Homework 5

PSTAT 115, Winter 2023

Due on March 19, 2022 at 11:59 pm

```r
options(tinytex.verbose = TRUE)
options(buildtools.check = function(action) TRUE )
eval = TRUE
#knitr::opts_chunk$set(eval=eval,echo=TRUE,cache=FALSE,fig.width=5,fig.height=5,fig.align='center')
suppressPackageStartupMessages(library(tidyverse))
```

```
## Warning in system("timedatectl", intern = TRUE): running command 'timedatectl'
## had status 1
```

```r
suppressPackageStartupMessages(library(cmdstanr))
suppressPackageStartupMessages(library(testthat))
library(coda)
```

Problem 1. Logistic regression for toxicity data

Logistic regression for pesticide toxicity data.

A environmental agency is testing the effects of a pesticide that can cause acute poisoning in bees, the world's most important pollinator of food crops. The environmental agency collects data on exposure to different levels of the pestidicide in parts per million (ppm). The agency also identifies collapsed beehives, which they expect could be due to acute pesticide poisoning. In the data they collect, each observation is pair (x_i, y_i), where x_i represents the dosage of the pollutant and y_i represents whether or not the hive survived. Take $y_i = 1$ means that the beehive has collapsed from poisoning and $y_i = 0$ means the beehive survived. The agency collects data at several different sites, each of which was exposed to a different dosages. The resulting data can be seen below:

```r
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)
```

Assume that beehive collapse, y_i, given pollutant exposure level x_i, is $Y_i \sim \text{Bernoulli}(\theta(x_i))$, where $\theta(x_i)$ is the probability of death given dosage x_i. We will assume that $\text{logit}(\theta_i(x_i)) = \alpha + \beta x_i$ where $\text{logit}(\theta)$ is defined as $\log(\theta/(1-\theta))$. This model is known as *logistic regression* and is one of the most common methods for modeling probabilities of binary events.

1a. Solve for $\theta_i(x_i)$ as a function of α and β by inverting the logit function. If you haven't seen logistic regression before (it is covered in more detail in PSTAT 127 and PSTAT131), it is essentially a generalization of linear regression for binary outcomes. The inverse-logit function maps the linear part, $\alpha + \beta x_i$, which can be any real-valued number into the interval [0, 1] (since we are modeling probabilities of binary outcome, we need the mean outcome to be confined to this range).

$$
\begin{aligned}
&logit(\theta_i(x_i)) = \alpha + \beta x_i \\
&\theta_i(x_i) = logit^{-1}(\alpha + \beta x_i) \\
&we\ know\ \ logit^{-1}(x) = \frac{e^x}{1+e^x} \\
&so\ \ \theta_i(x_i) = \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}
\end{aligned}
$$

1b The dose at which there is a 50% chance of beehvive collapse, $\theta(x_i) = 0.5$, is known as LD50 ("lethal dose 50%"), and is often of interest in toxicology studies. Solve for LD50 as a function of α and β.

$$
\begin{aligned}
&\theta(x_{LD50}) = 0.5 \\
&0.5 = logit^{-1}(\alpha + \beta_{x_{LD50}}) \\
&we\ know\ \ logit = ln(\frac{p}{1-p}) \\
&so\ \ ln(\frac{0.5}{1-0.5}) = \alpha + \beta x_{LD50} \\
&x_{LD50} = \frac{-\alpha}{\beta}
\end{aligned}
$$

