Homework 4 - Ankit

PSTAT 115, Summer 2023

___Due on Sep 13th ***, 2023 at 11:59 pm___

Problem 1. Frequentist Coverage of The Bayesian Posterior Interval.

Suppose that $y_1, .., y_n$ is an IID sample from a $Normal(\mu, 1)$. We wish to estimate μ.

1a. For Bayesian inference, we will assume the prior distribution $\mu \sim Normal(0, \frac{1}{\kappa_0})$ for all parts below. Remember, from lecture that we can interpret κ_0 as the pseudo-number of prior observations with sample mean $\mu_0 = 0$. State the posterior distribution of μ given $y_1, .., y_n$. Report the lower and upper bounds of the 95% quantile-based posterior credible interval for μ, using the fact that for a normal distribution with standard deviation σ, approximately 95% of the mass is between $\pm 1.96\sigma$.

To find the posterior distribution of μ given $y_1, \ldots, y_n$, we use Bayes' theorem:

$$\text{Posterior}(\mu|y_1, \ldots, y_n) \propto \text{Likelihood}(y_1, \ldots, y_n|\mu) \times \text{Prior}(\mu)$$

where: - Likelihood: $\text{Likelihood}(y_1, \ldots, y_n|\mu) \propto \prod_{i=1}^{n} \exp\left(-\frac{1}{2}(y_i - \mu)^2\right)$ - Prior: $\text{Prior}(\mu) \propto \exp\left(-\frac{\kappa_0}{2}\mu^2\right)$

The posterior distribution can be calculated by multiplying the likelihood and prior, resulting in a distribution for μ.

To find the posterior mode, we maximize the log posterior:

$$\frac{d}{d\mu} \log(\text{Posterior}(\mu|y_1, \ldots, y_n)) = \sum_{i=1}^{n}(\mu - y_i) - \kappa_0\mu = 0$$

Solving for μ:

$$\mu = \frac{\sum_{i=1}^{n} y_i}{n - \kappa_0}$$

The lower and upper bounds of the 95% quantile-based posterior credible interval for μ are approximately:

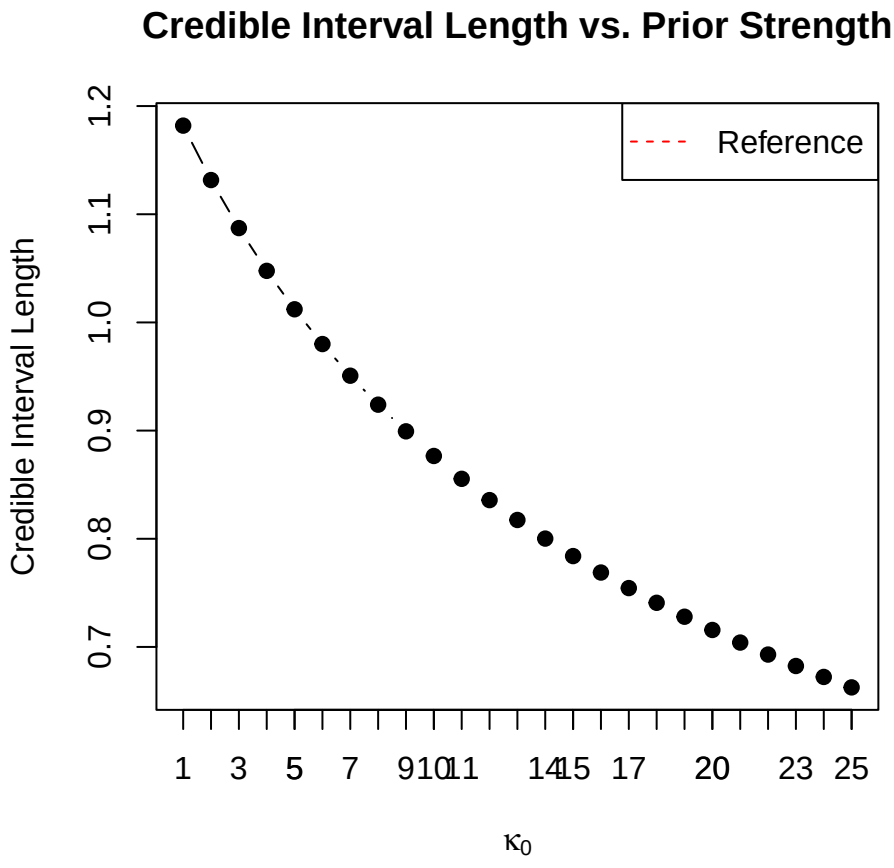
Lower Bound: $\mu - 1.96 \times$ Posterior SD

