CGS698C, Assignment 03

Himanshu Yadav

2024-06-21

Part 1: Estimating the posterior distribution using different computational methods

You are given 10 independent and identically distributed datapoints that are assumed to come from a Binomial distribution with sample size 20 and probability of success θ:

10, 15, 15, 14, 14, 14, 13, 11, 12, 16

Model:

Let y_i be the i^{th} datapoint,

$$y_i \sim Binomial(n = 20, \theta)$$

$$\theta \sim Beta(1, 1)$$

The analytically-derived posterior distribution of θ is a Beta distribution with shape parameters 135 and 67.

$$\theta|y \sim Beta(135, 67)$$

Use the above information to solve the following exercises.

1. Graph the analytically-derived posterior distribution of θ.
2. Use **grid approximation** to estimate the posterior density function of θ and graph the estimated posterior density function.
3. Use Monte Carlo integration to estimate the marginal likelihood of the model. (Hint: Draw a large number of samples from the prior `Beta(1,1)`, compute the likelihood for each sample, and calculate the average of all the likelihoods.)
4. Use **importance sampling** to draw samples from the posterior distribution of θ.

   (Hint: Draw N samples of θ from a proposal density function, say $q(\theta)$. Compute the likelihood $L(\theta_i|y)$, prior $p(\theta_i)$, and proposal density $q(\theta_i)$ for each sample. Store each sample θ_i and its corresponding weight w_i as a row in a dataframe, where $w_i = \frac{L(theta_i|y)p(\theta_i)}{q(\theta_i)}$. Now, select $N/4$ samples from the vector of N samples based on their weights. You can use the function $sample(theta\_vector, size = n/4, prob = weights\_vector)$. These new $N/4$ samples are the samples from the posterior.)

```
# Assuming df is the dataframe containing samples of theta and
# their correponding weights as the columns 'theta' and 'weights' respectively.
```

```
n <- length(df$theta)
sample(df$theta, size=n/4,prob=df$weights)
```

5. Use Markov chain Monte Carlo method to estimate the posterior distribution of θ.

   (Hint: You can use the Metropolis-Hastings algorithm code given on page 20 in the lecture notes.)

6. Graphically compare the posterior distributions of θ obtained using:

   - Importance sampling
   - Markov chain Monte Carlo
   - Analytical derivation

Part 2: Writing your own sampler for Bayesian inference

2.1 The research problem

In a visual word recognition experiment, a participant has to recognize whether a string shown on the screen is a meaningful word (e.g., "book") or a non-word (e.g., "bktr"). The participant is asked to answer "yes" if the shown string is a meaningful word, and "no" if it is a meaningless non-word. Suppose a participant is shown n words and n non-words on the screen one by one and you record the recognition time for each word/non-word.

Say, T_w is the vector of word recognition times, and T_{nw} is the vector of non-word recognition times.

You ask the following question:

Does it take longer to recognize the non-words compared to the words?

Technically,

Is the mean recognition time for the non-words larger than the mean recognition time for the words?

2.2 Hypothesis

Lexical-access hypothesis: The mean recognition time for the non-words is longer than the mean recognition time for the non-words.

2.3 Model

RT_i is the observed recognition time of the i^{th} string, which can be a word or a non-word.

Likelihood assumption:

$$RT_i \sim Normal(\mu_i, \sigma)$$

such that

$$\mu_i = \alpha + \beta \cdot type_i$$

where $type_i$ indicate whether the i^{th} string is a word or a non-word. If the string i is a non-word, the $type_i$ will be equal to 1, otherwise 0. α, β, and σ are the three parameters of the model; α represents mean recognition time for the words, and β represents to what extent non-words are recognized slower than words.

Priors:

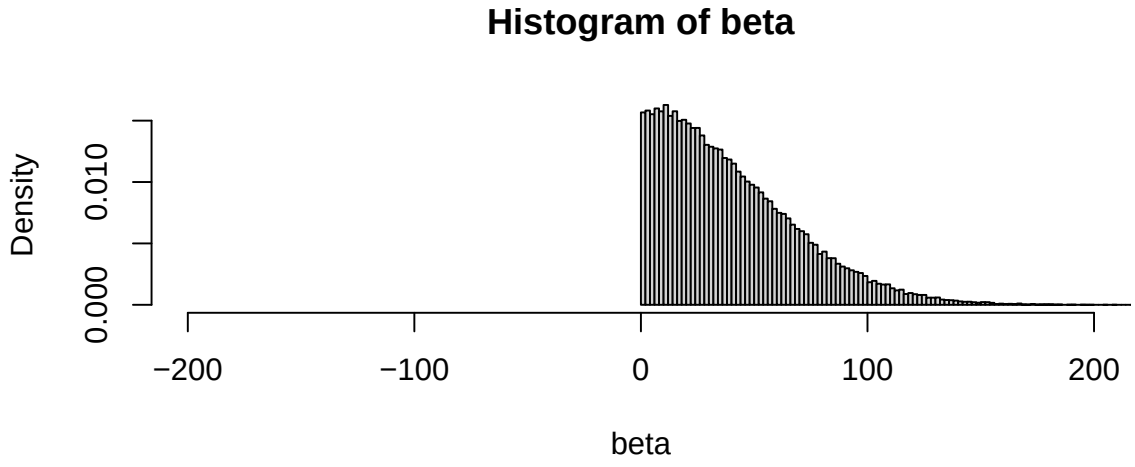
$$\alpha \sim Normal(400, 50)$$

$$\sigma = 30$$

$$\beta \sim Normal_+(0, 50)$$

where $Normal_+()$ represent a truncated normal distribution such that β would always be larger than 0.

```
library(truncnorm)
# You can generate from a truncated normal distribution using rtruncnorm
beta <- rtruncnorm(100000,a=0,b=Inf,mean=0,sd=50)
hist(beta,xlim = c(-200,200),probability = T,breaks = 100)
```




```
# You can calculate the probability density of obtaining a value x
# from the truncated normal distribution.
x <- 20
density_x <- dtruncnorm(x,a=0,b=Inf,mean=0,sd=50)
```

2.4 Data

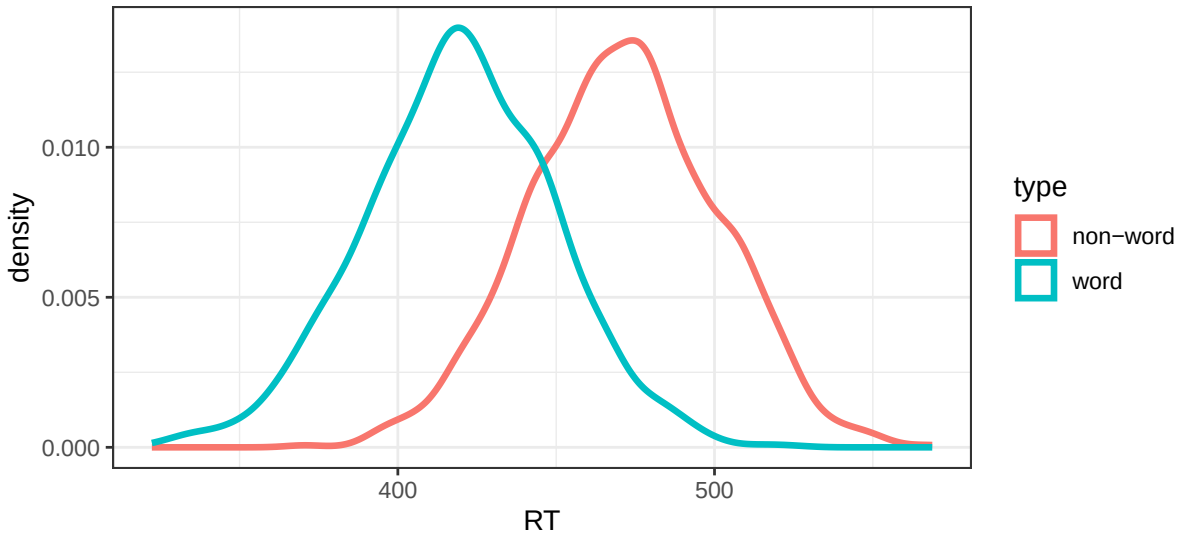
The file `word-recognition-times.csv` on github contains the recognition times data for 4000 strings which can be either words or non-words represented by the column *type*; the recognition times are given in the column *RT*. Use the following code to load the data. (Alternatively, you can directly download the csv file and read it using `read.table`.)

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

```
##       type       RT
## 1     word 423.1019
## 2     word 429.9432
## 3 non-word 486.9959
## 4 non-word 451.4400
## 5 non-word 482.2657
## 6 non-word 470.8003
```

```
ggplot(dat,aes(x=RT,color=type))+geom_density(size=1.2)+theme_bw()
```

```
## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```



2.5 Exercises

2.5.1 Estimate the parameters α and β using **Markov Chain Monte Carlo.**

(Hint: Transform posterior densities on log scale. See the algorithm on page 24 in the lecture notes.)

2.5.2 What is the 95% credible interval for each parameter?

(If you can say with 95% certainty that the true value of the parameter lies between a and b, then the range $[a, b]$ is called the credible interval. You can estimate the credible interval from the vector of samples from the posterior, using `quantile(post_samples,probs=c(.025,.975))`)

Part 3: Hamiltonian Monte Carlo sampler

You are given 500 data points that are assumed to come from a normal distribution with mean μ and variance σ^2,

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

Copy and paste the above chunk of code to generate data y.
Model:
Let y_i be i^{th} data point,
$y_i \sim Normal(\mu, \sigma^2)$
$\mu \sim Normal(m = 1000, s = 100)$
$\sigma \sim Normal(a = 10, b = 2)$
You have to estimate the posterior distributions of μ and σ using Hamiltonian Monte Carlo sampler.

You can directly use the following HMC sampler code to estimate the parameters of the above model.

```
#Gradient functions
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=T))+dnorm(mu,m,s,log=T)+dnorm(sigma,a,b,log=T))
  nlpd
}
```

```r
#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
    }
  }
```

```
  posteriors <- data.frame(mu_chain,sigma_chain)[-(1:nburn),]
  posteriors$sample_id <- 1:nrow(posteriors)
  posteriors
}
```

Exercise 3.1 Use the following values for the internal parameters of the HMC sampler,

- Total number of samples, $nsamp = 6000$
- Total number of burn-in samples, $nburn = 2000$ (a certain number of initial samples that you want to throw away)
- Step-size parameter, $step = 0.02$
- Number of leapfrog steps, $L = 12$
- Initializing value of the *mu* and *sigma* chain, $initial_q = c(1000, 11)$

Estimate and plot the posteriors for μ and σ.
(Hint: You can directly run the HMC function as follows:)

```
df.posterior <- HMC(y=y,n=length(y),          # data
                    m=1000,s=20,a=10,b=2,     # priors
                    step=0.02,                # step-size
                    L=12,                     # no. of leapfrog steps
                    initial_q=c(1000,11),      # Chain initialization
                    nsamp=6000,               # total number of samples
                    nburn=2000)               # number of burn-in samples
```

Exercise 3.2 Check posterior sensitivity to the total number of samples. How do the posteriors change with change in total number of samples?

Estimate and compare the posteriors obtained for the following values of total number of samples, $nsamp$ -

- $nsamp = 100$
- $nsamp = 1000$
- $nsamp = 6000$

Change burn-in samples $nburn$ accordingly. For example, you can set $nburn = nsamp/3$. Keep all other parameters same as in Expercise 1.1.

Exercise 3.3 How do the posteriors change with change in step-size parameter?

Estimate and compare the posteriors obtained when step-size had the ollowing values -

- $step = 0.001$
- $step = 0.005$

- $step = 0.02$

Keep all other parameter same as in Exercise 1.1.

Exercise 3.4 Visually inspect the *mu* and *sigma* chains obtained in expercise 3.3. Do you find anything problematic?

Exercise 3.5 Check the prior sensitivity for the μ parameter. Estimate and compare the posterior distribution of μ when the prior on μ are -

- $\mu \sim Normal(m = 400, s = 5)$
- $\mu \sim Normal(m = 400, s = 20)$
- $\mu \sim Normal(m = 1000, s = 5)$
- $\mu \sim Normal(m = 1000, s = 20)$
- $\mu \sim Normal(m = 1000, s = 100)$

(Keep all other parameters same as in Exercise 3.1).

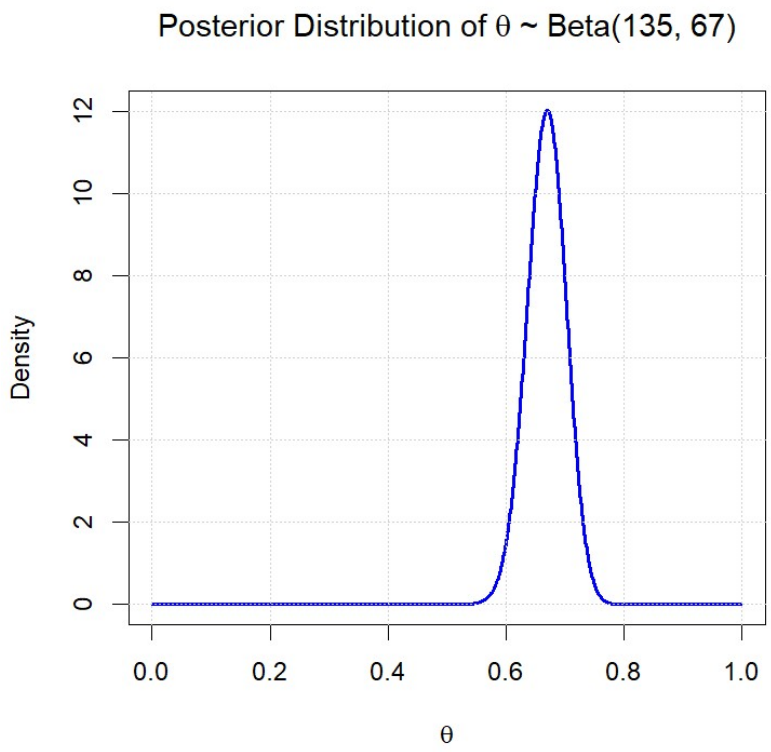
Part 1
Answer Number: 1

CODE -

assignment 3 solution.R

```r
#parameters
alpha <- 135
beta <- 67
#theta values from 0 to 1
theta <- seq(0, 1, length.out = 1000)
posterior <- dbeta(theta, alpha, beta)
#Plot
plot(theta, posterior, type = "l", lwd = 2, col = "blue",
     main = expression("Posterior Distribution of " * theta * " ~ Beta(135, 67)"),
     xlab = expression(theta), ylab = "Density")
grid()
```

OUTPUT - Graph of the analytically derived posterior distribution of θ,



Part 1
Answer Number: 2

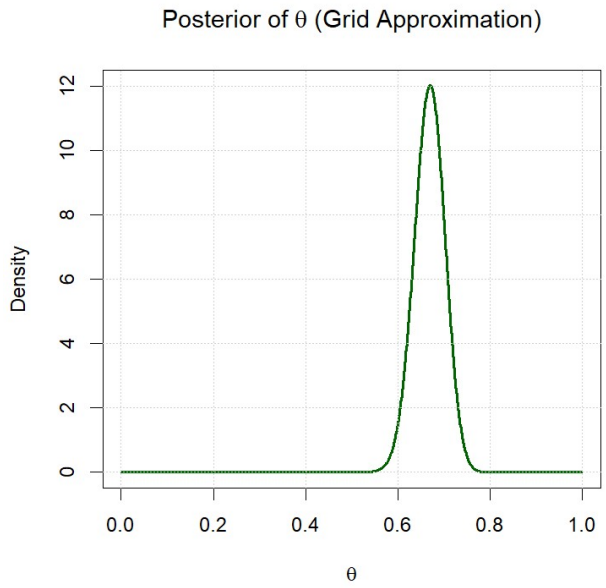
CODE -

assignment 3 solution.R*

Source on Save | Run | Source

```r
# Observed data
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n <- 20
# Prior parameters for Beta(1, 1)
a_prior <- 1
b_prior <- 1
# Grid of theta values
theta <- seq(0, 1, length.out = 1000)
likelihood <- sapply(theta, function(t) {
  prod(dbinom(y, size = n, prob = t))
})
# Compute the prior
prior <- dbeta(theta, a_prior, b_prior)
unnorm_posterior <- likelihood * prior
posterior <- unnorm_posterior / sum(unnorm_posterior * diff(theta)[1])
# Plot
plot(theta, posterior, type = "l", lwd = 2, col = "darkgreen",
     main = expression("Posterior of " * theta * " (Grid Approximation)"),
     xlab = expression(theta), ylab = "Density")
grid()
```

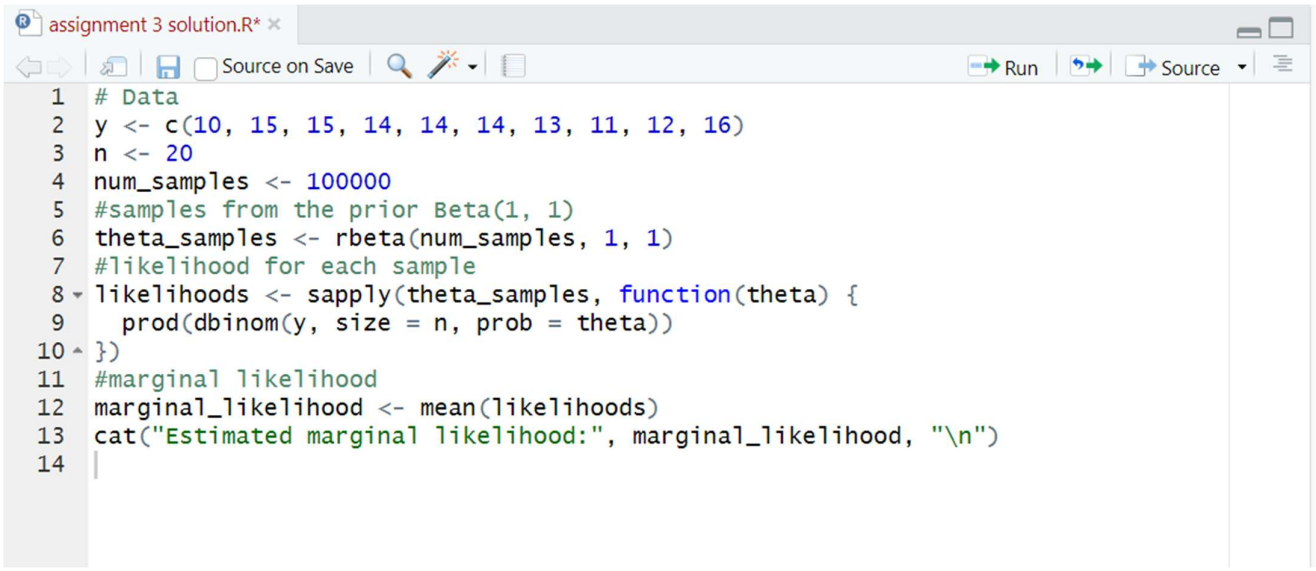
OUTPUT- Use of grid approximation to estimate the posterior density function of θ



Part 1

Answer Number: 3

CODE -



OUTPUT -

Answer -

Estimated marginal likelihood: 1.397006e-10

