STAFinalProject

2024-04-17

```r
require(leaps)
```

```
## Loading required package: leaps
```

```r
data = read.csv("/Users/zhaochong/Desktop/STA336/cancer.csv")
data$SMOKING = as.factor(data$SMOKING)
data$YELLOW_FINGERS = as.factor(data$YELLOW_FINGERS)
data$ANXIETY = as.factor(data$ANXIETY)
data$PEER_PRESSURE= as.factor(data$PEER_PRESSURE)
data$CHRONIC.DISEASE= as.factor(data$CHRONIC.DISEASE)
data$FATIGUE= as.factor(data$FATIGUE)
data$WHEEZING= as.factor(data$WHEEZING)
data$ALCOHOL.CONSUMING= as.factor(data$ALCOHOL.CONSUMING)
data$COUGHING= as.factor(data$COUGHING)
data$SHORTNESS.OF.BREATH= as.factor(data$SHORTNESS.OF.BREATH)
data$SWALLOWING.DIFFICULTY= as.factor(data$SWALLOWING.DIFFICULTY)
data$CHEST.PAIN= as.factor(data$CHEST.PAIN)
data$LUNG_CANCER= as.factor(data$LUNG_CANCER)
```

```r
# Load necessary library for combinations
if (!requireNamespace("combinat", quietly = TRUE)) {
    install.packages("combinat")
}
library(combinat)
```

```
##
## Attaching package: 'combinat'

## The following object is masked from 'package:utils':
##
##     combn
```

```r
# Define the function to fit logistic regression and calculate AIC
fit_logistic <- function(predictors, data, response = "LUNG_CANCER") {
    formula <- as.formula(paste(response, "~", paste(predictors, collapse = "+")))
    model <- glm(formula, data = data, family = "binomial")
    return(AIC(model))
}

# Prepare the data and define predictors
predictors <- names(data)[names(data) != "LUNG_CANCER"]  # Exclude the response variable from predictor
```

```r
