stat431_final_project

Hantang Qin

2024-12-15

```
library(readr)
data = ObesityDataSet_raw_and_data_sinthetic <- read_csv("ObesityDataSet_raw_and_data_sinthet
ic.csv")
```

```
## Rows: 2111 Columns: 17
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (9): Gender, family_history_with_overweight, FAVC, CAEC, SMOKE, SCC, CAL...
## dbl (8): Age, Height, Weight, FCVC, NCP, CH2O, FAF, TUE
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# 加载必要的库
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
```

```r
# 1. 目标变量 (NObeyesdad): 编码为有序数值
data$NObeyesdad <- factor(data$NObeyesdad,
                          levels = c("Insufficient_Weight", "Normal_Weight",
                                     "Overweight_Level_I", "Overweight_Level_II",
                                     "Obesity_Type_I", "Obesity_Type_II", "Obesity_Type_II
I"),
                          ordered = TRUE)
data$NObeyesdad <- as.numeric(data$NObeyesdad)

# 2. 分组变量: 将 Age 分为年龄段 (Age Brackets)
data$AgeBracket <- cut(data$Age,
                       breaks = c(0, 18, 35, 50, 100),
                       labels = c("0-18", "19-35", "36-50", "51+"),
                       right = TRUE)

# 将 AgeBracket 转换为数值型
data$AgeBracket <- as.numeric(data$AgeBracket)

# 3. 分类和二元变量: 转换为数值编码
data$Gender <- ifelse(data$Gender == "Male", 1, 0)  # Male = 1, Female = 0
data$family_history_with_overweight <- ifelse(data$family_history_with_overweight == "yes",
1, 0)
data$FAVC <- ifelse(data$FAVC == "yes", 1, 0)
data$SMOKE <- ifelse(data$SMOKE == "yes", 1, 0)
data$SCC <- ifelse(data$SCC == "yes", 1, 0)

# 处理有序分类变量（如 CAEC, CALC）
data$CAEC <- as.numeric(factor(data$CAEC, levels = c("no", "Sometimes", "Frequently", "Alway
s")))
data$CALC <- as.numeric(factor(data$CALC, levels = c("no", "Sometimes", "Frequently", "Alway
s")))

# 4. 标准化连续变量: Age, FAF, CH2O, NCP
data <- data %>%
  mutate(
    Age = scale(Age),
    FAF = scale(FAF),
    CH2O = scale(CH2O),
    NCP = scale(NCP)
  )

# 5. 检查最终的数据结构
str(data)
```

```
## tibble [2,111 × 18] (S3: tbl_df/tbl/data.frame)
##  $ Gender                        : num [1:2111] 0 0 1 1 1 1 0 1 1 1 ...
##  $ Age                           : num [1:2111, 1] -0.522 -0.522 -0.207 0.423 -0.364 ...
##   ..- attr(*, "scaled:center")= num 24.3
##   ..- attr(*, "scaled:scale")= num 6.35
##  $ Height                        : num [1:2111] 1.62 1.52 1.8 1.8 1.78 1.62 1.5 1.64 1.78
1.72 ...
##  $ Weight                        : num [1:2111] 64 56 77 87 89.8 53 55 53 64 68 ...
##  $ family_history_with_overweight: num [1:2111] 1 1 1 0 0 0 1 0 1 1 ...
##  $ FAVC                          : num [1:2111] 0 0 0 0 0 1 1 0 1 1 ...
##  $ FCVC                          : num [1:2111] 2 3 2 3 2 2 3 2 3 2 ...
##  $ NCP                           : num [1:2111, 1] 0.404 0.404 0.404 0.404 -2.167 ...
##   ..- attr(*, "scaled:center")= num 2.69
##   ..- attr(*, "scaled:scale")= num 0.778
##  $ CAEC                          : num [1:2111] 2 2 2 2 2 2 2 2 2 2 ...
##  $ SMOKE                         : num [1:2111] 0 1 0 0 0 0 0 0 0 0 ...
##  $ CH2O                          : num [1:2111, 1] -0.0131 1.6184 -0.0131 -0.0131 -0.0131
...
##   ..- attr(*, "scaled:center")= num 2.01
##   ..- attr(*, "scaled:scale")= num 0.613
##  $ SCC                           : num [1:2111] 0 1 0 0 0 0 0 0 0 0 ...
##  $ FAF                           : num [1:2111, 1] -1.19 2.34 1.16 1.16 -1.19 ...
##   ..- attr(*, "scaled:center")= num 1.01
##   ..- attr(*, "scaled:scale")= num 0.851
##  $ TUE                           : num [1:2111] 1 0 1 0 0 0 0 0 1 1 ...
##  $ CALC                          : num [1:2111] 1 2 3 3 2 2 2 2 3 1 ...
##  $ MTRANS                        : chr [1:2111] "Public_Transportation" "Public_Transporta
tion" "Public_Transportation" "Walking" ...
##  $ NObeyesdad                    : num [1:2111] 2 2 2 3 4 2 2 2 2 2 ...
##  $ AgeBracket                    : num [1:2111] 2 2 2 2 2 2 2 2 2 2 ...
```

```
# 6. 准备数据列表 (用于 rjags)
data_jags <- list(
  N = nrow(data),                       # 样本量
  Y = data$NObeyesdad,                  # 目标变量 (有序数值)
  AgeBracket = data$AgeBracket,         # 分组变量 (年龄段)
  FAVC = data$FAVC,                     # 高热量食物
  FAF = as.numeric(data$FAF),           # 身体活动 (标准化)
  CALC = data$CALC,                     # 饮酒频率
  family_history = data$family_history_with_overweight,  # 家族肥胖病史
  Gender = data$Gender                  # 性别
)

# 检查准备好的数据
summary(data_jags)
```

```
##                Length Class  Mode
## N                 1   -none- numeric
## Y              2111   -none- numeric
## AgeBracket     2111   -none- numeric
## FAVC           2111   -none- numeric
## FAF            2111   -none- numeric
## CALC           2111   -none- numeric
## family_history 2111   -none- numeric
## Gender         2111   -none- numeric
```

```
# 加载必要的库
library(rjags)
```

```
## Warning: package 'rjags' was built under R version 4.4.1
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.2
```

```
## Loaded modules: basemod,bugs
```

```
library(coda)

# 数据准备
K <- 7  # 目标变量类别数
J <- length(unique(data$AgeBracket))  # 年龄段的数量

data_jags <- list(
  N = nrow(data),                     # 样本量
  Y = data$NObeyesdad,                # 目标变量（有序编码，1-7）
  AgeBracket = data$AgeBracket,       # 分组变量（年龄段）
  FAVC = data$FAVC,                   # 高热量食物
  FAF = as.numeric(data$FAF),         # 身体活动
  CALC = data$CALC,                   # 饮酒频率
  family_history = data$family_history_with_overweight, # 家族病史
  Gender = data$Gender,               # 性别
  K = K,                              # 目标变量类别数
  J = J                               # 年龄段数
)

# 初始化值函数
inits <- function() {
  list(beta = rnorm(5, 0, 0.1),
       delta = 0,
       gamma = rnorm(J, 0, 1),
       sigma = 1,
       sigma_gamma = 1,
       tau = sort(runif(K-1, -2, 2)))  # K-1 = 6 个阈值
}
```

```
model_string <- "
model {
  # 1. Likelihood: Ordered Probit Model
  for (i in 1:N) {
    Z[i] ~ dnorm(mu[i], tau_z)   # 潜在连续变量

    # Ordered Probit 概率
    p[i, 1] <- phi(tau[1] - mu[i])
    for (k in 2:(K-1)) {
      p[i, k] <- phi(tau[k] - mu[i]) - phi(tau[k-1] - mu[i])
    }
    p[i, K] <- 1 - phi(tau[K-1] - mu[i])  # 最后一类别概率

    Y[i] ~ dcat(p[i, 1:K])  # 有序分类响应

    # 线性预测器
    mu[i] <- beta[1] + beta[2] * FAVC[i] + beta[3] * FAF[i] +
             beta[4] * CALC[i] + beta[5] * family_history[i] +
             gamma[AgeBracket[i]] + delta * Gender[i]
  }

  # 2. 随机效应：年龄段
  for (j in 1:J) {
    gamma[j] ~ dnorm(0, tau_gamma)
  }

  # 3. 固定效应的先验
  for (b in 1:5) {
    beta[b] ~ dnorm(0, 0.001)
  }
  delta ~ dnorm(0, 0.001)

  # 4. 方差的先验
  tau_gamma <- 1 / sigma_gamma^2
  sigma_gamma ~ dunif(0, 10)
  tau_z <- 1 / sigma^2
  sigma ~ dunif(0, 10)

  # 5. 阈值参数（确保递增）
  tau[1] ~ dnorm(0, 0.001)T(-10, )  # 第一个阈值有下界
  for (k in 2:(K-1)) {
    tau[k] ~ dnorm(0, 0.001)T(tau[k-1], )  # 单调递增约束
  }
}
"
```

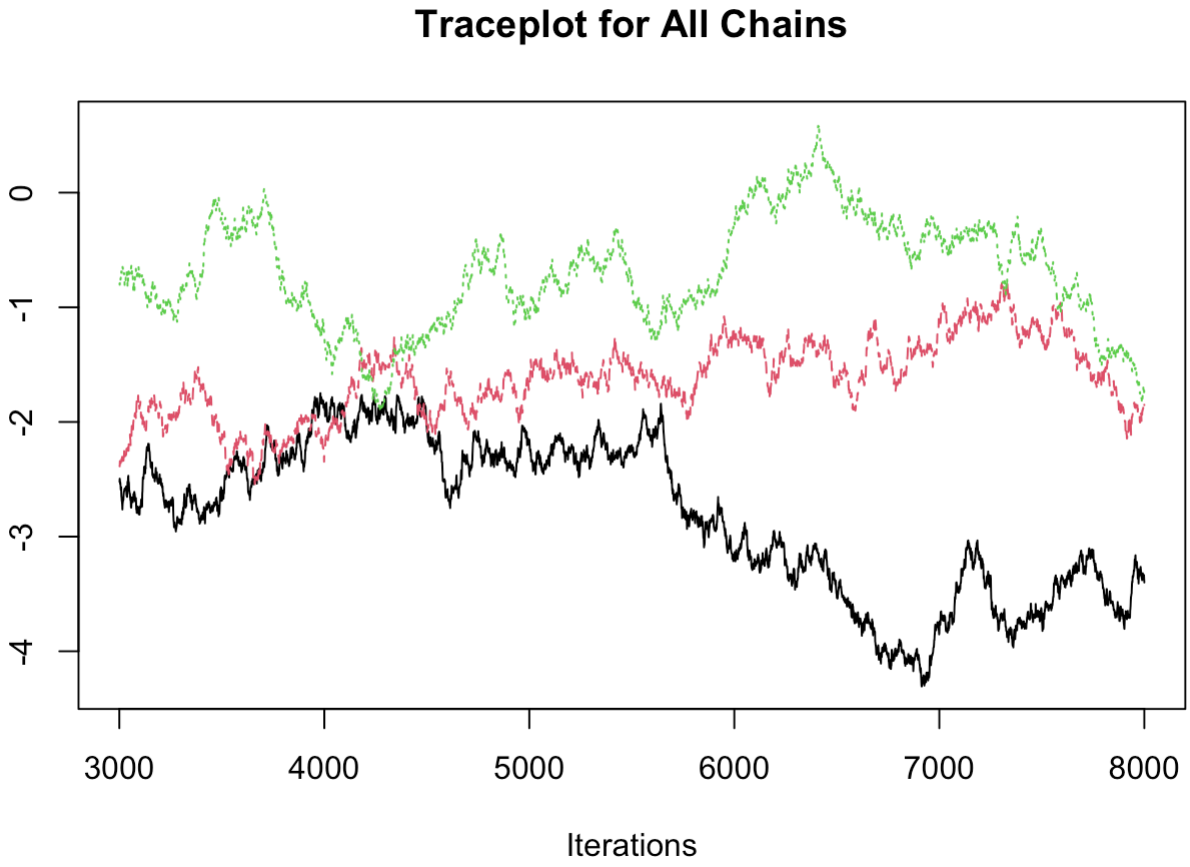
```r
# 参数列表
parameters <- c("beta", "delta", "sigma", "sigma_gamma", "tau", "gamma")

# 运行 JAGS 模型
jags_model <- jags.model(textConnection(model_string),
                         data = data_jags,
                         inits = inits,
                         n.chains = 3,
                         n.adapt = 1000)
```

