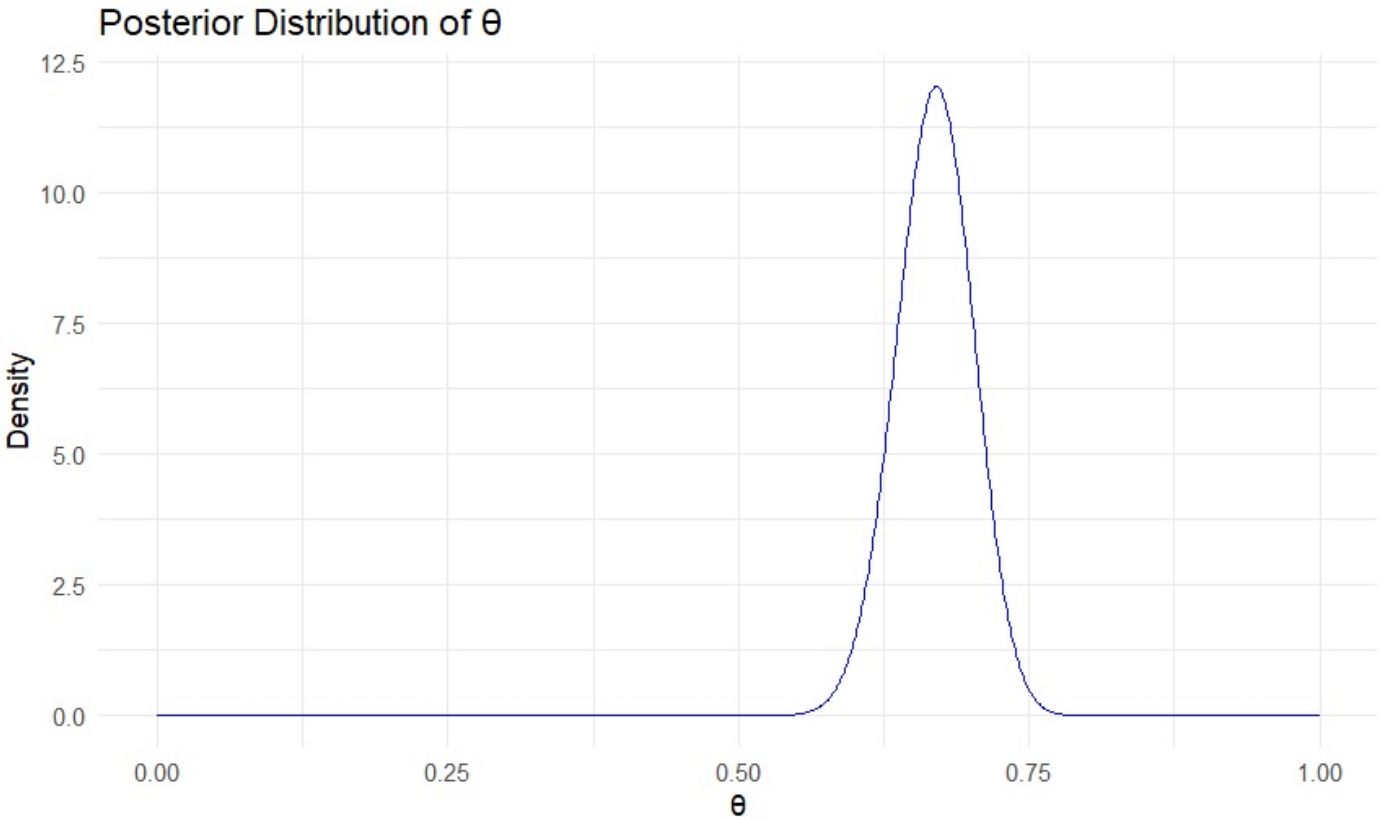
Assignment -3

Part 1: Estimating the posterior distribution using different computational methods

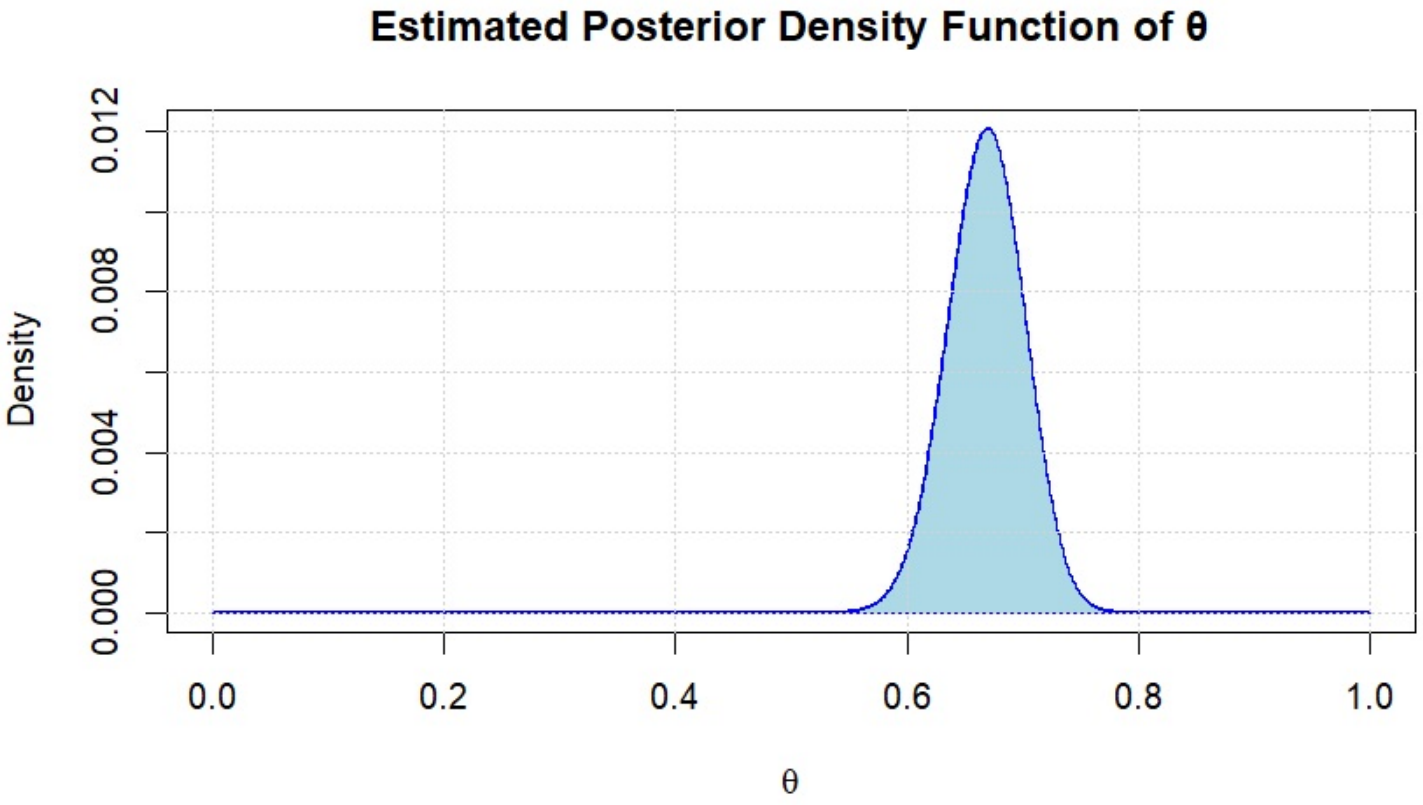
1.1

```
library(ggplot2)

# Define the shape parameters for the Beta distribution
alpha_post <- 135
beta_post <- 67

# Generate theta values from 0 to 1
theta_values <- seq(0, 1, length.out = 1000)

# Calculate the density of the Beta distribution (Analytical)
posterior_pdf <- dbeta(theta_values, alpha_post, beta_post)

# Create a data frame for plotting
data <- data.frame(theta = theta_values, density = posterior_pdf)

# Plot the posterior distribution using ggplot2
ggplot(data, aes(x = theta, y = density)) +
  geom_line(color = 'blue') +
  labs(title = "Posterior Distribution of θ", x = "θ", y = "Density") +
  theme_minimal()
```