# Generate all possible combinations of predictors, up to 5 predictors
subsets <- list()
max_predictors <- min(3, length(predictors))  # Limit to 5 predictors or fewer if there are not enough 
for (k in 1:max_predictors) {
    subsets[[k]] <- combn(predictors, k, simplify = FALSE)
}
subsets <- unlist(subsets, recursive = FALSE)

# Evaluate all subsets using the AIC of logistic regression
aic_values <- sapply(subsets, function(subset) {
    fit_logistic(subset, data = data, response = "LUNG_CANCER")
})

# Identify the best model based on the lowest AIC
best_model_index <- which.min(aic_values)
best_subset <- subsets[[best_model_index]]

# Fit the best logistic regression model
final_model <- glm(as.formula(paste("LUNG_CANCER ~", paste(best_subset, collapse = "+"))), data = data,
summary(final_model)
```

```
##
## Call:
## glm(formula = as.formula(paste("LUNG_CANCER ~", paste(best_subset,
##     collapse = "+"))), family = "binomial", data = data)
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -2.8096     0.6948  -4.044 5.27e-05 ***
## ALLERGY                   2.3838     0.5244   4.545 5.48e-06 ***
## COUGHING2                 1.6368     0.4425   3.699 0.000217 ***
## SWALLOWING.DIFFICULTY2    2.5511     0.5339   4.778 1.77e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 234.3  on 308  degrees of freedom
## Residual deviance: 153.6  on 305  degrees of freedom
## AIC: 161.6
##
## Number of Fisher Scoring iterations: 6
```

```r
data2=data
data2$LUNG_CANCER <- as.numeric(data$LUNG_CANCER)-1