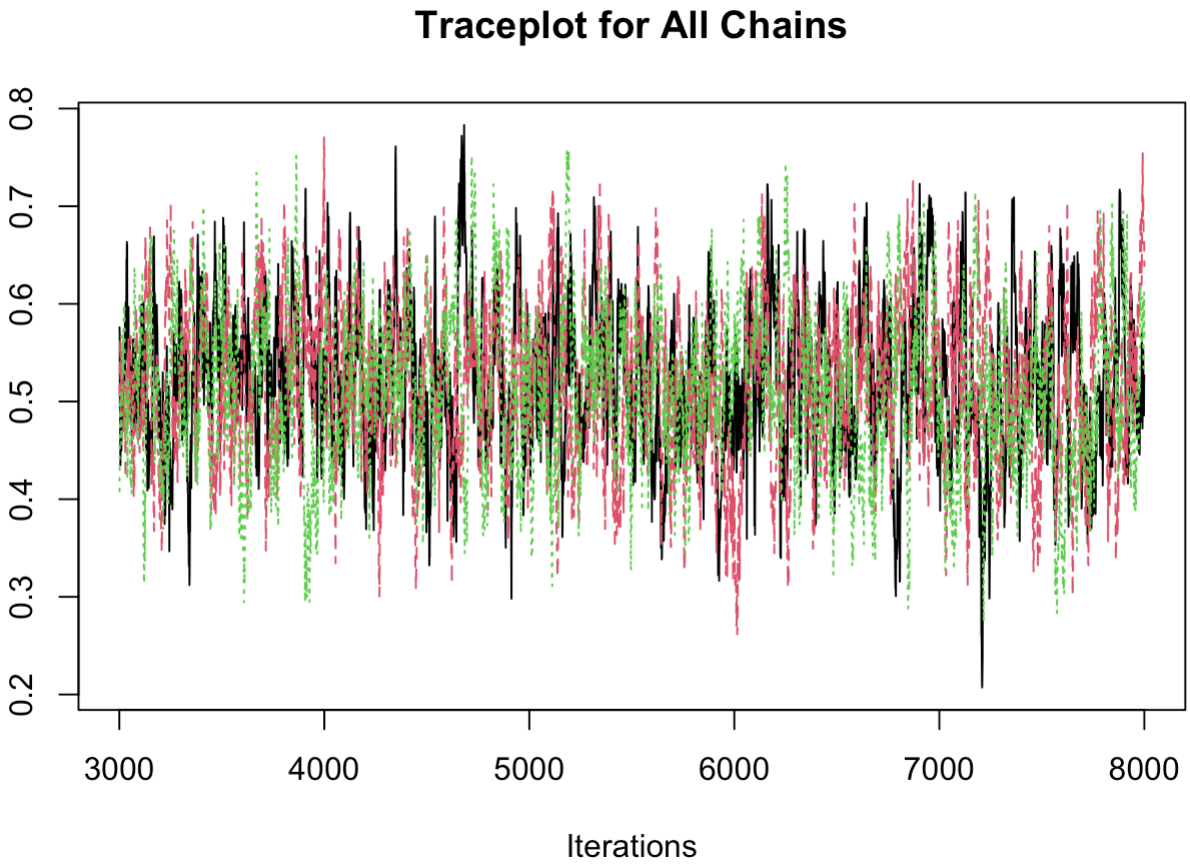
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2111
##    Unobserved stochastic nodes: 2129
##    Total graph size: 45975
##
## Initializing model
```

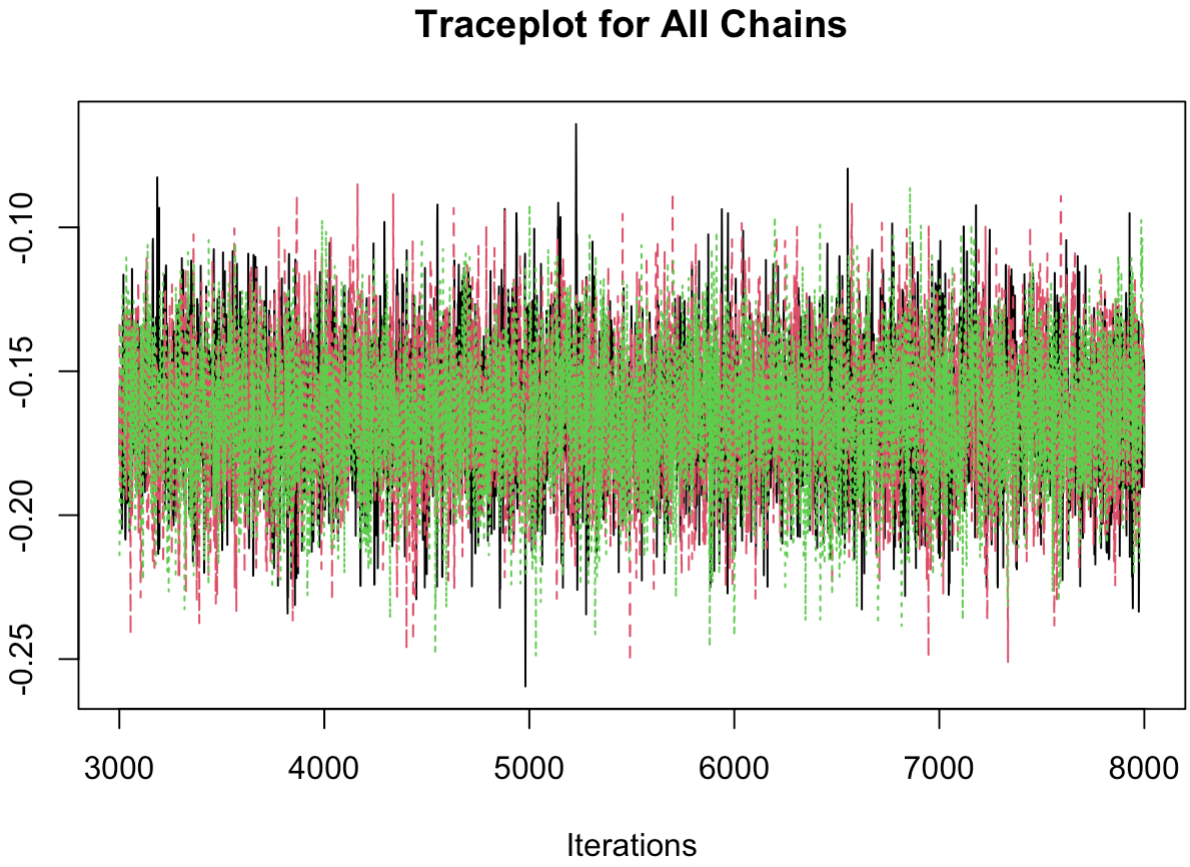
```r
# Burn-in 迭代
update(jags_model, 2000)

# 抽取样本
samples <- coda.samples(jags_model,
                        variable.names = parameters,
                        n.iter = 5000)

