Homework 5 - Due Friday, December 11 at 11:59pm

Yanjie Qi, Jianing (Julia) Chen

```
options(tinytex.verbose = TRUE)
options(buildtools.check = function(action) TRUE )
knitr::opts_chunk$set(echo = TRUE, eval=TRUE)
suppressPackageStartupMessages(library(tidyverse))
```

```
## Warning: package 'ggplot2' was built under R version 4.0.1
```

```
## Warning: package 'tibble' was built under R version 4.0.2
```

```
## Warning: package 'tidyr' was built under R version 4.0.2
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
suppressPackageStartupMessages(library(rstan))
suppressPackageStartupMessages(library(coda))
suppressPackageStartupMessages(library(testthat))
```

NOTE THAT THIS ASSIGNMENT IS DUE ON FRIDAY

Problem 1. Logistic regression for toxicity data

Logistic regression for pesticide toxicity data (part 2).

As a reminder from homework 5, an environmental agency is testing the effects of a pesticide that can cause acute poisoning in bees. In the last homework assignment, we inferred the effects of the pesticide by fitting a model in Stan. In order to develope a deeper understanding of MCMC, in this problem we will implement our own Metropolis-Hastings algoritihm. To do so, we need to first write a function to compute the *log* posterior density. Why the log posterior? In practice, the posterior density may have *extremely* small values, especially when we initialize the sampler and may be far from the high posterior mode areas. As such, computing the

For example, computing the ratio of a normal density 1000 standard deviations from the mean to a normal density 1001 standard deviations from the mean fails because in both cases `dnorm` evalutes to 0 due to numerical underflow and 0/0 returns NaN. However, we can compute the log ratio of densities:

```
dnorm(1000) / dnorm(1001)
```

```
## [1] NaN
```

```
dnorm(1000, log=TRUE) - dnorm(1001, log=TRUE)
```

```
## [1] 1000.5
```

Let $r = \min(1, \frac{p(\theta^*|y)}{p(\theta_t|y)})$. In the accept/reject step of the your implementation of the MH algorithm, rather than checking whether $u < r$, it is equivalent to check whether $log(u) < log(r)$. Doing the accept/reject on the log scale will avoid any underflow issues and prevent our code from crashing.

1a. Complete the specification for the log posterior for the data `x` and `y` by filling in the missing pieces of the function below.

```r
## Pesticide toxicity data
x <- c(1.06, 1.41, 1.85, 1.5, 0.46, 1.21, 1.25, 1.09,
       1.76, 1.75, 1.47, 1.03, 1.1, 1.41, 1.83, 1.17,
       1.5, 1.64, 1.34, 1.31)

y <- c(0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 0)

#Log posterior function.  Must incorporate x and y data above.
log_posterior <- function(theta) {

  alpha <- theta[1]
  beta <- theta[2]

  ## Compute the probabilities as a function of alpha and beta
  ## for the observed x, y data
  prob <- exp(alpha+beta*x)/(1+exp(alpha+beta*x))

  if(any(prob == 0) | any(prob == 1))
    -Inf ## log likelihood is -Inf is prob=0 or 1
  else
    return(sum(log(prob^y*(1-prob)^(1-y))))
}
```

```r
. = ottr::check("tests/q1a.R")
```

```
## All tests passed!
```