1.2

```
# Given data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
alpha_prior <- 1
beta_prior <- 1

# Grid approximation
theta_grid <- seq(0, 1, length.out = 1000)
likelihood <- sapply(theta_grid, function(theta) {
  prod(dbinom(data, size = n, prob = theta))
})
prior <- dbeta(theta_grid, alpha_prior, beta_prior)
unnormalized_posterior <- likelihood * prior
posterior <- unnormalized_posterior / sum(unnormalized_posterior)

# Plotting the estimated posterior density function
plot(theta_grid, posterior, type = "l", col = "blue", lwd = 2,
     main = "Estimated Posterior Density Function of θ",
     xlab = expression(theta), ylab = "Density")
polygon(theta_grid, posterior, col = "lightblue", border = "blue")
grid()
```



1.3

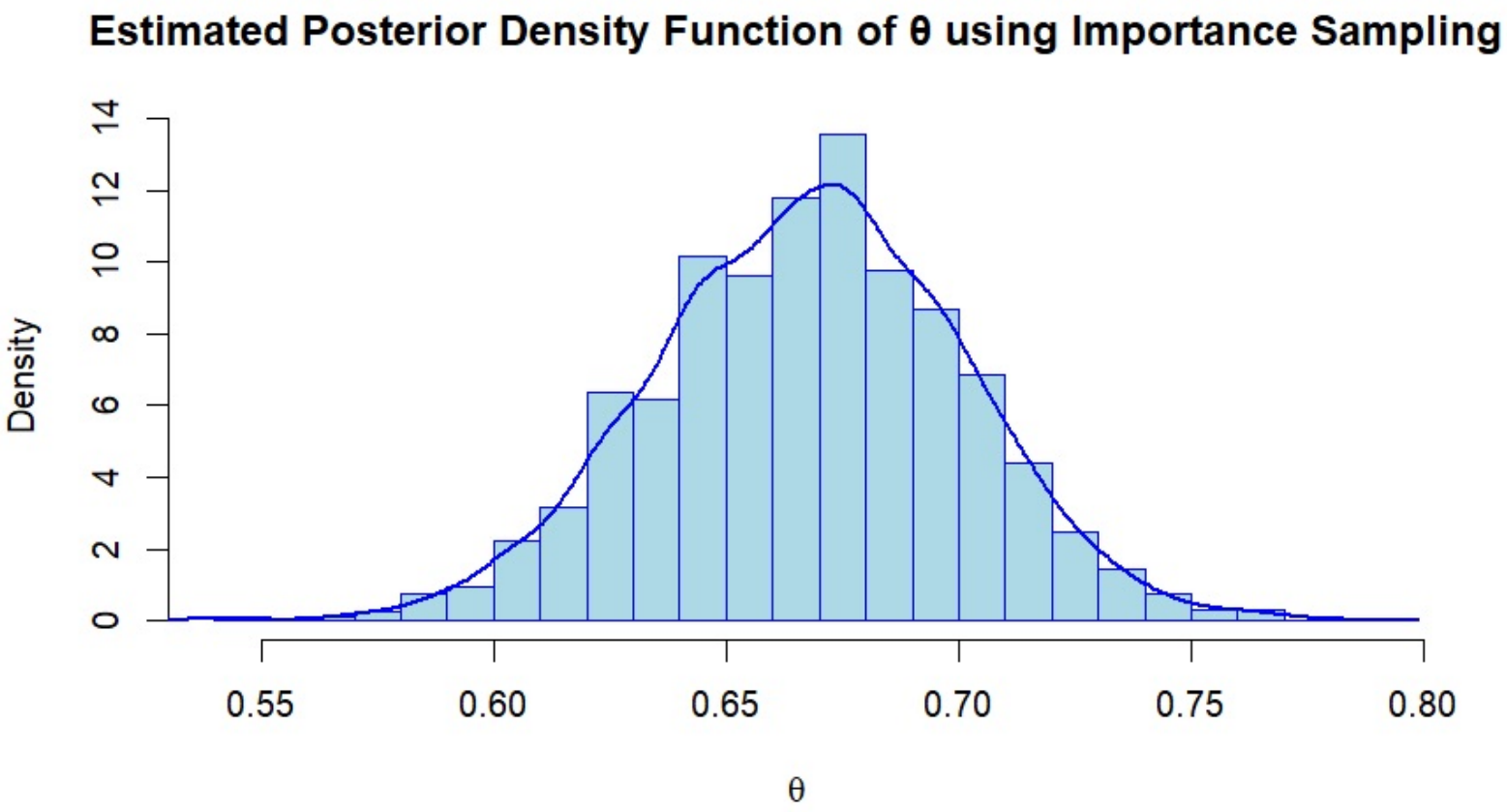
```
# Given data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
alpha_prior <- 1
beta_prior <- 1

# Monte Carlo parameters
num_samples <- 10000

# Step 1: Draw samples from the prior distribution
theta_samples <- rbeta(num_samples, alpha_prior, beta_prior)

# Step 2: Calculate the likelihood for each sample
likelihood_samples <- sapply(theta_samples, function(theta) {
  prod(dbinom(data, size = n, prob = theta))
})

# Step 3: Estimate the marginal likelihood
marginal_likelihood <- mean(likelihood_samples)
marginal_likelihood
```