# 查看结果
summary(samples)
```

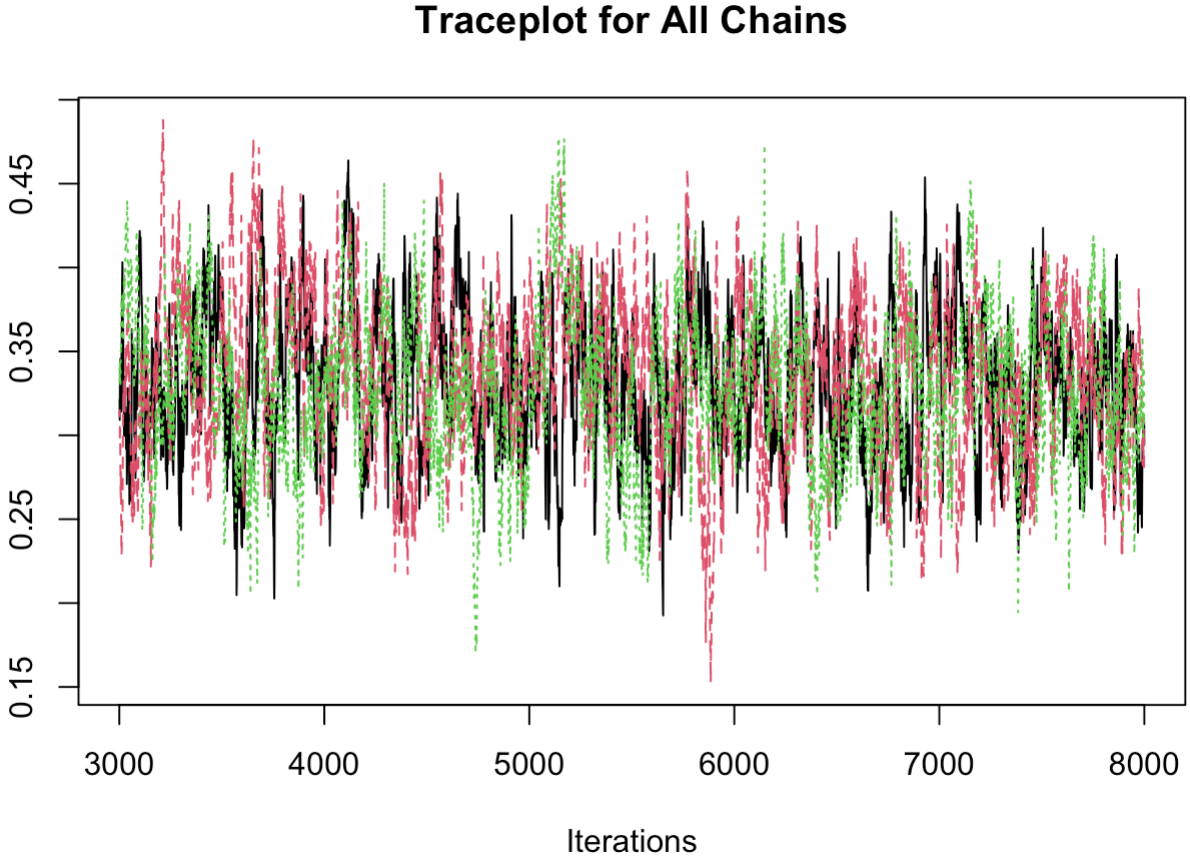
```
##
## Iterations = 3001:8000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean      SD  Naive SE Time-series SE
## beta[1]     -1.71387 1.00461 0.0082026      0.1568426
## beta[2]      0.51590 0.07708 0.0006293      0.0031187
## beta[3]     -0.16471 0.02392 0.0001953      0.0002734
## beta[4]      0.33035 0.04457 0.0003639      0.0020624
## beta[5]      1.49164 0.06753 0.0005514      0.0024149
## delta       -0.20519 0.04685 0.0003825      0.0008719
## gamma[1]    -0.80573 0.52760 0.0043078      0.1029949
## gamma[2]     0.01657 0.52064 0.0042510      0.1040575
## gamma[3]    -0.10105 0.52323 0.0042721      0.0972358
## gamma[4]    -0.01427 0.53841 0.0043961      0.0610979
## sigma        4.96266 2.89119 0.0236065      0.0236074
## sigma_gamma  0.90108 0.73877 0.0060320      0.0373224
## tau[1]      -1.12681 0.67751 0.0055318      0.1709526
## tau[2]      -0.45508 0.67709 0.0055284      0.1727328
## tau[3]       0.05994 0.67736 0.0055306      0.1760966
## tau[4]       0.50402 0.67816 0.0055372      0.1727037
## tau[5]       1.05175 0.67835 0.0055387      0.1764376
## tau[6]       1.64469 0.67864 0.0055411      0.1583978
##
## 2. Quantiles for each variable:
##
##                 2.5%     25%      50%      75%    97.5%
## beta[1]     -3.80564 -2.2913 -1.62643 -0.98713 -0.01620
## beta[2]      0.36449  0.4638  0.51450  0.56786  0.66906
## beta[3]     -0.21173 -0.1808 -0.16453 -0.14854 -0.11795
## beta[4]      0.24486  0.3001  0.32949  0.36125  0.41719
## beta[5]      1.35730  1.4463  1.49188  1.53614  1.62557
## delta       -0.29617 -0.2374 -0.20470 -0.17417 -0.11275
## gamma[1]    -2.08020 -1.1231 -0.66824 -0.44679  0.02582
## gamma[2]    -1.25946 -0.3059  0.15994  0.36248  0.86051
## gamma[3]    -1.37008 -0.4154  0.03169  0.24624  0.74753
## gamma[4]    -1.26373 -0.2915  0.05048  0.33453  0.92924
## sigma        0.25421  2.4603  4.92977  7.48693  9.72563
## sigma_gamma  0.24622  0.4428  0.66414  1.09631  2.90920
## tau[1]      -2.69780 -1.3508 -0.97859 -0.65198 -0.22831
## tau[2]      -2.02802 -0.6779 -0.30678  0.01699  0.44146
## tau[3]      -1.51613 -0.1623  0.21050  0.53539  0.95859
## tau[4]      -1.07525  0.2762  0.65618  0.98152  1.40509
## tau[5]      -0.52495  0.8237  1.20271  1.52844  1.95391
## tau[6]       0.07147  1.4149  1.79874  2.12269  2.54351
```

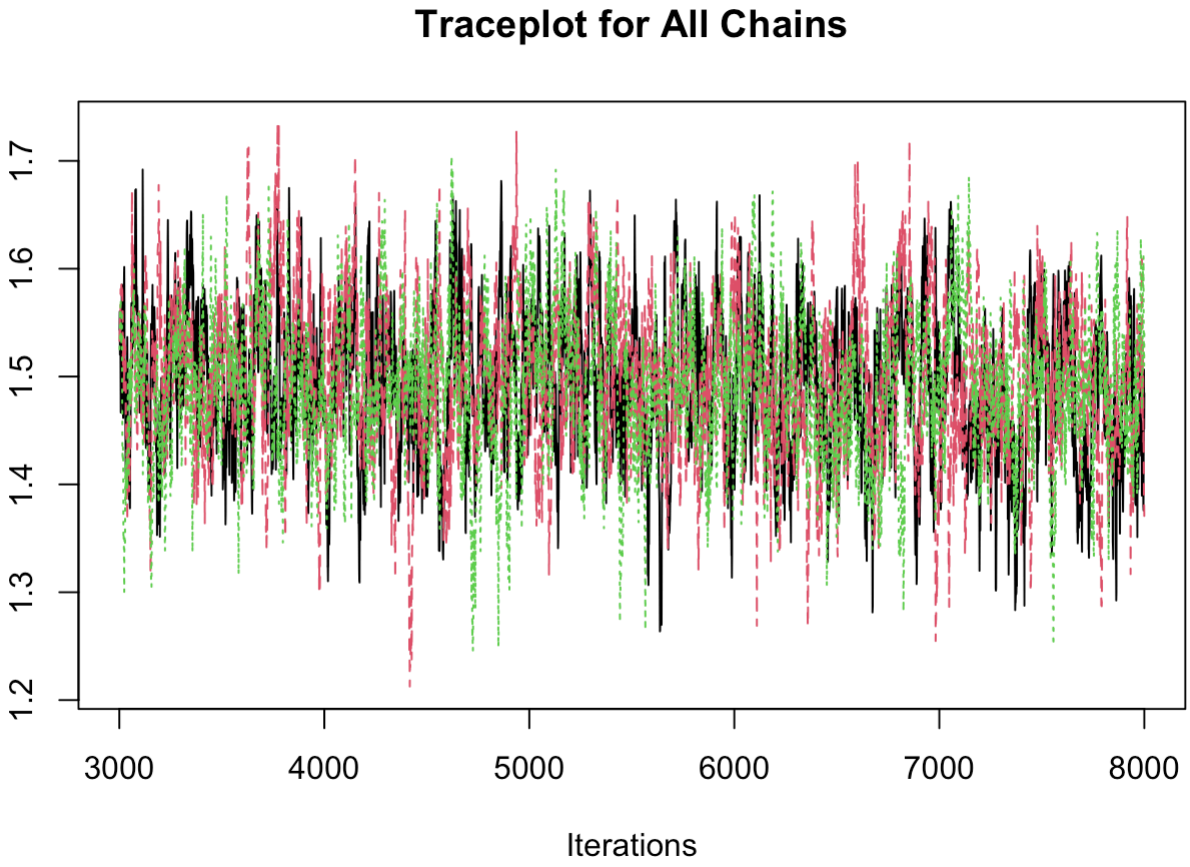
```r
# 加载必要的库
library(rjags)
library(coda)

# 假设 samples 是之前抽取的 MCMC 结果
# samples <- coda.samples(jags_model, variable.names = c("beta", "delta", "tau"), n.iter = 5000)