1c Implement the logistic regression model in stan by reproducing the stan model described here: https://mc-stan.org/docs/2_18/stan-users-guide/logistic-probit-regression-section.html. In this model, we assume the improper prior $p(\alpha, \beta) \propto$ const. Run the stan model on the beehive data to get Monte Carlo samples. Compute Monte Carlo samples of the LD50 by applying the function derived in the previous part to your α and β samples. Report and estimate of the posterior mean of the LD50 by computing the sample average of all Monte Carlo samples of LD50.

```
# Running the stan model
ld_model <- cmdstan_model("LD50.stan")

logistic_fit <- ld_model$sample(data=list(y = y, N = length(x), x = x), refresh=0, show_messages = FALSI
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 finished in 0.1 seconds.
## Chain 2 finished in 0.1 seconds.
## Chain 3 finished in 0.1 seconds.
## Chain 4 finished in 0.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.1 seconds.
## Total execution time: 0.6 seconds.
```

```
logistic_samples <- logistic_fit$draws(format="df")

alpha_samples <- logistic_samples$alpha
beta_samples <- logistic_samples$beta

# Posterior mean
ld50 <- sum((-alpha_samples) / beta_samples) / length(alpha_samples)
print(ld50)
```

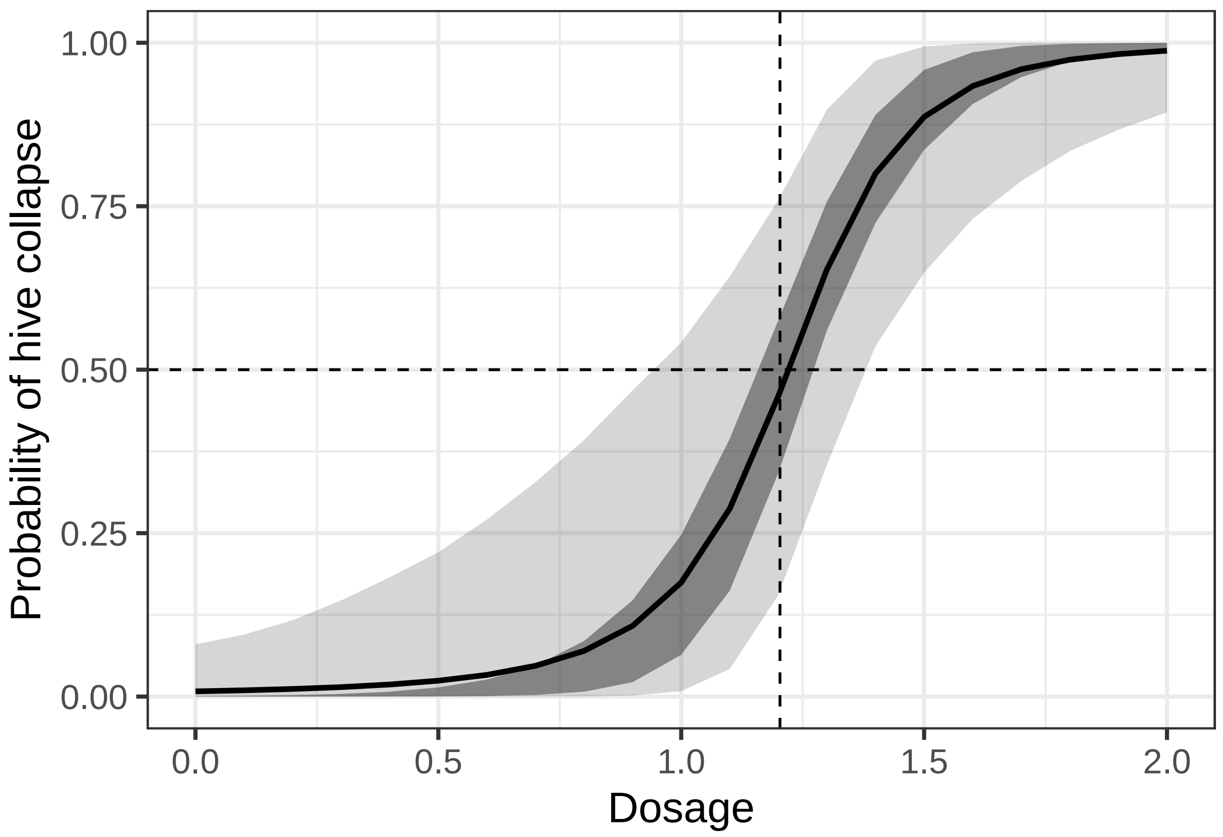
```
## [1] 1.203311
```

My estimate of the posterior mean is 1.196023.