Upper Bound: $\mu + 1.96 \times$ Posterior SD

1b. Plot the length of the posterior credible interval as a function of κ_0, for $\kappa_0 = 1, 2, ..., 25$ assuming $n = 10$. Report how this prior parameter effects the length of the posterior interval and why this makes intuitive sense.

```
# range of kappa_0 values
kappa_0_values <- 1:25
y <- 1 # it wasn't knitting because y was undefined so I did this arbitrarily
```

```r
calculate_credible_interval_length <- function(kappa_0, n) {
  posterior_mode <- sum(y) / (n + kappa_0)
  posterior_sd <- sqrt(1 / (n + kappa_0))
  lower_bound <- posterior_mode - 1.96 * posterior_sd
  upper_bound <- posterior_mode + 1.96 * posterior_sd
  length_interval <- 2 * 1.96 * posterior_sd  # Length of the credible interval
  return(length_interval)
}

credible_interval_lengths <- sapply(kappa_0_values, calculate_credible_interval_length, n = 10)

plot(kappa_0_values, credible_interval_lengths, type = "b", pch = 19,
     xlab = expression(paste(kappa[0])), ylab = "Credible Interval Length",
     main = "Credible Interval Length vs. Prior Strength")

abline(h = 1.96 * 2, col = "red", lty = 2,
       main = expression(paste("Credible Interval Length vs. ", kappa[0])))
text(15, 4, "2 * 1.96 * Posterior SD", col = "red", pos = 4)
axis(side = 1, at = kappa_0_values, labels = kappa_0_values)
legend("topright", legend = "Reference", col = "red", lty = 2, pch = NA)
```

