
## sigma2  2.603813 2.2217128  3.0590241
```

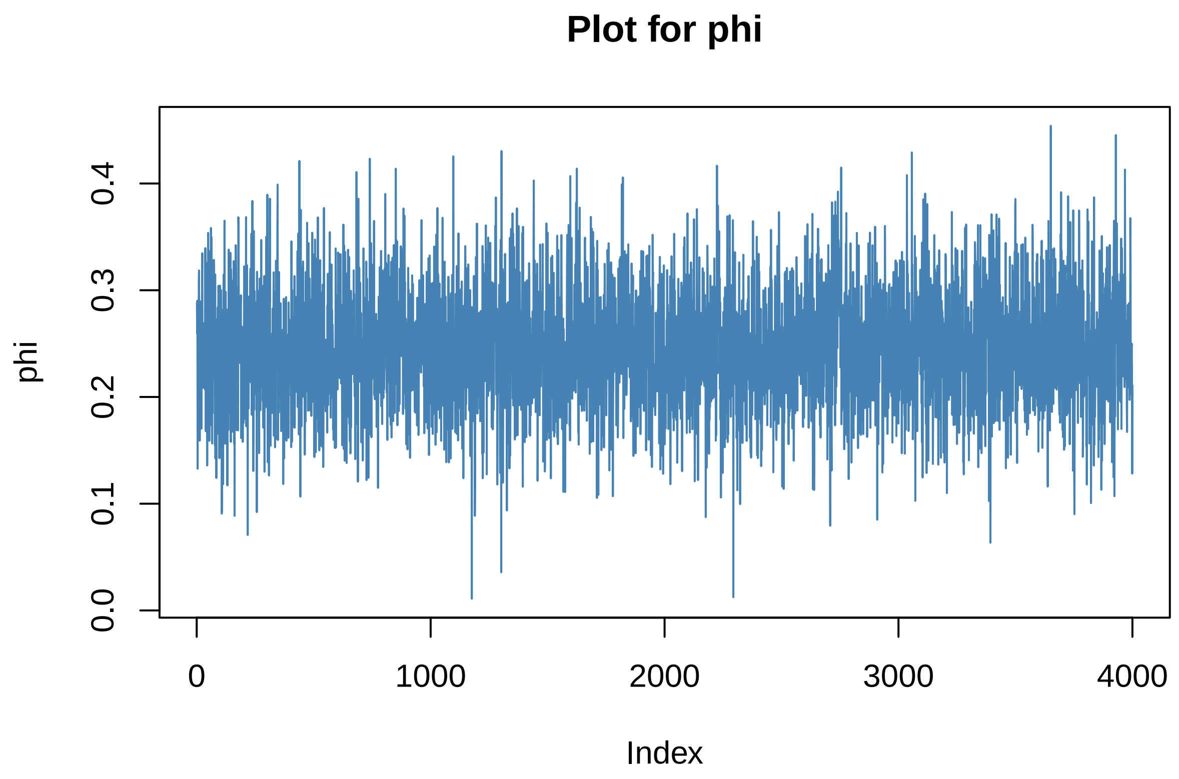
The given parameters are (mu,phi,sigma) = (13,0.2,3) and (mu,phi,sigma) = (13,0.95,3). From the stan results, we can see that when phi = 0.2, the model can match the parameters. But When phi = 0.95, the estimation of mu is not very precise.

The 95% credible intervals are also reported above.