```
> #marginal likelihood
> marginal_likelihood <- mean(likelihoods)
> cat("Estimated marginal likelihood:", marginal_likelihood, "\n")
Estimated marginal likelihood: 1.397006e-10
```

Part 1

Answer Number: 4

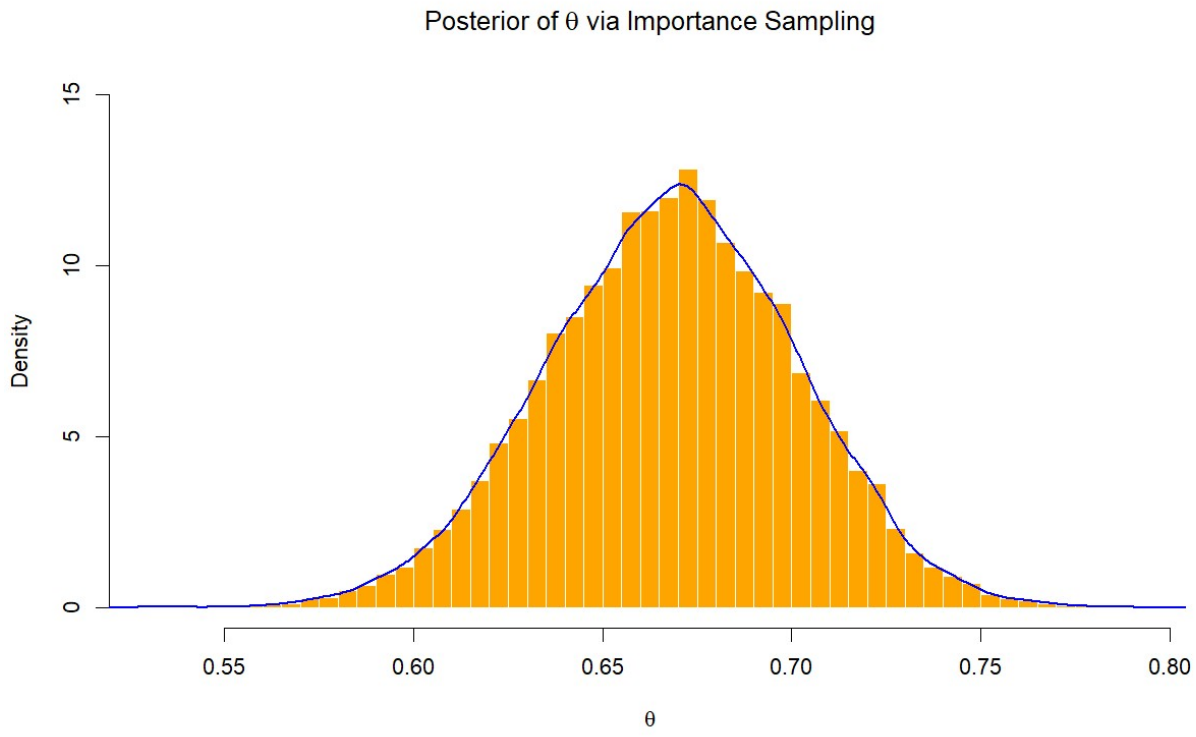
CODE –

assignment 3 solution.R*

```r
# Data
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n_obs <- 20
N <- 100000  # Number of samples
#proposal distribution
q_alpha <- 5
q_beta <- 5
theta_samples <- rbeta(N, q_alpha, q_beta)
#compute prior density
prior_density <- dbeta(theta_samples, 1, 1)
#compute proposal density
proposal_density <- dbeta(theta_samples, q_alpha, q_beta)
#compute likelihood for each theta
likelihood <- sapply(theta_samples, function(theta) {
  prod(dbinom(y, size = n_obs, prob = theta))
})
weights <- likelihood * prior_density / proposal_density
#normalize weights
weights_normalized <- weights / sum(weights)
#store in dataframe
df <- data.frame(theta = theta_samples, weights = weights)
#resample N/4 samples from theta based on weights
posterior_samples <- sample(df$theta, size = N/4, prob = df$weights, replace = TRUE)
#plot
hist(posterior_samples, breaks = 50, probability = TRUE, col = "orange",
     border = "white", main = expression("Posterior of " * theta * " via Importance Sampling"),
     xlab = expression(theta), ylim = c(0, 15))
lines(density(posterior_samples), col = "blue", lwd = 2)

```

OUTPUT -

Answer – Use of importance sampling to draw samples from the posterior distribution of θ



Part 1

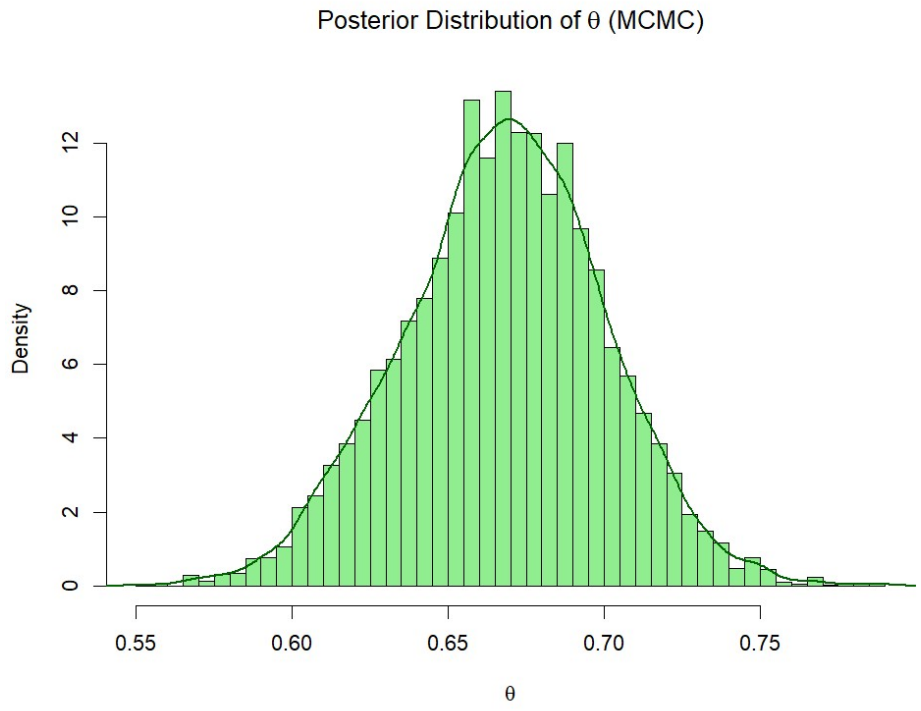
Answer Number: 5

CODE -

```r
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n_obs <- 20
n_iter <- 10000  #number of MCMC iterations
theta <- numeric(n_iter)
theta[1] <- runif(1)
proposal_sd <- 0.05# (tune for acceptance rate ~ 0.3)
log_prior <- function(theta) {
  if (theta <= 0 || theta >= 1) return(-Inf)
  return(0)
}
#Log-likelihood for all observations
log_likelihood <- function(theta) {
  sum(dbinom(y, size = n_obs, prob = theta, log = TRUE))
}
#MCMC via Metropolis-Hastings
for (i in 2:n_iter) {
  theta_current <- theta[i - 1]
  theta_proposed <- rnorm(1, mean = theta_current, sd = proposal_sd)
  log_p_current <- log_likelihood(theta_current) + log_prior(theta_current)
  log_p_proposed <- log_likelihood(theta_proposed) + log_prior(theta_proposed)
  log_accept_ratio <- log_p_proposed - log_p_current
  if (log(runif(1)) < log_accept_ratio) {
    theta[i] <- theta_proposed
  } else {
    theta[i] <- theta_current
  }
}
burn_in <- 2000 #discard burn-in and plot posterior
posterior_samples <- theta[(burn_in + 1):n_iter]
# Overlay histogram and density
hist(posterior_samples, breaks = 50, probability = TRUE, col = "lightgreen",
     main = expression("Posterior Distribution of " * theta * " (MCMC)"),
     xlab = expression(theta))
lines(density(posterior_samples), col = "darkgreen", lwd = 2)
```

OUTPUT -

ANSWER - Use of Markov chain Monte Carlo method to estimate the posterior distribution of θ



Part 1

Answer Number: 6

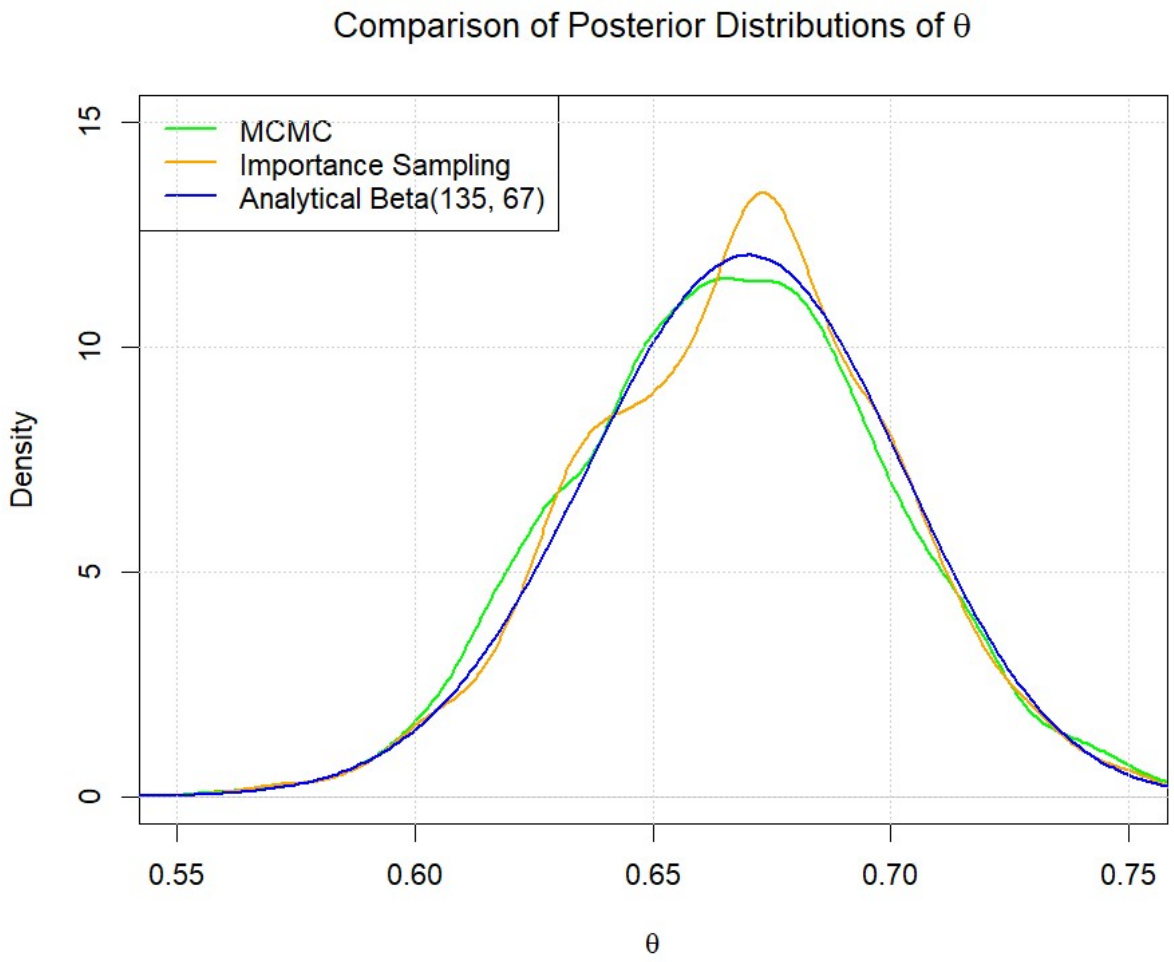
CODE-

assignment 3 solution.R*

```r
y <- c(10, 15, 15, 14, 14, 13, 11, 12, 16, 14)#data
n_trials <- 20
alpha_prior <- 1
beta_prior <- 1
alpha_post <- alpha_prior + sum(y)
beta_post <- beta_prior + (length(y) * n_trials - sum(y))
theta_grid <- seq(0, 1, length.out = 1000)
beta_density <- dbeta(theta_grid, alpha_post, beta_post)
#importance Sampling
set.seed(123)
N <- 10000
theta_samples_is <- rbeta(N, 2, 2)  #Beta(2, 2)
# Likelihood
likelihood_is <- sapply(theta_samples_is, function(theta) {
  prod(dbinom(y, size = n_trials, prob = theta))
})
# Prior
prior_is <- dbeta(theta_samples_is, alpha_prior, beta_prior)
proposal_is <- dbeta(theta_samples_is, 2, 2)
# Weights
weights <- (likelihood_is * prior_is) / proposal_is
weights <- weights / sum(weights)  # Normalize
#posterior samples based on weights
posterior_samples_is <- sample(theta_samples_is, size = N / 4, replace = TRUE, prob = weights)
#MCMC Metropolis-Hastings
set.seed(42)
mcmc_samples <- 10000
theta_mcmc <- numeric(mcmc_samples)
theta_mcmc[1] <- runif(1, 0, 1)
proposal_sd <- 0.05
for (i in 2:mcmc_samples) {
  theta_prop <- rnorm(1, theta_mcmc[i - 1], proposal_sd)
  if (theta_prop > 0 && theta_prop < 1) {
    log_lik_curr <- sum(dbinom(y, n_trials, theta_mcmc[i - 1], log = TRUE))
    log_lik_prop <- sum(dbinom(y, n_trials, theta_prop, log = TRUE))
    log_accept_ratio <- log_lik_prop - log_lik_curr
    if (log(runif(1)) < log_accept_ratio) {
      theta_mcmc[i] <- theta_prop
    } else {
      theta_mcmc[i] <- theta_mcmc[i - 1]
    }
  } else {
    theta_mcmc[i] <- theta_mcmc[i - 1]
  }
}
posterior_samples_mcmc <- theta_mcmc[5001:10000]  # Discard burn-in
#Densities for Plotting
dens_mcmc <- density(posterior_samples_mcmc)
dens_is <- density(posterior_samples_is)
#Plot
plot(dens_mcmc, col = "green", lwd = 2, xlim = c(0.55, 0.75), ylim = c(0, 15),
     main = expression("Comparison of Posterior Distributions of " * theta),
     xlab = expression(theta), ylab = "Density")
lines(dens_is, col = "orange", lwd = 2)
lines(theta_grid, beta_density, col = "blue", lwd = 2)

legend("topleft",
       legend = c("MCMC", "Importance Sampling", "Analytical Beta(135, 67)"),
       col = c("green", "orange", "blue"),
       lwd = 2)
grid()
```

ANSWER – Comparison of the posterior distributions of θ obtained using:

Method	Accuracy	Flexibility	Speed	When to Use
Analytical Posterior	Exact	Low	Fast	When conjugate priors are available
Importance Sampling	Approximate	Moderate	Depends on	When a good proposal is available
MCMC	Approximate	Very High	Slower	For complex or high-dimensional models



• **Green Curve - Markov Chain Monte Carlo (MCMC):**
This curve represents samples drawn via the Metropolis-Hastings algorithm. It shows a smooth, bell-shaped density centered around **θ ≈ 0.67**, matching the expected posterior well.

• **Orange Curve - Importance Sampling:**
This approximation also centers around **θ ≈ 0.67**, showing good alignment with MCMC. Some slight roughness or asymmetry may appear if the sample size is limited or the proposal distribution isn't optimal.

• **Blue Curve - Analytical Posterior (Beta (135, 67):**
This is the exact posterior derived using conjugate priors. It serves as the benchmark, with a sharp, narrow peak indicating high certainty around the mean.

Part 2

Answer Number: 2.5.1

CODE -

assignment 3 solution.R*

