HW10

Lun Li.14415

2025-04-20

```
library(coda)
```

```
## Warning: package 'coda' was built under R version 4.3.3
```

```
library(rjags)
```

```
## Warning: package 'rjags' was built under R version 4.3.3

## Linked to JAGS 4.3.2

## Loaded modules: basemod,bugs
```

```
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(coda)
```

Problem 1

a

```
frd <- read.table('/Users/landlee/Desktop/STAT 3303/HW10/friends.txt',
                      header = TRUE)
```

```r
# MCMC parameters
niter <- 30000
nburn <- 10000
nadapt <- 10000

# Number of schools and total observations
Ns <- length(unique(frd$school))
N <- nrow(frd)

# Prepare JAGS data
data_jags <- list(
  y = frd$y,
  x = frd$x,
  school = frd$school,
  Ns = Ns,
  N = N
)

# Initial values
set.seed(123)
inits <- list(
  mu_alpha = 0,
  mu_beta = 0,
  sigma_alpha = 1,
  sigma_beta = 1,
  alpha = rnorm(Ns, 0, 1),
  beta = rnorm(Ns, 0, 1)
)
```

```r
model_string <- "model {
  for (i in 1:N) {
    y[i] ~ dbern(theta[i])
    logit(theta[i]) <- alpha[school[i]] + beta[school[i]] * x[i]
  }

  for (j in 1:Ns) {
    alpha[j] ~ dnorm(mu_alpha, tau_alpha)
    beta[j] ~ dnorm(mu_beta, tau_beta)
  }

  mu_alpha ~ dnorm(0, 1/9)
  mu_beta ~ dnorm(0, 1/9)
  sigma_alpha ~ dunif(0, 3)
  sigma_beta ~ dunif(0, 3)

  tau_alpha <- pow(sigma_alpha, -2)
  tau_beta <- pow(sigma_beta, -2)
}"
```

```r
fit1 = jags.model(textConnection(model_string),
                  data = data_jags,
                  inits = inits,
                  n.chains = 1,
```

```
                 n.adapt = nadapt)
```

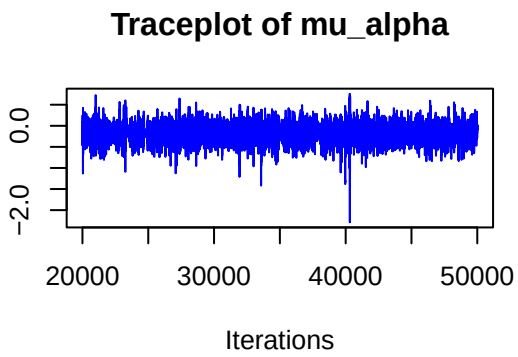
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1520
##    Unobserved stochastic nodes: 20
##    Total graph size: 4639
##
## Initializing model
```

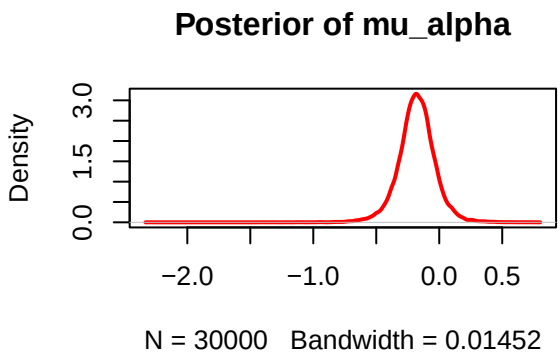
```
update(fit1, nburn)  # Burn-in phase

fit.samples = coda.samples(fit1,
                           c("mu_alpha", "mu_beta", "sigma_alpha", "sigma_beta", "alpha", "beta"),
                           n.iter = niter)

summary(fit.samples)
```

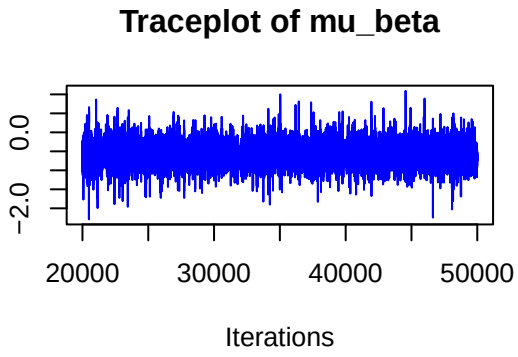
```
##
## Iterations = 20001:50000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean     SD  Naive SE Time-series SE
## alpha[1]    -0.61638 0.2566 0.0014815       0.007778
## alpha[2]    -0.20055 0.1619 0.0009345       0.001943
## alpha[3]    -0.07552 0.1669 0.0009639       0.002359
## alpha[4]    -0.07712 0.1649 0.0009520       0.002318
## alpha[5]     0.09777 0.1940 0.0011198       0.004531
## alpha[6]    -0.11047 0.1652 0.0009535       0.001989
## alpha[7]    -0.23545 0.1656 0.0009558       0.002279
## alpha[8]    -0.21835 0.1649 0.0009520       0.002150
## beta[1]     -1.26382 0.4147 0.0023945       0.010226
## beta[2]     -0.38304 0.2352 0.0013578       0.002917
## beta[3]     -0.41145 0.2380 0.0013742       0.002886
## beta[4]     -0.40862 0.2358 0.0013612       0.003169
## beta[5]     -0.18133 0.2679 0.0015465       0.005922
## beta[6]     -0.77696 0.2500 0.0014434       0.003076
## beta[7]     -0.76385 0.2524 0.0014575       0.003574
## beta[8]     -0.33368 0.2390 0.0013799       0.003230
## mu_alpha    -0.17986 0.1485 0.0008574       0.001679
## mu_beta     -0.56176 0.2302 0.0013292       0.002289
## sigma_alpha  0.31415 0.1819 0.0010500       0.005164
## sigma_beta   0.51230 0.2755 0.0015904       0.006636
##
## 2. Quantiles for each variable:
```

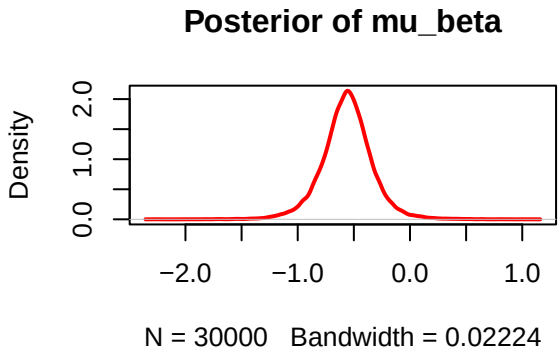
```
##
##                  2.5%      25%      50%        75%    97.5%
## alpha[1]    -1.11402 -0.79610 -0.61638 -0.4353529 -0.12936
## alpha[2]    -0.52798 -0.30519 -0.19644 -0.0937009  0.10944
## alpha[3]    -0.39485 -0.18791 -0.08178  0.0344720  0.26269
## alpha[4]    -0.39011 -0.18803 -0.08304  0.0298063  0.25752
## alpha[5]    -0.24503 -0.04358  0.09088  0.2324670  0.48675
## alpha[6]    -0.42870 -0.21922 -0.11432 -0.0050599  0.23056
## alpha[7]    -0.57545 -0.34332 -0.23008 -0.1250262  0.08054
## alpha[8]    -0.55365 -0.32540 -0.21445 -0.1081875  0.09711
## beta[1]     -2.12085 -1.54677 -1.24464 -0.9603454 -0.52941
## beta[2]     -0.82718 -0.54264 -0.39173 -0.2267795  0.09682
## beta[3]     -0.86836 -0.57374 -0.41844 -0.2516141  0.06539
## beta[4]     -0.86532 -0.56757 -0.41396 -0.2520415  0.06630
## beta[5]     -0.67261 -0.37233 -0.18618  0.0006705  0.35804
## beta[6]     -1.28792 -0.94149 -0.76711 -0.6032845 -0.30751
## beta[7]     -1.27974 -0.92862 -0.75429 -0.5913783 -0.29248
## beta[8]     -0.78439 -0.49870 -0.33838 -0.1763076  0.14862
## mu_alpha    -0.48478 -0.26359 -0.17848 -0.0936096  0.11113
## mu_beta     -1.03410 -0.69114 -0.55949 -0.4309090 -0.09974
## sigma_alpha  0.02787  0.19937  0.29020  0.3992051  0.72824
## sigma_beta   0.10276  0.33356  0.46639  0.6371775  1.19224
```

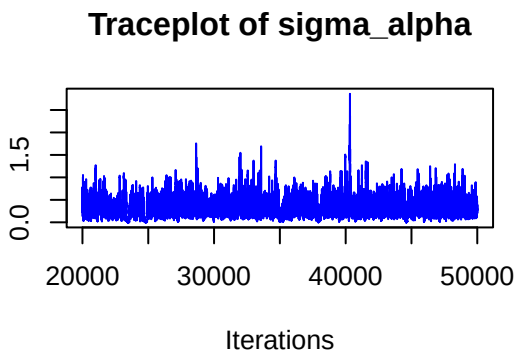
```r
# Extract MCMC samples into matrix
mcmc_matrix <- as.matrix(fit.samples)

# Choose key parameters to visualize
params <- c("mu_alpha", "mu_beta", "sigma_alpha", "sigma_beta")

# Set up plotting layout: 2 rows, 2 columns (each parameter gets trace + density)
par(mfrow = c(2, 2))  # 2 plots per row

# Loop through parameters and plot
for (param in params) {
  traceplot(fit.samples[, param],
            main = paste("Traceplot of", param),
            col = "blue", lwd = 1)

  plot(density(mcmc_matrix[, param]),
       main = paste("Posterior of", param),
       col = "red", lwd = 2)
}
```

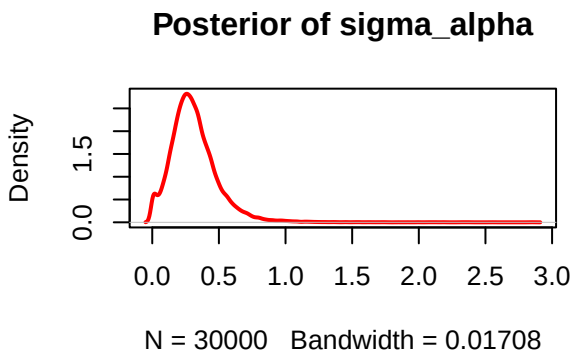

Traceplot of mu_alpha
−2.0
0.0
20000
30000
40000
50000
Iterations

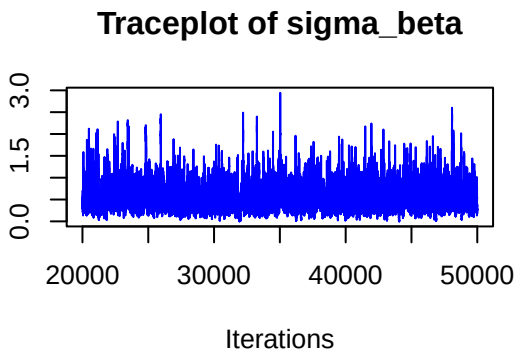

Posterior of mu_alpha
Density
0.0
1.5
3.0
−2.0
−1.0
0.0
0.5
N = 30000 Bandwidth = 0.01452

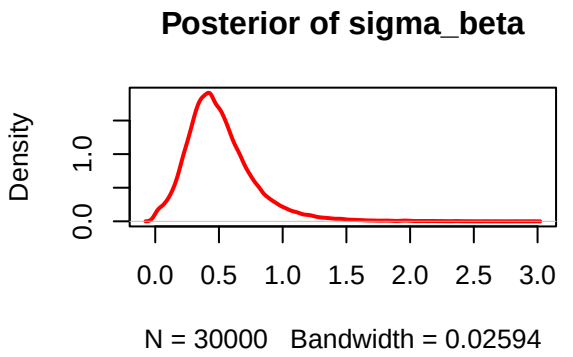

Traceplot of mu_beta
−2.0
0.0
20000
30000
40000
50000
Iterations


Posterior of mu_beta
Density
0.0
1.0
2.0
−2.0
−1.0
0.0
1.0
N = 30000 Bandwidth = 0.02224










```r
fit.samples.adapted <- window(fit.samples, start = nadapt + 1, end = niter)
```

