PSTAT 115 Homework 6

Aaron Barel / Kevin Ayala

December 8, 2018

1

(a)

$logit(\theta_i(x_i)) = \alpha + \beta x_i$

$log(\frac{\theta_i(x_i)}{1-\theta_i(x_i)}) = \alpha + \beta x_i$
$\rightarrow \frac{\theta_i(x_i)}{1-\theta_i(x_i)} = e^{\alpha+\beta x_i}$
$\rightarrow \theta_i(x_i) = e^{\alpha+\beta x_i} - (\theta_i(x_i))e^{\alpha+\beta x_i}$
$\rightarrow \theta_i(x_i)(1 + e^{\alpha+\beta}) = e^{\alpha+\beta x_i}$
$\theta_i(x_i) = \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}$

$y_i \sim Bin(\theta_i(x_i), n_i)$
$y_i \sim Bin(\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}, n_i)$
$\prod_{i=1}^{N} p(y_i|\theta_i(x_i), n_i = \prod_{i=1}^{N} p(y_i|\alpha, \beta, x_i, n_i)$
$\prod_{i=1}^{N} \binom{n_i}{y_i}(\frac{e^{\alpha+\beta}}{1+e^{\alpha+\beta}})^{y_i}(1 - \frac{e^{\alpha+\beta}}{1+e^{\alpha+\beta}})^{n_i-y_i}$
$= \prod_{i=1}^{N} \binom{n_i}{y_i}(e^{\alpha+\beta})^{y_i}(\frac{1}{1+e^{\alpha+\beta}})^{y_i}(\frac{1}{1+e^{\alpha+\beta}})^{n_i-y_i}$
$= \prod_{i=1}^{N} \binom{n_i}{y_i}(e^{\alpha+\beta})^{y_i}(\frac{1}{1+e^{\alpha+\beta}})^{n_i}$

(b)

$\theta_i(x_i) = .5$
$\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}} = .5$
$e^{\alpha+\beta x_i} = .5 + .5e^{\alpha+\beta x_i}$
$.5e^{\alpha+\beta x_i} = .5$
$e^{\alpha+\beta x_i} = 1$
$\alpha + \beta x_i = \ln(1)$
$\ln(1) = 0$
lethal when $\beta x_i = -\alpha \rightarrow x_i = -\frac{\alpha}{\beta}$

(c)

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------- tidyverse 1.2.1 --
## ✓ ggplot2 3.0.0     ✓ purrr   0.2.5
## ✓ tibble  1.4.2     ✓ dplyr   0.7.6
```

```
## ✓ tidyr   0.8.1     ✓ stringr 1.3.1
## ✓ readr   1.1.1     ✓ forcats 0.3.0
```

```
## -- Conflicts ---------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

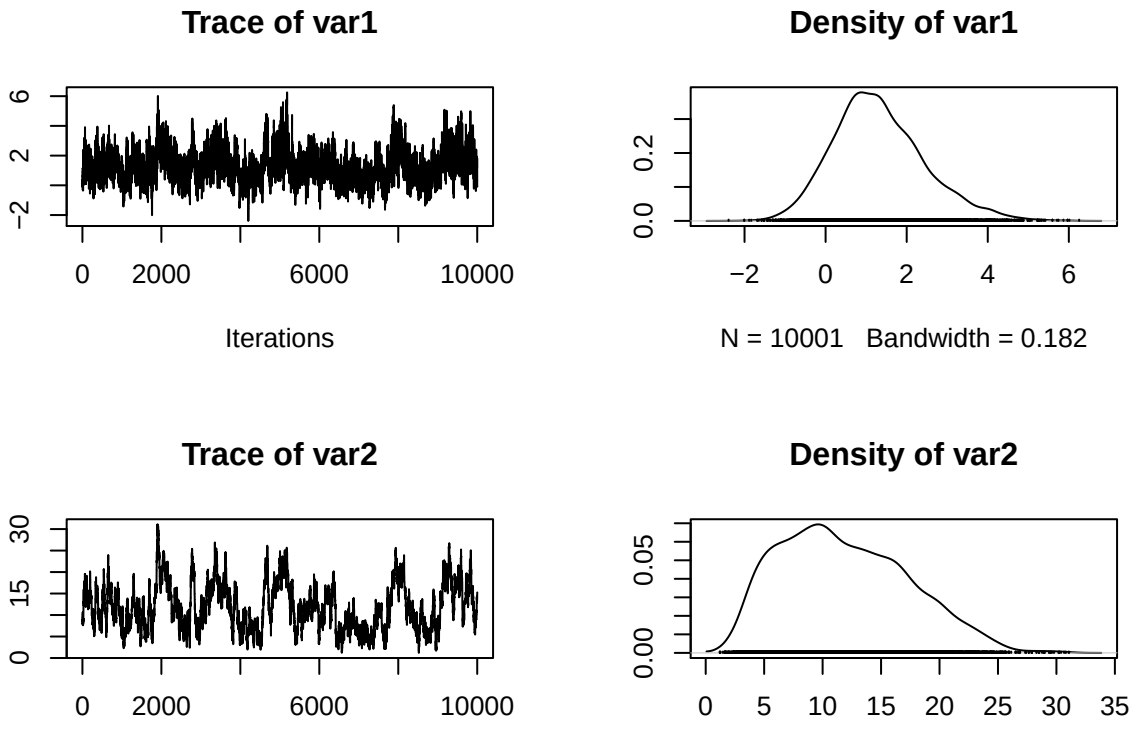
```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(coda)

x <- c(-0.86, -0.3, -0.05, 0.73)
n <- rep(5, 4)
y <- c(0, 1, 3, 5)

set.seed(1)
#1c

logp = function(theta, y, x, n) {
 alpha = theta[1]
 beta = theta[2]
 prob = exp((alpha+(beta*x)))/(1+exp((alpha+(beta*x))))
 return(sum(dbinom(y,n,prob, log = TRUE)))
}

metropolis = function(theta_s, cov, y, x, n) {
# Note: theta_s is a vector now!
theta_p = mvrnorm(1, theta_s, cov)
if(logp(theta_p, y, x, n) > logp(theta_s, y, x, n)) {
  return(theta_p)
} else {
  r = runif(1)
    if(log(r) < logp(theta_p, y, x, n) - logp(theta_s, y, x, n)) {
      return(theta_p)
      } else {
        return(theta_s)
            }
              }
}
# theta_0 is your initial parameter guess
# burnin is number of burnin samples
# maxit are the number of samples you generate (post burnin)
# cov is the proposal covariance matrix
rw_metrop_multi = function(theta_0, burnin, maxit, cov, y, x, n) {
  samples1 = matrix(0, ncol = length(theta_0), nrow = burnin + maxit)
  samples1[1,] = theta_0
  for(i in 1:(nrow(samples1) - 1)) {
    samples1[i + 1,] = metropolis(samples1[i,], cov, y, x, n)
  }
```

