Bayesian Survival Analysis Applied to SSBM

Final Project for Bayesian Statistics and Survival Analysis

J Dorsey

Fall 2018

Contents

Introduction

Super Smash Bros. Melee is a fighting game released in 2001 for the Nintendo GameCube. Though the game is 17 years old, it maintains a cult following and a lively competition scene. In the game, players hit their opponent to deal damage, called "percent", and send them flying. The more percent a character has, the farther they will fly when hit. When a player has enough percent and is hit with a powerful enough attack, they will fly so far that they are KO'd by entering the "blast zone" that surrounds the play area. In competitive play, games are one-on-one, and the player to first KO their opponent four times is the winner. Players usually play multiple games in a first to 3 format. This is called a set.

According to SSBMRank, the current #1 player in world is Hungrybox. The current #3 and #4 players in the world are Armada and Leffen respectively. Recently, Armada has been one of the only players to outperform Hungrybox in a head-to-head comparison. Of the last six sets Armada and Hungrybox played (as of October 2018), Armada won five. Of the last six sets Leffen and Hungrybox played (again as of October 2018), Leffen won two.

The goal of this project is to investigate if Armada KO's Hungrybox at a lower percentage than Leffen does, on average. I chose these three players because Hungrybox is #1 in the world, Armada is the only player to have a winning record against him, and Leffen is a player of comparable skill to Armada. Both Armada and Leffen play the character Fox, while Hungrybox plays the character Jigglypuff. This makes the comparisons direct and meaningful. For example, if Leffen played a different character, then the character might have different attacks that would KO at higher or lower percents. That's why it was important to pick someone who plays the same character.

Throughout the paper I have chosen to interleave code and text, like this.

```
# some code
print("This is a code chunk.")
```

```
## [1] "This is a code chunk."
```

This way the reader can see the output associated with each block of code, and the whole document can be executed to ensure that the analysis is reproducible.

Data

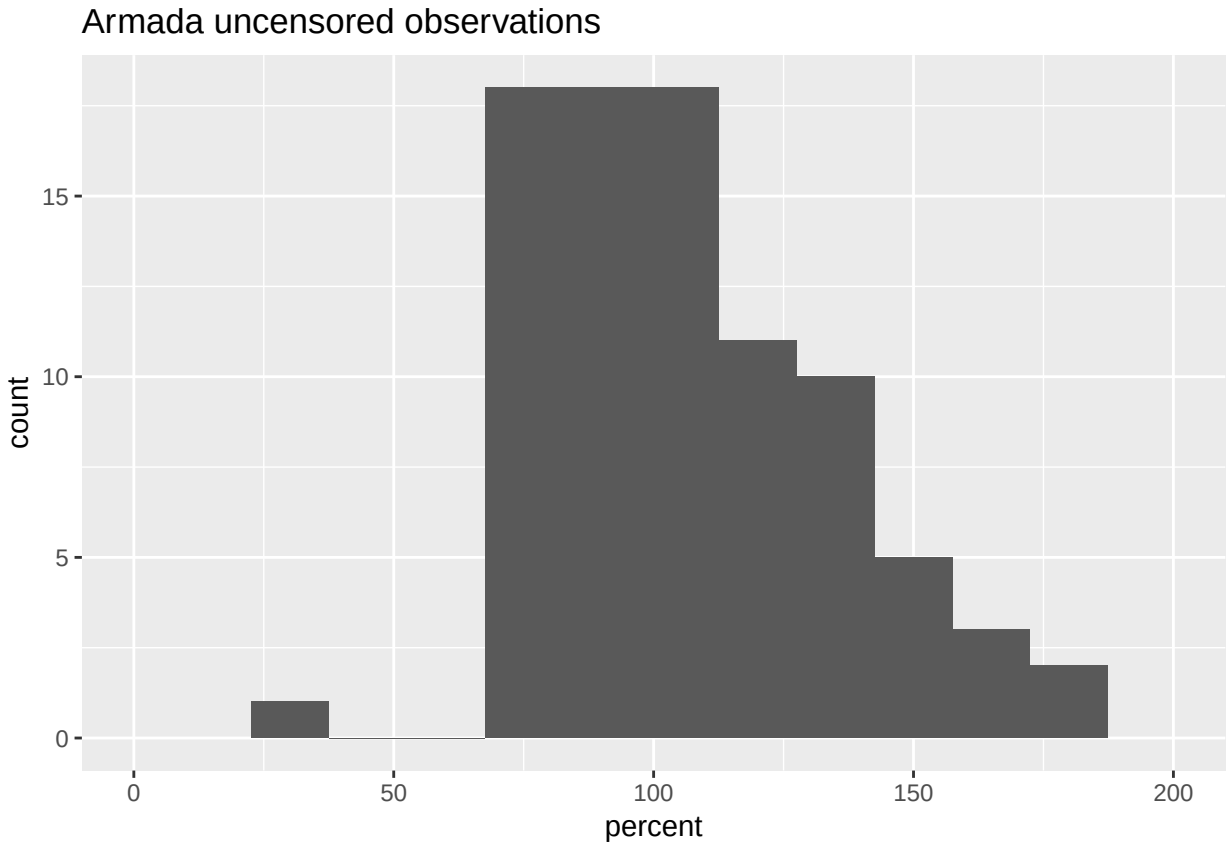
The data comes from the last six sets (as of October 2018) Armada and Leffen each played against Hungrybox. I watched each set, and for each stock I recorded the percent at which Hungrybox was KO'd. If Hungrybox won the game, then he was not KO'd on his last stock, in which case I recorded his percent and marked the observation as right censored. This resulted in 94 observations for Armada, 8 of which are censored, and 89 observations for Leffen, 15 of which are censored. The code below reads in the data and shows the first few observations.

```
# load the packages I'll use
suppressMessages(library(tidyverse))
suppressMessages(library(magrittr))
suppressMessages(library(readxl))

# set seed for reproducibility
set.seed(32135)

# read in the data
data <- read_excel("melee_project_data.xlsx")

# Take a look at it
data
```

