EslamiShafigh_Ehsan_rlab05

Ehsan Eslmai Shafigh

2023-05-26

Assignment 5

Exercise 1

We calculate a different posterior for each of the two sets of observations.

(a.1) Assuming a uniform prior:

```
sigma_y = 65 * 1 + 22 * 2 + 3 * 3 +  1 * 4
n = 109 + 65 + 22 + 3 + 1
n.sample = 1000
delta.lambda = 1/n.sample

alpha = 1 + sigma_y
mu = n
lambda = seq(from = 0 , by = delta.lambda , length.out = n.sample )

lambda_posterior = dgamma(lambda , alpha , mu)


mean = delta.lambda * sum(lambda * lambda_posterior)
variance = delta.lambda * sum(lambda ^ 2 * lambda_posterior) - mean ^ 2
median = qgamma(0.5 , alpha , mu )
ci = qgamma(c(0.025 , 0.975) , alpha , mu)

print(mean)
```

```
## [1] 0.615
```

```
print(variance)
```

```
## [1] 0.003075
```

```
print(median)
```

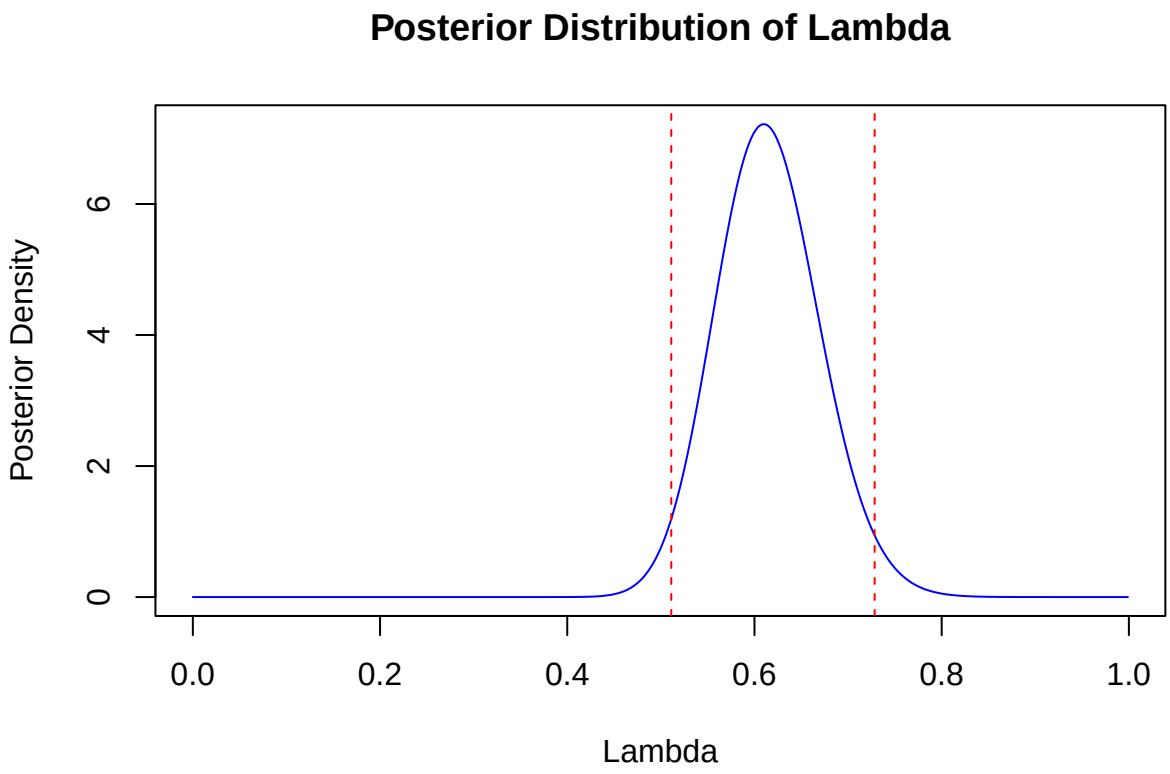
```
## [1] 0.6133341
```

```r
print(ci)
```

```
## [1] 0.5111251 0.7283408
```

```r
# Plot the lambda and lambda_posterior
plot(lambda, lambda_posterior, type = 'l', col = 'blue', xlab = "Lambda", ylab = "Posterior Density", ma

# Add vertical dashed lines for the confidence interval
abline(v = ci[1], lty = 'dashed', col = 'red')
abline(v = ci[2], lty = 'dashed', col = 'red')
```



(b.1) We repeat the same task for Jeffery's prior:

```r
sigma_y = 65 * 1 + 22 * 2 + 3 * 3 +  1 * 4
n = 109 + 65 + 22 + 3 + 1
n.sample = 1000
delta.lambda = 1/n.sample

alpha = 0.5 + sigma_y
mu = n
lambda = seq(from = 0 , by = delta.lambda , length.out = n.sample )

lambda_posterior = dgamma(lambda , alpha , mu)
```

```r
mean = delta.lambda * sum(lambda * lambda_posterior)
variance = delta.lambda * sum(lambda ^ 2 * lambda_posterior) - mean ^ 2
median = qgamma(0.5 , alpha , mu )
ci = qgamma(c(0.025 , 0.975) , alpha , mu)

print(mean)
```

```
## [1] 0.6125
```

```r
print(variance)
```

```
## [1] 0.0030625
```

```r
print(median)
```