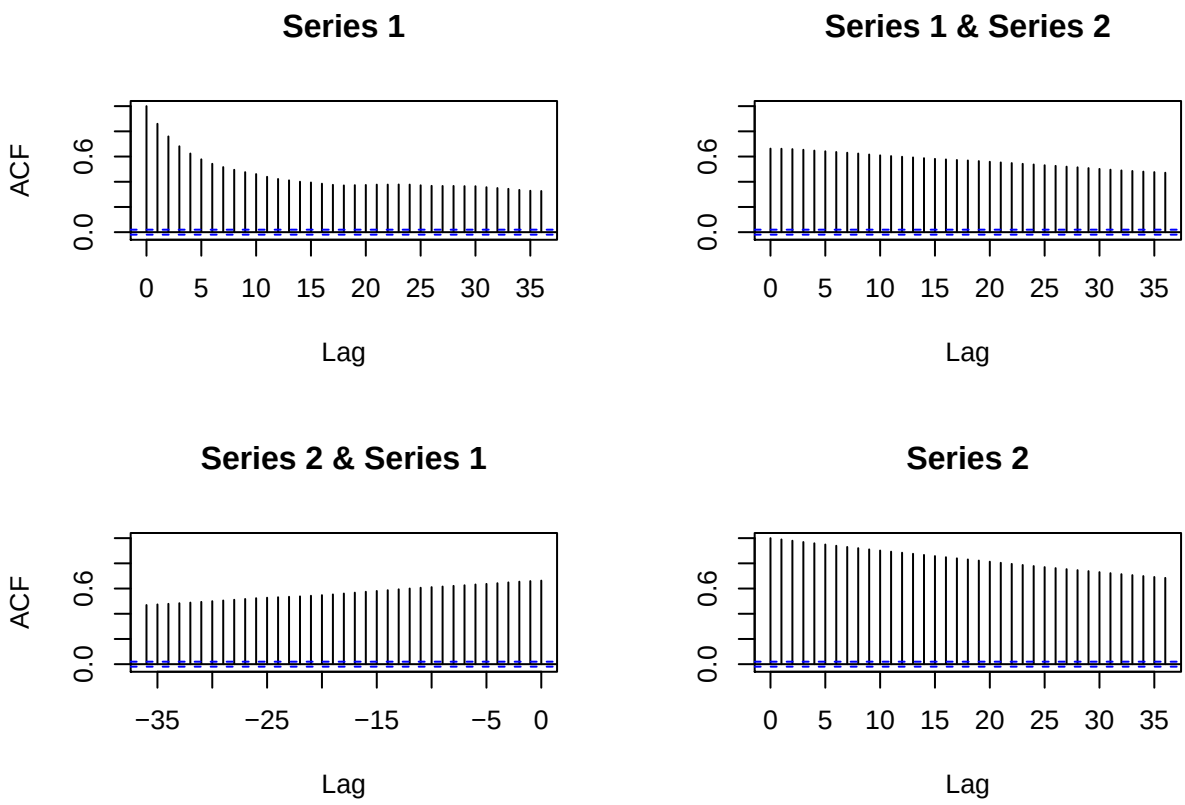
```
  samples1[burnin:(burnin + maxit),]
}

samples1 = rw_metrop_multi(c(10,10), 1000, 10000, matrix(c(1.0, 0.0, 0.0, 1.0), nrow = 2), y, x, n)

plot(as.mcmc(samples1))
```




```
effectiveSize(samples1)
```

```
##      var1      var2
## 146.42704  51.47561
```

```
acf(samples1)
```




```
acceptancerate<-1- rejectionRate(as.mcmc(samples1))
acceptancerate
```

