STA3105 Bayesian Statistics

Jeong Geonwoo

DUE Friday, October 20

Q1

1. (a) Let $X_i \in \mathbb{R}^4$ be the predictors for i th observation. For $i = 1, \cdots, 1,000$, simulate $X_i \sim N(0, \mathbf{I})$ independently. Here I is an identity matrix.

```r
X <- mvrnorm(n = 1000, mu = rep(0, 4), Sigma = diag(4))
```

1. (b) For $i = 1, \cdots, 1,000$, simulate $Y_i \sim \text{Binomial}\left(10, \frac{\exp(X_i'\beta)}{1+\exp(X_i'\beta)}\right)$ independently. Set the true regression coefficient value as $\beta = (0.5, -0.5, 0, 1)$.

```r
mu <- function(X, beta) {
  exp_Xbeta <- exp(X %*% beta)
  return(exp_Xbeta / (1 + exp_Xbeta))
}

beta_true <- c(0.5, -0.5, 0, 1)

mu_X <- mu(X, beta_true)
Y <- rbinom(1000, 10, mu_X)
```

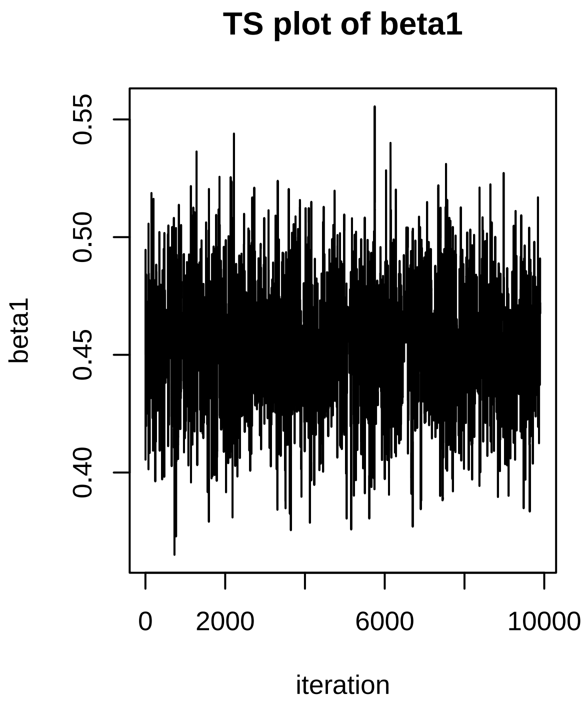
2. (30 points) Implement the MCMC algorithm using the simulated dataset in Problem 1. Here, you should write down the code without using any packages (e.g., nimble, adaptMCMC). Report the trace plots, density plots, 95% HPD intervals, posterior mean, acceptance probability, and effective sample size for all parameters.

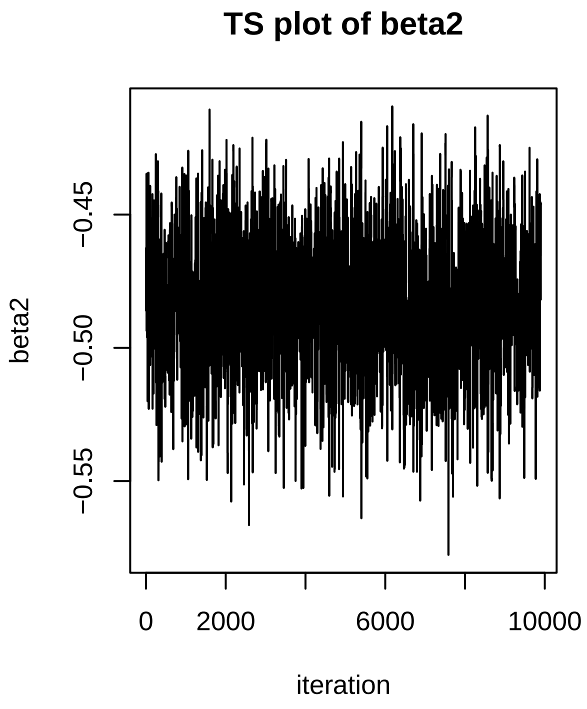
```r
log_likelihood <- function(Y, X, beta) {
  p <- mu(X, beta)
  return(sum(Y * log(p) + (10 - Y) * log(1 - p)))
}

log_prior <- function(beta) {
  return(-beta^2 / 20)
}

# Run the Metropolis-Hastings Algorithm
iterations <- 10000
proposal_sd <- rep(0.1,4)
beta_samples <- matrix(NA, nrow = iterations, ncol = ncol(X))
# initialization
current_beta <- rep(0,4)
```

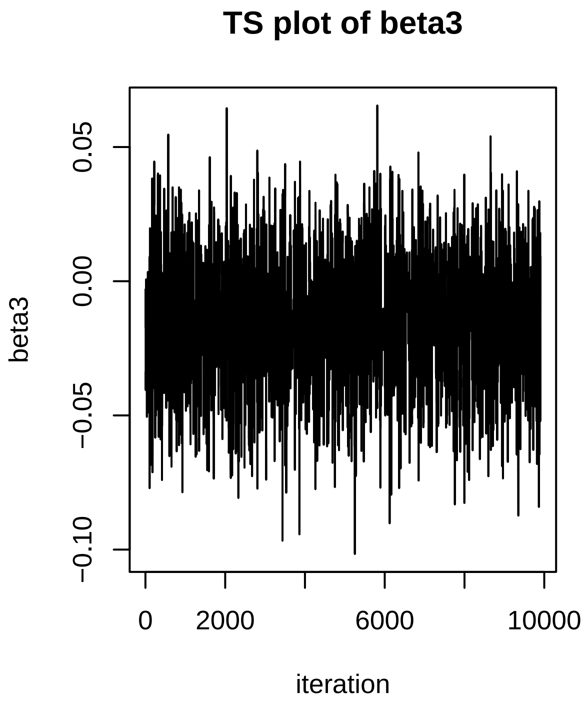
```r
beta_samples[1, ] <- current_beta
# the number of acceptance
acceptance = rep(0,4)

for(i in 2:iterations) {
  for (j in 1:ncol(X)){
    # Propose a beta_j
    proposed_beta <- current_beta
    proposed_beta[j] <- rnorm(1, current_beta[j], proposal_sd[j])
    # Calculate acceptance probability
    proposed_log_posterior <- log_likelihood(Y, X, proposed_beta) + log_prior(proposed_beta[j])
    current_log_posterior <- log_likelihood(Y, X, current_beta) + log_prior(current_beta[j])
    log_alpha <- proposed_log_posterior - current_log_posterior

    # Accept or reject the new beta
    if (is.nan(log_alpha) | abs(log_alpha)==Inf){
    # if at least one posterior is nan or inf, -inf
    # reject

    } else if(log(runif(1)) < log_alpha) {
      # the condition of accept
      current_beta[j] <- proposed_beta[j]
      acceptance[j] = acceptance[j] + 1
    }
    beta_samples[i, ] <- current_beta
  }
}

# Burn-in (After Checking TS plot, then I modified!)
burn_in <- 100
beta_samples <- beta_samples[seq(burn_in, nrow(beta_samples)), ]
```