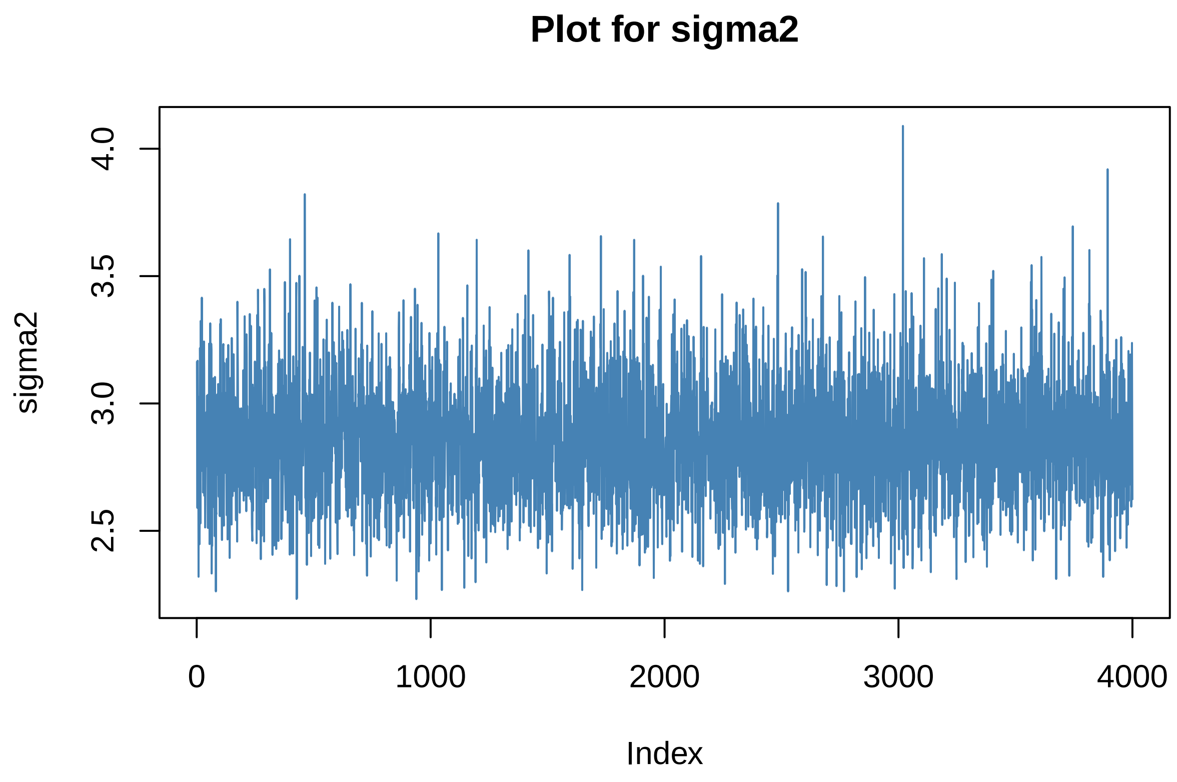
```
p1 <- extract(fit1)
plot(p1$mu[1:niter],type="l",ylab="mu",main="Plot for mu ", col="steelblue")
```



```
plot(p1$phi[1:niter],type="l",ylab="phi",main="Plot for phi", col="steelblue")
```

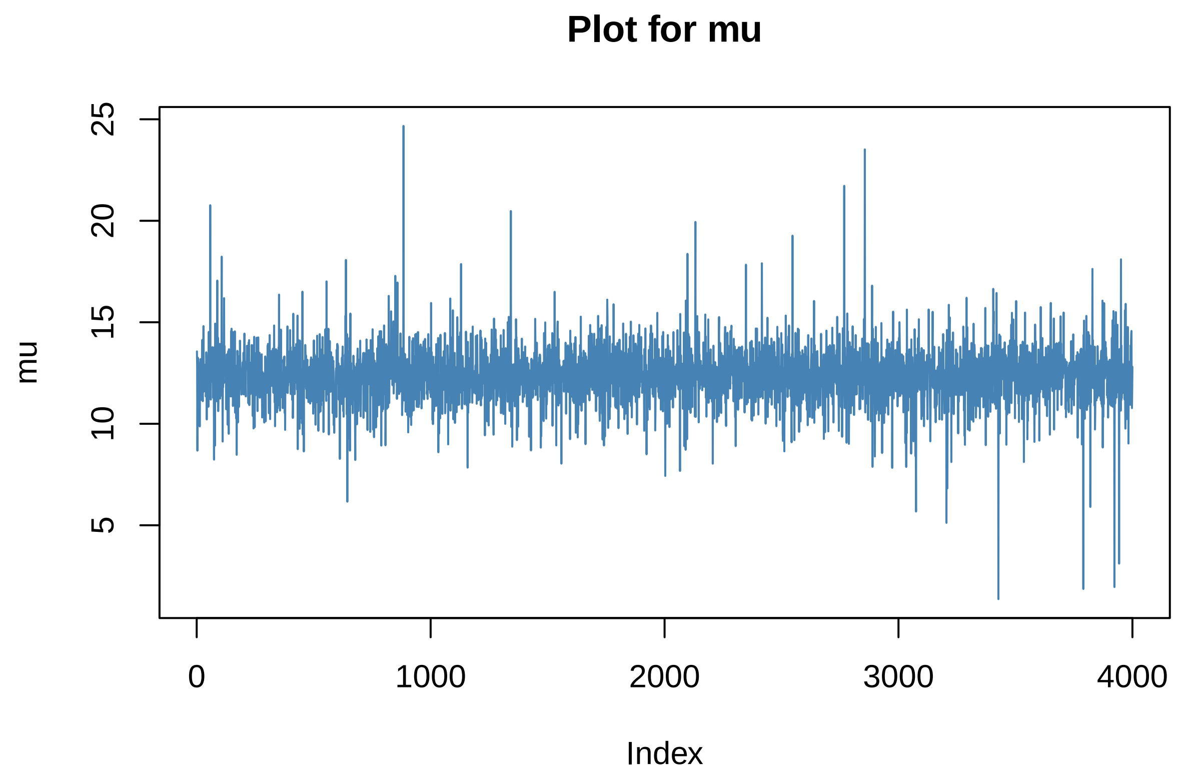


```
