STAT 1651 Final Project

Andrew Windsheimer

12/11/2021

(1) We begin by importing the data, removing players without sufficient at-bats, and reorganizing the columns for convenience.

```
bat <- data.frame(read.table("Bat.dat", header=T))
bat <- bat[bat$AB.1>10&bat$AB.2>10,]
bat <- bat[,c(1,2,4,3,5)]
```

Afterwards there remains

```
nrow(bat)

## [1] 491
```

players.

(2) Our Bayesian model for this part has the following form for a given player i in the first half of the season, given the Binomial(N_{i1}, p_i) likelihood for H_{i1} and a Normal(μ, σ^2) prior for p_i:

$$\begin{aligned} p(p_i|N_{i1}, H_{i1}) &\propto \frac{p(H_{i1}|N_{i1}, p_i)p(p_i)}{p(N_{i1}, p_i)} \\ &\propto p(H_{i1}|N_{i1}, p_i)p(p_i) \\ &\propto dbinom(H_{i1}, N_{i1}, p_i) * dnorm(p_i, \mu, \sigma^2) \end{aligned}$$

We apply a Normal prior for p_i so we can choose values μ and σ^2 that match our expectation. Using our prior knowledge of baseball, one would expect an average player to bat around .200, and it is reasonable to assume that nearly all players' true ability would fall within (.08, .320). Accordingly, we will use $\mu = .2$ and $\sigma^2 = .4^2$ and simplify our posterior as follows:

$$\begin{aligned} p(p_i|N_{i1}, H_{i1}) &\propto dbinom(H_{i1}, N_{i1}, p_i) * dnorm(p_i, \mu, \sigma^2) \\ &\propto dbinom(H_{i1}, N_{i1}, p_i) * dnorm(p_i, .2, .04^2) \\ &= \binom{N_{i1}}{H_{i1}} p_i^{H_{i1}}(1-p_i)^{H_{i1}} * \frac{1}{\sqrt{2\pi * .04}} \exp\left\{-\frac{1}{2 *. 04^2}(p_i - .2)^2\right\} \end{aligned}$$

Even if simplified further, this form is difficult to sample from, so we will use a Monte Carlo method to approximate its distribution. We will accomplish this by generating 1000 random iterates from the prior of p_i, $N(.2, .04^2)$, then, for each player in the analysis, plug into the posterior for p_i above using their given values of N_{i1} and H_{i1} along with the random iterates to compute a second-half prediction of batting average. Since the form of the posterior for p_i has the form of an exponential of a quadratic, we know it must be some

type of normal distribution. Accordingly, we can simply take the value with the largest likelihood to be the prediction.

(3) We will use a hierarchival model of the following form:

$$\begin{aligned} X_{i1} &\sim Normal(\theta_i, \sigma_{i1}^2) \\ \theta_i &\sim Normal(\mu, \tau^2) \\ \sigma_{i1}^2 &\sim Inv-Gamma(\nu_0/2, \nu_0\sigma_0^2/2) \\ \tau^2 &\sim Inv-Gamma(\eta_0/2, \eta_0\tau_0^2/2) \\ \mu &\sim Normal(\mu_0, \gamma_0^2) \end{aligned}$$

There are six parameters resulting from this model that we need to choose and justify. We will let $\nu_0 = 100{,}000$ and $\eta_0 = 1{,}000$, since we believe our prior knowledge to be quite informative. Based on our intuition from (2), we will center μ at $\mu_0 = \arcsin\sqrt{.2}$, or $\approx .4636$. To fit the range of (.08, .32) defined in (2), we will let our variance of μ be $\gamma_0^2 = ((\arcsin\sqrt{.32} - \arcsin\sqrt{.08})/6)^2 \approx .052^2$. We will let our within-group and between-group variability be slightly larger than this value, such that $\sigma_0^2 = \tau_0^2 = .07^2$.