data2$YELLOW_FINGERS <- as.numeric(data$YELLOW_FINGERS)-1
data2$PEER_PRESSURE <- as.numeric(data$PEER_PRESSURE)-1
data2$FATIGUE <- as.numeric(data$FATIGUE)-1
data2$ALLERGY <- as.numeric(data$ALLERGY)-1
data2$ALCOHOL.CONSUMING <- as.numeric(data$ALCOHOL.CONSUMING)-1

final_model <- glm(LUNG_CANCER~YELLOW_FINGERS+PEER_PRESSURE+FATIGUE+ALLERGY+ALCOHOL.CONSUMING, data=dat
summary(final_model)
```

```
##
## Call:
## glm(formula = LUNG_CANCER ~ YELLOW_FINGERS + PEER_PRESSURE +
##     FATIGUE + ALLERGY + ALCOHOL.CONSUMING, family = "binomial",
##     data = data2)
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -3.3346     0.6977  -4.780 1.76e-06 ***
## YELLOW_FINGERS      2.7642     0.6088   4.541 5.60e-06 ***
## PEER_PRESSURE       1.4921     0.5111   2.919  0.00351 **
## FATIGUE             2.4645     0.5836   4.223 2.41e-05 ***
## ALLERGY             2.5733     0.6116   4.207 2.58e-05 ***
## ALCOHOL.CONSUMING   2.8220     0.5896   4.786 1.70e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 234.3  on 308  degrees of freedom
## Residual deviance: 125.3  on 303  degrees of freedom
## AIC: 137.3
##
## Number of Fisher Scoring iterations: 7
```

```r
# Adjusted Log-Likelihood function for logistic regression
log_likelihood <- function(beta, data) {

    eta <- beta[1] + beta[2] * data$YELLOW_FINGERS + beta[3] * data$PEER_PRESSURE +
       beta[4] * data$FATIGUE + beta[5] * data$ALLERGY + beta[6] * data$ALCOHOL.CONSUMING

    #print(paste("eta", eta))

    p <- 1 / (1 + exp(-eta))

    # Adjust probabilities to avoid 0 or 1