plot(p1$sigma2[1:niter],type="l",ylab="sigma2",main="Plot for sigma2", col="steelblue")
```

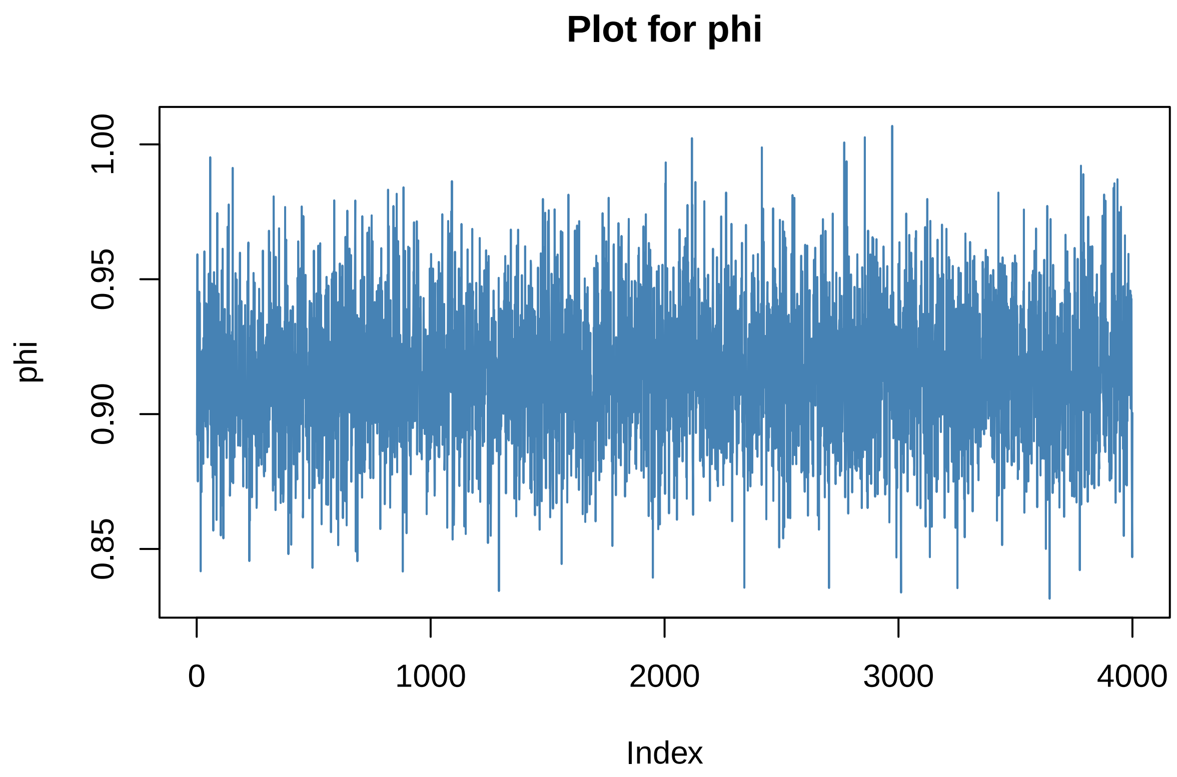


The plots above show the results for data1 where given parameters are (mu,phi,sigma) =(13,0.2,3). From the results ,we can see that all three parameters converge very quickly, so most samples are effective samples.