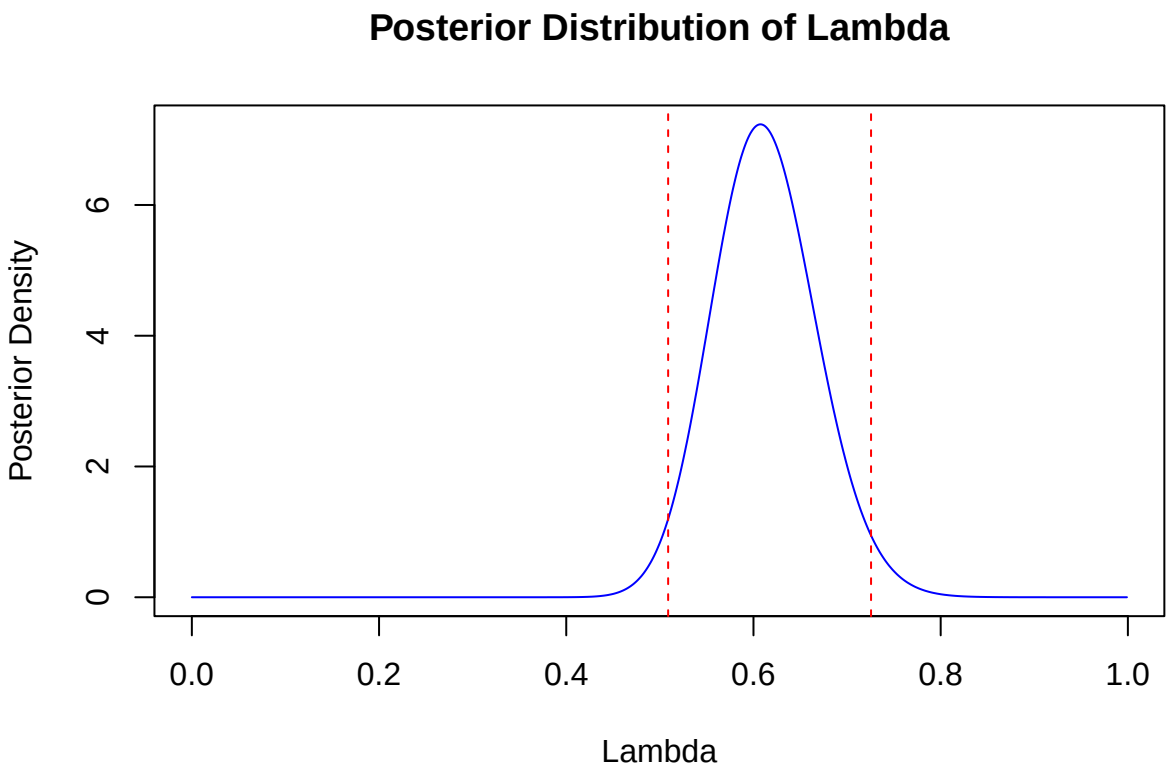
```
## [1] 0.6108341
```

```r
print(ci)
```

```
## [1] 0.5088464 0.7256196
```

```r
# Plot the lambda and lambda_posterior
plot(lambda, lambda_posterior, type = 'l', col = 'blue', xlab = "Lambda", ylab = "Posterior Density", ma

# Add vertical dashed lines for the confidence interval
abline(v = ci[1], lty = 'dashed', col = 'red')
abline(v = ci[2], lty = 'dashed', col = 'red')
```



(a.2) We can repeat the same for the second set of observations:

```
sigma_y = 91 * 1 + 32 * 2 + 11 * 3 + 2 * 4
n = 144 + 91 + 32 + 11 + 2
n.sample = 1000
delta.lambda = 1/n.sample

alpha = 1 + sigma_y
mu = n
lambda = seq(from = 0 , by = delta.lambda , length.out = n.sample )

lambda_posterior = dgamma(lambda , alpha , mu)


mean = delta.lambda * sum(lambda * lambda_posterior)
variance = delta.lambda * sum(lambda ^ 2 * lambda_posterior) - mean ^ 2
median = qgamma(0.5 , alpha , mu )
ci = qgamma(c(0.025 , 0.975) , alpha , mu)

print(mean)
```

```
## [1] 0.7035714
```

```r
print(variance)
```

```
## [1] 0.002512784
```

```r
print(median)
```

