Lab 2

Dustin Duncan

2024-01-16

Bayesian Statistical Modeling Winter 2024

Lab Exercise, Week 2.5

This assignment is due on Thursday, 1/27/2024. Submit the Rmd and pdf to gradescope:

These problems follow up on the example of estimating the percentage of the Earth's surface that is water.

Problem 1 (Warm up)

Let's refresh ourselves on how grid approximation works. Say that after nine tosses, get the following sequence of water/land observations.

W L W L L W W W W

We wish to estimate how much of the globe is actually covered by water. To do this implement a grid approximation algorithm and plot the posterior. You should get a plot which looks like figure 2.7 in the book.

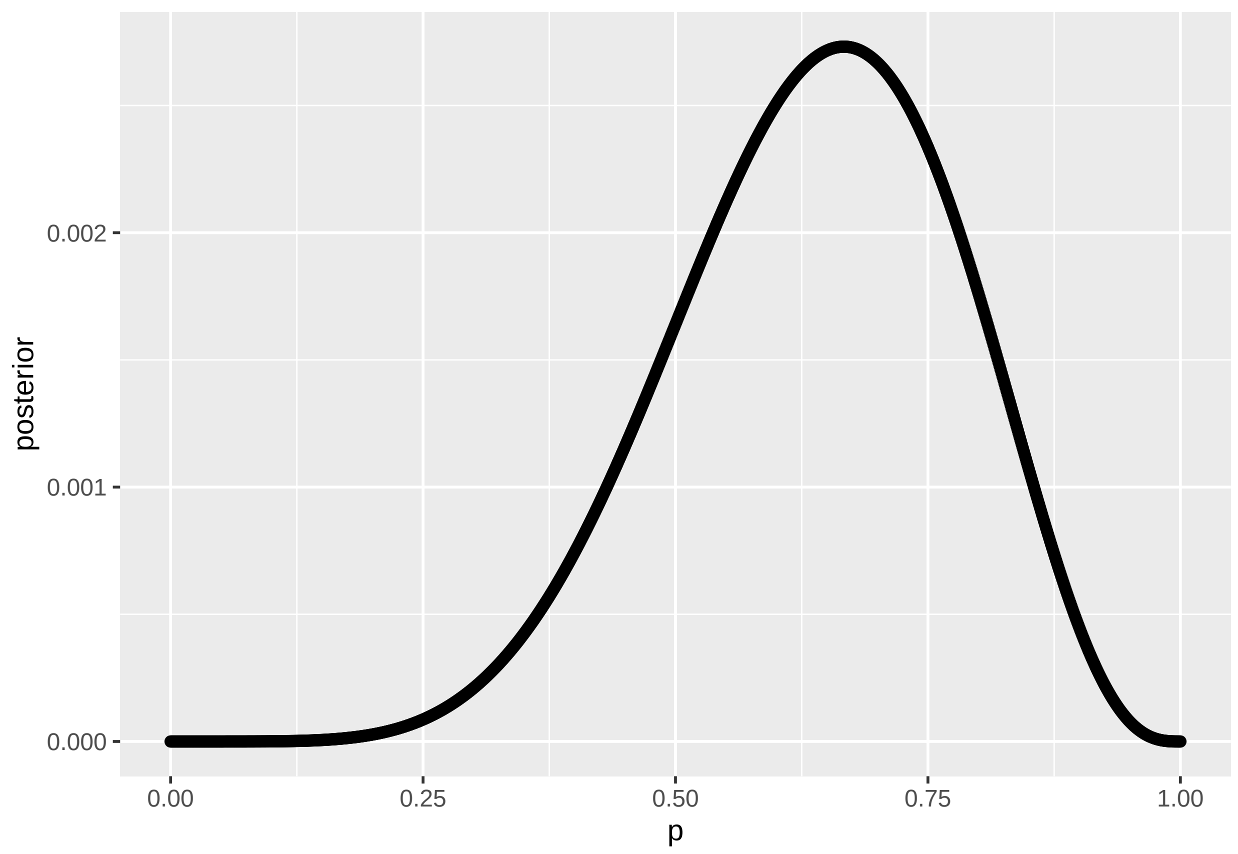
(Try your best to write this from memory and using tibbles. The answer is below, so if you get stuck sneak a peak at it. Your previous homework/class materials, section 2.4.3 of the book, or the first part of chapter 3's exercises also have code for grid approximation, but not all of them use tibbles.)