Fill in the compute curve function, which computes the probability of hive collapse for each value of x in `xgrid`. Then run the code below to make a plot showing both 50% and 95% confidence band for the probability of a hive collapse as a function of pollutant exposure, $\Pr(y = 1 \mid \alpha, \beta, x)$. This will plot your predicted hive collapse probabilities for dosages from $x = 0$ to 2. Verify that you computed the LD50 correctly by identifying the x-value at which the posterior mean crosses 0.5.

```r
xgrid <- seq(0, 2, by=0.1)

## Evaluate probability on the xgrid for one alpha, beta sample
compute_curve <- function(sample) {
  alpha <- sample[1]
  beta <- sample[2]

  prob <- exp(alpha + beta*xgrid) / (1 + exp(alpha + beta*xgrid))

}

predictions <- apply(cbind(alpha_samples, beta_samples), 1, compute_curve)

quantiles <- apply(predictions, 1, function(x) quantile(x, c(0.025, 0.25, 0.75, 0.975)))
posterior_mean <- rowMeans(predictions)

tibble(x=xgrid,
       q025=quantiles[1, ],
       q25=quantiles[2, ],
       q75=quantiles[3,],
       q975=quantiles[4, ],
       mean=posterior_mean) %>%
  ggplot() +
  geom_ribbon(aes(x=xgrid, ymin=q025, ymax=q975), alpha=0.2) +
  geom_ribbon(aes(x=xgrid, ymin=q25, ymax=q75), alpha=0.5) +
  geom_line(aes(x=xgrid, y=posterior_mean), size=1) +
  geom_vline(xintercept = ld50, linetype="dashed") +
  geom_hline(yintercept = 0.5, linetype="dashed") +
  theme_bw(base_size=16) + ylab("Probability of hive collapse") + xlab("Dosage")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```



My computed LD50 correctly corresponds to the x-value at which the posterior mean crosses 0.5.

Problem 2. Implementing your own Metropolis-Hastings Algorithm

Stan implements a particular MCMC algorithm to get samples. In order to develop a deeper understanding of MCMC, in this problem we will implement our own Metropolis-Hastings algorithm. 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:

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

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

```r
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.

2a. 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(y*log(prob)) + sum((1 - y)*log(1 - prob)))
}
```

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

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

2b. 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.

```
################################################
## 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, theta_t, 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 <- c(log_posterior(theta_p), log_posterior(theta_t))

        ## Update theta_t based on whether the proposal is accepted or not
        u <- runif(1, 0, 1)
        r <- min(0, logr[1] - logr[2])