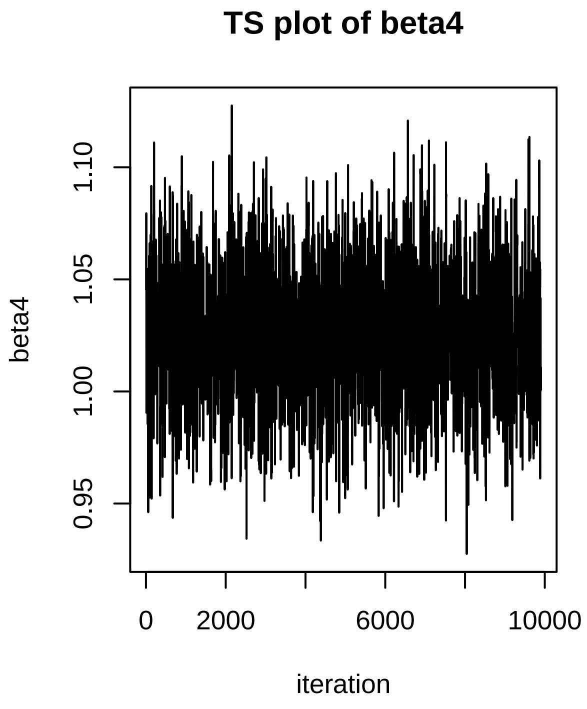
First, Trace Plot

```r
par(mfrow=c(2,2))
ts.plot(beta_samples[,1], main="TS plot of beta1", xlab="iteration", ylab="beta1")
ts.plot(beta_samples[,2], main="TS plot of beta2", xlab="iteration", ylab="beta2")
ts.plot(beta_samples[,3], main="TS plot of beta3", xlab="iteration", ylab="beta3")
ts.plot(beta_samples[,4], main="TS plot of beta4", xlab="iteration", ylab="beta4")
```



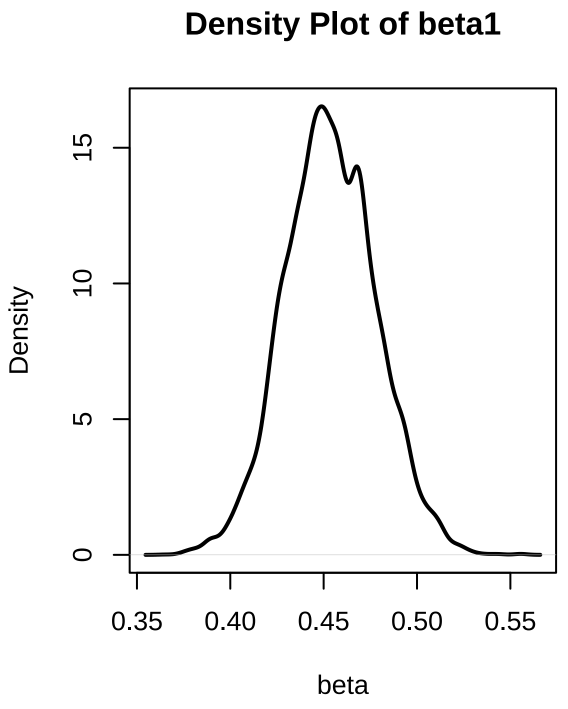


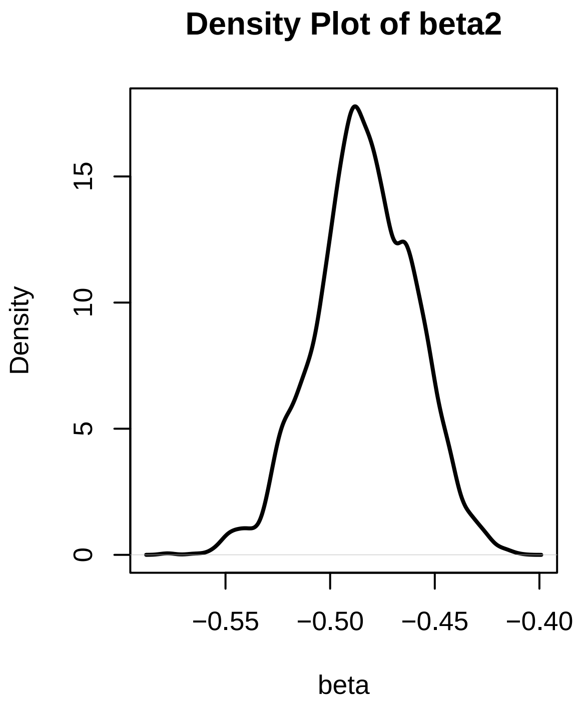


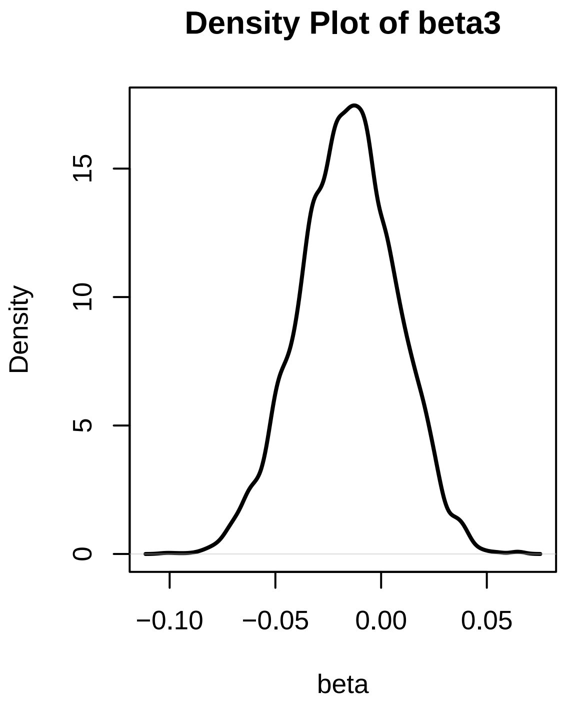


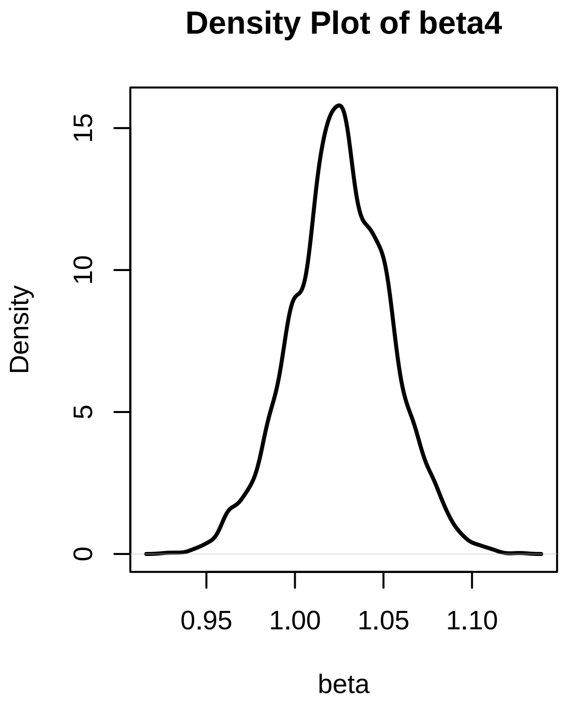
Second, Density Plot

```r
par(mfrow=c(2,2))
plot(density(beta_samples[,1]), main="Density Plot of beta1", xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples[,2]), main="Density Plot of beta2", xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples[,3]), main="Density Plot of beta3", xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples[,4]), main="Density Plot of beta4", xlab="beta", ylab="Density", lwd=2)
```