### Step 1: 绘制收敛性诊断图
# Traceplot: 绘制所有链的轨迹图
traceplot(samples, main = "Traceplot for All Chains")
```

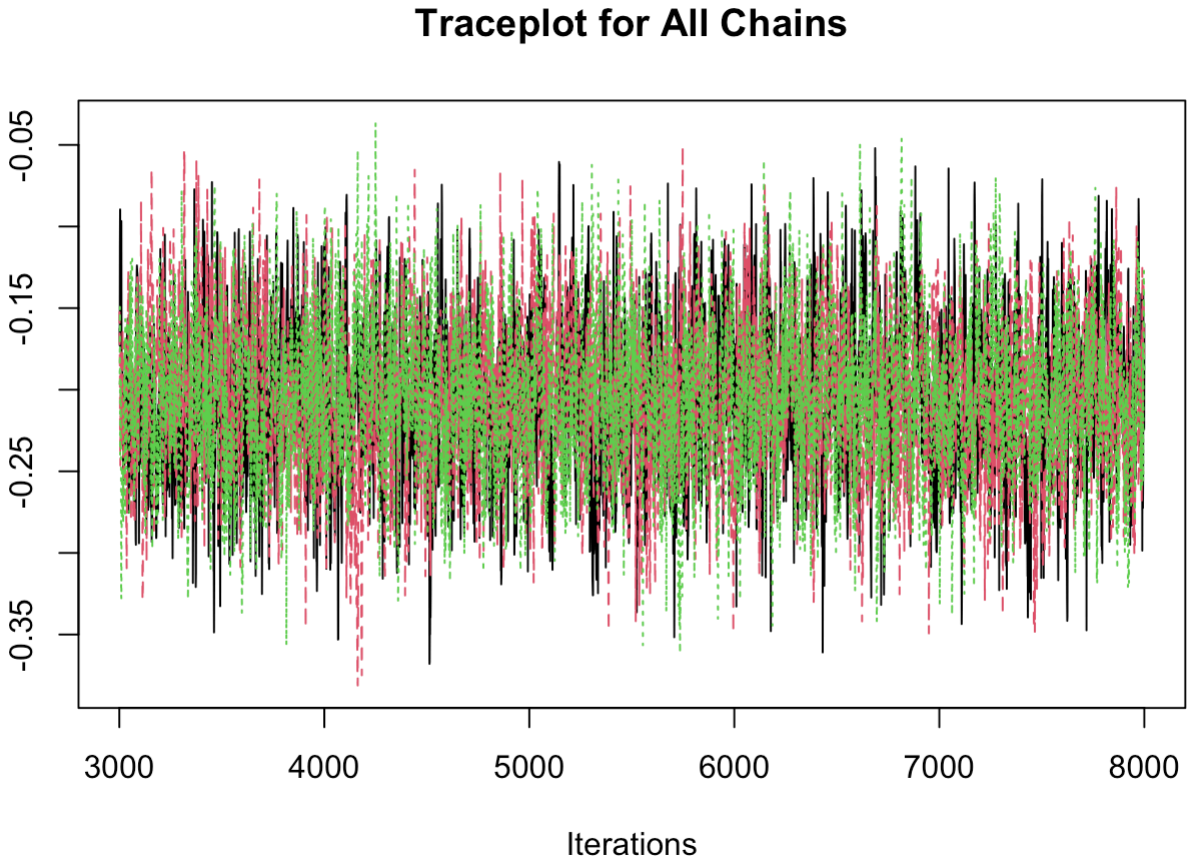

Traceplot for All Chains
0
-1
-2
-3
-4
3000
4000
5000
6000
7000
8000
Iterations

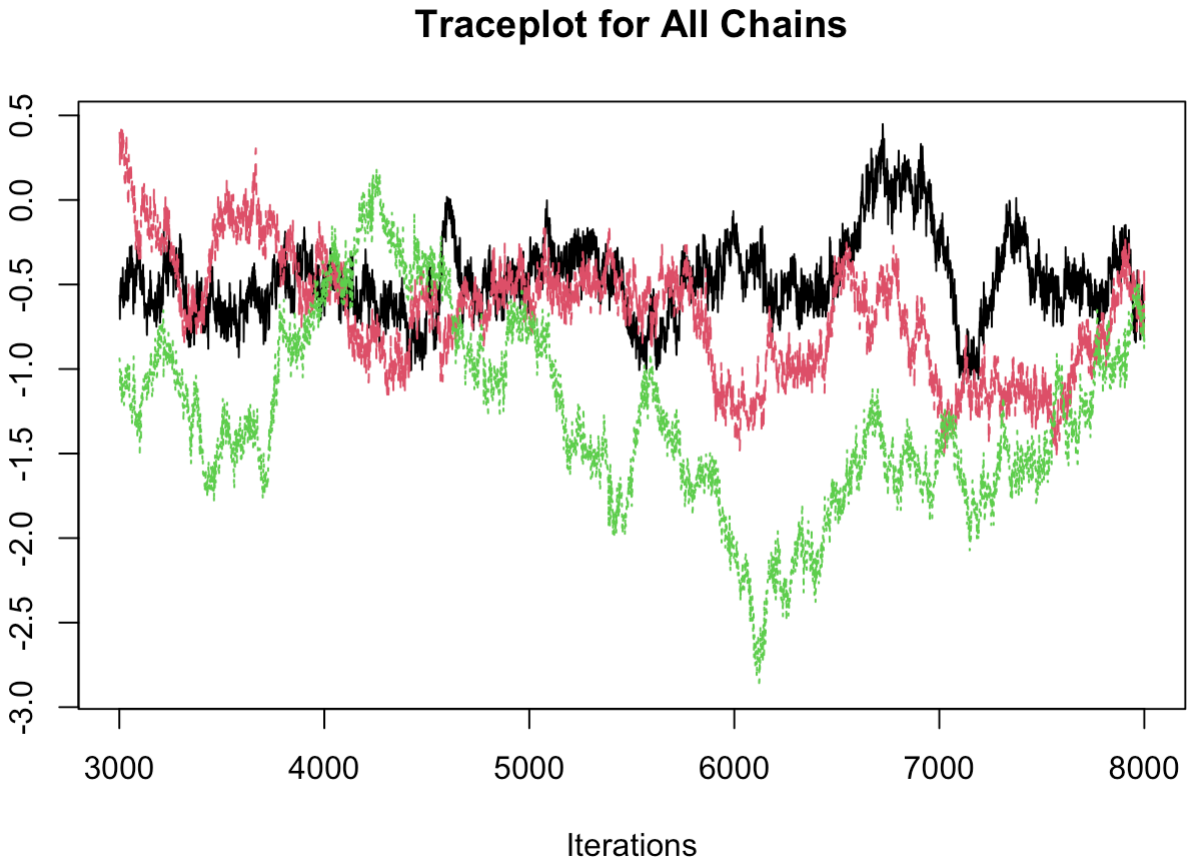

Traceplot for All Chains
0.8
0.7
0.6
0.5
0.4
0.3
0.2
3000
4000
5000
6000
7000
8000
Iterations

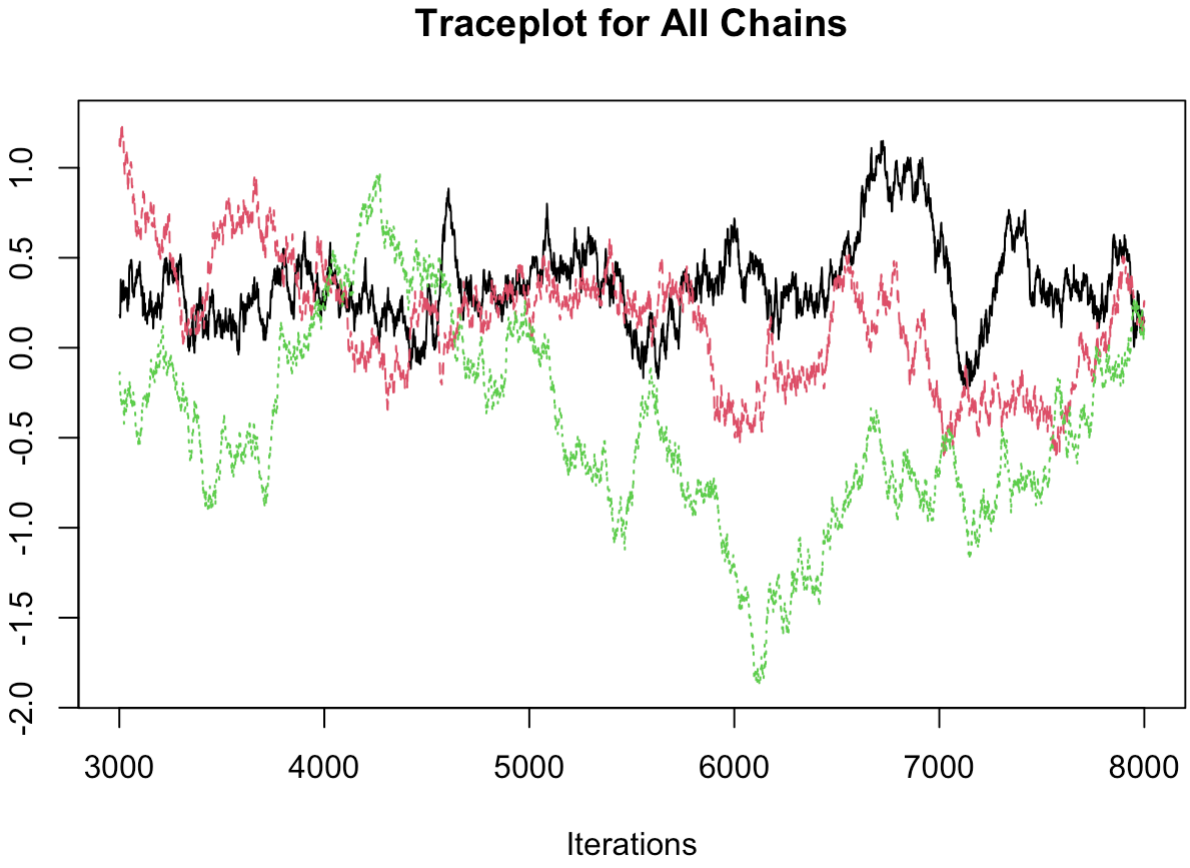

Traceplot for All Chains
-0.10
-0.15
-0.20
-0.25
3000
4000
5000
6000
7000
8000
Iterations

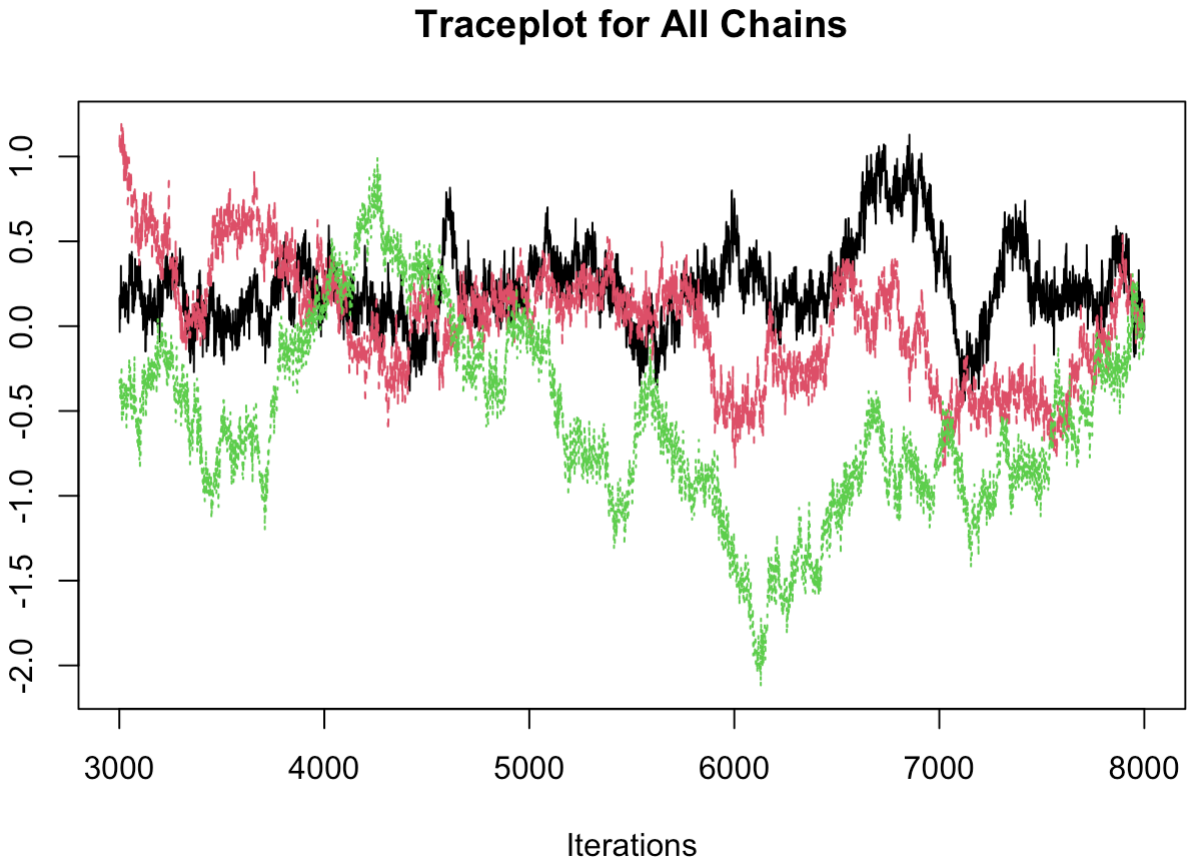

Traceplot for All Chains
0.45
0.35
0.25
0.15
3000
4000
5000
6000
7000
8000
Iterations

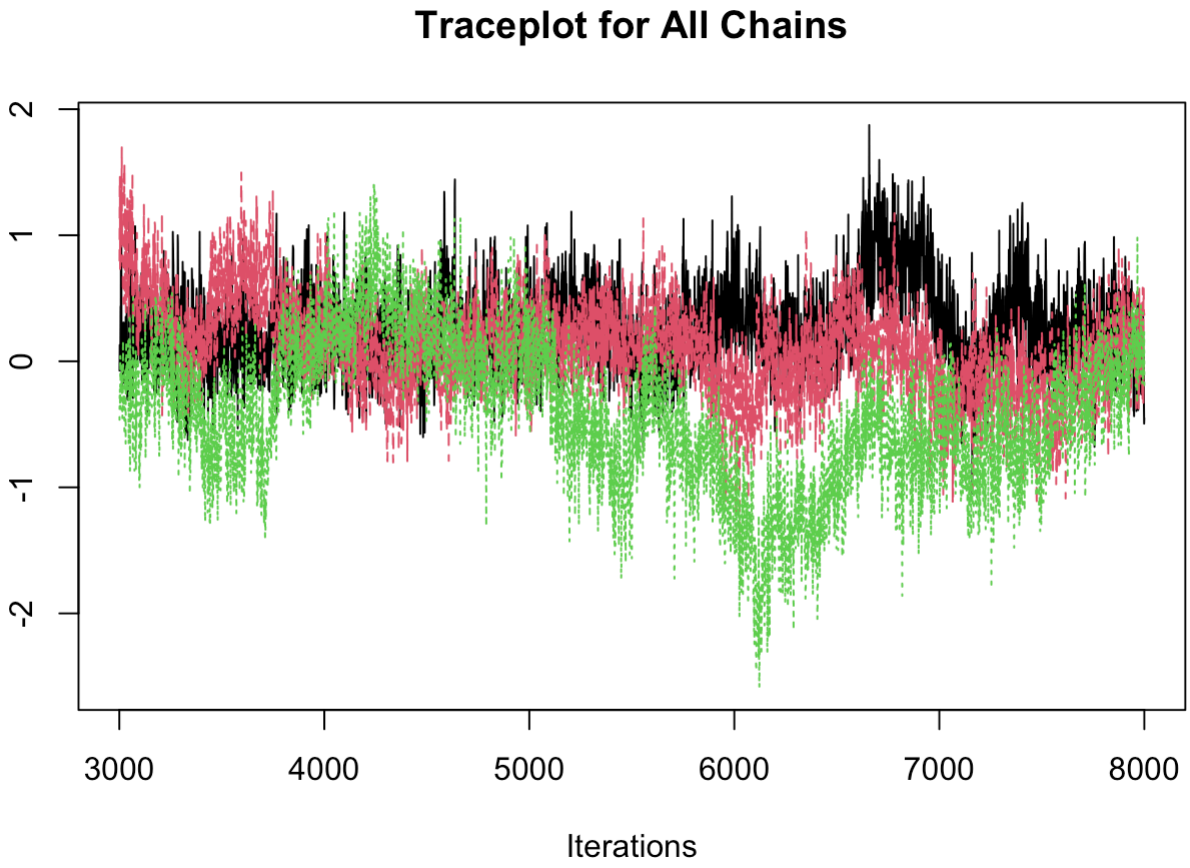

Traceplot for All Chains
1.2
1.3
1.4
1.5
1.6
1.7
3000
4000
5000
6000
7000
8000
Iterations