    p <- pmin(pmax(p, 1e-10), 1 - 1e-10)

    # Calculate the log-likelihood
    likelihood_value <- sum(dbinom(data$LUNG_CANCER, size = 1, prob = p, log = TRUE))
    return(likelihood_value)
}

# Define the prior distribution (normal in this case)
log_prior <- function(beta, mean = 0, sd = 50) {
    sum(dnorm(beta, mean = mean, sd = sd, log = TRUE))
}

# Metropolis-Hastings algorithm
# Metropolis-Hastings algorithm with enhanced debugging
metropolis_hastings <- function(data, iterations, init_beta, proposal_sd) {
    chain <- matrix(NA, nrow = iterations, ncol = length(init_beta))
    chain[1,] <- init_beta
    for (i in 2:iterations) {
        current_beta <- chain[i-1,]
        proposed_beta <- rnorm(length(current_beta), mean = current_beta, sd = proposal_sd)

        current_likelihood <- log_likelihood(current_beta, data)
        #print(paste("Current likelihood:", current_likelihood))
        proposed_likelihood <- log_likelihood(proposed_beta, data)
        #print(paste("proposed likelihood:", proposed_likelihood))
        current_prior <- log_prior(current_beta)
        #print(paste("current_prior:", current_prior))
        proposed_prior <- log_prior(proposed_beta)
        #print(paste("proposed_prior:", proposed_prior))

        log_acceptance_ratio <- proposed_likelihood + proposed_prior - current_likelihood - current_pri


        if (log(runif(1)) < log_acceptance_ratio) {
            chain[i,] <- proposed_beta
        } else {
            chain[i,] <- current_beta
        }
    }
    return(chain)
}
```