Output = 1.420804e-10

1.4

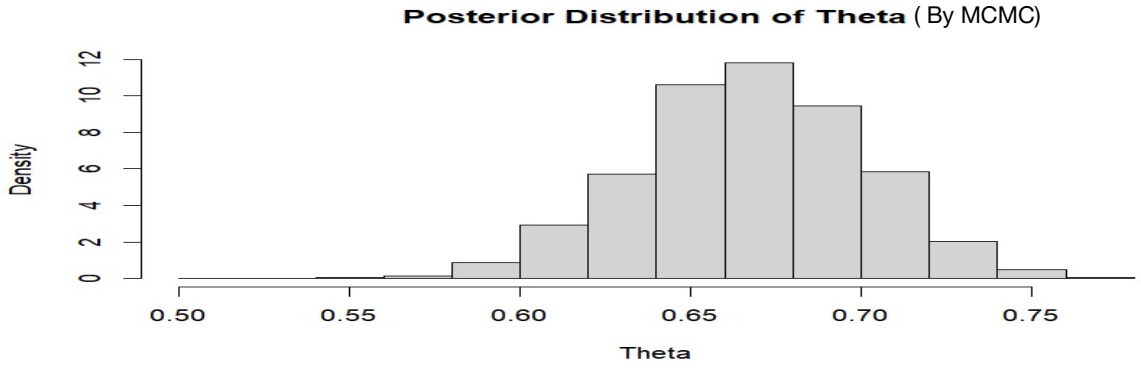
```
# Given data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
alpha_prior <- 1
beta_prior <- 1

# Importance sampling parameters
N <- 10000

# Step 1: Draw N samples from the proposal density function q(θ)
# Here we use a uniform proposal density between 0 and 1
proposal_samples <- runif(N, 0, 1)

# Step 2: Compute the likelihood, prior, and proposal density for each sample
likelihood <- sapply(proposal_samples, function(theta) {
  prod(dbinom(data, size = n, prob = theta))
})
prior <- dbeta(proposal_samples, alpha_prior, beta_prior)
proposal_density <- dunif(proposal_samples, 0, 1)

# Step 3: Compute the importance weights
weights <- likelihood * prior / proposal_density

# Step 4: Normalize the weights
normalized_weights <- weights / sum(weights)

# Step 5: Resample from the proposal samples using the normalized weights
posterior_samples <- sample(proposal_samples, size = N/4, replace = TRUE, prob =
normalized_weights)

# Plotting the estimated posterior density function
hist(posterior_samples, breaks = 30, probability = TRUE, col = "lightblue", border =
"blue",
     main = "Estimated Posterior Density Function of θ using Importance Sampling",
     xlab = expression(theta))
# Given data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
alpha_prior <- 1
beta_prior <- 1

# Importance sampling parameters
N <- 10000

# Step 1: Draw N samples from the proposal density function q(θ)
#  assume uniform proposal density between 0 and 1
proposal_samples <- runif(N, 0, 1)

# Step 2: Compute the likelihood, prior, and proposal density for each sample
likelihood <- sapply(proposal_samples, function(theta) {
  prod(dbinom(data, size = n, prob = theta))
})
prior <- dbeta(proposal_samples, alpha_prior, beta_prior)
proposal_density <- dunif(proposal_samples, 0, 1)

# Step 3: Compute the importance weights
weights <- likelihood * prior / proposal_density

# Step 4: Normalize the weights
normalized_weights <- weights / sum(weights)

# Step 5: Resample from the proposal samples using the normalized weights
posterior_samples <- sample(proposal_samples, size = N/4, replace = TRUE, prob =
normalized_weights)
```

```
# Plotting the estimated posterior density function
hist(posterior_samples, breaks = 30, probability = TRUE, col = "lightblue", border =
"blue",
     main = "Estimated Posterior Density Function of θ using Importance Sampling",
     xlab = expression(theta))
lines(density(posterior_samples), col = "blue", lwd = 2)
```



1.5

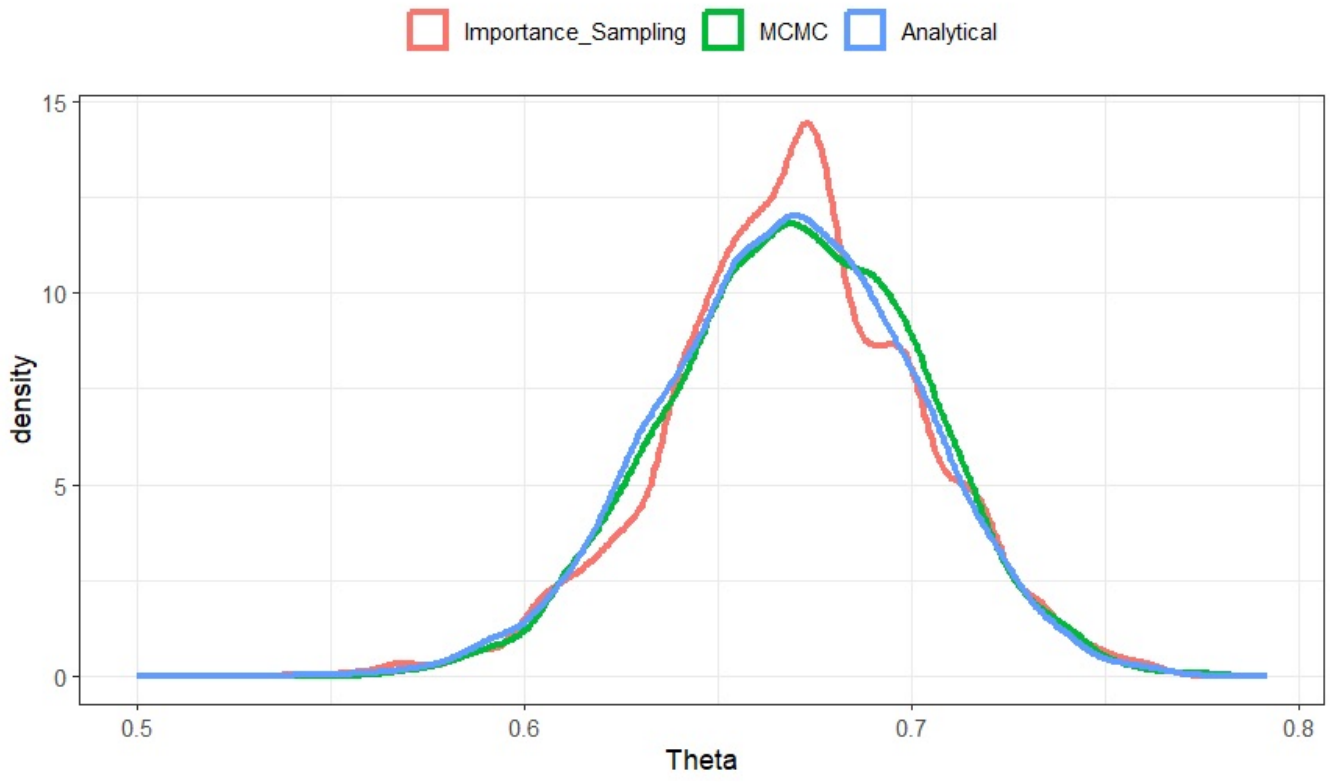
```
# Define the likelihood function for a binomial distribution
likelihood <- function(theta, y) {
  return(prod(dbinom(y, size = 20, prob = theta)))
}

# Define the prior function for a beta distribution
prior <- function(theta) {
  return(dbeta(theta, shape1 = 1, shape2 = 1))
}

# Define the Metropolis-Hastings algorithm
metropolis_hastings <- function(data, iterations, theta_init, proposal_sd) {
  theta <- rep(NA, iterations)
  theta[1] <- theta_init
  accept <- 0

  for (i in 2:iterations) {
    # Propose new theta value
    theta_proposal <- rnorm(1, theta[i-1], proposal_sd)
    #calculate hasting ratio
    H= likelihood(theta_proposal, data) * prior(theta_proposal) /
      (likelihood(theta[i-1], data) * prior(theta[i-1]))
    # Calculate acceptance probability
    alpha <- min(1, H)

    if (is.nan(alpha)) {
      alpha <- 0
    }
    # Accept or reject proposal
    if (runif(1) < alpha) {
      theta[i] <- theta_proposal
      accept <- accept + 1
    } else {
      theta[i] <- theta[i-1]
    }
  }

  acceptance_rate <- accept / iterations
  return(list(theta = theta, acceptance_rate = acceptance_rate))
}

# Data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)

# Set initial values
iterations <- 10000
theta_init <- 0.5
proposal_sd <- 0.1

# Run Metropolis-Hastings algorithm
result <- metropolis_hastings(data, iterations, theta_init, proposal_sd)

# Plot posterior distribution
hist(result$theta, main = "Posterior Distribution of Theta", xlab = "Theta", prob = TRUE)
```