```
## Warning in FUN(X[[i]], ...): start value not changed
```

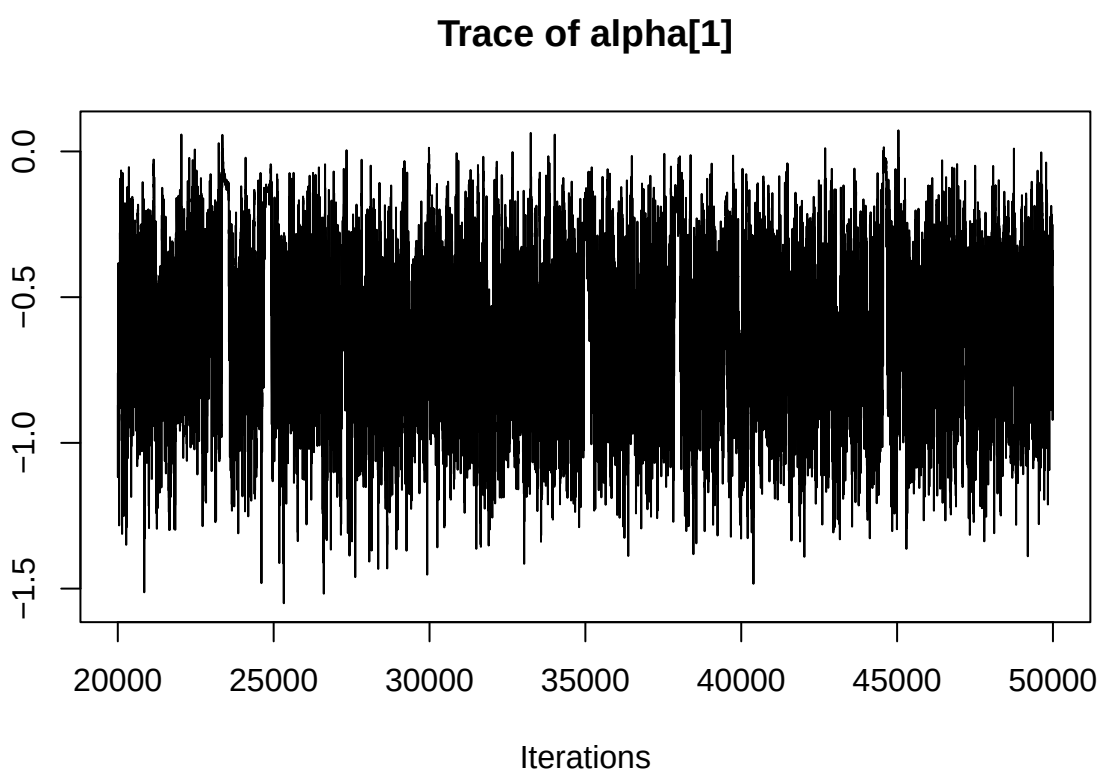
```r
# ESS
ess <- effectiveSize(fit.samples.adapted)
cat("Effective Sample Size (ESS):\n")
```

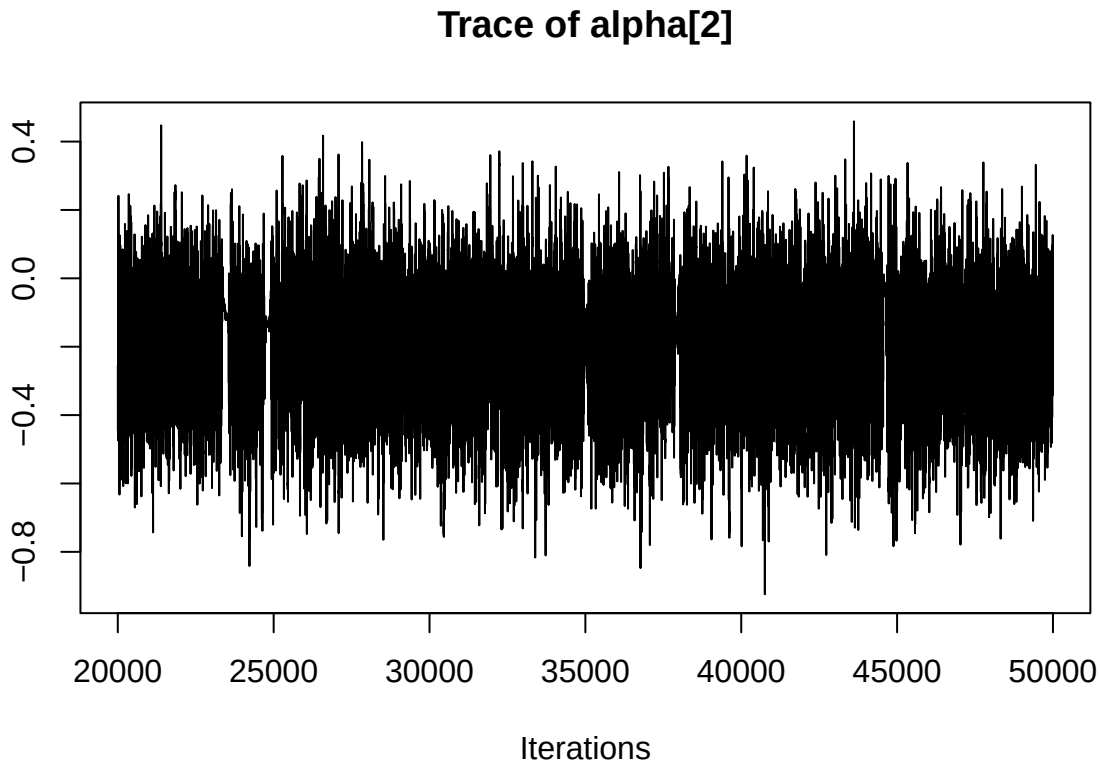
```
## Effective Sample Size (ESS):
```

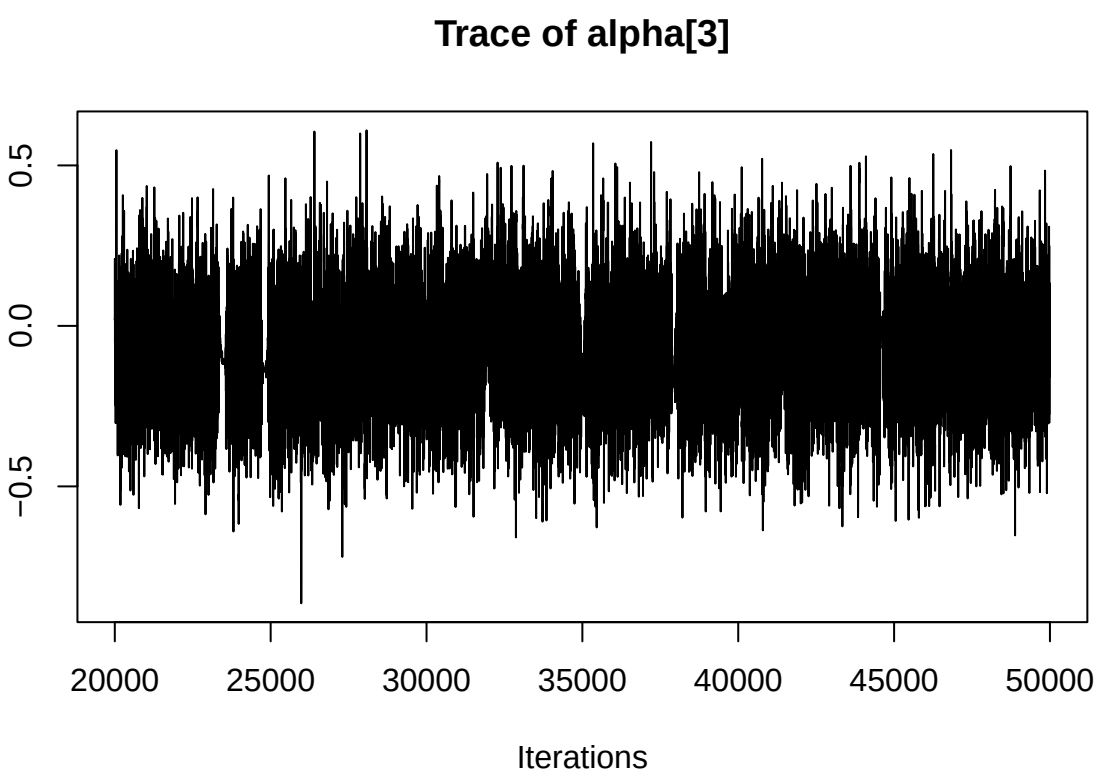
```r
print(round(ess, 2))
```

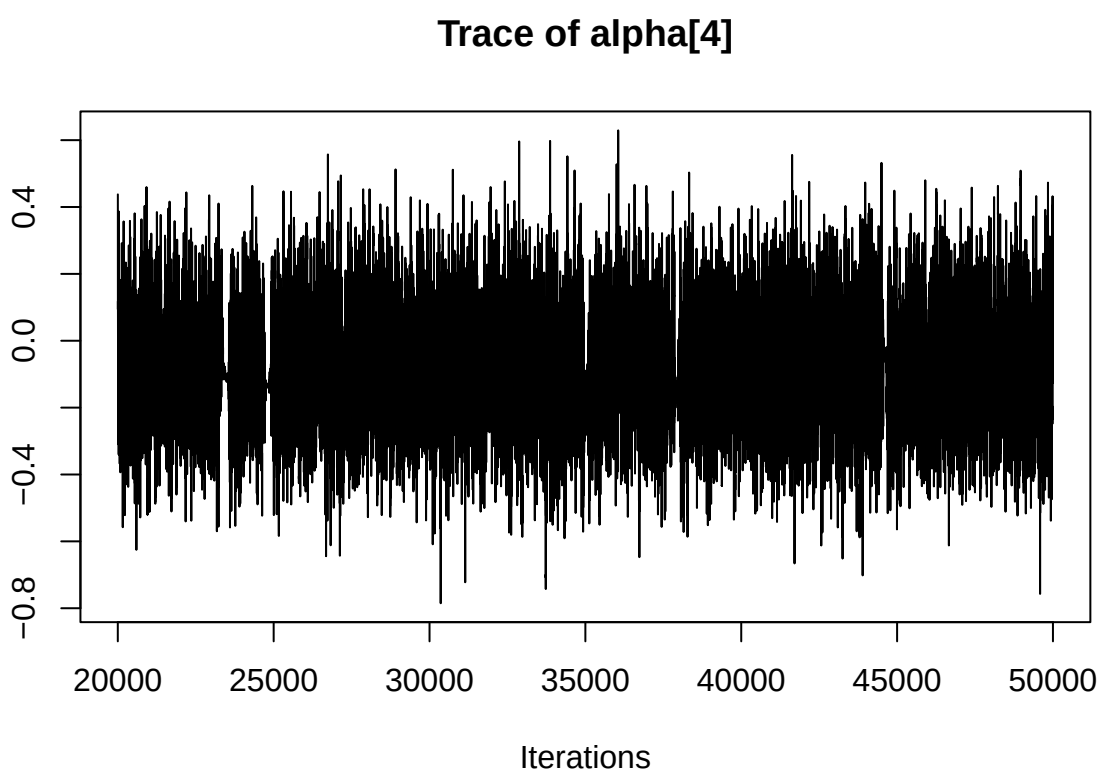
```
##    alpha[1]    alpha[2]    alpha[3]    alpha[4]    alpha[5]    alpha[6]
##      362.54     2222.50     1958.60     1881.29      769.51     2334.86
##    alpha[7]    alpha[8]     beta[1]     beta[2]     beta[3]     beta[4]
##     1600.65     1630.39      522.25     2237.81     2485.14     2046.31
##     beta[5]     beta[6]     beta[7]     beta[8]    mu_alpha     mu_beta
##      897.24     2379.70     1836.31     1632.58     2193.41     3458.82
## sigma_alpha  sigma_beta
##      493.17      609.02
```

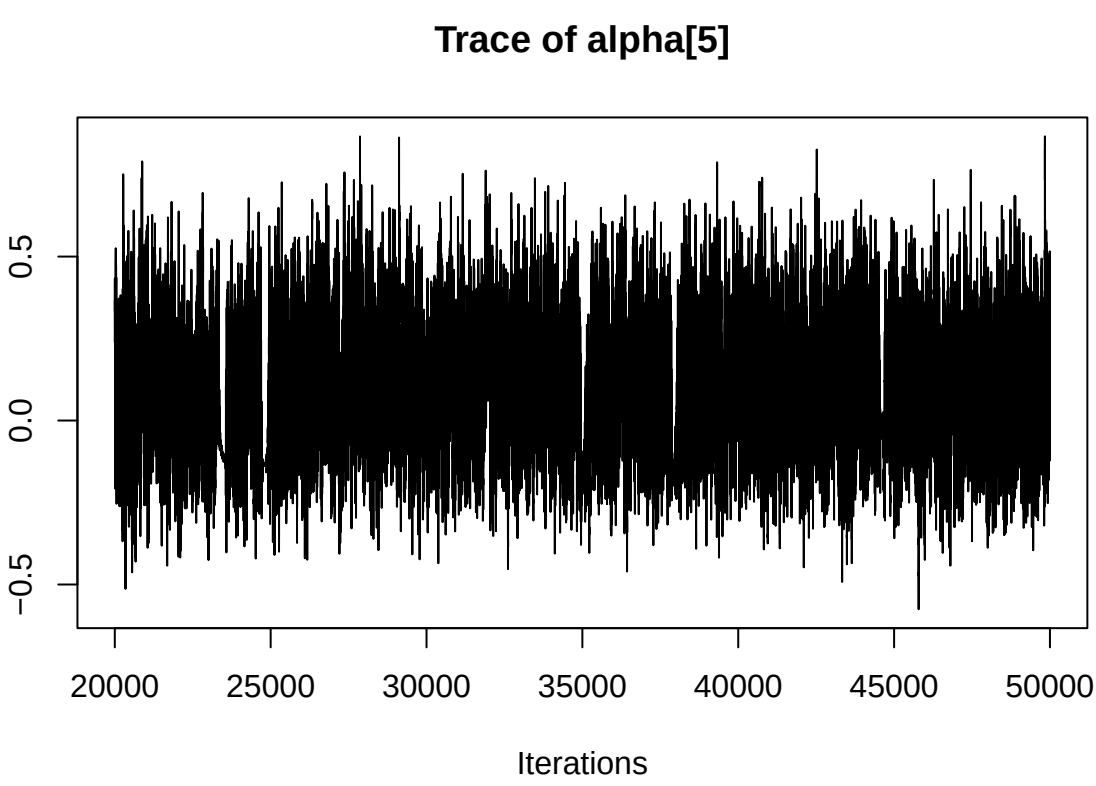
```r
traceplot(fit.samples)
```

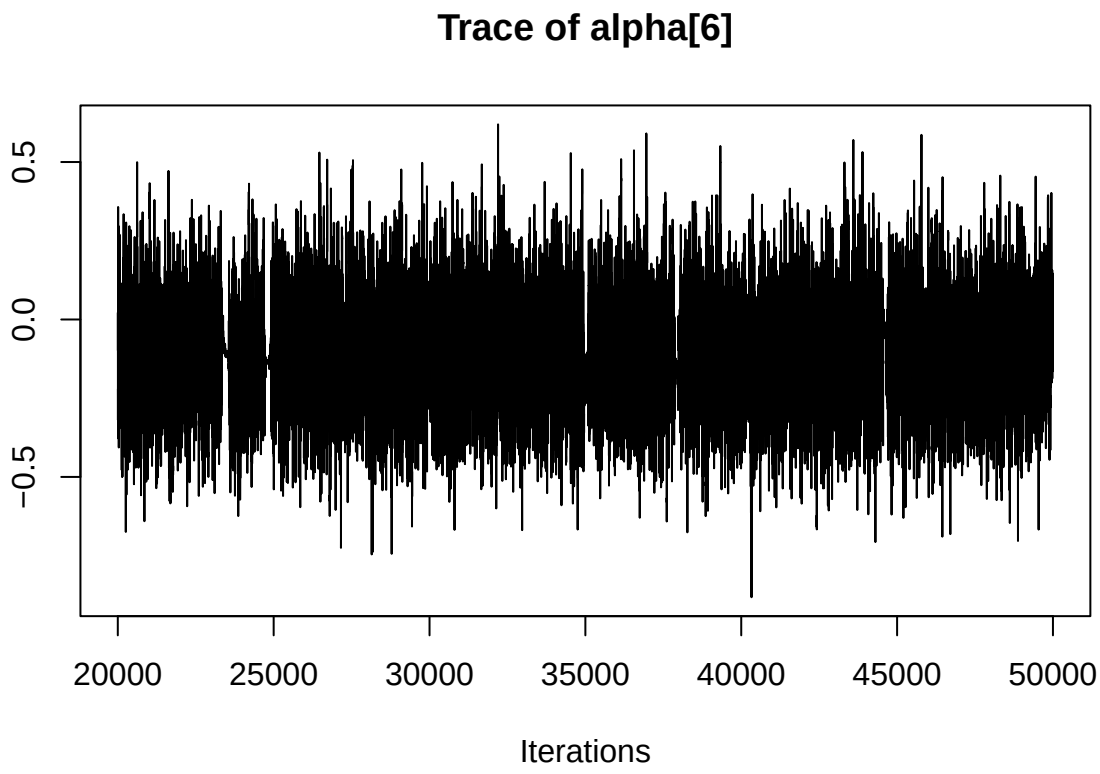

Trace of alpha[1]
0.0
−0.5
−1.0
−1.5
20000
25000
30000
35000
40000
45000
50000
Iterations