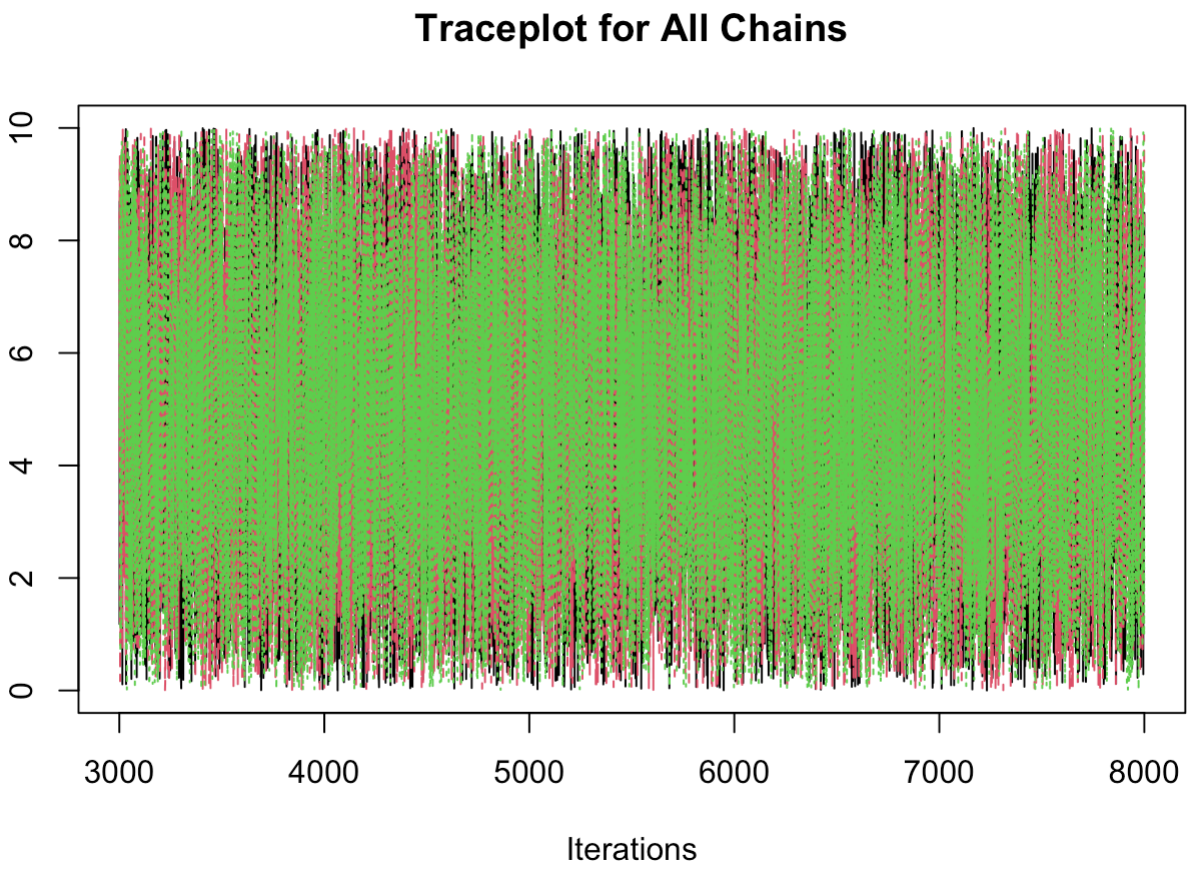

Traceplot for All Chains
-0.35
-0.25
-0.15
-0.05
3000
4000
5000
6000
7000
8000
Iterations

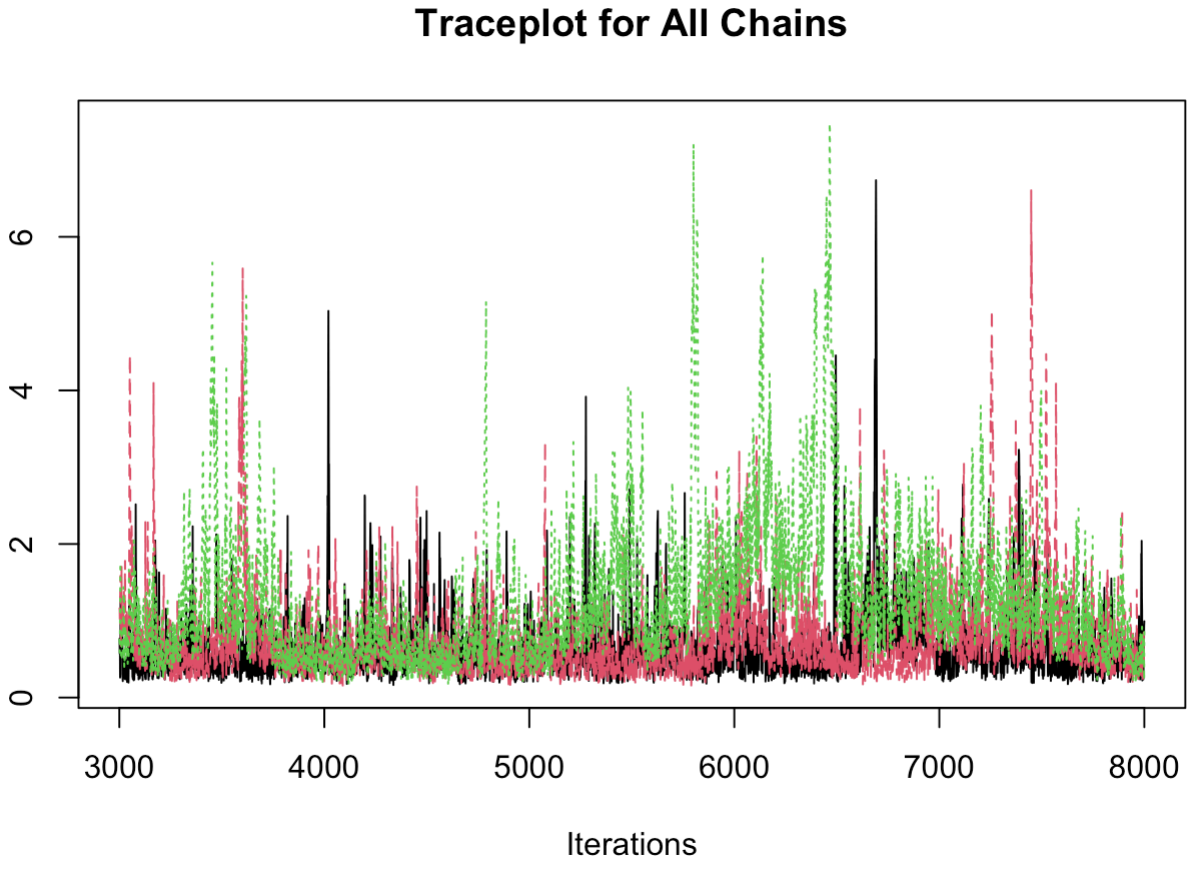

Traceplot for All Chains
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
3000
4000
5000
6000
7000
8000
Iterations

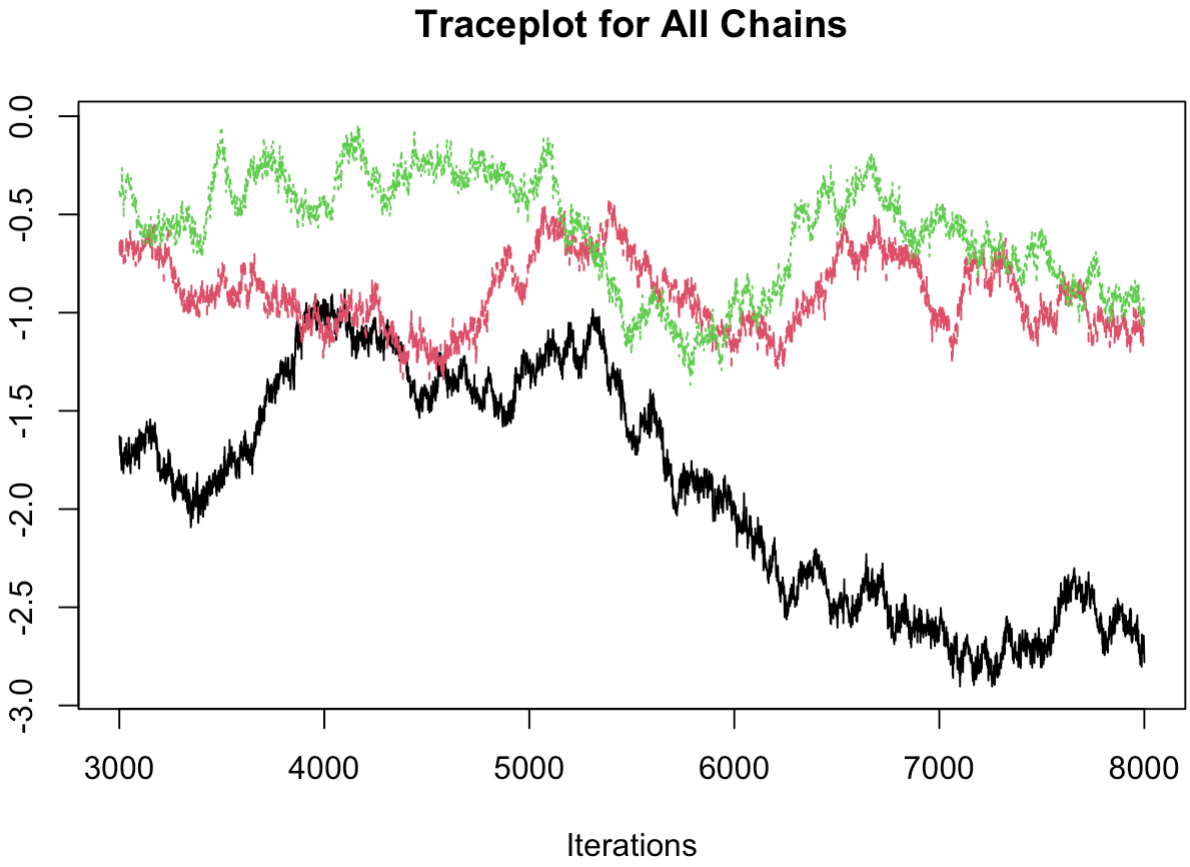

Traceplot for All Chains
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
3000
4000
5000
6000
7000
8000
Iterations

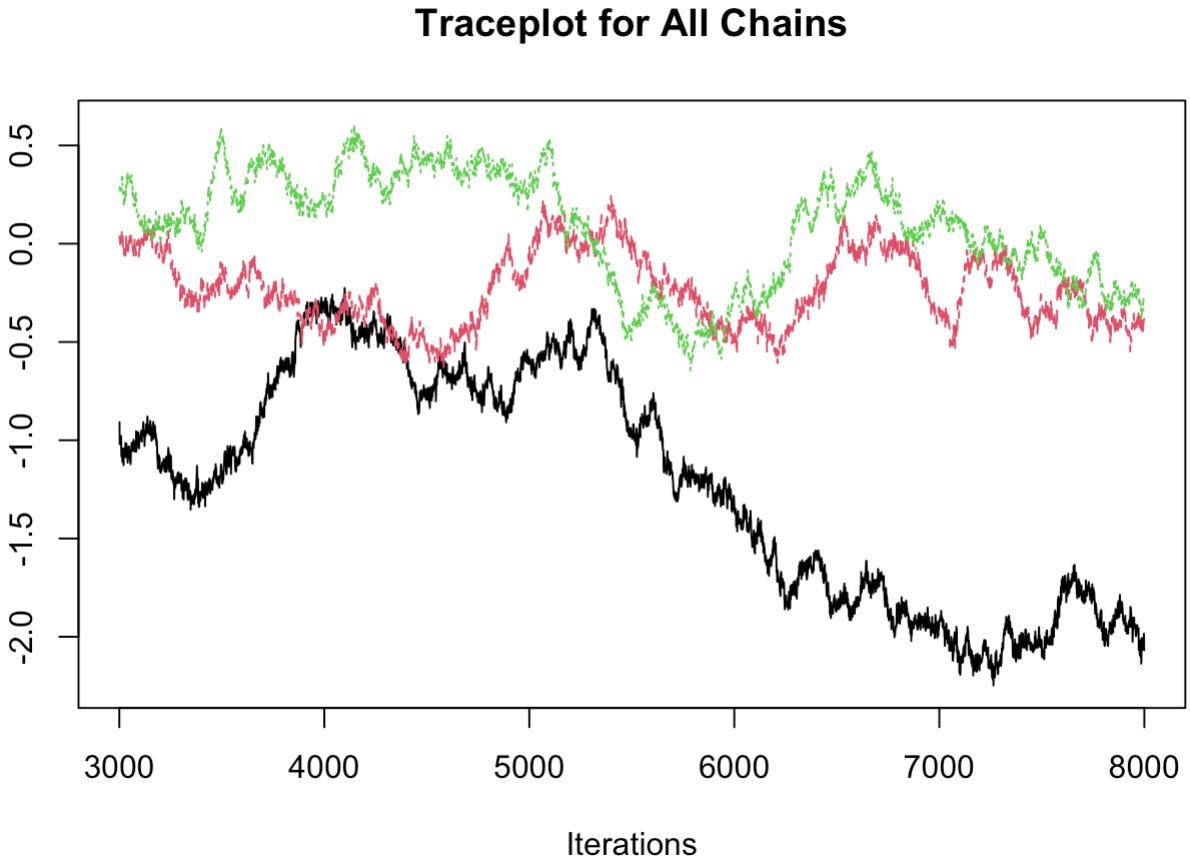

Traceplot for All Chains
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
3000
4000
5000
6000
7000
8000
Iterations

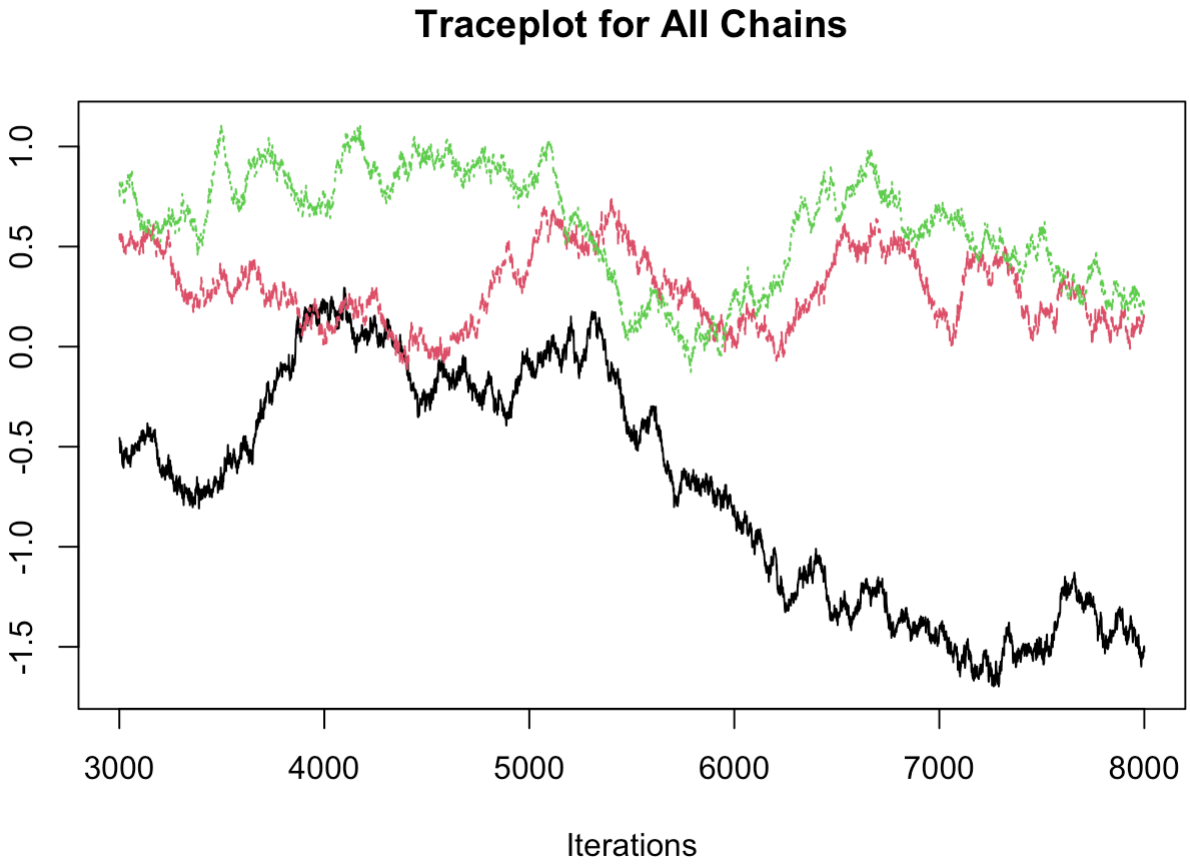

Traceplot for All Chains
2
1
0
-1
-2
3000
4000
5000
6000
7000
8000
Iterations