1.6

```
library(ggplot2)
library(reshape2)
# Given data
data <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
alpha_prior <- 1
beta_prior <- 1

# Importance sampling parameters
N <- 10000

# Step 1: Draw N samples from the proposal density function q(θ)
# Here we use a uniform proposal density between 0 and 1
proposal_samples <- runif(N, 0, 1)

# Step 2: Compute the likelihood, prior, and proposal density for each sample
likelihood <- sapply(proposal_samples, function(theta) {
  prod(dbinom(data, size = n, prob = theta))
})
prior <- dbeta(proposal_samples, alpha_prior, beta_prior)
proposal_density <- dunif(proposal_samples, 0, 1)

# Step 3: Compute the importance weights
weights <- likelihood * prior / proposal_density

# Step 4: Normalize the weights
normalized_weights <- weights / sum(weights)

# Step 5: Resample from the proposal samples using the normalized weights
posterior_samples_importance <- sample(proposal_samples, size = N/4, replace = TRUE, prob
= normalized_weights)

# Define the likelihood function for a binomial distribution
likelihood <- function(theta, y) {
  return(prod(dbinom(y, size = 20, prob = theta)))
}

# Define the prior function for a beta distribution
prior <- function(theta) {
  return(dbeta(theta, shape1 = 1, shape2 = 1))
}

# Define the Metropolis-Hastings algorithm
metropolis_hastings <- function(data, iterations, theta_init, proposal_sd) {
  theta <- rep(NA, iterations)
  theta[1] <- theta_init
  accept <- 0

  for (i in 2:iterations) {
    # Propose new theta value
    theta_proposal <- rnorm(1, theta[i-1], proposal_sd)
    #calculate hasting ratio
    H= likelihood(theta_proposal, data) * prior(theta_proposal) /
      (likelihood(theta[i-1], data) * prior(theta[i-1]))
    # Calculate acceptance probability
    alpha <- min(1, H)

    if (is.nan(alpha)) {
      alpha <- 0
    }
    # Accept or reject proposal
    if (runif(1) < alpha) {
      theta[i] <- theta_proposal
```

```
      accept <- accept + 1
    } else {
      theta[i] <- theta[i-1]
    }
  }

  acceptance_rate <- accept / iterations
  return(list(theta = theta, acceptance_rate = acceptance_rate))
}

# Set initial values
iterations <- 10000
theta_init <- 0.5
proposal_sd <- 0.1

# Run Metropolis-Hastings algorithm
result_mcmc <- metropolis_hastings(data, iterations, theta_init, proposal_sd)

# Generate samples from the analytical posterior
analytical_posterior <- rbeta(5000, 135, 67)

# Combine the posterior distributions into a data frame
posteriors <- data.frame(
  Importance_Sampling = posterior_samples_importance,
  MCMC = result_mcmc$theta,
  Analytical = analytical_posterior
)

# Melt the data frame to long format for plotting
posteriors_melted <- melt(posteriors)

# Plotting
ggplot(posteriors_melted, aes(x = value, colour = variable)) +
  geom_density(size = 1.2) +
  theme_bw() +
  xlab("Theta") +
  theme(
    legend.title = element_blank(),
    legend.position = "top"
  )
```



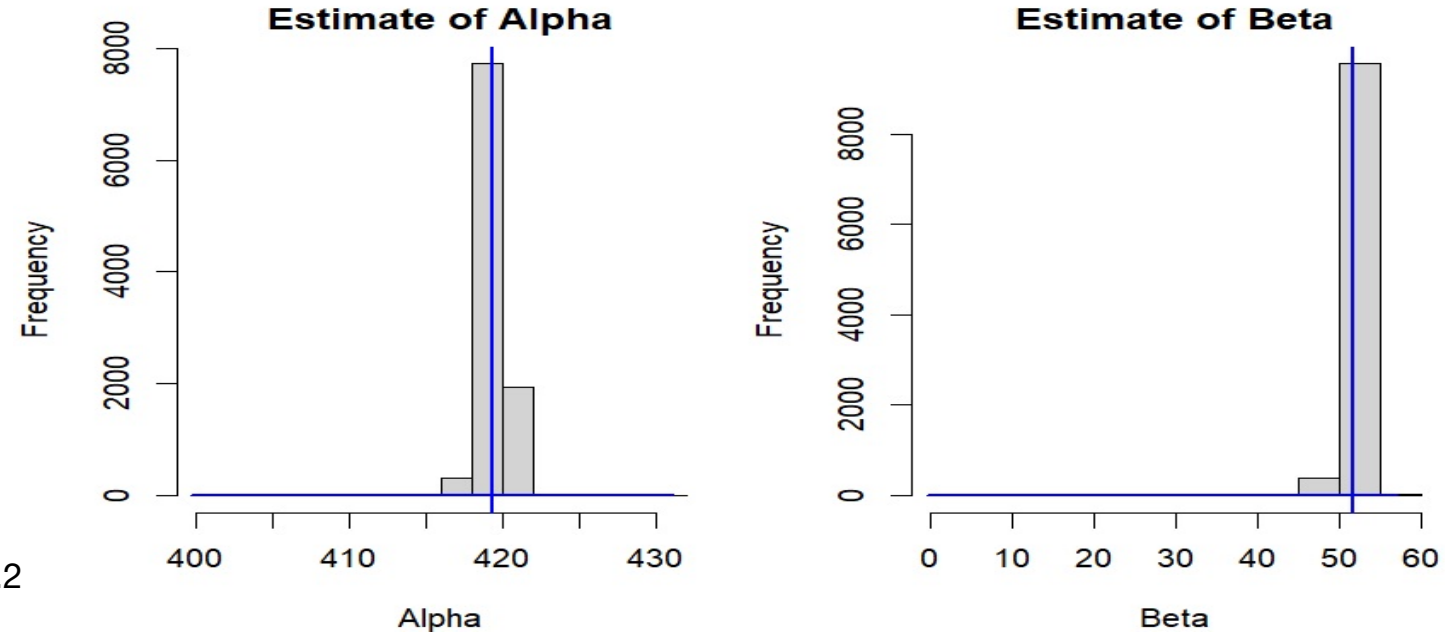
Part 2: Writing your own sampler for Bayesian inference