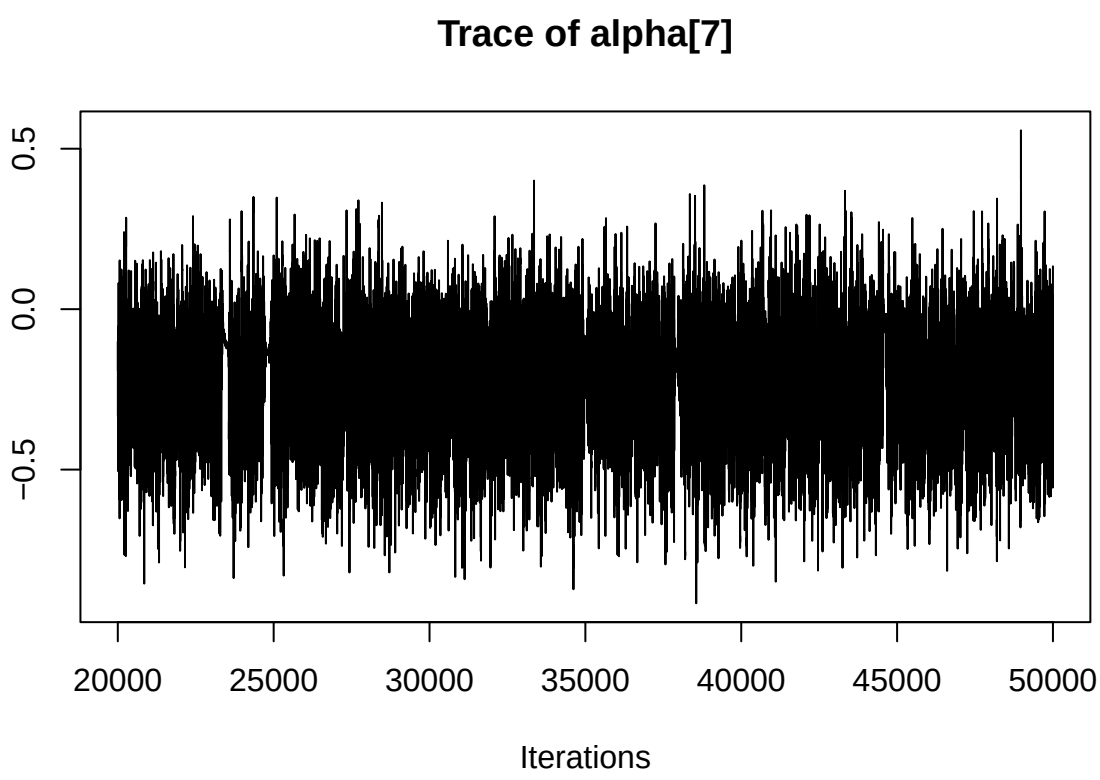

Trace of alpha[2]
0.4
0.0
−0.4
−0.8
20000
25000
30000
35000
40000
45000
50000
Iterations

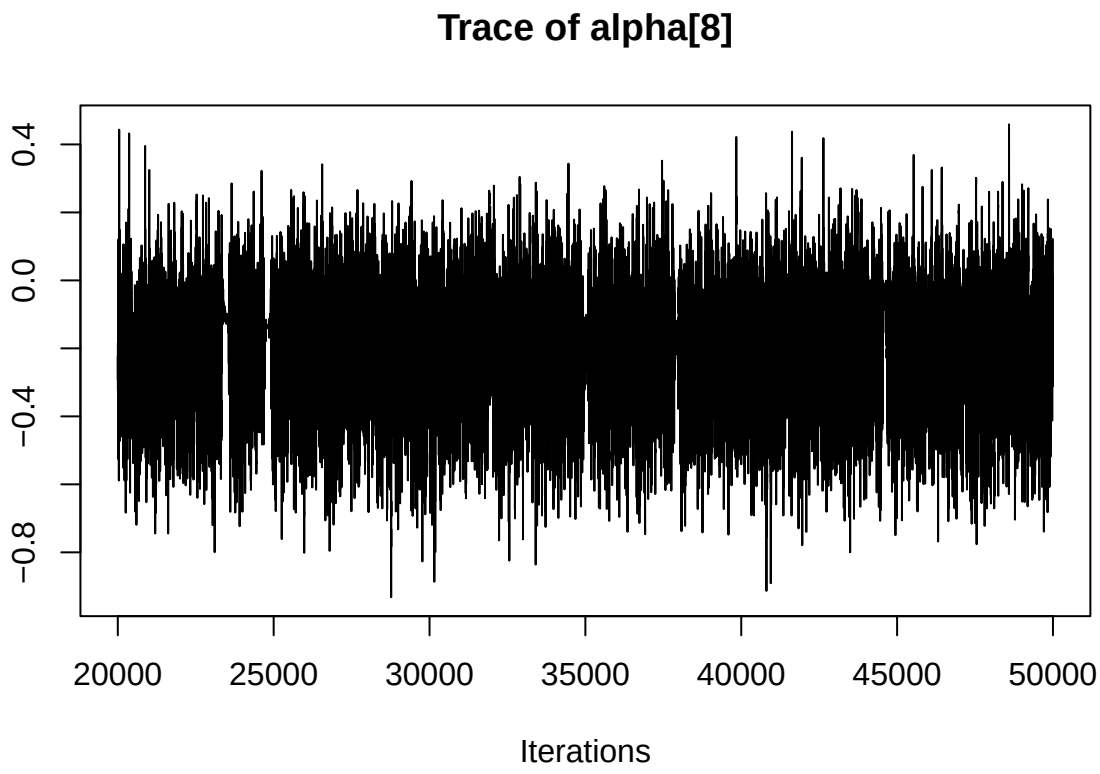

Trace of alpha[3]
0.5
0.0
−0.5
20000
25000
30000
35000
40000
45000
50000
Iterations

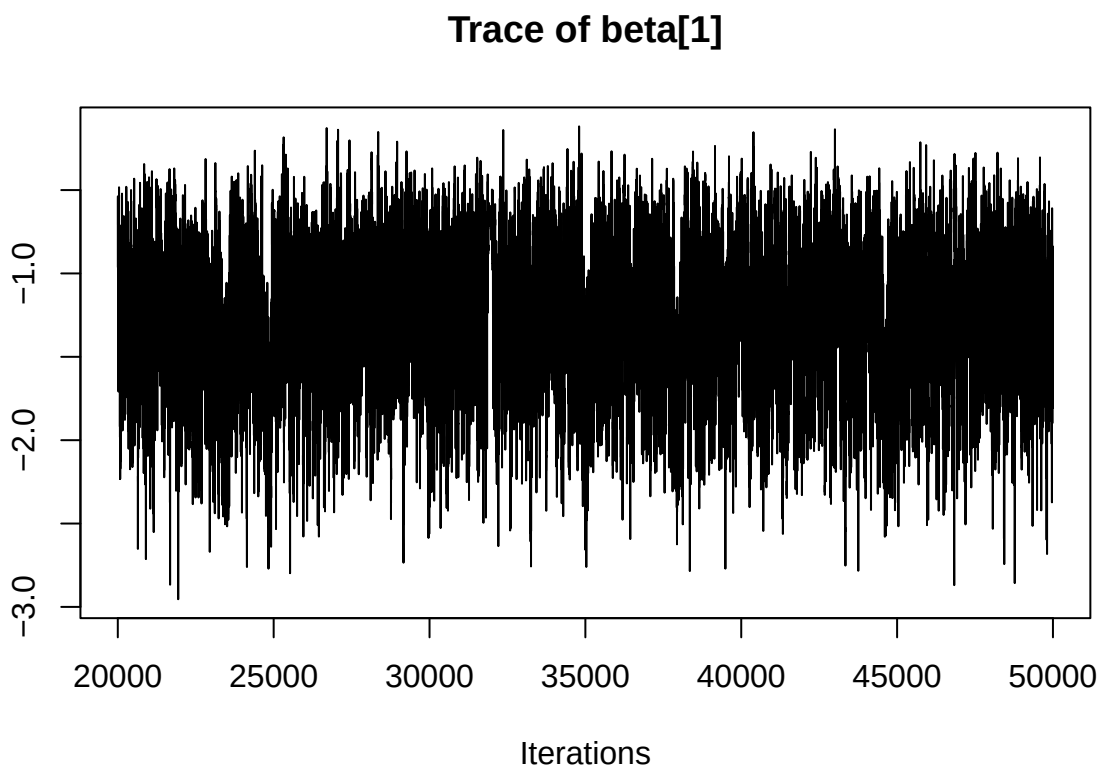

Trace of alpha[4]
0.4
0.0
−0.4
−0.8
20000
25000
30000
35000
40000
45000
50000
Iterations

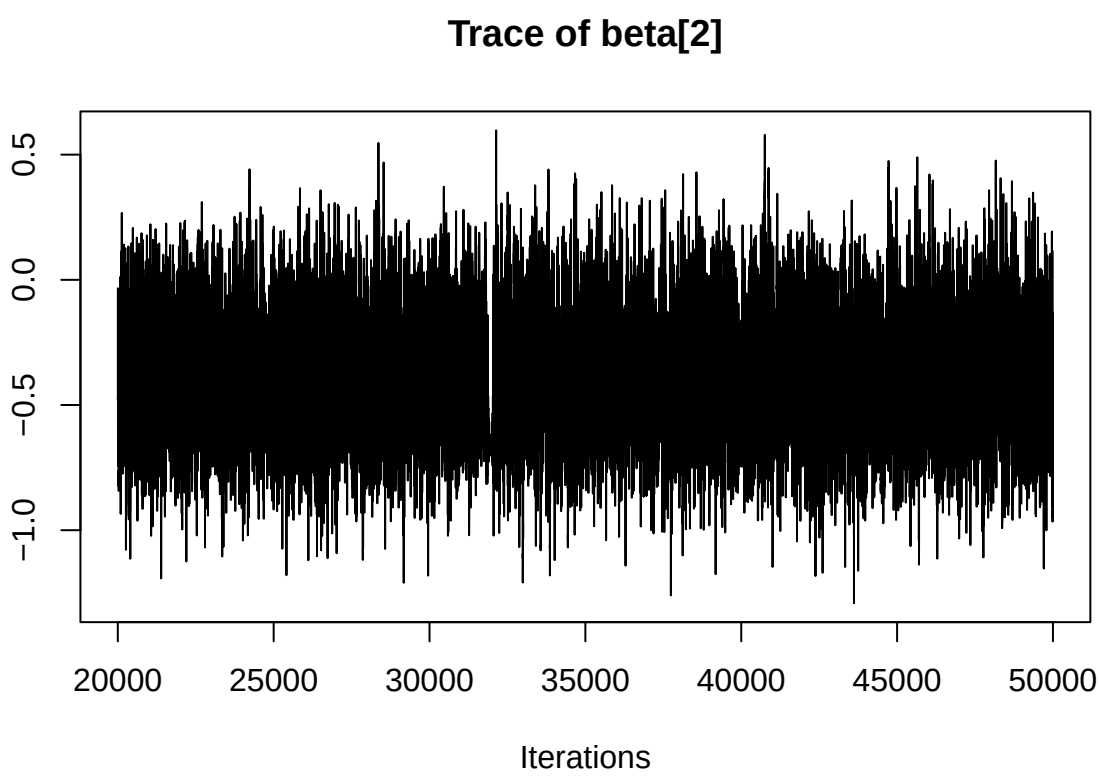

Trace of alpha[5]
0.5
0.0
−0.5
20000
25000
30000
35000
40000
45000
50000
Iterations

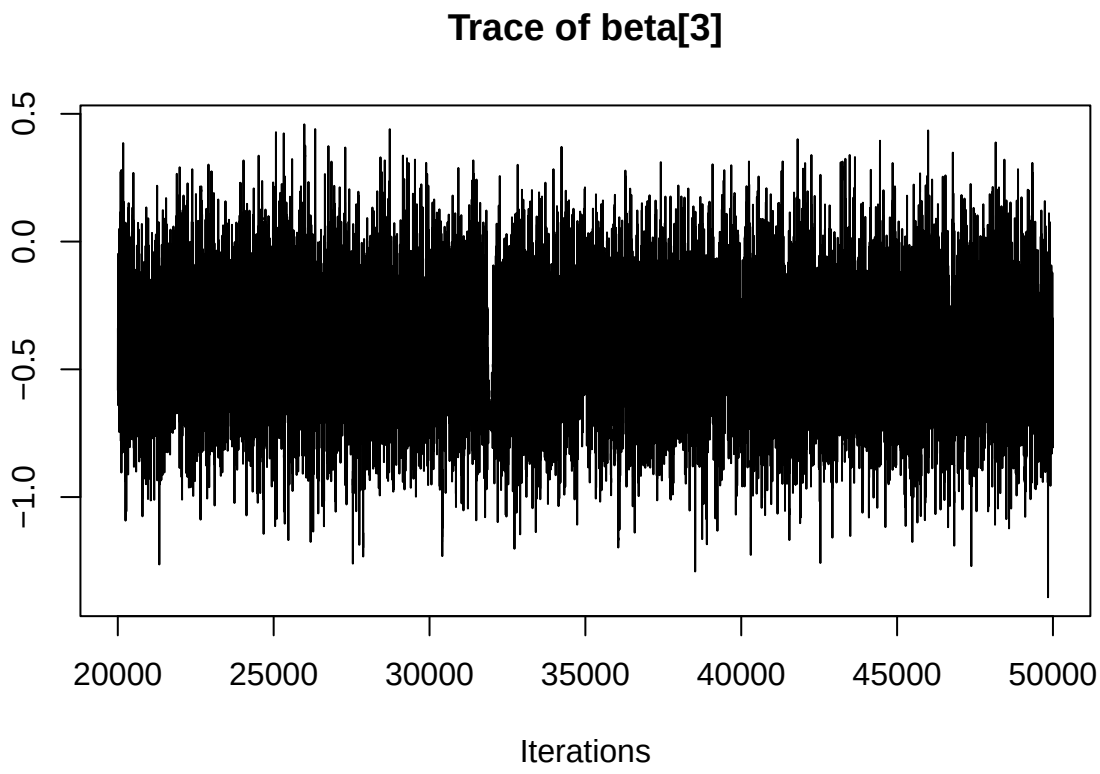

Trace of alpha[6]
0.5
0.0
−0.5
20000
25000
30000
35000
40000
45000
50000
Iterations

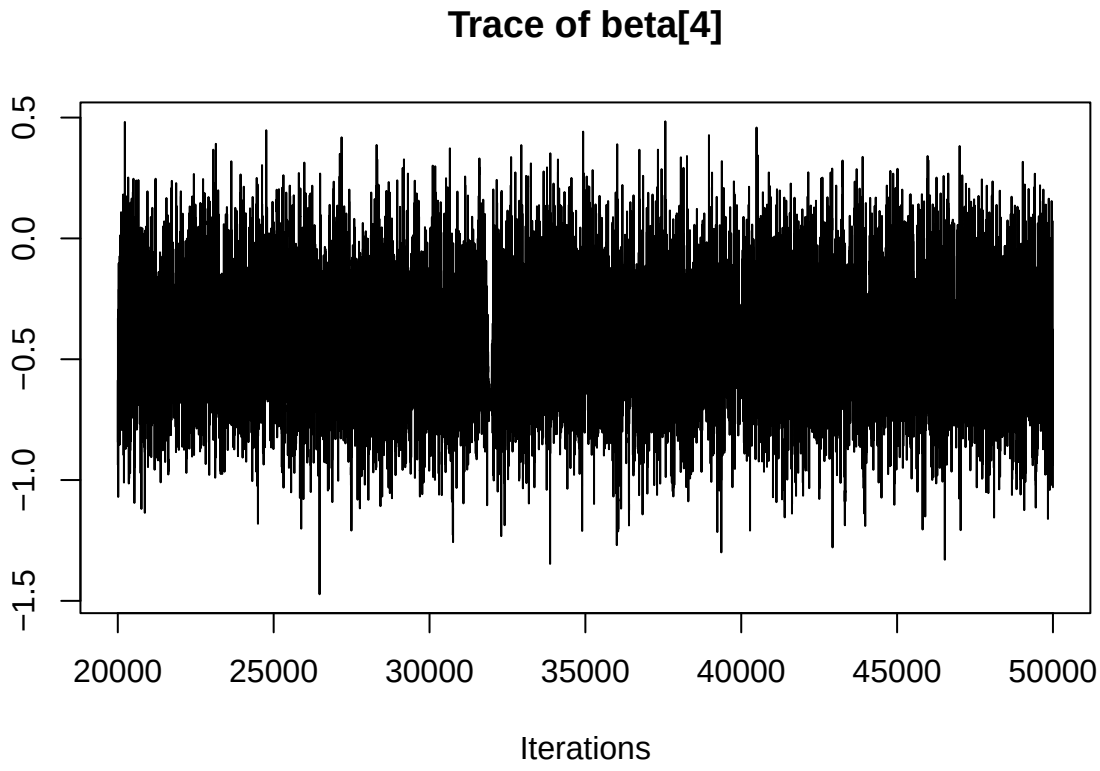

Trace of alpha[7]
0.5
0.0
−0.5
20000
25000
30000
35000
40000
45000
50000
Iterations

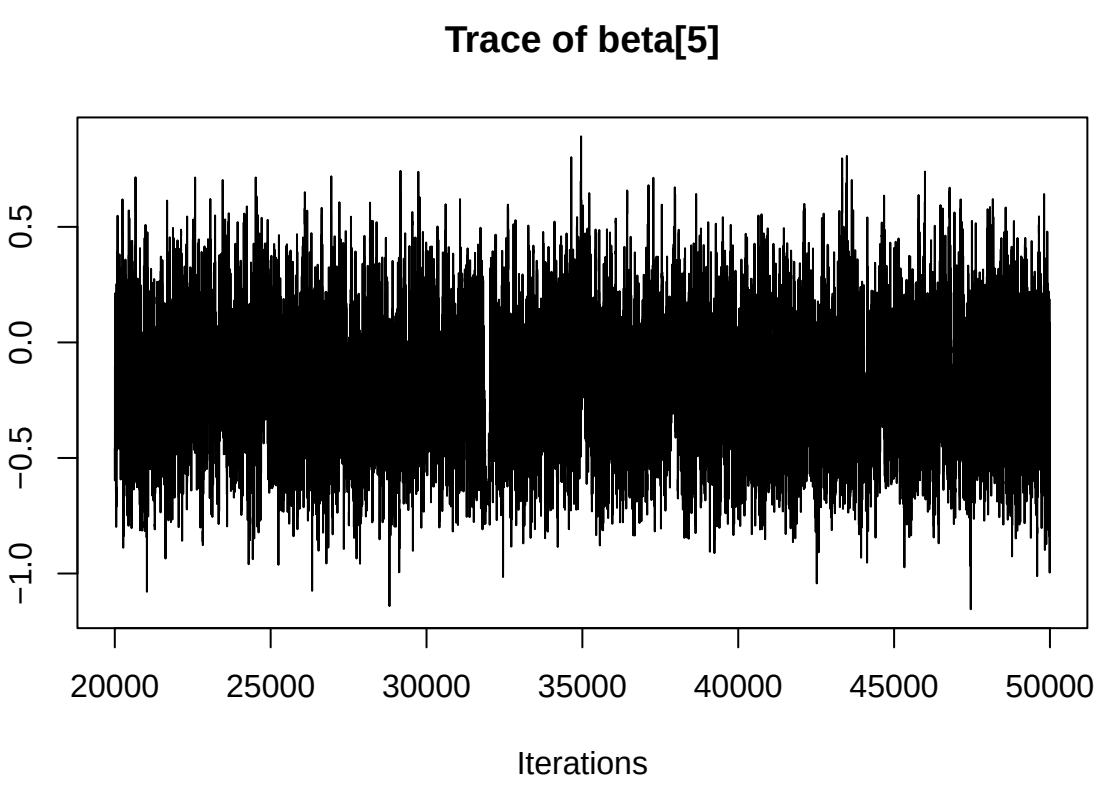

Trace of alpha[8]
0.4
0.0
−0.4
−0.8
20000
25000
30000
35000
40000
45000
50000
Iterations

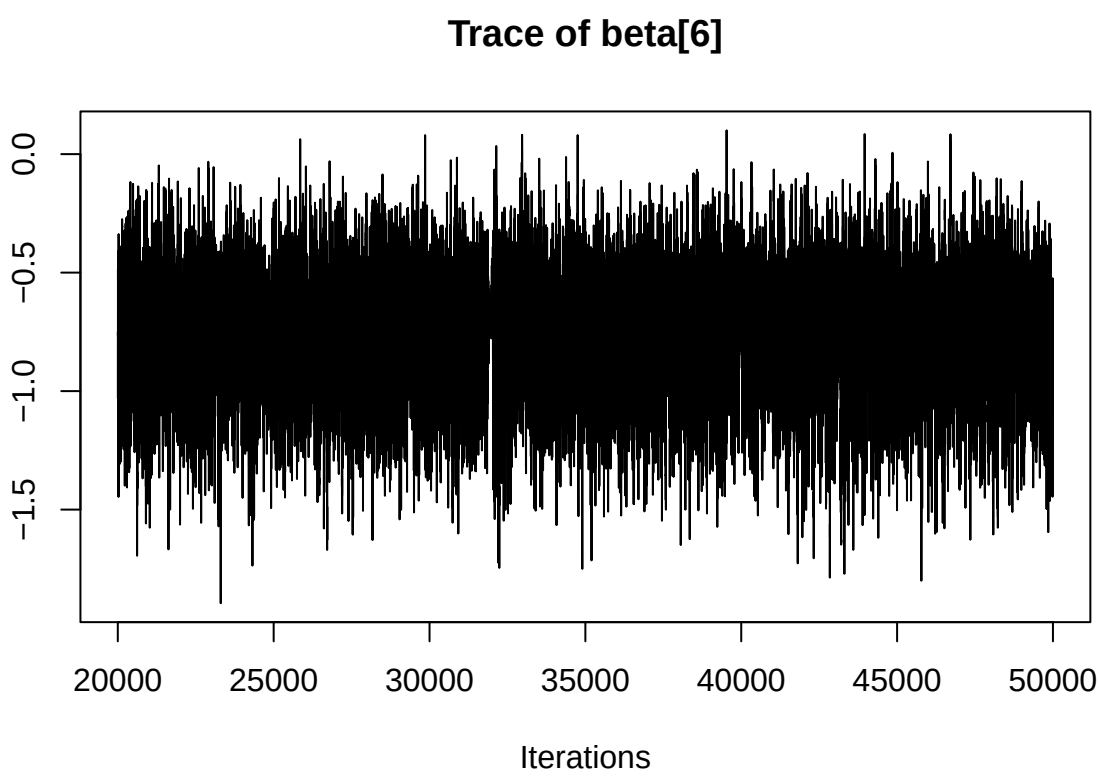

Trace of beta[1]
−1.0
−2.0
−3.0
20000
25000
30000
35000
40000
45000
50000
Iterations

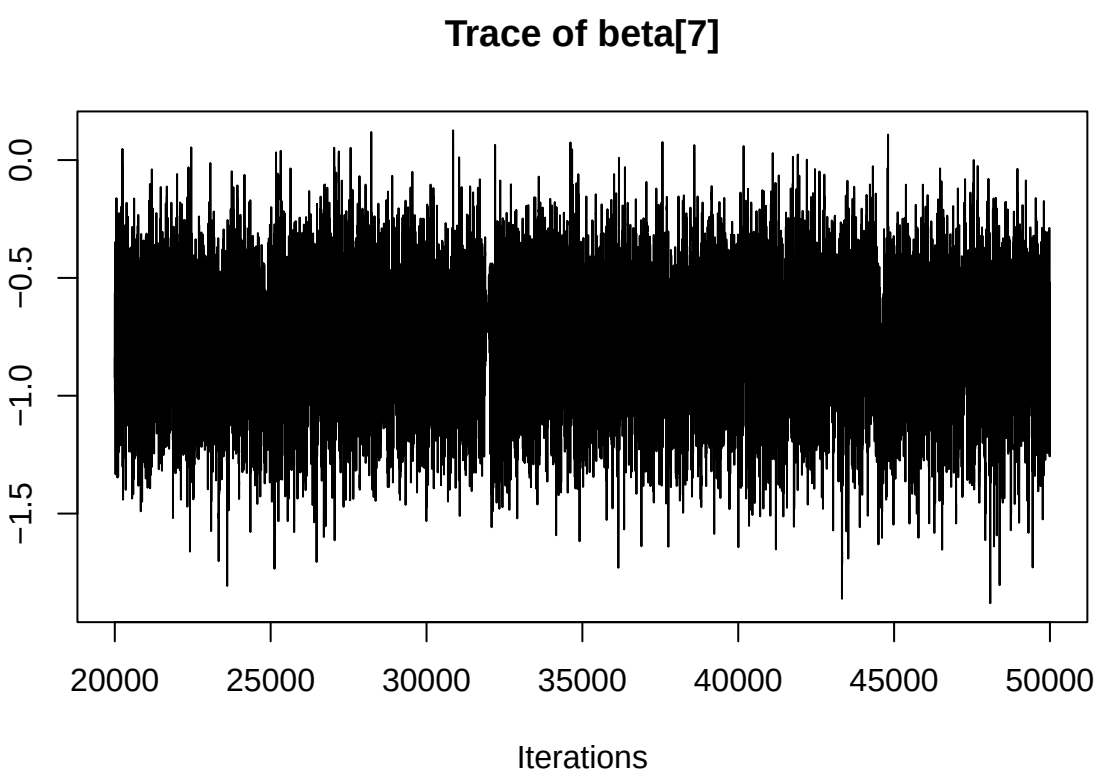

Trace of beta[2]
0.5
0.0
−0.5
−1.0
20000
25000
30000
35000
40000
45000
50000