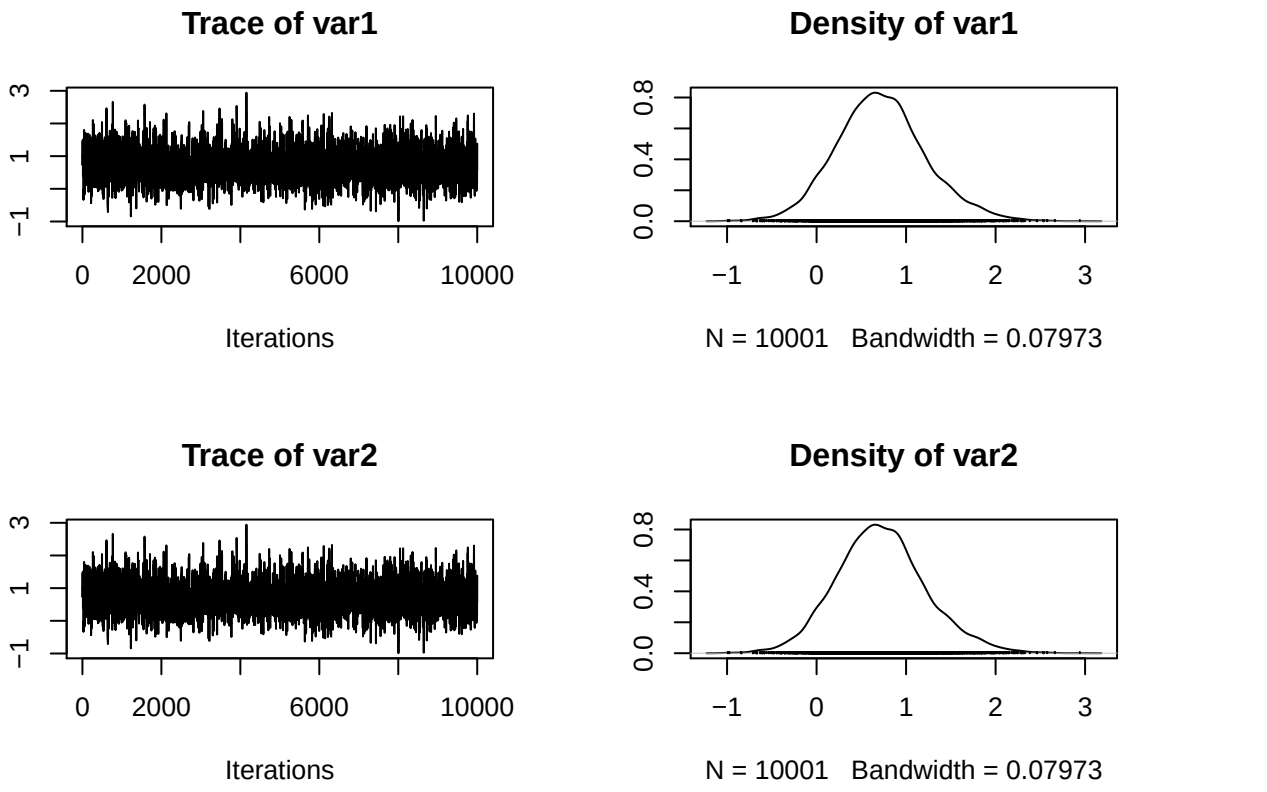
```
##   var1   var2
## 0.6346 0.6346
```

```
#acceptance rate is .6256 for alpha and beta
```

(d)

```
set.seed(2)
logp = function(theta, y, x, n) {
 alpha = theta[1]
 beta = theta[2]
 prob = exp((alpha+(beta*x)))/(1+exp((alpha+(beta*x))))
 return(sum(dbinom(y,n,prob, log = TRUE)))
}

metropolis = function(theta_s, cov, y, x, n) {
# Note: theta_s is a vector now!
theta_p = mvrnorm(1, theta_s, cov)
if(logp(theta_p, y, x, n) > logp(theta_s, y, x, n)) {
  return(theta_p)
} else {
  r = runif(1)
    if(log(r) < logp(theta_p, y, x, n) - logp(theta_s, y, x, n)) {
      return(theta_p)
      } else {
```