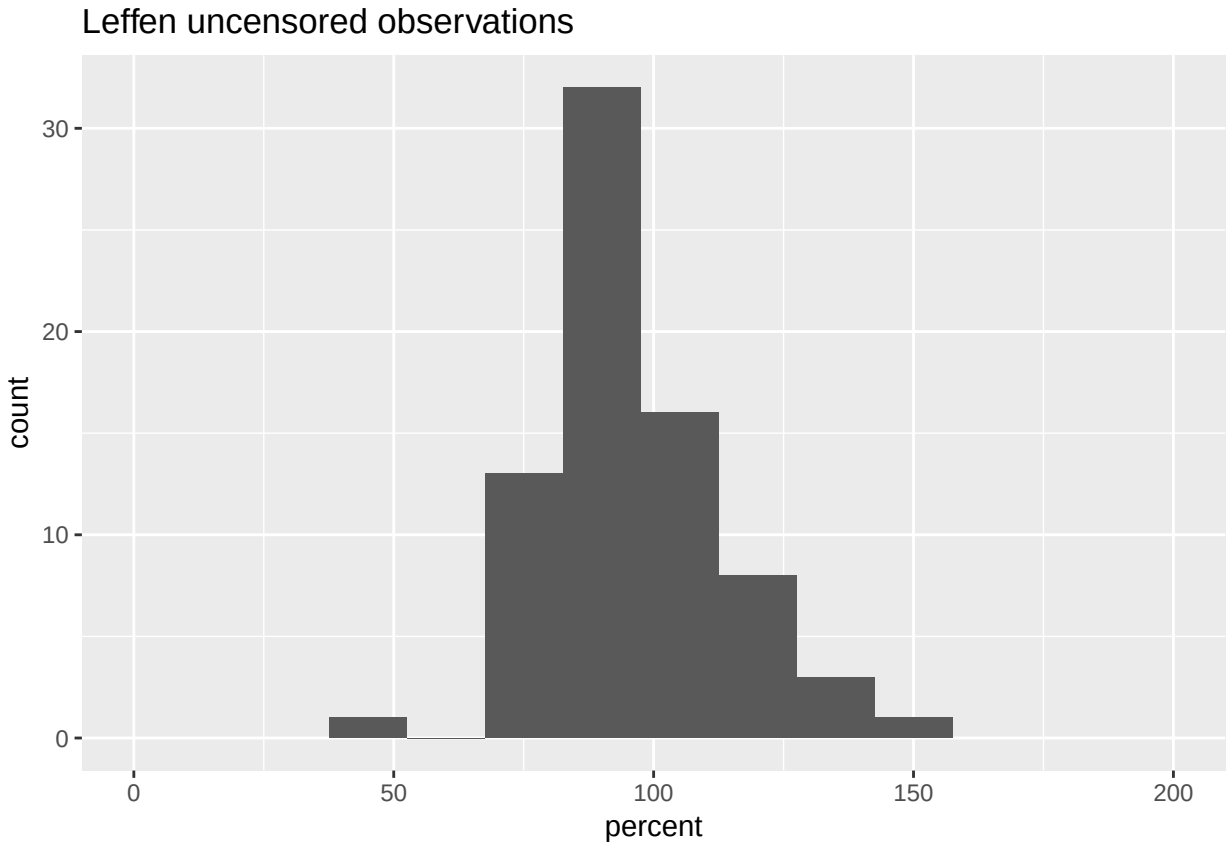
```
## # A tibble: 183 x 3
##    player percent censored
##    <chr>    <dbl> <chr>
##  1 Armada      96 no
##  2 Armada     128 no
##  3 Armada      71 no
##  4 Armada      74 no
##  5 Armada      90 no
##  6 Armada      83 no
##  7 Armada      92 no
##  8 Armada      81 no
##  9 Armada     136 no
## 10 Armada      95 no
## # ... with 173 more rows
```

Next we'll create some histograms of the uncensored observations, to get a feel for the data. Below you can see the histograms for each player and the code that creates it. Both players have 1 or 2 outlying low percent KO's. Also, for the both players, the data looks right-skew and thus non-normal. The mean KO percent for Armada is 107, and the standard deviation is 27.5. For Leffen, the mean KO percent is 95.9, and the standard deviation is 16.8.

```
# Create histogram for Armada
data %>%
    filter(player == "Armada", censored == "no") %>%
    ggplot(aes(percent)) +
    geom_histogram(binwidth = 15) +
    ggtitle("Armada uncensored observations") +
    coord_cartesian(xlim = c(0, 200))
```




```
# Create histogram for Leffen
data %>%
    filter(player == "Leffen", censored == "no") %>%
    ggplot(aes(percent)) +
    geom_histogram(binwidth = 15) +
    ggtitle("Leffen uncensored observations") +
    coord_cartesian(xlim = c(0, 200))
```



Methods

For each player we fit an identical independent model to find the mean. Because of the right-skewness of the data, I have chosen to model it with a gamma distribution. This is as opposed to something like a normal distribution because the gamma distribution allows the asymmetry that the data exhibit. I also assume that the censored observations follow the same distribution as the uncensored ones, as is common. Finally, I assume that the shape and rate parameters of the gamma distribution have unit exponential priors. Because we actually have quite of lot of data for each player, and the model we are fitting is low dimensional, the choice of prior is not so critical. The exponential distribution is chosen for its conjugacy and mathematical simplicity. The full statement of the model is given below, with the conditional independencies left implied.

$$
\begin{aligned}
x_i|\alpha,\beta &\sim \text{Gamma}(\alpha,\beta) \\
z_i|c_i,\alpha,\beta &\sim \text{TruncGamma}(c_i,\alpha,\beta) \\
\beta &\sim \text{Expo}(1) \\
\alpha &\sim \text{Expo}(1)
\end{aligned}
$$

Here, each x_i is an uncensored observation. Each z_i is a censored observation, and corresponding to each z_i is a censoring limit c_i. For example, if Hungrybox lived to 60% and won the game at this percent, then associated c_i is 60, and the associated z_i is the unknown percent that he would have been KO'd at, had the game continued. All we know about z_i is that it is greater than 60, and follows the same distribution as the x's. This is why the z_i get the truncated gamma distribution. The truncated gamma is just the distribution of a gamma random variable conditional on it being larger than some threshold.