Iterations

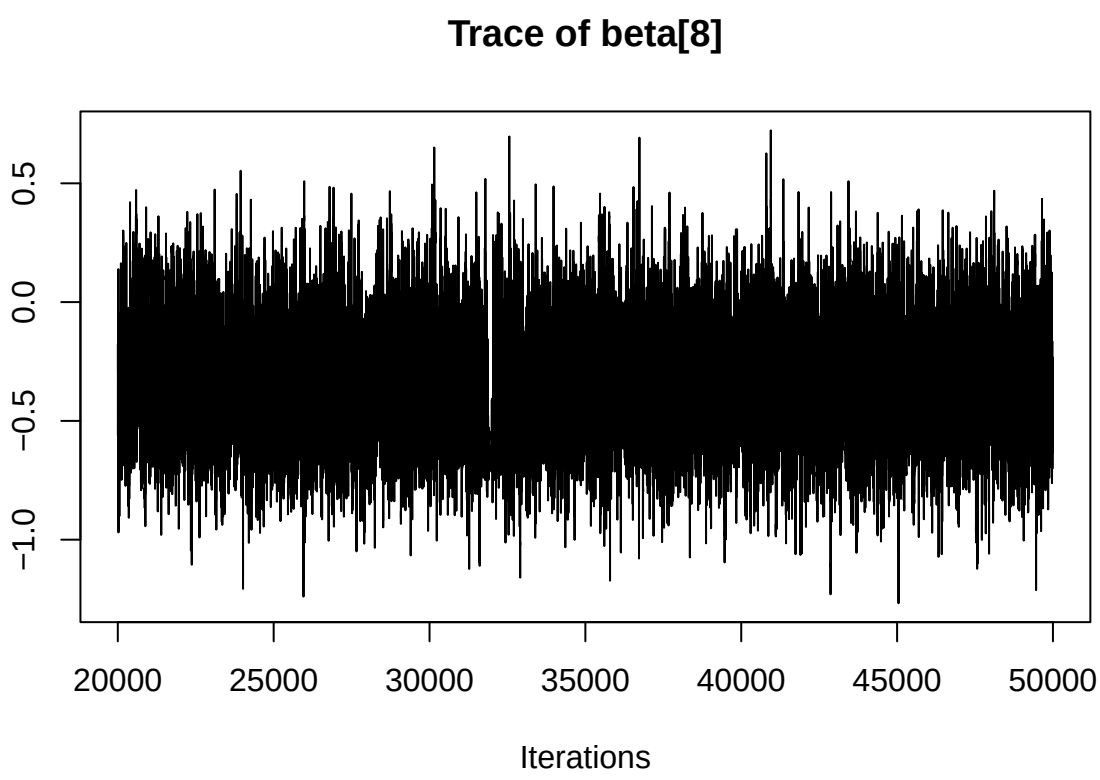

Trace of beta[3]
0.5
0.0
−0.5
−1.0
20000
25000
30000
35000
40000
45000
50000
Iterations

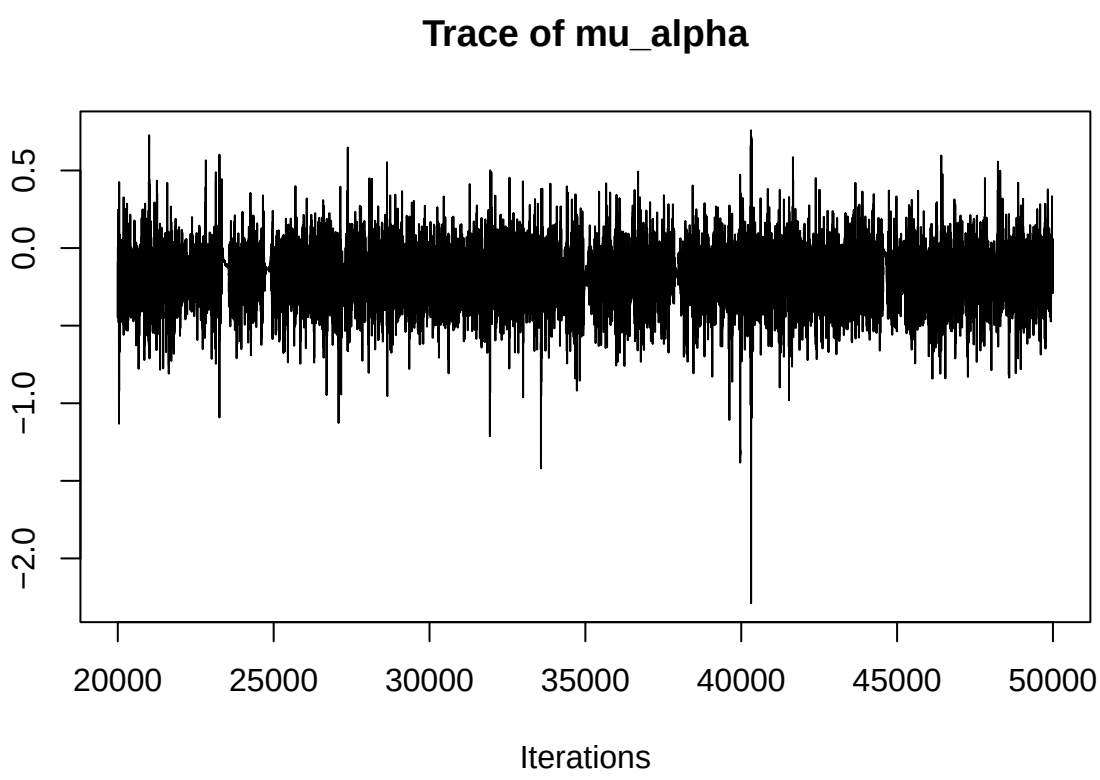

Trace of beta[4]
0.5
0.0
−0.5
−1.0
−1.5
20000
25000
30000
35000
40000
45000
50000
Iterations

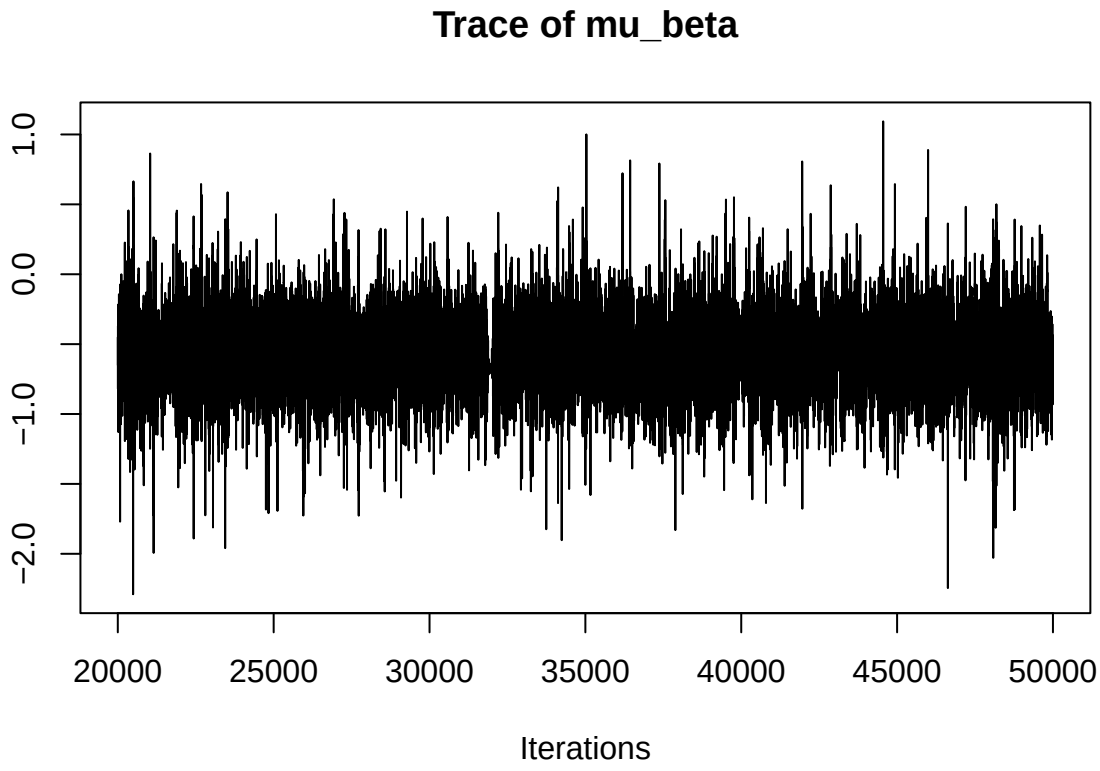

Trace of beta[5]
0.5
0.0
−0.5
−1.0
20000
25000
30000
35000
40000
45000
50000
Iterations

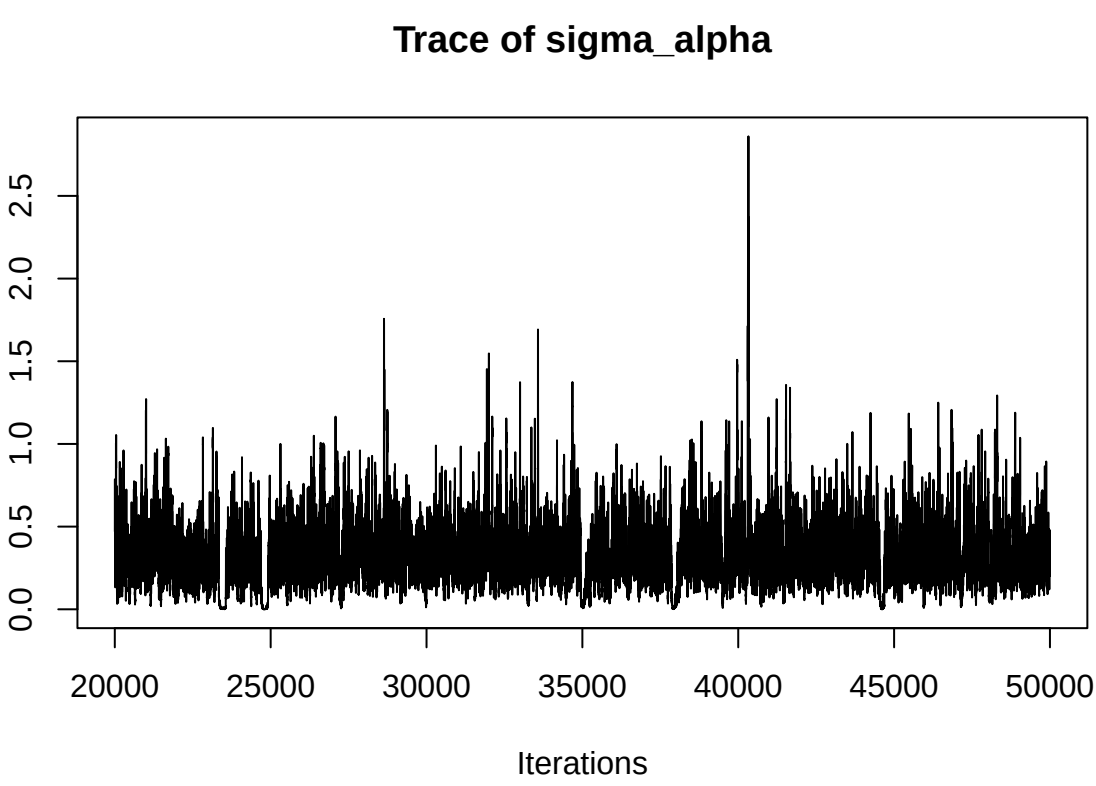

Trace of beta[6]
0.0
−0.5
−1.0
−1.5
20000
25000
30000
35000
40000
45000
50000
Iterations

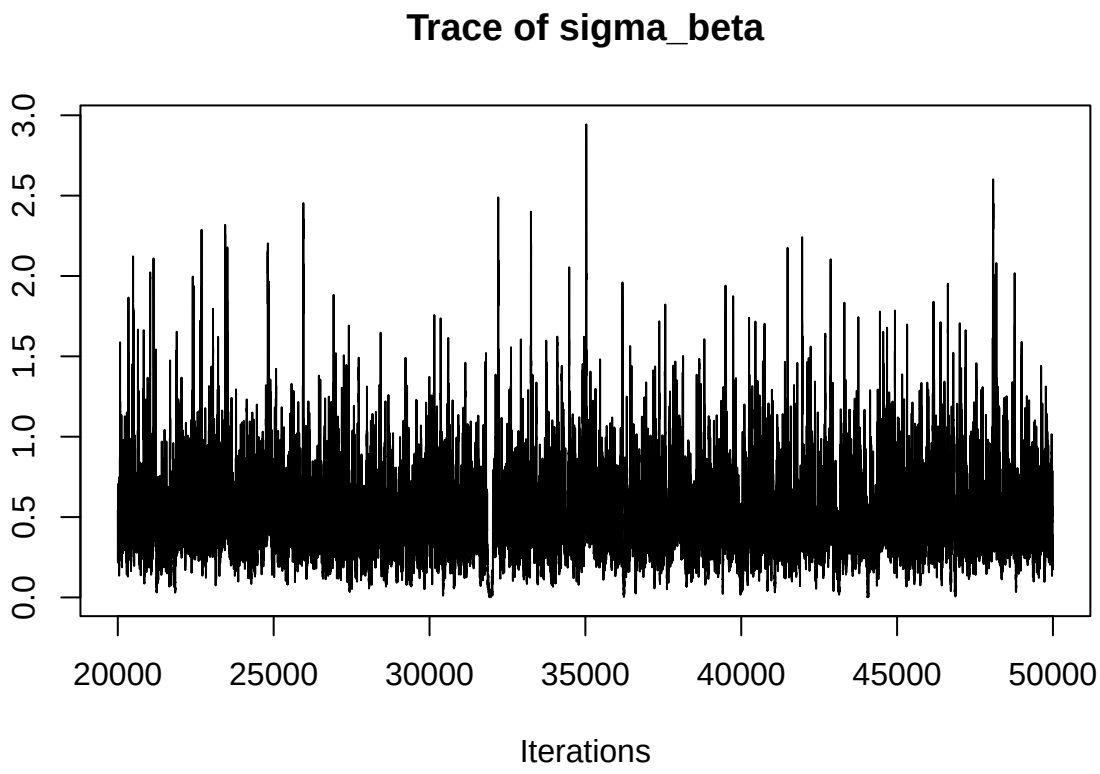

Trace of beta[7]
0.0
−0.5
−1.0
−1.5
20000
25000
30000
35000
40000
45000
50000
Iterations


Trace of beta[8]
0.5
0.0
−0.5
−1.0
20000
25000
30000
35000
40000
45000
50000
Iterations


Trace of mu_alpha
0.5
0.0
−1.0
−2.0
20000
25000
30000
35000
40000
45000
50000
Iterations

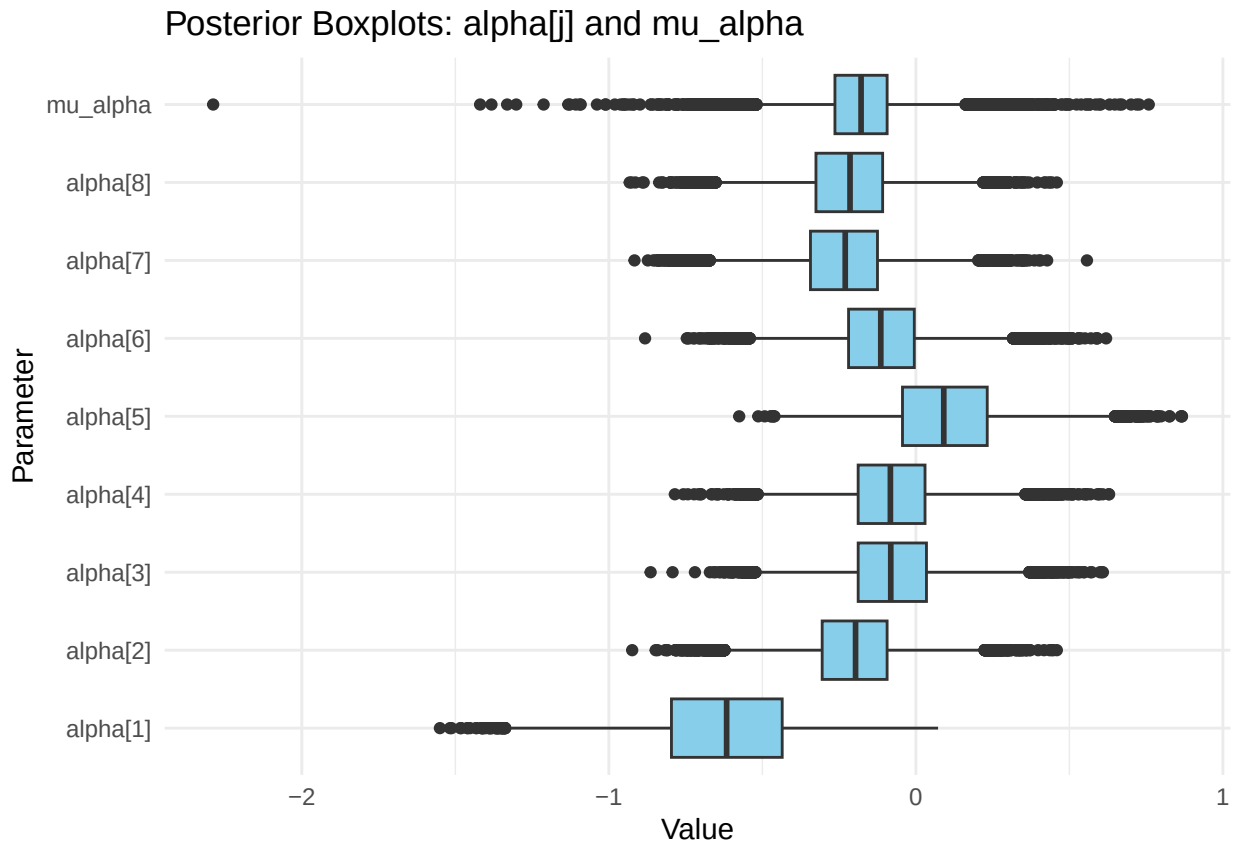

Trace of mu_beta
1.0
0.0
−1.0
−2.0
20000
25000
30000
35000
40000
45000
50000
Iterations


Trace of sigma_alpha
0.0
0.5
1.0
1.5
2.0
2.5
20000
25000
30000
35000
40000
45000
50000
Iterations

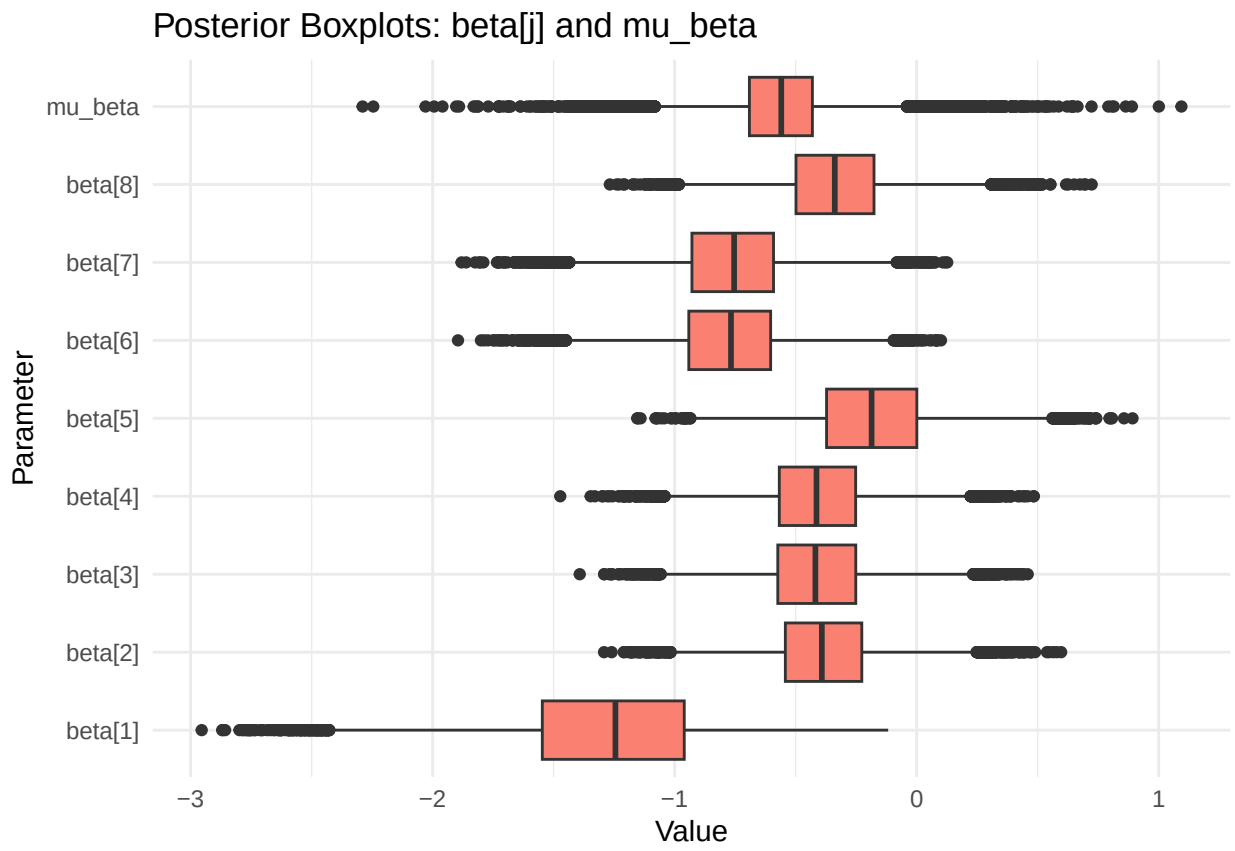


b