        if(r > log(u)){
          theta_t = theta_p
        }

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

```
## [1] 1.216609
```

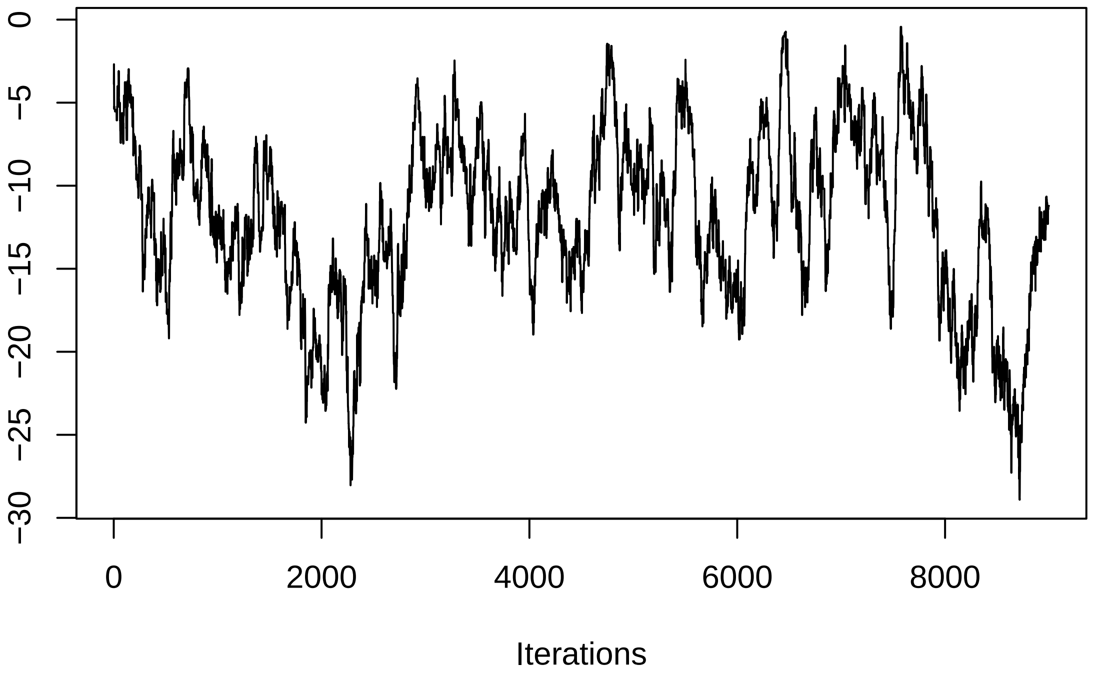
```
. = ottr::check("tests/q2b.R")
```

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

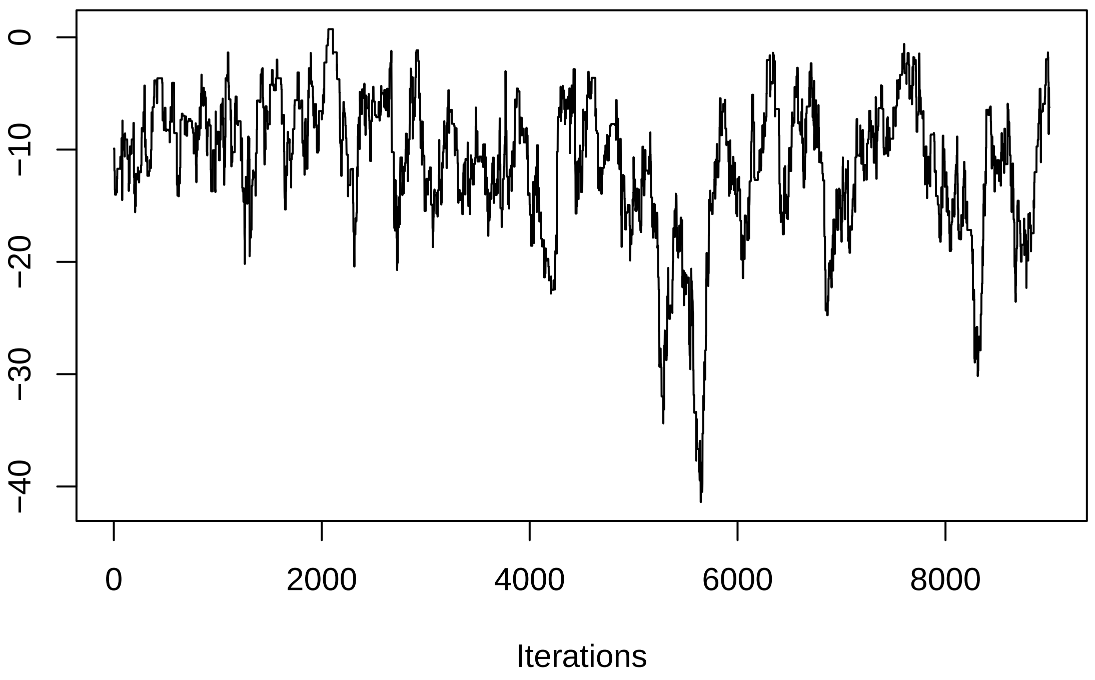
2c. Report the effective sample size for the alpha samples using the `coda::effectiveSize` function. Make a traceplot of the samples of the alpha parameter. If `alpha_samples` were the name of the samples of the alpha parameter, then you can plot the traceplot using `coda::traceplot(as.mcmc(alpha_samples))`. 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)
```

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

alpha_samples <- samples[,1]

alpha_ess <- coda::effectiveSize(alpha_samples)

# TRACEPLOT HERE
coda::traceplot(as.mcmc(alpha_samples))
```



```
## Re run the sampler using your new setting of cov
samples_new <- mh_logistic(c(0, 0), 1000, 10000, cov = matrix(c(6, .25, .25, 6), nrow = 2))
alpha_samples_new <- samples_new[,1]
alpha_ess_new <- coda::effectiveSize(alpha_samples_new)

# TRACEPLOT HERE
coda::traceplot(as.mcmc(alpha_samples_new))
```



```r
. = ottr::check("tests/q2c.R")
```