```r
library(ggplot2)# Load libraries
#data
dat <- read.csv("C:/Users/piyus/OneDrive - IIT Kanpur/Desktop/summer term/word-recognition-times.csv")
# Encode type: 0 for word, 1 for non-word
dat$type_num <- ifelse(dat$type == "word", 0, 1)
#define the log-posterior
log_posterior <- function(alpha, beta, sigma, RT, type) {
  mu <- alpha + beta * type
  log_likelihood <- sum(dnorm(RT, mean = mu, sd = sigma, log = TRUE))
  # Priors
  log_prior_alpha <- dnorm(alpha, mean = 400, sd = 50, log = TRUE)
  log_prior_beta  <- if (beta > 0) dnorm(beta, mean = 0, sd = 50, log = TRUE) else -Inf
  return(log_likelihood + log_prior_alpha + log_prior_beta)
}
#initialize parameters
n_iter <- 10000
alpha <- numeric(n_iter)
beta <- numeric(n_iter)
sigma <- 30
#starting values
alpha[1] <- 400
beta[1] <- 10
#proposal standard deviations
prop_sd_alpha <- 2
prop_sd_beta <- 2
#MCMC loop
for (i in 2:n_iter) {
  #propose new alpha and beta
  alpha_prop <- rnorm(1, alpha[i-1], prop_sd_alpha)
  beta_prop <- rnorm(1, beta[i-1], prop_sd_beta)
  #compute log-posterior values
  log_post_current <- log_posterior(alpha[i-1], beta[i-1], sigma, dat$RT, dat$type_num)
  log_post_proposed <- log_posterior(alpha_prop, beta_prop, sigma, dat$RT, dat$type_num)
  #acceptance probability
  log_accept_ratio <- log_post_proposed - log_post_current
  if (log(runif(1)) < log_accept_ratio) {
    alpha[i] <- alpha_prop
    beta[i] <- beta_prop
  } else {
    alpha[i] <- alpha[i-1]
    beta[i] <- beta[i-1]
  }
}
#discard burn-in
burn_in <- 2000
alpha_post <- alpha[(burn_in+1):n_iter]
beta_post <- beta[(burn_in+1):n_iter]
#posterior summaries
cat("Posterior mean of alpha:", mean(alpha_post), "\n")
cat("Posterior mean of beta:", mean(beta_post), "\n")
#plot posteriors
par(mfrow=c(1,2))
hist(alpha_post, main="Posterior of Alpha", xlab="alpha", col="skyblue", breaks=50)
hist(beta_post, main="Posterior of Beta", xlab="beta", col="orange", breaks=50)
```

We adopt a **Bayesian approach** using **Markov Chain Monte Carlo (MCMC)** to estimate the parameters:

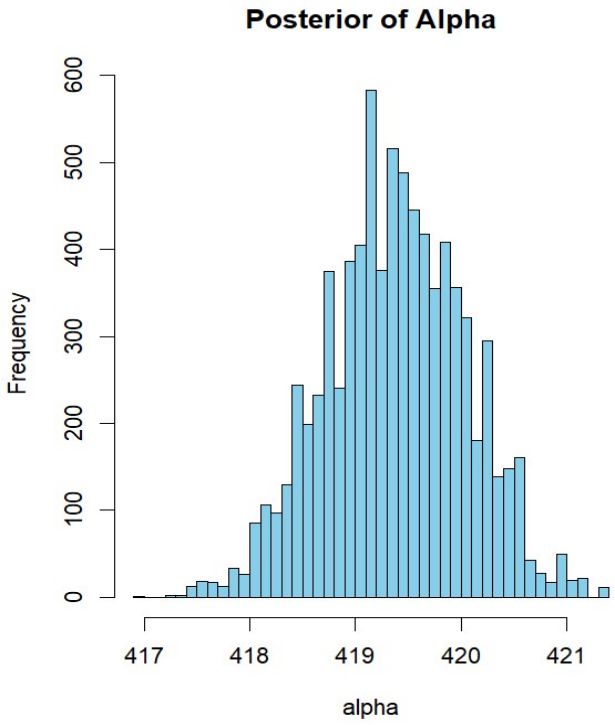
- **α:** Mean recognition time for words.
- **β:** Additional time required to recognize non-words (compared to words).
- **σ:** Known standard deviation of recognition times (set to 30).

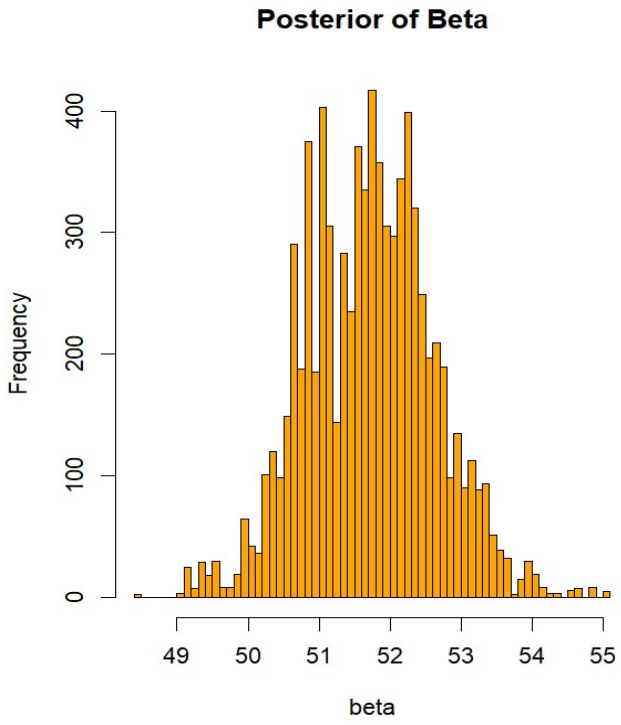
Posterior Inference via MCMC
To estimate α and β, we use the **Metropolis algorithm**, a type of MCMC. This technique allows us to sample from the posterior distributions by proposing small random changes and accepting or rejecting them based on how well they fit the data .

We use the **log scale** for evaluating the posterior to improve numerical stability and avoid underflow from multiplying small probabilities.
After convergence (verified by discarding an initial burn-in), we summarize the posterior samples of α and β using their means and visualize them using histograms.

Conclusion
This Bayesian framework, implemented via MCMC, provides a principled way to estimate and quantify uncertainty in the recognition time difference between words and non-words.

OUTPUT – Estimation of the parameters α and β using Markov Chain Monte Carlo.





Part 2

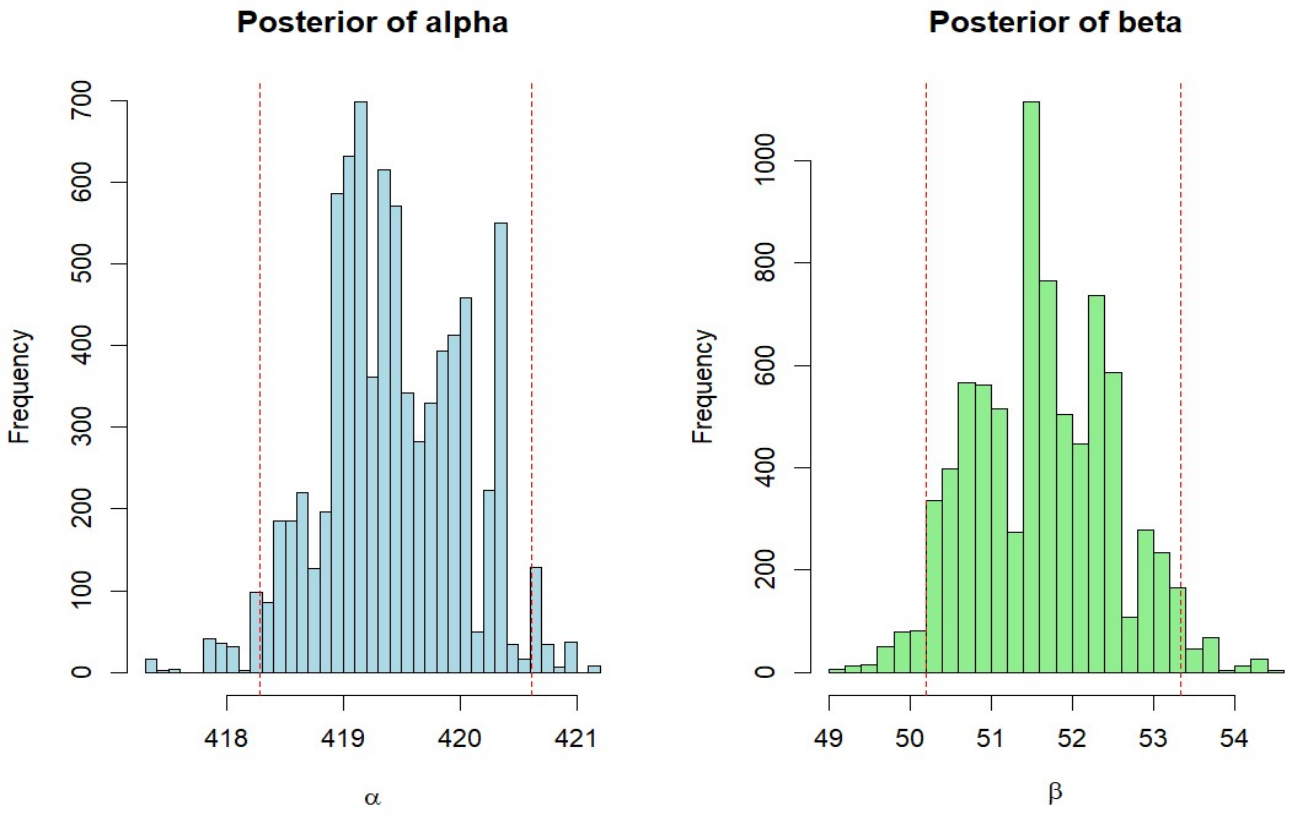
Answer Number: 2.5.2

CODE –

assignment 3 solution.R*

```r
library(truncnorm)# Load required packages
#data
dat <- read.csv("C:/Users/piyus/OneDrive - IIT Kanpur/Desktop/summer term/word-recognition-times.csv")
dat$type <- ifelse(dat$type == "non-word", 1, 0)  # Convert type to 0 (word), 1 (non-word)
#known parameters
sigma <- 30
n_iter <- 10000
burn_in <- 2000
# Extract variables
RT <- dat$RT
type <- dat$type
n <- length(RT)
alpha.samples <- numeric(n_iter)
beta.samples <- numeric(n_iter)
# Initialize parameters
alpha.curr <- 400
beta.curr <- 20
# Define log-posterior function
log_posterior <- function(alpha, beta, RT, type, sigma) {
  if (beta < 0) return(-Inf)  # Enforce truncation
  mu <- alpha + beta * type
  log_likelihood <- sum(dnorm(RT, mean = mu, sd = sigma, log = TRUE))
  log_prior_alpha <- dnorm(alpha, mean = 400, sd = 50, log = TRUE)
  log_prior_beta <- dnorm(beta, mean = 0, sd = 50, log = TRUE) - log(1 - pnorm(0, mean = 0, sd = 50))
  return(log_likelihood + log_prior_alpha + log_prior_beta)
}
# Run MCMC using random-walk Metropolis
for (i in 1:n_iter) {
  alpha.prop <- rnorm(1, mean = alpha.curr, sd = 5)
  beta.prop  <- rnorm(1, mean = beta.curr, sd = 5)

  log_post_curr <- log_posterior(alpha.curr, beta.curr, RT, type, sigma)
  log_post_prop <- log_posterior(alpha.prop, beta.prop, RT, type, sigma)

  log_accept_ratio <- log_post_prop - log_post_curr

  if (log(runif(1)) < log_accept_ratio) {
    alpha.curr <- alpha.prop
    beta.curr <- beta.prop
  }

  alpha.samples[i] <- alpha.curr
  beta.samples[i] <- beta.curr
}
# Discard burn-in samples
alpha.samples <- alpha.samples[(burn_in + 1):n_iter]
beta.samples <- beta.samples[(burn_in + 1):n_iter]
# Posterior summary
mean_alpha <- mean(alpha.samples)
mean_beta <- mean(beta.samples)
# 95% credible interval
alpha.ci <- quantile(alpha.samples, probs = c(0.025, 0.975))
beta.ci <- quantile(beta.samples, probs = c(0.025, 0.975))
#results
cat("Posterior mean of alpha:", round(mean_alpha, 2), "\n")
cat("Posterior mean of beta:", round(mean_beta, 2), "\n")
cat("95% credible interval for alpha:", round(alpha.ci[1], 2), "to", round(alpha.ci[2], 2), "\n")
cat("95% credible interval for beta:", round(beta.ci[1], 2), "to", round(beta.ci[2], 2), "\n")
# Plot posterior distributions
par(mfrow = c(1, 2))
hist(alpha.samples, main = "Posterior of alpha", xlab = expression(alpha), col = "lightblue", breaks = 30)
abline(v = alpha.ci, col = "red", lty = 2)
hist(beta.samples, main = "Posterior of beta", xlab = expression(beta), col = "lightgreen", breaks = 30)
abline(v = beta.ci, col = "red", lty = 2)
```

OUTPUT – PLOTS:



OUTPUT - VALUES:

- With 95% certainty, the mean recognition time for **words** (for alpha): **418.28 to 420.61**
- With 95% certainty, the additional recognition time for **non-words** (for beta): **50.2 to 53.33**, confirming they are recognized slower than words.

```
> # Print results
> cat("Posterior mean of alpha:", round(mean_alpha, 2), "\n")
Posterior mean of alpha: 419.43
> cat("Posterior mean of beta:", round(mean_beta, 2), "\n")
Posterior mean of beta: 51.64
> cat("95% credible interval for alpha:", round(alpha.ci[1], 2), "to", round(alpha.ci[2], 2), "\n")
95% credible interval for alpha: 418.28 to 420.61
> cat("95% credible interval for beta:", round(beta.ci[1], 2), "to", round(beta.ci[2], 2), "\n")
95% credible interval for beta: 50.2 to 53.33
```

Part 3

Answer Number: 3.1

CODE –

assignment 3 solution.R*

Source on Save | Run | Source

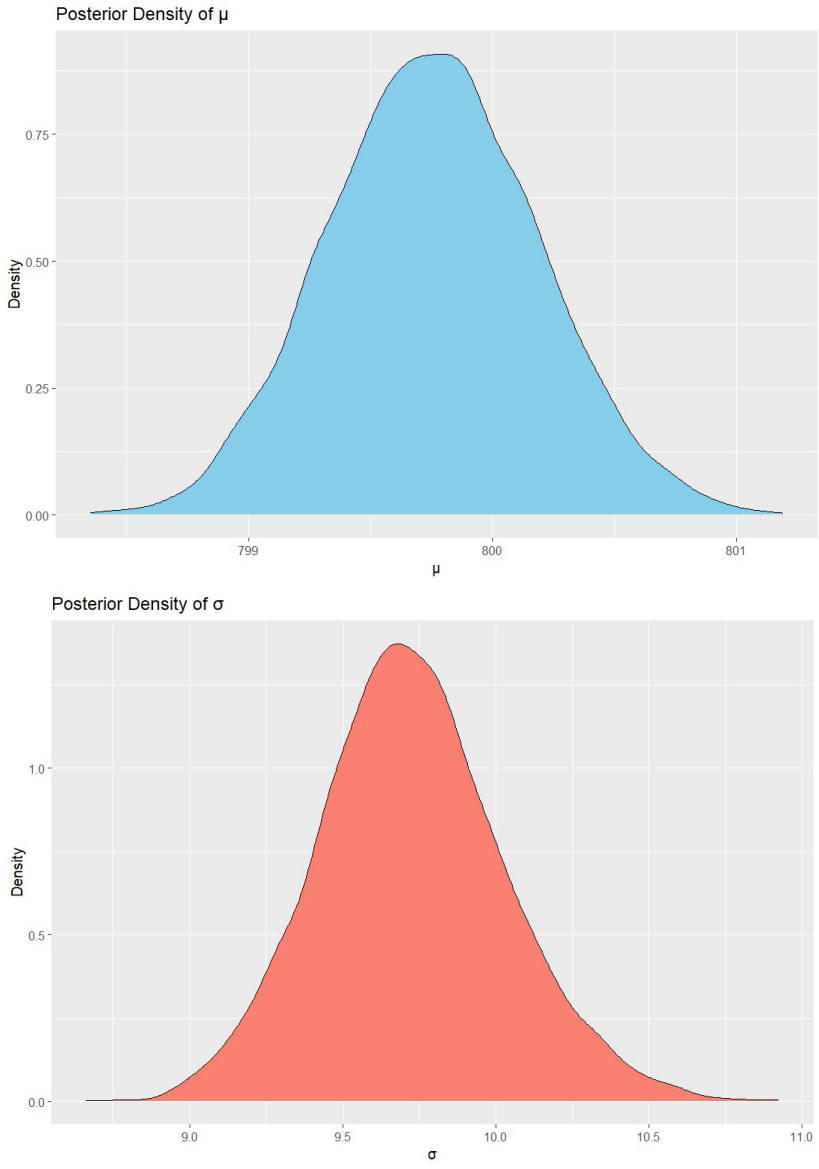
```r
set.seed(42)#data
true_mu <- 800
true_var <- 100
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))
gradient <- function(mu, sigma, y, n, m, s, a, b) {# Gradient of the negative log-posterior
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}
V <- function(mu, sigma, y, n, m, s, a, b) {# Potential energy function (negative log posterior)
  - (sum(dnorm(y, mu, sigma, log = TRUE)) +
       dnorm(mu, m, s, log = TRUE) +
       dnorm(sigma, a, b, log = TRUE))
}
# 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

  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]

  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i])
    p <- rnorm(length(q), 0, 1)
    current_q <- q
    current_p <- p