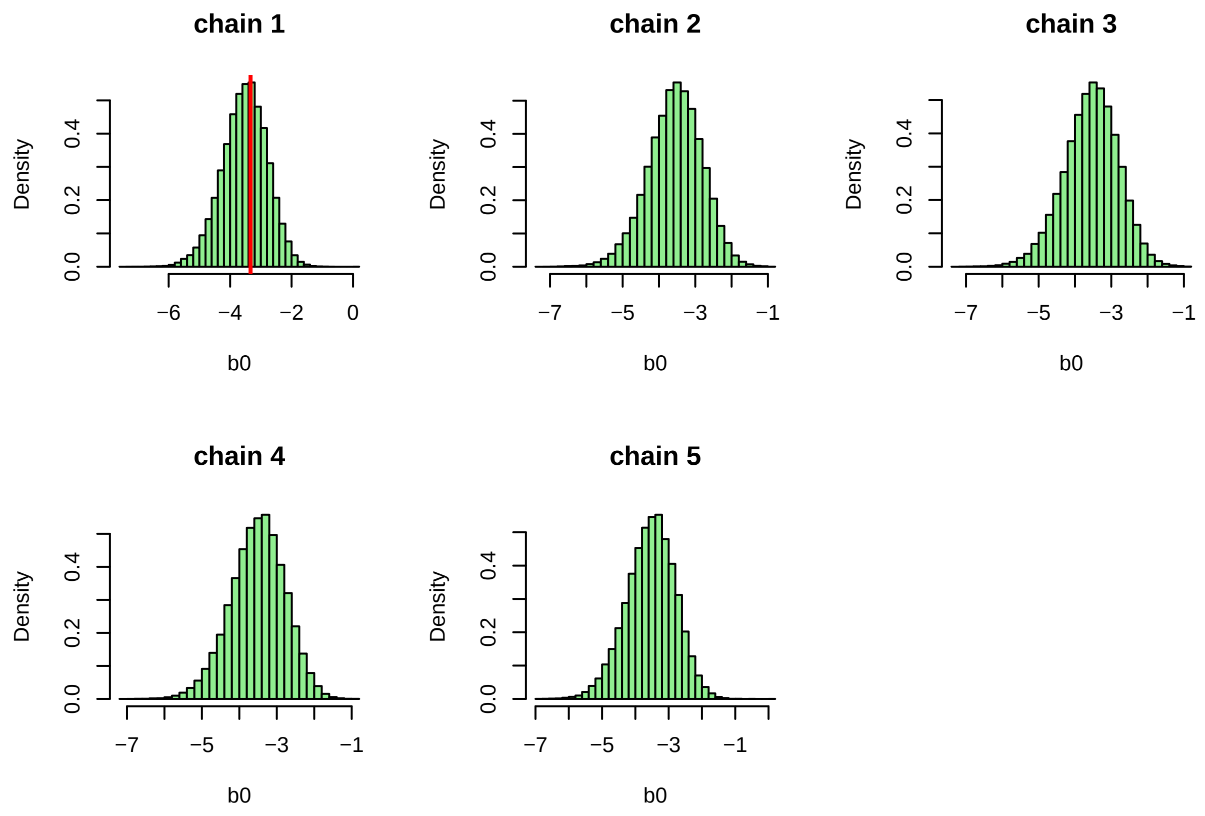
Fine Tuning

```
# Calculate acceptance rate
# Initial proposal standard deviation
proposal_sd <- 0.1  # Start with a reasonable guess

# Run the MH algorithm
chain <- metropolis_hastings(data = data2, iterations = 10000, init_beta = rep(0, 6), proposal_sd = prop

# Calculate acceptance rate
acceptance_rate <- mean(diff(chain) != 0)

# Adjust proposal_sd based on the acceptance rate
if (acceptance_rate < 0.2) {
    proposal_sd <- proposal_sd * 0.5  # Decrease if too few acceptances
} else if (acceptance_rate > 0.5) {
    proposal_sd <- proposal_sd * 2  # Increase if too many acceptances
}
```

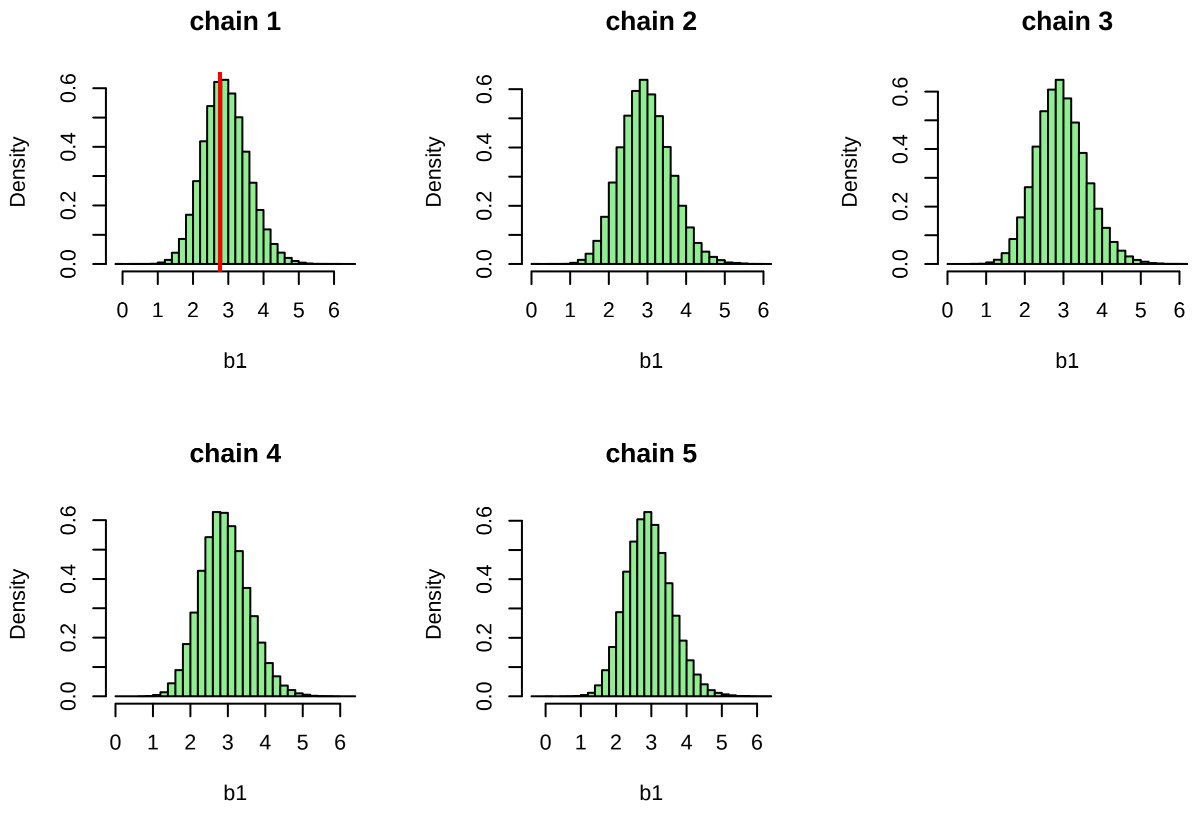
```
# Parameters
init_beta <- c(-3,0,0,0,0,0)
proposal_sd <- rep(0.4, 6)
iterations <- 500000

# Run five separate chains
chains <- list()
```

```
for (j in 1:5) {
    chains[[j]] <- metropolis_hastings(data2, iterations, init_beta, proposal_sd)
}
```

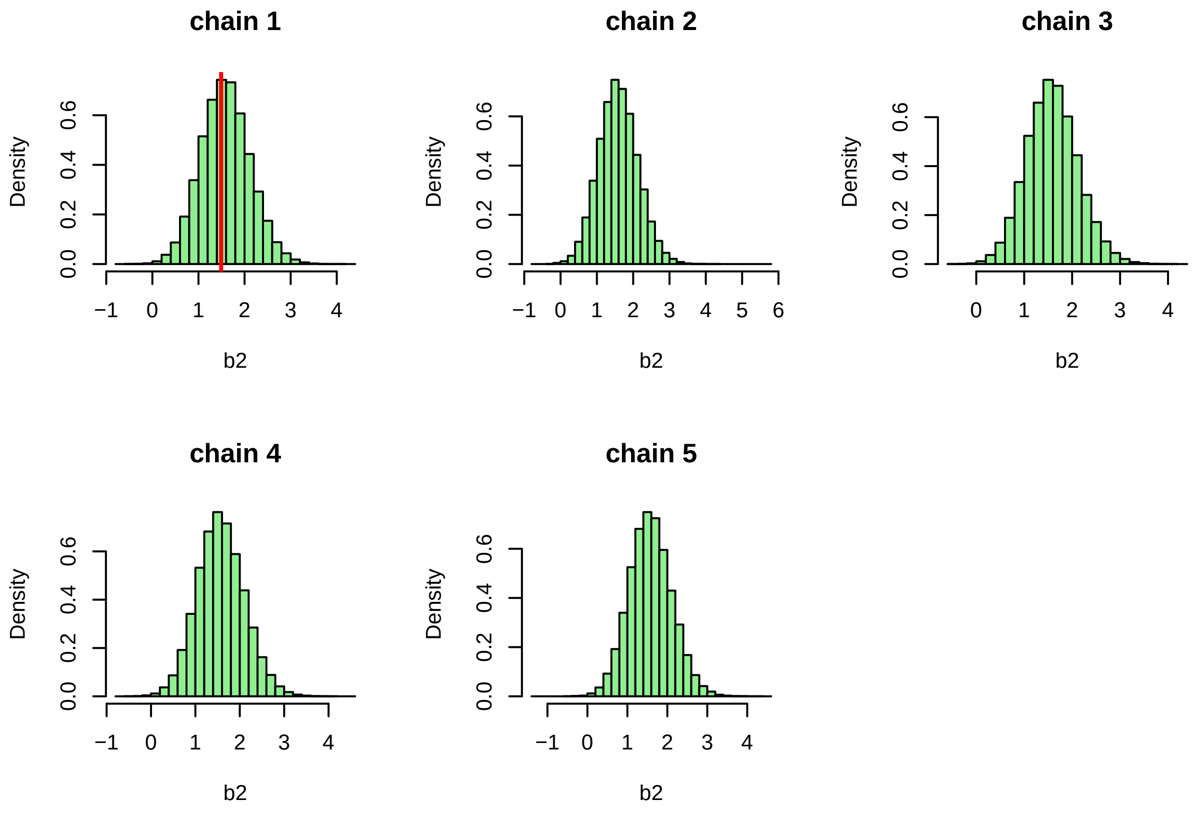
```
par(mfrow = c(2, 3))

hist(chains[[1]][,1], col = "lightgreen", breaks=50,border = "black",xlab = "b0", ylab = "Density", mai
abline(v = -3.3346, col = "red", lwd = 2)
hist(chains[[2]][,1], col = "lightgreen", breaks=30,border = "black",xlab = "b0", ylab = "Density", mai
hist(chains[[3]][,1], col = "lightgreen", breaks=30,border = "black",xlab = "b0", ylab = "Density", mai
hist(chains[[4]][,1], col = "lightgreen", breaks=30,border = "black",xlab = "b0", ylab = "Density", mai
hist(chains[[5]][,1], col = "lightgreen",  breaks=30,border = "black",xlab = "b0", ylab = "Density", ma