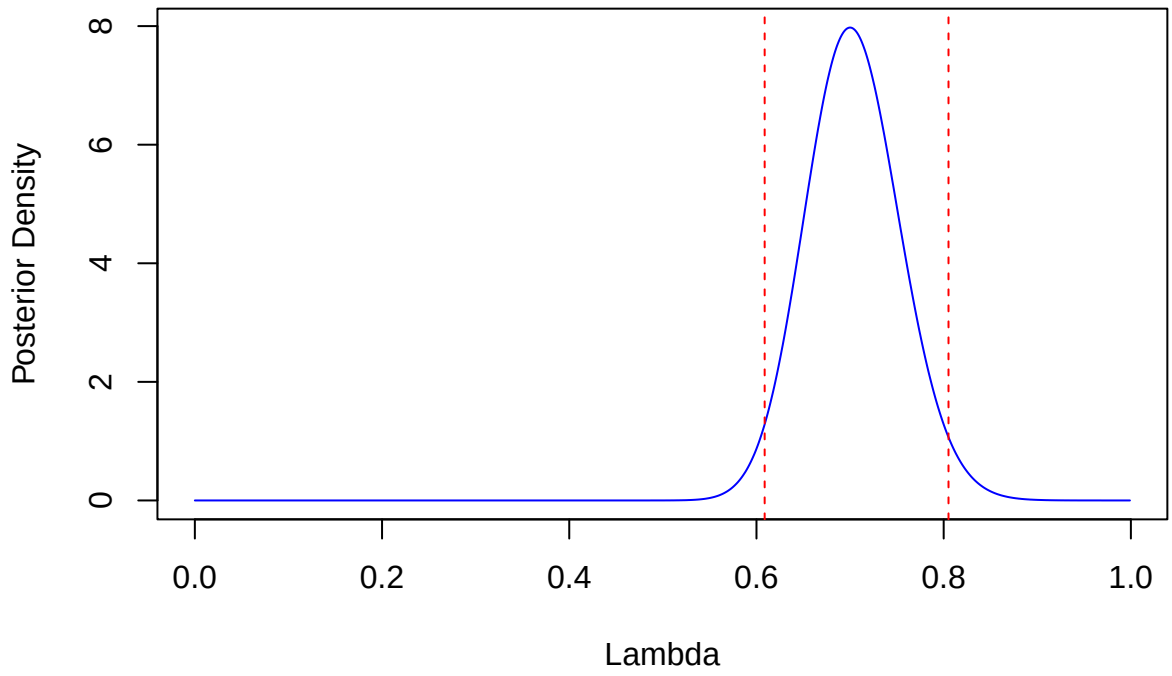
```
## [1] 0.7023813
```

```r
print(ci)
```

```
## [1] 0.6087487 0.8051570
```

```r
# Create the plot
plot(lambda, lambda_posterior, type = 'l', col = 'blue', xlab = "Lambda", ylab = "Posterior Density")

# Add vertical dashed lines for the confidence interval
abline(v = ci[1], lty = 'dashed', col = 'red')
abline(v = ci[2], lty = 'dashed', col = 'red')
```



```r
sigma_y = 91 * 1 + 32 * 2 + 11 * 3 + 2 * 4
n = 144 + 91 + 32 + 11 + 2
n.sample = 1000
delta.lambda = 1/n.sample

alpha = 0.5 + sigma_y
mu = n
lambda = seq(from = 0 , by = delta.lambda , length.out = n.sample )

lambda_posterior = dgamma(lambda , alpha , mu)


mean = delta.lambda * sum(lambda * lambda_posterior)
variance = delta.lambda * sum(lambda ^ 2 * lambda_posterior) - mean ^ 2
median = qgamma(0.5 , alpha , mu )
ci = qgamma(c(0.025 , 0.975) , alpha , mu)

print(mean)
```

(b.2)

```
## [1] 0.7017857
```

```r
print(variance)
```

```
## [1] 0.002506402
```

```r
print(median)
```

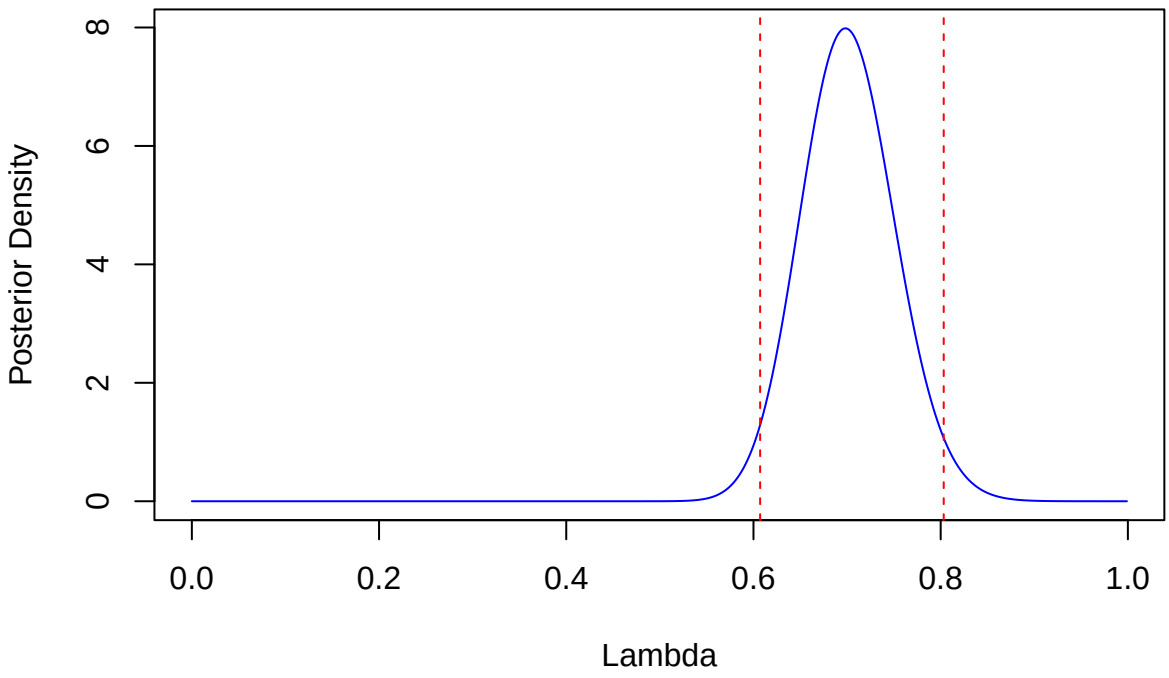
```
## [1] 0.7005956
```

```r
print(ci)
```

```
## [1] 0.6070878 0.8032465
```

```r
# Create the plot
plot(lambda, lambda_posterior, type = 'l', col = 'blue', xlab = "Lambda", ylab = "Posterior Density")

# Add vertical dashed lines for the confidence interval
abline(v = ci[1], lty = 'dashed', col = 'red')
abline(v = ci[2], lty = 'dashed', col = 'red')
```