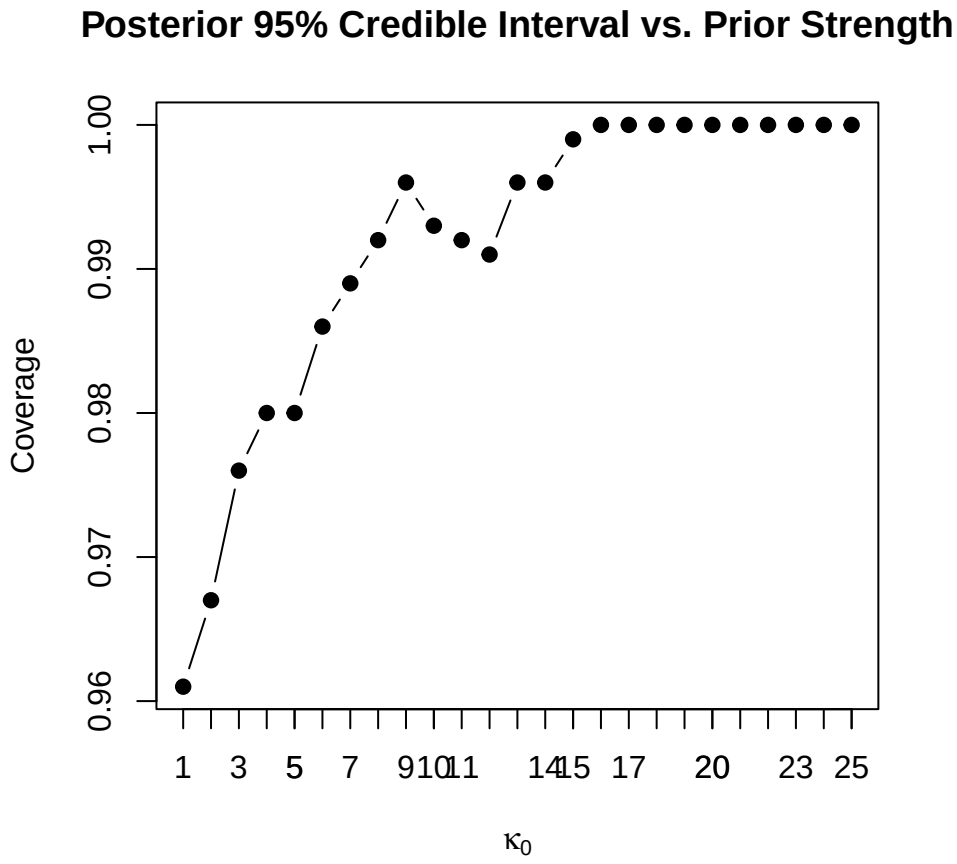


The credible interval tends to be wider when κ_0 is small (weaker prior) and narrower as κ_0 increases (stronger

prior). This makes sense because a stronger prior reduces the influence of the data, leading to a narrower interval around the prior mean.

1c. Now we will evaluate the *frequentist coverage* of the posterior credible interval on simulated data. Generate 1000 data sets where the true value of $\mu = 0$ and $n = 10$. For each dataset, compute the posterior 95% interval endpoints (from the previous part) and see if it the interval covers the true value of $\mu = 0$. Compute the frequentist coverage as the fraction of these 1000 posterior 95% credible intervals that contain $\mu = 0$. Do this for each value of $\kappa_0 = 1, 2, ..., 25$. Plot the coverage as a function of κ_0.

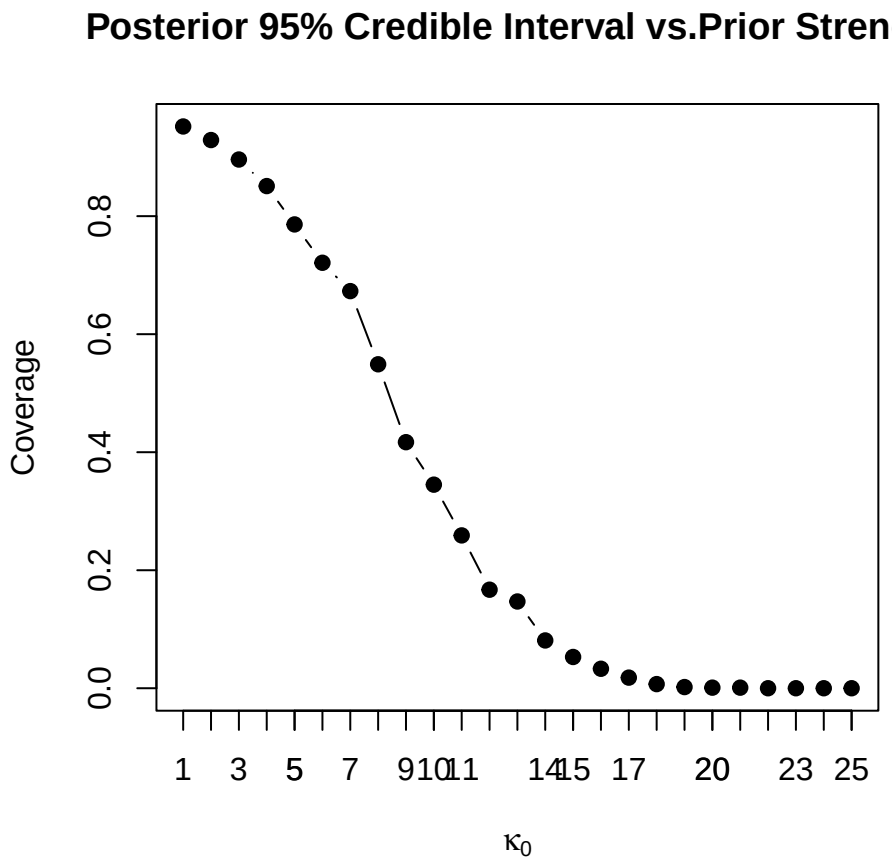
```
compute_coverage <- function(kappa_0, n, num_simulations) {
  coverage_count <- 0

  for (i in 1:num_simulations) {
    true_mu <- 0
    data <- rnorm(n, mean = true_mu, sd = 1)

    # posterior credible interval
    posterior_mode <- sum(data) / (n + kappa_0)
    posterior_sd <- sqrt(1 / (n + kappa_0))
    lower_bound <- posterior_mode - 1.96 * posterior_sd
    upper_bound <- posterior_mode + 1.96 * posterior_sd

    # does interval cover the true value of mu?
    if (lower_bound <= true_mu && upper_bound >= true_mu) {
      coverage_count <- coverage_count + 1
    }
  }
  coverage_fraction <- coverage_count / num_simulations
  return(coverage_fraction)
}

num_simulations <- 1000
kappa_0_values <- 1:25
coverage_values <- sapply(kappa_0_values, compute_coverage, n = 10,
                          num_simulations = num_simulations)

plot(kappa_0_values, coverage_values, type = "b", pch = 19,
     xlab = expression(paste(kappa[0])), ylab = "Coverage",
     main = "Posterior 95% Credible Interval vs. Prior Strength")

axis(side = 1, at = kappa_0_values, labels = kappa_0_values)
```