```r
library(ggplot2)
library(tidyr)
library(dplyr)

# Convert MCMC samples to data frame
samples_df <- as.data.frame(as.matrix(fit.samples))

# ---------- Boxplot for alpha[j] and mu_alpha ----------
alpha_cols <- grep("^alpha\\[", names(samples_df), value = TRUE)

alpha_long <- samples_df %>%
  select(all_of(alpha_cols), mu_alpha) %>%
  pivot_longer(cols = everything(), names_to = "Parameter", values_to = "Value")

ggplot(alpha_long, aes(x = Parameter, y = Value)) +
  geom_boxplot(fill = "skyblue") +
  theme_minimal() +
  coord_flip() +
  labs(title = "Posterior Boxplots: alpha[j] and mu_alpha", y = "Value", x = "Parameter")
```




```
# ---------- Boxplot for beta[j] and mu_beta ----------
beta_cols <- grep("^beta\\[", names(samples_df), value = TRUE)

beta_long <- samples_df %>%
  select(all_of(beta_cols), mu_beta) %>%
  pivot_longer(cols = everything(), names_to = "Parameter", values_to = "Value")

ggplot(beta_long, aes(x = Parameter, y = Value)) +
  geom_boxplot(fill = "salmon") +
  theme_minimal() +
  coord_flip() +
  labs(title = "Posterior Boxplots: beta[j] and mu_beta", y = "Value", x = "Parameter")
```



c

```r
set.seed(123)

# Inverse logit function
invlogit <- function(x) {
  1 / (1 + exp(-x))
  }

# Convert samples to matrix
samples_df <- as.data.frame(as.matrix(fit.samples))

# Get posterior draws of mu_alpha, mu_beta, sigma_alpha, sigma_beta
N_samples <- nrow(samples_df)
theta_same <- numeric(N_samples)
theta_diff <- numeric(N_samples)

for (i in 1:N_samples) {
  # Simulate alpha* and beta* for a new school
  alpha_star <- rnorm(1, samples_df$mu_alpha[i], samples_df$sigma_alpha[i])
  beta_star <- rnorm(1, samples_df$mu_beta[i], samples_df$sigma_beta[i])