Third, 95% HPD interval

```r
cat("95% HPD intervals of beta1:", HPDinterval(as.mcmc(beta_samples[,1]), prob = 0.95), "\n",
    "95% HPD intervals of beta2:", HPDinterval(as.mcmc(beta_samples[,2]), prob = 0.95), "\n",
    "95% HPD intervals of beta3:", HPDinterval(as.mcmc(beta_samples[,3]), prob = 0.95), "\n",
    "95% HPD intervals of beta4:", HPDinterval(as.mcmc(beta_samples[,4]), prob = 0.95))
```

```
## 95% HPD intervals of beta1: 0.4050029 0.5001633
##  95% HPD intervals of beta2: -0.5296173 -0.4391619
##  95% HPD intervals of beta3: -0.06120063 0.02710687
##  95% HPD intervals of beta4: 0.9706863 1.080578
```

Fourth, Posterior mean

```r
cat("posterior mean of beta1:", mean(beta_samples[,1]), "\n",
    "posterior mean of beta2:", mean(beta_samples[,2]), "\n",
    "posterior mean of beta3:", mean(beta_samples[,3]), "\n",
    "posterior mean of beta4:", mean(beta_samples[,4]))
```

```
## posterior mean of beta1: 0.4536954
##  posterior mean of beta2: -0.4835865
##  posterior mean of beta3: -0.01574688
##  posterior mean of beta4: 1.025854
```

Fifth, Acceptance probability

```r
cat("acceptance probability of beta1:", length(unique(beta_samples[,1]))/iterations, "\n",
    "acceptance probability of beta2:", length(unique(beta_samples[,2]))/iterations, "\n",
    "acceptance probability of beta3:", length(unique(beta_samples[,3]))/iterations, "\n",
    "acceptance probability of beta4:", length(unique(beta_samples[,4]))/iterations)
```

```
## acceptance probability of beta1: 0.2792
##  acceptance probability of beta2: 0.2841
##  acceptance probability of beta3: 0.2687
##  acceptance probability of beta4: 0.3142
```

Sixth, Effective sample size (after thinning)

```r
# Thinning (If we need !)
# thinning = 2
# beta_samples <- beta_samples[seq(1, nrow(beta_samples), by=thinning), ]

# Effective sample size (after thinning)
cat("effective sample size of beta1:", effectiveSize(beta_samples[,1]), "\n",
    "effective sample size of beta2:", effectiveSize(beta_samples[,2]), "\n",
    "effective sample size of beta3:", effectiveSize(beta_samples[,3]), "\n",
    "effective sample size of beta4:", effectiveSize(beta_samples[,4]))
```

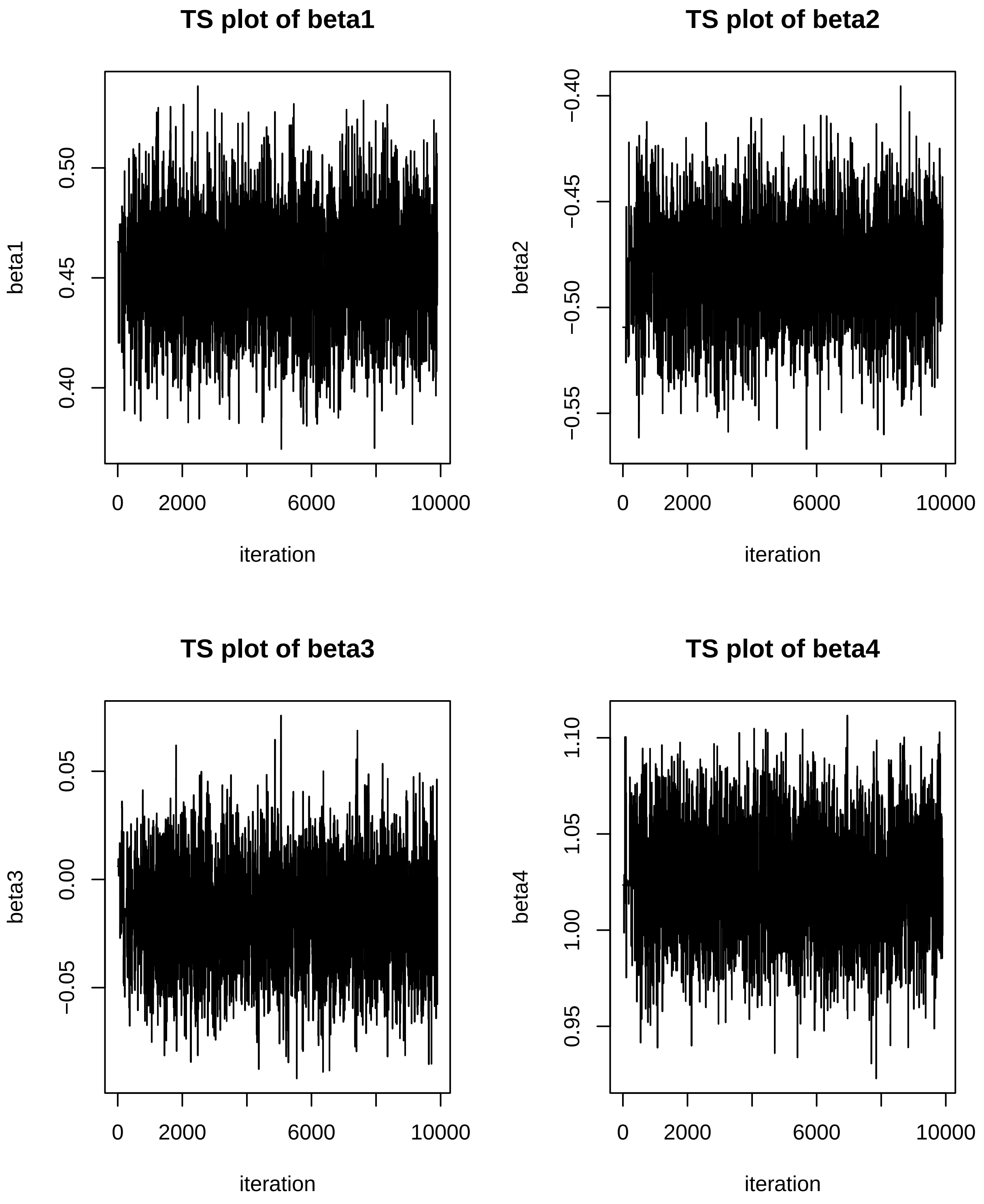
```
## effective sample size of beta1: 1972.499
##  effective sample size of beta2: 1847.905
##  effective sample size of beta3: 1793.851
##  effective sample size of beta4: 1934.049
```