Once we fit the model for each player, we get a joint posterior distribution over (α, β) for each player. Then, using the fact that if $X \sim \text{Gamma}(\alpha, \beta)$, $EX = \alpha/\beta$, we can transform the posterior into a posterior over the mean for each player. Then we can subtract those posteriors from each other to get a posterior over the difference in means. Thus the final object of our analysis is the posterior distribution over the difference in mean KO percent between Armada and Leffen. With this posterior, we can estimate the difference in average KO percentage with a point estimate (using the mean or median of the distribution) and with a credible interval (using quantiles).

I fit the model with a Gibbs sampler. The whole point of the Gibbs sampler that you can ultimately sample from $P(\boldsymbol{\theta}|D)$, the distribution of the parameters conditional on the data, while only directly sampling from $P(\theta_j|\boldsymbol{\theta}_{-j}, D)$, the distribution of each parameter given the data and all the other parameters. Moreover, compared to something like the Metropolis-Hastings algorithm, the Gibbs sampler involves no tuning parameters for acceptance rate. Every sample is accepted.

Furthermore, even when you cannot directly sample from $P(\theta_j|\boldsymbol{\theta}_{-j}, D)$, you can construct a Metropolis-Hastings sampler for this univariate distribution. You only need to perform one step of Metropolis-Hastings each time you're supposed to sample from this conditional distribution. Doing so luckily leaves the stationary distribution the same. In this case, the point of Gibbs sampling is still the fact that you can decompose the problem of sampling from a high dimensional posterior into the problem of sampling from many low dimensional distributions. Tuning an MH algorithm in the univariate case is fairly simple. Tuning a high-dimensional MH algorithm quickly becomes a headache.

The Gibbs sampler also lends itself naturally to missing or censored data. In each pass of the Gibbs sampler we sample the z_i from the truncated gamma distribution. Then to sample the α and β random variables, it is as if we have no censoring anymore, because we know the exact values of all the x_i and z_i. To sample the β values, we use the fact that the gamma distribution (of which the exponential is a special case) is a conjugate prior for the rate parameter of a gamma distribution, when α is known.

$$\beta|\alpha, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c} \sim \text{Gamma}(1 + n\alpha, 1 + \Sigma\boldsymbol{x} + \Sigma\boldsymbol{z})$$

To sample, α we use a single step of the Metropolis algorithm with the a standard normal proposal distribution. My experiments showed that the standard deviation of 1 gave an acceptance rate between .3 and .5, which I deemed acceptable. Finally, sampling from the truncated gamma was achieved in three ways. If $\alpha > 1$ and the truncation point is left of the mode, then we simply sample from the truncated gamma and accept the first value to the right of the truncation point. If $\alpha > 1$ and the truncation point is to the right of the mode, then we employ a more sophisticated rejection sampling algorithm based on bounding the tail of the gamma by an exponential distribution. When $\alpha \leq 1$, we employ a similar rejection sampling algorithm based on the exponential distribution. The full details can be seen in the code below, and the paper these algorithms come from can be found in the references.

This code shows the whole Gibbs sampler:

```
# samples from the truncated gamma distribution
# so that X ~ truncated_gamma(cens, a, b) if
# P(X = x) = P(X' = x | X' >= cens) where
# X' ~ gamma(a, b)
# only works for scalars cens, a, b
truncated_gamma <- function(cens, a, b) {

    # max number of rejections before termination
    limit <- 1000000

    # There are three cases to consider:
    # a > 1:
    #   cens > (a - 1) / b (cens to the right of the mode)
    #   cens <= (a - 1) / b (cens to the left of the mode)
```