    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b)
    current_T <- sum(current_p^2) / 2
    #leapfrog integration
    for (l in 1:L) {
      q <- q + step * p
      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_T <- sum(proposed_p^2) / 2
    accept.prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))
    if (accept.prob > runif(1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
  }
  posteriors <- data.frame(mu_chain, sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}
posterior_df <- HMC( #HMC
  y = y, n = length(y),
  m = 1000, s = 100, a = 10, b = 2,
  step = 0.02, L = 12,
  initial_q = c(1000, 11),
  nsamp = 6000, nburn = 2000
)
library(ggplot2)
ggplot(posterior_df, aes(x = mu_chain)) + #density plot
  geom_density(fill = "skyblue") +
  labs(title = "Posterior Density of μ", x = "μ", y = "Density")
ggplot(posterior_df, aes(x = sigma_chain)) +
  geom_density(fill = "salmon") +
  labs(title = "Posterior Density of σ", x = "σ", y = "Density") #density plot

```

In Bayesian inference, we aim to estimate the **posterior distribution** of unknown parameters given observed data and prior beliefs.
We have:

- Data $y \sim \text{Normal}(\mu, \sigma^2)$
- Priors:
  - $\mu \sim \text{Normal}(1000, 100^2)$ → belief that mean is around 1000, but uncertain
  - $\sigma \sim \text{Normal}(10, 2^2)$ → belief that standard deviation is around 10

The goal is to sample from the **joint posterior** $p(\mu, \sigma|y)$ pushing the **HMC** technique, which improves sampling efficiency by incorporating gradients to explore the space better than basic MCMC.

OUTPUT – Plot of the posteriors for μ and σ.



• Posterior of μ should be centered around the **true value** (800), pulled slightly by the prior (which assumed 1000).

• Posterior of σ should be around **10** (square root of true_var), also influenced by the prior.

Part 3

Answer Number: 3.2

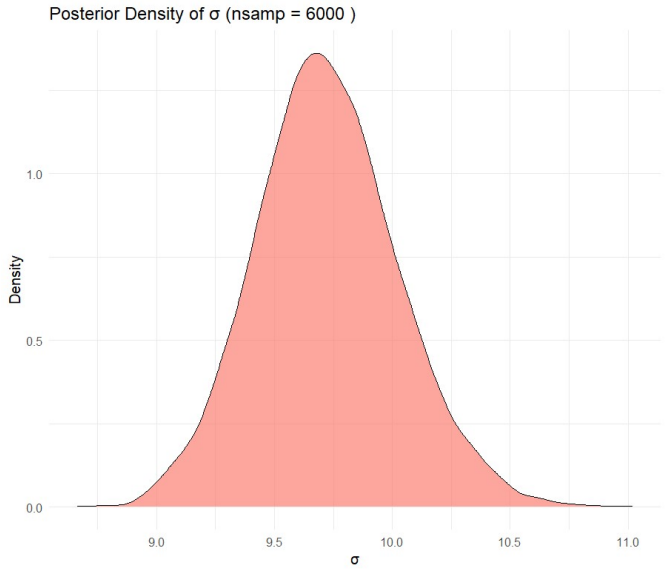
Code -

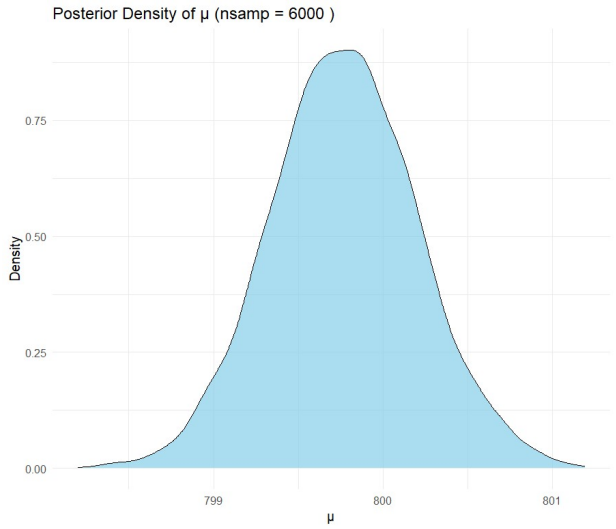
assignment 3 solution.R*

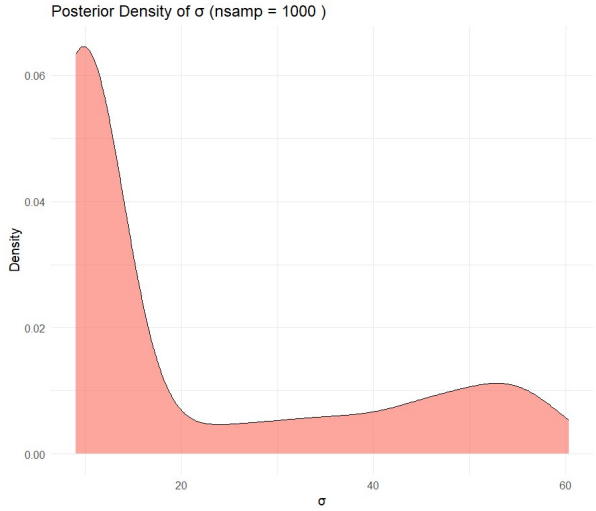
Source on Save | Run | Source

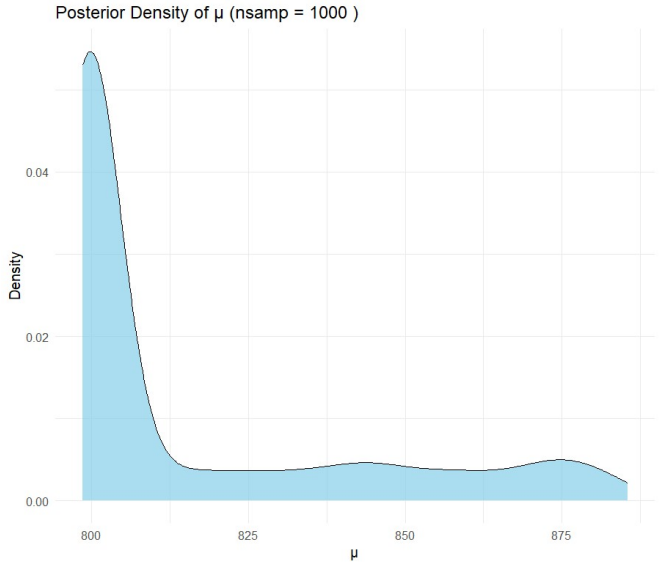
```r
set.seed(42) #Generate Data
true_mu <- 800
true_var <- 100
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))
gradient <- function(mu, sigma, y, n, m, s, a, b) { #gradient Function
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}
V <- function(mu, sigma, y, n, m, s, a, b) { #potential Energy Function
  - (sum(dnorm(y, mu, sigma, log = TRUE)) +
       dnorm(mu, m, s, log = TRUE) +
       dnorm(sigma, a, b, log = TRUE))
}
#HMC Function
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) {
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i])
    p <- rnorm(2, 0, 1)
    current_V <- V(q[1], q[2], y, n, m, s, a, b)
    current_T <- sum(p^2) / 2
    for (l in 1:L) {
      p <- p - (step / 2) * gradient(q[1], q[2], y, n, m, s, a, b)
      q <- q + step * p
      p <- p - (step / 2) * gradient(q[1], q[2], y, n, m, s, a, b)
    }
    proposed_V <- V(q[1], q[2], y, n, m, s, a, b)
    proposed_T <- sum(p^2) / 2
    if (runif(1) < exp(current_V + current_T - proposed_V - proposed_T)) {
      mu_chain[i + 1] <- q[1]
      sigma_chain[i + 1] <- q[2]
      i <- i + 1
    }
  }
  post <- data.frame(mu = mu_chain[-(1:nburn)], sigma = sigma_chain[-(1:nburn)])
  return(post)
}
#run HMC for Each nsamp Value
nsamps <- c(100, 1000, 6000)
library(ggplot2)
for (ns in nsamps) {
  nburn <- floor(ns / 3)
  post <- HMC(
    y = y, n = length(y),
    m = 1000, s = 100, a = 10, b = 2,
    step = 0.02, L = 12,
    initial_q = c(1000, 11),
    nsamp = ns, nburn = nburn
  )
  # Plot for μ
  p_mu <- ggplot(post, aes(x = mu)) +
    geom_density(fill = "skyblue", alpha = 0.7) +
    labs(title = paste("Posterior Density of μ (nsamp =", ns, ")"),
         x = "μ", y = "Density") +
    theme_minimal()
  print(p_mu)
  # Plot for σ
  p_sigma <- ggplot(post, aes(x = sigma)) +
    geom_density(fill = "salmon", alpha = 0.7) +
    labs(title = paste("Posterior Density of σ (nsamp =", ns, ")"),
         x = "σ", y = "Density") +
    theme_minimal()
  print(p_sigma)
}
```

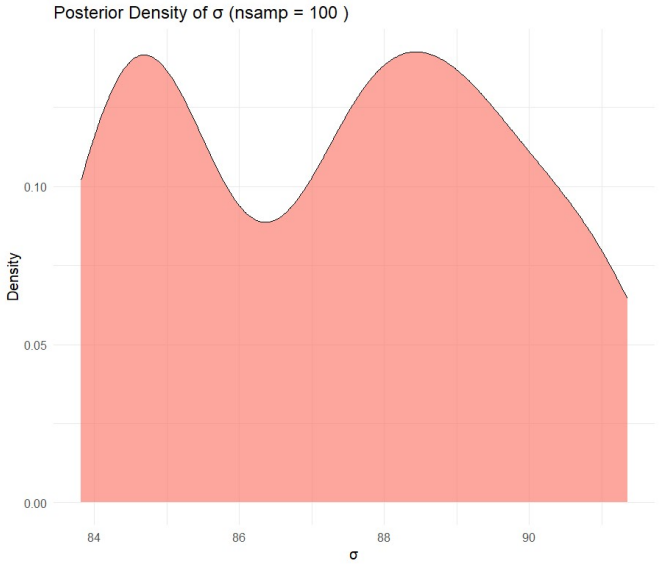
OUTPUT - Posteriors obtained for the given values of total number of samples.

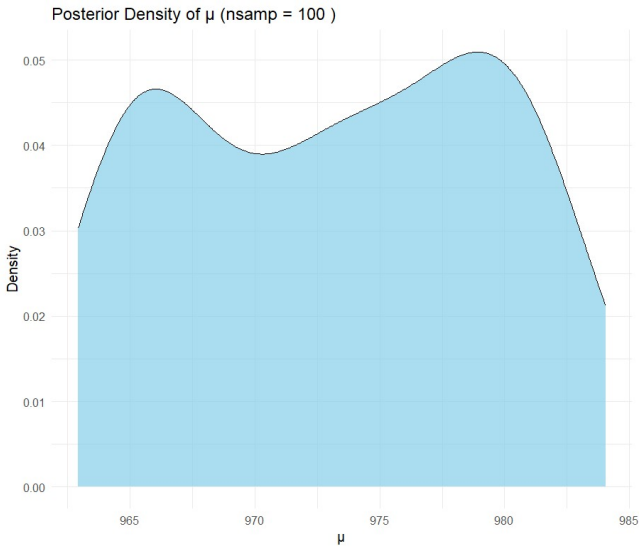












Setting	nsamp	nburn
A	100	33
B	1000	333
C	6000	2000

All other parameters (step size, leapfrog steps, priors, initial values) are kept the same as in Exercise 1.1.

Setting	Posterior Shape	Stability	Comments
100	Very noisy	Poor	Posterior is bumpy due to low sample size
1000	Smoother	Moderate	Better estimate but still some variance
6000	Very smooth	Strong	Posterior closely matches true values, more confidence

- More **samples → better approximation** of the true posterior.
- Fewer **samples → unreliable estimates** (not just noisy plots, but real inference problems).
- For practical use, always aim for **converged and stable posteriors** (like nsamp = 6000).

Part 3

Answer Number: 3.3

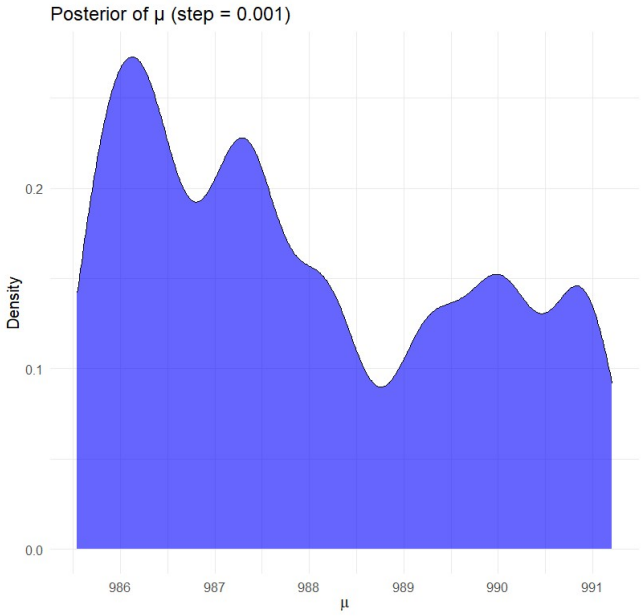
Code –

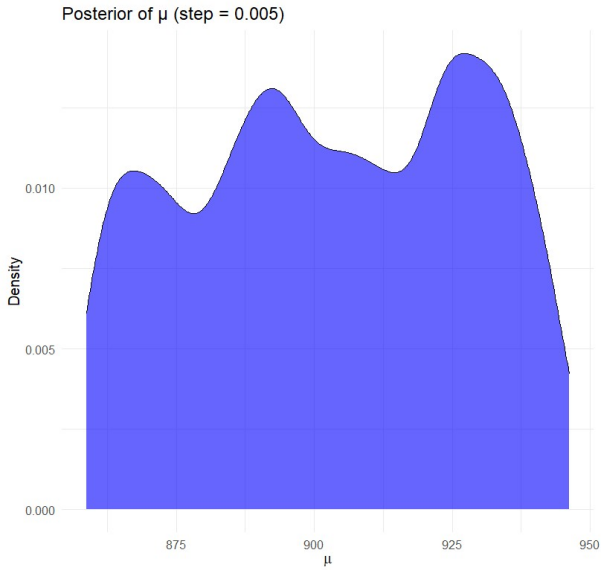
assignment 3 solution.R*

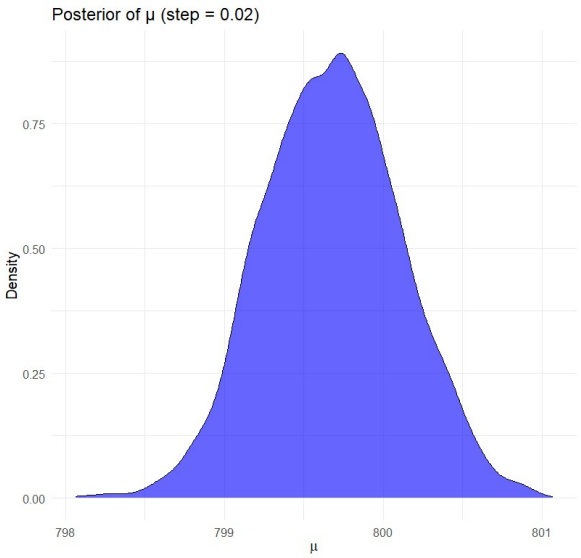
Source on Save | Run | Source

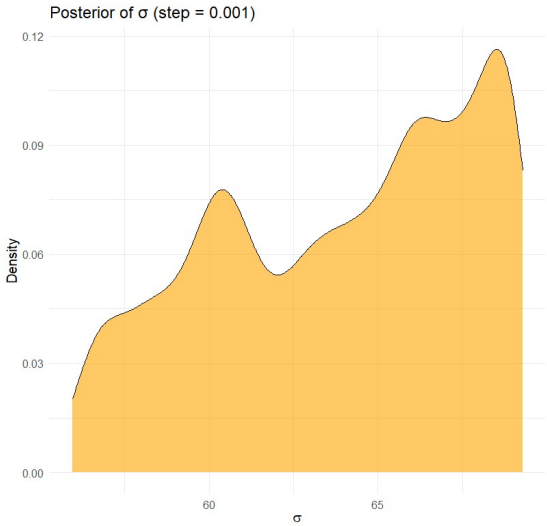
```r
#necessary libraries
library(ggplot2)
#run HMC for different step sizes
run_HMC_step <- function(step_size, label) {
  post <- HMC(
    y = y, n = length(y),
    m = 1000, s = 100, a = 10, b = 2,
    step = step_size, L = 12,
    initial_q = c(1000, 11),
    nsamp = 6000, nburn = 2000
  )
  post$step_label <- label
  return(post)
}
#Run HMC for different step sizes
posterior_001 <- run_HMC_step(0.001, "0.001")
posterior_005 <- run_HMC_step(0.005, "0.005")
posterior_020 <- run_HMC_step(0.02,  "0.02")
# μ posterior
ggplot(posterior_001, aes(x = mu)) +
  geom_density(fill = "blue", alpha = 0.6) +
  labs(title = "Posterior of μ (step = 0.001)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(posterior_005, aes(x = mu)) +
  geom_density(fill = "blue", alpha = 0.6) +
  labs(title = "Posterior of μ (step = 0.005)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(posterior_020, aes(x = mu)) +
  geom_density(fill = "blue", alpha = 0.6) +
  labs(title = "Posterior of μ (step = 0.02)", x = expression(mu), y = "Density") +
  theme_minimal()
# σ posterior
ggplot(posterior_001, aes(x = sigma)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of σ (step = 0.001)", x = expression(sigma), y = "Density") +
  theme_minimal()
ggplot(posterior_005, aes(x = sigma)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of σ (step = 0.005)", x = expression(sigma), y = "Density") +
  theme_minimal()
ggplot(posterior_020, aes(x = sigma)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of σ (step = 0.02)", x = expression(sigma), y = "Density") +
  theme_minimal()

```

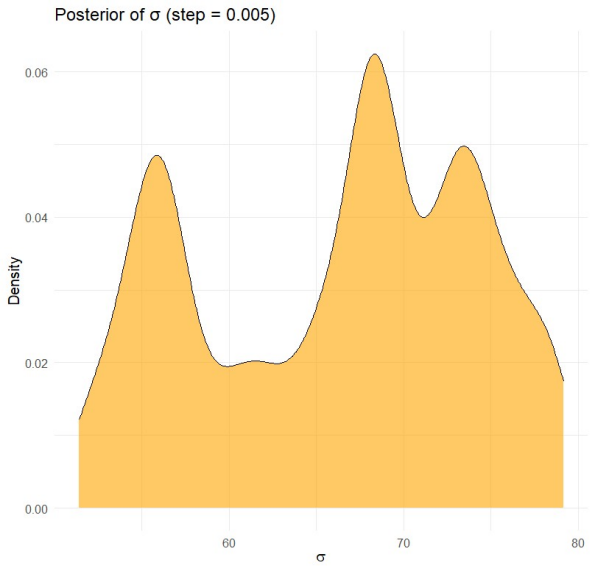
Scenario	Step size	Other Parameters
A	0.001	Same as Ex 1.1
B	0.005	Same
C	0.02	Same (default)

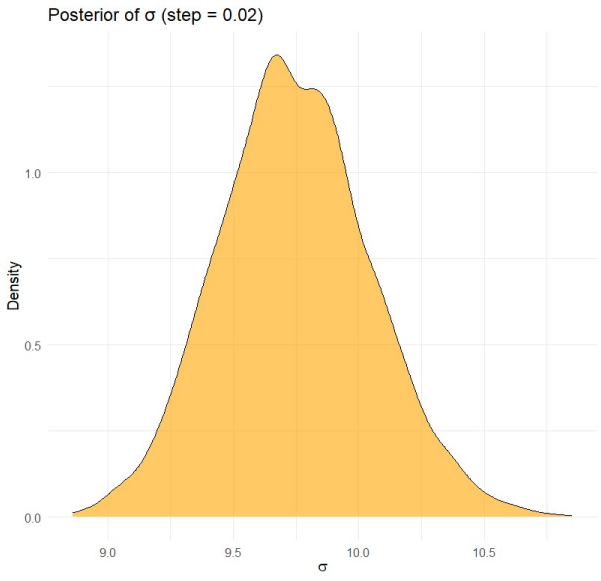

Posterior of μ (step = 0.001)
Density
0.0
0.1
0.2
986
987
988
989
990
991
μ


Posterior of μ (step = 0.005)
Density
0.000
0.005
0.010
875
900
925
950
μ


Posterior of μ (step = 0.02)
Density
0.00
0.25
0.50
0.75
798
799
800
801
μ


Posterior of σ (step = 0.001)
Density
0.00
0.03
0.06
0.09
0.12
60
65
σ


Posterior of σ (step = 0.005)
Density
0.00
0.02
0.04
0.06
60
70
80
σ


Posterior of σ (step = 0.02)
Density
0.0
0.5
1.0
9.0
9.5
10.0
10.5
σ

Step Size	Posterior Behavior	Explanation
0.001	Smooth but very narrow; maybe underexplored	Sampler moves very slowly and may not explore the whole posterior well.
0.005	Better than 0.001, still stable	Good balance between accuracy and exploration.
0.02	Smooth and well-spread	Default in Ex 1.1; efficient movement with acceptable accuracy.

- Too **small step** → the sampler takes forever to move slowly mixing.
- Too **large step** → leapfrog steps become inaccurate → poor acceptance.
- Optimal step size ensures **good exploration** of the posterior **without wasting time or accuracy**.

Part 3

Answer Number: 3.4

Code –

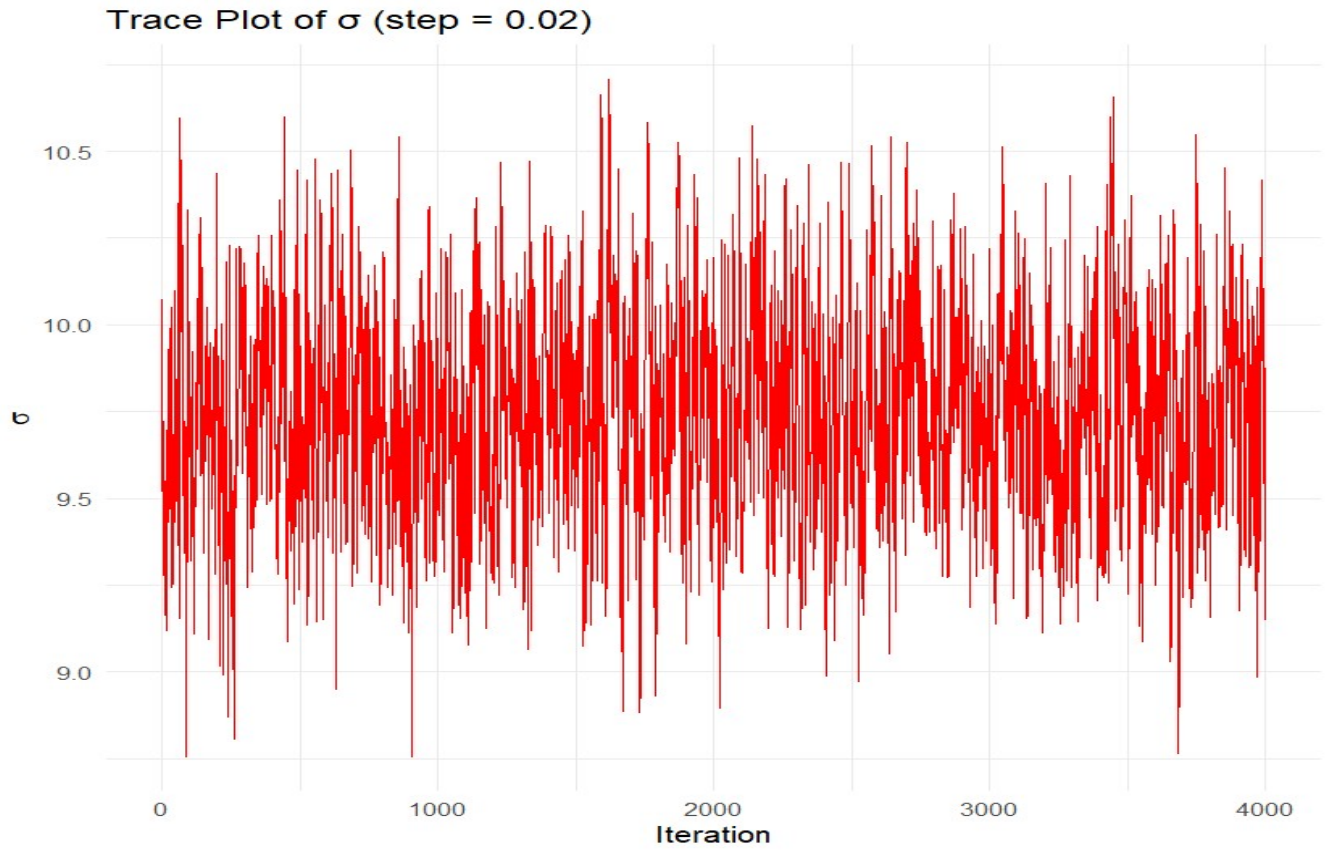
assignment 3 solution.R*

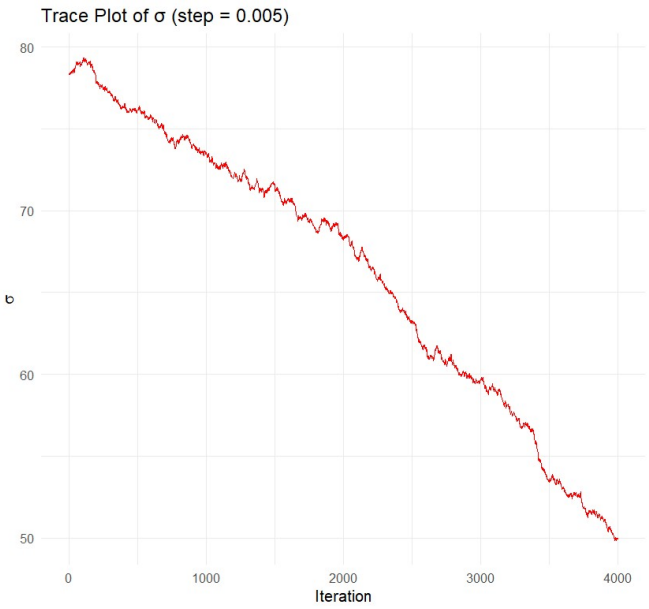
Source on Save | Run | Source

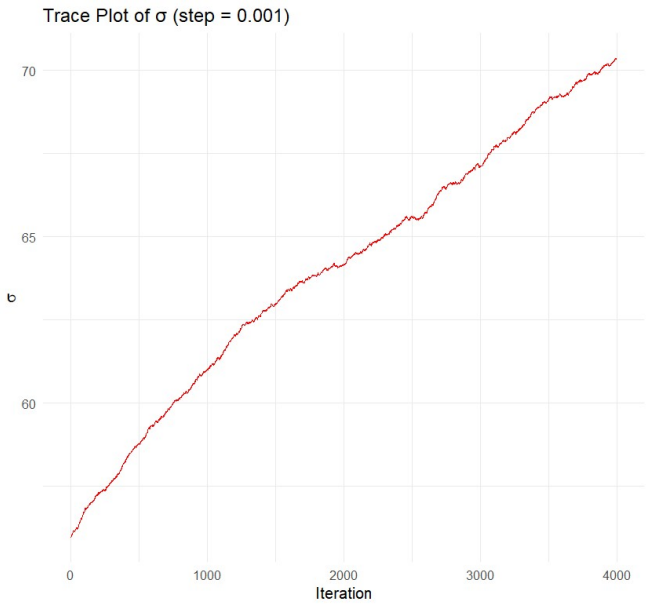
```r
# Load required libraries
library(ggplot2)
library(dplyr)
#Simulate Data
set.seed(123)
true_mu <- 800
true_var <- 100
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))
#Define Gradient Function
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}
V <- function(mu, sigma, y, n, m, s, a, b) { #Potential Energy Function