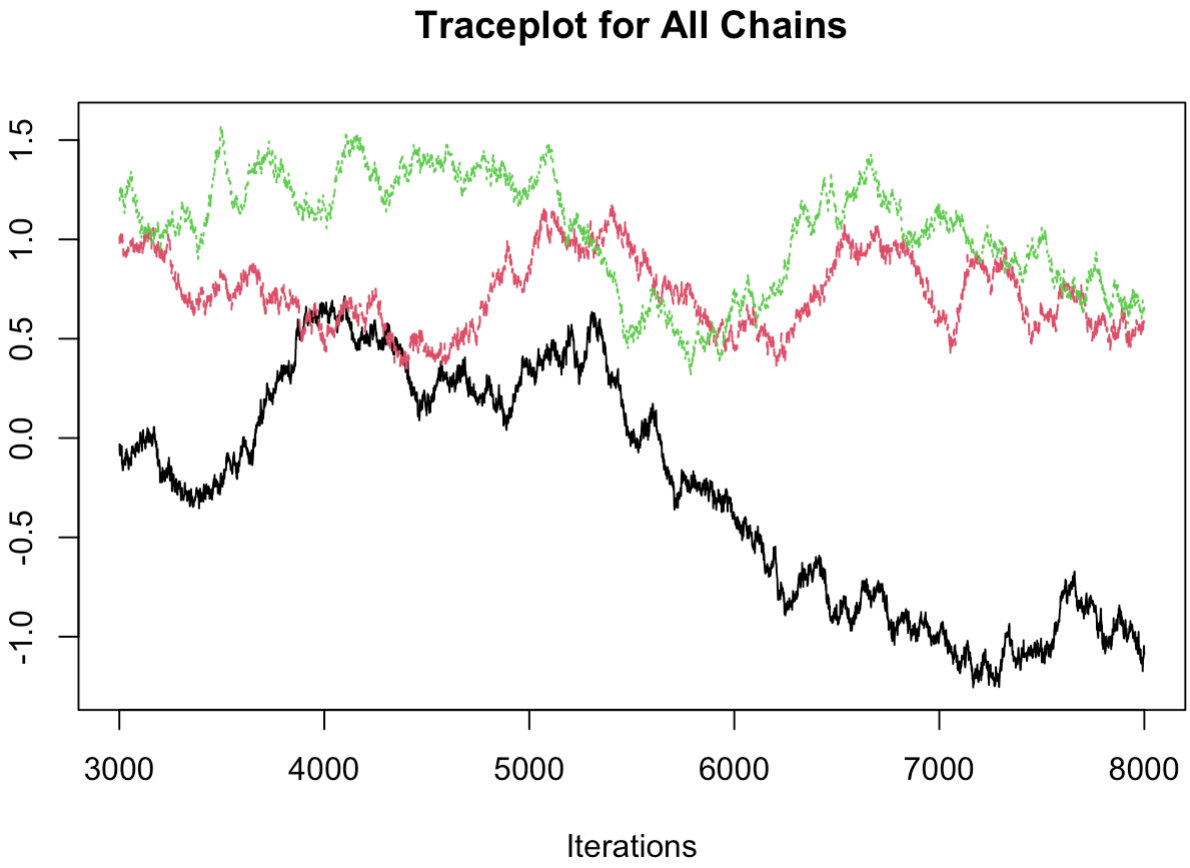

Traceplot for All Chains
0
2
4
6
8
10
3000
4000
5000
6000
7000
8000
Iterations

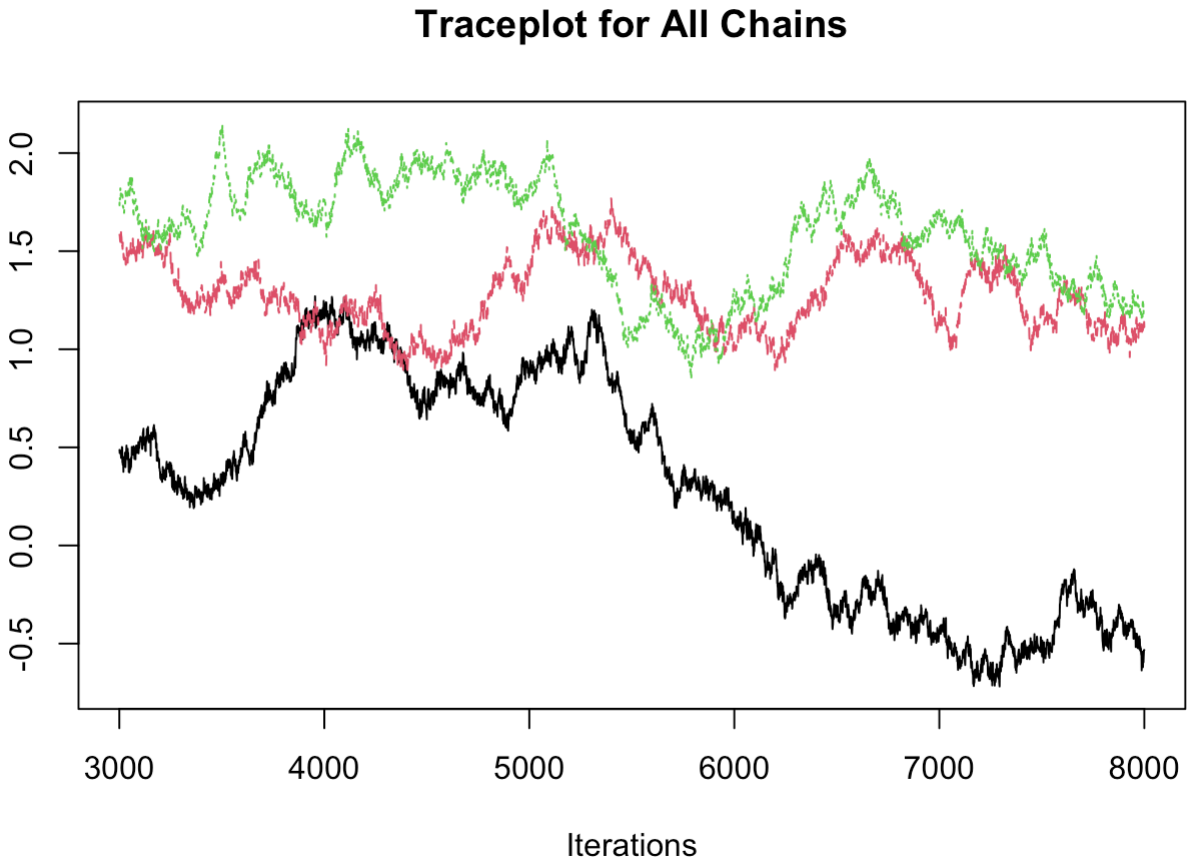

Traceplot for All Chains
0
2
4
6
3000
4000
5000
6000
7000
8000
Iterations

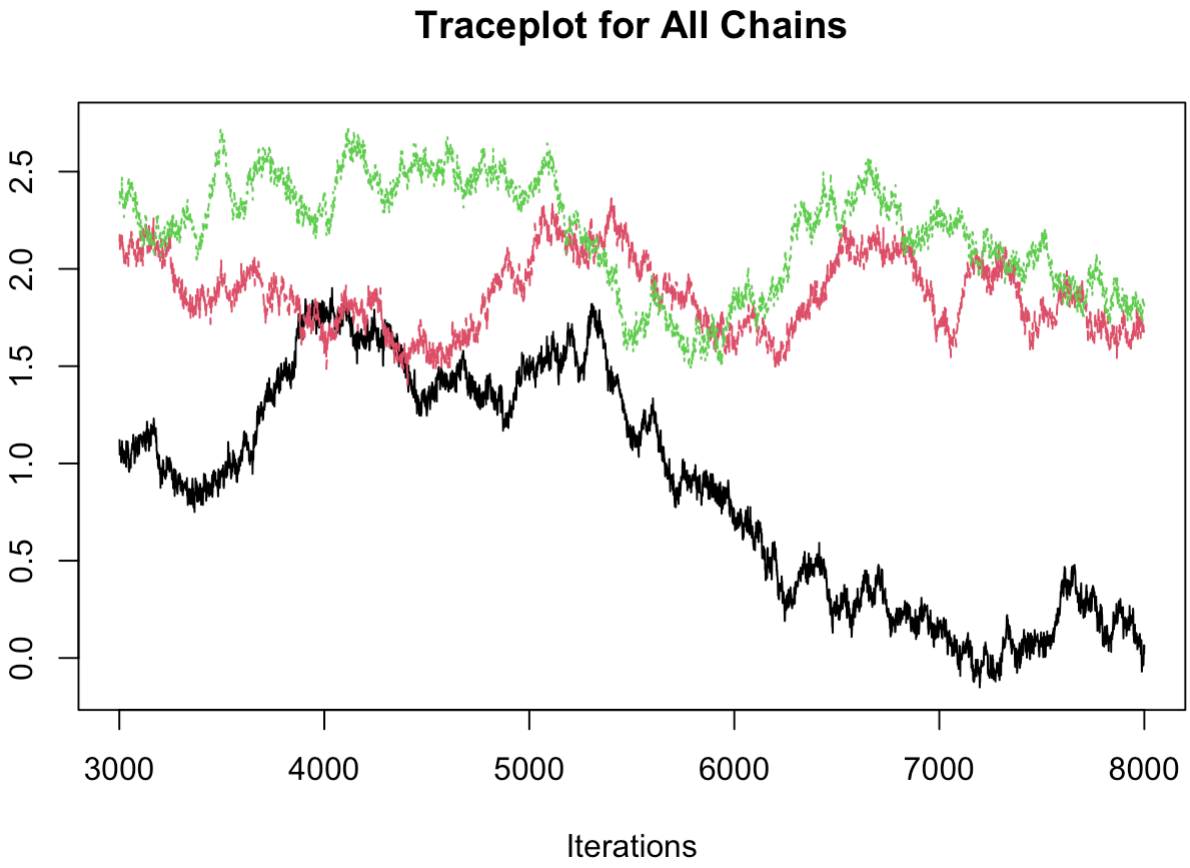

Traceplot for All Chains
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
3000
4000
5000
6000
7000
8000
Iterations


Traceplot for All Chains
0.5
0.0
-0.5
-1.0
-1.5
-2.0
3000
4000
5000
6000
7000
8000
Iterations

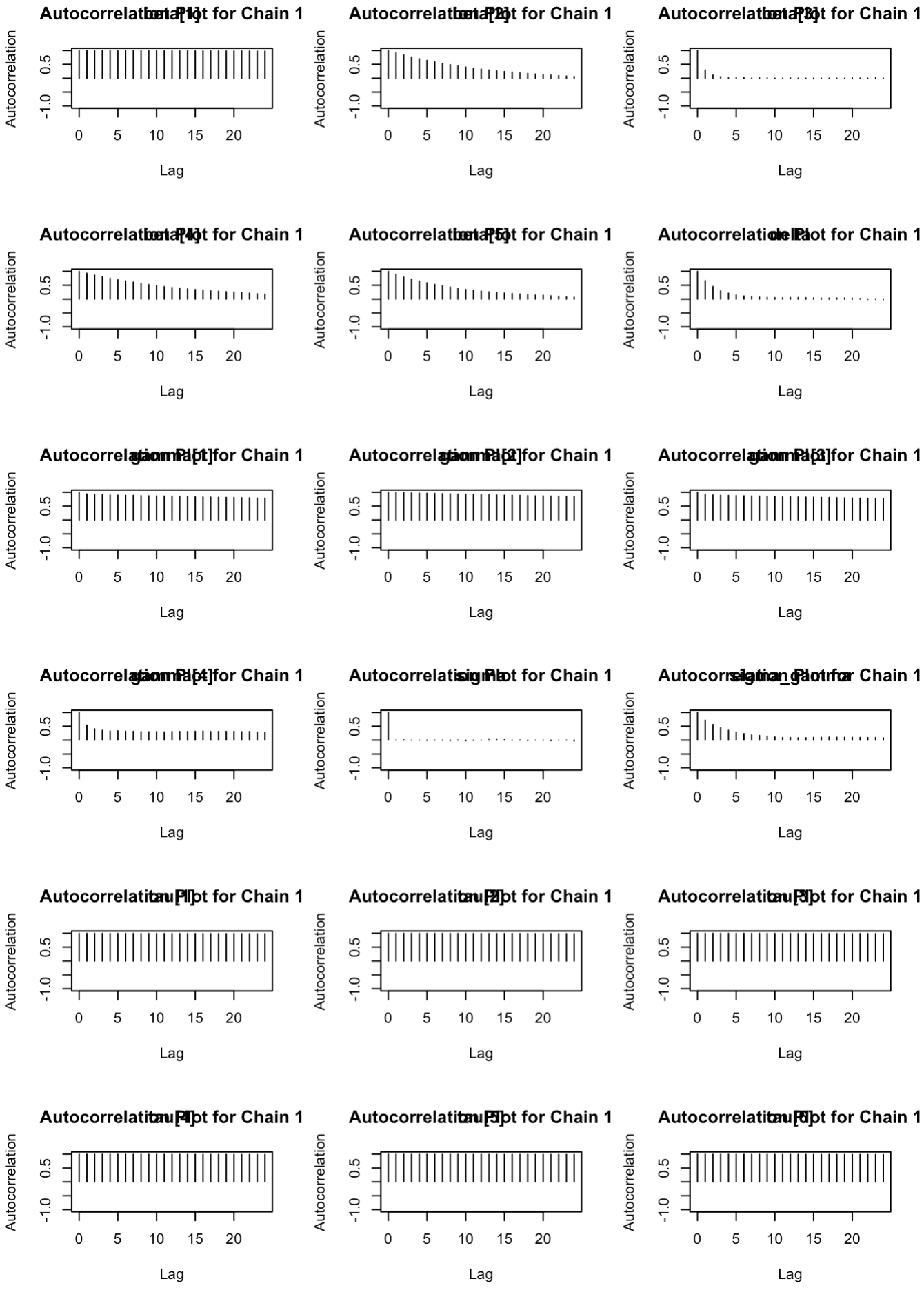

Traceplot for All Chains
1.0
0.5
0.0
-0.5
-1.0
-1.5
3000
4000
5000
6000
7000
8000
Iterations


Traceplot for All Chains
1.5
1.0
0.5
0.0
-0.5
-1.0
3000
4000
5000
6000
7000
8000
Iterations





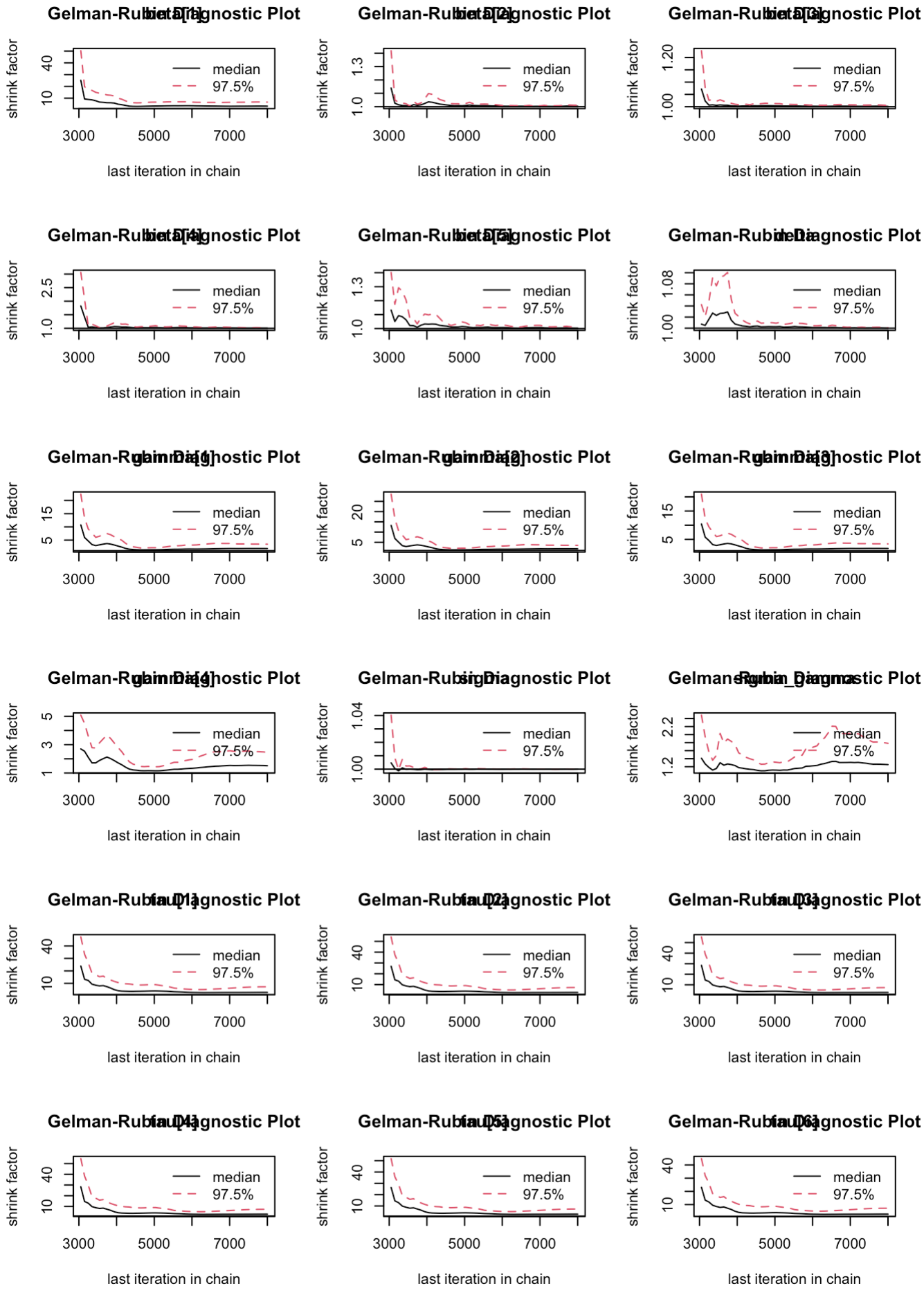

