# Lab 5

*Senan Hogan-H.*

*24 April 2018*

Completed with James Kinney, around 50/50 work completed between us.

## 2 Normal Data

## 1b

A Gibbs sampler is a good option for this graph because there are various ‘levels’ which only rely on one other level. For example . $\{f(\bar{x}_1, \ldots, f(\bar{x}_n)\}$ only depend on $\{P(\theta_1, \ldots, P(\theta_n))\}$, which only depend on $\{P_n(\mu), P_n(\tau)\}$. Thus, if we know $\theta_i$ then we don’t care about any other values when we sample $y_i$. These clean dependency levels make a Gibbs Sampler appetizing.

## 1c

Run the code from lecture 17

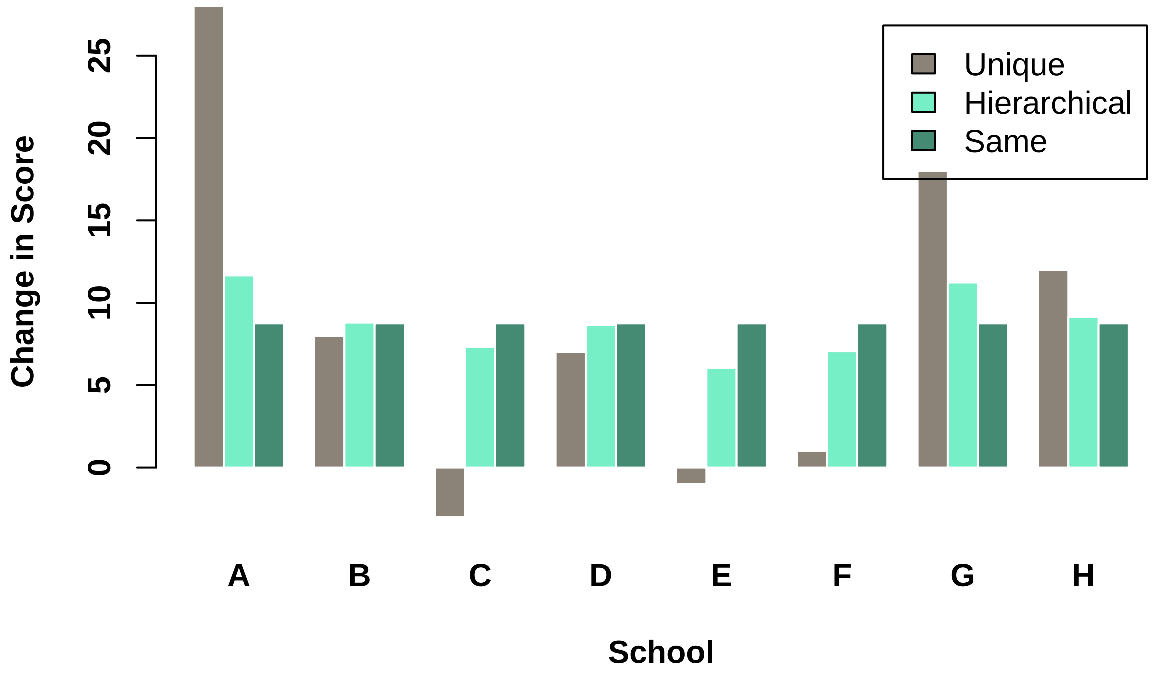
```
schools <- read.table("schools.dat", header=T)
J <- nrow (schools)
y <- schools$estimate
sigma.y <- schools$sd
data <- list ("J", "y", "sigma.y")
inits <- function()
  list (theta=rnorm(J,0,100), mu.theta=rnorm(1,0,100), sigma.theta=runif(1,0,100))
parameters <- c("theta", "mu.theta", "sigma.theta")
theta.update <- function (){
  theta.hat <- (mu/tau^2 + y/sigma.y^2)/(1/tau^2 + 1/sigma.y^2)
  V.theta <- 1/(1/tau^2 + 1/sigma.y^2)
  rnorm(J, theta.hat, sqrt(V.theta))
}
mu.update <- function (){
  rnorm(1, mean(theta), tau/sqrt(J))
}
tau.update <- function (){
  sqrt(sum((theta-mu)^2)/rchisq(1,J-1))
}
n.chains <- 5
n.iter <- 1000
sims <- array (NA, c(n.iter, n.chains, J+2))
dimnames(sims) <- list (NULL, NULL, c(paste("theta[", 1:8, "]", sep=""), "mu", "tau"))
for (m in 1:n.chains){
  mu <- rnorm (1, mean(y), sd(y))
  tau <- runif (1, 0, sd(y))
  for (t in 1:n.iter){
    theta <- theta.update()
    mu <- mu.update()
    tau <- tau.update()
    sims[t,m,] <- c(theta, mu, tau)
  } }
```