2.5

```
# Load necessary library
library(truncnorm)

# Likelihood function
likelihood <- function(alpha, beta, type, RT) {
  mu <- alpha + beta * as.numeric(type == "non-word")
  return(sum(dnorm(RT, mu, 30, log = TRUE)))
}

# Prior function for alpha
prior_alpha <- function(alpha) {
  return(dnorm(alpha, 400, 50, log = TRUE))
}

# Prior function for beta (truncated normal)
prior_beta <- function(beta) {
  d <- dtruncnorm(beta, 0, Inf, 0, 50)
  return(log(d))
}

# Metropolis-Hastings algorithm
metropolis_hastings <- function(data, iterations, alpha_init, beta_init, proposal_sd) {
  alpha_chain <- numeric(iterations)
  beta_chain <- numeric(iterations)
  alpha_chain[1] <- alpha_init
  beta_chain[1] <- beta_init
  accept <- 0

  for (i in 2:iterations) {
    # Propose new alpha and beta values
    alpha_proposal <- rnorm(1, alpha_chain[i-1], proposal_sd)
    beta_proposal <- rtruncnorm(1, mean = beta_chain[i-1], sd = proposal_sd, a = 0)

    # Compute posterior density on log scale
    post_new <- likelihood(alpha_proposal, beta_proposal, data$type, data$RT) +
      prior_alpha(alpha_proposal) +
      prior_beta(beta_proposal)

    post_prev <- likelihood(alpha_chain[i-1], beta_chain[i-1], data$type, data$RT) +
      prior_alpha(alpha_chain[i-1]) +
      prior_beta(beta_chain[i-1])

    # Compute Hastings ratio
    Hastings_ratio <- exp(post_new - post_prev)
    p_str <- min(Hastings_ratio, 1)  # probability of acceptance

    if (is.nan(p_str)) {
      p_str <- 0
    }
    # Accept or reject proposal
    if (!is.na(p_str) && p_str > runif(1)) {
      alpha_chain[i] <- alpha_proposal
      beta_chain[i] <- beta_proposal
      accept <- accept + 1
    } else {
      alpha_chain[i] <- alpha_chain[i-1]
      beta_chain[i] <- beta_chain[i-1]
    }
  }

  acceptance_rate <- accept / iterations
  return(list(alpha = alpha_chain, beta = beta_chain, acceptance_rate = acceptance_rate))
}

# Load data
```

```
dat <-
read.table("https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/Data/word-
recognition-times.csv", sep = ",", header = TRUE)[,-1]

# Set initial values
iterations <- 10000
alpha_init <- 400
beta_init <- 0  # We're starting from a small positive value
proposal_sd <- 5

# Run Metropolis-Hastings algorithm
result_mcmc <- metropolis_hastings(dat, iterations, alpha_init, beta_init, proposal_sd)

# Plot histograms of alpha and beta chains
par(mfrow = c(1, 2))
hist(result_mcmc$alpha, main = "Estimate of Alpha", xlab = "Alpha")
abline(v = mean(result_mcmc$alpha), col = "blue", lwd = 2)
lines(density(result_mcmc$alpha), col = "blue", lwd = 2)
hist(result_mcmc$beta, main = "Estimate of Beta", xlab = "Beta")
abline(v = mean(result_mcmc$beta), col = "blue", lwd = 2)
lines(density(result_mcmc$beta), col = "blue", lwd = 2)
par(mfrow = c(1, 1))

# Compute 95% credible interval for alpha
alpha_cred_interval <- quantile(result_mcmc$alpha, probs = c(0.025, 0.975))

# Compute 95% credible interval for beta
beta_cred_interval <- quantile(result_mcmc$beta, probs = c(0.025, 0.975))

# Print the credible intervals and means
print(paste("95% credible interval for alpha:", round(alpha_cred_interval[1], 2), "-",
round(alpha_cred_interval[2], 2)))
print(paste("Mean of alpha:", round(mean(result_mcmc$alpha), 2)))
print(paste("95% credible interval for beta:", round(beta_cred_interval[1], 2), "-",
round(beta_cred_interval[2], 2)))
print(paste("Mean of beta:", round(mean(result_mcmc$beta), 2)))
```

2.5.1



2.5.2

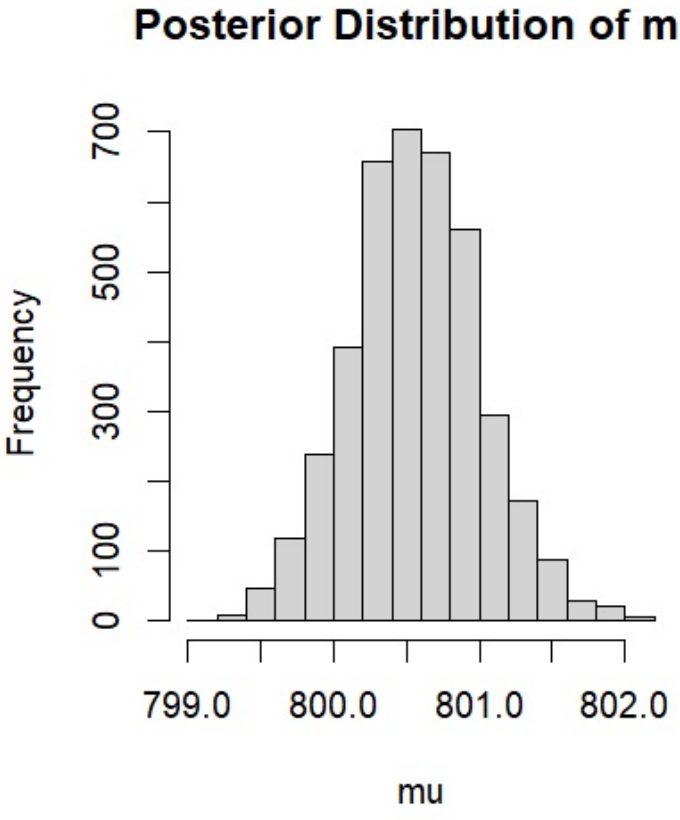
[1] "95% credible interval for alpha: 417.97 - 420.88"
[1] "Mean of alpha: 419.33"
[1] "95% credible interval for beta: 49.6 - 53.64"
[1] "Mean of beta: 51.64"