Exercise 2

I use the code written for the MCMC algorithm in one of the slides:

```
# Parameters:
# func : a function whose first argument is a real vector of parameters
# func returns a log10 of the likelihood function
# theta.init : the initial value of the Markov Chain (and of func)
# n.sample: number of required samples
# sigma : standar deviation of the gaussian MCMC sampling pdf

metropolis.1dim <- function(func , theta.init , n.sample , sigma) {
theta.cur <- theta.init
func.Cur <- func(theta.cur)
func.Samp <- matrix(data=NA, nrow=n.sample , ncol=2+1)
n.accept <- 0
rate.accept <- 0.0

for (n in 1:n.sample) {
theta.prop <- rnorm(n=1, mean = theta.cur, sigma)
func.Prop <- func(theta.prop)
logMR <- func.Prop - func.Cur # Log10 of the Metropolis ratio
if ( logMR >=0 || logMR >log10(runif(1)) ) {
```

```
theta.cur <- theta.prop
func.Cur <- func.Prop
n.accept <- n.accept + 1
}
func.Samp[n, 1] <- func.Cur
func.Samp[n, 2] <- theta.cur
func.Samp[n, 3] <- n
}
return(func.Samp)
}
```

Now we need to define the posterior function we are going to sample from:

```
testfunc <- function(lambda) {
return(dgamma(lambda , shape = alpha , rate = mu))}

testfunc.metropolis <- function(lambda) {
return(log10(testfunc(lambda)))}
```

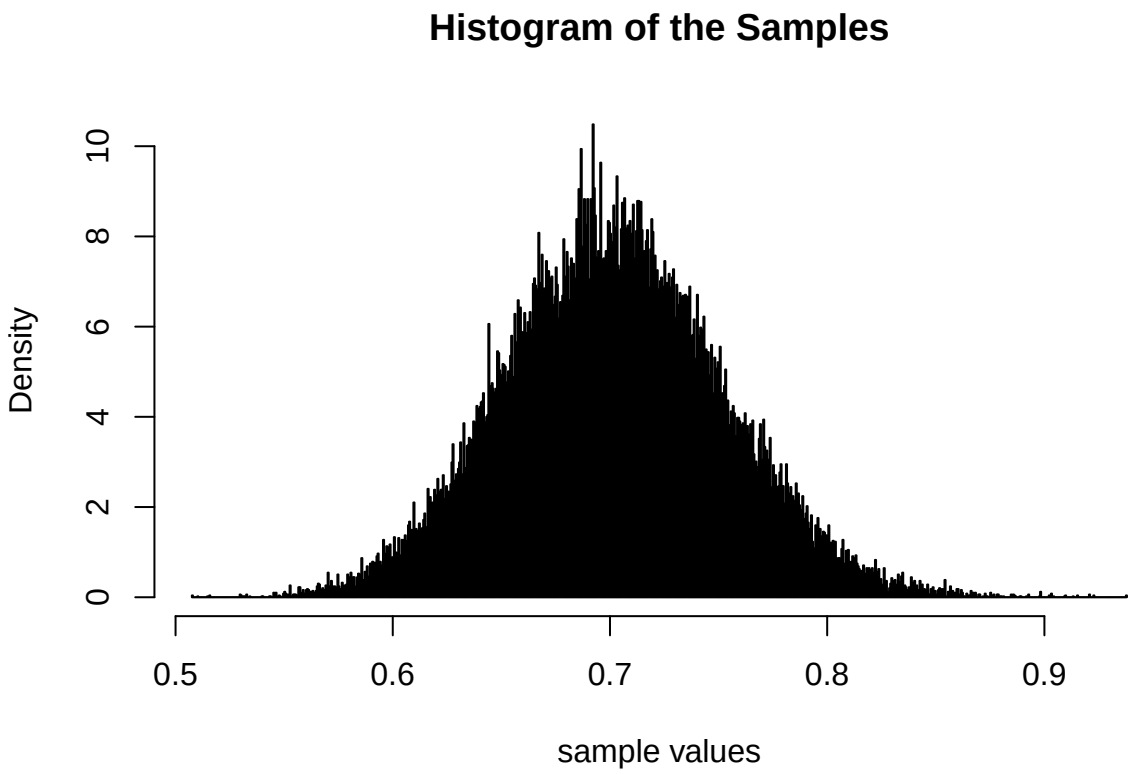
Now we are ready to sample from the desired posterior:

```
lambda.init <- 0.1
sample.sig <- 0.1
n.sample <- 10^5
demo <- TRUE

set.seed(20190513)
chain <- metropolis.1dim(func=testfunc.metropolis ,
theta.init = lambda.init ,
n.sample = n.sample ,
sigma = sample.sig)
```

```
mcmc.data = chain[1000:nrow(chain) , 2]

hist(mcmc.data , breaks = 1000 , freq = F , main = 'Histogram of the Samples'  , xlab = 'sample values'
```




```
mean = mean(mcmc.data)
variance = mean(mcmc.data ^ 2)  - mean ^ 2
median = quantile(mcmc.data , 0.5)
ci = quantile(mcmc.data , c(0.025 , 0.975))

print(mean)
```

```
## [1] 0.7018181
```

```
print(variance)
```

```
## [1] 0.00250942
```