Now calculate the estimates by taking the mean across the chains

```
# Mean increase across all schools
meanBayes = mean(mean(sims[,1,9]), mean(sims[,2,9]), mean(sims[,3,9]), mean(sims[,4,9]), mean(sims[,5,9
thetaFinals = c('theta1', 'theta2', 'theta3', 'theta4', 'theta5', 'theta6', 'theta7', 'theta9')
thetasBayes=c()
for(i in 1:8){
  thetasBayes[i] = mean(mean(sims[,1,i]), mean(sims[,2,i]), mean(sims[,3,i]), mean(sims[,4,i]), mean(si
}
```

Plot the difference between the estimates we would get from the hierarchical Bayesian approach versus the frequentists approaches where we would either assume each school is unique or each school is a draw from the same distribution.

```
thetasUnique = schools$estimate
thetasSame = rep(mean(schools$estimate), 8)

data=matrix(c(thetasUnique, thetasBayes, thetasSame), byrow= TRUE, nrow=3)
rownames(data)=c('Unique','Hierarchical', 'Same')
colnames(data)=schools$school

# Grouped barplot
barplot(data, col=colors()[c(7,10,12)] , border="white", font.axis=2, beside=T, legend=rownames(data),
```



We see that the hierachical model estimates are between the two frequentist estimators. This is a more realistic guess because we assume that a huge change in score is partly do to a lucky or unlucky set of students. It may be that the students learned a lot from the test prep, but it is likely also due to something like the fact that they ate breakfast that morning, or another confounding variable. Thus modeling the school estimates as unique draws from the same distribution as we do with the hierachical model is a good idea.

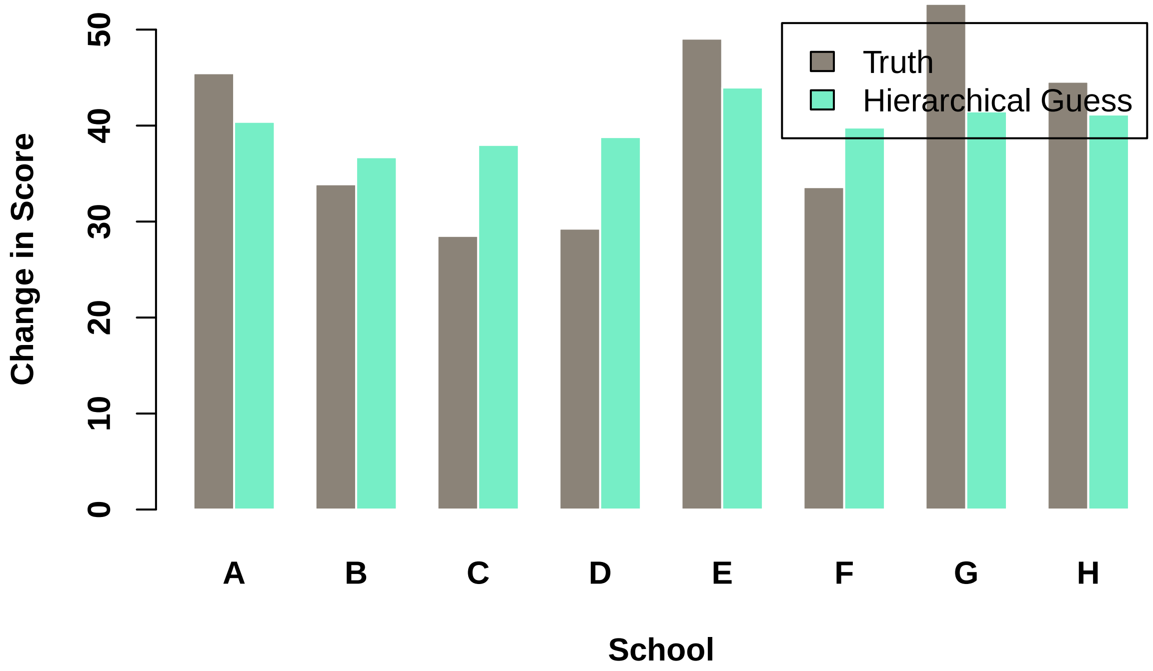
## 1d

Now we want to simulate from the model by choosing $\mu$ and $\tau$ and seeing how well we estimate the thetas. Let $\mu = 40$ $\tau = 10$ and draw the 8 $\theta_i$ values. Suppose there are 14 participants at each school and we know the variance of typical scores is $15^2$.

```
mu = 40
tau=10
trueThetas = rnorm(8, mu, tau)
estimates = c()
sds = c()
for(i in 1:8){
  schoolSample = rnorm(14, trueThetas[i], 15)
  estimates[i] = mean(schoolSample)
  sds[i] = sd(schoolSample)
}
school = schools$school
sampleData = data.frame(schools$school, estimates, sds)

J <- nrow(sampleData)
y <- sampleData$estimate
sigma.y <- sampleData$sd
data <- list ("J", "y", "sigma.y")
inits <- function()
  list (theta=rnorm(J,0,100), mu.theta=rnorm(1,0,100), sigma.theta=runif(1,0,100))
parameters <- c("theta", "mu.theta", "sigma.theta")
theta.update <- function (){
  theta.hat <- (mu/tau^2 + y/sigma.y^2)/(1/tau^2 + 1/sigma.y^2)
  V.theta <- 1/(1/tau^2 + 1/sigma.y^2)
  rnorm(J, theta.hat, sqrt(V.theta))
}
mu.update <- function (){
  rnorm(1, mean(theta), tau/sqrt(J))
}
tau.update <- function (){
  sqrt(sum((theta-mu)^2)/rchisq(1,J-1))
}
n.chains <- 5
n.iter <- 1000
sims <- array (NA, c(n.iter, n.chains, J+2))
dimnames(sims) <- list (NULL, NULL, c(paste("theta[", 1:8, "]", sep=""), "mu", "tau"))
for (m in 1:n.chains){
  mu <- rnorm (1, mean(y), sd(y))
  tau <- runif (1, 0, sd(y))
  for (t in 1:n.iter){
    theta <- theta.update()
    mu <- mu.update()
    tau <- tau.update()
    sims[t,m,] <- c(theta, mu, tau)
  }
}
```