One way:

```r
tosses <- 9
water <- 6

stepsize = 0.001

post <- tibble(p = seq(from = 0, to = 1, by = stepsize)) %>%
  mutate(likelihood = dbinom(water, size = tosses, prob = p),
         prior = 1,
         raw_posterior = likelihood*prior,
         posterior = raw_posterior/sum(raw_posterior))

(post)
```

```
## # A tibble: 1,001 x 5
##        p likelihood prior raw_posterior posterior
##    <dbl>      <dbl> <dbl>         <dbl>     <dbl>
##  1 0       0            1      0         0
##  2 0.001   8.37e-17     1      8.37e-17  8.37e-19
##  3 0.002   5.34e-15     1      5.34e-15  5.34e-17
##  4 0.003   6.07e-14     1      6.07e-14  6.07e-16
##  5 0.004   3.40e-13     1      3.40e-13  3.40e-15
##  6 0.005   1.29e-12     1      1.29e-12  1.29e-14
##  7 0.006   3.85e-12     1      3.85e-12  3.85e-14
##  8 0.007   9.68e-12     1      9.68e-12  9.68e-14
##  9 0.008   2.15e-11     1      2.15e-11  2.15e-13
## 10 0.009   4.34e-11     1      4.34e-11  4.34e-13
## # i 991 more rows
```

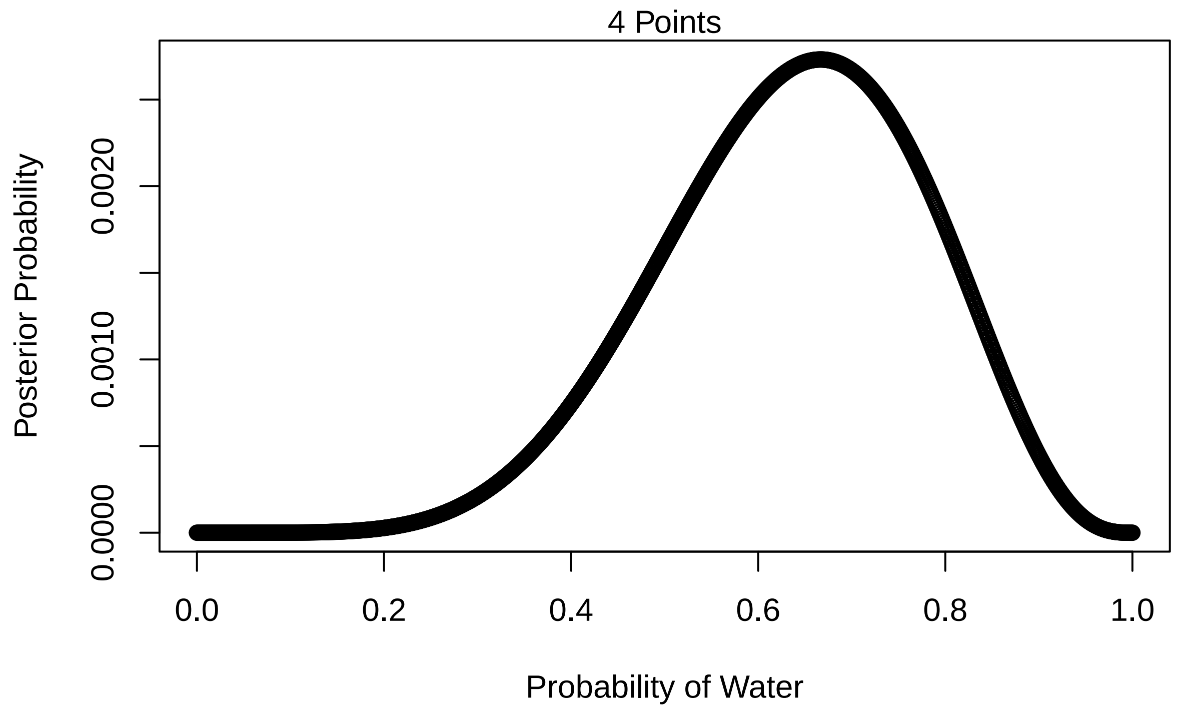
```r
ggplot(data = post, aes(x = p, y = posterior)) +
  geom_point()
```