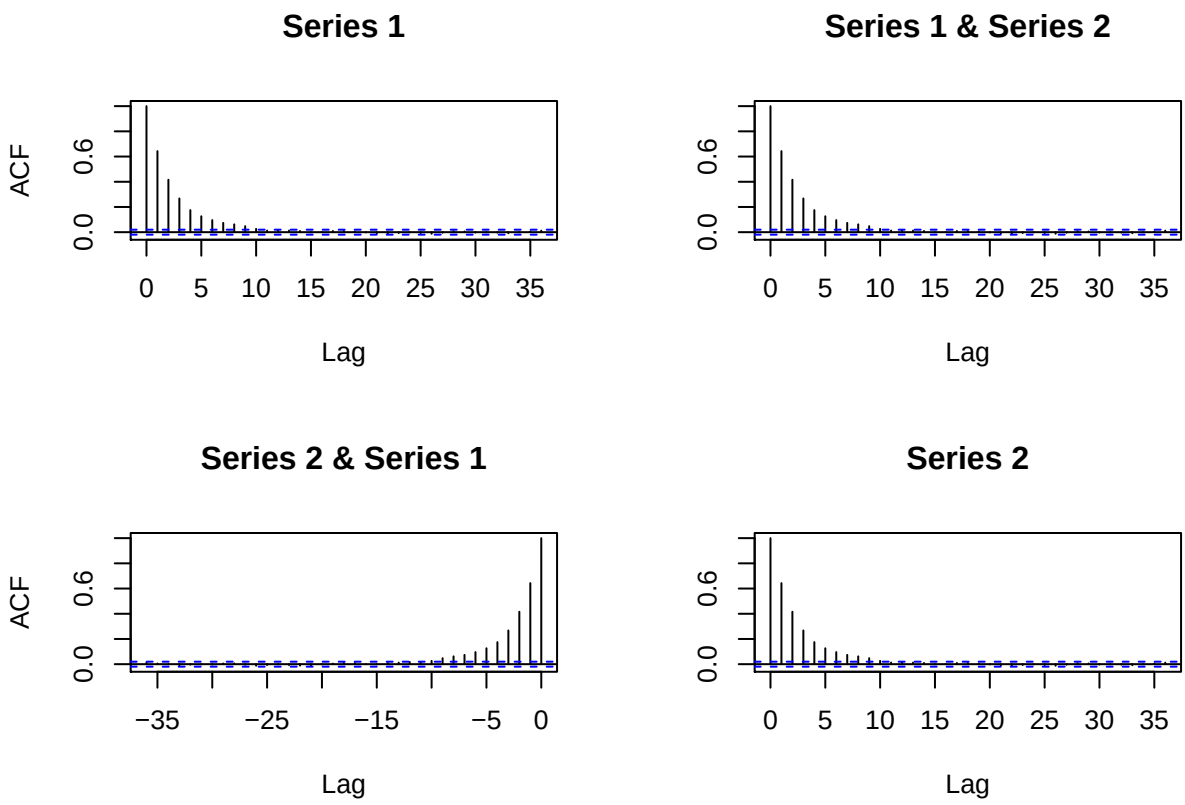
```
        return(theta_s)
            }
              }
}
# theta_0 is your initial parameter guess
# burnin is number of burnin samples
# maxit are the number of samples you generate (post burnin)
# cov is the proposal covariance matrix
rw_metrop_multi = function(theta_0, burnin, maxit, cov, y, x, n) {
  samples = matrix(0, ncol = length(theta_0), nrow = burnin + maxit)
  samples[1,] = theta_0
  for(i in 1:(nrow(samples) - 1)) {
    samples[i + 1,] = metropolis(samples[i,], cov, y, x, n)
  }
  samples[burnin:(burnin + maxit),]
}

samples = rw_metrop_multi(c(10,10), 1000, 10000, matrix(c(1, 1, 1, 1), nrow = 2), y, x, n)

plot(as.mcmc(samples))
```




```
effectiveSize(samples) #effective size
```

```
##     var1     var2
## 2170.735 2170.735
```

```
acf(samples)
```



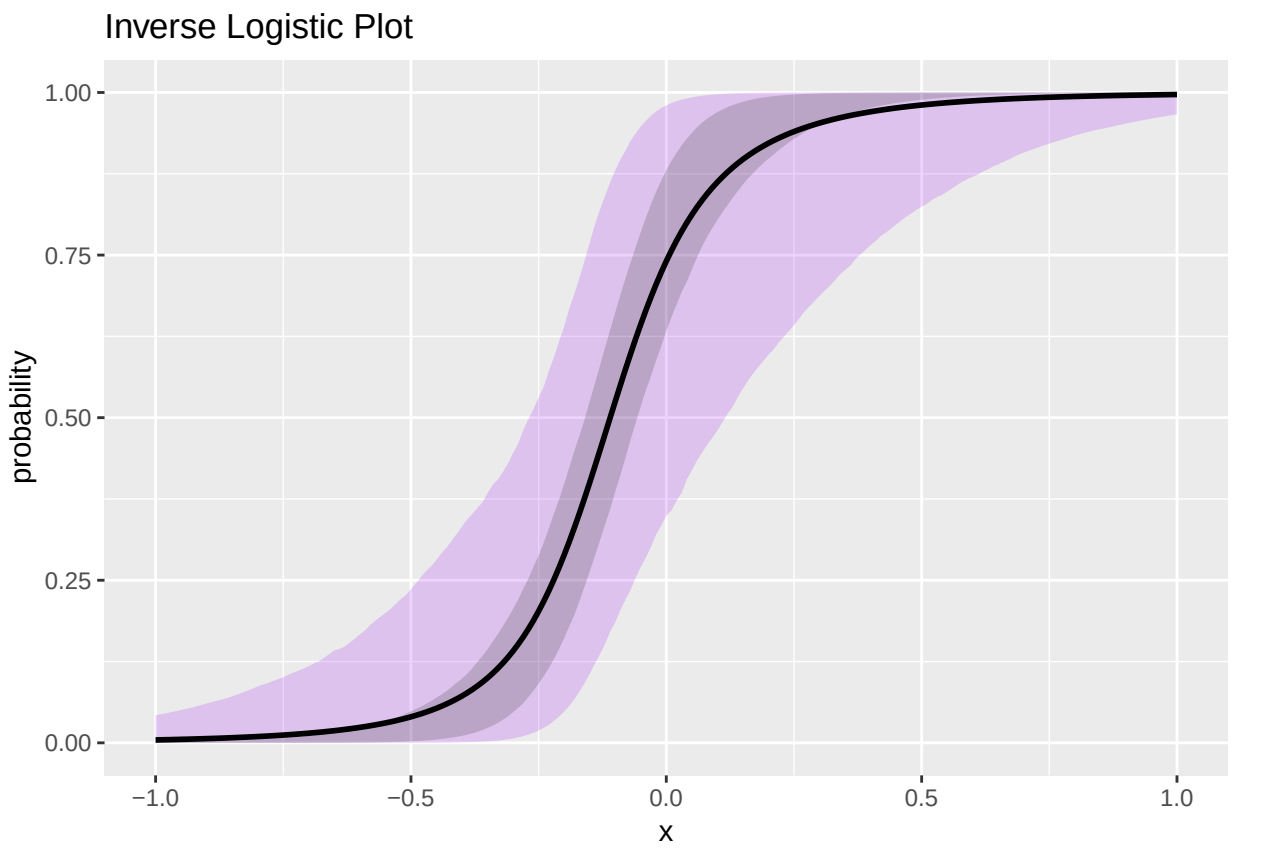

```
acceptancerate2<-1- rejectionRate(as.mcmc(samples))
acceptancerate2
```

```
##    var1    var2
## 0.4874 0.4874
```

```
#acceptance rate is about 50%, which is close to the sweet spot of 30-40%, when
#when p = sigma_alpha = sigma_beta. We conclude it is passable
```