We will employ the Markov Chain-Monte Carlo method by continuously sampling from the posterior conditional distributions of each θ_i and σ_{i1}^2, as well as τ^2 and μ, and updating their values accordingly at each step. We derive the posterior conditional distributions for each of these parameters from the following:

$$\begin{aligned} p(\theta_1, \dots, \theta_m, \mu, \tau^2, \sigma^2 | y_1, \dots, y_m) &\propto p(\mu, \tau^2, \sigma^2) p(\theta_1, \dots, \theta_m | \mu, \tau^2, \sigma^2) P(y_1, \dots, y_m | \theta_1, \dots, \theta_m, \mu, \tau^2, \sigma^2) \\ &= p(\mu)p(\tau^2)p(\sigma^2)\Pi_{j=1}^m p(\theta_j|\mu, \tau^2)\Pi_{j=1}^m \Pi_{i=1}^{n_j} p(y_{i,j}|\theta_j, \sigma^2) \end{aligned}$$

We first derive the full conditional for μ:

$$\begin{aligned} p(\mu|\theta_1, \dots, \theta_m, \tau^2, \sigma^2, y_1, \dots, y_m) &\propto p(\mu)\Pi_{j=1}^m p(\theta_j|\mu, \tau^2) \\ &\propto \exp\left\{-\frac{1}{2\gamma_0^2}(\mu^2 - 2\mu\mu_0)\right\} \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^m (\theta_i - \mu)^2\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left[\frac{1}{\gamma_0^2}(\mu^2 - 2\mu\mu_0 + \mu_0^2) + \frac{1}{\tau^2}(\sum\theta_i^2 - 2\mu\sum\theta_i + m\mu^2)\right]\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left[(\frac{1}{\gamma_0^2} + \frac{m}{\tau^2})\mu^2 - 2(\frac{\mu_0}{\gamma_0^2} + \frac{m\bar{\theta}}{\tau^2})\mu\right]\right\} \\ &\sim Normal(\frac{m\bar{\theta}/\tau^2 + \mu_0/\gamma_0^2}{m/\tau^2 + 1/\gamma_0^2}, \frac{1}{m/\tau^2 + 1/\gamma_0^2}) \end{aligned}$$

Full conditional for θ_i:

$$
\begin{aligned}
p(\theta_i|\mu,\tau^2,\sigma^2,y_1,\dots,y_m) &\propto p(\theta_i|\mu,\tau^2)\Pi_{j=1}^{n_i}p(y_{j,i}|\theta_i,\sigma_{i1}^2) \\
&\propto \exp\left\{-\frac{1}{2\tau^2}(\theta_i^2-2\mu\theta_i)\right\}\exp\left\{-\frac{1}{2\sigma_{i1}^2}\sum_{i=1}^{n_i}(y_{j,i}-\theta_i)^2\right\} \\
&\propto \exp\left\{-\frac{1}{2}[\frac{1}{\tau^2}(\theta_i^2-2\mu\theta_i)+\frac{1}{\sigma_{i1}^2}(\sum y_{j,i}^2-2\theta_i\sum y_{j,i}+n_i\theta_i^2)]\right\} \\
&\propto \exp\left\{-\frac{1}{2}[(\frac{1}{\tau^2}+\frac{n_i}{\sigma_{i1}^2})\theta_i^2-2(\frac{\mu}{\tau^2}+\frac{n_i\bar{y}_{i1}}{\sigma_{i1}^2})\theta_i]\right\} \\
&\sim Normal(\frac{n_i\bar{y}_{i1}/\sigma_{i1}^2+\mu/\tau^2}{n_i/\sigma_{i1}^2+1/\tau^2},\frac{1}{n_i/\sigma_{i1}^2+1/\tau^2})
\end{aligned}
$$