Now see how different our estimates are form the real values of $\theta_i$.

```
# Mean increase across all schools
meanBayes = mean(mean(sims[,1,9]), mean(sims[,2,9]), mean(sims[,3,9]), mean(sims[,4,9]), mean(sims[,5,9
thetaFinals = c('theta1', 'theta2', 'theta3', 'theta4', 'theta5', 'theta6', 'theta7', 'theta9')
thetasBayes=c()
for(i in 1:8){
  thetasBayes[i] = mean(mean(sims[,1,i]), mean(sims[,2,i]), mean(sims[,3,i]), mean(sims[,4,i]), mean(si
}

data=matrix(c(trueThetas, thetasBayes), byrow= TRUE, nrow=2)
rownames(data)=c('Truth','Hierarchical Guess')
colnames(data)=sampleData$school

# Grouped barplot
barplot(data, col=colors()[c(7,10)] , border="white", font.axis=2, beside=T, legend=rownames(data), arg
```



We see that this model does ok, but does not capture enough of the extremes.

## 3 Binomial Data

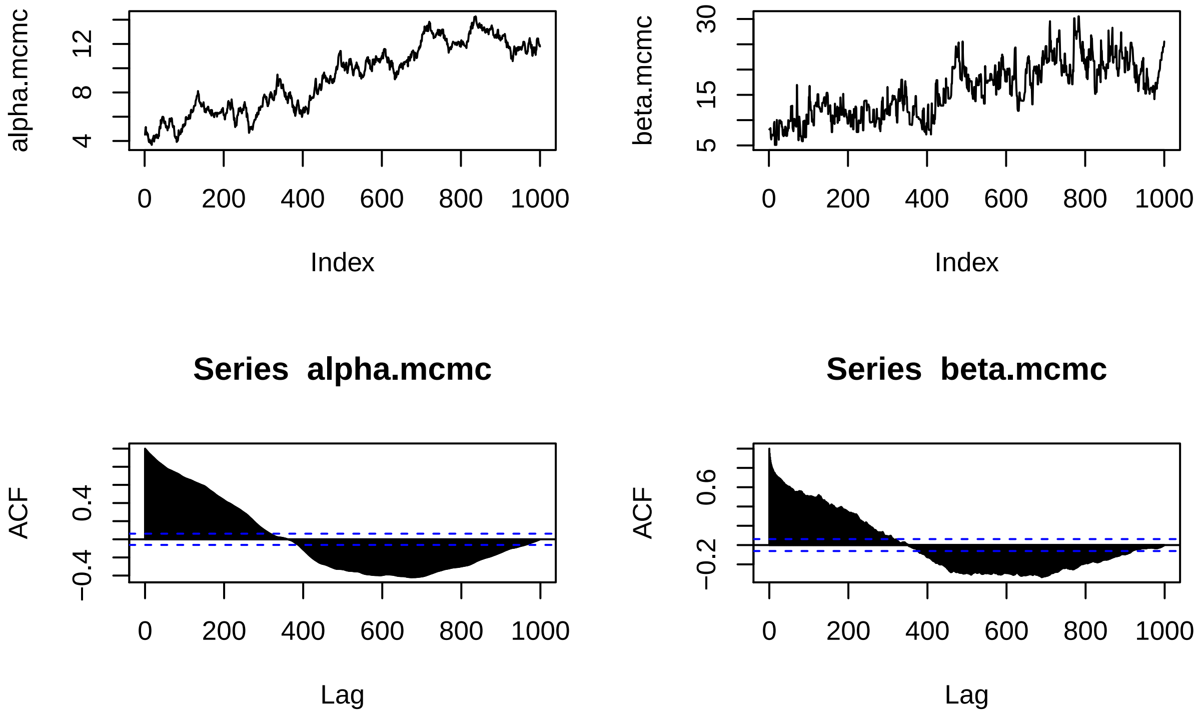
e. We start with a known distribution Beta$(\alpha, \beta)$, where $\alpha = \frac{\beta}{2} = 5$. First step is to sample from this know distrution to get $p_i$ for $i = 1, \ldots, k$. And consider $k = 10$, for a relatively small data set.

```r
alpha_known <- 5
beta_known <- 10
k <- 10
id <- c(1:k)

N <- round(rnorm(k, mean = 30, sd = 3)) #random sample sizes, not important in decision
p <- rbeta(k, alpha_known, beta_known)
y <- round(p*N) #standardised to make possible to be observed

data <- as.data.frame(cbind(id, p, y, N), row.names = NULL)
```

The next step is to draw from an MC chain in order to estimate $\alpha$ and $\beta$. Note that this code is taken from the given sources, and so is not my own work, instead minorly edited for this data set.