3. (30 points) Implement the MCMC algorithm using the simulated dataset in Problem 1. Here, you should use nimble package to implement the algorithm. Report the trace plots, density plots, 95% HPD intervals, posterior mean, acceptance probability, and effective sample size for all parameters.

```r
model_string <- nimbleCode({
  for(i in 1:length(Y)) {
    logit(p[i]) <- inprod(X[i, 1:4], beta[1:4])
    Y[i] ~ dbin(p[i], 10)
  }
  for(j in 1:4) {
    beta[j] ~ dnorm(0, sd = sqrt(10))
  }
})

model <- nimbleModel(
  code = model_string,
  data = list(Y = Y),
  constants = list(X = X),
  inits = list(beta = rep(0,4)),
  name = "LogisticRegression"
)

compiled_model <- compileNimble(model)
mcmc <- buildMCMC(compiled_model)
compiled_mcmc <- compileNimble(mcmc, project = compiled_model)
results <- runMCMC(compiled_mcmc, niter = iterations, nburnin = burn_in, thin = 1)
beta_samples_nimble <- as.matrix(results, 'beta')
```

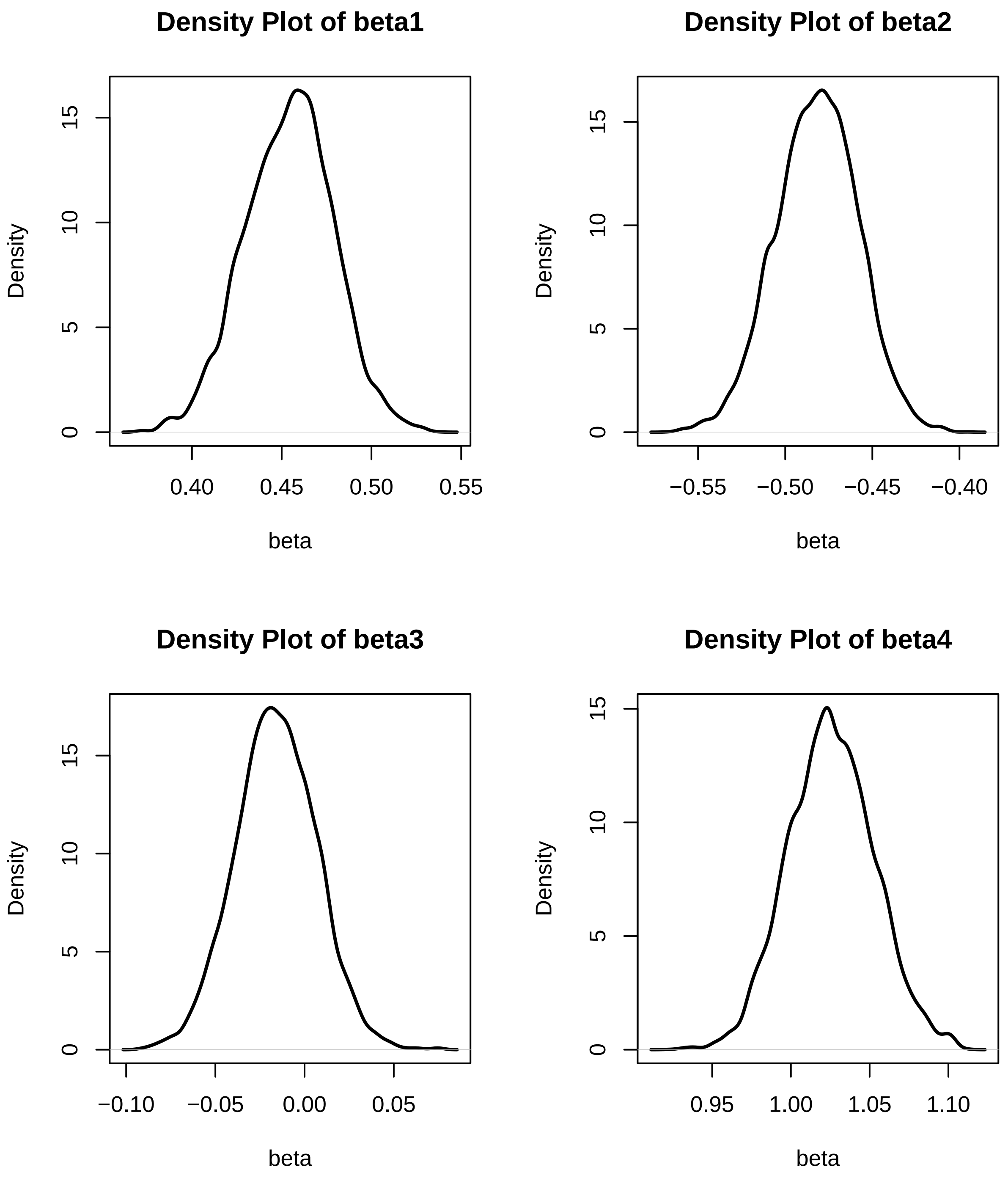
First, Trace Plot

```
par(mfrow=c(2,2))
ts.plot(beta_samples_nimble[,1], main="TS plot of beta1", xlab="iteration", ylab="beta1")
ts.plot(beta_samples_nimble[,2], main="TS plot of beta2", xlab="iteration", ylab="beta2")
ts.plot(beta_samples_nimble[,3], main="TS plot of beta3", xlab="iteration", ylab="beta3")
ts.plot(beta_samples_nimble[,4], main="TS plot of beta4", xlab="iteration", ylab="beta4")
```



Second, Density Plot

```r
par(mfrow=c(2,2))
plot(density(beta_samples_nimble[,1]), main="Density Plot of beta1",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_nimble[,2]), main="Density Plot of beta2",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_nimble[,3]), main="Density Plot of beta3",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_nimble[,4]), main="Density Plot of beta4",
     xlab="beta", ylab="Density", lwd=2)
```



Third, 95% HPD interval

```r
cat("95% HPD intervals of beta1:", HPDinterval(as.mcmc(beta_samples_nimble[,1]), prob = 0.95), "\n",
    "95% HPD intervals of beta2:", HPDinterval(as.mcmc(beta_samples_nimble[,2]), prob = 0.95), "\n",
    "95% HPD intervals of beta3:", HPDinterval(as.mcmc(beta_samples_nimble[,3]), prob = 0.95), "\n",
    "95% HPD intervals of beta4:", HPDinterval(as.mcmc(beta_samples_nimble[,4]), prob = 0.95))
```

```
## 95% HPD intervals of beta1: 0.4038295 0.4998055
##  95% HPD intervals of beta2: -0.5278333 -0.436364
##  95% HPD intervals of beta3: -0.05966474 0.02849809
##  95% HPD intervals of beta4: 0.9736398 1.079281
```

Fourth, Posterior mean

```r
cat("posterior mean of beta1:", mean(beta_samples_nimble[,1]), "\n",
    "posterior mean of beta2:", mean(beta_samples_nimble[,2]), "\n",
    "posterior mean of beta3:", mean(beta_samples_nimble[,3]), "\n",
    "posterior mean of beta4:", mean(beta_samples_nimble[,4]))
```

```
## posterior mean of beta1: 0.4541151
##  posterior mean of beta2: -0.4819212
##  posterior mean of beta3: -0.01601147
##  posterior mean of beta4: 1.025706
```