  - (sum(dnorm(y, mu, sigma, log = TRUE)) +
       dnorm(mu, m, s, log = TRUE) +
       dnorm(sigma, a, b, log = TRUE))
}
HMC <- function(y, n, m, s, a, b, step, L, initial_q, nsamp, nburn) { #HMC Sampler
  mu_chain <- rep(NA, nsamp)
  sigma_chain <- rep(NA, nsamp)
  reject <- 0
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i])
    p <- rnorm(length(q), 0, 1)
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b)
    current_T <- sum(current_p^2) / 2
    for (l in 1:L) {
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      q <- q + step * p
      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_T <- sum(proposed_p^2) / 2
    accept_prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))
    if (accept_prob > runif(1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
  }
  posteriors <- data.frame(mu = mu_chain, sigma = sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}
#5Wrapper Function for Step Sensitivity
run_HMC_step <- function(step_size, label) {
  post <- HMC(
    y = y, n = length(y),
    m = 1000, s = 100, a = 10, b = 2,
    step = step_size, L = 12,
    initial_q = c(1000, 11),
    nsamp = 6000, nburn = 2000
  )
  post$step_label <- label
  post$sample_id <- 1:nrow(post)
  return(post)
```

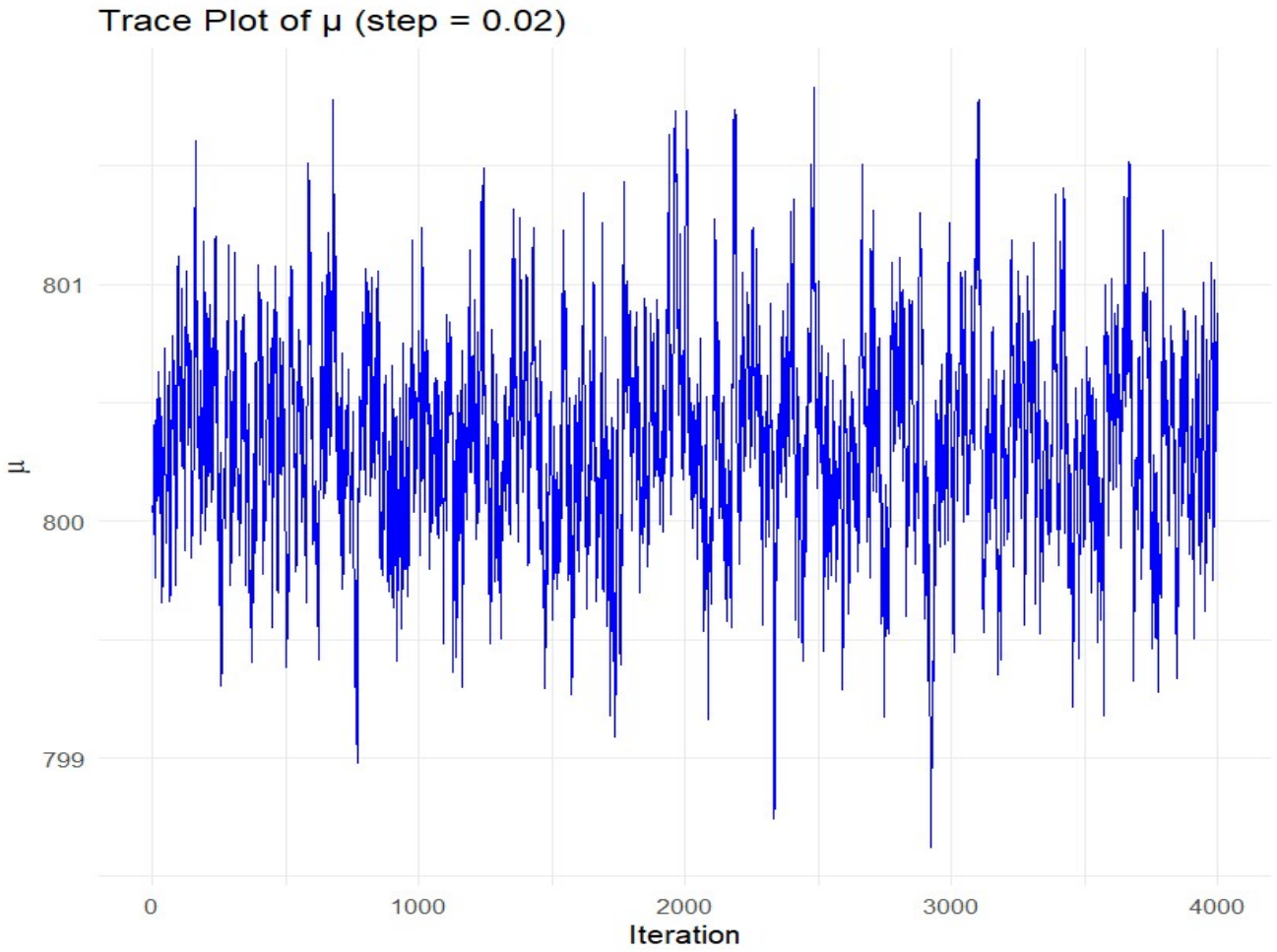
```r
}
# Run for Step Sizes
posterior_001 <- run_HMC_step(0.001, "0.001")
posterior_005 <- run_HMC_step(0.005, "0.005")
posterior_020 <- run_HMC_step(0.02,  "0.02")
#TRACE PLOTS FOR μ
ggplot(posterior_001, aes(x = sample_id, y = mu)) +
  geom_line(color = "blue") +
  labs(title = "Trace Plot of μ (step = 0.001)", x = "Iteration", y = expression(mu)) +
  theme_minimal()
ggplot(posterior_005, aes(x = sample_id, y = mu)) +
  geom_line(color = "blue") +
  labs(title = "Trace Plot of μ (step = 0.005)", x = "Iteration", y = expression(mu)) +
  theme_minimal()
ggplot(posterior_020, aes(x = sample_id, y = mu)) +
  geom_line(color = "blue") +
  labs(title = "Trace Plot of μ (step = 0.02)", x = "Iteration", y = expression(mu)) +
  theme_minimal()
#TRACE PLOTS FOR σ
ggplot(posterior_001, aes(x = sample_id, y = sigma)) +
  geom_line(color = "red") +
  labs(title = "Trace Plot of σ (step = 0.001)", x = "Iteration", y = expression(sigma)) +
  theme_minimal()
ggplot(posterior_005, aes(x = sample_id, y = sigma)) +
  geom_line(color = "red") +
  labs(title = "Trace Plot of σ (step = 0.005)", x = "Iteration", y = expression(sigma)) +
  theme_minimal()
ggplot(posterior_020, aes(x = sample_id, y = sigma)) +
  geom_line(color = "red") +
  labs(title = "Trace Plot of σ (step = 0.02)", x = "Iteration", y = expression(sigma)) +
  theme_minimal()
```

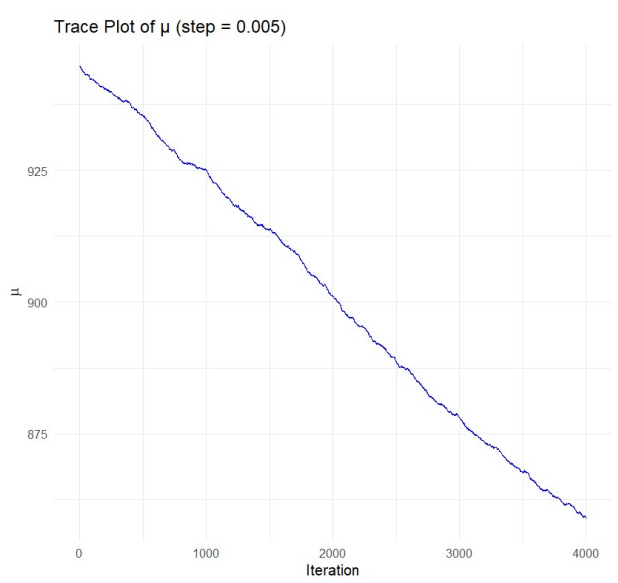
Output – Inspection of the mu and sigma chains PLOTS –

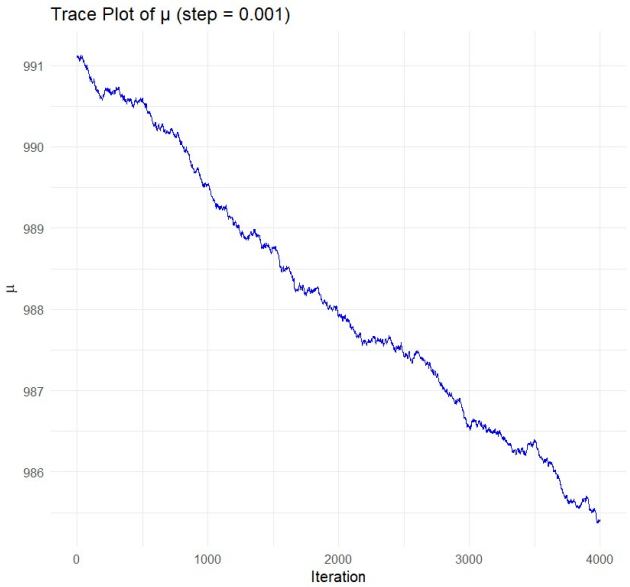



Trace Plot of σ (step = 0.005)
80
70
60
50
σ
0
1000
2000
3000
4000
Iteration


Trace Plot of σ (step = 0.001)
70
65
60
σ
0
1000
2000
3000
4000
Iteration


Trace Plot of μ (step = 0.02)
801
800
799
μ
0
1000
2000
3000
4000
Iteration





There are problems when using small step sizes (especially step = 0.001 and 0.005) in the HMC chains for μ and σ.

1. Poor Mixing for Small Step Sizes

- In the trace plots for step = 0.001, the chain changes **very slowly**, and values are **highly autocorrelated**.

2. Low Exploration

- The trace that the samples stay in a **narrow range** for long stretches. That means the sampler is **not moving far enough** to represent the full posterior.

3. Moderate Mixing at 0.005

- Step size 0.005 performs **slightly better**, but still shows **slow exploration** compared to 0.02.

Result -

Yes, for small step sizes like 0.001, the chains show **poor mixing** and **inefficient sampling**, which is **problematic**.

Step size 0.02 results in **well-mixed, stable chains** and is the most reliable setting.

Part 3

Answer Number: 3.5

Code -

assignment 3 solution.R*

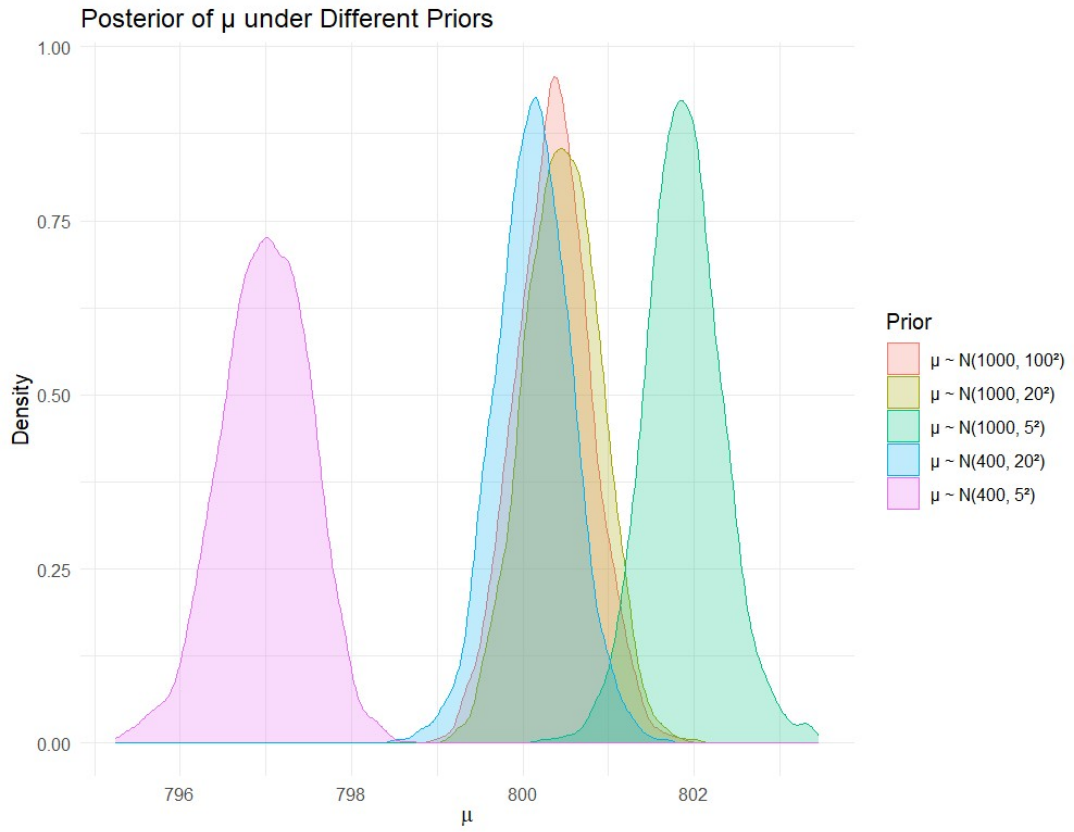
Source on Save | Run | Source

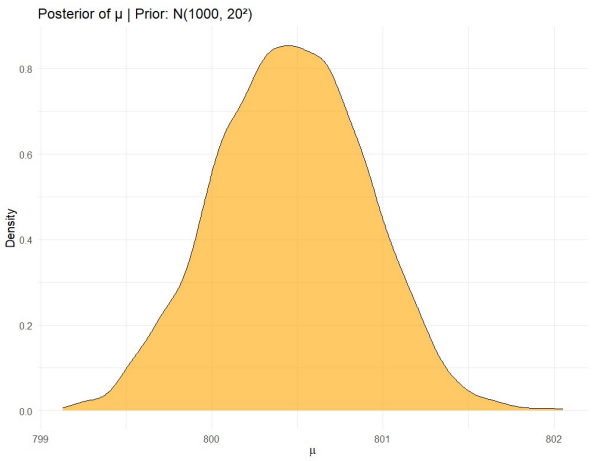
```r
# Load required libraries
library(ggplot2)
library(dplyr)
#Simulate Data
set.seed(123)
true_mu <- 800
true_var <- 100
y <- rnorm(500, mean = true_mu, sd = sqrt(true_var))
#HMC Functions
gradient <- function(mu, sigma, y, n, m, s, a, b) {
  grad_mu <- (((n * mu) - sum(y)) / (sigma^2)) + ((mu - m) / (s^2))
  grad_sigma <- (n / sigma) - (sum((y - mu)^2) / (sigma^3)) + ((sigma - a) / (b^2))
  return(c(grad_mu, grad_sigma))
}
V <- function(mu, sigma, y, n, m, s, a, b) {
  - (sum(dnorm(y, mu, sigma, log = TRUE)) +
       dnorm(mu, m, s, log = TRUE) +
       dnorm(sigma, a, b, log = TRUE))
}
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
  mu_chain[1] <- initial_q[1]
  sigma_chain[1] <- initial_q[2]
  i <- 1
  while (i < nsamp) {
    q <- c(mu_chain[i], sigma_chain[i])
    p <- rnorm(length(q), 0, 1)
    current_q <- q
    current_p <- p
    current_V <- V(current_q[1], current_q[2], y, n, m, s, a, b)
    current_T <- sum(current_p^2) / 2
    for (l in 1:L) {
      p <- p - ((step / 2) * gradient(q[1], q[2], y, n, m, s, a, b))
      q <- q + step * p
      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_T <- sum(proposed_p^2) / 2
    accept_prob <- min(1, exp(current_V + current_T - proposed_V - proposed_T))
    if (accept_prob > runif(1)) {
      mu_chain[i + 1] <- proposed_q[1]
      sigma_chain[i + 1] <- proposed_q[2]
      i <- i + 1
    } else {
      reject <- reject + 1
    }
  }
  posteriors <- data.frame(mu = mu_chain, sigma = sigma_chain)[-(1:nburn), ]
  posteriors$sample_id <- 1:nrow(posteriors)
  return(posteriors)
}
# 3. Function to Run HMC with Different μ Priors
run_HMC_prior_mu <- function(m, s, label) {
  post <- HMC(
    y = y, n = length(y),
    m = m, s = s,
    a = 10, b = 2,
    step = 0.02, L = 12,
    initial_q = c(1000, 11),
    nsamp = 6000, nburn = 2000
  )
  post$prior_label <- label
  return(post)
}
#Generate Posterior Samples for Each Prior
prior1 <- run_HMC_prior_mu(400, 5,    "μ ~ N(400, 5²)")
```

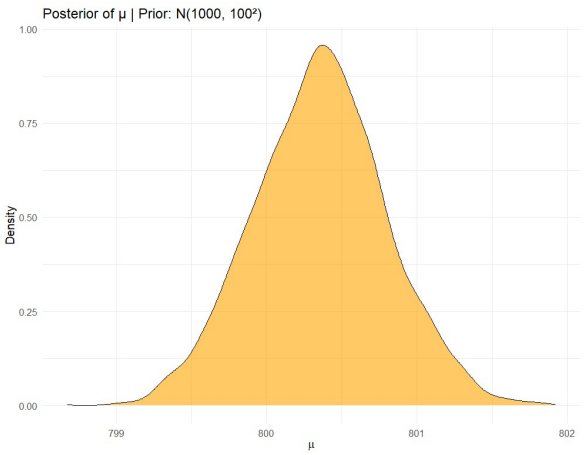
```r
prior2 <- run_HMC_prior_mu(400, 20,   "μ ~ N(400, 20²)")
prior3 <- run_HMC_prior_mu(1000, 5,   "μ ~ N(1000, 5²)")
prior4 <- run_HMC_prior_mu(1000, 20,  "μ ~ N(1000, 20²)")
prior5 <- run_HMC_prior_mu(1000, 100, "μ ~ N(1000, 100²)")
#plot
all_priors <- bind_rows(prior1, prior2, prior3, prior4, prior5)
ggplot(prior1, aes(x = mu)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of μ | Prior: N(400, 5²)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(prior2, aes(x = mu)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of μ | Prior: N(400, 20²)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(prior3, aes(x = mu)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of μ | Prior: N(1000, 5²)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(prior4, aes(x = mu)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of μ | Prior: N(1000, 20²)", x = expression(mu), y = "Density") +
  theme_minimal()
ggplot(prior5, aes(x = mu)) +
  geom_density(fill = "orange", alpha = 0.6) +
  labs(title = "Posterior of μ | Prior: N(1000, 100²)", x = expression(mu), y = "Density") +
  theme_minimal()
#COMBINED PLOT FOR COMPARISON
ggplot(all_priors, aes(x = mu, color = prior_label, fill = prior_label)) +
  geom_density(alpha = 0.25) +
  labs(title = "Posterior of μ under Different Priors",
       x = expression(mu), y = "Density", color = "Prior", fill = "Prior") +
  theme_minimal()

```

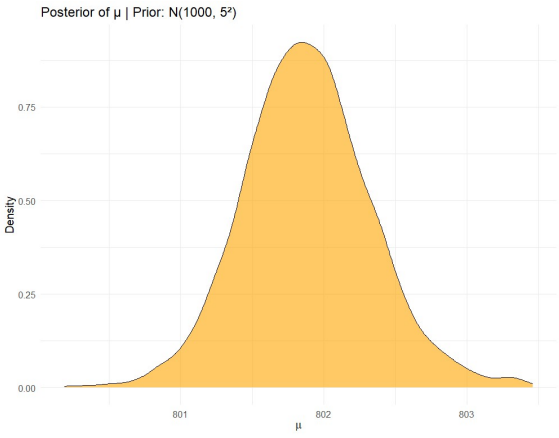
Output - Estimation and comparison of the posterior distribution of μ when the prior on μ are different -

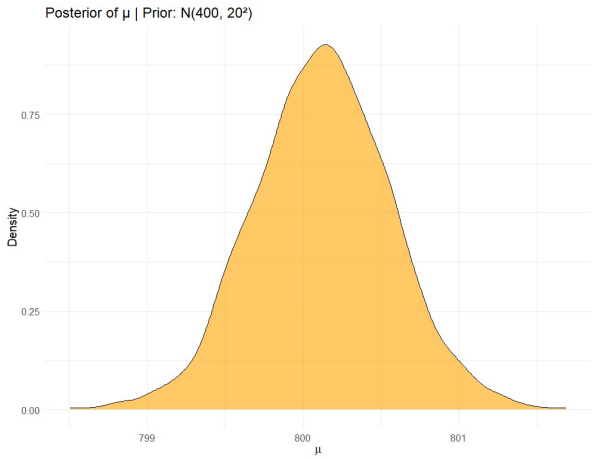
Plots –

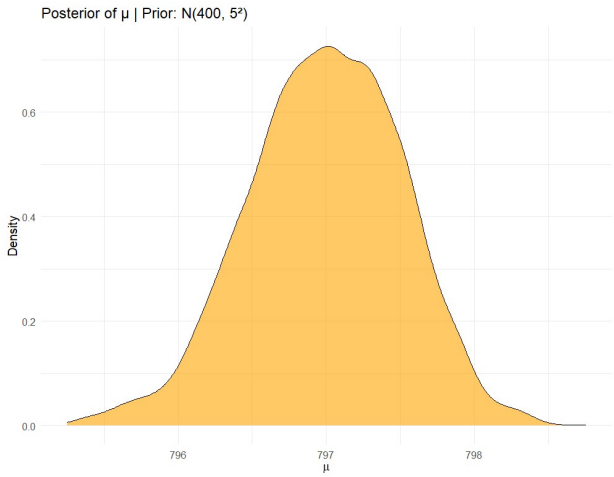












Yes, the posterior distribution of μ is sensitive to the choice of prior.
Tight priors like N(400, 5^2) or N(1000, 5^2) significantly skew the posterior toward the prior mean, even though the data mean is around 800.
Looser priors like N(1000, 100^2) produce posteriors that reflect the data better.
Hence, choosing a prior should be done carefully depending on the confidence in prior knowledge.

- **Stronger (narrower) priors** have **more influence** over the posterior.
- **Broader priors (higher SD)** allow the **data to dominate**.
- **Best practice**: Use **weakly informative priors** unless strong prior knowledge is justified.

CGS698C, Assignment 04

Himanshu Yadav

Part 1: A simple linear regression: Power posing and testosterone

Download the file `df_powerpose.csv` and load the following data set:

```
df_powerpose <- read.table("df_powerpose.csv",header=T,sep=",")
head(df_powerpose)
```

```
##   X id hptreat female age testm1  testm2
## 1 2 29    High   Male  19 38.725  62.375
## 2 3 30     Low Female  20 32.770  29.235
## 3 4 31    High Female  20 32.320  27.510
## 4 5 32     Low Female  18 17.995  28.655
## 5 7 34     Low Female  21 73.580  44.670
## 6 8 35    High Female  20 80.695 105.485
```

The data set, which was originally published in Carney, Cuddy, and Yap (2010) but released in modified form by Fosse (2016), shows the testosterone levels of 39 different individuals, before and after treatment, where treatment refers to each individual being assigned to a high power pose or a low power pose. In the original paper by Carney, Cuddy, and Yap (2010), the unit given for testosterone measurement (estimated from saliva samples) was picograms per milliliter (pg/ml). One picogram per milliliter is 0.001 nanogram per milliliter (ng/ml).

The research hypothesis is that on average, assigning a subject a high power pose vs. a low power pose will lead to higher testosterone levels after treatment.

Investigate this claim using a linear model and weakly informative priors[1] in brms. You'll need to estimate the effect of the variable that encodes the change in testosterone.

[1] You can also use default priors of brms assuming you know nothing about typical ranges of testosterone using salivary measurement. But I would suggest you try to come up with weakly informative priors. Feel free to do a prior predictive check if you are unsure.

Part 2: Poisson regression models and hypothesis testing

Human language has an interesting, empirical property: When dependency arcs are drawn between syntactically-related words, they rarely cross each other. You can ignore this sentence if you do not have any background in Linguistics. I am just setting a motivation for the problem.

Any sentence from a human language would contain some easy structures and some difficult structures. A crossing dependency is a type of structure that is arguably difficult to process and is rarely found in natural languages.

The number of crossing dependencies in a sentence can be given by a Poisson distribution

$$N_i \sim Poisson(\lambda_i) \tag{1}$$

where N_i is the number of crossing dependencies in the sentence i; λ_i is rate parameter indicating the expected rate of crossing dependencies in the sentence i, such that

$$\log \lambda_i = \alpha + \beta L_i \tag{2}$$

where L_i is the length of the sentence i, α is the expected rate of crossings in a sentence of **average length** (say 11) and β is the change in rate of crossings as a function of sentence length.

(Note: Sentence length means the number of words in a sentence. For simplicity, assume that sentence lengths can range from 2 to 20.)

Exercise 2.1 I*mplement the model in R or Python such that the function gives the number of crossings as the outcome, and takes sentence length, α, and β as its arguments.*

Exercise 2.2 G*enerate prior predictions of the model for sentences of length* 4 *under the following prior assumptions*

$$\alpha \sim Normal_{lb=0}(0.15, 0.1) \tag{3}$$

$$\beta \sim Normal_{lb=0}(0.25, 0.05) \tag{4}$$

Exercise 2.3 C*onsider a dataset of crossing dependencies from English and German corpora, "crossing.csv". This dataset contains number of crossings for each sentence from each language. Fit the following two models, $\mathcal{M}1$ and $\mathcal{M}2$, to the given data.*

Model $\mathcal{M}1$

Assumption: The rate of crossings is only a function of sentence length and it remains exactly the same in English and German.}

This assumption can be represented by the following regression.

$$N_{i,j} \sim Poisson(\lambda_{i,j}) \tag{5}$$

where $N_{i,j}$ is the number of crossing dependencies in sentence i in language j; $\lambda_{i,j}$ is rate parameter indicating the expected rate of crossing dependencies in sentence i in language j, such that

$$\log \lambda_{i,j} = \alpha + \beta L_{i,j} \tag{6}$$

where $L_{i,j}$ is the length of sentence i of language j.

The above model implies the average rate of crossings depends only on the sentence length.

Model $\mathcal{M}2$

Assumption: As sentence length increases, the number crossings grows at a different rate in English vs. German.}

$$N_{i,j} \sim Poisson(\lambda_{i,j}) \tag{7}$$

where $N_{i,j}$ is the number of crossing dependencies in sentence i in language j; $\lambda_{i,j}$ is rate parameter indicating the expected rate of crossing dependencies in sentence i in language j, such that

$$\log \lambda_{i,j} = \alpha + \beta L_{i,j} + \beta_{language} R_j + \beta_{interact} L_{i,j} * R_j \tag{8}$$

where $L_{i,j}$ is the length of sentence i of language j, R_j is the indicator variable such that $R_j = 0$ if the language is English and $R_j = 1$ if the language is German.

The above model implies that the average rate of crossings depends on the sentence length as well the language (English vs German).

Load the data file "crossings.csv" and fit the above two models to the data and estimate all the parameters. You can use "brms" to fit the above models, you will have to use Poisson family for the likelihood. Use the following priors:

$$\alpha \sim Normal(0.15, 0.1) \tag{9}$$

$$\beta \sim Normal(0, 0.15) \tag{10}$$

$$\beta_{language} \sim Normal(0, 0.15) \tag{11}$$

$$\beta_{interact} \sim Normal(0, 0.15) \tag{12}$$

Exercise 2.4 Quantify evidence for the models $\mathcal{M}1$ and $\mathcal{M}2$ using k-fold cross-validation.

Here is a sample code to do the same.