See here for the original source: http://www.stat.cmu.edu/~brian/724/week06/lec15.r with the corresponding lecture notes http://www.stat.cmu.edu/~brian/724/week06/lec15-mcmc2.pdf

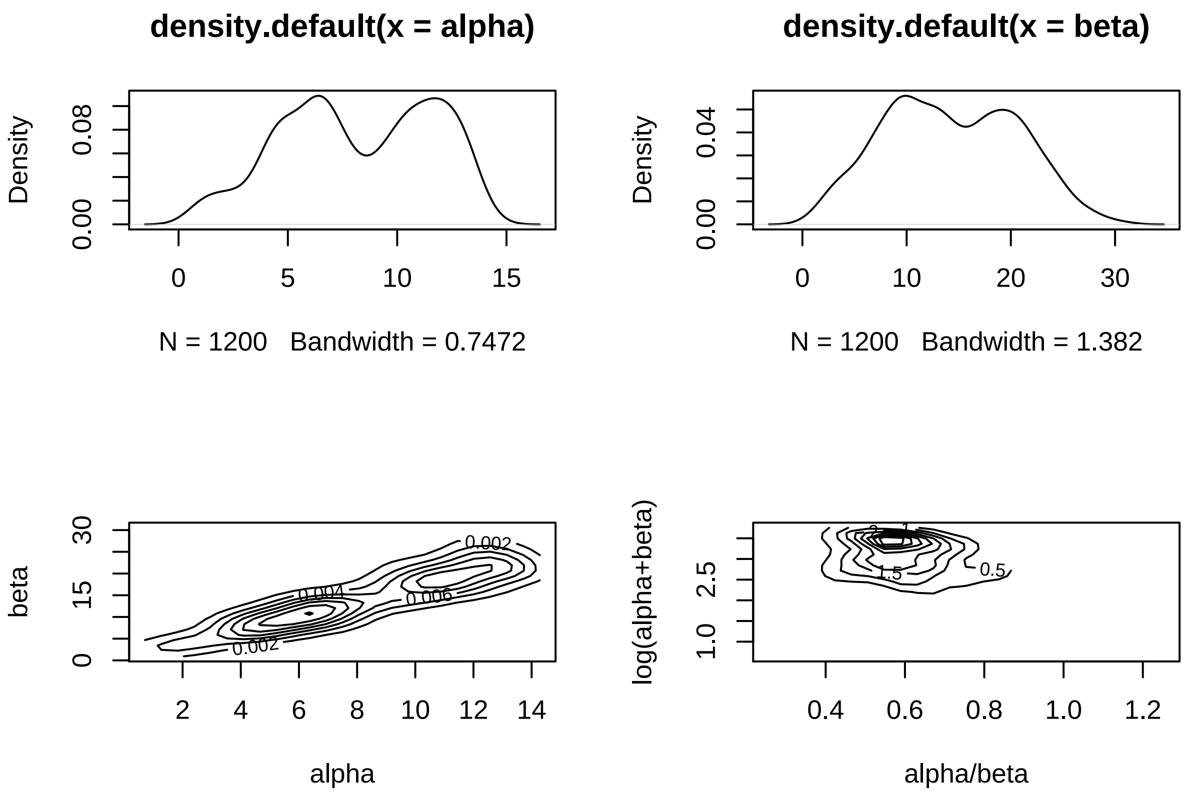
```r
y <- data$y
n <- data$N
J <- length(y)

log.prior <- function(alpha,beta) {
  {-2.5}*log(alpha + beta)
}

draw.thetas <- function(alpha,beta) {
  return(rbeta(J,alpha+y,beta+n-y))
}

draw.alpha <- function(alpha,beta,theta,prop.sd) {
  alpha.star <- rnorm(1,alpha,prop.sd)
  num <- J*(lgamma(alpha.star+beta) - lgamma(alpha.star)) +
    alpha.star*sum(log(theta)) + log.prior(alpha.star,beta)
  den <- J*(lgamma(alpha+beta)      - lgamma(alpha)) +
    alpha     *sum(log(theta)) + log.prior(alpha,beta)
# print(c(alpha,alpha.star,num,den))
  acc <- ifelse((log(runif(1))<=num - den)&&(alpha.star>0),1,0)
  alpha.acc <<- alpha.acc + acc
  return(ifelse(acc,alpha.star,alpha))
}

draw.beta <- function(alpha,beta,theta,prop.sd) {
  beta.star <- rnorm(1,beta,prop.sd)
  num <- J*(lgamma(alpha+beta.star) - lgamma(beta.star)) +
    beta.star*sum(log(1-theta)) + log.prior(alpha,beta.star)
  den <- J*(lgamma(alpha+beta)      - lgamma(beta)) +
    beta     *sum(log(1-theta)) + log.prior(alpha,beta)
# print(c(beta,beta.star,num,den))
  acc <- ifelse((log(runif(1))<=num - den)&&(beta.star>0),1,0)
  beta.acc <<- beta.acc + acc
  return(ifelse(acc,beta.star,beta))
}

################################################################

B <- 200 # define cut off
M <- 1000 # length of chain

MM <- B + M

alpha <- matrix(NA,MM)
beta <- alpha
theta <- matrix(NA,nrow=MM,ncol=J)

# Metropolis tuning parameters
alpha.prop.sd <- 0.25
beta.prop.sd <- 3

# Initial values for the chain
alpha[1] <- 1
beta[1] <- 1
theta[1,] <- draw.thetas(alpha[1],beta[1]) # or theta[1,] <- (y+.5/(n+.5)

# Monitor acceptance frequency
alpha.acc <- 0
beta.acc <- 0

# MCMC simulation
for (m in 2:MM) {
  alpha[m] <- draw.alpha(alpha[m-1],beta[m-1],theta[m-1,],alpha.prop.sd)
  beta[m] <- draw.beta(alpha[m],beta[m-1],theta[m-1,],beta.prop.sd)
  theta[m,] <- draw.thetas(alpha[m],beta[m])
}

good <- (B+1):MM

alpha.mcmc <- alpha[good]
beta.mcmc <- beta[good]
theta.mcmc <- theta[good,]


par(mfrow=c(2,2))
plot(alpha.mcmc,type="l")
plot(beta.mcmc,type="l")
acf(alpha.mcmc,1000)
acf(beta.mcmc,1000)
```