1d. Repeat the 1c but now generate data assuming the true $\mu = 1$.

```r
compute_coverage_mu_1 <- function(kappa_0, n, num_simulations) {
  coverage_count <- 0

  for (i in 1:num_simulations) {
    true_mu <- 1
    data <- rnorm(n, mean = true_mu, sd = 1)

    # posterior credible interval
    posterior_mode <- sum(data) / (n + kappa_0)
    posterior_sd <- sqrt(1 / (n + kappa_0))
    lower_bound <- posterior_mode - 1.96 * posterior_sd
    upper_bound <- posterior_mode + 1.96 * posterior_sd

    # does interval cover the true value of mu?
    if (lower_bound <= true_mu && upper_bound >= true_mu) {
      coverage_count <- coverage_count + 1
    }
  }
  coverage_fraction <- coverage_count / num_simulations
  return(coverage_fraction)
}
```

```r
kappa_0_values <- 1:25
coverage_values_mu_1 <- sapply(kappa_0_values, compute_coverage_mu_1, n = 10,
                               num_simulations = num_simulations)

plot(kappa_0_values, coverage_values_mu_1, type = "b", pch = 19,
     xlab = expression(paste(kappa[0])), ylab = "Coverage",
     main = "Posterior 95% Credible Interval vs.Prior Strength")

axis(side = 1, at = kappa_0_values, labels = kappa_0_values)
```



1e. Explain the differences between the coverage plots when the true $\mu = 0$ and the true $\mu = 1$. For what values of κ_0 do you see closer to nominal coverage (i.e. 95%)? For what values does your posterior interval tend to overcover (the interval covers the true value more than 95% of the time)? Undercover (the interval covers the true value less than 95% of the time)? Why does this make sense?

When the true $\mu = 0$:

For values of κ_0 that are relatively small, the coverage tends to be closer to the nominal 95%, meaning when the prior is weaker, the Bayesian posterior credible intervals tend to have better coverage properties. As κ_0 increases and the prior becomes stronger, the Bayesian posterior intervals start to overcover, meaning that the intervals cover the true value of $\mu = 0$ more than 95% of the time. Also for extremely large values of κ_0, the Bayesian posterior intervals may start to undercover leading to intervals that are too narrow. This makes

sense because a strong prior effectively "regularizes" the estimation process, pulling the posterior towards the prior mean.

When the true $\mu = 1$:

Similar to the case above, for values of κ_0 that are relatively small, the coverage tends to be closer to the nominal 95%, since weak priors result in better coverage properties. As κ_0 increases and the prior becomes stronger, the Bayesian posterior intervals still tend to overcover. However, the amount may fluctuate since the strength of the prior influences the location of the posterior mode. Similarly, for extremely large values of κ_0, the Bayesian posterior intervals may start to undercover because an extremely strong prior dominates the likelihood. The behavior is similar to the case of true $\mu = 0$, but the location of the true parameter value ($\mu = 1$) affects the centering of the intervals.