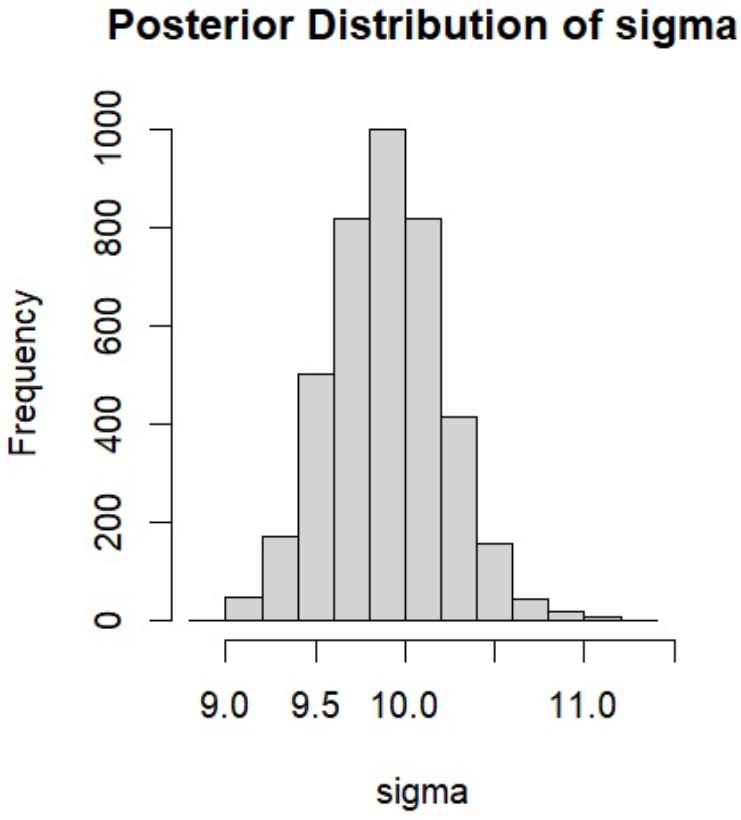
Part 3: Hamiltonian Monte Carlo sampler

3.1

```
true_mu <- 800
true_var <- 100 # sigma^2
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))

# Gradient function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}

# Potential energy function
V <- function(mu, sigma, y, n, m, s, a, b) {
  nlpd <- -(sum(dnorm(y, mu, sigma, log = TRUE)) + dnorm(mu, m, s, log = TRUE) +
dnorm(sigma, a, b, log = TRUE))
  return(nlpd)
}

# HMC sampler
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0

  # Initialization of Markov chain
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]

  # Evolution of Markov chain
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i]) # Current position of the particle
    p <- rnorm(length(q), 0, 1) # Generate random momentum at the current position
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b) # Current potential
energy
    current_T <- sum(current_p^2) / 2 # Current kinetic energy

    # Take L leapfrog steps
    for (l in 1:L) {
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      # Change in position in 'step' time
      q <- q + step * p
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
    }

    proposed_q <- q
    proposed_p <- p
    proposed_V <- V(proposed_q[1], proposed_q[2], y, n, m, s, a, b) # Proposed potential
energy
    proposed_T <- sum(proposed_p^2) / 2 # Proposed kinetic energy
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))

    # Accept/reject the proposed position q
    if (accept.prob > runif(1, 0, 1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
```

```
    }
  }

  # Discard burn-in samples
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}

# Run HMC sampler
df.posterior <- HMC(y = y, n = length(y), m = 1000, s = 20, a = 10, b = 2,
                    step = 0.02, L = 12, initial_q = c(1000, 11),
                    nsamp = 6000, nburn = 2000)

par(mfrow = c(1, 2))
hist(df.posterior$mu_chain, main = "Posterior Distribution of mu", xlab = "mu")
hist(df.posterior$sigma_chain, main = "Posterior Distribution of sigma", xlab = "sigma")
par(mfrow = c(1, 2))
```

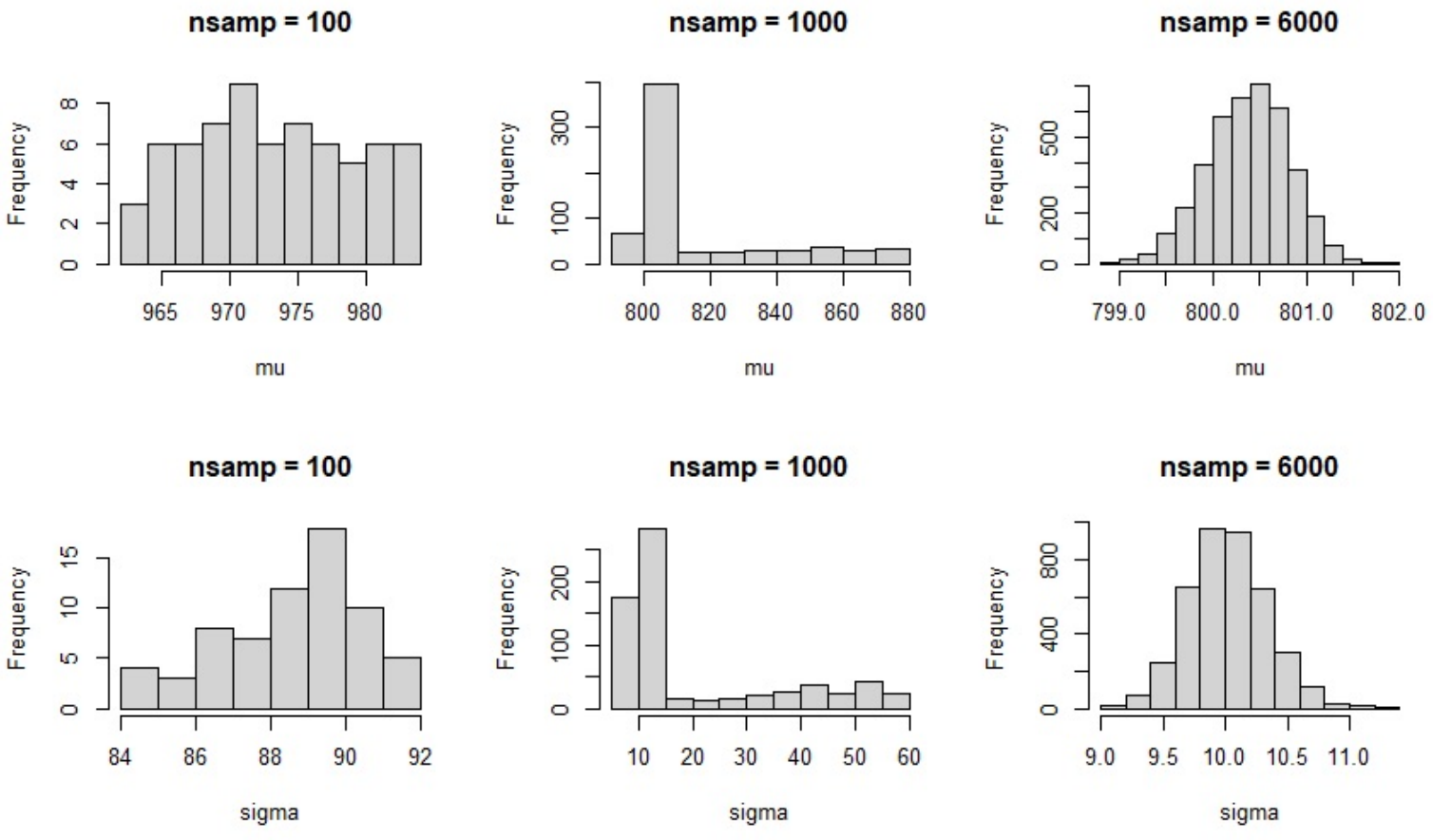




3.2

```
true_mu <- 800
true_var <- 100 # sigma^2
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))

# Gradient function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}

# Potential energy function
V <- function(mu, sigma, y, n, m, s, a, b) {
  nlpd <- -(sum(dnorm(y, mu, sigma, log = TRUE)) + dnorm(mu, m, s, log = TRUE) +
dnorm(sigma, a, b, log = TRUE))
  return(nlpd)
}

# HMC sampler
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0

  # Initialization of Markov chain
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]

  # Evolution of Markov chain
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i]) # Current position of the particle
    p <- rnorm(length(q), 0, 1) # Generate random momentum at the current position
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b) # Current potential
energy
    current_T <- sum(current_p^2) / 2 # Current kinetic energy

    # Take L leapfrog steps
    for (l in 1:L) {
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      # Change in position in 'step' time
      q <- q + step * p
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
    }

    proposed_q <- q
    proposed_p <- p
    proposed_V <- V(proposed_q[1], proposed_q[2], y, n, m, s, a, b) # Proposed potential
energy
    proposed_T <- sum(proposed_p^2) / 2 # Proposed kinetic energy
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))

    # Accept/reject the proposed position q
    if (accept.prob > runif(1, 0, 1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
```