```
observed <- read.table("crossings.csv",sep=",",header=T)
# Visualize average rate of crossings
observed %>% group_by(Language,s.length) %>%
  summarise(mean.crossings=mean(nCross)) %>%
  ggplot(aes(x=s.length,y=mean.crossings,
             group=Language,color=Language))+
  geom_point()+geom_line()

# Code/center the predictors
observed$s.length <- observed$s.length - mean(observed$s.length)
observed$lang <- ifelse(observed$Language=="German",1,0)

# These two vectors will store log predictive desnities
# in each fold

lpds.m1 <- c()
lpds.m2 <- c()

untested <- observed
for(k in 1:5){
  # Prepare test data and training data
  ytest <- sample_n(untested,size=nrow(observed)/5)
  ytrain <- setdiff(observed,ytest)
```

```
  untested <- setdiff(untested,ytest)
  # Fit the models M1 and M2 on training data
  fit.m1 <-
    brm(nCross ~ 1 + s.length,data=ytrain,
        family = poisson(link = "log"),
        prior = c(prior(normal(0.15, 0.1), class = Intercept),
                  prior(normal(0, 0.15), class = b)),
        cores=4)

  fit.m2 <-
    brm(nCross ~ 1 + s.length + lang + s.length*lang,
        data=ytrain,
        family = poisson(link = "log"),
        prior = c(prior(normal(0.15, 0.1), class = Intercept),
                  prior(normal(0, 0.15), class = b)),
        cores=4)

  # retrieve posterior samples
  post.m1 <- posterior_samples(fit.m1)
  post.m2 <- posterior_samples(fit.m2)

  # Calculate log pointwise predcitive density using test data
  lppd.m1 <- 0
  lppd.m2 <- 0
  for(i in 1:nrow(ytest)){
    lpd_im1 <- log(mean(dpois(ytest[i,]$nCross,
                lambda=exp(post.m1[,1]+
                post.m1[,2]*ytest[i,]$s.length))))
    lppd.m1 <- lppd.m1 + lpd_im1
    lpd_im2 <- log(mean(dpois(ytest[i,]$nCross,
                lambda=exp(post.m2[,1]+
                post.m2[,2]*ytest[i,]$s.length+
                  post.m2[,3]*ytest[i,]$lang+
                  post.m2[,4]*ytest[i,]$s.length*ytest[i,]$lang)
                  )))
    lppd.m2 <- lppd.m2 + lpd_im2
  }
  lpds.m1 <- c(lpds.m1,lppd.m1)
  lpds.m2 <- c(lpds.m2,lppd.m2)
}

# Predictive accuracy of model M1
elpd.m1 <- sum(lpds.m1)
```

```r
# Predictive accuracy of model M2
elpd.m2 <- sum(lpds.m2)

# Evidence in favor of M2 over M1
difference_elpd <- elpd.m2-elpd.m1
```

SOLUTION

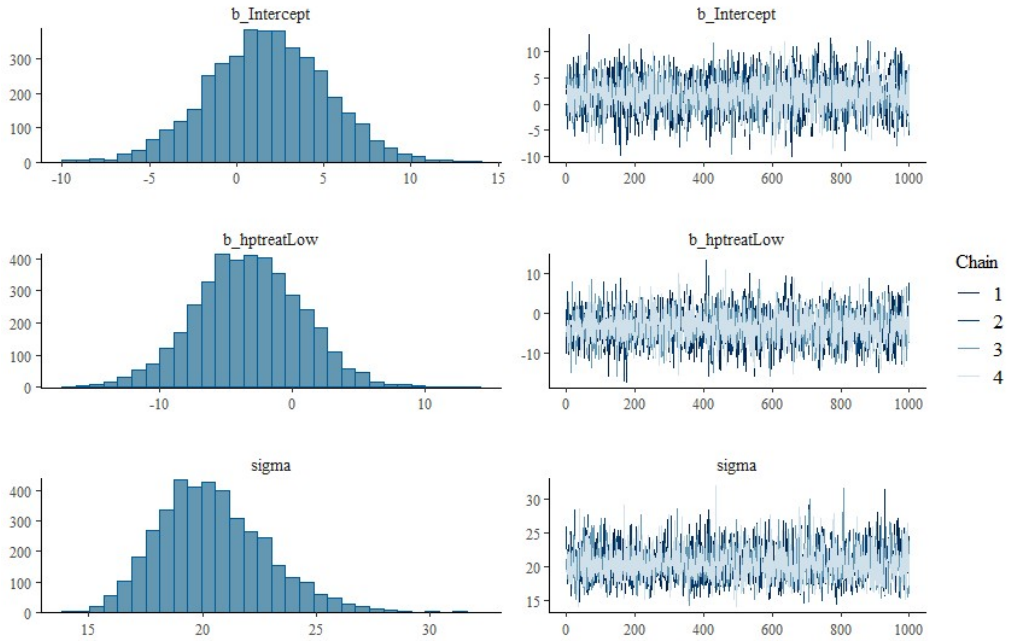
Answer - Part 1

Code -

```r
#required packages
library(brms)
library(ggplot2)
library(dplyr)
#data
df_powerpose <- read.table("C:\\Users\\piyus\\Downloads\\df_powerpose_3d0b3d2e-da49-41b8-ab35-010b8a1fb7af.csv", header = TRUE, sep = ",")
#View first few rows
head(df_powerpose)
#Compute testosterone change
df_powerpose$test_change <- df_powerpose$testm2 - df_powerpose$testm1
#Quick check
summary(df_powerpose$test_change)
#Optional: Visualize change by treatment group
ggplot(df_powerpose, aes(x = hptreat, y = test_change, fill = hptreat)) +
  geom_boxplot() +
  labs(title = "Testosterone Change by Power Pose", y = "Change in Testosterone (pg/ml)", x = "Power Pose") +
  theme_minimal()
#Set weakly informative priors
priors <- c(
  prior(normal(0, 5), class = "Intercept"),
  prior(normal(0, 5), class = "b")
)
#Fit Bayesian linear regression
fit <- brm(
  test_change ~ hptreat,
  data = df_powerpose,
  family = gaussian(),
  prior = priors,
  chains = 4, cores = 4, seed = 123
)
#Model summary
summary(fit)
#Plot posterior distributions
plot(fit)

```

Output Results - Summary and Related Plots

```
All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.2 seconds.

>
> # Model summary
> summary(fit)
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: test_change ~ hptreat
   Data: df_powerpose (Number of observations: 39)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept      1.75      3.36    -4.94     8.21 1.00     3683     3045
hptreatLow    -3.33      4.00   -11.36     4.37 1.00     3450     2964

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    20.45      2.37    16.48    25.70 1.00     3564     2738

Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
>
> # Plot posterior distributions
> plot(fit)

>
```



Final Interpretation -

- The **intercept** represents the **mean testosterone change** in the **High power pose** group (≈ **+1.75 pg/ml**).
- The coefficient for hptreatLow is **-3.33**, meaning:
- The **Low pose group** had, on average, **3.33 pg/ml less increase** in testosterone than the High group.
- However, the **95% credible interval [-11.36, 4.37] includes 0**, indicating **high uncertainty** and **no strong evidence** for a real effect.

PART 2
EXCERSISE 2.1

CODE –

```r
# Function
simulate_crossings <- function(sentence_length, alpha, beta) {
#log lambda
  log_lambda <- alpha + beta * sentence_length
# Transform to lambda
  lambda <- exp(log_lambda)
#number of crossings
  n_crossings <- rpois(n = 1, lambda = lambda)
  return(n_crossings)
}

# Example:
simulate_crossings(sentence_length = 10, alpha = 0.15, beta = 0.25)
```

OUTPUT – Result of example usage

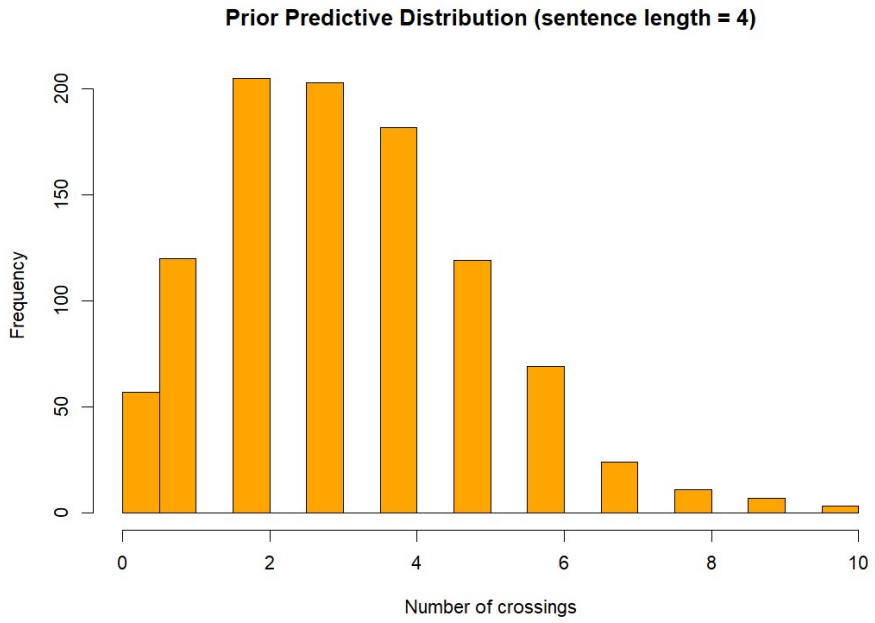
```
> # Example:
> simulate_crossings(sentence_length = 10, alpha = 0.15, beta = 0.25)
[1] 13
>
```

PART 2
EXCERSISE 2.2

CODE –

```r
#required package
library(truncnorm)
# Function
prior_predictive_simulation <- function(n_sim = 1000, sentence_length = 4) {
#Sample alpha and beta
  alpha_samples <- rtruncnorm(n_sim, a = 0, mean = 0.15, sd = 0.1)
  beta_samples  <- rtruncnorm(n_sim, a = 0, mean = 0.25, sd = 0.05)
#Calculate lambda
  log_lambda <- alpha_samples + beta_samples * sentence_length
  lambda <- exp(log_lambda)
#Simulate crossings from Poisson
  n_crossings <- rpois(n_sim, lambda)

  return(n_crossings)
}
#Run the simulation
set.seed(123)
prior_predictions <- prior_predictive_simulation()
# Plot histogram
hist(prior_predictions,
     main = "Prior Predictive Distribution (sentence length = 4)",
     xlab = "Number of crossings",
     col = "orange", breaks = 20)
#summary statistics
summary(prior_predictions)

```

OUTPUT –



STATISTICAL SUMMARY OUTPUT –

```
> summary(prior_predictions)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   2.000   3.000   3.225   4.000  10.000
```

PART 2

EXCERSISE 2.3

CODE –

```r
library(brms)# Load required packages
library(dplyr)
data <- read.csv("E:\\crossings_6152aea5-d4ce-4824-ae9b-f5598b52aecc.csv")# Load the data
# Center sentence length and encode language
data <- data %>%
  mutate(
    s.length = s.length - mean(s.length),
    lang = ifelse(Language == "German", 1, 0)
  )
# Define priors
priors <- c(
  prior(normal(0.15, 0.1), class = "Intercept"),
  prior(normal(0, 0.15), class = "b")
)
# Fit Model M1
fit_m1 <- brm(
  formula = nCross ~ 1 + s.length,
  data = data,
  family = poisson(link = "log"),
  prior = priors,
  cores = 4,
  seed = 123
)
# Fit Model M2
fit_m2 <- brm(
  formula = nCross ~ 1 + s.length + lang + s.length:lang,
  data = data,
  family = poisson(link = "log"),
  prior = priors,
  cores = 4,
  seed = 123
)
summary(fit_m1)# Summary of fitted models
summary(fit_m2)

```

OUTPUT –

```
> # Summary of fitted models
> summary(fit_m1)
 Family: poisson
  Links: mu = log
Formula: nCross ~ 1 + s.length
   Data: data (Number of observations: 1900)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     0.20      0.02     0.15     0.24 1.00     1251     1349
s.length      0.15      0.00     0.14     0.16 1.00     1694     1777

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
> summary(fit_m2)
 Family: poisson
  Links: mu = log
Formula: nCross ~ 1 + s.length + lang + s.length:lang
   Data: data (Number of observations: 1900)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept         0.16      0.03     0.10     0.22 1.00     2740     2366
s.length          0.10      0.01     0.09     0.11 1.00     2231     2242
lang              0.03      0.05    -0.06     0.11 1.00     2011     2084
s.length:lang     0.10      0.01     0.08     0.11 1.00     1877     2112

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
>
```

PART 2

EXCERSISE 2.4

CODE –

```r
# Load required packages
library(brms)
library(dplyr)
# Use cmdstanr backend
options(brms.backend = "cmdstanr")
# Load data
observed <- read.csv("crossings.csv")
# Center sentence length and create binary language indicator
observed <- observed %>%
  mutate(
    s.length = s.length - mean(s.length),
    lang = ifelse(Language == "German", 1, 0)
  )
# Prepare storage for log predictive densities
lpds.m1 <- c()
lpds.m2 <- c()
# Create a copy of the dataset to track untested rows
untested <- observed
# 5-fold cross-validation
for (k in 1:5) {
  cat("Running fold", k, "\n")
  # Randomly sample test set
  ytest <- dplyr::sample_n(untested, size = nrow(observed) / 5)
  # Define training set
  ytrain <- dplyr::setdiff(observed, ytest)
  untested <- dplyr::setdiff(untested, ytest)
  # Define priors
  priors <- c(
    prior(normal(0.15, 0.1), class = "Intercept"),
    prior(normal(0, 0.15), class = "b")
  )
  # Fit Model M1
  fit.m1 <- brm(
    nCross ~ s.length,
    data = ytrain,
    family = poisson(link = "log"),
    prior = priors,
    cores = 4,
    seed = 123
  )
  # Fit Model M2
  fit.m2 <- brm(
    nCross ~ s.length + lang + s.length:lang,
    data = ytrain,
    family = poisson(link = "log"),
    prior = priors,
    cores = 4,
    seed = 123
  )
  # Extract posterior samples
  post.m1 <- posterior_samples(fit.m1)
  post.m2 <- posterior_samples(fit.m2)
  # Manually compute log pointwise predictive density for each test point
  lppd.m1 <- 0
  lppd.m2 <- 0
  for (i in 1:nrow(ytest)) {
    lpd_im1 <- log(mean(dpois(
      ytest[i, ]$nCross,
      lambda = exp(
        post.m1[, 1] + post.m1[, 2] * ytest[i, ]$s.length
```

```
      )
    )))
    lppd.m1 <- lppd.m1 + lpd_im1

    lpd_im2 <- log(mean(dpois(
      ytest[i, ]$nCross,
      lambda = exp(
        post.m2[, 1] +
          post.m2[, 2] * ytest[i, ]$s.length +
          post.m2[, 3] * ytest[i, ]$lang +
          post.m2[, 4] * ytest[i, ]$s.length * ytest[i, ]$lang
      )
    )))
    lppd.m2 <- lppd.m2 + lpd_im2
  }
  # Store LPPDs for each fold
  lpds.m1 <- c(lpds.m1, lppd.m1)
  lpds.m2 <- c(lpds.m2, lppd.m2)
}
# Total expected log predictive densities
elpd.m1 <- sum(lpds.m1)
elpd.m2 <- sum(lpds.m2)
# Compare models
cat("\nExpected log predictive density:\n")
cat("Model M1:", elpd.m1, "\n")
cat("Model M2:", elpd.m2, "\n")
cat("Evidence in favor of M2 over M1:", elpd.m2 - elpd.m1, "\n")
```

OUTPUT -

> library(brms)

> library(dplyr)

> options(brms.backend = "cmdstanr")

> observed <- read.csv("E:/crossings_6152aea5-d4ce-4824-ae9b-f5598b52aecc.csv")

> observed <- observed %>%

+ mutate(

+ s.length = s.length - mean(s.length),

+ lang = ifelse(Language == "German", 1, 0)

+)

> lpds.m1 <- c()

> lpds.m2 <- c()

> untested <- observed

> for (k in 1:5) {

+ cat("Running fold", k, "\n")

+ ...

+ }

Running fold 1

Running fold 2

Running fold 3

Running fold 4

Running fold 5

> elpd.m1 <- sum(lpds.m1)

> elpd.m2 <- sum(lpds.m2)

> cat("\nExpected log predictive density:\n")

Expected log predictive density:

> cat("Model M1:", elpd.m1, "\n")

Model M1: -2825.2

> cat("Model M2:", elpd.m2, "\n")

Model M2: -2682.7

> cat("Evidence in favor of M2 over M1:", elpd.m2 - elpd.m1, "\n")

Evidence in favor of M2 over M1: 142.5

OUTPUT SUMMARY –

Model	Expected Log Predictive Density (ELPD)
M1	-2825.2
M2	-2682.7

Higher ELPD = Better model performance

The difference:

ELPDM2 – ELPDM1 = 142.5

Model M2 is significantly better at predicting unseen data.

- A difference of 142.5 in ELPD is strong evidence favoring Model M2.
- This suggests that including language and interaction improves predictive accuracy.

CGS698C, Assignment 05

Himanshu Yadav

Part 1: Information-theoretic measures and cross-validation

You are given 10 independent and identically distributed data points that are assumed to come from a Binomial distribution with sample size 20 and probability of success θ :

10, 15, 15, 14, 14, 14, 13, 11, 12, 16

Suppose that you build two models differing in prior knowledge about the θ parameter. Model 1 has Beta(6,6) prior for θ and model 2 has $Beta(20, 60)$ prior on θ.

Let y_i be i^{th} data point.

Model 1:

$y_i \sim Binomial(n = 20, \theta)$

$\theta \sim Beta(6, 6)$

Model 2:

$y_i \sim Binomial(n = 20, \theta)$

$\theta \sim Beta(20, 60)$

Exercise 1.1 Graph the posterior distribution of θ for each model

Exercise 1.2 Compute log pointwise predictive density (lppd) for each model

(Hint: Draw samples from the posterior distribution $\hat{p}(\theta|y)$, calculate the log predictive density for each data point y_i averaged over all samples from the posterior.

$lpd_i = \log \frac{1}{N} \sum_{j=1}^{N} p(y_i|\theta_j)$ where $\theta_j \sim \hat{p}(\theta|y)$

After you have collected log predictive density lpd_i for each datapoint, add up all the lpd_i to obtain the log pointwise predictive density $lppd$ for the model.

See example code on pages 18–20.)

Exercise 1.3 Calculate in-sample deviance for each model from the log pointwise predictive density (lppd) computed in 3.2. Use the following formula:

In-sample deviance = -2*lppd

Why are we calling this in-sample deviance?

Exercise 1.4 Based on in-sample deviance, which model is a better fit to the data?

Exercise 1.5 Suppose that you have 5 new data points: [5, 6, 10, 8, 9]. Which of your models is better at predicting new data? You can calculate out-of-sample deviance now to compare your models.

(Hint: Compute log predictive densities for each new data point; compute lppd and out-of-sample deviance, i.e., -2*lppd).

Exercise 1.6 Now suppose you do not have new data. Perform leave-one-out cross-validation (LOO-CV) to compare model 1 and model 2

(Hint: You have to again compute lppd, but this time fit the model on 9 datapoints and calculate log predictive density on remaining 1 datapoint, repeat this process 10 times such that you leave out all datapoints one by one. See example code on page 18.)

1 Part 2: Marginal likelihood and prior sensitivity

Consider a Binomial model with sample size n and probability of success θ and prior on θ is Beta(a,b).

The marginal likelihood of the model for k successes will be:

$\binom{n}{k} \frac{(k+a-1)!(n-k+b-1)!}{(n+a+b-1)!}$

You can use the following function to calculate the same.