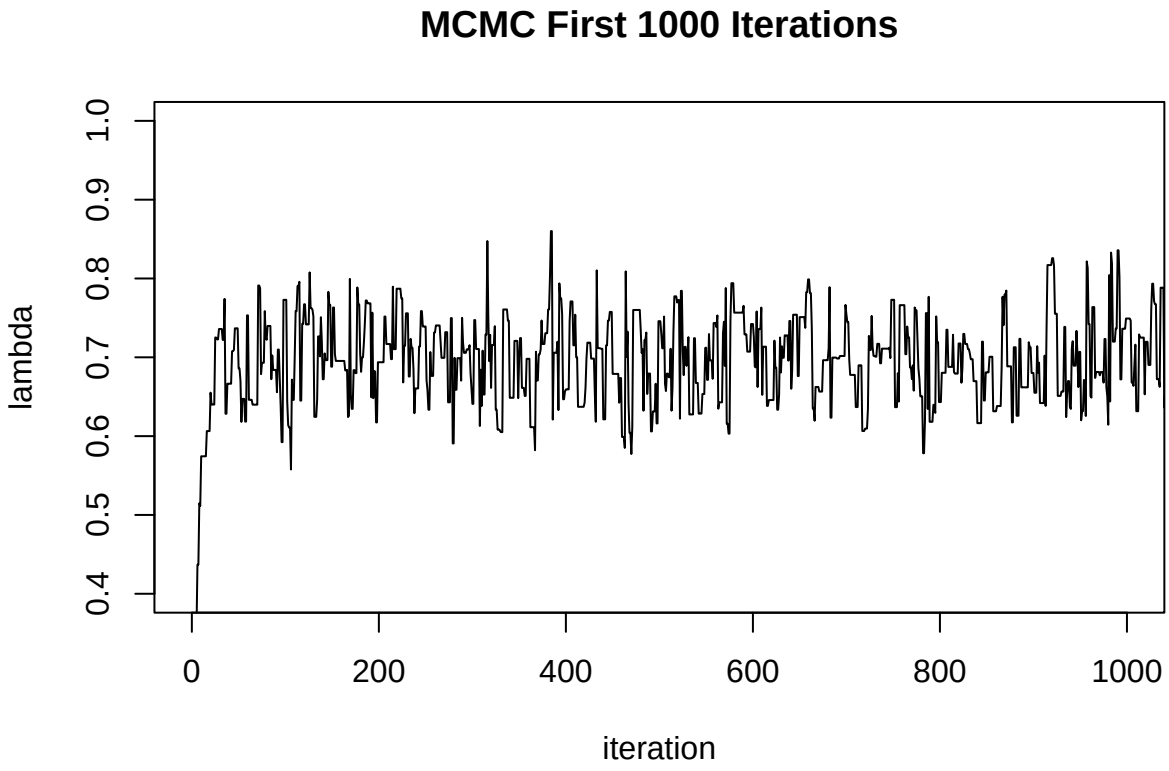
```
print(median)
```

```
##       50%
## 0.7009178
```

```
print(ci)
```

```
##      2.5%     97.5%
## 0.6065487 0.8023146
```

```
plot(chain[,3] , chain[,2] , type ='l' , ylim = c(0.4 , 1) , xlab = 'iteration' , ylab = 'lambda', xlim
```



Exercise 3

(a) The unbiased frequentist estimator for a Bernoulli process is simply:

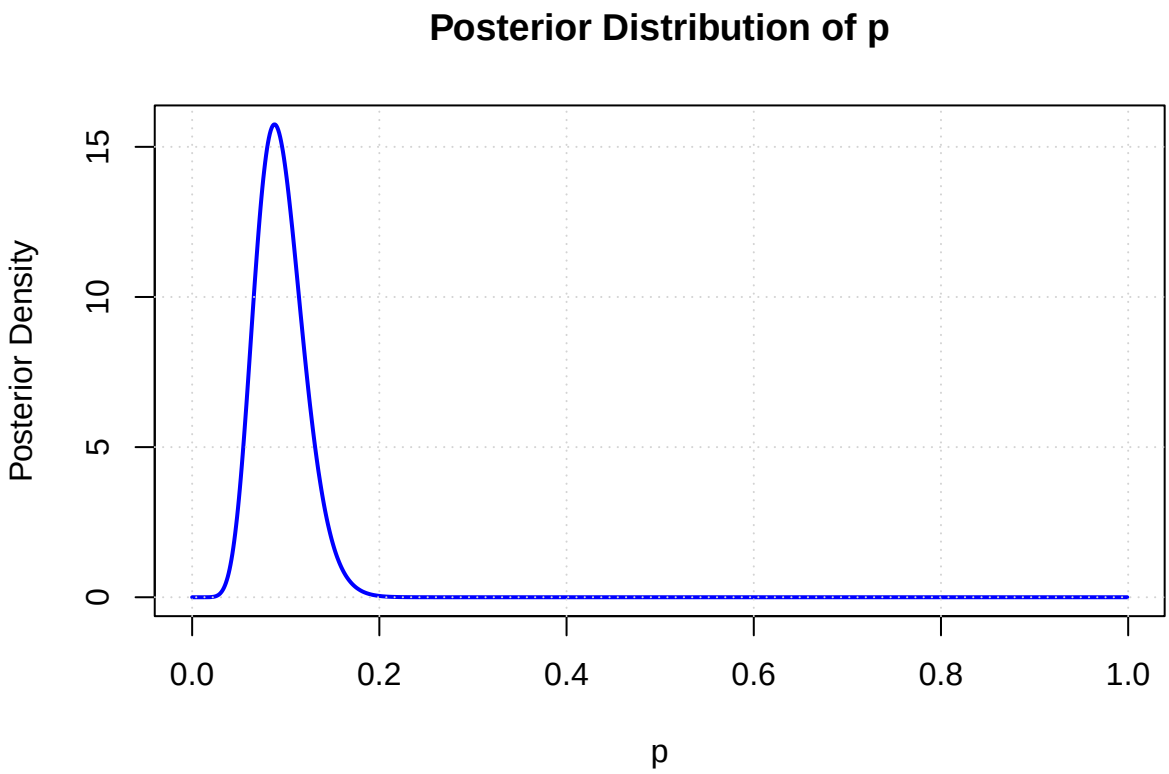
```
y = 11
n = 116
p.hat = y/n
p.hat
```

```
## [1] 0.09482759
```

```
p <- seq(from = 0, by = 0.001, length.out = 1000)
post.p <- dbeta(p, 1 + y, n - y + 10)

# Create the plot
plot(p, post.p, type = 'l', xlab = "p", ylab = "Posterior Density",
     main = "Posterior Distribution of p", col = "blue", lwd = 2)

# Add grid lines
grid()
```



(b)

```
# Adjust the plot margins
par(mar = c(5, 5, 4, 2) + 0.1)
```

```
mean = 0.001 * sum(post.p * p)
variance = 0.001 * sum(p ^ 2 * post.p) - mean ^ 2

sigma = sqrt(variance)

print(mean)
```

(c)

```
## [1] 0.09448819
```

```
print(c(mean - 2 * sigma , mean + 2 * sigma))
```

```
## [1] 0.04277982 0.14619656
```

So the mean of the posterior is considered as the Bayesian estimator for p, and the 95% credibility area is considered as the area 2*σ around the mean.

(d) For the frequentist approach, we need to assume the null hypothesis to be true, and using the binomial distribution, find the corresponding p-value :

```
p.value = pbinom(11 , n , 0.1 , lower.tail = T) + pbinom(105 , n , 0.1 , lower.tail = F)
p.value
```

```
## [1] 0.5043125
```

As we see the p-value is much higher than 0.05, so the null hypothesis is not rejected.

For the Bayesian approach, we can use the result of the posterior from the last section. p = 0.1 is within the 95% credibility interval, therefore the null hypothesis is not rejected.

(e) We need to to the same task with a new value of y:

```
y = 9
n = 165
p.hat = y/n
p.hat
```