1b. You will now complete the Metropolis-Hastings sampler by filling in the missing pieces of the algorithm below. `theta_0` is a vector of length 2, with the first argument as the initial alpha value and the second argument as the initial beta value. As your proposal, use $J(\theta * |\theta_t) \sim Normal(\theta_t, \Sigma)$. You can sample from the multivariate normal using `mvtnorm::rmvnorm`. The effectiveness of your sampler will be determined by the tuning parameter, Σ, the covariance of the bivariate normal distribution. This determines the size / shape of the proposal. Σ is determined by the `cov` argument in your sampler. Run the sampler with `cov = diag(2)`, the default. In homework 5 you showed that the dose at which there is a 50% chance of hive collapse, the LD50, can be expressed as $-\alpha/\beta$. Run your sampler for 10000 iterations with a burnin of 1000 iterations. Verify that the posterior mean LD50 based on your sampler is close to 1.2, as it was with stan.

```r
################################################
## Metropolis-Hastings for the Logistic Model
################################################

## Function to generate samples using the Metropolis-Hasting Sampler

## theta_0: initialization of the form c(alpha_init, beta_init) for some values alpha_init, beta_init
## burnin: amount of iterations to discard to reduce dependence on starting point
## iters: total number of iterations to run the algorithm (must be greater than `burnin`)

mh_logistic <- function(theta_0, burnin, iters, cov=diag(2)){

    # Initialize parameters.
    theta_t <- theta_0

    ## Create a matrix where we will store samples
    theta_out <- matrix(0, nrow=iters, ncol=2, dimnames=list(1:iters, c("alpha", "beta")))
```

```
    for(i in 1:iters){

        ## Propose new theta = (alpha, beta)
        ## The proposal will be centered the current
        ## value theta_t.  Use mvtnorm::rmvnorm

        theta_p <- mvtnorm::rmvnorm(1, mean = theta_t, sigma=cov)

        ## Accept/reject step.  Keep theta prev if reject, otherwise take theta_p
        ## Will require evaluting `log_posterior` function twice
        ## Log-rejection ratio for symmetric proposal
        logr <- min(log(1), log_posterior(theta_p)-log_posterior(theta_t))

        ## Update theta_t based on whether the proposal is accepted or not
        if (log_posterior(theta_p) > log_posterior(theta_t)){
          theta_t <- theta_p
          } else {
            random_num = runif(1)
            if (log(random_num) < logr){
              theta_t <- theta_p
              } else {
                theta_t <- theta_t
              }
            }


        ## Save the draw
        theta_out[i, ] <- theta_t
    }

    ## Chop off the first part of the chain -- this reduces dependence on the starting point.
    if(burnin == 0)
      theta_out
    else
      theta_out[-(1:burnin), ]
}

samples <- mh_logistic(c(0, 0), 1000, 10000)

ld50_posterior_mean <- mean(-samples[,1]/samples[,2])
ld50_posterior_mean
```

```
## [1] 1.198455
```

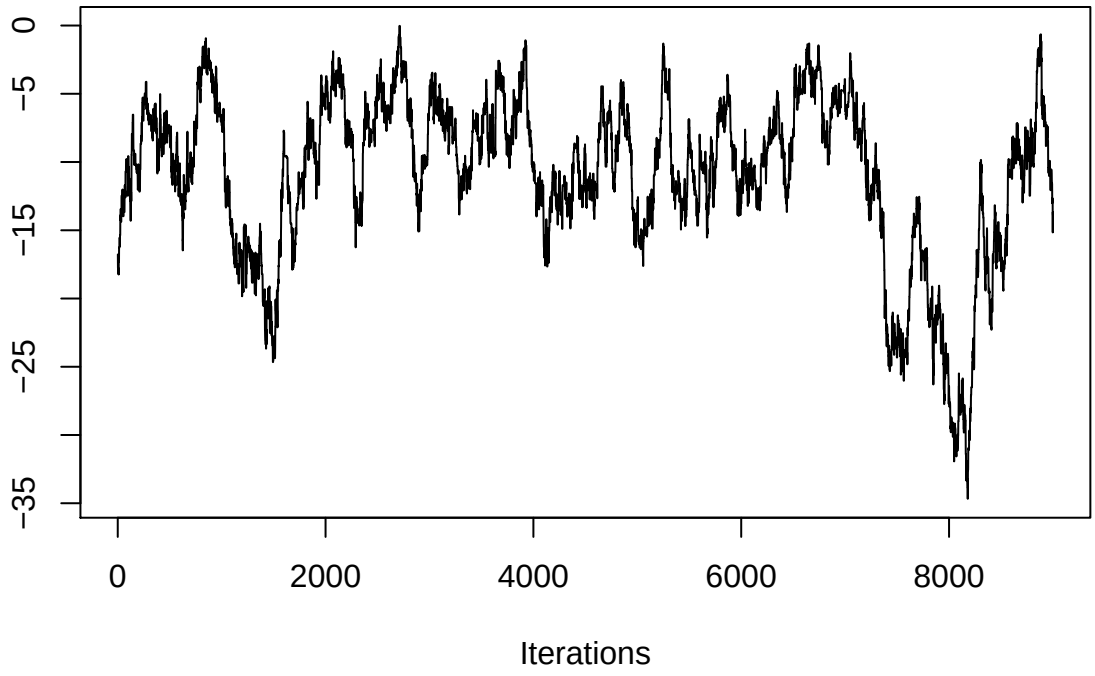
```
. = ottr::check("tests/q1b.R")
```

```
## All tests passed!
```