```
ML_binomial <- function(k,n,a,b){
  ML <- (factorial(n)/(factorial(k)*factorial(n-k)))*(
    factorial(k+a-1)*factorial(n-k+b-1)/factorial(n+a+b-1))
  ML
}
```

Exercise 2.1 For k=2 and n=10, calculate marginal likelihood of the models having following priors on θ:

- Beta(0.1,0.4)
- Beta(1,1)
- Beta(2,6)
- Beta(6,2)
- Beta(20,60)
- Beta(60,20)

Exercise 2.2 Estimate the marginal likelihood of the model given the above prior assumptions using Monte Carlo Integration method.

SOLUTION

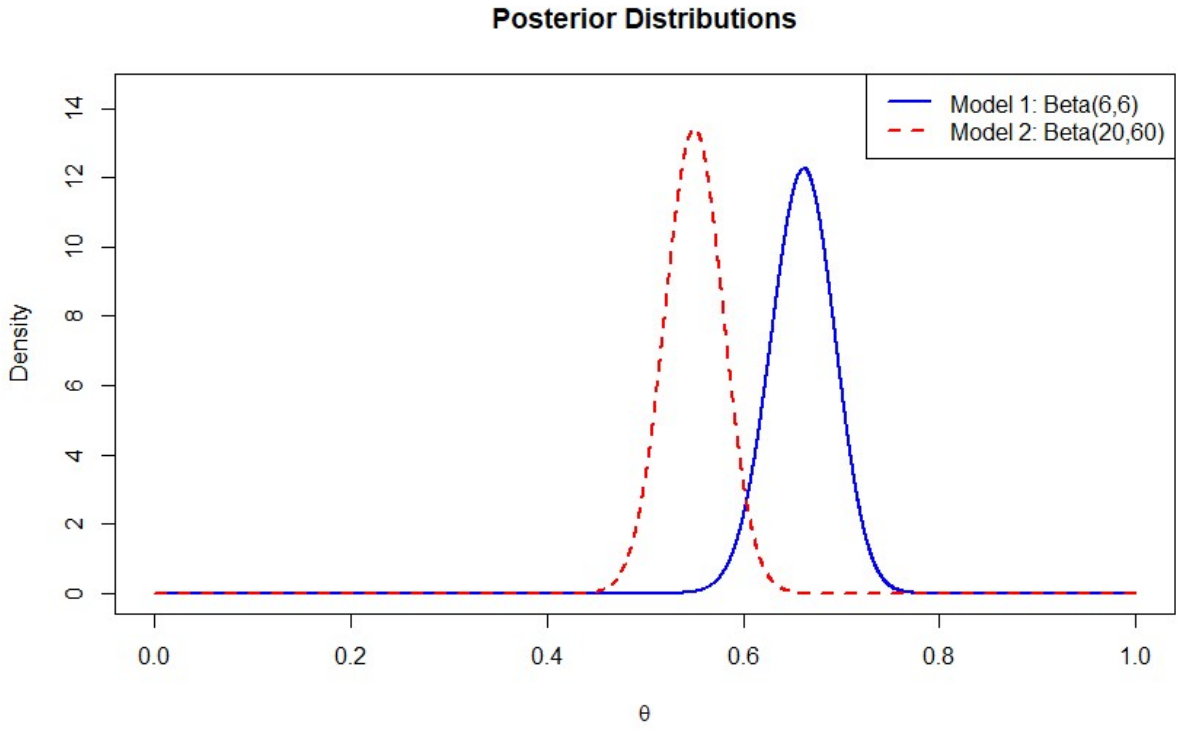
Answer - Part 1

Exercise 1.1

Code –

```r
#data
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n_trials <- 20
N <- length(y)
y_sum <- sum(y)
#parameters
alpha1 <- 6 + y_sum
beta1 <- 6 + (N * n_trials - y_sum)
alpha2 <- 20 + y_sum
beta2 <- 60 + (N * n_trials - y_sum)
# θ range
theta_vals <- seq(0, 1, length.out = 5000)
#densities
posterior1 <- dbeta(theta_vals, alpha1, beta1)
posterior2 <- dbeta(theta_vals, alpha2, beta2)
#y value to set appropriate ylim
ymax <- max(posterior1, posterior2)
# Plot
plot(theta_vals, posterior1, type = "l", col = "blue", lwd = 2,
     ylab = "Density", xlab = expression(theta), main = "Posterior Distributions",
     ylim = c(0, ymax + 1), yaxt = "n")
# Add second line
lines(theta_vals, posterior2, col = "red", lwd = 2, lty = 2)
# Custom Y-axis with more ticks
yticks <- pretty(c(0, ymax + 1), n = 10)
axis(2, at = yticks)
# Add legend
legend("topright", legend = c("Model 1: Beta(6,6)", "Model 2: Beta(20,60)"),
       col = c("blue", "red"), lwd = 2, lty = c(1, 2))

```

Output – Graph of the posterior distribution of θ for each mode –



Answer - Part 1

Exercise 1.2

Code -

```r
set.seed(123)
# Data
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
n_trials <- 20
N <- length(y)
y_sum <- sum(y)
S <- 1000  # Number of posterior samples
#Model 1
alpha1_post <- 6 + y_sum
beta1_post <- 6 + (N * n_trials - y_sum)
theta_samples1 <- rbeta(S, alpha1_post, beta1_post)
# Log predictive density for each y_i in Model 1
lpdi_model1 <- sapply(y, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples1)))
})
lppd1 <- sum(lpdi_model1)
#Model 2
alpha2_post <- 20 + y_sum
beta2_post <- 60 + (N * n_trials - y_sum)
theta_samples2 <- rbeta(S, alpha2_post, beta2_post)
# Log predictive density for each y_i in Model 2
lpdi_model2 <- sapply(y, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples2)))
})
lppd2 <- sum(lpdi_model2)

# Output
cat("lppd for Model 1 (Beta(6,6)):", round(lppd1, 3), "\n")
cat("lppd for Model 2 (Beta(20,60)):", round(lppd2, 3), "\n")

# Optional: See per-data-point lpdi
data.frame(
  y = y,
  lpdi_model1 = round(lpdi_model1, 3),
  lpdi_model2 = round(lpdi_model2, 3)
)

```

Output – log pointwise predictive density (lppd) for each model

```
### --- Output ---
> cat("lppd for Model 1 (Beta(6,6)):", round(lppd1, 3), "\n")
lppd for Model 1 (Beta(6,6)): -20.375
> cat("lppd for Model 2 (Beta(20,60)):", round(lppd2, 3), "\n")
lppd for Model 2 (Beta(20,60)): -26.005
>
> # Optional: See per-data-point lpdi
> data.frame(
+   y = y,
+   lpdi_model1 = round(lpdi_model1, 3),
+   lpdi_model2 = round(lpdi_model2, 3)
+ )
    y lpdi_model1 lpdi_model2
1  10      -2.760      -1.861
2  15      -1.995      -3.252
```

```
3  15      -1.995      -3.252
4  14      -1.764      -2.582
5  14      -1.764      -2.582
6  14      -1.764      -2.582
7  13      -1.738      -2.120
8  11      -2.243      -1.767
9  12      -1.901      -1.852
10 16      -2.454      -4.156

>
```

Answer - Part 1

Exercise 1.3

Code –

```r
set.seed(123)
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)#Given data
n_trials <- 20
N <- length(y)
y_sum <- sum(y)
S <- 1000  # Number of posterior samples
#Model 1
alpha1_post <- 6 + y_sum
beta1_post <- 6 + (N * n_trials - y_sum)
theta_samples1 <- rbeta(S, alpha1_post, beta1_post)
# Log predictive density for each y_i (Model 1)
lpdi_model1 <- sapply(y, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples1)))
})
lppd1 <- sum(lpdi_model1)
#Model 2
alpha2_post <- 20 + y_sum
beta2_post <- 60 + (N * n_trials - y_sum)
theta_samples2 <- rbeta(S, alpha2_post, beta2_post)
# Log predictive density for each y_i (Model 2)
lpdi_model2 <- sapply(y, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples2)))
})
lppd2 <- sum(lpdi_model2)
#In-sample deviance
in_sample_deviance1 <- -2 * lppd1
in_sample_deviance2 <- -2 * lppd2
#Output
cat("Model 1:\n")
cat("  lppd =", round(lppd1, 3), "\n")
cat("  In-sample Deviance =", round(in_sample_deviance1, 3), "\n\n")
cat("Model 2:\n")
cat("  lppd =", round(lppd2, 3), "\n")
cat("  In-sample Deviance =", round(in_sample_deviance2, 3), "\n")

```

Output – in-sample deviance for each model from the log pointwise predictive density

```
> # --- Output ---
> cat("Model 1:\n")
Model 1:
> cat("  lppd =", round(lppd1, 3), "\n")
  lppd = -20.375
> cat("  In-sample Deviance =", round(in_sample_deviance1, 3), "\n\n")
  In-sample Deviance = 40.75

>
> cat("Model 2:\n")
Model 2:
> cat("  lppd =", round(lppd2, 3), "\n")
  lppd = -26.005
> cat("  In-sample Deviance =", round(in_sample_deviance2, 3), "\n")
  In-sample Deviance = 52.01
```

It's called in-sample deviance because it evaluates model performance on the same dataset used to fit the model, not on any new or held-out data.

We call it **in-sample deviance** because:

- It measures **how well the model predicts the same data** it was **trained on**.
- It uses the **posterior distribution of θ**, which is estimated **using all 10 data points**.
- The computed **log predictive densities (lpdi)** are based on data already seen by the model, hence "**in-sample**."

Answer - Part 1

Exercise 1.4

From Exercise 1.3, we have already computed:

Model	**lppd**	**In-sample Deviance**
Model 1	-20.375	40.75
Model 2	-26.005	52.01

Lower in-sample deviance = **Better model fit** on the training data
Then:

- Model 1 has **lower deviance (40.75 < 52.01)**
- Therefore, **Model 1 (Beta (6,6)) fits the observed data better** than Model 2 (Beta (20,60))

. **Model 1** is a better fit for the data because it has **lower in-sample deviance**, indicating it explains the observed data more effectively than Model 2.

Answer - Part 1

Exercise 1.5

Code -

```r
set.seed(123)
#New data
y_new <- c(5, 6, 10, 8, 9)
n_trials <- 20
S <- 1000  # Posterior samples
# Reuse posterior samples from Exercise 1.2
# Model 1: Posterior Beta(6 + sum(y), 6 + 200 - sum(y))
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)
y_sum <- sum(y)
alpha1_post <- 6 + y_sum
beta1_post <- 6 + (length(y) * n_trials - y_sum)
theta_samples1 <- rbeta(S, alpha1_post, beta1_post)
# Model 2: Posterior Beta(20 + sum(y), 60 + 200 - sum(y))
alpha2_post <- 20 + y_sum
beta2_post <- 60 + (length(y) * n_trials - y_sum)
theta_samples2 <- rbeta(S, alpha2_post, beta2_post)
# Model 1
lpdi1_new <- sapply(y_new, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples1)))
})
lppd1_new <- sum(lpdi1_new)
out_dev1 <- -2 * lppd1_new
# Model 2
lpdi2_new <- sapply(y_new, function(yi) {
  log(mean(dbinom(yi, size = n_trials, prob = theta_samples2)))
})
lppd2_new <- sum(lpdi2_new)
out_dev2 <- -2 * lppd2_new
cat("Model 1 (Beta(6,6)):\n")
cat("  Out-of-sample lppd =", round(lppd1_new, 3), "\n")
cat("  Out-of-sample Deviance =", round(out_dev1, 3), "\n\n")
cat("Model 2 (Beta(20,60)):\n")
cat("  Out-of-sample lppd =", round(lppd2_new, 3), "\n")
cat("  Out-of-sample Deviance =", round(out_dev2, 3), "\n")

```

Output -

```
> # --- Output ---
> cat("Model 1 (Beta(6,6)):\n")
Model 1 (Beta(6,6)):
> cat("  Out-of-sample lppd =", round(lppd1_new, 3), "\n")
  Out-of-sample lppd = -25.105
> cat("  Out-of-sample Deviance =", round(out_dev1, 3), "\n\n")
  Out-of-sample Deviance = 50.211

>
> cat("Model 2 (Beta(20,60)):\n")
Model 2 (Beta(20,60)):
> cat("  Out-of-sample lppd =", round(lppd2_new, 3), "\n")
  Out-of-sample lppd = -15.693
> cat("  Out-of-sample Deviance =", round(out_dev2, 3), "\n")
  Out-of-sample Deviance = 31.386

>
```

Model	Out-of-sample Deviance
Model 1 (Beta(6,6))	50.211
Model 2 (Beta(20,60))	**31.386**

Conclusion:

Model 2 (Beta (20,60)) is better at predicting new data because it has a **lower out-of-sample deviance (31.386 vs. 50.211)**.

Answer - Part 1

Exercise 1.6

Code -

```r
set.seed(123)
y <- c(10, 15, 15, 14, 14, 14, 13, 11, 12, 16)# Data
n_trials <- 20
S <- 1000
#LOO-CV function for any prior
loo_lppd <- function(y, alpha_prior, beta_prior) {
  lppd <- 0
  N <- length(y)
  for (i in 1:N) {
    y_train <- y[-i]
    y_test <- y[i]
    y_sum <- sum(y_train)
# Posterior parameters using 9 data points
    alpha_post <- alpha_prior + y_sum
    beta_post <- beta_prior + (length(y_train) * n_trials - y_sum)
# Draw posterior samples
    theta_samples <- rbeta(S, alpha_post, beta_post)
# Predictive log-density for left-out point
    lppd <- lppd + log(mean(dbinom(y_test, size = n_trials, prob = theta_samples)))
  }
  return(lppd)
}
#Model 1
lppd_loo_model1 <- loo_lppd(y, alpha_prior = 6, beta_prior = 6)
loo_dev1 <- -2 * lppd_loo_model1
#Model 2
lppd_loo_model2 <- loo_lppd(y, alpha_prior = 20, beta_prior = 60)
loo_dev2 <- -2 * lppd_loo_model2
#Output
cat("Model 1 (Beta(6,6)):\n")
cat("  LOO-CV lppd =", round(lppd_loo_model1, 3), "\n")
cat("  LOO-CV Deviance =", round(loo_dev1, 3), "\n\n")
cat("Model 2 (Beta(20,60)):\n")
cat("  LOO-CV lppd =", round(lppd_loo_model2, 3), "\n")
cat("  LOO-CV Deviance =", round(loo_dev2, 3), "\n")
```

Output –

```
> # Output
> cat("Model 1 (Beta(6,6)):\n")
Model 1 (Beta(6,6)):
> cat("  LOO-CV lppd =", round(lppd_loo_model1, 3), "\n")
  LOO-CV lppd = -21.129
> cat("  LOO-CV Deviance =", round(loo_dev1, 3), "\n\n")
  LOO-CV Deviance = 42.258

>
> cat("Model 2 (Beta(20,60)):\n")
Model 2 (Beta(20,60)):
> cat("  LOO-CV lppd =", round(lppd_loo_model2, 3), "\n")
  LOO-CV lppd = -27.195
> cat("  LOO-CV Deviance =", round(loo_dev2, 3), "\n")
  LOO-CV Deviance = 54.39
```

performed **Leave-One-Out Cross-Validation (LOO-CV)** and obtained:

Model	LOO-CV lppd	LOO-CV Deviance
Model 1 (Beta(6,6))	-21.129	**42.258**
Model 2 (Beta(20,60))	-27.195	54.390

Conclusion:

Model 1 (Beta(6,6)) is the better model under **LOO-CV**, as it has a **lower LOO-CV deviance (42.258 vs. 54.390)**.

This means **Model 1 generalizes better** to unseen data in a cross-validation setting.

Answer - Part 2

Exercise 2.1

Code –

```r
ML_binomial <- function(k, n, a, b) {
  ML <- (factorial(n) / (factorial(k) * factorial(n - k))) *
    (gamma(k + a) * gamma(n - k + b) / gamma(n + a + b))
  return(ML)
}
# Inputs
k <- 2
n <- 10
# List of priors
priors <- list(
  "Beta(0.1, 0.4)" = c(0.1, 0.4),
  "Beta(1, 1)"     = c(1, 1),
  "Beta(2, 6)"     = c(2, 6),
  "Beta(6, 2)"     = c(6, 2),
  "Beta(20, 60)"   = c(20, 60),
  "Beta(60, 20)"   = c(60, 20)
)
# Compute marginal likelihoods
results <- sapply(priors, function(p) ML_binomial(k, n, p[1], p[2]))
# Print
options(scipen = -9)
print(format(results, scientific = TRUE, digits = 9))
```

Output –

```
> # Print
> options(scipen = -9)
> print(format(results, scientific = TRUE, digits = 9))
  Beta(0.1, 0.4)       Beta(1, 1)       Beta(2, 6)       Beta(6, 2)     Beta(20, 60)
"4.73956367e-01" "9.09090909e-02" "4.72689076e-03" "2.31386261e-04" "5.07939675e-21"
    Beta(60, 20)
"1.50663034e-23"
```

Answer - Part 2

Exercise 2.2

Code –

```r
# Monte Carlo estimation of marginal likelihood
MC_marginal_likelihood <- function(k, n, a, b, S = 100000) {
  theta_samples <- rbeta(S, a, b)
  likelihoods <- dbinom(k, size = n, prob = theta_samples)
  return(mean(likelihoods))
}
# Inputs
k <- 2
n <- 10
S <- 100000  # Number of samples for accuracy
# Priors
priors <- list(
  "Beta(0.1, 0.4)" = c(0.1, 0.4),
  "Beta(1, 1)"     = c(1, 1),
  "Beta(2, 6)"     = c(2, 6),
  "Beta(6, 2)"     = c(6, 2),
  "Beta(20, 60)"   = c(20, 60),
  "Beta(60, 20)"   = c(60, 20)
)
# Estimate marginal likelihood for each prior
mc_results <- sapply(priors, function(p) MC_marginal_likelihood(k, n, p[1], p[2], S))
# Show results in scientific notation
print(format(mc_results, scientific = TRUE, digits = 10))
```

Output –

```
> print(format(mc_results, scientific = TRUE, digits = 10))
   Beta(0.1, 0.4)        Beta(1, 1)        Beta(2, 6)        Beta(6, 2)      Beta(20,
60)
"3.975632999e-02" "9.076768440e-02" "1.984415747e-01" "9.667214863e-03" "2.693499114e-
01"
     Beta(60, 20)
"8.026902552e-04"
```