Another way: (by the book), making everything as individual objects

```r
# define grid
p_grid <- seq(from = 0, to = 1, by = stepsize)

# define prior
prior <- rep(1, )

# compute likelihood at each value in grid
likelihood <- dbinom(6, size = 9, prob = p_grid)

# compute product of likelihood and prior
```

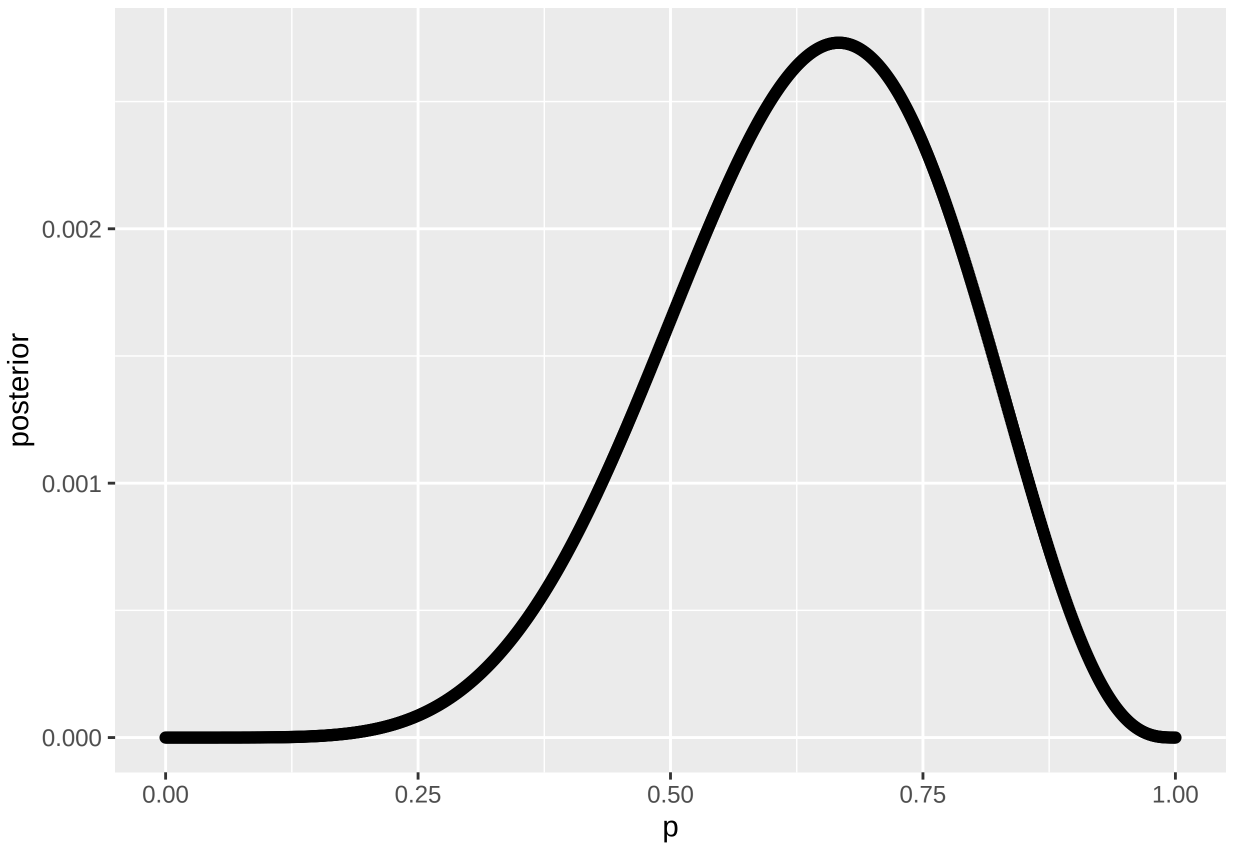
```
unstd.posterior <- likelihood * prior

# standardize the posterior so it sums up to 1
posterior <- unstd.posterior / sum(unstd.posterior)

# display posterior distribution
plot(p_grid, posterior, type = "b",
     xlab = "Probability of Water", ylab = "Posterior Probability")
mtext("4 Points")
```



```
# One possible answer for grid Approximation using tibbles.
# The rest of this lab assumes that your posterior grid approximation is
# stored in a tibble and that the proposed p values are in a column called "p".
tosses<-9
waters<-6
stepsize=0.001  # How small each grid cell is


post_approx <- tibble(p = seq( from=0 , to=1 , by=stepsize ) ) %>%
  mutate(  likelihood = dbinom( waters , size=tosses , prob=p),
           prior  = 1,
           raw.posterior=likelihood*prior,
           posterior = raw.posterior/sum(raw.posterior))

(post_approx)
```

```
## # A tibble: 1,001 x 5
##         p likelihood prior raw.posterior posterior
##     <dbl>      <dbl> <dbl>         <dbl>     <dbl>
##   1 0       0              1       0          0
##   2 0.001   8.37e-17     1       8.37e-17  8.37e-19
##   3 0.002   5.34e-15     1       5.34e-15  5.34e-17
##   4 0.003   6.07e-14     1       6.07e-14  6.07e-16
##   5 0.004   3.40e-13     1       3.40e-13  3.40e-15
##   6 0.005   1.29e-12     1       1.29e-12  1.29e-14
##   7 0.006   3.85e-12     1       3.85e-12  3.85e-14
##   8 0.007   9.68e-12     1       9.68e-12  9.68e-14
##   9 0.008   2.15e-11     1       2.15e-11  2.15e-13
##  10 0.009   4.34e-11     1       4.34e-11  4.34e-13
## # i 991 more rows
```

```r
ggplot(data=post_approx,aes(x=p,y=posterior)) +
  geom_point()
```