  # Predict friendship probability
```

```
  theta_same[i] <- invlogit(alpha_star + beta_star * 1)
  theta_diff[i] <- invlogit(alpha_star + beta_star * 0)
}

# Final results
mean(theta_same)  # Predicted probability for x = 1
```

```
## [1] 0.338367
```

```
mean(theta_diff)  # Predicted probability for x = 0
```

```
## [1] 0.456396
```

Problme 2

a

1. α_i: Baseline mean change in ΔPCV for cats of breed i, after adjusting for dose and type.
2. β_1: Overall shift in ΔPCV across all cats, when breed, dose, type, and domestic status are held constant.
3. β_2: Effect of dose on ΔPCV (expected change per unit increase in dose).
4. β_3: Effect of disease type (Type = 1 for primary, 0 for secondary) on ΔPCV.
5. μ_{ij}: Expected ΔPCV for the jth cat of breed i.
6. σ^2: Residual variance in ΔPCV across individual cats.
7. γ: Difference in breed-level baseline ΔPCV between domestic ($= 1$) and non-domestic ($= 0$) cats.
8. τ_α^2: Variance in breed-specific intercepts α_i across all breeds.

b

```
cats <- read.table('/Users/landlee/Desktop/STAT 3303/HW10/cats.txt',
                   header = TRUE)

cats <- cats %>%
  mutate(
    breed_index = as.numeric(factor(Breed, levels = c("A", "B", "C", "D", "E"))),
    domestic = ifelse(Breed %in% c("A", "B", "C"), 1, 0)
  )
```

```
# MCMC parameters
niter <- 10000
nburn <- 1000
nadapt <- 1000

# Number of breeds and total observations
```

```r
Nb <- length(unique(cats$Breed))
N <- nrow(cats)

# Prepare JAGS data
data_jags <- list(
  y = cats$DeltaPCV,
  type = cats$Type,
  dose = cats$Dose,
  breed = cats$breed_index,
  domestic = c(1,1,1,0,0),
  Nb = Nb,
  N = N
)

# Initial values
inits <- list(
  beta1 = 0,
  beta2 = 0,
  beta3 = 0,
  gamma = 0,
  tau = 1,
  tau_alpha = 1,
  alpha = rep(0, Nb)
)
```

```r
model <- "model {
  for (i in 1:N) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha[breed[i]] + beta1 + beta2 * dose[i] + beta3 * type[i]
  }

  for (j in 1:Nb) {
    alpha[j] ~ dnorm(gamma * domestic[j], tau_alpha)
  }

  beta1 ~ dnorm(0, 1/100)
  beta2 ~ dnorm(0, 1/100)
  beta3 ~ dnorm(0, 1/100)
  gamma ~ dnorm(0, 1/100)
  tau ~ dgamma(3, 10)
  tau_alpha ~ dgamma(3, 10)

  sigma <- 1 / sqrt(tau)
}"
```

```r
fit <- jags.model(textConnection(model),
                  data = data_jags,
                  inits = inits,
                  n.chains = 1,
                  n.adapt = nadapt)
```

```
## Compiling model graph
##    Resolving undeclared variables
```

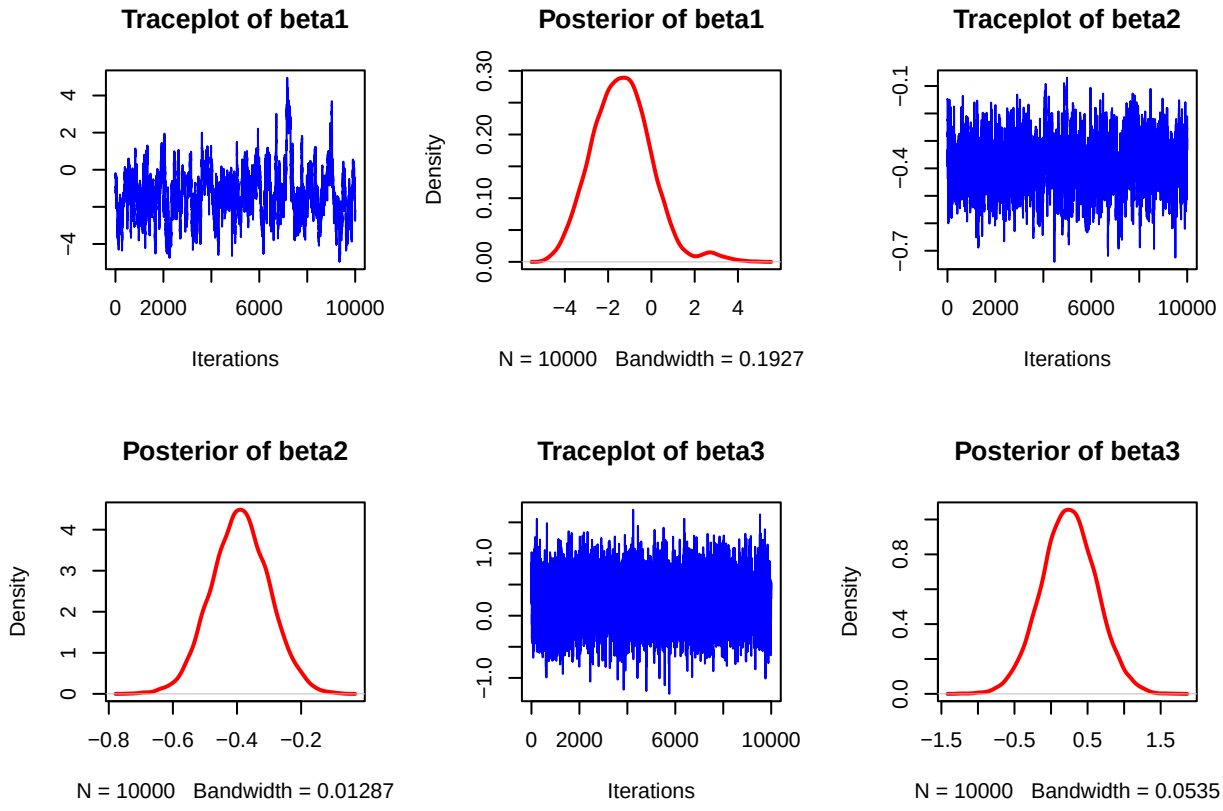
```
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 40
##    Unobserved stochastic nodes: 11
##    Total graph size: 228
##
## Initializing model
```

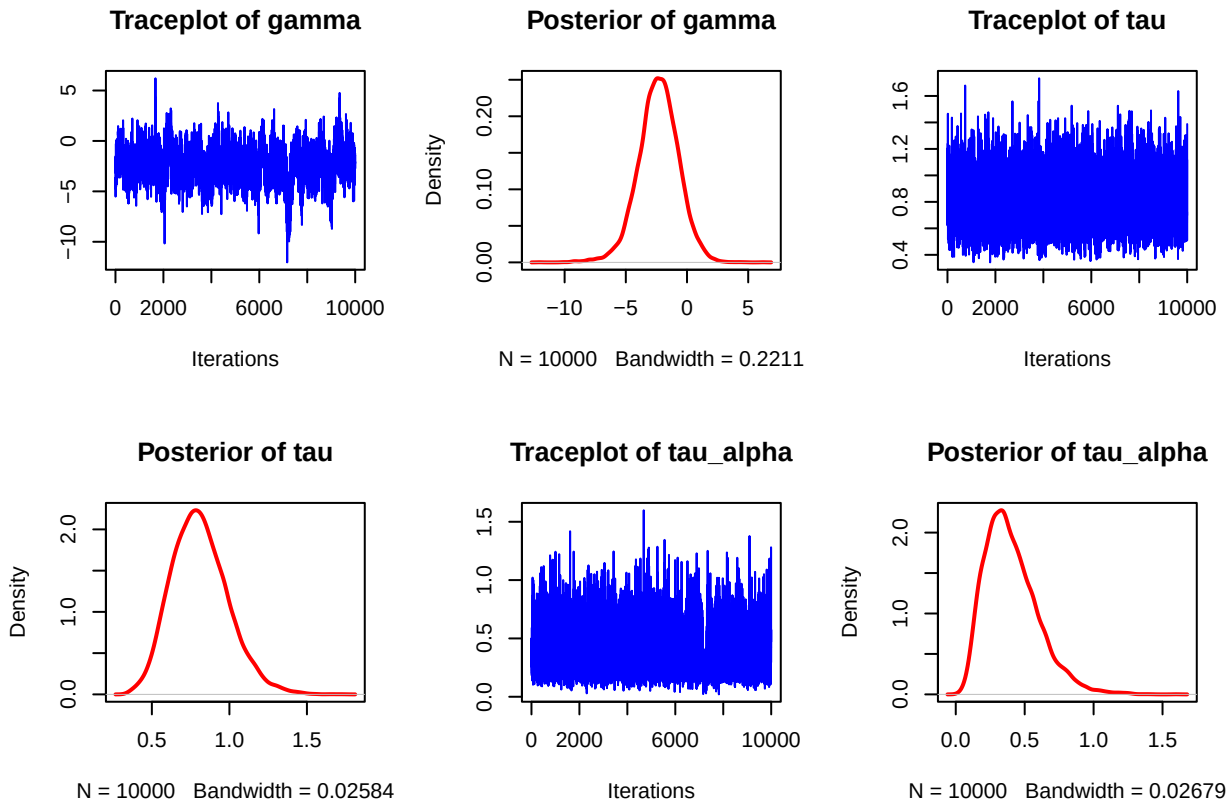
```
params <- c("beta1", "beta2", "beta3", "gamma", "tau", "tau_alpha", "alpha", "sigma")
cat.samples <- coda.samples(fit, variable.names = params, n.iter = niter)
summary(cat.samples)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## alpha[1]  -2.8542 1.37009 0.0137009       0.125250
## alpha[2]  -1.3676 1.35136 0.0135136       0.129822
## alpha[3]  -2.8312 1.33598 0.0133598       0.140356
## alpha[4]   0.3676 1.27734 0.0127734       0.123278
## alpha[5]  -0.5469 1.28079 0.0128079       0.123330
## beta1     -1.3596 1.38097 0.0138097       0.149997
## beta2     -0.3899 0.09021 0.0009021       0.003651
## beta3      0.2428 0.37526 0.0037526       0.007299
## gamma     -2.3372 1.65047 0.0165047       0.129536
## sigma      1.1327 0.13101 0.0013101       0.001657
## tau        0.8101 0.18209 0.0018209       0.002227
## tau_alpha  0.4054 0.19229 0.0019229       0.003683
##
## 2. Quantiles for each variable:
##
##              2.5%       25%     50%     75%   97.5%
## alpha[1]  -5.7474 -3.706496 -2.8293 -1.9179 -0.3065
## alpha[2]  -4.2479 -2.214109 -1.3626 -0.4341  1.1510
## alpha[3]  -5.6908 -3.669838 -2.8101 -1.9253 -0.3786
## alpha[4]  -2.3734 -0.402899  0.3957  1.2240  2.6867
## alpha[5]  -3.2498 -1.339166 -0.5164  0.3378  1.7429
## beta1     -3.8454 -2.310527 -1.4022 -0.4998  1.5853
## beta2     -0.5647 -0.450291 -0.3902 -0.3289 -0.2105
## beta3     -0.4979 -0.007459  0.2416  0.4952  0.9780
## gamma     -5.7262 -3.338920 -2.3103 -1.2619  0.8444
## sigma      0.9118  1.040459  1.1198  1.2118  1.4242
## tau        0.4930  0.680980  0.7975  0.9237  1.2029
## tau_alpha  0.1233  0.264557  0.3742  0.5162  0.8530
```

```
# Convert MCMC samples to matrix
mcmc_matrix <- as.matrix(cat.samples)
```

```r
# Parameters to plot
params <- c("beta1", "beta2", "beta3", "gamma", "tau", "tau_alpha")

# Set layout: 2 rows, 3 columns
par(mfrow = c(2, 3))

# Loop over each parameter
for (param in params) {
  # Traceplot for one parameter at a time
  traceplot(cat.samples[, param],
            main = paste("Traceplot of", param),
            col = "blue", lwd = 1)