(e)

```
# Make up some hypothethical xs to plot at. Sometimes you use xs from your data from this, sometimes yo
# You'll need to change this for the homework
xgrid <- seq(-1, 1, by = 0.01)
# Compute the function we want to understand the uncertainty of (as a function of the different paramet
compute_curve <- function(samp) {
alpha <- samp[1]
beta <- samp[2]
exp(alpha + beta * xgrid) / (1 + exp(alpha + beta*xgrid))
}

probability <- apply(samples1, 1, compute_curve)
quantiles <- apply(probability, 1, function(x) quantile(x, c(0.025, 0.25, 0.75, 0.975)))
quantiles <- t(quantiles)
probability <- rowMeans(probability)
data.frame(x=xgrid, q025=quantiles[, 1], q25=quantiles[, 2], q75=quantiles[,3], q975=quantiles[, 4], mea
geom_ribbon(aes(x=x, ymin=q25, ymax=q75), alpha= 0.2) +
geom_line(aes(x=x, y = probability), size = 1)+ggtitle("Inverse Logistic Plot")
```



2

```
baseball_data <- read_csv("lad.csv", col_types=cols())
## observed hits in the first 2 months
y <- baseball_data$y
## observed at bats in the first 2 months
n <- baseball_data$n
## observed batting average in the first 2 months (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
```

(a)

```
# theta pool is sum of at bats divided by sum of attempts
theta_pool <- sum(y) / sum(n)

# rmse for theta mle
sqrt(sum((theta_mle - val)^2) /J)
```

```
## [1] 0.02479514
```

```
# rmse for theta_pool
sqrt(sum((theta_pool - val)^2) /J)
```

```
## [1] 0.02779054
```

No pooling gave a better RMSE ($0.02479514 < 0.02779054$), so we will concludde no pooling gives a better estimate than complete pooling for end-of-year batting averages.

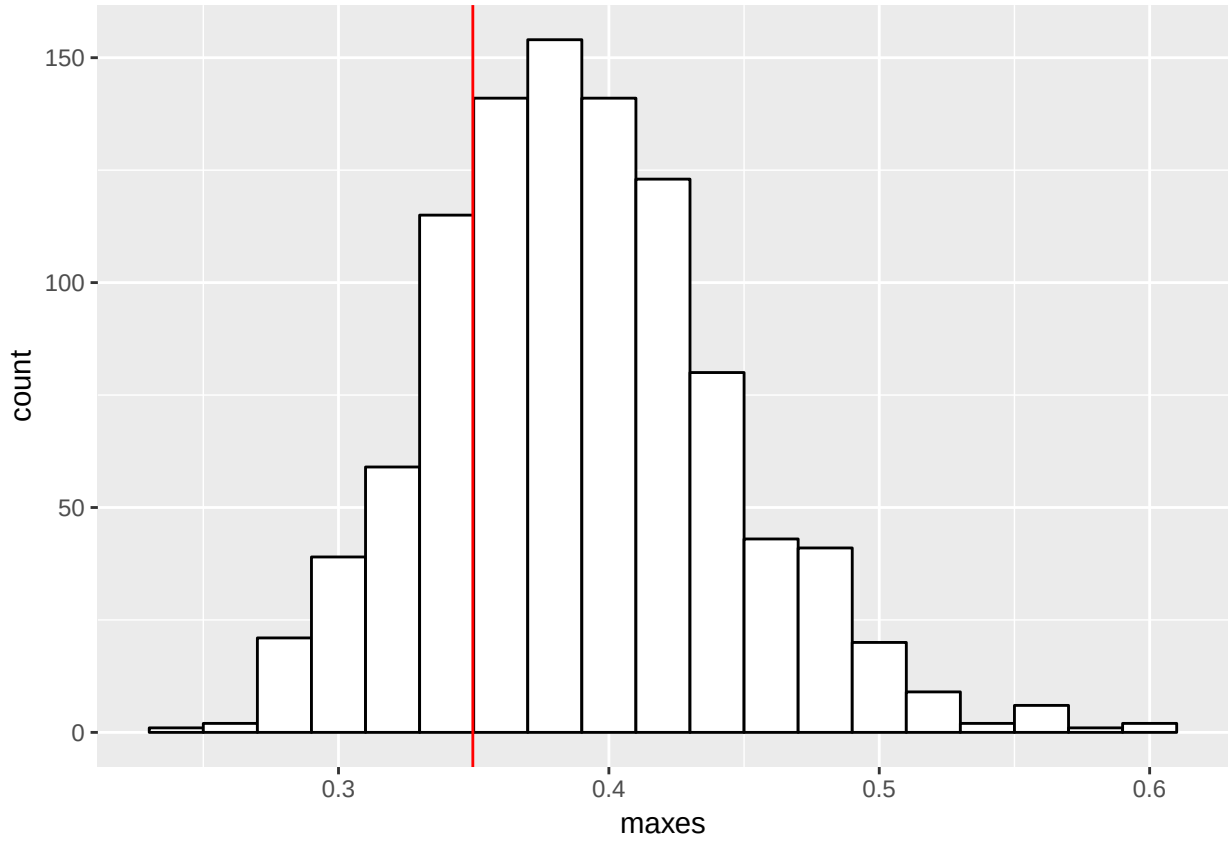
(b)

```
set.seed(69)
# writing function for prior predictive distribution of theta_mle
pred_prior <- function(observations, mu, theta, tau, n){
  # observations is how many we want (in our problem 1000)
  # mu is prior mean (in our case theta_pool)
  # theta is theta_i drawn from rnorm(1, mu, 0.05)
  # tau is prior standard devation (in our case 0.05)
  # n is at bats each player
  # predictive prior is a Normal with mean = prior mean and variance = data variance + prior variance
  # rnorm uses standard deviation so we will use sqrt
  # for this example the data sd is approximated using central limit theorem for binomial distribution
  data_sd <- sqrt(theta * (1-theta) / n)
  return(rnorm(observations, mu, data_sd + tau))
}

#theta_mle.df <- data.frame(matrix(ncol = 10, nrow = 1000))
#colnames(theta_mle.df) <- baseball_data$names

theta_mle.df <- setNames(data.frame(matrix(ncol = 10, nrow = 1000)), baseball_data$name)
for (i in 1:J){
  theta_i <- mean(rnorm(1000, theta_pool, 0.05))
  theta_i_mle <- pred_prior(1000, theta_pool, theta_i, 0.05, n[i])
  theta_mle.df[,i] <- theta_i_mle
}

colnames(theta_mle.df) <- baseball_data$name

as.tibble(theta_mle.df)
```