Fifth, Acceptance probability

```r
cat("acceptance probability of beta1:", length(unique(beta_samples_nimble[,1]))/iterations, "\n",
    "acceptance probability of beta2:", length(unique(beta_samples_nimble[,2]))/iterations, "\n",
    "acceptance probability of beta3:", length(unique(beta_samples_nimble[,3]))/iterations, "\n",
    "acceptance probability of beta4:", length(unique(beta_samples_nimble[,4]))/iterations)
```

```
## acceptance probability of beta1: 0.4194
##  acceptance probability of beta2: 0.4184
##  acceptance probability of beta3: 0.4196
##  acceptance probability of beta4: 0.4196
```

Sixth, Effective sample size

```r
# Effective sample size (after thinning)
cat("effective sample size of beta1:", effectiveSize(beta_samples_nimble[,1]), "\n",
    "effective sample size of beta2:", effectiveSize(beta_samples_nimble[,2]), "\n",
    "effective sample size of beta3:", effectiveSize(beta_samples_nimble[,3]), "\n",
    "effective sample size of beta4:", effectiveSize(beta_samples_nimble[,4]))
```

```
## effective sample size of beta1: 2043.874
##  effective sample size of beta2: 1953.349
##  effective sample size of beta3: 2110.868
##  effective sample size of beta4: 2062.898
```

4. (30 points) Implement the MCMC algorithm using the same simulated dataset (Problem 1) but with different model specification as follows.

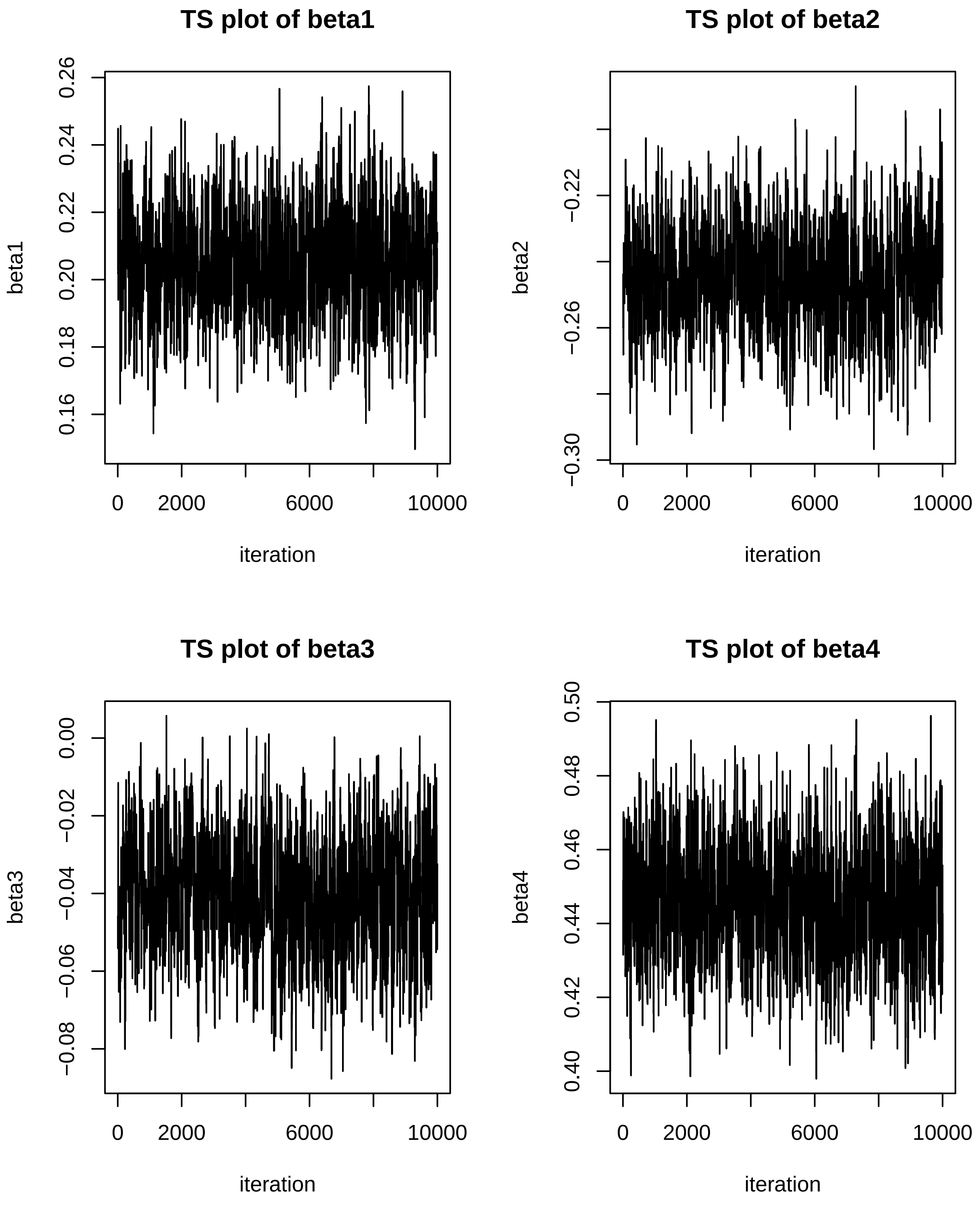
$$\mathbf{Y} \sim \text{Binomial}(20, \mu(\mathbf{X}))$$
$$\mu(\mathbf{X}) = \frac{\exp(\mathbf{X}\beta)}{1+\exp(\mathbf{X}\beta)}$$
$$\beta_j \sim N(0, 10) \text{ for } j = 1, \cdots, 4 \text{ independently}$$

Note that there is a model misspecification issue for this problem, because the simulated dataset is generated with $\text{Binomial}(10, \mu(\mathbf{X}))$ **but is fitted to** $\text{Binomial}(20, \mu(\mathbf{X}))$**. Here, you should use adaptMCMC) package to implement the algorithm. Report the trace plots, density plots, 95% HPD intervals, posterior mean, acceptance probability, and effective sample size for all parameters.**

```r
library(adaptMCMC)

init.pars = c(beta1=0, beta2=0, beta3=0, beta4=0)

# Define log posterior
log_posterior <- function(pars) {
  with(as.list(pars),{
    beta = pars
    logit_p <- X %*% beta
    p <- exp(logit_p) / (1 + exp(logit_p))
    log_prior <- sum(dnorm(beta, 0, sqrt(10), log = TRUE))
    log_lik <- sum(dbinom(Y, 20, p, log = TRUE))
    return (log_prior + log_lik)
  })
}

iterations = 13000
mcmc_results <- MCMC(p=log_posterior, n=iterations, init=rep(0,4),
                     scale=rep(10,4), adapt=TRUE, acc.rate=0.3)
beta_samples_adaptMCMC <- mcmc_results$samples

# Burn-in (After Checking TS plot, then I modified!)
burn_in <- 3000
beta_samples_adaptMCMC <- beta_samples_adaptMCMC[seq(burn_in, nrow(beta_samples_adaptMCMC)), ]
```