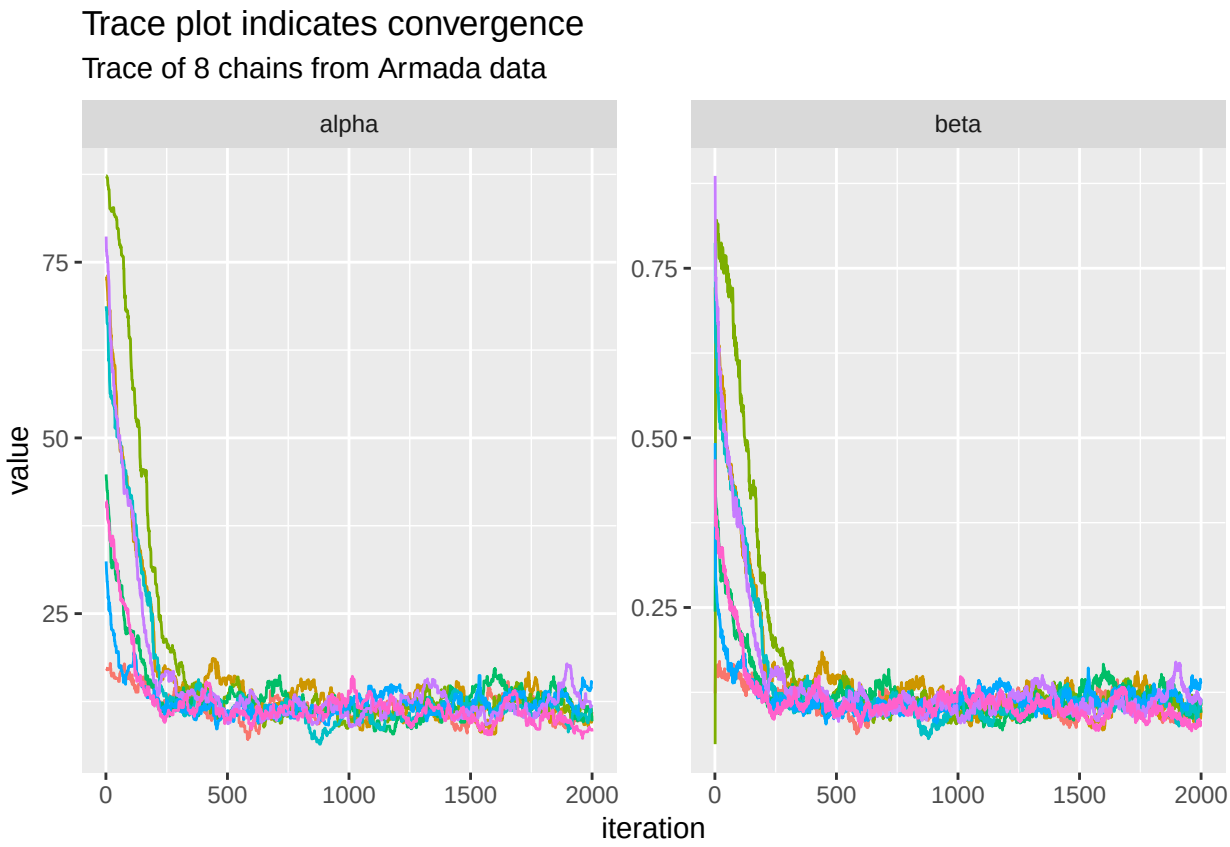
```
    # a <= 1

    # handles the degenerate case
    if (cens == 0) {
        return (rgamma(1, a, b))
    }

    if (a > 1) {
        if (cens <= (a - 1) / b) {
            # simple rejection scheme here
            x <- rgamma(1, a, b)
            counter <- 1
            while (x < cens) {
                x <- rgamma(1, a, b)
                counter <- counter + 1
                if (counter > limit) {
                    stop("limit reached in simple rejection scheme")
                }
            }
            #print(glue("simple rejection scheme: counter = {counter}"))
            return (x)
        } else {
            # algo 3.1 from Chung
            lambda <- 1 / (2 * cens) *
                (b * cens - a + sqrt((b*cens - a) ^ 2 + 4 * b * cens))
            lambda <- min(lambda, b)
            x <- rexp(1, lambda) + cens
            counter <- 1
            while (runif(1) > x ^ (a - 1) * exp(x * (lambda - b)) /
                   (((a - 1) / (b - lambda)) ^ (a - 1) * exp(1 - a))) {
                x <- rexp(1, lambda) + cens
                counter <- counter + 1
                if (counter > limit) {
                    stop("limit reached in algo 3.1")
                }
            }
            #print(glue("algo 3.1: counter = {counter}"))
            return (x)
        }
    } else {
        # algo 3.2
        lambda <- min(1 / cens, b)
        x <- rexp(1, lambda) + cens
        counter <- 1
        while (runif(1) > x ^ (a - 1) * exp(x * (lambda - b)) *
               cens ^ (1 - a) * exp(cens * (b - lambda))) {
            x <- rexp(1, lambda) + cens
            counter <- counter + 1
            if (counter > limit) {
                    print(glue("cens = {cens}, a = {a}, b = {b}"))
                    stop("limit reached in algo 3.2")
            }
        }
```

```
        #print(glue("algo 3.2: counter = {counter}"))
        return (x)
    }
}

# performs gibbs sampling where:
# x is a vector of fully observed data
# cens is a vector of censoring points for right censored data
# alpha_0, beta_0 are hyperparameters for gamma prior on alpha
# alpha_1, beta_1 are hyperparameters for gamma prior on beta
# sigma is the sd of the normal proposal distribution for the MH step for alpha
# n_sim is the total length of the desired chain
gibbs <- function(x, cens, alpha_0, beta_0, alpha_1, beta_1, sigma, n_sim) {

    # acceptance rate for MH steps
    acc_rate <- 0

    # the output keeps only the alpha and beta values
    alpha <- numeric(n_sim)
    beta <- numeric(n_sim)

    # a value we will repeatedly use, total sample size
    n <- length(x) + length(cens)

    # initialize alpha and beta
    # the initialization is intentionally dispersed for checking convergence
    alpha[1] <- runif(1, min = 0, max = 100)
    beta[1] <- runif(1, min = 0, max = 1)

    # run the chain
    for (i in 2:n_sim) {

        # sample latent variables
        # uses map to vectorize my truncated_gamma function
        z <- map_dbl(cens, ~truncated_gamma(., alpha[i - 1], beta[i - 1]))

        # sample beta
        beta[i] <- rgamma(1, alpha_1 + n * alpha[i - 1], beta_1 + sum(x) + sum(z))

        # now use one step of M-H to sample alpha
        proposed_alpha <- rnorm(1, mean = alpha[i - 1], sd = sigma)

        if (proposed_alpha < 0) {
            alpha[i] <- alpha[i - 1]
        } else {
            lp_old <- sum(dgamma(c(x, z), alpha[i - 1], beta[i], log = TRUE)) +
                dgamma(alpha[i - 1], alpha_0, beta_0, log = TRUE)
            lp_new <- sum(dgamma(c(x, z), proposed_alpha, beta[i], log = TRUE)) +
                dgamma(proposed_alpha, alpha_0, beta_0, log = TRUE)
            if (log(runif(1)) < lp_new - lp_old) {
                alpha[i] <- proposed_alpha
                acc_rate <- acc_rate + 1
            } else {
```