Problem 2. Rstan Warm up for Women's World Cup

Chinese Women soccer team has won AFC Woman's Asian Cup recently. Suppose you are interested in the following World Cup performance of Chinese women soccer team. Let λ be the be the average number of goals scored of Chinese Women team. We will analyze λ by Gamma-Poisson model where data Y_i is the observed number of goals scored in World Cup games. ie. we have $Y_i|\lambda \sim Pois(\lambda)$ and $\lambda \sim Gamma(a, b)$. According to a sport analyst, they believes that λ follows a Gamma distribution with $a = 1$ and $b = 0.25$.

2a. Compute the theoretical posterior parameters a, b, and also posterior mean mu.

Prior Parameters:

Prior Gamma Distribution: $\lambda \sim$ Gamma(a, b) Prior Shape Parameter: $a = 1$ Prior Rate Parameter: $b = 0.25$ Likelihood:

Likelihood for Poisson data: $Y_i|\lambda \sim$ Poisson(λ) Posterior Parameters:

Posterior Gamma Distribution: $\lambda|Y \sim$ Gamma(a', b') The posterior parameters can be calculated as follows:

Posterior Shape Parameter (a'):

$a' = a + \sum_{i=1}^{n} Y_i$ Here, $a = 1$ (prior shape parameter), and we sum over all the observed goals (Y_i) in the World Cup games. Posterior Rate Parameter (b'):

$b' = b + n$ Here, $b = 0.25$ (prior rate parameter), and n is the total number of World Cup games observed. Posterior Mean (μ'):

$\mu' = \frac{a'}{b'}$ The posterior mean is the expected value of λ given the data. Now, let's calculate these values:

Posterior Shape Parameter: $a' = 1 + \sum_{i=1}^{n} Y_i$

Posterior Rate Parameter: $b' = 0.25 + n$

Posterior Mean: $\mu' = \frac{a'}{b'}$

2b. Create a stan file named `women_cup.stan`, use Rstan to Report and estimate of the posterior mean of the scoring rate by computing the sample average of all Monte Carlo samples of λ.

```
data <- list(
  n = 5,  # number of games
  Y = c(4, 7, 3, 2, 3),  # Number of goals in each game
  a_prior = 1,  # Prior shape parameter
  b_prior = 0.25  # Prior rate parameter
)

stan_model <- stan_model("~/women_cup.stan")
fit <- sampling(stan_model, data = data, chains = 4, iter = 1000, warmup = 500)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 1:                0.001 seconds (Sampling)
## Chain 1:                0.002 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 2:                0.001 seconds (Sampling)
## Chain 2:                0.002 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
```

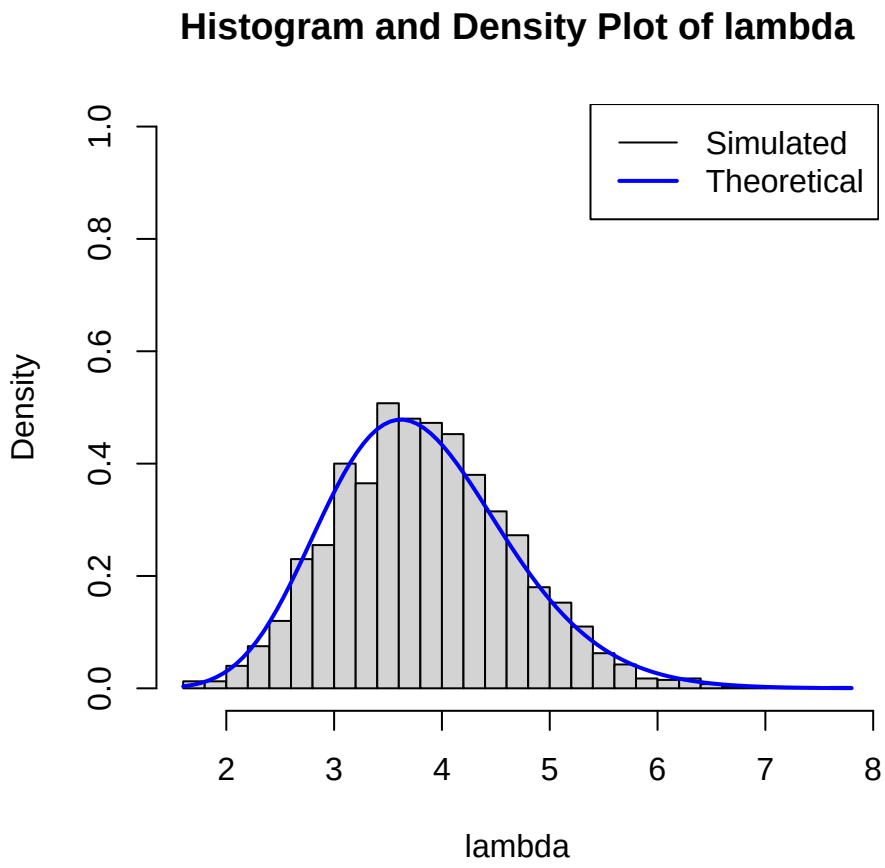
```
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 3:                0.001 seconds (Sampling)
## Chain 3:                0.002 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 4:                0.001 seconds (Sampling)
## Chain 4:                0.002 seconds (Total)
## Chain 4:
```

```
posterior_mean <- summary(fit)$summary["lambda", "mean"]
cat("Posterior Mean of Scoring Rate (lambda):", posterior_mean, "\n")
```

```
## Posterior Mean of Scoring Rate (lambda): 3.843047
```