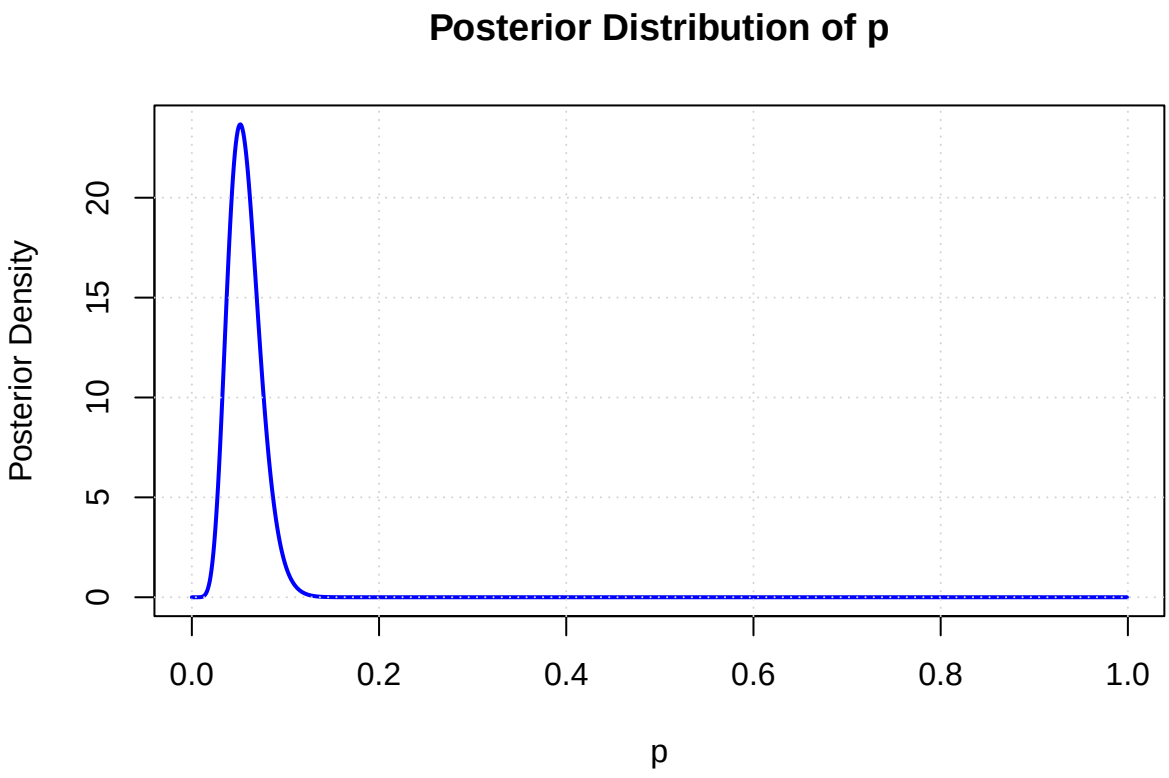
```
## [1] 0.05454545
```

(f) First let's do the Beta prior first:

```
p <- seq(from = 0, by = 0.001, length.out = 1000)
post.p <- dbeta(p, 1 + y, n - y + 10)

# Create the plot
plot(p, post.p, type = 'l', col = 'blue', lwd = 2, xlab = 'p', ylab = 'Posterior Density',
     main = 'Posterior Distribution of p')

# Add grid lines
grid()
```




```
# Adjust the plot margins
par(mar = c(5, 5, 4, 2) + 0.1)
```

```
mean = 0.001 * sum(post.p * p)
variance = 0.001 * sum(p ^ 2 * post.p) - mean ^ 2

sigma = sqrt(variance)

print(mean)
```

```
## [1] 0.05681818
```

```
print(c(mean - 2 * sigma , mean + 2 * sigma))
```

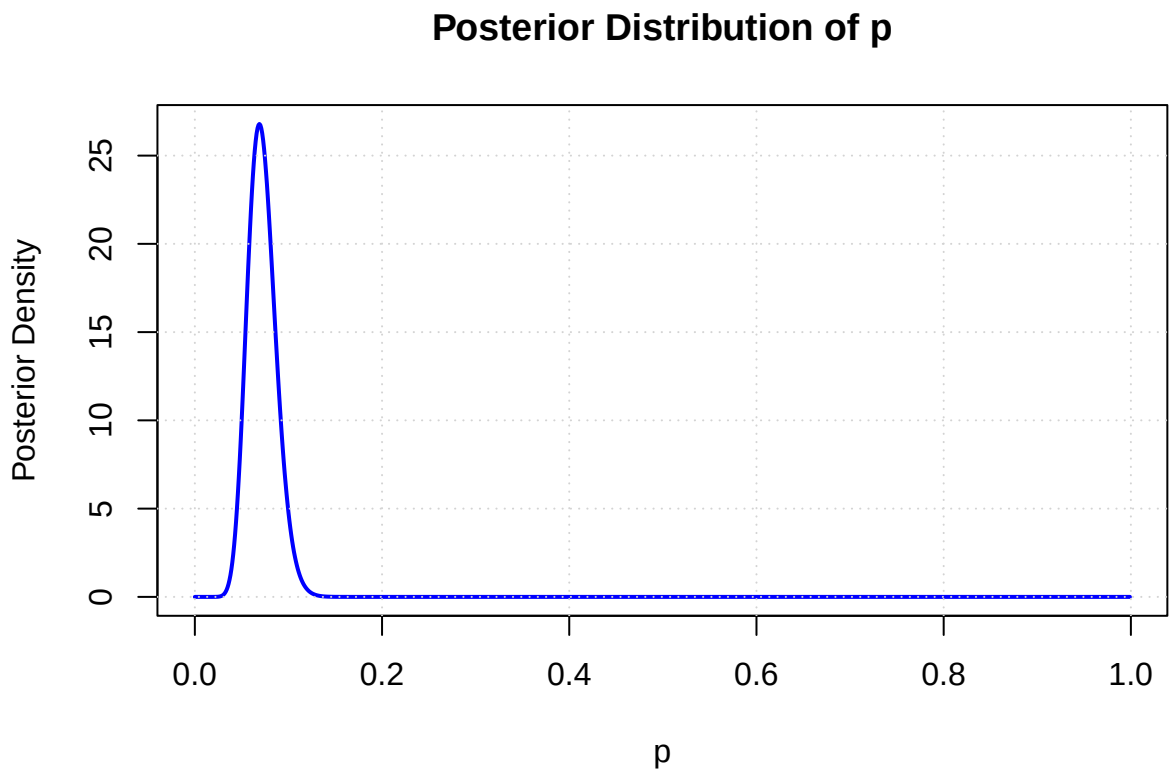
```
## [1] 0.02201774 0.09161862
```

Assuming the previous measurement as a prior we get:

```
p <- seq(from = 0, by = 0.001, length.out = 1000)
p.prior <- dbeta(p, 1 + 11, 116 - 11 + 10)
post.p <- dbeta(p, 1 + 11 + 9, 165 - 9 + 116 - 11 + 10)
```

```r
# Create the plot
plot(p, post.p, type = 'l', col = 'blue', lwd = 2, xlab = 'p', ylab = 'Posterior Density',
     main = 'Posterior Distribution of p')

# Add grid lines
grid()
```




```r
# Adjust the plot margins
par(mar = c(5, 5, 4, 2) + 0.1)
```

```r
mean = 0.001 * sum(post.p * p)
variance = 0.001 * sum(p ^ 2 * post.p) - mean ^ 2

sigma = sqrt(variance)

print(mean)
```