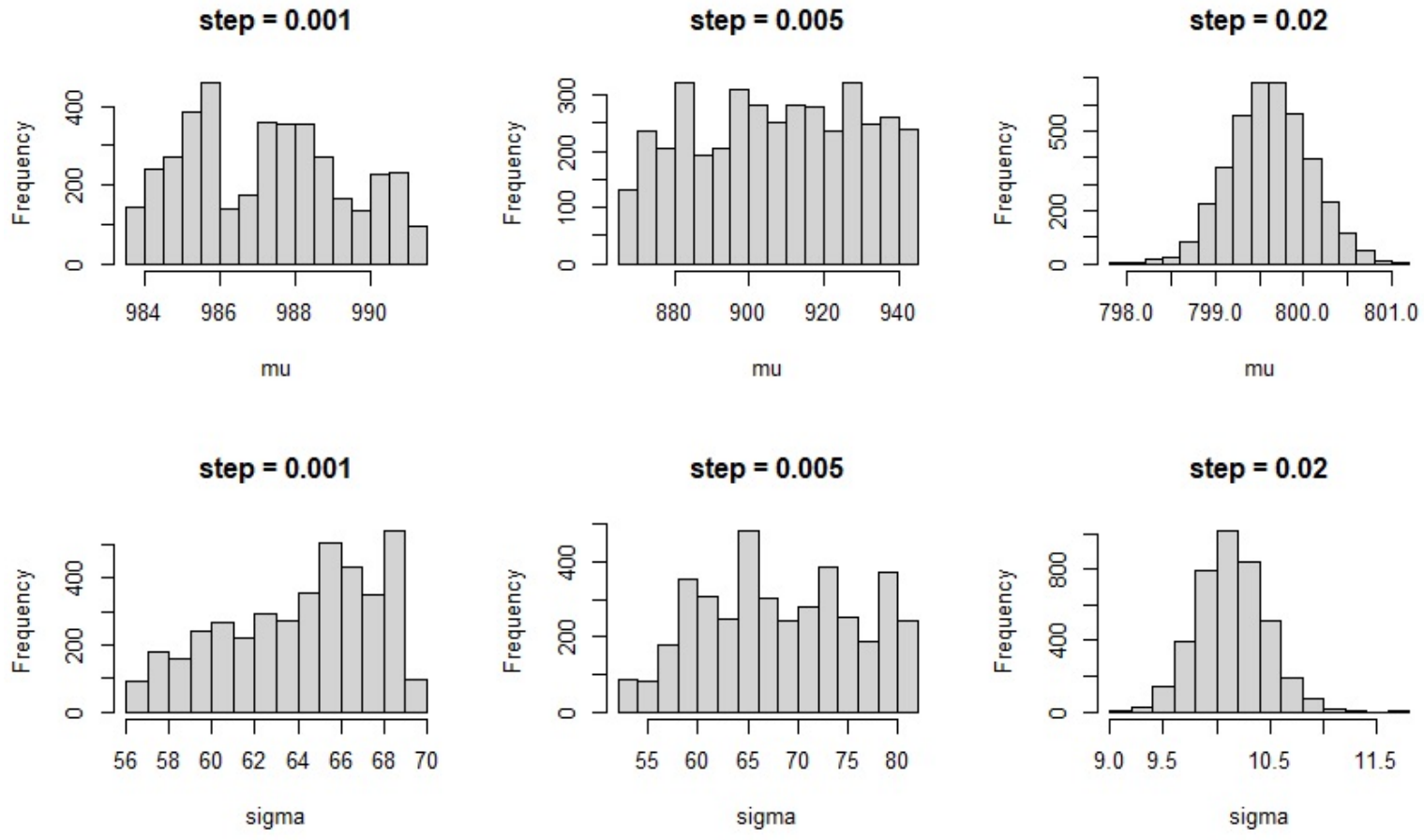
```
  }

  # Discard burn-in samples
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}


# Define a function to run HMC sampler for different values of nsamp
run_HMC <- function(nsamp_value) {
  nburn <- nsamp_value / 3

  df.posterior <- HMC(y = y, n = length(y), m = 1000, s = 20, a = 10, b = 2,
                      step = 0.02, L = 12, initial_q = c(1000, 11),
                      nsamp = nsamp_value, nburn = nburn)

  return(df.posterior)
}

# Run HMC sampler for different nsamp values
df_posterior_100 <- run_HMC(100)
df_posterior_1000 <- run_HMC(1000)
df_posterior_6000 <- run_HMC(6000)

# Plot the posteriors for mu and sigma
par(mfrow = c(2, 3))
hist(df_posterior_100$mu_chain, main = "nsamp = 100", xlab = "mu")
hist(df_posterior_1000$mu_chain, main = "nsamp = 1000", xlab = "mu")
hist(df_posterior_6000$mu_chain, main = "nsamp = 6000", xlab = "mu")
hist(df_posterior_100$sigma_chain, main = "nsamp = 100", xlab = "sigma")
hist(df_posterior_1000$sigma_chain, main = "nsamp = 1000", xlab = "sigma")
hist(df_posterior_6000$sigma_chain, main = "nsamp = 6000", xlab = "sigma")
par(mfrow = c(1, 1))
```



3.3

```
true_mu <- 800
true_var <- 100 # sigma^2
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))

# Gradient function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}

# Potential energy function
V <- function(mu, sigma, y, n, m, s, a, b) {
  nlpd <- -(sum(dnorm(y, mu, sigma, log = TRUE)) + dnorm(mu, m, s, log = TRUE) +
dnorm(sigma, a, b, log = TRUE))
  return(nlpd)
}

# HMC sampler
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0

  # Initialization of Markov chain
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]

  # Evolution of Markov chain
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i]) # Current position of the particle
    p <- rnorm(length(q), 0, 1) # Generate random momentum at the current position
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b) # Current potential
energy
    current_T <- sum(current_p^2) / 2 # Current kinetic energy

    # Take L leapfrog steps
    for (l in 1:L) {
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      # Change in position in 'step' time
      q <- q + step * p
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
    }

    proposed_q <- q
    proposed_p <- p
    proposed_V <- V(proposed_q[1], proposed_q[2], y, n, m, s, a, b) # Proposed potential
energy
    proposed_T <- sum(proposed_p^2) / 2 # Proposed kinetic energy
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))

    # Accept/reject the proposed position q
    if (accept.prob > runif(1, 0, 1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
```

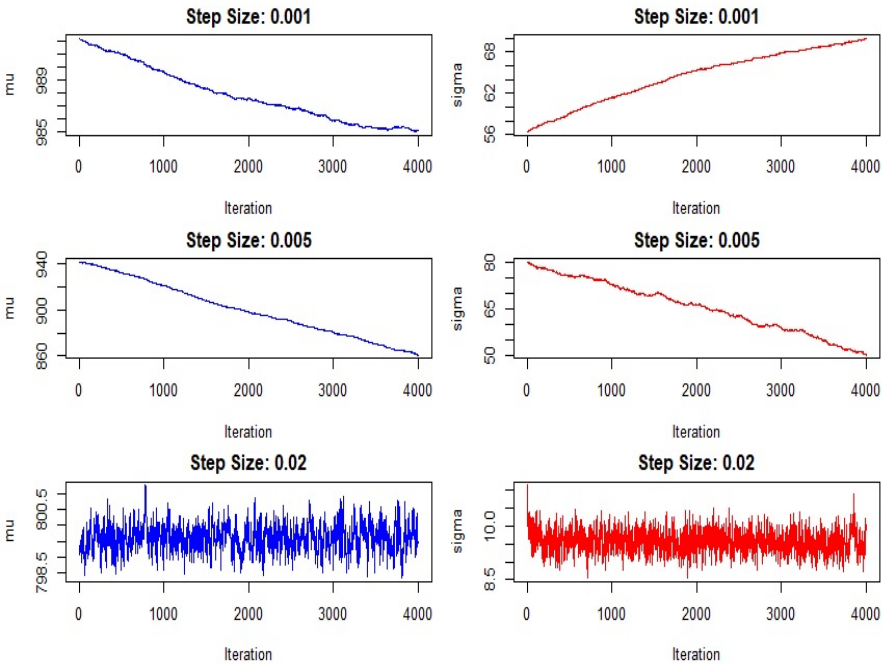
```
  }

  # Discard burn-in samples
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}


run_HMC <- function(step_value) {
  df.posterior <- HMC(y = y, n = length(y), m = 1000, s = 20, a = 10, b = 2,
                      step = step_value, L = 12, initial_q = c(1000, 11),
                      nsamp = 6000, nburn = 2000)

  return(df.posterior)
}

# Run HMC sampler for different step values
df_posterior_0.001 <- run_HMC(0.001)
df_posterior_0.005 <- run_HMC(0.005)
df_posterior_0.02 <- run_HMC(0.02)

# Plot the posteriors for mu and sigma
par(mfrow = c(2, 3))
hist(df_posterior_0.001$mu_chain, main = "step = 0.001", xlab = "mu")
hist(df_posterior_0.005$mu_chain, main = "step = 0.005", xlab = "mu")
hist(df_posterior_0.02$mu_chain, main = "step = 0.02", xlab = "mu")
hist(df_posterior_0.001$sigma_chain, main = "step = 0.001", xlab = "sigma")
hist(df_posterior_0.005$sigma_chain, main = "step = 0.005", xlab = "sigma")
hist(df_posterior_0.02$sigma_chain, main = "step = 0.02", xlab = "sigma")
par(mfrow = c(1, 1))
```