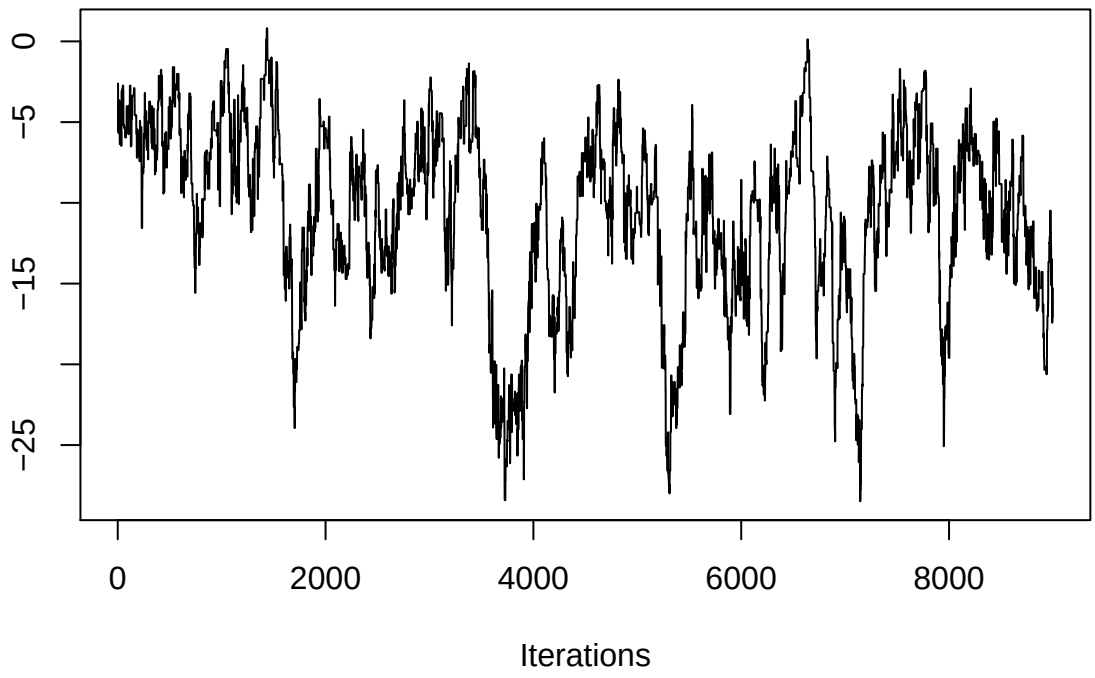
1c. Report the effective sample size for the alpha samples using the `coda::effectiveSize` function. Make a traceplot of the samples of the alpha parameter. If `alpha_samples` were the name of the samples of the alpha parameter, then you can plot the traceplot using `coda::traceplot(as.mcmc(alpha_samples))`. Improve upon this effective sample size from your first run by finding a new setting for `cov`. *Hint:* try variants of `k*diag(2)` for various values of k to increase or decrease the proposal variance. If you are ambitious, try proposing using a covariance matrix with non-zero correlation between the two parameters. What effective

sample size were you able to achieve? You should be able to at least double the effective sample size from your first run. Plot the traceplot based on the new value of `cov`.

```
library(coda)


alpha_samples <- samples[,1]
alpha_ess <- coda::effectiveSize(alpha_samples)
# TRACEPLOT HERE
traceplot(as.mcmc(alpha_samples))
```




```
k <- 3
cov <- k*diag(2)

## Re run the sampler using your new setting of cov
samples_new <- mh_logistic(c(0, 0), 1000, 10000, cov=cov)

alpha_samples_new <- samples_new[,1]
alpha_ess_new <- effectiveSize(alpha_samples_new)
# TRACEPLOT HERE
traceplot(as.mcmc(alpha_samples_new))
```




```
. = ottr::check("tests/q1c.R")
```

```
## All tests passed!
```