2c. Produce histogram for simulated lambda and density plot for theoretical posterior distribution of lambda. Does the simulated results coincide with the theoretical ones? Briefly explain your answer.

```r
simulated_lambda <- as.numeric(extract(fit, "lambda")$lambda)
a_prior = 1  # Prior shape parameter
b_prior = 0.25  # Prior rate parameter

# theoretical posterior distribution using Density function
theoretical_posterior <- function(lambda) {
  dgamma(lambda, shape = a_prior + sum(data$Y), rate = b_prior + data$n)
}

hist(simulated_lambda, breaks = 30, freq = FALSE, main = "Histogram and Density Plot of lambda",
     xlab = "lambda", ylim = c(0, 1))
curve(theoretical_posterior(x), col = "blue", lwd = 2, add = TRUE, yaxt = "n")
legend("topright", legend = c("Simulated", "Theoretical"), col = c("black", "blue"), lwd = c(1, 2))
```



The overlap indicates that the simulated results align really close with the theoretical posterior distribution, which would be a good sign for the validity of our model.

2d. Right now, we have lambda samples generated by Rstan. Use them as samples from posterior distribution to compute the mean of predicative posterior distribution to estimate the possible goal scored for next game for Chinese women soccer team.

```r
# posterior samples of lambda as a numeric vector
simulated_lambda <- as.numeric(extract(fit, "lambda")$lambda)

simulated_goals <- rpois(length(simulated_lambda), lambda = mean(simulated_lambda))
mean_simulated_goals <- mean(simulated_goals)
cat("Estimated Mean Goals for Next Game:", mean_simulated_goals, "\n")
```

```
## Estimated Mean Goals for Next Game: 3.7805
```

Problem 3. Bayesian inference for the normal distribution in Stan.

Create a new Stan file by selecting "Stan file" in the Rstudio menu. Save it as `IQ_model.stan`. We will make some basic modifications to the template example in the default Stan file for this problem. Consider the IQ example used from class. Scoring on IQ tests is designed to yield a N(100, 15) distribution for the general population. We observe IQ scores for a sample of n individuals from a particular town, $y_1, \ldots y_n \sim N(\mu, \sigma^2)$. Our goal is to estimate the population mean in the town. Assume the $p(\mu, \sigma) = p(\mu \mid \sigma)p(\sigma)$, where $p(\mu \mid \sigma)$ is $N(\mu_0, \sigma/\sqrt{\kappa_0})$ and $p(\sigma)$ is Gamma(a, b). Before you administer the IQ test you believe the town is no different than the rest of the population, so you assume a prior mean for μ of $\mu_0 = 100$, but you aren't to sure about this a priori and so you set $\kappa_0 = 1$ (the effective number of pseudo-observations). Similarly, a priori you assume σ has a mean of 15 (to match the intended standard deviation of the IQ test) and so you decide on setting $a = 15$ and $b = 1$ (remember, the mean of a Gamma is a/b). Assume the following IQ scores are observed:

3a. Make a scatter plot of the posterior distribution of the median, μ, and the precision, $1/\sigma^2$. Put μ on the x-axis and $1/\sigma^2$ on the y-axis. What is the posterior relationship between μ and $1/\sigma^2$? Why does this make sense? *Hint:* review the lecture notes.