3.4

```
true_mu <- 800
true_var <- 100 # sigma^2
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))

# Gradient function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}

# Potential energy function
V <- function(mu, sigma, y, n, m, s, a, b) {
  nlpd <- -(sum(dnorm(y, mu, sigma, log = TRUE)) + dnorm(mu, m, s, log = TRUE) +
dnorm(sigma, a, b, log = TRUE))
  return(nlpd)
}

# HMC sampler
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0

  # Initialization of Markov chain
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]

  # Evolution of Markov chain
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i]) # Current position of the particle
    p <- rnorm(length(q), 0, 1) # Generate random momentum at the current position
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b) # Current potential
energy
    current_T <- sum(current_p^2) / 2 # Current kinetic energy

    # Take L leapfrog steps
    for (l in 1:L) {
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      # Change in position in 'step' time
      q <- q + step * p
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
    }

    proposed_q <- q
    proposed_p <- p
    proposed_V <- V(proposed_q[1], proposed_q[2], y, n, m, s, a, b) # Proposed potential
energy
    proposed_T <- sum(proposed_p^2) / 2 # Proposed kinetic energy
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))

    # Accept/reject the proposed position q
    if (accept.prob > runif(1, 0, 1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
```

```
  }

  # Discard burn-in samples
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}



# Define a function to run HMC sampler with different step sizes and plot trace plots
run_and_plot_hmc <- function(step_sizes) {
  par(mfrow = c(length(step_sizes), 2), mar = c(4, 4, 2, 1))  # Set up multiple plots

  for (i in 1:length(step_sizes)) {
    step <- step_sizes[i]
    df.posterior <- HMC(y = y, n = length(y), m = 1000, s = 20, a = 10, b = 2,
                        step = step, L = 12, initial_q = c(1000, 11),
                        nsamp = 6000, nburn = 2000)

    # Plot trace plots for mu and sigma chains
    plot(df.posterior$mu_chain, type = "l", main = paste("Step Size:", step),
         xlab = "Iteration", ylab = "mu", col = "blue", ylim =
range(df.posterior$mu_chain))
    plot(df.posterior$sigma_chain, type = "l", main = paste("Step Size:", step),
         xlab = "Iteration", ylab = "sigma", col = "red", ylim =
range(df.posterior$sigma_chain))
  }

  par(mfrow = c(1, 1))  # Reset plotting settings
}

# Define different step sizes
step_sizes <- c(0.001, 0.005, 0.02)

# Run HMC sampler with different step sizes and plot trace plots
run_and_plot_hmc(step_sizes)

# Plot the sigma chain
plot(sigma_chain, type = "l", col = "green", xlab = "Iteration", ylab = "sigma", main =
"Trace Plot of sigma Chain")

# Add a horizontal line at the true value of sigma for reference
abline(h = true_var, col = "red")
```



**

Upon examining the mu and sigma chains for different step-size values, we observed that larger step-sizes improved the stationarity of the posterior density graphs for mu and sigma. Conversely, smaller step-sizes resulted in more pronounced upward or downward trends in these graphs

3.5

```
true_mu <- 800
true_var <- 100 # sigma^2
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))

# Gradient function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}

# Potential energy function
V <- function(mu, sigma, y, n, m, s, a, b) {
  nlpd <- -(sum(dnorm(y, mu, sigma, log = TRUE)) + dnorm(mu, m, s, log = TRUE) +
dnorm(sigma, a, b, log = TRUE))
  return(nlpd)
}

# HMC sampler
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0

  # Initialization of Markov chain
  mu_chain[1] <- in'tial_q[1]
  sigma_chain[1] <- initial_q[2]

  # Evolution of Markov chain
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i]) # Current position of the particle
    p <- rnorm(length(q), 0, 1) # Generate random momentum at the current position
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b) # Current potential
energy
    current_T <- sum(current_p^2) / 2 # Current kinetic energy

    # Take L leapfrog steps
    for (l in 1:L) {
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      # Change in position in 'step' time
      q <- q + step * p
      # Change in momentum in 'step/2' time
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
    }

    proposed_q <- q
    proposed_p <- p
    proposed_V <- V(proposed_q[1], proposed_q[2], y, n, m, s, a, b) # Proposed potential
energy
    proposed_T <- sum(proposed_p^2) / 2 # Proposed kinetic energy
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))

    # Accept/reject the proposed position q
    if (accept.prob > runif(1, 0, 1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
```

```
  }

  # Discard burn-in samples
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}

run_HMC <- function(m_value, s_value) {
  df.posterior <- HMC(y = y, n = length(y), m = m_value, s = s_value, a = 10, b = 2,
                      step = 0.02, L = 12, initial_q = c(1000, 11),
                      nsamp = 6000, nburn = 2000)

  return(df.posterior)
}

# Run HMC sampler for different prior specifications of mu
df_posterior_mu_400_5 <- run_HMC(400, 5)
df_posterior_mu_400_20 <- run_HMC(400, 20)
df_posterior_mu_1000_5 <- run_HMC(1000, 5)
df_posterior_mu_1000_20 <- run_HMC(1000, 20)
df_posterior_mu_1000_100 <- run_HMC(1000, 100)

# Plot the posteriors for mu
par(mfrow = c(3, 2))
hist(df_posterior_mu_400_5$mu_chain, main = "μ ~ Normal(m = 400, s = 5)", xlab = "mu")
hist(df_posterior_mu_400_20$mu_chain, main = "μ ~ Normal(m = 400, s = 20)", xlab = "mu")
hist(df_posterior_mu_1000_5$mu_chain, main = "μ ~ Normal(m = 1000, s = 5)", xlab = "mu")
hist(df_posterior_mu_1000_20$mu_chain, main = "μ ~ Normal(m = 1000, s = 20)", xlab = "mu")
hist(df_posterior_mu_1000_100$mu_chain, main = "μ ~ Normal(m = 1000, s = 100)", xlab =
"mu")
par(mfrow = c(1, 1))
```