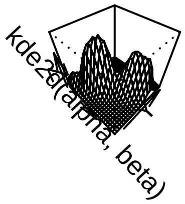
```r
print(round(c(alpha.rate=alpha.acc/MM,beta.rate=beta.acc/MM),2))
```

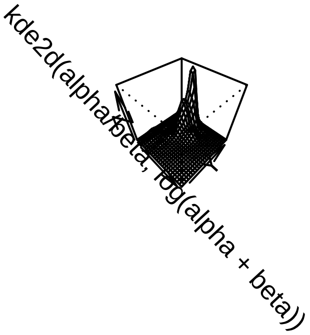
```
## alpha.rate  beta.rate
##       0.92       0.56
```

```r
######################################################################

#par(mfrow=c(3,2))

plot(density(alpha))
plot(density(beta))

contour(kde2d(alpha,beta),xlab="alpha",ylab="beta")
contour(kde2d(alpha/beta,log(alpha+beta)),
        xlab="alpha/beta",ylab="log(alpha+beta)")
```



```r
persp(kde2d(alpha,beta),theta=45,phi=45)
persp(kde2d(alpha/beta,log(alpha+beta)),theta=45,phi=45)
```





i. This part involves using my own code from here on. The following compares the estimator to MSE to the two frequentist extremes. Frequentists would generally assume that all draws are the same, so take the mean of the posterior, or consider every draw different so use a draw as an estimator of itself.

First suppose that all draws are the same, so $p_i$ =mean(Beta($\alpha, \beta$)).

```
data$p_bar <- qbeta(0.5, mean(alpha.mcmc), mean(beta.mcmc))
data$p_bar_mse <- (data$p_bar - data$p)^2

# MSE for this frequentist estimator
mean(data$p_bar_mse)
```

```
## [1] 0.00797249
```

Second suppose that all draws are different, so $p_i = y_i \sim Beta(\alpha, \beta)$.

```
data$p_hat <- rbeta(k, mean(alpha.mcmc), mean(beta.mcmc))
data$p_hat_mse <- (data$p_hat - data$p)^2

# MSE for this frequentist estimator
mean(data$p_hat_mse)
```

```
## [1] 0.01347999
```

Next, move away from frequentist methods to a Bayesian Gibbs sampler.