Full conditional for τ^2:

$$
\begin{aligned}
p(\tau^2|\mu,\theta,\sigma^2,y_1,\dots,y_m) &\propto p(\tau^2)\Pi_{i=1}^{m}p(\theta_i|\mu,\tau^2) \\
&\propto (\tau^2)^{-\eta_0/2+1}\exp\left\{-\frac{\eta_0\tau_0^2}{2\tau^2}\right\}\Pi_{i=1}^{m}(\tau^2)^{-1}\exp\left\{-\frac{1}{2\tau^2}(\theta_i-\mu)^2\right\} \\
&\propto (\tau^2)^{-\eta_0/2-m/2+1}\exp\left\{-\frac{1}{2\tau^2}(\eta_0\tau_0^2+\sum_{i=1}^{m}(\theta_i-\mu)^2)\right\} \\
&\sim inv-gamma(\frac{\eta_0+m}{2},\frac{\eta_0\tau_0^2+\sum_{i=1}^{m}(\theta_i-\mu)^2}{2})
\end{aligned}
$$

Full conditional for $sigma^2$:

$$
\begin{aligned}
p(\sigma^2|\mu,\theta,\tau^2,y_1,\dots,y_m) &\propto p(\sigma^2)\Pi_{j=1}^{m}\Pi_{i=1}^{n_j}p(y_{ij}|\theta_j,\sigma_j^2) \\
&\propto (\sigma^2)^{-\nu_0/2+1}\exp\left\{-\frac{\nu_0\sigma_0^2}{2\sigma^2}\right\}\Pi_{j=1}^{m}\Pi_{i=1}^{n_j}(\sigma^2)^{-1}\exp\left\{-\frac{1}{2\sigma^2}(y_{ij}-\theta_j)^2\right\} \\
&\propto (\sigma^2)^{-\nu_0/2-\sum n_j/2+1}\exp\left\{-\frac{1}{2\sigma^2}(\nu_0\sigma_0^2+\sum_{j=1}^{m}\sum_{i=1}^{n_j}(y_{ij}-\theta_j)^2)\right\} \\
&\sim inv-gamma(\frac{\nu_0-\sum_{j=1}^{m}n_j}{2},\frac{\nu_0\sigma_0^2+\sum_{j=1}^{m}\sum_{i=1}^{n_j}(y_{ij}-\theta_j)^2}{2})
\end{aligned}
$$

At the conclusion of the MCMC simulation, we will let the estimate for each player's batting average be the untransformed mean of the 10000 thetas sampled from their own respective distribution i.

(4)

(a) Following is the implementation of the Monte Carlo simulation described in (2):

```
set.seed(5)

iter <- 1000 #number of random iterates for p_i to be considered
```

```
p <- rnorm(iter, .2, .04) #generate iter random iterates for p_i

vals <- matrix(0, nrow=nrow(bat), ncol=iter) #matrix to hold likelihoods of
each p for each player i
est <- rep(0, nrow(bat)) #vector to hold estimates of p for each player i

for(j in 1:nrow(bat)) {
  #recall player i's at-bats and hits
  n <- bat$AB.1[[j]]
  h <- bat$H.1[[j]]

  #compute likelihood of each p given n and h
  vals[j,] <- dbinom(h, n, p)*dnorm(p, .2, .04)

  #store most likely value of p as estimate for player i
  est[[j]] <- p[[which.max(vals[j,])]]
}

bat$Q2 <- est #store the predictions in the data frame
```

To prevent pages and pages of values from being printed, we will display the predictions for only the first ten players in the dataset:

```
bat[c(1:10),c("AB.1", "H.1", "Q2")]
```

```
##    AB.1 H.1        Q2
## 1    18   5 0.2114121
## 2   287  52 0.1857704
## 5   190  29 0.1672792
## 6    16   1 0.1796360
## 8   206  36 0.1825343
## 10   71  14 0.1988259
## 11  225  46 0.2030686
## 16  299  64 0.2104270
## 17  101  21 0.2039982
## 19   46  13 0.2245505
```