  # Density plot
  plot(density(mcmc_matrix[, param]),
       main = paste("Posterior of", param),
       col = "red", lwd = 2)
}
```

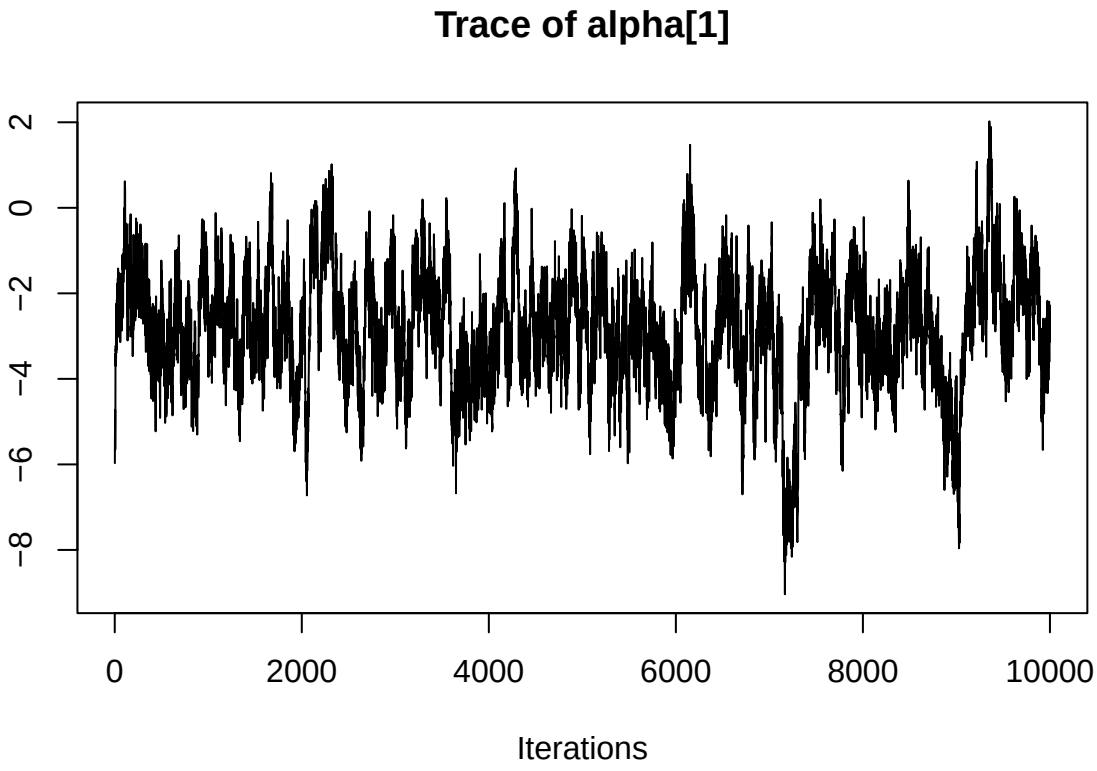


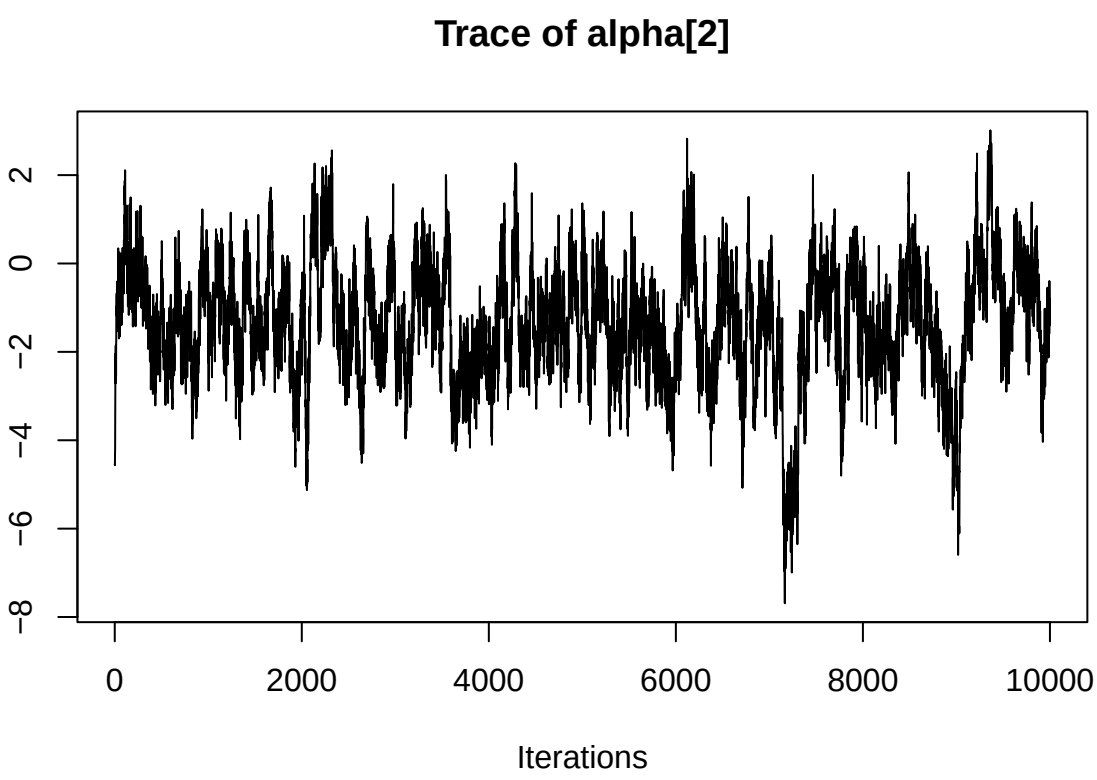


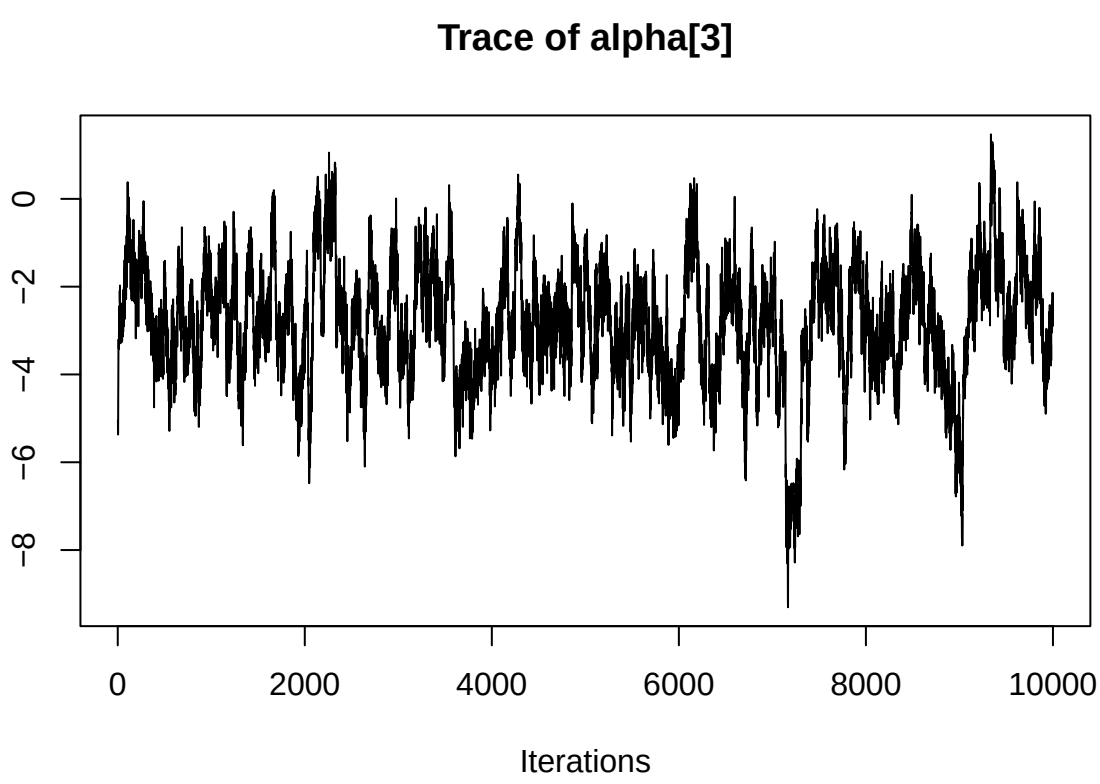