par(mfrow = c(2, 3))
```



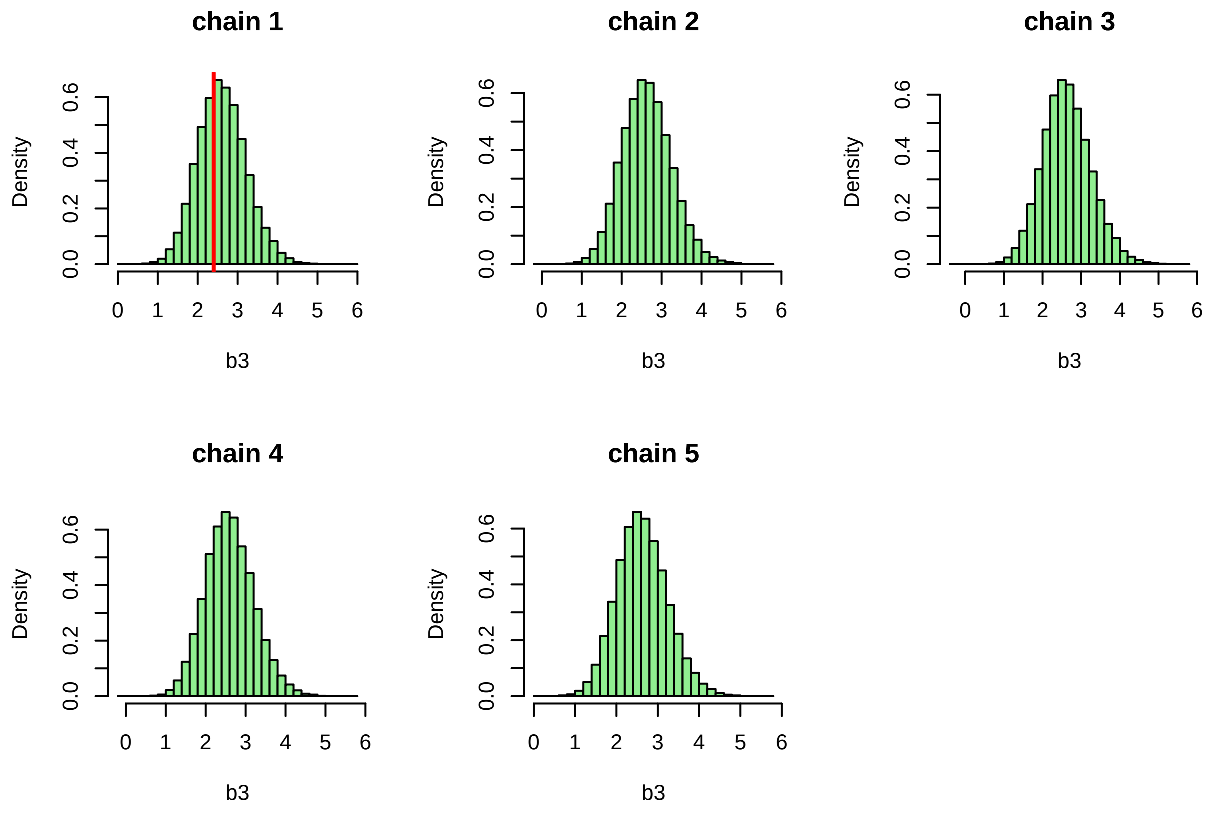
```
hist(chains[[1]][,2], col = "lightgreen",breaks=30, border = "black",xlab = "b1", ylab = "Density", mai
abline(v = 2.7642, col = "red", lwd = 2)
hist(chains[[2]][,2], col = "lightgreen",breaks=30, border = "black",xlab = "b1", ylab = "Density", mai
hist(chains[[3]][,2], col = "lightgreen",breaks=30, border = "black",xlab = "b1", ylab = "Density", mai
hist(chains[[4]][,2], col = "lightgreen",breaks=30, border = "black",xlab = "b1", ylab = "Density", mai
hist(chains[[5]][,2], col = "lightgreen",breaks=30,border = "black",xlab = "b1", ylab = "Density", main

par(mfrow = c(2, 3))
```



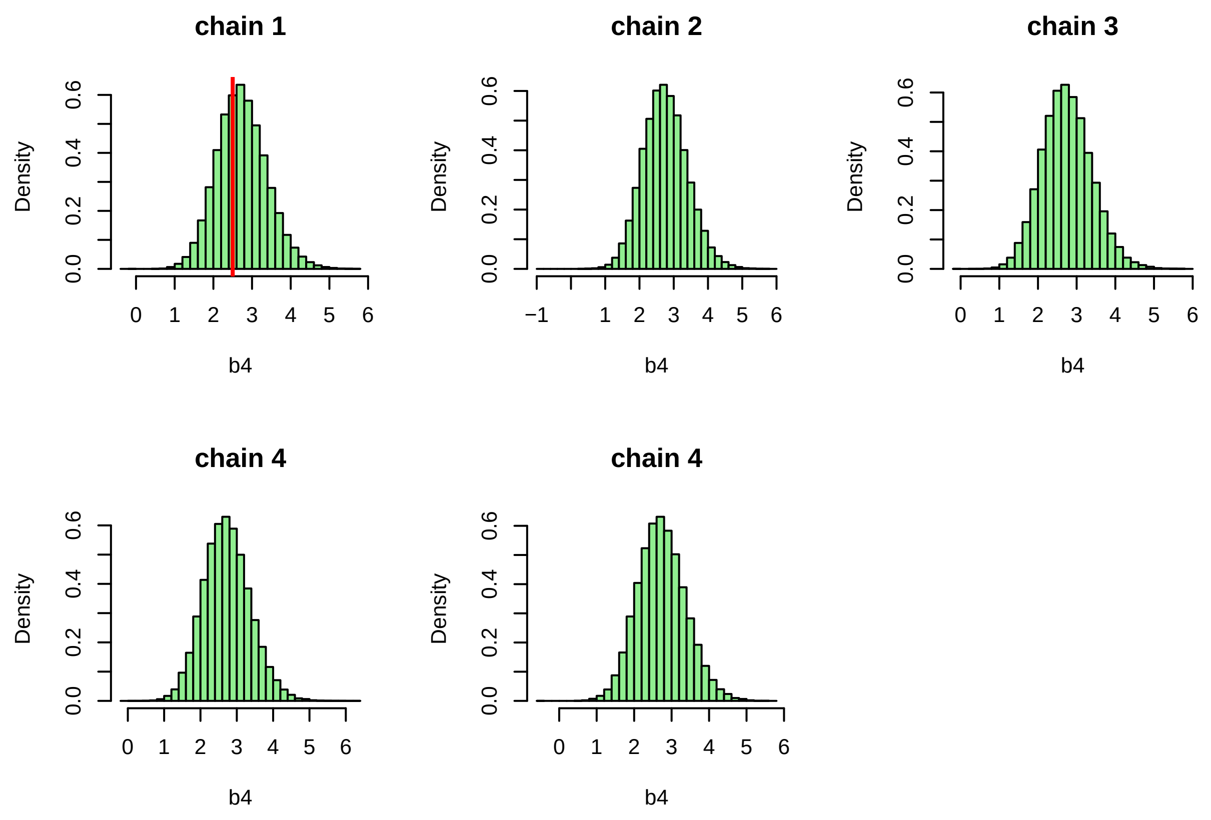
```
hist(chains[[1]][,3], col = "lightgreen",breaks=30, border = "black",xlab = "b2", ylab = "Density", mai
abline(v = 1.4921, col = "red", lwd = 2)
hist(chains[[2]][,3], col = "lightgreen",breaks=30, border = "black",xlab = "b2", ylab = "Density", mai
hist(chains[[3]][,3], col = "lightgreen",breaks=30, border = "black",xlab = "b2", ylab = "Density", mai
hist(chains[[4]][,3], col = "lightgreen",breaks=30, border = "black",xlab = "b2", ylab = "Density", mai
hist(chains[[5]][,3], col = "lightgreen",breaks=30, border = "black",xlab = "b2", ylab = "Density", mai


par(mfrow = c(2, 3))
```



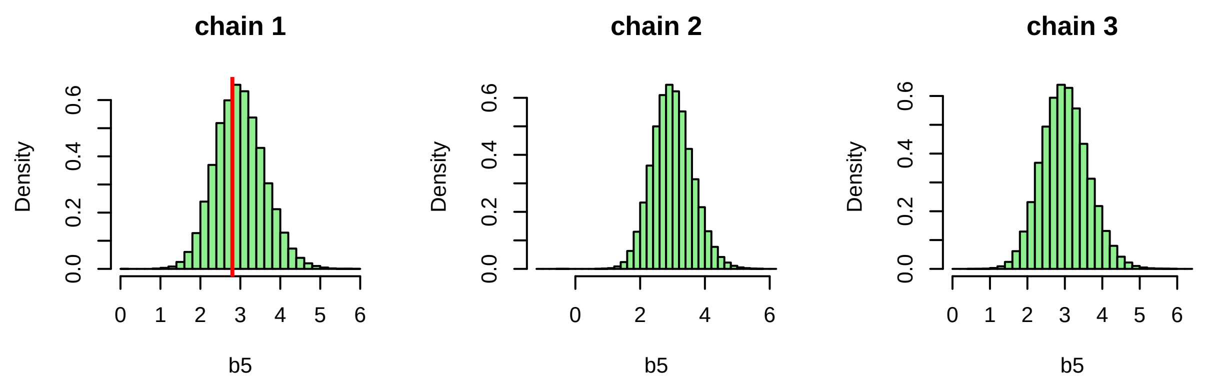
```
hist(chains[[1]][,4], col = "lightgreen",breaks=30, border = "black",xlab = "b3", ylab = "Density", mai
abline(v = 2.4, col = "red", lwd = 2)