(b) We now present the MCMC method described in (3):

```
library(invgamma)

#list of data for each player, including their first half
#at-bats, X_ij, and s2_ij
Y <- list(length(nrow(bat)))
for(i in 1:nrow(bat)){
  Y[[i]] <- c(bat$AB.1[[i]],
asin(sqrt((bat$H.1[[i]]+.25)/(bat$AB.1[[i]]+.5))), 1/(4*bat$AB.1[[i]]))
}

#transformed observations for each player, list in a string of 0s for non-
```

```r
hits
#and arcsin(sqrt(1))=pi/2s for hits
Z <- list(length(nrow(bat)))
for(j in 1:nrow(bat)){
  Z[[j]] <- c(rep(0, bat$AB.1[[j]]-bat$H.1[[j]]), rep(pi/2, bat$H.1[[j]]))
}

m<-length(Y) #number of groups = number of players
n<-sv<-ybar<-rep(NA,m)

#populate size, sample mean, sample variance vectors
for (j in 1:m) {
  n[j]<-Y[[j]][[1]]
  ybar[j]<-Y[[j]][[2]]
  sv[j]<-Y[[j]][[3]]
}

#priors
nu0<-1e5 ; s20<-.07^2
eta0<-1e3 ; t20<-.07^2
mu0<-.4636 ; g20<-.052^2

#assign initial values to each parameter
theta<-ybar; sigma2<-mean(sv)
mu<-mean(theta); tau2<-var(theta)

set.seed(5)
S<-5000 #iterations for MCMC algorithm

#matrices to hold estimates of parameters
THETA<-matrix(nrow=S, ncol=m)
SMT<-matrix(nrow=S, ncol=3)

# MCMC algorithm
for (s in 1:S) {

  # sample theta
  for(j in 1:m) {
    vtheta <- 1/(n[j] / sigma2+1/tau2)
    etheta <- vtheta*(ybar[j] * n[j] / sigma2+mu/tau2)
    theta[j] <- rnorm(1, etheta, sqrt(vtheta))
  }

  #sample sigma2
  nun <- nu0+sum(n)
  ss <- nu0*s20
  for(j in 1:m){
    ss <- ss+sum((Z[[j]] - theta[j])^2)
  }
```

```
  sigma2 <- 1/rgamma(1, nun/2, ss/2)

  #sample mu
  vmu<- 1/(m/tau2+1/g20)
  emu<- vmu*(m*mean(theta)/ tau2 + mu0/g20)
  mu<-rnorm(1, emu, sqrt(vmu))

  # sample tau2
  etam<-eta0+m
  ss<- eta0 * t20 + sum((theta-mu)^2)
  tau2<-1/rgamma(1, etam/2 , ss/2)

  #store results
  THETA[s,]<-theta
  SMT[s,]<-c(sigma2,mu,tau2)
}

#let the estimate for each player's batting average be
#the untransformed mean of the sampled thetas
bat$Q3 <- (sin(colMeans(THETA)))^2
```

Similarly to part (a), we will only display the predictions for the first ten players in the analysis:

```
bat[c(1:10),c("AB.1", "H.1", "Q3")]
```

```
##    AB.1 H.1        Q3
## 1    18   5 0.2028607
## 2   287  52 0.1800431
## 5   190  29 0.1554988
## 6    16   1 0.1360950
## 8   206  36 0.1733816
## 10   71  14 0.1874145
## 11  225  46 0.1992071
## 16  299  64 0.2084991
## 17  101  21 0.1977576
## 19   46  13 0.2303140
```

(5) We now calculate the MSE for each of our methods, as well as the MSE for simply using the frequentist estimate for batting average:

```
#declare the frequentist estimates of the batting averages from each half
bat$BA1 <- bat$H.1/bat$AB.1
bat$BA2 <- bat$H.2/bat$AB.2

#calc MSE
(MLE_MSE <- mean((bat$BA2-bat$BA1)^2))
```

```
## [1] 0.01266148
```

```
(Q2_MSE <- mean((bat$BA2-bat$Q2)^2))
```

```
## [1] 0.008924377
```

```
(Q3_MSE <- mean((bat$BA2-bat$Q3)^2))
```

```
## [1] 0.01080361
```