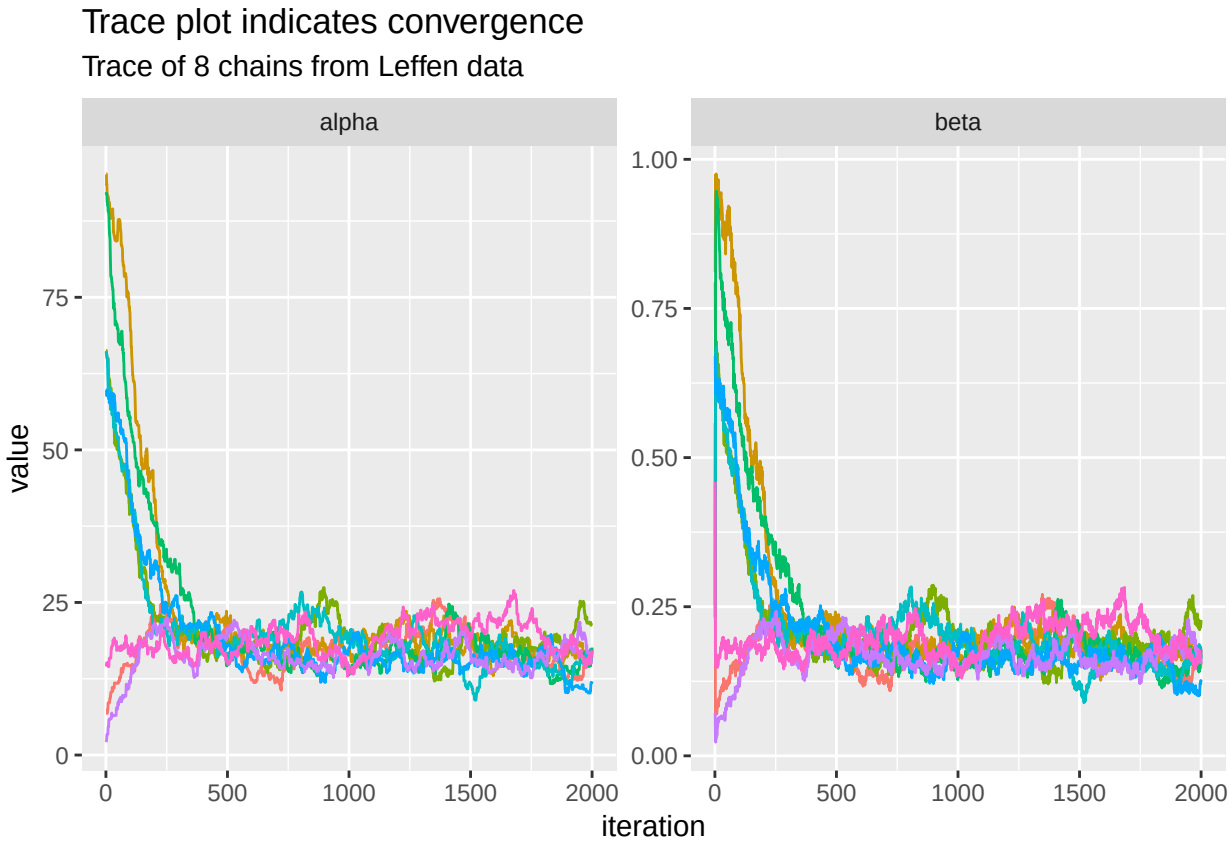
```
                alpha[i] <- alpha[i - 1]
            }
        }
    }

    # print acceptance rate for tuning reasons
    # print(acc_rate / n_sim)

    # package the output and return
    list(alpha = alpha, beta = beta)
}
```

Now we run the sampler to create 8 chains, each 2000 iterations long. I purposefully used many chains and over-dispersed the starting values so that I could gauge convergence. Below we see the trace plots for both α and β and for both Leffen and Armada. We can see informally that by around 500 iterations, the chains have converged: no matter where they started they are all overlapping.

```
x <- filter(data, player == "Armada", censored == "no")$percent
cens <- filter(data, player == "Armada", censored == "yes")$percent
chain_length <- 2000
num_chains <- 8
chains <- list()
for (i in 1:num_chains) {
    chains[[i]] <- as_tibble(gibbs(x, cens, 1, 1, 1, 1, 1, chain_length)) %>%
        mutate(iteration = 1:chain_length, chain = i)
}

armada_samples <- bind_rows(chains)

armada_samples %>%
    gather(key = key, value = value, alpha, beta) %>%
    ggplot(aes(iteration, value, color = as.factor(chain))) +
    geom_line() +
    facet_wrap(~key, scales = "free") +
    labs(title = "Trace plot indicates convergence",
         subtitle = "Trace of 8 chains from Armada data") +
    theme(legend.position = "none")
```




```
x <- filter(data, player == "Leffen", censored == "no")$percent
cens <- filter(data, player == "Leffen", censored == "yes")$percent
chain_length <- 2000
num_chains <- 8
chains <- list()
for (i in 1:num_chains) {
    chains[[i]] <- as_tibble(gibbs(x, cens, 1, 1, 1, 1, 1, chain_length)) %>%
        mutate(iteration = 1:chain_length, chain = i)
}

leffen_samples <- bind_rows(chains)

leffen_samples %>%
    gather(key = key, value = value, alpha, beta) %>%
    ggplot(aes(iteration, value, color = as.factor(chain))) +
    geom_line() +
    facet_wrap(~key, scales = "free") +
    labs(title = "Trace plot indicates convergence",
         subtitle = "Trace of 8 chains from Leffen data") +
    theme(legend.position = "none")
```



Thus, we remove the first 500 iterations from each chain. In the end, we get 12,000 samples per player, which is probably plenty to make the Monte Carlo error negligible for practical purposes.