hist(chains[[2]][,4], col = "lightgreen",breaks=30, border = "black",xlab = "b3", ylab = "Density", mai
hist(chains[[3]][,4], col = "lightgreen",breaks=30, border = "black",xlab = "b3", ylab = "Density", mai
hist(chains[[4]][,4], col = "lightgreen",breaks=30, border = "black",xlab = "b3", ylab = "Density", mai
hist(chains[[5]][,4], col = "lightgreen",breaks=30, border = "black",xlab = "b3", ylab = "Density", mai

par(mfrow = c(2, 3))
```

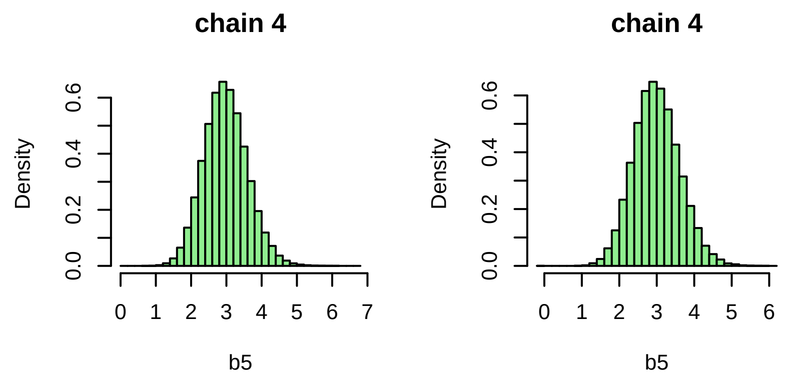


```
hist(chains[[1]][,5], col = "lightgreen",breaks=30, border = "black",xlab = "b4", ylab = "Density", mai
abline(v = 2.5, col = "red", lwd = 2)
hist(chains[[2]][,5], col = "lightgreen",breaks=30, border = "black",xlab = "b4", ylab = "Density", mai
hist(chains[[3]][,5], col = "lightgreen",breaks=30, border = "black",xlab = "b4", ylab = "Density", mai
hist(chains[[4]][,5], col = "lightgreen",breaks=30, border = "black",xlab = "b4", ylab = "Density", mai
hist(chains[[5]][,5], col = "lightgreen",breaks=30, border = "black",xlab = "b4", ylab = "Density", mai

par(mfrow = c(2, 3))
```



```
hist(chains[[1]][,6], col = "lightgreen",breaks=30, border = "black",xlab = "b5", ylab = "Density", mai
abline(v = 2.8, col = "red", lwd = 2)
hist(chains[[2]][,6], col = "lightgreen",breaks=30, border = "black",xlab = "b5", ylab = "Density", mai
hist(chains[[3]][,6], col = "lightgreen",breaks=30, border = "black",xlab = "b5", ylab = "Density", mai
hist(chains[[4]][,6], col = "lightgreen",breaks=30, border = "black",xlab = "b5", ylab = "Density", mai