Problem 2. Estimating Skill In Baseball

In baseball, the batting average is defined as the fraction of base hits (successes) divided by "at bats" (attempts). We can conceptualize a player's "true" batting skill as $p_i = \lim_{n_i \to \infty} \frac{y_i}{n_i}$. In other words, if each at bat was independent (a simplifying assumption), p_i describes the total fraction of success for player i as the number of attempts gets very large. Our goal is to estimate the true skill of all player as best as possible using only a limited amount of data. As usual, for independent counts of success/fail data it is reasonable to assume that $Y_i \sim \text{Bin}(n_i, p_i)$. The file "lad.csv" includes the number of hits, `y` and the number of attempts `n` for $J = 10$ players on the Los Angeles Dodgers after the first month of the most recent baseball season. The variable `val` includes the end-of-season batting average and will be used to validate the quality of various estimates. If you are interested, at the end of the assignment we have included the code that was used to scrape the data.

```
baseball_data <- read_csv("lad.csv", col_types=cols())
baseball_data
```

```
## # A tibble: 10 x 4
##    name                y     n   val
##    <chr>           <dbl> <dbl> <dbl>
##  1 Austin Barnes      18    86 0.206
##  2 Chase Utley        22   106 0.208
##  3 Chris Taylor       52   210 0.255
##  4 Cody Bellinger     48   199 0.265
```

```
##  5 Corey Seager         27    94 0.287
##  6 Enrique Hernandez    26   122 0.257
##  7 Joc Pederson         32   129 0.249
##  8 Matt Kemp            57   163 0.292
##  9 Yasiel Puig          36   137 0.274
## 10 Yasmani Grandal      39   155 0.24
```

```
## observed hits in the first month
y <- baseball_data$y

## observed at bats in the first month
n <- baseball_data$n

## observed batting average in the first month (same as MLE)
theta_mle <- y/n

## number of players
J <- nrow(baseball_data)

## end of the year batting average, used to evaluate estimates
val <- baseball_data$val
```

2a. Compute the standard deviation of the empirical batting average, y/n and then compute the sd of the "true skill", (the `val` variable representing the end of season batting average). Which is smaller? Why does this make sense? *Hint:* What sources of variation are present in the empirical batting average?

```
empirical_sd <- sd(theta_mle)
true_sd <- sd(val)
print(empirical_sd)
```

```
## [1] 0.04264024
```

```
print(true_sd)
```

```
## [1] 0.02925007
```

```
. = ottr::check("tests/q2a.R")
```

```
## All tests passed!
```

The true standard deviation is smaller. It makes sense because the empiracal batting averages are for the players only after the first month of the most recent baseball season, but the true batting averages record the end-of-season batting averages, which mean there there should be more sampling variability for the empirical batting average.

2b. Consider two estimates for the true skill of player i, p_i: 1) $\hat{p}_i^{(\text{mle})} = \frac{y_i}{n_i}$ and 2) $\hat{p}_i^{(\text{comp})} = \frac{\sum_j y_j}{\sum_j n_j}$. Estimator 1) is the MLE for each player and ignores any commonalities between the observations. This is sometimes termed the "no pooling" estimator since each parameter is estimating separately without "pooling" information between them. Estimator 2) assumes all players have identical skill and is sometimes called the "complete pooling" estimator, because the data from each problem is completely "pooled" into one common set. In this problem, we'll treat the end-of-season batting average as a proxy for true skill, p_i. Compute the root mean squared error (RMSE), $\sqrt{\frac{1}{J}\sum_i (\hat{p}_i - p_i)^2}$ for the "no pooling" and "complete pooling" estimators using the variable `val` as a stand-in for the true p_i. Does "no pooling" or "complete pooling" give you a better estimate of the end-of-year batting averages in this specific case?

```
# Maximum likelihood estimate
phat_mle <- y/n
```

```
# Pooled estimate
phat_pooled <- sum(y)/sum(n) # YOUR CODE HERE

rmse_complete_pooling <- sqrt(1/J * sum((phat_pooled - val)^2))
rmse_no_pooling <- sqrt(1/J *sum((phat_mle - val)^2))

print(sprintf("MLE: %f", rmse_no_pooling))
```

```
## [1] "MLE: 0.024795"
```

```
print(sprintf("Pooled: %f", rmse_complete_pooling))
```

```
## [1] "Pooled: 0.027791"
```

```
. = ottr::check("tests/q2b.R")
```

```
## All tests passed!
```

From the RMSE of MLE and Pooled, since MLE has smaller value, "no pooling" gives us better estimate.