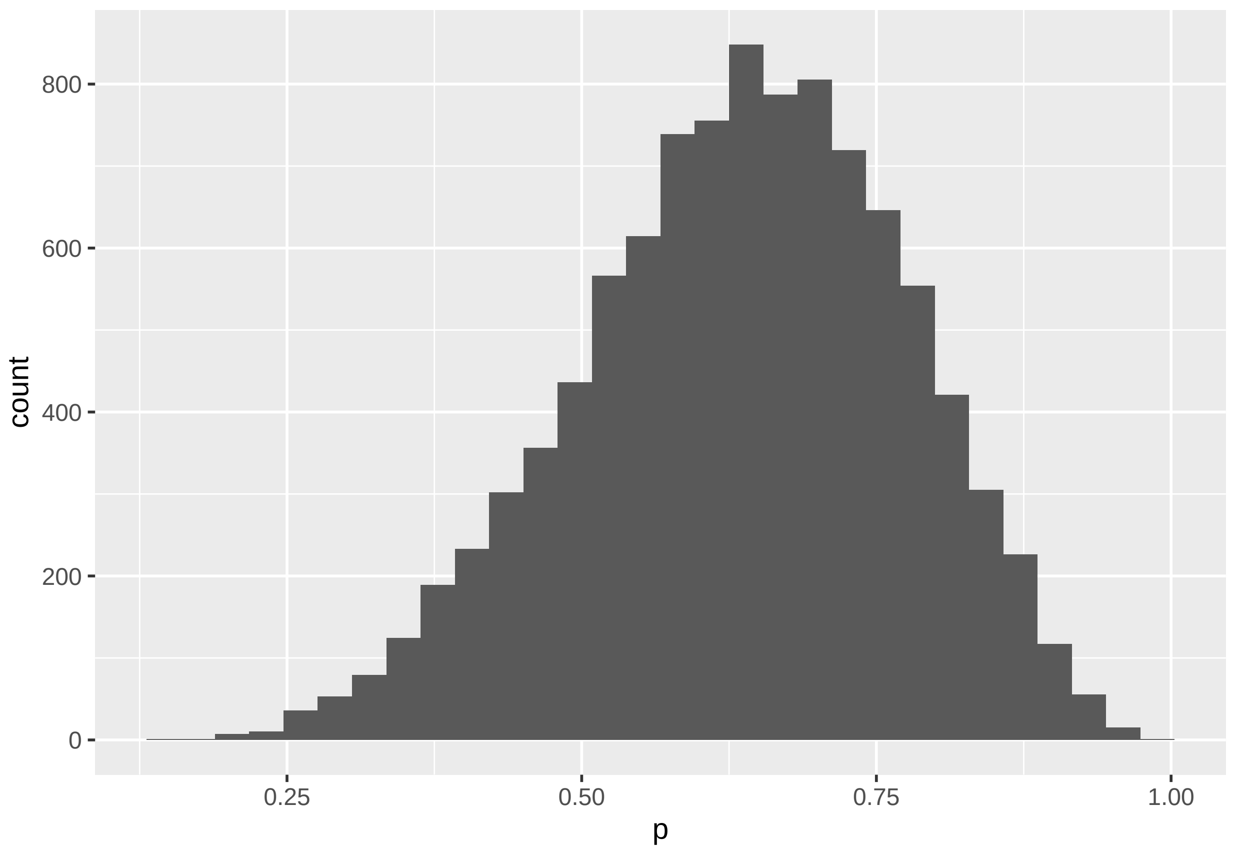
Problem 2

Take your posterior distribution and draw 10000 samples from it. It is often good form to set a random seed to set up the pseudo-random number drawing routine used by R. By setting this seed you will always get the exact same “random” series of draws, which makes stochastic algorithms replicable. We’ll use the seed of 100.

```r