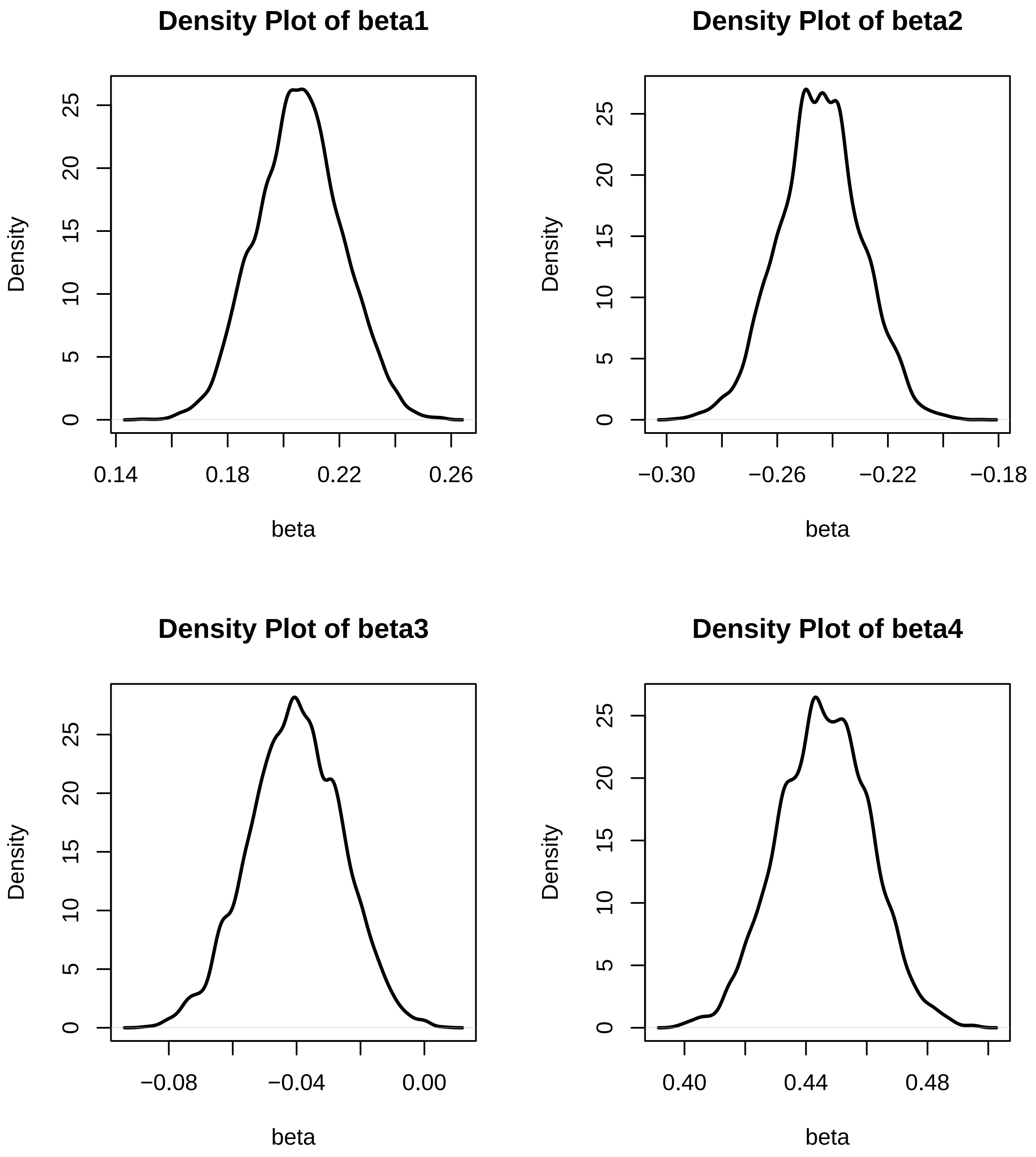
First, Trace Plot

```r
par(mfrow=c(2,2))
ts.plot(beta_samples_adaptMCMC[,1], main="TS plot of beta1", xlab="iteration", ylab="beta1")
ts.plot(beta_samples_adaptMCMC[,2], main="TS plot of beta2", xlab="iteration", ylab="beta2")
ts.plot(beta_samples_adaptMCMC[,3], main="TS plot of beta3", xlab="iteration", ylab="beta3")
ts.plot(beta_samples_adaptMCMC[,4], main="TS plot of beta4", xlab="iteration", ylab="beta4")
```



Second, Density Plot

```r
par(mfrow=c(2,2))
plot(density(beta_samples_adaptMCMC[,1]), main="Density Plot of beta1",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_adaptMCMC[,2]), main="Density Plot of beta2",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_adaptMCMC[,3]), main="Density Plot of beta3",
     xlab="beta", ylab="Density", lwd=2)
plot(density(beta_samples_adaptMCMC[,4]), main="Density Plot of beta4",
     xlab="beta", ylab="Density", lwd=2)
```



Third, 95% HPD interval

```r
cat("95% HPD intervals of beta1:", HPDinterval(as.mcmc(beta_samples_adaptMCMC[,1]), prob = 0.95), "\n",
    "95% HPD intervals of beta2:", HPDinterval(as.mcmc(beta_samples_adaptMCMC[,2]), prob = 0.95), "\n",
    "95% HPD intervals of beta3:", HPDinterval(as.mcmc(beta_samples_adaptMCMC[,3]), prob = 0.95), "\n",
    "95% HPD intervals of beta4:", HPDinterval(as.mcmc(beta_samples_adaptMCMC[,4]), prob = 0.95))
```

```
## 95% HPD intervals of beta1: 0.17687 0.2356215
##  95% HPD intervals of beta2: -0.2729568 -0.2148287
##  95% HPD intervals of beta3: -0.06686919 -0.01167976
##  95% HPD intervals of beta4: 0.4177836 0.4760812
```

Fourth, Posterior mean

```r
cat("posterior mean of beta1:", mean(beta_samples_adaptMCMC[,1]), "\n",
    "posterior mean of beta2:", mean(beta_samples_adaptMCMC[,2]), "\n",
    "posterior mean of beta3:", mean(beta_samples_adaptMCMC[,3]), "\n",
    "posterior mean of beta4:", mean(beta_samples_adaptMCMC[,4]))
```

```
## posterior mean of beta1: 0.205659
##  posterior mean of beta2: -0.2445997
##  posterior mean of beta3: -0.04042502
##  posterior mean of beta4: 0.4459479
```

Fifth, Acceptance probability

```r
cat("acceptance probability of beta1:", length(unique(beta_samples_adaptMCMC[,1]))/iterations, "\n",
    "acceptance probability of beta2:", length(unique(beta_samples_adaptMCMC[,2]))/iterations, "\n",
    "acceptance probability of beta3:", length(unique(beta_samples_adaptMCMC[,3]))/iterations, "\n",
    "acceptance probability of beta4:", length(unique(beta_samples_adaptMCMC[,4]))/iterations)
```

```
## acceptance probability of beta1: 0.2223846
##  acceptance probability of beta2: 0.2223846
##  acceptance probability of beta3: 0.2223846
##  acceptance probability of beta4: 0.2223846
```