The no pooling and complete pooling estimators are at opposite ends of a spectrum. There is a more reasonable compromise: "partial pooling" of information between players. Although we assume the number of hits follow a binomial distribution. To complete this specification, we assume $\text{logit}(p_i) \sim N(\mu, \tau^2)$ for each player i. μ is the "global mean" (on the logit scale), $\exp(\mu)/(1+\exp(\mu)$ is the overall average batting average across all players. τ describes how much variability there is in the true skill of players. If $\tau = 0$ then all players are identical and the only difference in the observed hits is presumed to be due to chance. If τ^2 is very large then the true skill differences between players is assumed to be large and our estimates will be close to the "no pooling" estimator. How large should τ be? We don't know but we can put a prior distribution over the parameter and sample it along with the p_i's! Assume the following model:

$$
\begin{aligned}
&y_i \sim Bin(n_i, p_i) \\
&\theta_i = logit(p_i) \\
&\theta \sim N(\mu, \tau^2) \\
&p(\mu) \propto \text{const} \\
&p(\tau) \propto \text{Cauchy}(0,1)^+, \text{ (the Half-cauchy distribution, see part d.)}
\end{aligned}
$$

2c. State the correct answer in each case: as $\tau \to \infty$, the posterior mean estimate of p_i in this model will approach the (complete pooling / no pooling) estimator and as $\tau \to 0$ the posterior mean estimate of p_i will approach the (complete pooling / no pooling) estimator. Give a brief justification for your answer.

As $\tau \to \infty$, the posterior mean estimate of p_i in this model will approach the no pooling estimator and as $\tau \to 0$ the posterior mean estimate of p_i will approach the complete pooling estimator. Since as $\tau \to \infty$, there are true skill differences between players, hence we treat each player as an complete indepedent subject, which means it would approach pooling estimator as $\tau \to \infty$. Similarly, as $\tau \to 0$, the differences between players are supposed to small enough to be ignored; hence we treat players' true skills are identical, which means we would approach complete pooling.

2d. Implement the hierarchical binomial model in Stan. As a starting point for your Stan file modify the `eight_schools.stan` file we have provided and save it as `baseball.stan`. To write the hierarchical binomial model, we need the following modifications to the normal hierarchical model:

- Since we are fitting a hierarchical binomial model, not a normal distribution, we no longer need sampling variance σ_i^2. Remove this from the data block.
- The outcomes `y` are now integers. Change `y` to an array of integer types in the data block.

- We need to include the number of at bats for each player (this is part of the binomial likelihood). Add an array of integers, `n` of length J to the data block.
- Replace the sampling model for y with the binomial-logit: `binomial_logit(n, theta)`. This is equivalent to `binomial(n, inv_logit(theta))`.
- The model line for `eta` makes $\theta_i \sim N(\mu, \tau^2)$. Leave this in the model.
- Add a half-cauchy prior distribution for τ: `tau ~ cauchy(0, 1);`. The half-cauchy has been suggested as a good default prior distribution for group-level standard deviations in hierarchical models. See http://www.stat.columbia.edu/~gelman/research/published/taumain.pdf.

Find the posterior means for each of the players batting averages by looking at the samples for `inv_logit(theta_samples)`. Report the RMSE for hierchical estimaftor. How does this compare to the RMSE of the complete pooling and no pooling estimators? Which estimator had the lowest error?

```
# Run Stan and compute the posterior mean
stan_model <- rstan::stan_model(file="baseball.stan")

results <- rstan::sampling(stan_model,
                           data=list(n=n, y=y, J=J),
                           refresh = 0)
```

```
## Warning: There were 1 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Theta samples are logit scale
theta_samples <- extract(results)$theta
# Get batting averages by inverting with this function
inv_logit <- function(x) {
  exp(x) / (1+exp(x))
}
# and compute the posterior mean for each theta
pm <- vector()

for (i in 1:10){
  mean <- mean(inv_logit(theta_samples[,i]))
  pm <- c(pm, mean)
}

# RMSE From Stan posterior means
rmse_partial_pooling <- sqrt(1/J * sum((pm - val)^2)) # YOUR CODE HERE

print(c(rmse_complete_pooling, rmse_no_pooling, rmse_partial_pooling))
```

```
## [1] 0.02779054 0.02479514 0.01986701
```