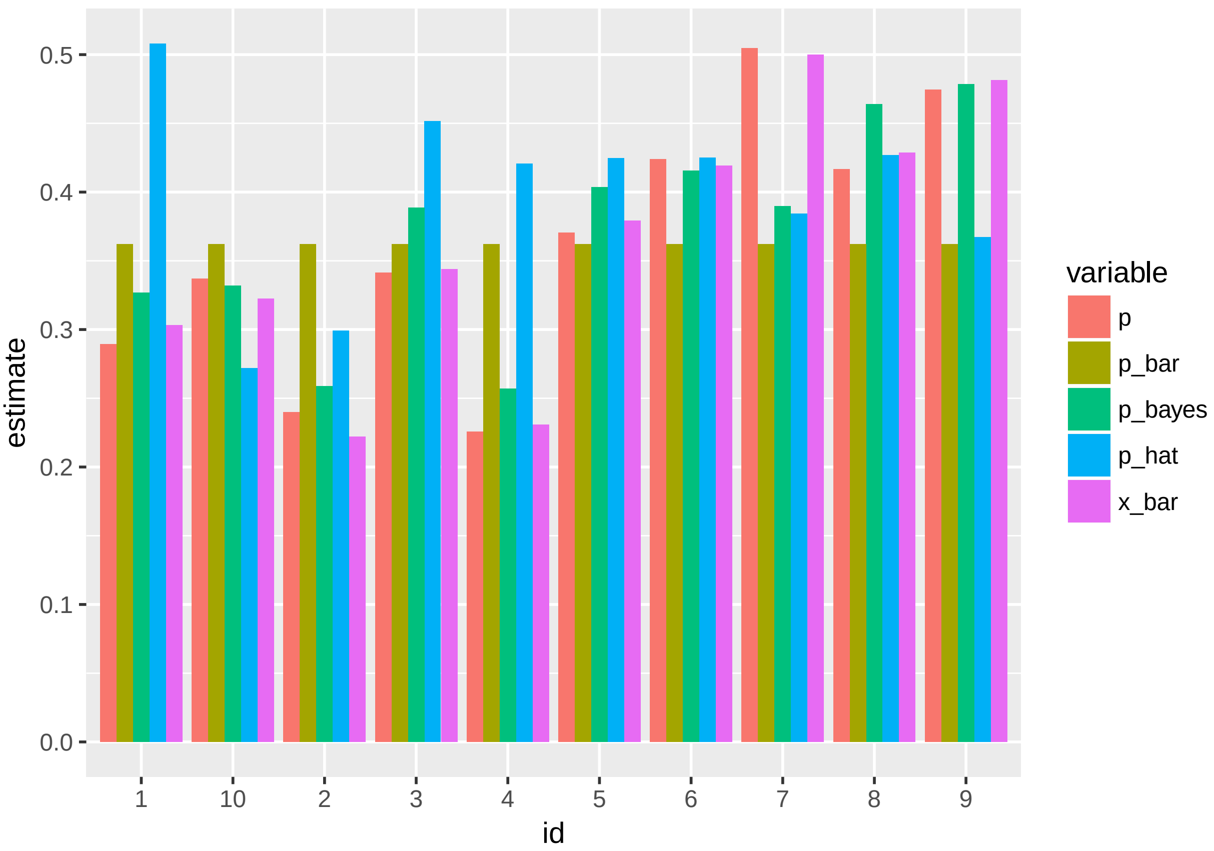
```
n.repeats <- 1000

alpha_sampler.matrix <- matrix(data = NA, nrow = n.repeats, ncol = k)
beta_sampler.matrix <- matrix(data = NA, nrow = n.repeats, ncol = k)
```

```r
p_sampler.matrix <- matrix(data = NA, nrow = n.repeats, ncol = k)

alpha_sampler.matrix[1,] <- mean(alpha.mcmc) # initialise sampler
beta_sampler.matrix[1,] <- mean(beta.mcmc)

for (i in c(1:k)){
  n <- data$N[i]
  x <- data$y[i]
  alpha_sampler.matrix[1,i] <- mean(alpha.mcmc) + x
  beta_sampler.matrix[1,i] <- mean(beta.mcmc) + n - x

  p_sampler.matrix[1,i] <- rbeta(1, alpha_sampler.matrix[1,i],
                                 beta_sampler.matrix[1,i])
  for (j in c(2:n.repeats)){
    x <- rbinom(1, size = n, prob = p_sampler.matrix[(j-1),i])

    alpha_sampler.matrix[j,i] <- mean(alpha_sampler.matrix[,i][1:(j-1)]) + x
    beta_sampler.matrix[j,i] <- mean(beta_sampler.matrix[,i][1:(j-1)]) + n - x

    p_sampler.matrix[j,i] <- rbeta(1, alpha_sampler.matrix[j,i],
                                 beta_sampler.matrix[j,i])
  }
}

data_p_bayes <- NA
for (i in c(1:k)){
  data$p_bayes[i] <- mean(p_sampler.matrix[,i][round(n.repeats)/5 : n.repeats])
}

data$p_bayes_mse <- (data$p_bayes - data$p)^2

# MSE for this Bayesian estimator
mean(data$p_bayes_mse)
```

```
## [1] 0.002165937
```