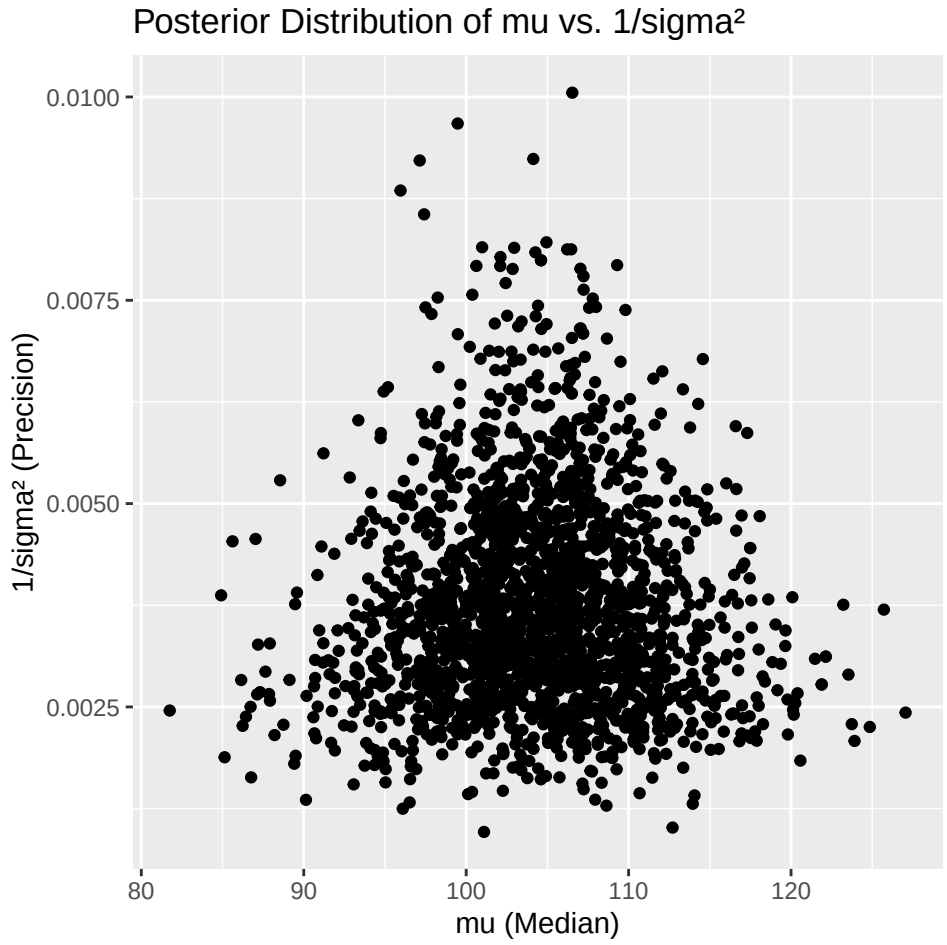
```r
IQ_scores <- c(70, 85, 111, 111, 115, 120, 123) # Observed IQ scores

data <- list(
  n = length(IQ_scores),
  y = IQ_scores,
  mu_0 = 100,
  kappa_0 = 1,
  a = 15,
  b = 1
)

stan_model <- stan_model("~/IQ_model.stan")
fit <- sampling(stan_model, data = data, chains = 4, iter = 1000, warmup = 500)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
```

```
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 1:                0.001 seconds (Sampling)
## Chain 1:                0.003 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 2:                0.002 seconds (Sampling)
## Chain 2:                0.004 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
```

```
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 3:                0.001 seconds (Sampling)
## Chain 3:                0.003 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.002 seconds (Warm-up)
## Chain 4:                0.002 seconds (Sampling)
## Chain 4:                0.004 seconds (Total)
## Chain 4:
```

```r
posterior_samples <- extract(fit)
plot_data <- data.frame(mu = posterior_samples$mu, precision = 1 / (posterior_samples$sigma^2))

# scatter plot
ggplot(plot_data, aes(x = mu, y = precision)) +
  geom_point() +
  labs(x = "mu (Median)", y = "1/sigma² (Precision)") +
  ggtitle("Posterior Distribution of mu vs. 1/sigma²")
```