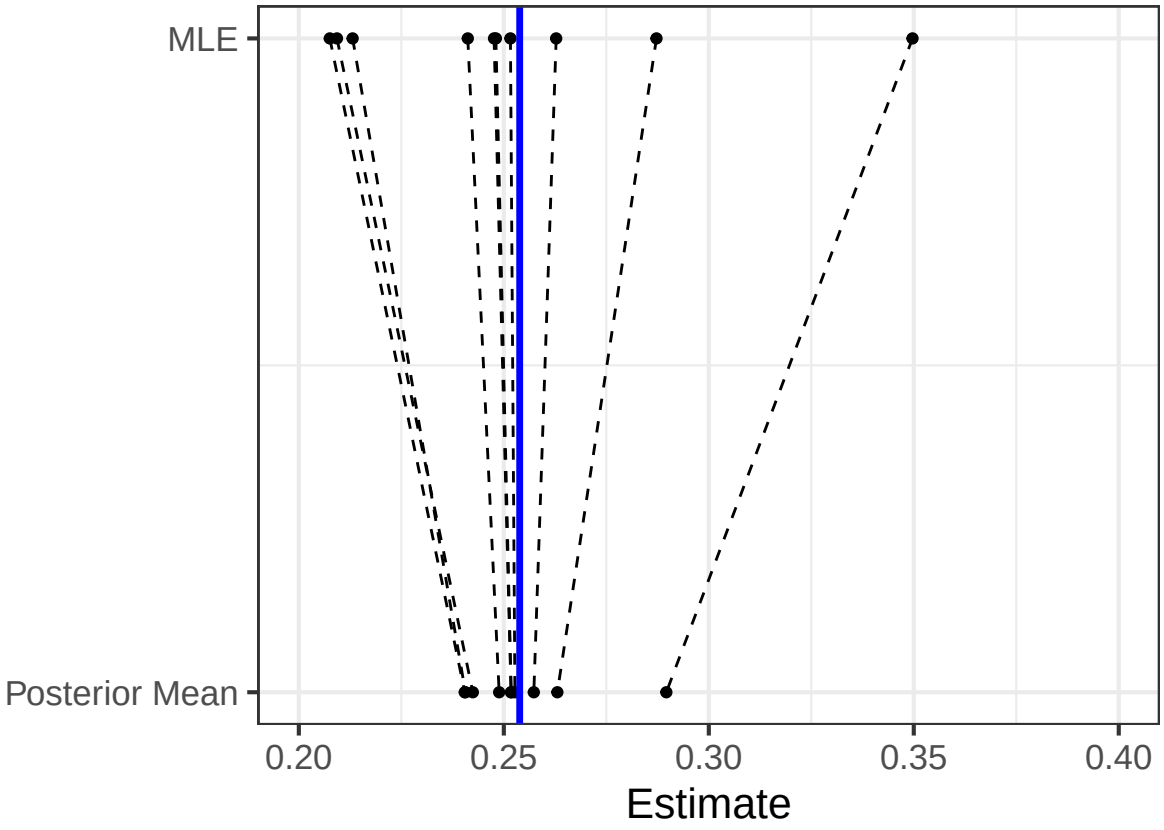
```
. = ottr::check("tests/q2d.R")
```

```
## All tests passed!
```

RMSE with partical pooling had the lowest error.

2e. Use the `shrinkage_plot` function provided below to show how the posterior means shrink the empirical batting averages. Pass in `y/n` and the posterior means of p_i as arguments.