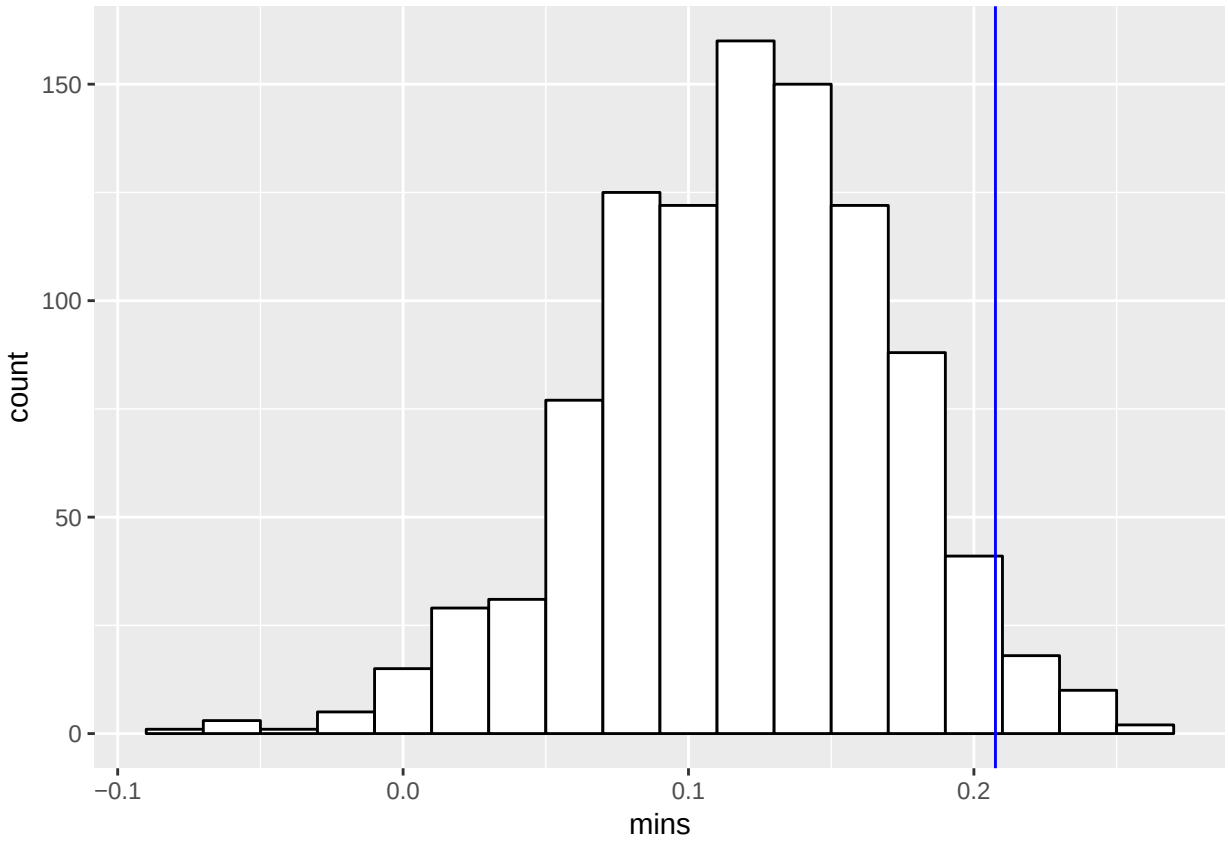
```
## # A tibble: 1,000 x 10
##    `Austin Barnes` `Chase Utley` `Chris Taylor` `Cody Bellinger`
##              <dbl>         <dbl>          <dbl>            <dbl>
##  1           0.343         0.219          0.136           0.335
##  2           0.350         0.308          0.182           0.333
##  3           0.482         0.164          0.277           0.332
##  4           0.370         0.389          0.309           0.231
##  5           0.126         0.253          0.319           0.371
##  6           0.154         0.247          0.202           0.297
##  7           0.345         0.275          0.316           0.184
##  8           0.381         0.301          0.291           0.372
##  9           0.207         0.307          0.315           0.254
## 10           0.313         0.145          0.273           0.0783
```

```
## # ... with 990 more rows, and 6 more variables: `Corey Seager` <dbl>,
## #   `Enrique Hernandez` <dbl>, `Joc Pederson` <dbl>, `Matt Kemp` <dbl>,
## #   `Yasiel Puig` <dbl>, `Yasmani Grandal` <dbl>
```

```r
maxes <- apply(theta_mle.df[, 1:10],1,max)
mins <- apply(theta_mle.df[,1:10], 1, min)

ggplot(data.frame(maxes), aes(x = maxes)) + geom_histogram(binwidth = 0.02, fill = 'white', color = 'bla
```




```r
ggplot(data.frame(mins), aes(x = mins)) + geom_histogram(binwidth = 0.02, fill = 'white', color = 'black
```