It is no surprise that our models perform better than the frequentist estimate, which likely struggles with players who have relatively few at-bats. The shrinkage provided by both of our Bayesian models accounts for the fact that extreme results are much more likely to present themselves in small sample sizes, and we accordingly weight our sample with our prior knowledge to attain a better estimate of batting average. With regards to why the method in (2) performs better than the method in (3), perhaps our prior assumptions for the hierarchical variances, while seeemingly appropriate, may not be sufficiently adequate estimates of the true values. The estimates generated pass the eye test, but there may be something else at play.

(6) For this final question, we will separately address our approaches to non-pitchers and pitchers, then compare the results at the end. The structure of both approaches are identical to what was previously used; the differences appear with regards to the priors implemented.

(a) In our first case, we deal with only non-pitchers. We would collectively expect non-pitchers to have a much higher batting average than pitchers, since it is much more of their focus to be good hitters. We therefore increase the presumed overall average to .22, maintaining the same variance of .04^2 such that the p_is are normally distributed and have nearly all of their values in (.1, .34). The code for the first method, Monte Carlo simulation directly follows:

```
##Monte Carlo sim##
batters <- bat[bat$Pitcher==0,]

iter <- 1000 #number of random iterates for p_i to be considered
p <- rnorm(iter, .22, .04) #generate iter random iterates for p_i

vals <- matrix(0, nrow=nrow(batters), ncol=iter) #matrix to hold likelihoods
of each p for each player i
est <- rep(0, nrow(batters)) #vector to hold estimates of p for each player i

for(j in 1:nrow(batters)) {
  #recall player i's at-bats and hits
  n <- batters$AB.1[[j]]
  h <- batters$H.1[[j]]

  #compute likelihood of each p given n and h
  vals[j,] <- dbinom(h, n, p)*dnorm(p, .22, .04)

  #store most likely value of p as estimate for player i
  est[[j]] <- p[[which.max(vals[j,])]]
}
```

```
batters$Q2 <- est #save predictions
```

The first ten predictions from this method:

```
batters[c(1:10),c("AB.1", "H.1", "Q2")]
```

```
##    AB.1 H.1        Q2
## 1    18   5 0.2282138
## 2   287  52 0.1909546
## 5   190  29 0.1746028
## 6    16   1 0.1980890
## 8   206  36 0.1890917
## 10   71  14 0.2107803
## 11  225  46 0.2093555
## 16  299  64 0.2156027
## 17  101  21 0.2140344
## 19   46  13 0.2380124
```