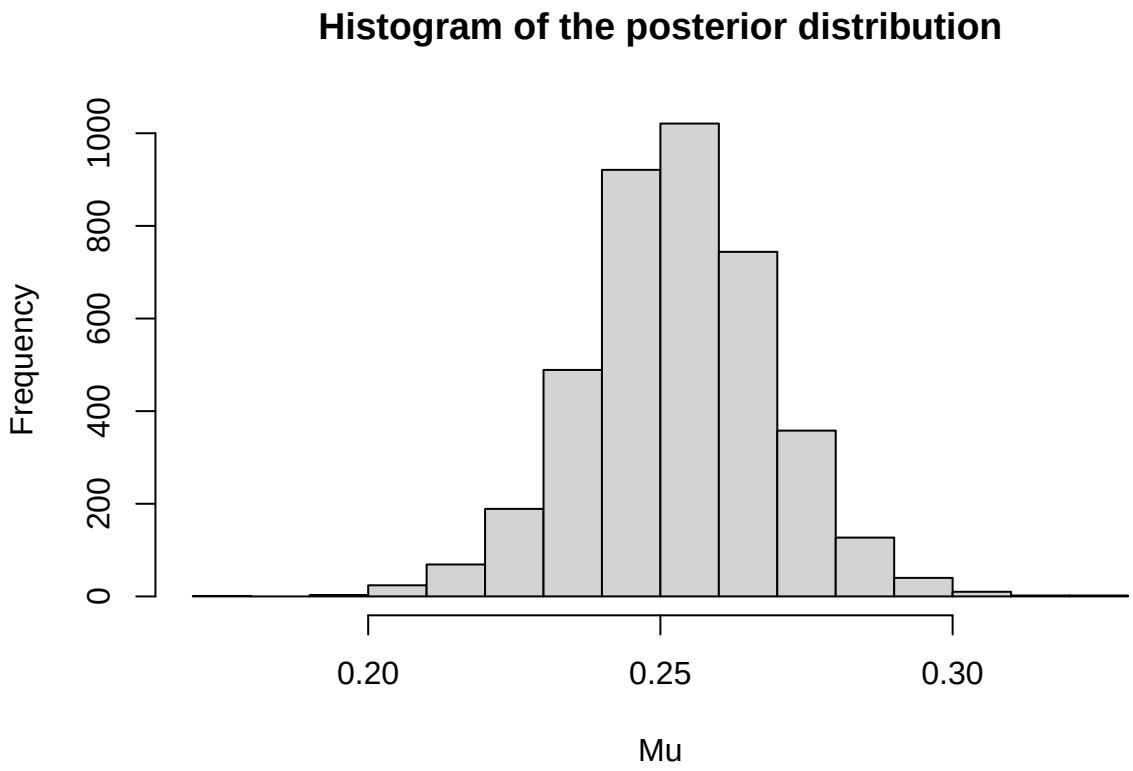
```
shrinkage_plot <- function(empirical, posterior_mean,
                           shrink_point=mean(posterior_mean)) {

  tibble(y=empirical, pm=posterior_mean) %>%
    ggplot() +
    geom_segment(aes(x=y, xend=pm, y=1, yend=0), linetype="dashed") +
    geom_point(aes(x=y, y=1)) +
    geom_point(aes(x=pm, y=0)) +
    theme_bw(base_size=16) +
    geom_vline(xintercept=shrink_point, color="blue", size=1.2) +
    ylab("") + xlab("Estimate") +
    xlim(c(0.2, 0.4)) +
    scale_y_continuous(breaks=c(0, 1),
                       labels=c("Posterior Mean", "MLE"),
                       limits=c(0,1))
}

shrinkage_plot(y/n, pm)
```



2f. Make a histogram of the posterior distribution for the global batting average, $\frac{e^{\mu}}{1+e^{\mu}}$, based on the LAD data. True or false: as the observed at bats for each of the 10 LAD batters $n_i \to \infty$, our estimate of the global batting average converges to a constant. Why or why not?

```
mu <- extract(results)$mu

hist(inv_logit(mu),
```

```
     main = "Histogram of the posterior distribution",
     xlab = "Mu")
```



According to the histogram, I think as n increases, the global batting average seems to converge to 0.25, so the statement should be true. ### Appendix: Code for scraping Dodgers baseball data

http://billpetti.github.io/baseballr/

```
## Install the baseballr package
devtools::install_github("BillPetti/baseballr")

library(baseballr)
library(tidyverse)

## Download data from the chosen year
year <- 2019

one_month <- daily_batter_bref(t1 = sprintf("%i-04-01", year), t2 = sprintf("%i-05-01", year))
one_year <- daily_batter_bref(t1 = sprintf("%i-04-01", year), t2 = sprintf("%i-10-01", year))

## filter to only include players who hat at least 10 at bats in the first month
one_month <- one_month %>% filter(AB > 10)
one_year <- one_year %>% filter(Name %in% one_month$Name)

one_month <- one_month %>% arrange(Name)
one_year <- one_year %>% arrange(Name)
```

```r
## Look at only the Dodgers
LAD <- one_year %>% filter(Team == "Los Angeles" & Level == "MLB-NL") %>% .$Name

lad_month <- one_month %>% filter(Name %in% LAD)
lad_year <- one_year %>% filter(Name %in% LAD)

write_csv(tibble(name=lad_month$Name,
                 y=lad_month$H,
                 n=lad_month$AB,
                 val=lad_year$BA),
          path="lad.csv")
```