set.seed(100)

nsamp <- 1e4  # Number of samples we will draw

samples1 <- tibble(p = sample(post_approx$p, size = nsamp, prob = post_approx$posterior, replace = TRUE
```

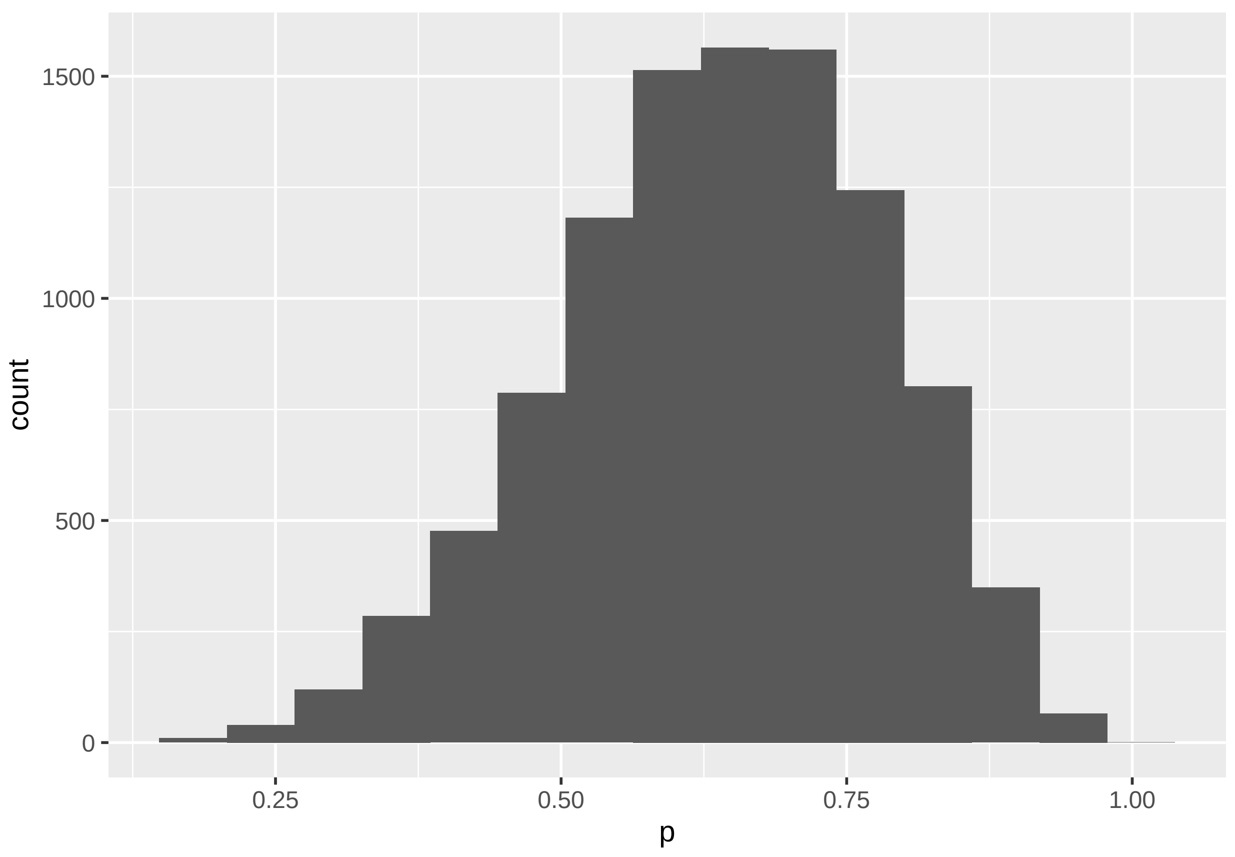
Have a look at the histogram of the sampled values for p. It should reflect our posterior distribution.

```r
ggplot(data=samples1 , aes(x=p))+ geom_histogram(bins=30)
```