hist(chains[[5]][,6], col = "lightgreen",breaks=30, border = "black",xlab = "b5", ylab = "Density", mai
```





```r
# Calculate 95% credible interval for each coefficient
credible_intervals <- lapply(chains, function(chain) {
  apply(chain, 2, function(col) quantile(col, c(0.025, 0.975)))
})

# Print the credible intervals
for (i in 1:length(credible_intervals)) {
  cat("Chain", i, "credible intervals:\n")
  print(credible_intervals[[i]])
}
```

```
## Chain 1 credible intervals:
##            [,1]     [,2]      [,3]     [,4]     [,5]     [,6]
## 2.5%  -5.038325 1.761263 0.5778848 1.494595 1.545539 1.853269
## 97.5% -2.183009 4.254544 2.6678183 3.871531 4.088933 4.252106
## Chain 2 credible intervals:
##            [,1]     [,2]      [,3]     [,4]     [,5]     [,6]
## 2.5%  -5.088620 1.776297 0.5734547 1.487928 1.562444 1.853155
## 97.5% -2.185993 4.298580 2.6868372 3.916924 4.087263 4.281321
## Chain 3 credible intervals:
##            [,1]     [,2]      [,3]     [,4]     [,5]     [,6]
## 2.5%  -5.108920 1.754197 0.5799466 1.470830 1.565453 1.857577
## 97.5% -2.176271 4.334558 2.6852006 3.935077 4.082043 4.282954
## Chain 4 credible intervals:
##            [,1]     [,2]      [,3]     [,4]     [,5]     [,6]
## 2.5%  -5.006396 1.751423 0.5740011 1.478376 1.536285 1.840449
## 97.5% -2.168273 4.244325 2.6606531 3.865187 4.060382 4.243550
## Chain 5 credible intervals:
##            [,1]     [,2]      [,3]     [,4]     [,5]     [,6]
## 2.5%  -5.051233 1.768732 0.5674334 1.499675 1.545296 1.854785
## 97.5% -2.185121 4.283813 2.6595040 3.905303 4.072782 4.262923
```