We would hope the observed data lines would fall in the mode of the barplot; however, this is not the case. The observed line was closer to the mode for the maximum values and much farther for the minimum values. Overall we will conclude, at least for this metric, the data generating model is not very consistent especially for the minimum values.

(c)

As $\tau \to \infty$ the posterior mean estimate will approach the no pooling estimator and as $\tau \to 0$ the posterior mean estimate will approach the complete pooling model. This is obvious since if the prior variance is almost 0 then when we generate a θ_i from $N(\mu, \tau^2)$, the generated values for all of the i will always be extremely close to μ. Also, if the prior variance is very large, then when we generate θ_i it will greatly deviate from μ for each i.

(d)

```
set.seed(24)
gibbs_update_thetas <- function(mu, tau, y, n){
  theta.mle <- y / n
  data.sd <- sqrt(theta.mle*(1-theta.mle)/n)

  post_var <- (tau^-2 + n*data.sd^-2)^-1
  post_mean <- (mu * tau^-2 + n*data.sd^-2 * theta.mle) * post_var
  post_sd <- sqrt(post_var)

  return(rnorm(1, post_mean, post_sd))
```

```
}


gibbs_update_mu <- function(mu, mu_sd, tau, thetas){
  post_var <- (mu_sd^-2 + length(thetas) * tau^-2)^-1
  post_mean <- (mu*mu_sd^-2 + sum(thetas)*tau^-2)*post_var
  post_sd <- sqrt(post_var)

  return(rnorm(1, post_mean, post_sd))
}

samples <- matrix(0, ncol = 11, nrow = 10000)

mu_0 <- theta_pool
sd_0 <- sqrt(theta_pool*(1-theta_pool)/sum(n))
theta_samples <- rep(mu_0, 10)

samples[1,1:10] <- theta_samples
samples[1,11] <- mu_0

for(i in 2:nrow(samples)){
  for(j in 1:10){
    theta_samples[j] <- gibbs_update_thetas(samples[i-1, 11], 0.05, y[j], n[j])
  }

  samples[i, 1:10] <- theta_samples

  samples[i, 11] <- gibbs_update_mu(samples[i-1,11], sd_0, 0.05, theta_samples)

  #sd_0 <- sqrt(samples[i,11] * (1-samples[i,11] / sum(n)))
}


mu <- as.mcmc(samples[,11])
thetas <- as.mcmc(samples[,1:10])

theta_pms <- c()
for(k in 1:10){
  theta_pms <- c(theta_pms, mean(thetas[,k]))
}


sqrt(sum((theta_pms - val)^2) /J)
```

```
## [1] 0.02467066
```

The RMSE for partial pooling gave the best RMSE at 0.02467066.