We now revisit the MCMC method, this time with only non-pitchers. Under the same assumptions described for the Monte Carlo method, we increase μ_0 to $\arcsin(\sqrt{.22}) \approx .488$ and set $\gamma_0^2 = ((\arcsin(\sqrt{.34}) - \arcsin(\sqrt{.1}))/6 \approx .05^2$. Setting ν_0 and η_0 as we did before, and also giving $sigma_0^2$ and τ_0^2 values slightly larger than γ_0^2 like in (3), we proceed as follows in the MCMC simulation:

```
##MCMC##

#list of data for each player, including their first half
#at-bats, X_ij, and s2_ij
Y <- list(length(nrow(batters)))
for(i in 1:nrow(batters)){
  Y[[i]] <- c(batters$AB.1[[i]],
asin(sqrt((batters$H.1[[i]]+.25)/(batters$AB.1[[i]]+.5))),
1/(4*batters$AB.1[[i]]))
}

#transformed observations for each player, list in a string of 0s for non-
hits
#and arcsin(sqrt(1))=pi/2s for hits
Z <- list(length(nrow(batters)))
for(j in 1:nrow(batters)){
  Z[[j]] <- c(rep(0, batters$AB.1[[j]]-batters$H.1[[j]]), rep(pi/2,
batters$H.1[[j]]))
}

m<-length(Y) #number of groups = number of players
n<-sv<-ybar<-rep(NA,m)

#populate size, sample mean, sample variance vectors
for (j in 1:m) {
```

```
  n[j]<-Y[[j]][[1]]
  ybar[j]<-Y[[j]][[2]]
  sv[j]<-Y[[j]][[3]]
}

#priors
nu0<-1e5 ; s20<-.06^2
eta0<-1e3 ; t20<-.06^2
mu0<-.488 ; g20<-.05^2

#assign initial values to each parameter
theta<-ybar; sigma2<-mean(sv)
mu<-mean(theta); tau2<-var(theta)

set.seed(5)
S<-5000 #iterations for MCMC algorithm

#matrices to hold estimates of parameters
THETA<-matrix(nrow=S, ncol=m)
SMT<-matrix(nrow=S, ncol=3)

# MCMC algorithm
for (s in 1:S) {

  # sample theta
  for(j in 1:m) {
    vtheta <- 1/(n[j] / sigma2+1/tau2)
    etheta <- vtheta*(ybar[j] * n[j] / sigma2+mu/tau2)
    theta[j] <- rnorm(1, etheta, sqrt(vtheta))
  }

  #sample sigma2
  nun <- nu0+sum(n)
  ss <- nu0*s20
  for(j in 1:m){
    ss <- ss+sum((Z[[j]] - theta[j])^2)
  }
  sigma2 <- 1/rgamma(1, nun/2, ss/2)

  #sample mu
  vmu<- 1/(m/tau2+1/g20)
  emu<- vmu*(m*mean(theta)/ tau2 + mu0/g20)
  mu<-rnorm(1, emu, sqrt(vmu))

  # sample tau2
  etam<-eta0+m
  ss<- eta0 * t20 + sum((theta-mu)^2)
  tau2<-1/rgamma(1, etam/2 , ss/2)
```

```
  #store results
  THETA[s,]<-theta
  SMT[s,]<-c(sigma2,mu,tau2)
}

batters$Q3 <- (sin(colMeans(THETA)))^2 #save predictions
```

The first ten predictions:

```
batters[c(1:10), c("AB.1", "H.1", "Q3")]
```

```
##    AB.1 H.1        Q3
## 1    18   5 0.2010043
## 2   287  52 0.1799663
## 5   190  29 0.1573505
## 6    16   1 0.1456367
## 8   206  36 0.1750203
## 10   71  14 0.1880572
## 11  225  46 0.1994971
## 16  299  64 0.2084075
## 17  101  21 0.1972654
## 19   46  13 0.2242176
```

The MSEs of the three methods based on non-pitchers only:

```
#declare the frequentist estimates of the batting averages from each half
batters$BA1 <- batters$H.1/batters$AB.1
batters$BA2 <- batters$H.2/batters$AB.2

#calc MSE
(MLE_MSE_b <- mean((batters$BA2-batters$BA1)^2))
```

```
## [1] 0.01230672
```

```
(Q2_MSE_b <- mean((batters$BA2-batters$Q2)^2))
```

```
## [1] 0.007929139
```

```
(Q3_MSE_b <- mean((batters$BA2-batters$Q3)^2))
```

```
## [1] 0.01067664
```

It is worth nothing here that the two Bayesian methods perform better than the frequentist estimate, with the Monte Carlo simulation performing the best. This again could be because it is a fundamentally simpler model than the MCMC algorithm, which may require a good bit of fine tuning to be accurate. Regardless, the Bayesian methods are able to pull back extreme values towards a global mean, something the frequentist estimate is incapable of doing.