How the book would do it:

```r
# drawing our samples
samples2 <- tibble( p = sample(p_grid, prob = posterior, size = 10000, replace = TRUE))

# plotting our samples
ggplot(data=samples2 , aes(x=p))+ geom_histogram(bins=15)
```



Aside: seeds Computers are in some sense deterministic, which makes it hard to generate true random numbers. Most languages by default use pseudo-random numbers. This is a deterministic algorithm which can generate numbers that look random, and are perfectly fine for most applications. The psuedo-random algorithm will start with a seed number and from there create numbers. If you don't set the seed manually then it is from something like the current time and date.

```r
set.seed(100)  # Set the seed
sample(1:100, 3)  # Draw some random numbers
sample(1:100, 3)  # Draw some more. The pseudo-random generator algorithm is now in a different state, 
```

Each time you draw random numbers the generator will update its state. If you wanted to draw the same set of random numbers twice then you'll have to reset the seed.

```r
set.seed(100)  # Set the seed
sample(1:100, 3)  # Draw some random numbers
set.seed(100)  # Set the seed back to starting value.
sample(1:100, 3)  # Draw some random numbers. These should be the same.
```

The answers in this lab are from a seed of 100, but you still might get a different answer depending on how you code up the lab, the functions you use, the version of R you use, and the order in which you run things. What's important is that your answer is somewhat close (plus or minus say 50%).

Problem(s) 3

Pretend you don't have the complete posterior but only the samples, as would be common when working on real life problems. Use the samples you've made to answer all the easy problems at the end of chapter three. We've written in the answers so that you can double check that you get something similar, but write the code to calculate the answer yourself.

3E1

How much of the posterior is below $p = 0.2$?

```r
# Answer: approx 2e-04, but can vary by an order of magnitude because the number of samples below p=0.2

sum(samples1$p < 0.2) / 1e4
```

```
## [1] 4e-04
```

3E2

How much of the posterior is above $p = 0.8$?

```r
# Answer: ~0.1122

sum(samples1 > 0.8) / 1e4
```

```
## [1] 0.1122
```

3E3

How much of the posterior is between $p = 0.2$ and $p = 0.8$?

```r
# Answer: ~0.8856

sum(samples1 > 0.2 & samples1 < 0.8) / 1e4
```

```
## [1] 0.8856
```

3E4

20% of the posterior probability lies below which value of p? (The quantile function computes this from a sample.)

```r
# Answer: p = ~0.517

quantile(samples1$p, 0.2)
```

```
##   20%
## 0.517
```

3E5

20% of the posterior probability lies above which value of p? We still use the quantile function but now compute the 80th percentile

```r
# Answer: p = ~0.756

quantile(samples1$p, 1 - 0.2)
```

```
##   80%
## 0.756
```

3E6

Which values of p contain the narrowest interval equal to 66% of the posterior probability? (The `HPDI` function finds this "highest posterior density interval".)

```r
# Answer: p approx between 0.506 and 0.773

HPDI(samples1, prob = 0.66)
```

```
## |0.66 0.66|
## 0.506 0.773
```

3E7

Which values of p contain the central most 66% of the posterior distribution? (This can be calculated using the traditional `PI`, percentile interval, function.)

** This works when your samples are stored as a value but not as a dataframe. For some reason HPDI works but PI you will need to specify the variable that you want the values from.

```r
# Answer: p approx between 0.501 and 0.770
PI(samples2$p, prob = 0.66)
```

```
##   17%   83%
## 0.502 0.775
```

```r
PI(samples1$p, prob = 0.66)
```

```
##   17%   83%
## 0.501 0.770
```

Problem(s) 4

Now do all the medium chapter 3 problems.

3M1

Redo the grid approximation but with a different set of data where there are 8 water in 15 tosses.

```r
tosses1 <- 15
waters1 <- 8
stepsize = 0.001

post_approx2 <- tibble(p = seq(from = 0, to = 1, by = stepsize)) %>%
  mutate(likelihood = dbinom(waters1, size = tosses1, prob = p),
         prior = 1,
         raw_posterior = likelihood*prior,
         posterior = raw_posterior / sum(raw_posterior)
         )
```

3M2

Draw 10,000 samples and compute the 0.9 HPDI.

```r
# Answer: p approx between 0.332 and 0.719
set.seed(100)

nsamp <- 10000

samples3 <- tibble(p = sample(post_approx2$p, size = nsamp, prob = post_approx2$posterior, replace = TRU

HPDI(samples3, prob = 0.9)
```

```
##  |0.9  0.9|
## 0.332 0.719
```

3M3

What is the probability of observing 8 out of 15 water from our posterior?