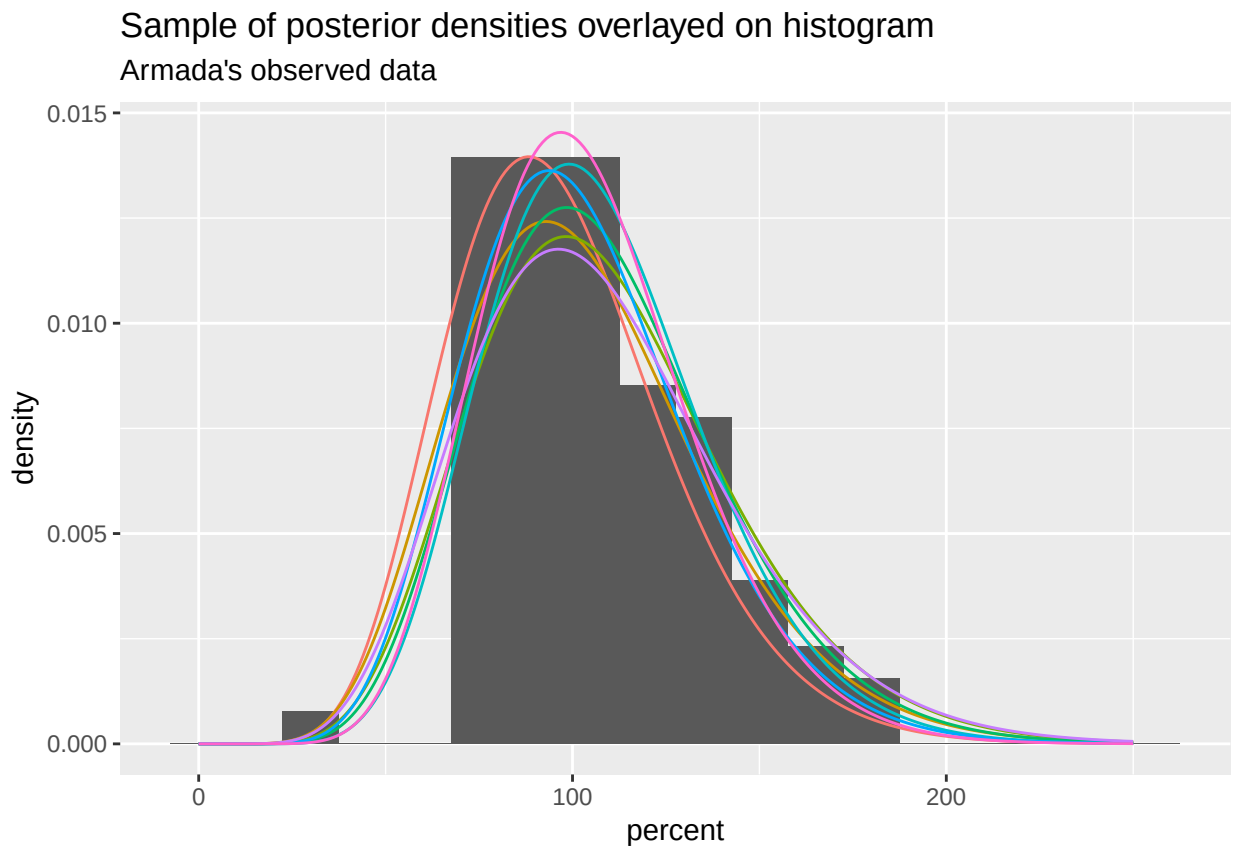
```
# remove first 500 samples from each chain (burn in)
armada_samples <- armada_samples %>%
    filter(iteration > 500)

leffen_samples <- leffen_samples %>%
    filter(iteration > 500)
```

Results

First, we briefly examine the fit of our model to the data. We do this by looking at the histograms again and overlaying some densities corresponding to parameters sampled from the posterior. We can see that the posterior provides perhaps a reasonable fit to Armada's uncensored data, but does not fit as well with Leffen's uncensored data. Leffen's data appears to have a higher peak than the posterior accounts for. Of course, this is without including the censored observations, so the conclusions should be taken with a grain of salt. Finally, I'll note that because of the large amount of data available for each player, the exact fit of the gamma distribution is not terribly important, for Central Limit Theorem type considerations.