(b) We now perform estimation with regards to only pitchers, whom we would generally expect to be much worse hitters than non-pitchers. With this in mind, we decrease the

expected batting average to .1, using a variance of .04^2 and letting the p_is for pitchers be normally distributed with nearly all values contained in (.01, .19).

```
##Monte Carlo sim##
pitchers <- bat[bat$Pitcher==1,]

iter <- 1000 #number of random iterates for p_i to be considered
p <- rnorm(iter, .1, .03) #generate iter random iterates for p_i

vals <- matrix(0, nrow=nrow(pitchers), ncol=iter) #matrix to hold likelihoods
of each p for each player i
est <- rep(0, nrow(pitchers)) #vector to hold estimates of p for each player
i

for(j in 1:nrow(pitchers)) {
  #recall player i's at-bats and hits
  n <- pitchers$AB.1[[j]]
  h <- pitchers$H.1[[j]]

  #compute likelihood of each p given n and h
  vals[j,] <- dbinom(h, n, p)*dnorm(p, .1, .03)

  #store most likely value of p as estimate for player i
  est[[j]] <- p[[which.max(vals[j,])]]
}

pitchers$Q2 <- est #save predictions
```

The first ten predictions from this method:

```
pitchers[c(1:10),c("AB.1", "H.1", "Q2")]
```

```
##     AB.1 H.1         Q2
## 20    15   1 0.09549867
## 29    28   4 0.10885783
## 44    31   2 0.09092143
## 54    17   2 0.10251366
## 96    32   2 0.09023088
## 115   32   3 0.09847632
## 116   35   1 0.07829556
## 154   21   1 0.09023088
## 156   29   4 0.10805097
## 192   34   3 0.09702219
```

Here we again present the MCMC method, this time with only non-pitchers. Under the same assumptions described for the Monte Carlo method, we increase μ_0 to $\arcsin(\sqrt{.1}) \approx .3217$ and set $\gamma_0^2 = ((\arcsin(\sqrt{.19}) - \arcsin(\sqrt{.01}))/6 \approx .06^2$. Setting ν_0 and η_0 as we did before, and also giving $sigma_0^2$ and τ_0^2 values slightly larger than γ_0^2 like in (3), we proceed as follows in the MCMC simulation:

```r
##MCMC##

#list of data for each player, including their first half
#at-bats, X_ij, and s2_ij
Y <- list(length(nrow(pitchers)))
for(i in 1:nrow(pitchers)){
  Y[[i]] <- c(pitchers$AB.1[[i]],
asin(sqrt((pitchers$H.1[[i]]+.25)/(pitchers$AB.1[[i]]+.5))),
1/(4*pitchers$AB.1[[i]]))
}

#transformed observations for each player, list in a string of 0s for non-
hits
#and arcsin(sqrt(1))=pi/2s for hits
Z <- list(length(nrow(pitchers)))
for(j in 1:nrow(pitchers)){
  Z[[j]] <- c(rep(0, pitchers$AB.1[[j]]-pitchers$H.1[[j]]), rep(pi/2,
pitchers$H.1[[j]]))
}

m<-length(Y) #number of groups = number of players
n<-sv<-ybar<-rep(NA,m)

#populate size, sample mean, sample variance vectors
for (j in 1:m) {
  n[j]<-Y[[j]][[1]]
  ybar[j]<-Y[[j]][[2]]
  sv[j]<-Y[[j]][[3]]
}

#priors
nu0<-1e5 ; s20<-.07^2
eta0<-1e3 ; t20<-.07^2
mu0<-.3217 ; g20<-.06^2

#assign initial values to each parameter
theta<-ybar; sigma2<-mean(sv)
mu<-mean(theta); tau2<-var(theta)

set.seed(5)
S<-5000 #iterations for MCMC algorithm

#matrices to hold estimates of parameters
THETA<-matrix(nrow=S, ncol=m)
SMT<-matrix(nrow=S, ncol=3)

# MCMC algorithm
for (s in 1:S) {
```

```
  # sample theta
  for(j in 1:m) {
    vtheta <- 1/(n[j] / sigma2+1/tau2)
    etheta <- vtheta*(ybar[j] * n[j] / sigma2+mu/tau2)
    theta[j] <- rnorm(1, etheta, sqrt(vtheta))
  }

  #sample sigma2
  nun <- nu0+sum(n)
  ss <- nu0*s20
  for(j in 1:m){
    ss <- ss+sum((Z[[j]] - theta[j])^2)
  }
  sigma2 <- 1/rgamma(1, nun/2, ss/2)

  #sample mu
  vmu<- 1/(m/tau2+1/g20)
  emu<- vmu*(m*mean(theta)/ tau2 + mu0/g20)
  mu<-rnorm(1, emu, sqrt(vmu))

  # sample tau2
  etam<-eta0+m
  ss<- eta0 * t20 + sum((theta-mu)^2)
  tau2<-1/rgamma(1, etam/2 , ss/2)

  #store results
  THETA[s,]<-theta
  SMT[s,]<-c(sigma2,mu,tau2)
}

pitchers$Q3 <- (sin(colMeans(THETA)))^2 #save predictions
```