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

I tried using a cov setting where I had two variables with a variance of 6 and covariance of 0.25. With this new cov setting I was able to achieve a effective sample size of about 60. This is almost four times as much as the original sample size of about 15. Of course these are samples, so the effective sample size varies each time the algorithm is run.

Problem 3. 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 2019 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.

```r
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
```

3a. 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/q3a.R")
```

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

The true standard deviation is smaller than the empirical standard deviation. This makes sense because the empirical standard deviation is calculated based on data from only one month, while the true standard deviation is a full year. This means the empirical sd will have less observations, and in turn, higher variance. While the true standard deviation is based off of a full year with many more observations and hence a smaller standard deviation.

3b. 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) $\hat{p}_i^{(\text{comp})} = \frac{\sum_j y_j}{\sum n_j}$. 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 <- theta_mle
```

```r
# Pooled estimate
phat_pooled <- sum(y) / sum(n)

rmse_complete_pooling <- sqrt((1/10)*sum((phat_pooled - val)**2))
rmse_no_pooling <- sqrt((1/10)*sum((phat_mle - val)**2))

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

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

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

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

```r
. = ottr::check("tests/q3b.R")
```

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

In this specific case, the no pooling model gives us a better estimate of end-of-year batting averages.

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:

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

3c. 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 model. This is because when τ is large, the prior for θ_i is not very strong so most of the weight is given to the MLE.

As $\tau \to 0$ the posterior mean estimate of p_i will approach the complete pooling model. This is because when $\tau \to 0$ we assume a priori that θ_i are very close and is a stronger prior, thus giving the prior much more weight than the MLE.

3d. 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 hierarchical estimator. 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_data <- list(J=J, n=n, y=y)
baseball_model <- cmdstan_model("baseball.stan")
baseball_results <- baseball_model$sample(data=stan_data, iter_sampling=2000, refresh=0)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 finished in 0.1 seconds.
## Chain 2 finished in 0.1 seconds.
## Chain 3 finished in 0.1 seconds.
## Chain 4 finished in 0.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.1 seconds.
## Total execution time: 0.9 seconds.
```

```
baseball_draws <- baseball_results$draws(format="df")

# Get the matrix of Theta samples for all players
# Should be a matrix of size num_draws * num_players
# where num_players is 10
# Note: theats are on logit scale, need to convert back to prob
theta_samples <- baseball_draws[14:23]
```

```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
# Get batting averages by inverting with this function
inv_logit <- function(x){
  exp(x) / (1 + exp(x))
}

# Theta probabilities
probs <- inv_logit(theta_samples)

# and compute the posterior mean for each theta
pm <- colSums(probs) / 8000

# RMSE From Stan posterior means
rmse_partial_pooling <- sqrt(sum((pm - val)^2) / J)

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