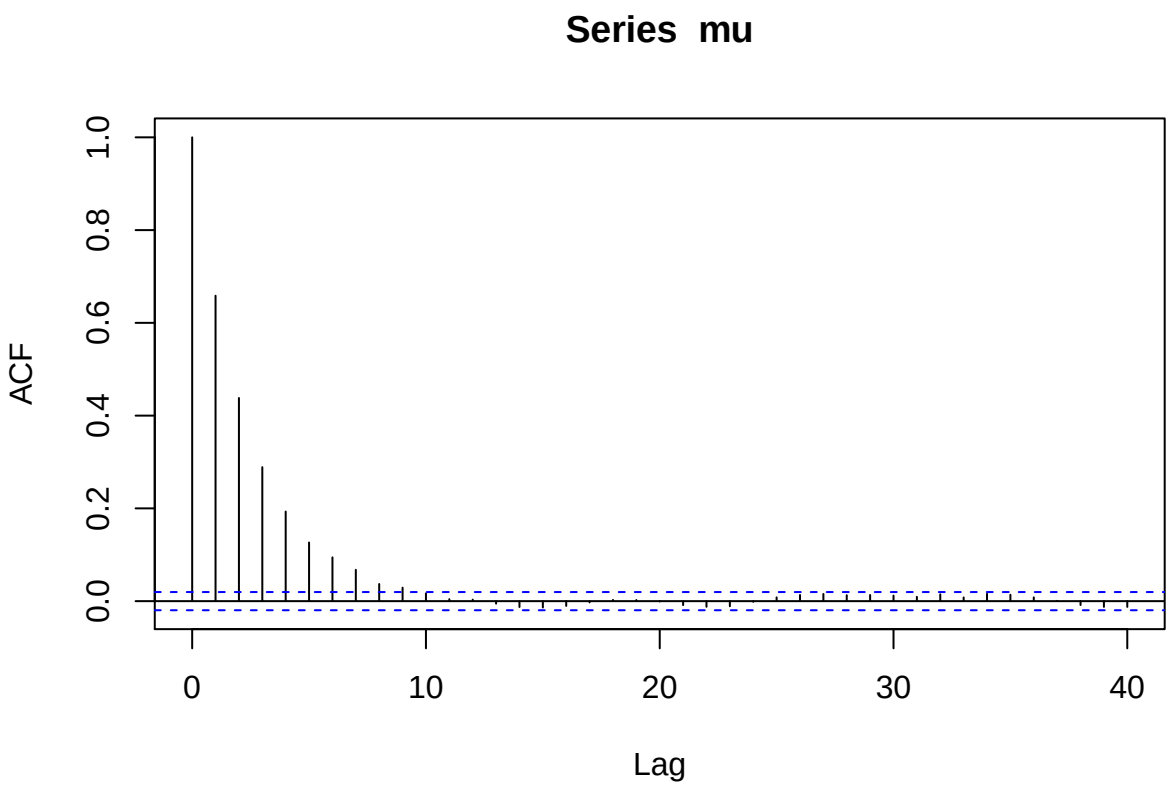
(e)

```
effectiveSize(mu)
```

```
##     var1
```

```
## 2058.691
```

```
acf(mu)
```




```
#tau = 0.001
samples <- matrix(0, ncol = 11, nrow = 10000)

mu_0 <- theta_pool
sd_0 <- sqrt(theta_pool*(1-theta_pool)/sum(n))
theta_samples <- rep(mu_0, 10)

samples[1,1:10] <- theta_samples
samples[1,11] <- mu_0

for(i in 2:nrow(samples)){
  for(j in 1:10){
    theta_samples[j] <- gibbs_update_thetas(samples[i-1, 11], 0.001, y[j], n[j])
  }

  samples[i, 1:10] <- theta_samples

  samples[i, 11] <- gibbs_update_mu(samples[i-1,11], sd_0, 0.001, theta_samples)

  #sd_0 <- sqrt(samples[i,11] * (1-samples[i,11] / sum(n)))
}


mu <- as.mcmc(samples[,11])
thetas <- as.mcmc(samples[,1:10])
```

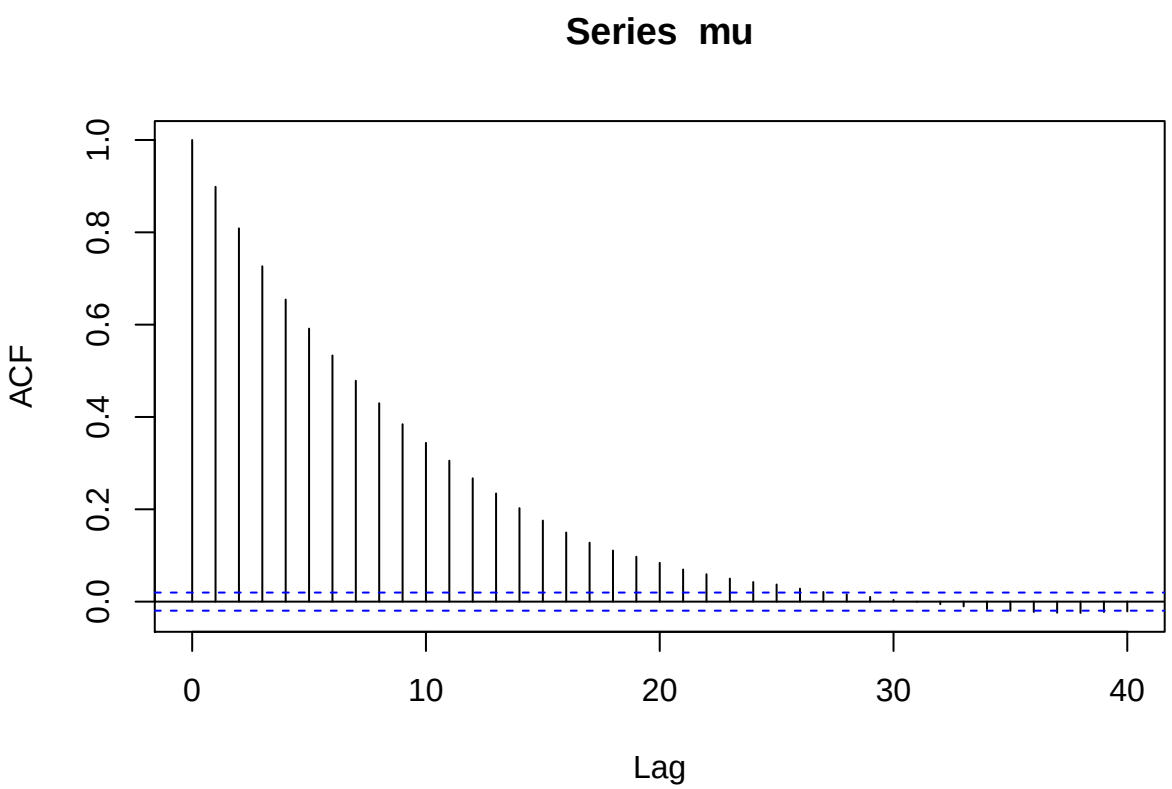
```r
theta_pms <- c()
for(k in 1:10){
  theta_pms <- c(theta_pms, mean(thetas[,k]))
}


sqrt(sum((theta_pms - val)^2) /J)
```

```
## [1] 0.02560074
```

```r
effectiveSize(mu)
```

```
##     var1
## 533.9897
```

```r
acf(mu)
```



We observe for smaller τ we have smaller effective size and higher acf.