The first ten predictions:

```
pitchers[c(1:10), c("AB.1", "H.1", "Q3")]
```

```
##     AB.1 H.1         Q3
## 20    15   1 0.08165545
## 29    28   4 0.14549136
## 44    31   2 0.07228193
## 54    17   2 0.12501613
## 96    32   2 0.07025331
## 115   32   3 0.09938738
## 116   35   1 0.03694629
## 154   21   1 0.05990406
## 156   29   4 0.14097811
## 192   34   3 0.09398700
```

The MSEs of the three methods based on pitchers only:

```
#declare the frequentist estimates of the batting averages from each half
pitchers$BA1 <- pitchers$H.1/pitchers$AB.1
pitchers$BA2 <- pitchers$H.2/pitchers$AB.2

#calc MSE
(MLE_MSE_p <- mean((pitchers$BA2-pitchers$BA1)^2))

## [1] 0.01520985

(Q2_MSE_p <- mean((pitchers$BA2-pitchers$Q2)^2))

## [1] 0.01103889

(Q3_MSE_p <- mean((pitchers$BA2-pitchers$Q3)^2))

## [1] 0.01382162
```

The same pattern observed in the previous two iterations of these methods appears once again. The Monte Carlo simulation method triumphs over the MCMC algorithm, which is better than the frequentist method. The simplicity of the Monte Carlo simulation may ultimately be the reason it succeeds, and the Bayesian models together continue their run of excellence because of their ability to implement prior knowledge.

(c) To bring everything together, here are the MSEs for each of the three prediction methods, with regards to the dataset and both of its subsets:

```
MLE_MSE

## [1] 0.01266148

Q2_MSE

## [1] 0.008924377

Q3_MSE

## [1] 0.01080361

MLE_MSE_b

## [1] 0.01230672

Q2_MSE_b

## [1] 0.007929139

Q3_MSE_b

## [1] 0.01067664

MLE_MSE_p

## [1] 0.01520985
```

```
Q2_MSE_p

## [1] 0.01103889

Q3_MSE_p

## [1] 0.01382162
```

The same relative ranking occurs between the methods with each dataset, although we do see that predicting the pitcher's batting average is a fundamentally more difficult problem than predicting non-pitchers or all batters together. Regardless, the frequentist estimates can be viewed as a quick-and-dirty baseline that fail to implement any prior knowledge. This is the reason why the Bayesian methods are continually more successful: we can leverage our ideas about what we would expect the distribution of batting averages to look like, and build a model that reflects these ideas, instead of being completely controlled by the data. The shrinkage provided by the Bayesian methods inherently represents regression towards the mean. It is very unlikely for players to maintain batting averages well above or below the expected mean for very long, and one would think there is significant value in implementing this idea into our model. These MSEs certainly support this, and one can be quite sure that the Bayesian models are an excellent way to approach this prediction problem.