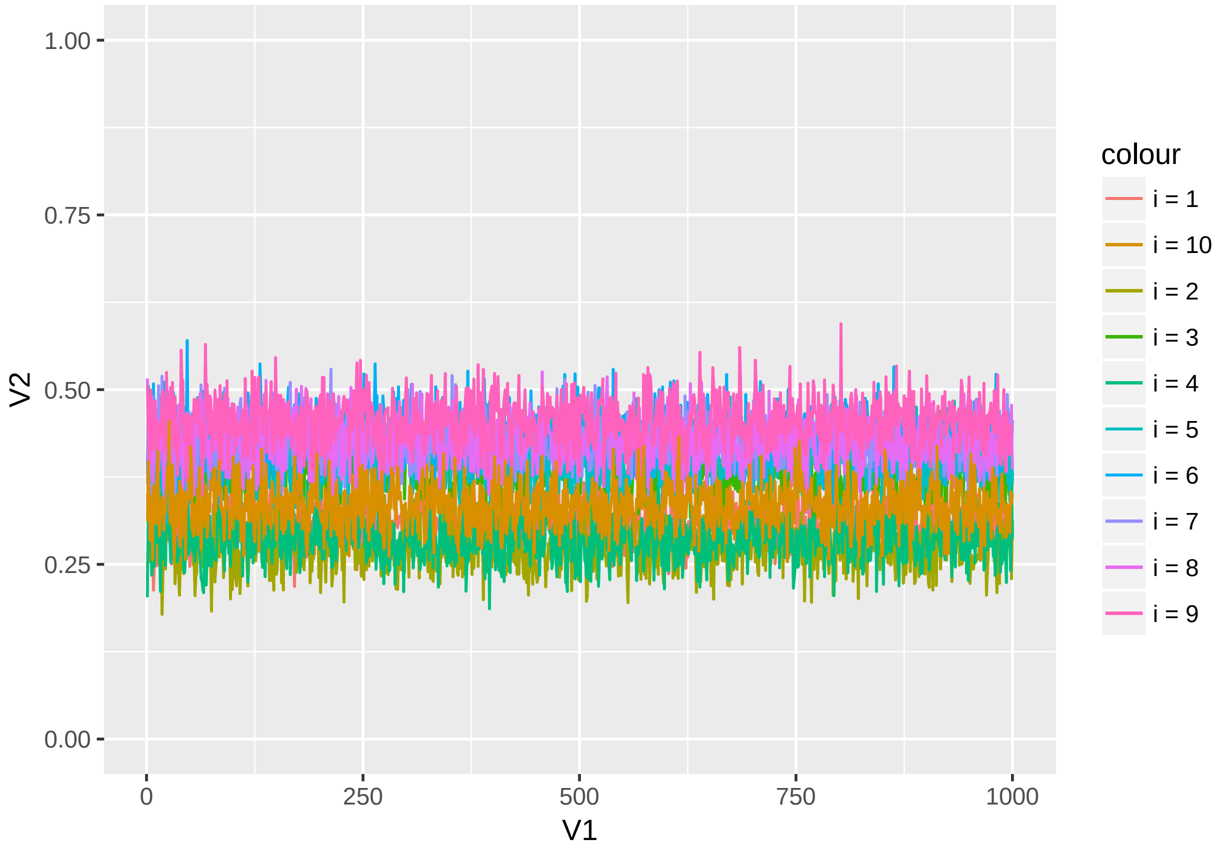
Here follows a cool plot showing the truth (p), the data (x_bar), and the estimates (p_bar, p_bayes, p_hat). It is clear that the Bayesian estimator is the most consistent and best estimator across the sample.

```r
data$id <- as.character(data$id)
data$x_bar <- data$y/data$N

data %>%
  gather(variable, estimate,
         c('p', 'x_bar', 'p_bar', 'p_hat', 'p_bayes' )) %>%
  ggplot(aes(x = id, y = estimate)) +
  geom_bar(aes(fill = variable), position = "dodge", stat="identity")
```



ii. The chains all converge in the space, though exhibit some variability, as is to be expected.

```
as.data.frame(cbind(
  c(1:n.repeats),
  p_sampler.matrix[,1],
  p_sampler.matrix[,2],
  p_sampler.matrix[,3],
  p_sampler.matrix[,4],
  p_sampler.matrix[,5],
  p_sampler.matrix[,6],
  p_sampler.matrix[,7],
  p_sampler.matrix[,8],
  p_sampler.matrix[,9],
  p_sampler.matrix[,10]),
  row.names = NULL) %>%
  ggplot(aes(x = V1)) +
  geom_line(aes(y = V2, colour = 'i = 1')) +
  geom_line(aes(y = V3, colour = 'i = 2')) +
  geom_line(aes(y = V4, colour = 'i = 3')) +
  geom_line(aes(y = V5, colour = 'i = 4')) +
  geom_line(aes(y = V6, colour = 'i = 5')) +
  geom_line(aes(y = V7, colour = 'i = 6')) +
  geom_line(aes(y = V8, colour = 'i = 7')) +
  geom_line(aes(y = V9, colour = 'i = 8')) +
  geom_line(aes(y = V10, colour = 'i = 9')) +
  geom_line(aes(y = V11, colour = 'i = 10')) +
  ylim(0,1)
```



iii. My smaller data set suggests that $\alpha = 78.66$ and $\beta = 224.87$ are not reasonable estimates.

```
quantile(alpha.mcmc, prob = c(0.025, 0.975))
```

```
##      2.5%     97.5%
##  4.303003 13.491259
```

```
quantile(beta.mcmc, prob = c(0.025, 0.975))
```

```
##      2.5%     97.5%
##  6.978784 26.088342
```