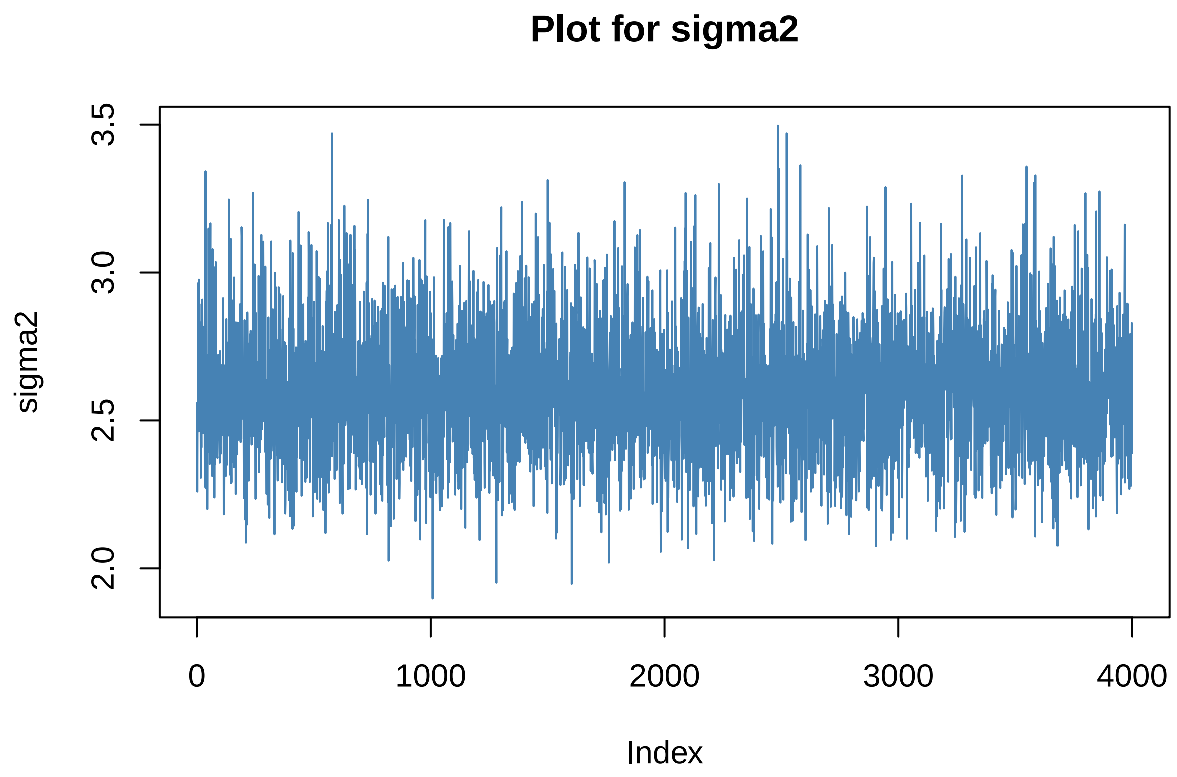
```
p2 <- extract(fit2)
plot(p2$mu[1:niter],type="l",ylab="mu",main="Plot for mu", col="steelblue")
```



```
plot(p2$phi[1:niter],type="l",ylab="phi",main="Plot for phi", col="steelblue")
```



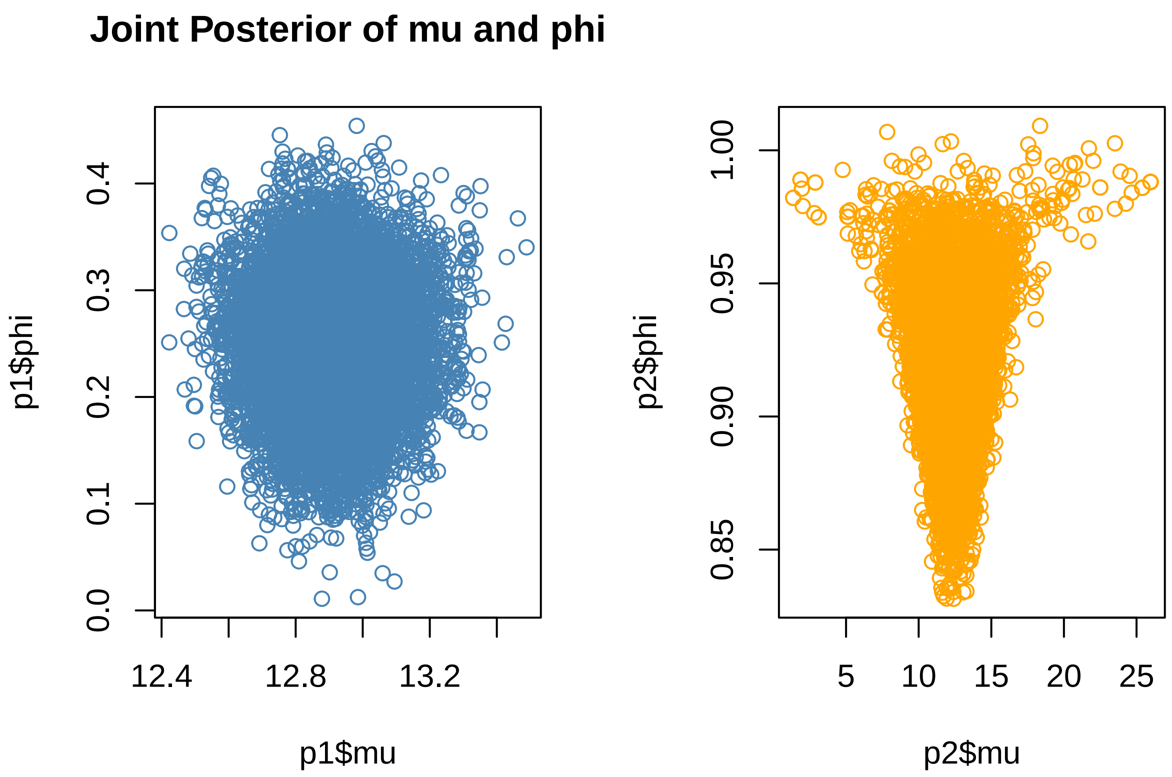
```
plot(p2$sigma2[1:niter],type="l",ylab="sigma2",main="Plot for sigma2", col="steelblue")
```



The plots above show the results for data2 where given parameters are (mu,phi,sigma) =(13,0.95,3). From the results ,we can see t hat for `phi` and `sigma`, they converge very fast which indicate that most samples are effective. However, we can see that `mu` can not converge, so I would say there are very limited samples being effective.

b.ii

```
par(mfrow = c(1,2))
plot(p1$mu, p1$phi, main = "Joint Posterior of mu and phi", col="steelblue")
plot(p2$mu, p2$phi, col="orange")
```



From the results, we can see that p1 has better result than p2 since the posterior of both parameters are normal models so the shape of joint posterior should also follow 'normal shape'.

As for the reason, in the question `a` we make comparison between bigger absolute value `phi` and smaller absolute value `phi` and know that bigger absolute value `phi` will result in bigger variance. And in the `b.i` we can see parameter `mu` in p2 never converge.