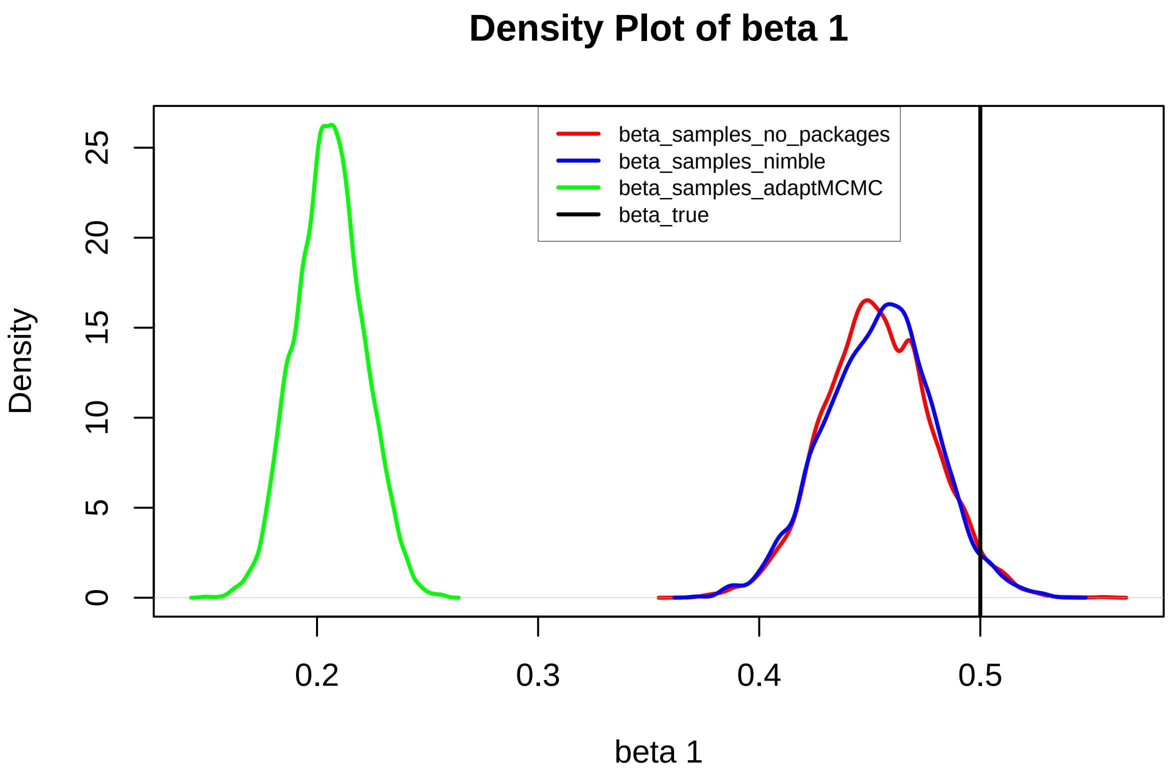
Sixth, Effective sample size

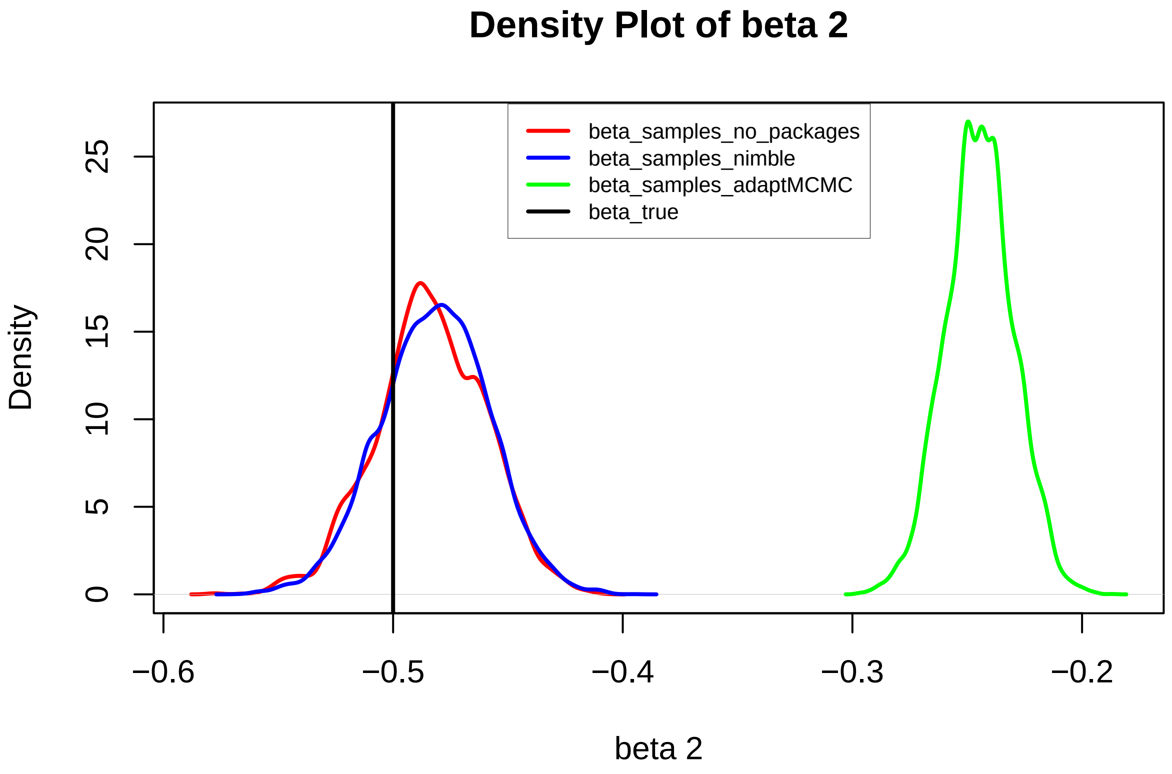
```r
# Thinning (If we need)
# thinning = 2
# beta_samples_adaptMCMC <- beta_samples_adaptMCMC[seq(1, nrow(beta_samples_adaptMCMC), by=thinning), ]

# Effective sample size (after thinning)
cat("effective sample size of beta1:", effectiveSize(beta_samples_adaptMCMC[,1]), "\n",
    "effective sample size of beta2:", effectiveSize(beta_samples_adaptMCMC[,2]), "\n",
    "effective sample size of beta3:", effectiveSize(beta_samples_adaptMCMC[,3]), "\n",
    "effective sample size of beta4:", effectiveSize(beta_samples_adaptMCMC[,4]))
```