```
# Autocorrelation plot: 检查第一个链的自相关性
autocorr.plot(samples[[1]], main = "Autocorrelation Plot for Chain 1")
```


Autocorrelation Plot for Chain 1
Autocorrelation
Lag
-1.0
0.5
0
5
10
15
20

```r
### Step 2: Gelman-Rubin Diagnostic
# Gelman-Rubin 统计量（检查链的收敛性）
gelman_results <- gelman.diag(samples, autoburnin = FALSE)
print(gelman_results)  # 输出 Gelman-Rubin 统计量
```

```
## Potential scale reduction factors:
##
##             Point est. Upper C.I.
## beta[1]           3.25       6.65
## beta[2]           1.00       1.01
## beta[3]           1.00       1.00
## beta[4]           1.01       1.02
## beta[5]           1.00       1.01
## delta             1.00       1.00
## gamma[1]          1.81       3.46
## gamma[2]          1.82       3.50
## gamma[3]          1.79       3.40
## gamma[4]          1.51       2.45
## sigma             1.00       1.00
## sigma_gamma       1.25       1.78
## tau[1]            2.79       7.26
## tau[2]            2.80       7.32
## tau[3]            2.80       7.34
## tau[4]            2.80       7.33
## tau[5]            2.79       7.29
## tau[6]            2.78       7.24
##
## Multivariate psrf
##
## 2.45
```