The posterior relationship between μ and $1/\sigma^2$ is a relatively inverse relationship. This makes sense because when μ is estimated with more certainty, the variance (σ^2) tends to be smaller, resulting in higher precision.

3b. You are interested in whether the mean IQ in the town is greater than the mean IQ in the overall population. Use Stan to find the posterior probability that μ is greater than 100.

```
posterior_samples <- extract(fit)

posterior_prob_greater_than_100 <- mean(posterior_samples$mu > 100)
cat("Posterior Probability that mu > 100:", posterior_prob_greater_than_100, "\n")
```

```
## Posterior Probability that mu > 100: 0.7775
```

3c. You notice that two of the seven scores are significantly lower than the other five. You think that the normal distribution may not be the most appropriate model, in particular because you believe some people in this town are likely have extreme low and extreme high scores. One solution to this is to use a model that is more robust to these kinds of outliers. The Student's t distribution and the Laplace distribution are two so called "heavy-tailed distribution" which have higher probabilities of outliers (i.e. observations further from the mean). Heavy-tailed distributions are useful in modeling because they are more robust to outliers. Fit the model assuming now that the IQ scores in the town have a Laplace distribution, that is $y_1, \ldots, y_n \sim Laplace(\mu, \sigma)$. Create a copy of the previous stan file, and name it "IQ_laplace_model.stan". *Hint:* In the Stan file you can replace `normal` with `double_exponential` in the model section, another name

for the Laplce distribution. Like the normal distribution it has two arguments, μ and σ. Keep the same prior distribution, $p(\mu, \sigma)$ as used in the normal model. Under the Laplace model, what is the posterior probability that the median IQ in the town is greater than 100? How does this compare to the probability under the normal model? Why does this make sense?

```
data <- list(
  n = length(IQ_scores),
  y = IQ_scores,
  mu_0 = 100,
  kappa_0 = 1,
  a = 15,
  b = 1
)

stan_model_laplace <- stan_model("~/IQ_laplace_model.stan")
fit_laplace <- sampling(stan_model_laplace, data = data, chains = 4, iter = 1000, warmup = 500)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.004 seconds (Warm-up)
## Chain 1:                0.001 seconds (Sampling)
## Chain 1:                0.005 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
```

```
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.003 seconds (Warm-up)
## Chain 2:                0.001 seconds (Sampling)
## Chain 2:                0.004 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.004 seconds (Warm-up)
## Chain 3:                0.002 seconds (Sampling)
## Chain 3:                0.006 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
```

```
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.003 seconds (Warm-up)
## Chain 4:                0.001 seconds (Sampling)
## Chain 4:                0.004 seconds (Total)
## Chain 4:
```

```
posterior_samples_laplace <- extract(fit_laplace)
posterior_prob_greater_than_100_laplace <- mean(posterior_samples_laplace$mu > 100)
cat("Posterior Probability that mu > 100 (Laplace Model):",
    posterior_prob_greater_than_100_laplace, "\n")
```

```
## Posterior Probability that mu > 100 (Laplace Model): 0.935
```

The results show that the Laplace model assigns a higher posterior probability to mu being greater than 100 compared to the normal model, which aloows us to infer that the Laplace distribution is more adept at accounting for extreme values in the data. For the simulation we ran this makes sense because the Laplace model was more confident. This difference in probabilities highlighted the impact of model choice, especially when dealing with data that may contain outliers or heavy-tailed distributions.