(g)

```
## [1] 0.07191781
```

```r
print(c(mean - 2 * sigma , mean + 2 * sigma))
```

```
## [1] 0.04173167 0.10210395
```

(h) Frequentist approach:

```r
p.value = pbinom(9 , n , 0.1 , lower.tail = T) + pbinom(107 , n , 0.1 , lower.tail = F)
p.value
```

```
## [1] 0.0274604
```

We see that the p-value is smaller than 0.05 and therefore the null hypothesis is rejected.

Bayesian approach: The p = 0.1 is almost inside the 95% credibility interval.

Exercise 4

We start by defining the model:

```r
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```r
library(coda)

data <- list(n = 116, y = 11)

model <- jags.model("model.txt", data = data)
```

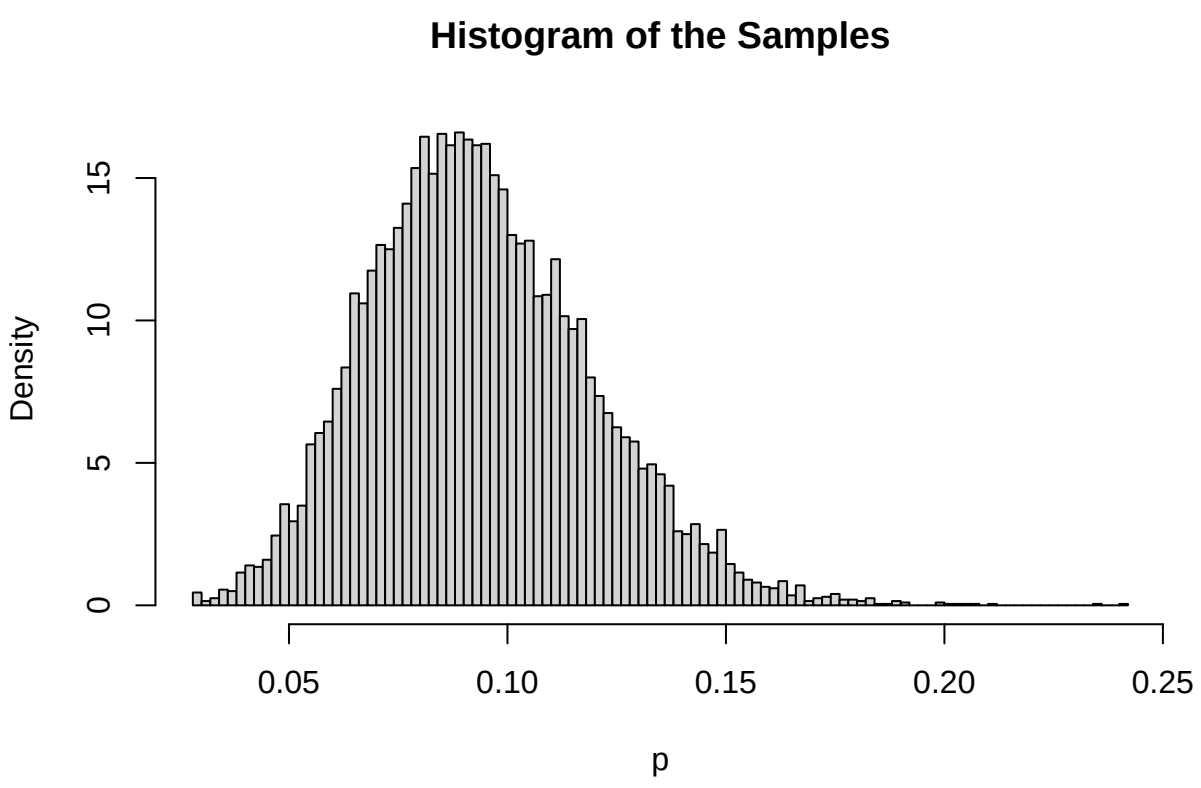
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 1
##    Total graph size: 5
##
## Initializing model
```

```r
samples <- coda.samples(model, variable.names = "p", n.iter = 10000)

summary(samples)
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##      0.0938052      0.0253570      0.0002536      0.0003207
##
## 2. Quantiles for each variable:
##
##    2.5%     25%     50%     75%   97.5%
## 0.04961 0.07587 0.09171 0.11009 0.14812
```

```
p.df = as.data.frame(as.mcmc(samples))
hist(p.df$p , breaks = 100 , freq = F , main = 'Histogram of the Samples'  , xlab = 'p' )
```




```
mean = mean(p.df$p)
std = sd(p.df$p)
ci = c(mean - 2 *std , mean + 2 * std)

print(mean)
```

```
## [1] 0.09380517
```

```
print(std)
```

```
## [1] 0.02535696
```

```
print(ci)
```

```
## [1] 0.04309125 0.14451908
```