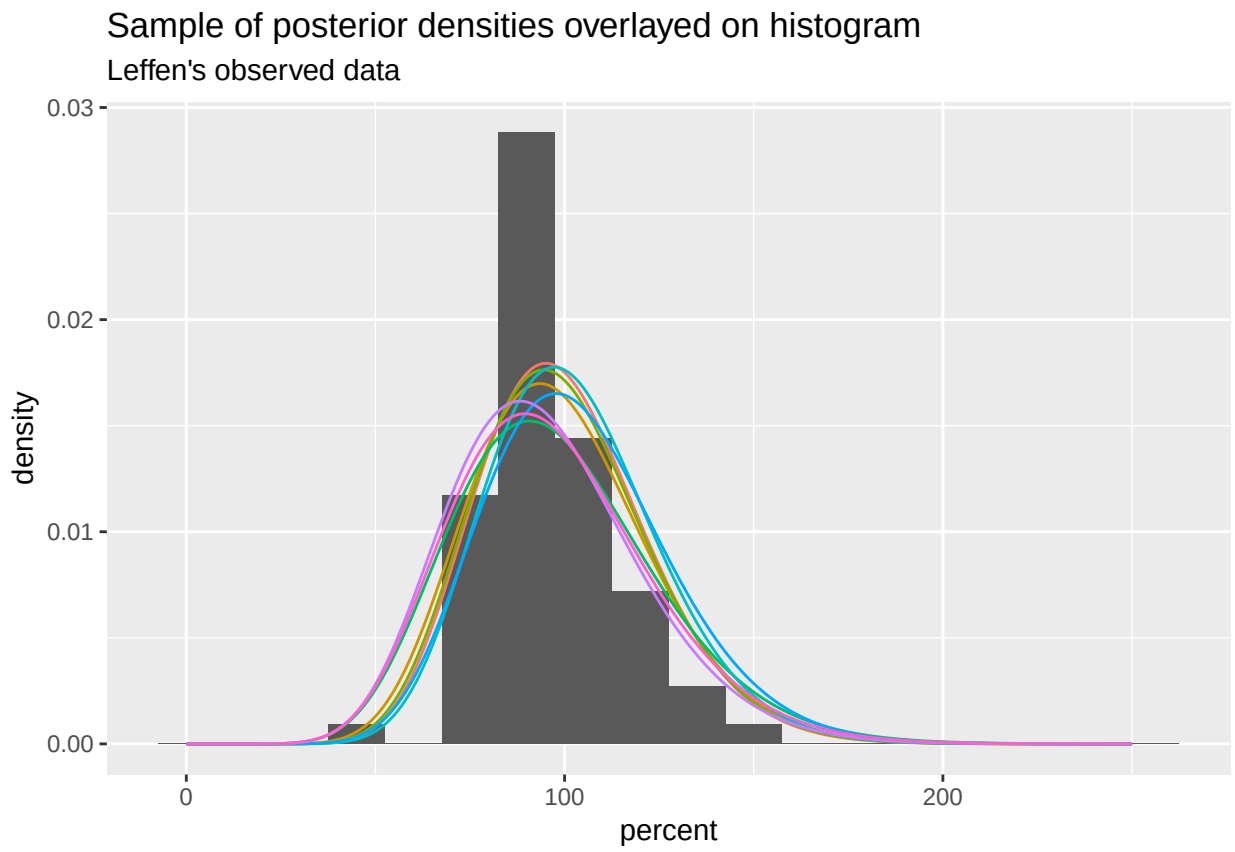
```
# overlay some densities on top of the observed data
ab <- armada_samples %>%
    filter(iteration == 1000) %>%
    select(alpha, beta)
```

```
x <- 0:250
curves <- list()
for (i in 1:8) {
    curves[[i]] <- dgamma(x, ab$alpha[i], ab$beta[i])
}

names(curves) <- c("1", "2", "3", "4", "5", "6", "7", "8")

curves <- curves %>%
    as_tibble %>%
    mutate(x = x) %>%
    gather(key = key, value = value, -x)

data %>%
    filter(player == "Armada", censored == "no") %>%
    ggplot(aes(percent)) +
    geom_histogram(aes(y = ..density..), binwidth = 15) +
    geom_line(data = curves, aes(x, value, color = key)) +
    theme(legend.position = "none") +
    labs(title = "Sample of posterior densities overlayed on histogram",
         subtitle = "Armada's observed data")
```




```
ab <- leffen_samples %>%
    filter(iteration == 1000) %>%
    select(alpha, beta)
```

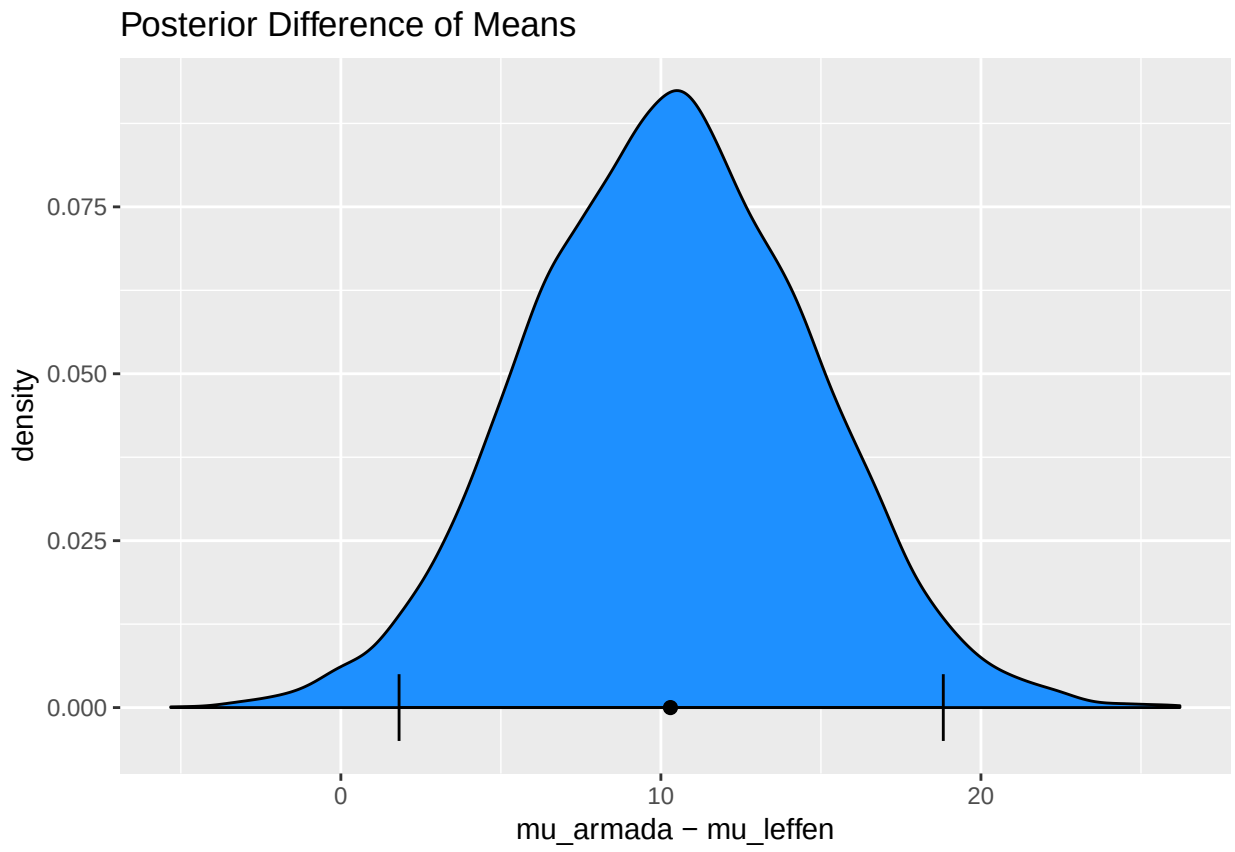
```r
x <- 0:250
curves <- list()
for (i in 1:8) {
    curves[[i]] <- dgamma(x, ab$alpha[i], ab$beta[i])
}

names(curves) <- c("1", "2", "3", "4", "5", "6", "7", "8")

curves <- curves %>%
    as_tibble %>%
    mutate(x = x) %>%
    gather(key = key, value = value, -x)

data %>%
    filter(player == "Leffen", censored == "no") %>%
    ggplot(aes(percent)) +
    geom_histogram(aes(y = ..density..), binwidth = 15) +
    geom_line(data = curves, aes(x, value, color = key)) +
    theme(legend.position = "none") +
    labs(title = "Sample of posterior densities overlayed on histogram",
         subtitle = "Leffen's observed data")
```