```
## [1] 0.02779054 0.02479514 0.02035958
```

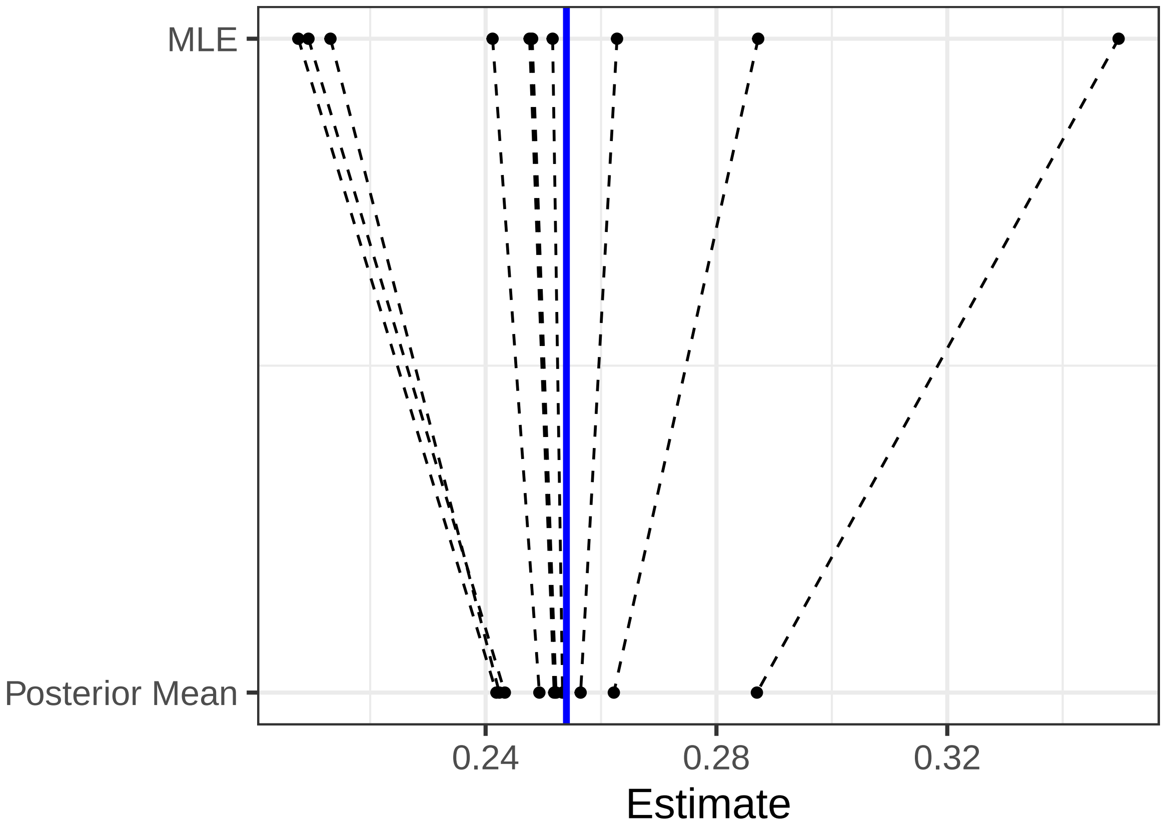
```
. = ottr::check("tests/q3d.R")
```

```
##
```

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

3e. 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") +

    scale_y_continuous(breaks=c(0, 1),
                       labels=c("Posterior Mean", "MLE"),
                       limits=c(0,1))

}

shrinkage_plot(y/n, pm)
```



Appendix: Code for scraping Dodgers baseball data

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

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

library(baseballr)
library(tidyverse)

## Download data from the chosen year
year <- 2021

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)

## 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")
```