Let's show how the same approach estimates $\alpha_0$ and $\beta_0$ in my smaller data set.

```
library(stats4)
library(VGAM)
```

```
## Loading required package: splines

##
## Attaching package: 'VGAM'

## The following object is masked from 'package:tidyr':
##
##     fill
```

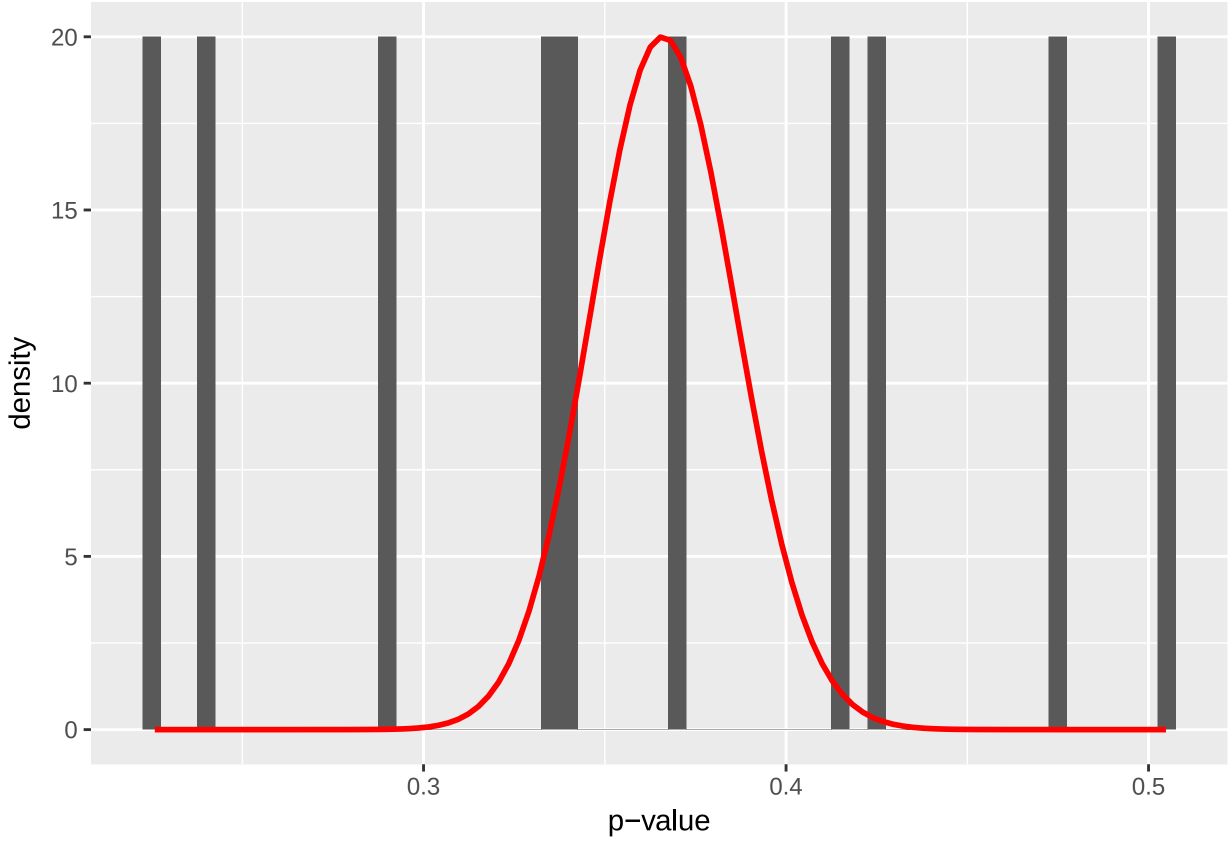
```
# log-likelihood function
ll <- function(alpha, beta) {
  x <- data$y
  total <- data$N
  -sum(VGAM::dbetabinom.ab(x, total, alpha, beta, log = TRUE))
}

# maximum likelihood estimation
m <- mle(ll, start = list(alpha = 1, beta = 5), method = "L-BFGS-B",
                 lower = c(0.0001, .1))
ab <- coef(m)

alpha0 <- ab[1]
beta0 <- ab[2]

alpha0
```

```
##    alpha
## 214.0107
```

```
beta0
```

```
##    beta
## 369.955
```

```
data %>% ggplot() +
  geom_histogram(aes(p, y = ..density..), binwidth = .005) +
  stat_function(fun = function(x) dbeta(x, alpha0, beta0),
                colour = "red", size = 1) +
  xlab("p-value")
```



Clearly the same approach produces different estimates for $\alpha$ and $\beta$ in the smaller data set, more similar to the blog post's estimates, though the approach seems to be less appropriate in such a small sample size, as seen in the graph of the function.