```
ess <- effectiveSize(cat.samples)
print(round(ess, 2))
```

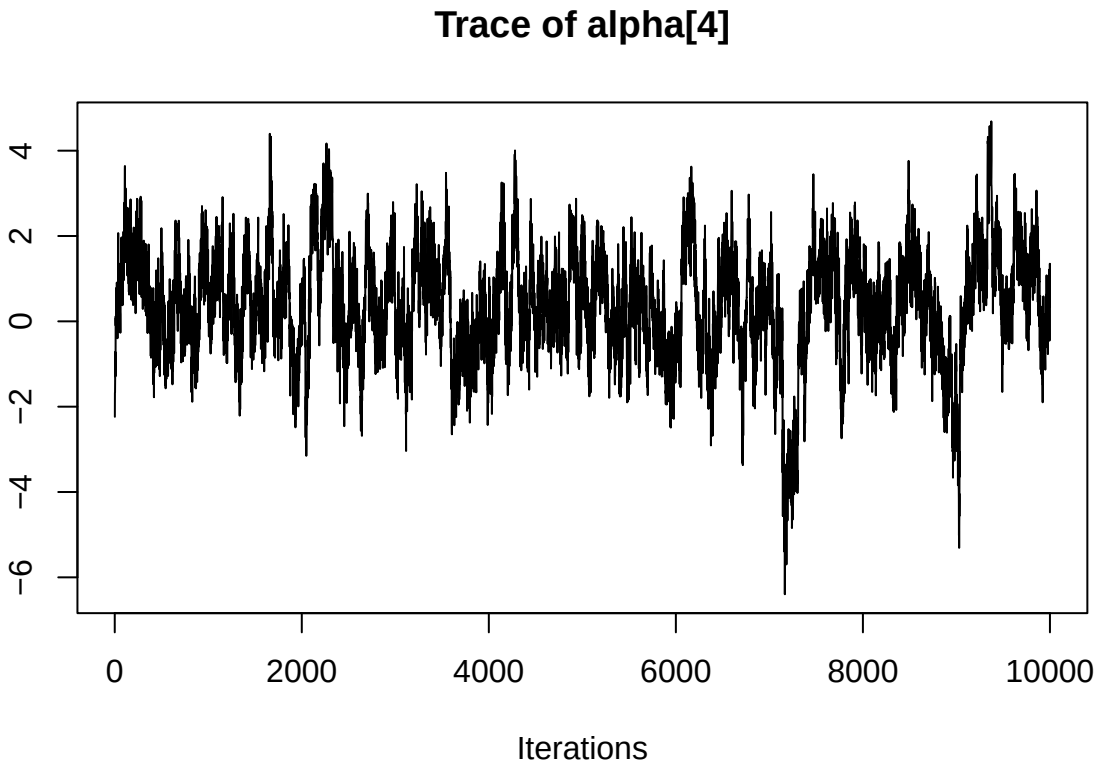
```
##  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]     beta1     beta2     beta3
##    119.66    108.35     90.60    107.36    107.85     84.76    610.65   2643.02
##     gamma     sigma       tau tau_alpha
##    162.34   6249.89   6683.84   2726.16
```

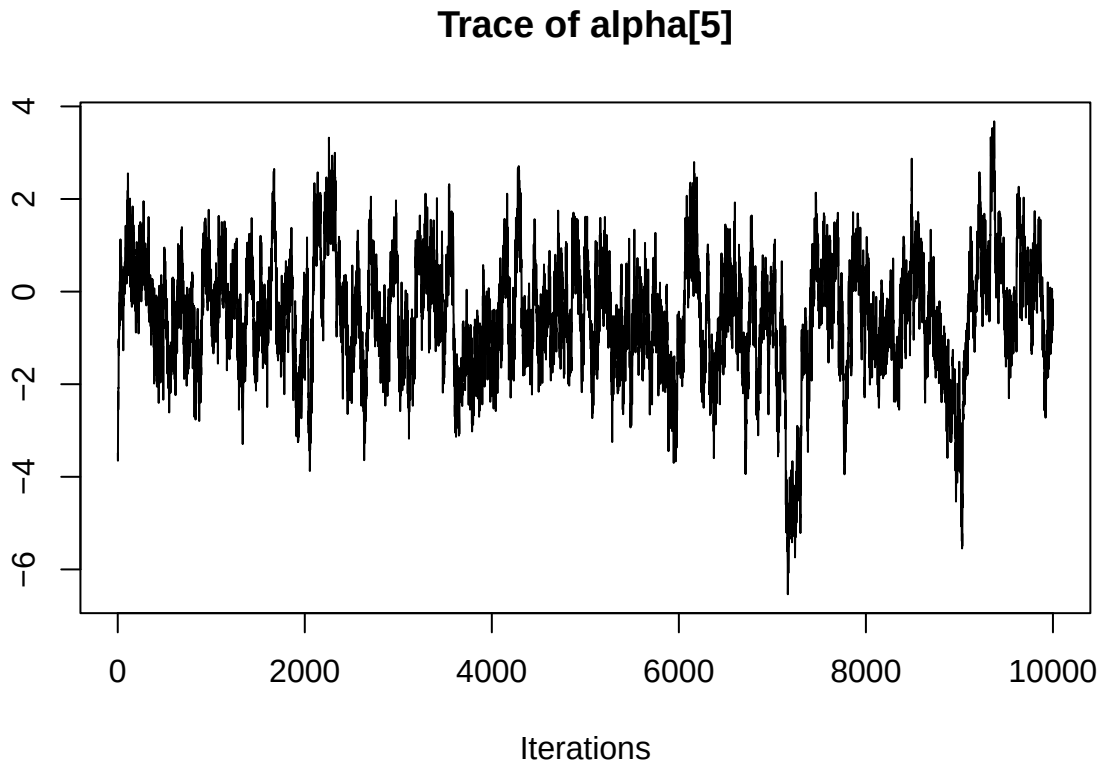
```
traceplot(cat.samples)
```

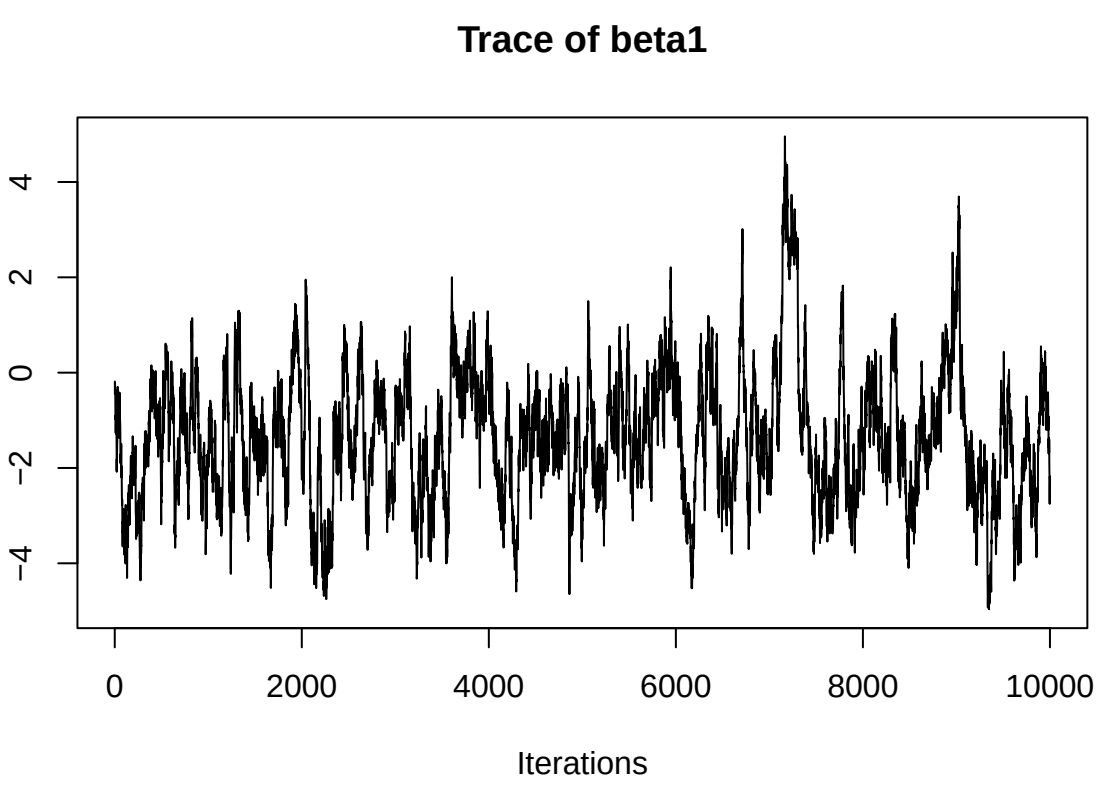

Trace of alpha[1]
2
0
−2
−4
−6
−8
0
2000
4000
6000
8000
10000
Iterations

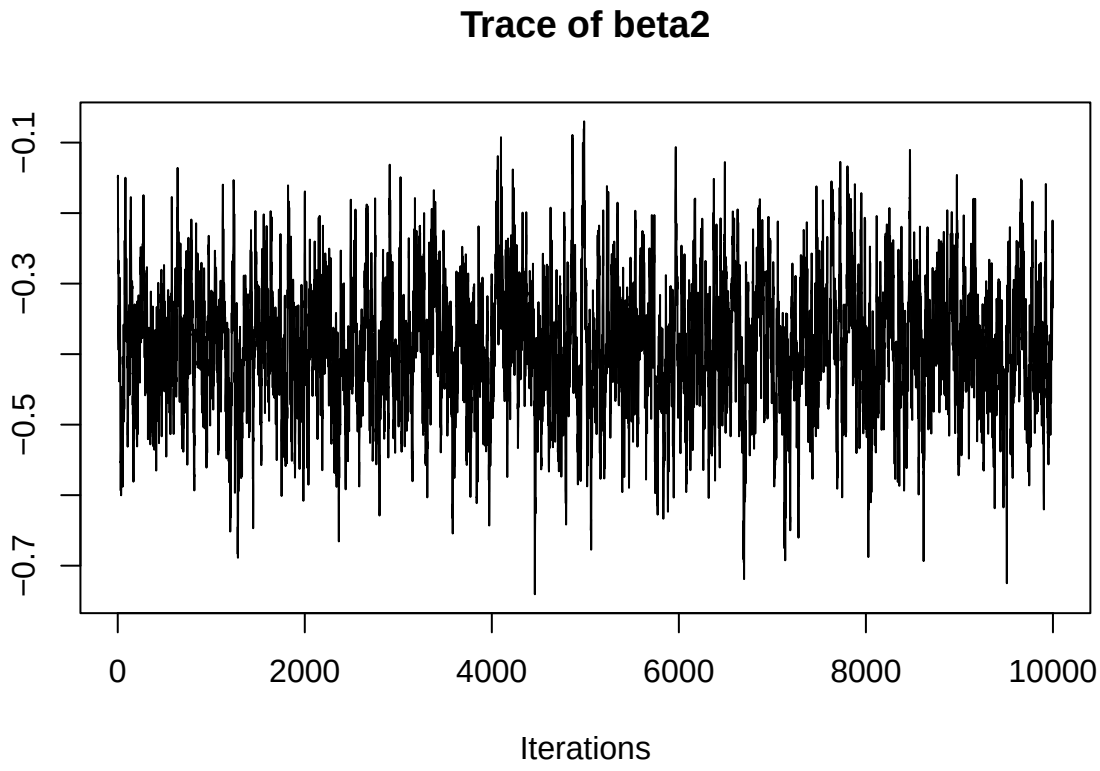

Trace of alpha[2]
2
0
−2
−4
−6
−8
0
2000
4000
6000
8000
10000
Iterations

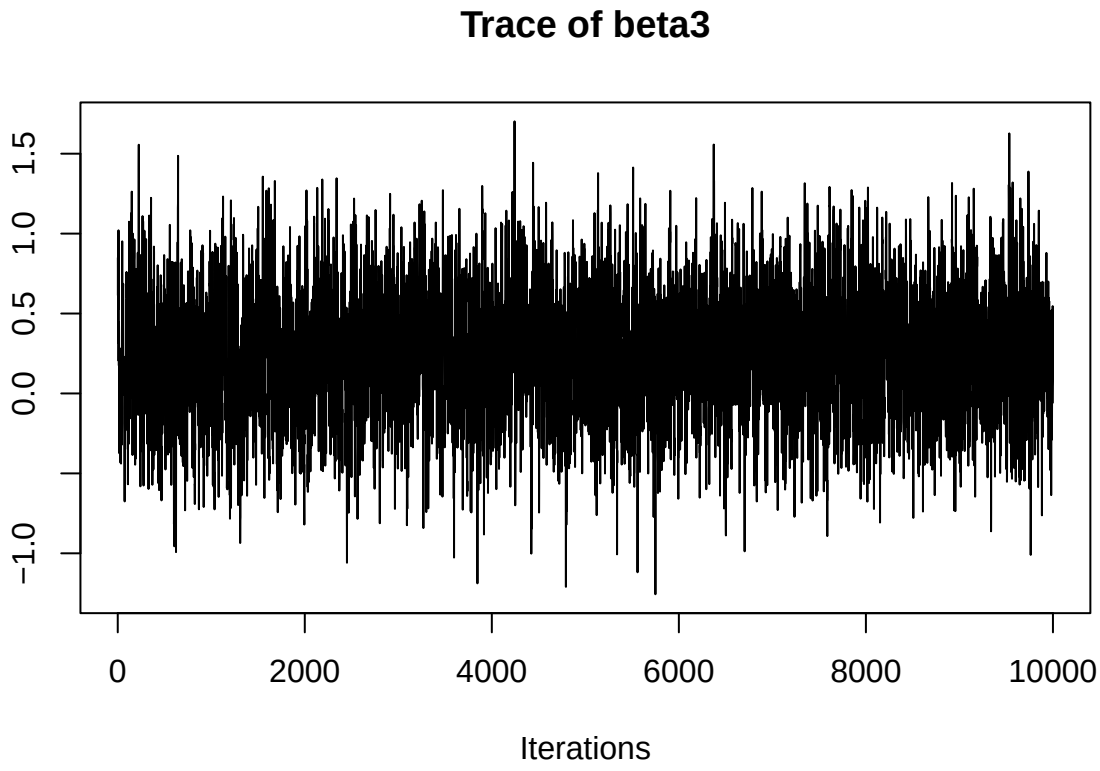

Trace of alpha[3]
0
−2
−4
−6
−8
0
2000
4000
6000
8000
10000
Iterations

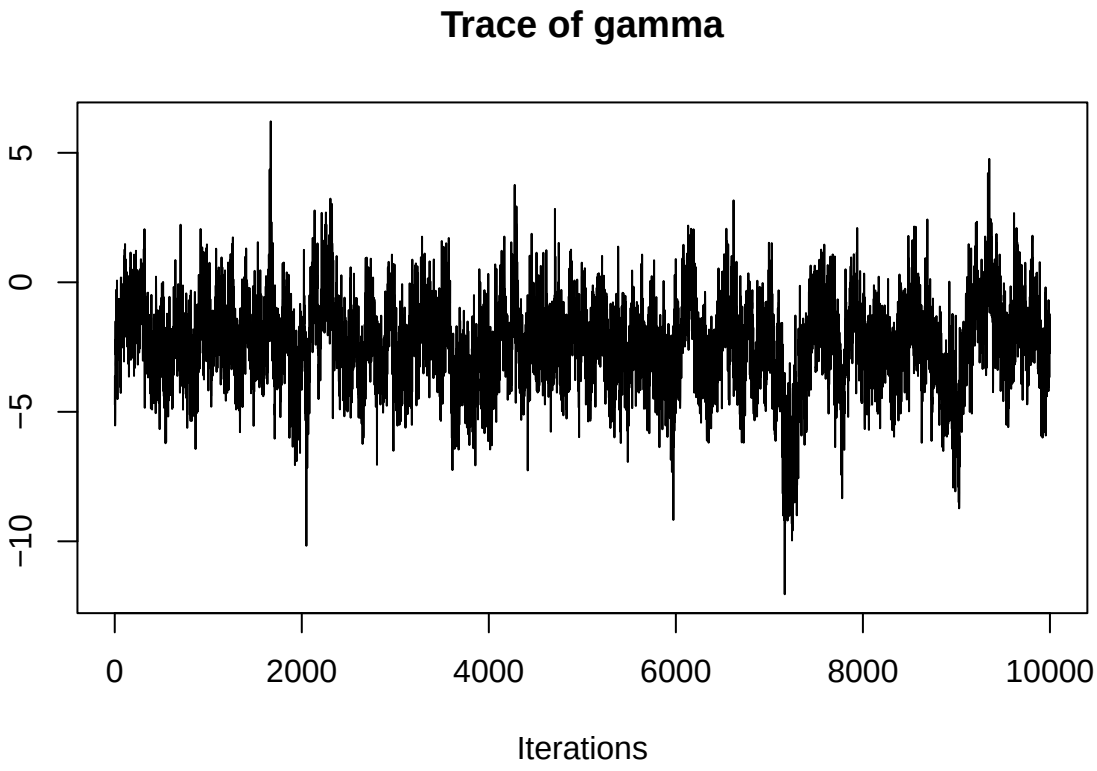

Trace of alpha[4]
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000
Iterations

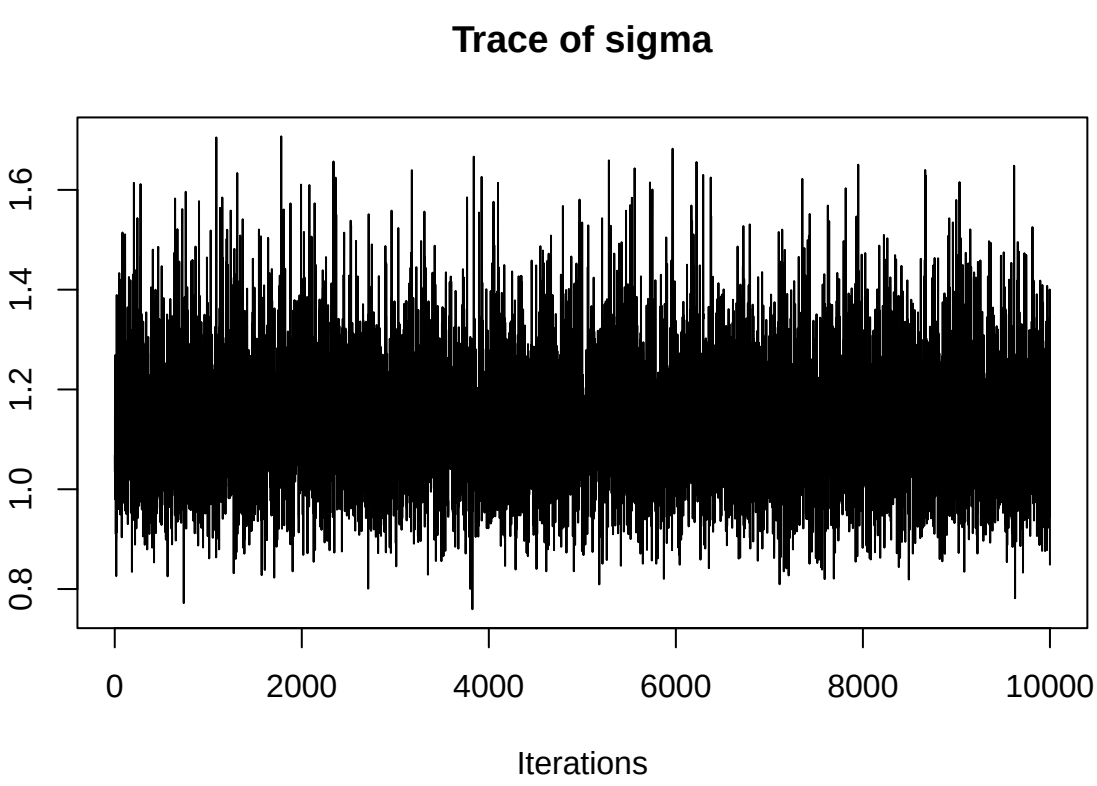

Trace of alpha[5]
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000
Iterations

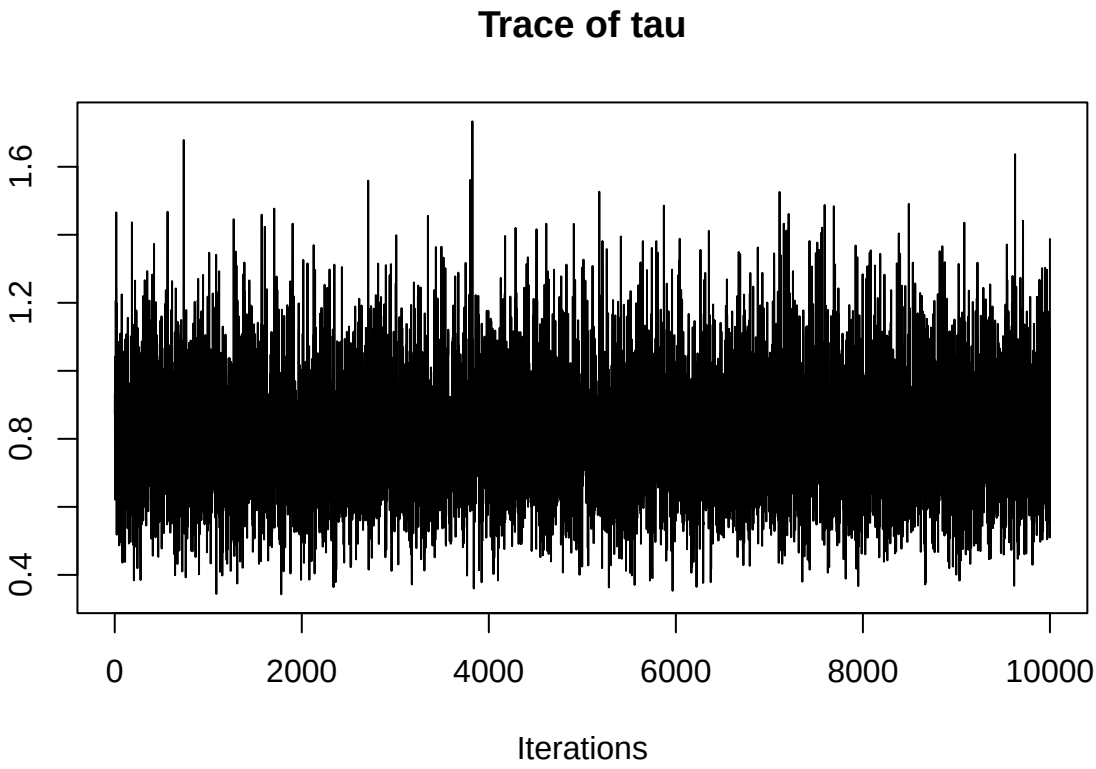

Trace of beta1
4
2
0
−2
−4
0
2000
4000
6000
8000
10000
Iterations

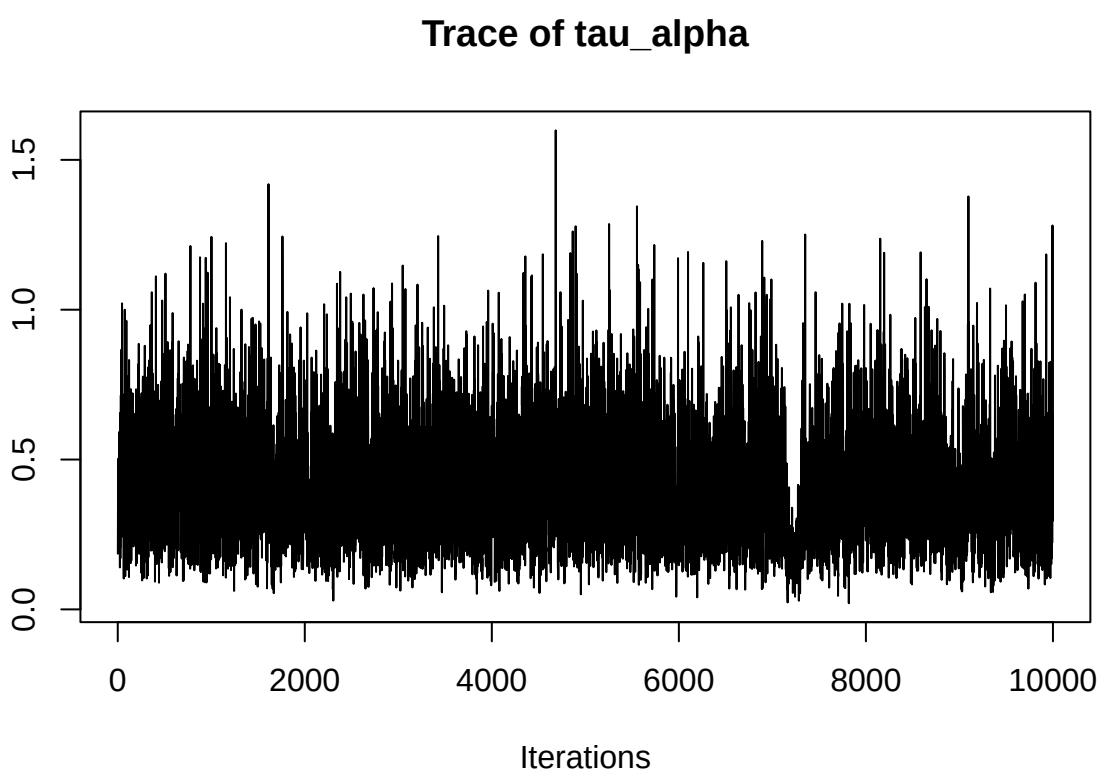

Trace of beta2
−0.1
−0.3
−0.5
−0.7
0
2000
4000
6000
8000
10000
Iterations


Trace of beta3
1.5
1.0
0.5
0.0
−1.0
0
2000
4000
6000
8000
10000
Iterations


Trace of gamma
5
0
−5
−10
0
2000
4000
6000
8000
10000
Iterations


Trace of sigma
0.8
1.0
1.2
1.4
1.6
0
2000
4000
6000
8000
10000
Iterations


Trace of tau
0.4
0.8
1.2
1.6
0
2000
4000
6000
8000
10000
Iterations



c

```
        Mean      SD  Naive SE Time-series SE
```

beta2 -0.3928 0.08698 0.0008698 0.003443 2.5% 25% 50% 75% 97.5% beta2 -0.5591 -0.45095 -0.3946 -0.3365 -0.2162

Yes. beta2 = -0.3960 and the confidence interval doesn't include 0

d

```
        Mean      SD  Naive SE Time-series SE
```

gamma -2.3542 1.56515 0.0090364 0.0630482 2.5% 25% 50% 75% 97.5% gamma -5.6342 -3.32079 -2.3110 -1.3267 0.7726

No. Because the confidence interval contains 0.

e

The measure of uncertainty I used in both part (c) and part (d) is the 95% posterior credible interval, based on the 2.5% and 97.5% quantiles of the posterior distributions which can be found in the summary.