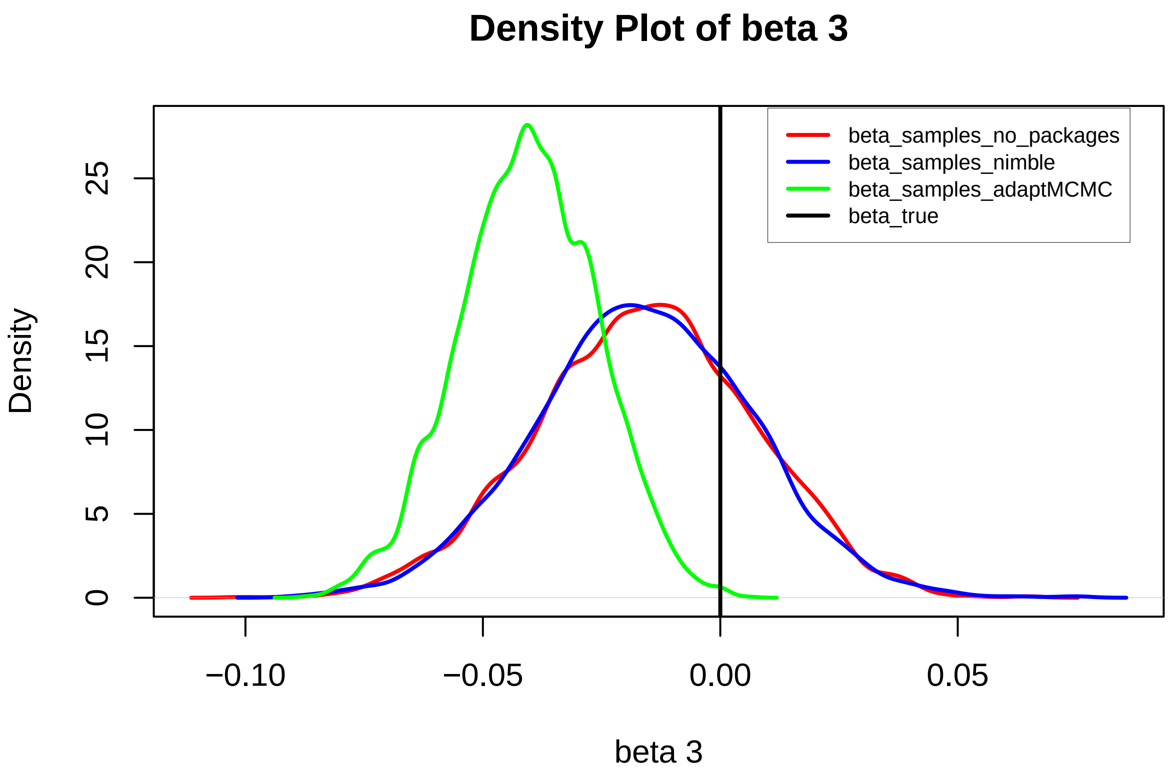
```
## effective sample size of beta1: 694.6009
##  effective sample size of beta2: 612.3535
##  effective sample size of beta3: 618.073
##  effective sample size of beta4: 832.7881
```

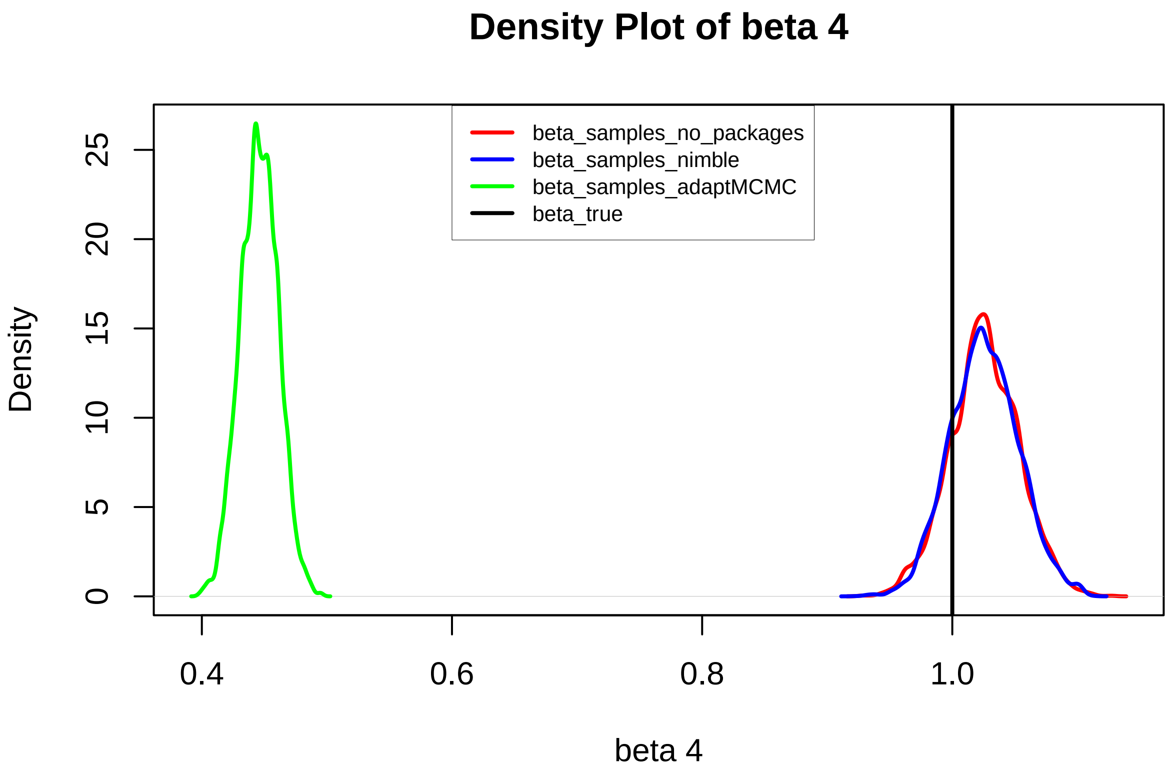
5. (5 points) Compare the posterior densities obtained from Problems 2-4. Especially, you should overlap the density of each β_j for all different methods. Draw the true β_j as a vertical line in each posterior density. Discuss whether all MCMC algorithms can recover the true β_j well.

```r
for (i in 1:4){
  x_lim = range(c(
    density(beta_samples[, i])$x,
    density(beta_samples_nimble[, i])$x,
    density(beta_samples_adaptMCMC[, i])$x
  ))
  y_lim = range(c(
    density(beta_samples[, i])$y,
    density(beta_samples_nimble[, i])$y,
    density(beta_samples_adaptMCMC[, i])$y
  ))
  # Initial plot settings
  plot(
    density(beta_samples[, i]),
    main = paste("Density Plot of beta", i),
    xlab = paste("beta", i),
    ylab = "Density",
    lwd = 2,
    col = "red",
    xlim = x_lim,
    ylim = y_lim
  )

  lines(density(beta_samples_nimble[, i]), lwd = 2, col = "blue")
  lines(density(beta_samples_adaptMCMC[, i]), lwd = 2, col = "green")
  abline(v = beta_true[i], lwd = 2, col = "black")

  legend_location_x = c(0.3, -0.45, 0.01, 0.6)
  legend(x = legend_location_x[i], y = max(y_lim)+1,
    legend = c("beta_samples_no_packages", "beta_samples_nimble",
               "beta_samples_adaptMCMC","beta_true"),
    col = c("red", "blue", "green","black"),
    lwd = 2, cex = 0.7, box.lwd = 0, bg = "transparent",
  )
  cat('\n','\n','\n')
}
```


Density Plot of beta 1
beta_samples_no_packages
beta_samples_nimble
beta_samples_adaptMCMC
beta_true
Density
0
5
10
15
20
25
0.2
0.3
0.4
0.5
beta 1


Density Plot of beta 2
beta_samples_no_packages
beta_samples_nimble
beta_samples_adaptMCMC
beta_true
Density
0
5
10
15
20
25
−0.6
−0.5
−0.4
−0.3
−0.2
beta 2





The beta obtained without using the package and the beta obtained using the nimble package estimated beta_true well. But the beta obtained using the adaptMCMC package did not estimate beta_true due to model misspecification.

The reason for this is that : when using the adaptMCMC package, Y is considered to be data generated from $Binomial(20, \mu(X))$, (even though we generated Y from $Binomial(10, \mu(X))$). As a result, all the betas obtained using the adaptMCMC package were estimated to be close to half of beta_true.