```r
# Gelman-Rubin Diagnostic Plot
gelman.plot(samples, autoburnin = FALSE, main = "Gelman-Rubin Diagnostic Plot")
```




```
### Step 3: Summarize Posterior Distribution (after burn-in)
# 去除前 1000 次迭代作为 burn-in
burned_samples <- window(samples, start = 1000)
```

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

```
# 后验分布的摘要统计量
summary_results <- summary(burned_samples)
print(summary_results)
```

```
##
## Iterations = 3001:8000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean      SD  Naive SE Time-series SE
## beta[1]     -1.71387 1.00461 0.0082026      0.1568426
## beta[2]      0.51590 0.07708 0.0006293      0.0031187
## beta[3]     -0.16471 0.02392 0.0001953      0.0002734
## beta[4]      0.33035 0.04457 0.0003639      0.0020624
## beta[5]      1.49164 0.06753 0.0005514      0.0024149
## delta       -0.20519 0.04685 0.0003825      0.0008719
## gamma[1]    -0.80573 0.52760 0.0043078      0.1029949
## gamma[2]     0.01657 0.52064 0.0042510      0.1040575
## gamma[3]    -0.10105 0.52323 0.0042721      0.0972358
## gamma[4]    -0.01427 0.53841 0.0043961      0.0610979
## sigma        4.96266 2.89119 0.0236065      0.0236074
## sigma_gamma  0.90108 0.73877 0.0060320      0.0373224
## tau[1]      -1.12681 0.67751 0.0055318      0.1709526
## tau[2]      -0.45508 0.67709 0.0055284      0.1727328
## tau[3]       0.05994 0.67736 0.0055306      0.1760966
## tau[4]       0.50402 0.67816 0.0055372      0.1727037
## tau[5]       1.05175 0.67835 0.0055387      0.1764376
## tau[6]       1.64469 0.67864 0.0055411      0.1583978
##
## 2. Quantiles for each variable:
##
##                 2.5%     25%      50%      75%    97.5%
## beta[1]     -3.80564 -2.2913 -1.62643 -0.98713 -0.01620
## beta[2]      0.36449  0.4638  0.51450  0.56786  0.66906
## beta[3]     -0.21173 -0.1808 -0.16453 -0.14854 -0.11795
## beta[4]      0.24486  0.3001  0.32949  0.36125  0.41719
## beta[5]      1.35730  1.4463  1.49188  1.53614  1.62557
## delta       -0.29617 -0.2374 -0.20470 -0.17417 -0.11275
## gamma[1]    -2.08020 -1.1231 -0.66824 -0.44679  0.02582
## gamma[2]    -1.25946 -0.3059  0.15994  0.36248  0.86051
## gamma[3]    -1.37008 -0.4154  0.03169  0.24624  0.74753
## gamma[4]    -1.26373 -0.2915  0.05048  0.33453  0.92924
## sigma        0.25421  2.4603  4.92977  7.48693  9.72563
## sigma_gamma  0.24622  0.4428  0.66414  1.09631  2.90920
## tau[1]      -2.69780 -1.3508 -0.97859 -0.65198 -0.22831
## tau[2]      -2.02802 -0.6779 -0.30678  0.01699  0.44146
## tau[3]      -1.51613 -0.1623  0.21050  0.53539  0.95859
## tau[4]      -1.07525  0.2762  0.65618  0.98152  1.40509
## tau[5]      -0.52495  0.8237  1.20271  1.52844  1.95391
## tau[6]       0.07147  1.4149  1.79874  2.12269  2.54351
```

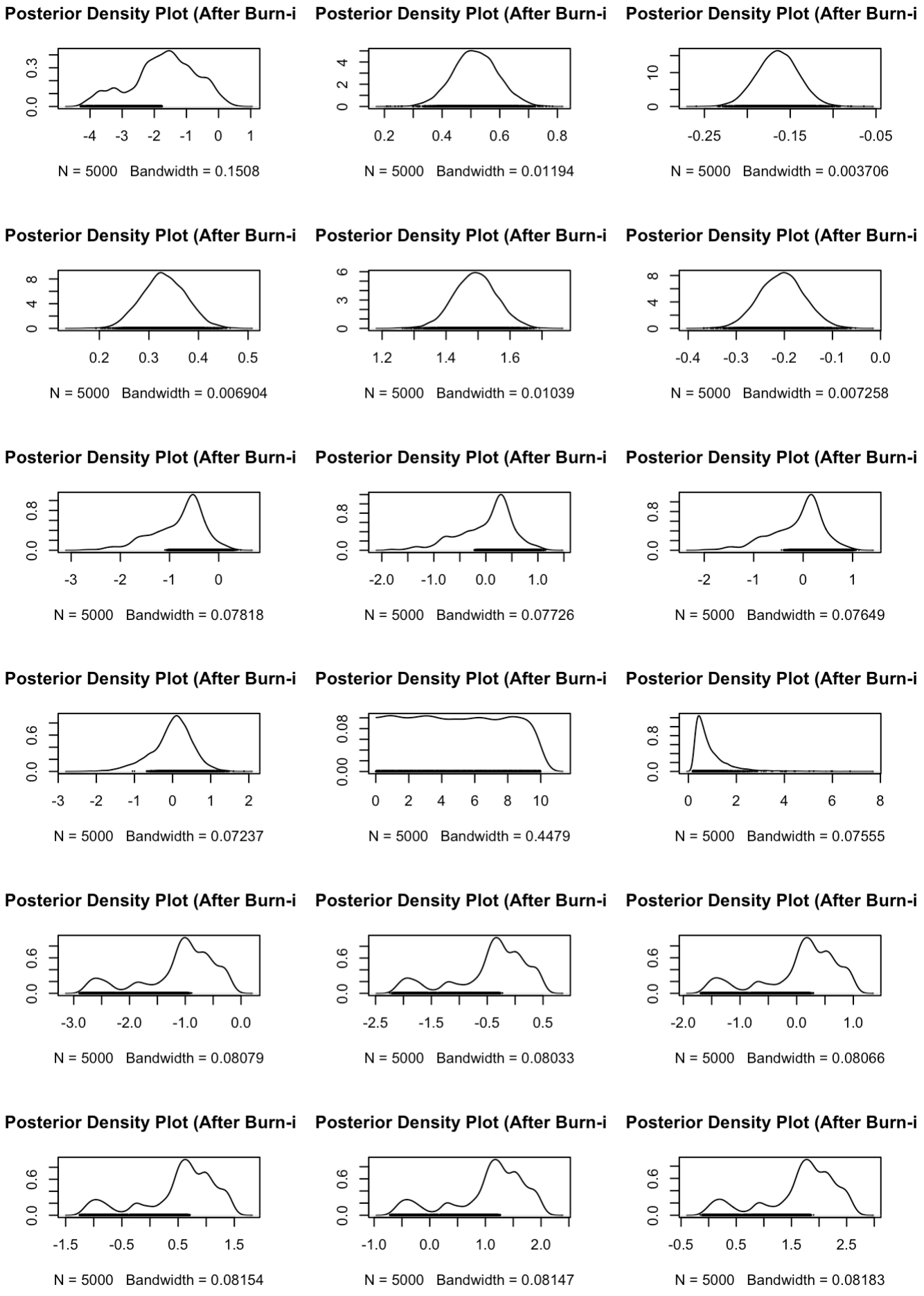
```
# Time-series SE: 验证 SE 是否小于 SD 的 1/20
cat("Time-series SE check:\n")
```

```
## Time-series SE check:
```

```
print(summary_results$statistics[, "Time-series SE"])
```

```
##      beta[1]      beta[2]      beta[3]      beta[4]      beta[5]        delta
## 0.1568425586 0.0031186715 0.0002734002 0.0020623507 0.0024149427 0.0008718736
##     gamma[1]     gamma[2]     gamma[3]     gamma[4]        sigma  sigma_gamma
## 0.1029948842 0.1040574745 0.0972358449 0.0610979465 0.0236073744 0.0373224132
##       tau[1]       tau[2]       tau[3]       tau[4]       tau[5]       tau[6]
## 0.1709525547 0.1727328479 0.1760965612 0.1727036782 0.1764375620 0.1583977789
```

```
### Step 4: 绘制后验密度图
# 绘制后验密度（去除 burn-in 的样本）
plot(burned_samples, trace = FALSE, density = TRUE,
     main = "Posterior Density Plot (After Burn-in)")
```


Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.1508
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.01194
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.003706
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.006904
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.01039
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.007258
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.07818
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.07726
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.07649
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.07237
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.4479
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.07555
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08079
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08033
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08066
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08154
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08147
Posterior Density Plot (After Burn-i
N = 5000 Bandwidth = 0.08183