Here we get the main object of our analysis: the posterior distribution over the difference in means. We calculate a sample from this distribution by dividing the α and β distributions for each player and then subtracting these from each other. We subtracted Armada's mean from Leffen's mean. We plot a kernel density estimate below. We also have a dot indicating the median, and two vertical bars indicating the 2.5

and 97.5 percentiles. This gives a point estimate of about 10.3 percent, and a 95% credible interval of (1.8, 18.8). Since the 95% credible interval excludes zero, we can say that Leffen does get KOs at lower percent than Armada.

```
armada_mean <- armada_samples %>%
    mutate(mu = alpha / beta) %>%
    select(mu_armada = mu)

leffen_mean <- leffen_samples %>%
    mutate(mu = alpha / beta) %>%
    select(mu_leffen = mu)

diff_mean <- bind_cols(armada_mean, leffen_mean) %>%
    mutate(diff_mean = mu_armada - mu_leffen) %>%
    select(diff_mean)

# get quantiles for point estimate and credible interval
qs <- quantile(diff_mean$diff_mean, probs = c(.025, .5, .975))

# KDE of the posterior over difference in means
ggplot(data = tibble(x = qs[2], y = 0, lo = qs[1], hi = qs[3])) +
    geom_density(data = diff_mean, fill = "dodger blue", aes(diff_mean)) +
    geom_point(aes(x, y), size = 2) +
    geom_errorbarh(aes(y = y, height = .01, xmin = lo, xmax = hi)) +
    ggtitle("Posterior Difference of Means") +
    xlab("mu_armada - mu_leffen")
```



Discussion and Conclusion

It is interesting that Leffen gets KOs at lower percents than Armada, despite performing worse against Hungrybox. The 95% credible interval is quite wide though, ranging from a 1.8% difference to an 18.8% difference. So perhaps the real difference is closer to 1.8% than 10%. On the other hand, the discrepancy could be a sign of Armada's patience in the matchup. Maybe he is willing to rack up higher percents without doing anything risky to get the KO. This fits with popular perception. Of course, many other factors contribute to winning matches, so we should not ascribe too much importance to the simple investigation presented here.

Above I commented that although the gamma distribution may be a poor fit in some ways, it doesn't really matter since we have enough data for a Central Limit Theorem type effect. If this is true, couldn't we have just done some kind of simple t-test from the start? Yes, this would have worked fine I think, because we have lots of data, and not too many censored observation. In fact a little back-of-the-envelope calculation can reach very similar conclusions to those the Bayesian model finds. Let's say each player has around 81 uncensored observations, so we get a round square root. The original distribution for Armada had a mean of 107 and a standard deviation of about 27. Therefore, the sample mean has a mean of 107 and a standard deviation of around 3, since this is 27 divided by the square root of the sample size. Leffen's data has a mean of 96 and a standard deviation of roughly 18, so the sample mean has a mean of 96 and a standard deviation of 2. Therefore the difference in sample means has a mean of 11 and a standard deviation of $\sqrt{3^2 + 2^2} = \sqrt{13}$. Using the mean plus or minus two standard deviations as a 95% interval, we get (3.8, 18.8). All things considered, this is pretty close, especially since I spent about 2 minutes working this out, compared to hours and hours researching and writing the Gibbs sampler. So was it all a waste? I don't think so. In the end, the point of the project for me was not only to analyze the data effectively, but also to learn about MCMC and handling censored data.

References

1. Robert and Casella, *Introducing Monte Carlo Methods with R*, Chapter 7
2. Chung, *Simulation of truncated gamma variables*, Korean J. Comput. & Appl. Math. Vol. 5 (1998), No. 3, pp. 601 - 610

[1] Explains how to use Gibbs sampling for latent variables and missing data, and also explains how to use a step of the Metropolis algorithm within a Gibbs sampler.

[2] Explains how to sample from the truncated gamma efficiently.