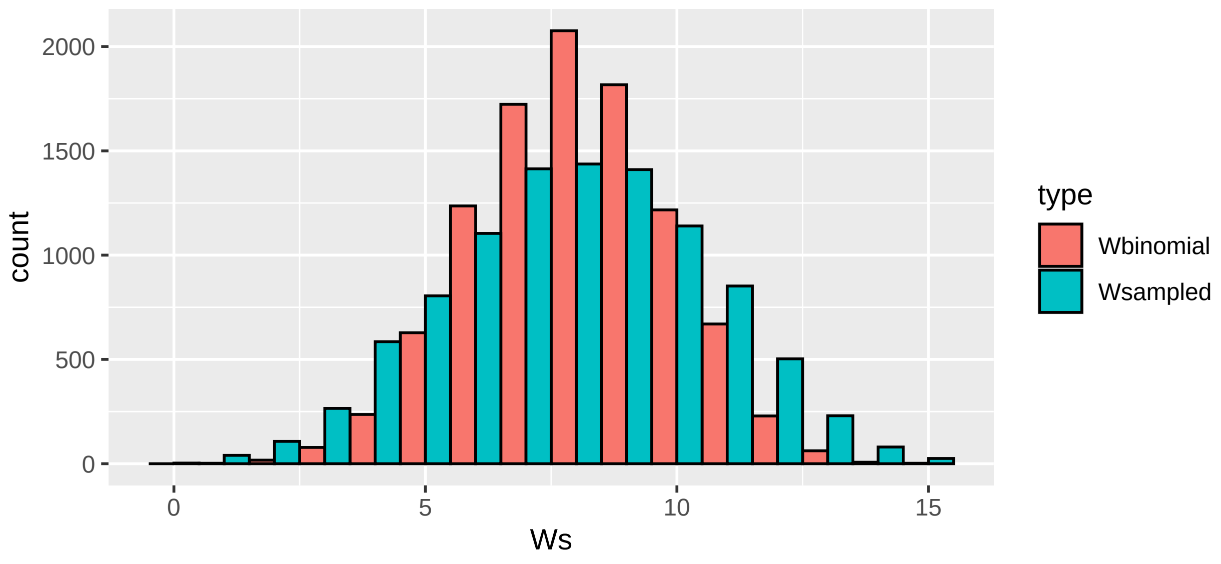
To calculate this we'll need a posterior predictive check. This means simulating full datasets while taking into account both the variance in the parameters (ie the posterior uncertainty in p) along with the variance in the samples themselves.

We've already sampled values for p from the previous problem, so those will be our sample. Add onto that dataframe binomial draws with a total of 15 tosses per dataset.

We can use the mutate command here, but it is a bit complicated. We need to create a list of binomial draws that is as long as the samples table. We can get the length with *n()*. We'll also add a binomial sample using the same ratio of W to L as in the data (i.e. 8/15).

```r
# In this code n() gives the number of rows in the current tibble
post.sims <-
  mutate(
    samples3,
    Wsampled = rbinom(n(), size = 15, prob = p),
    Wbinomial = rbinom(n(), size = 15, prob = 8 / 15)
  )

?gather
# Put the data in another form so we can make a histogram
# replace 6 and 7 with the index numbers for your Wsampled and Wbinomial column
```

```r
post.sims.long <- gather(post.sims, "type", "Ws", c("Wsampled", "Wbinomial"))

post.sims.longer <- post.sims %>%
  pivot_longer(col = c("Wsampled", "Wbinomial"),
               names_to = "type",
               values_to = "Ws")


ggplot(data = post.sims.long, aes(x = Ws, fill = type)) +
  geom_histogram(binwidth = 1,
                 colour = "black",
                 position = "dodge")
```



To answer the question, what is the probability of observing 8 our of 15 water from our posterior?

```r
# Answer: ~0.1423

mean(post.sims.long$Ws == 8)
```

```
## [1] 0.17565
```

3M4

Use the posterior to determine the probability of observing 6 out of 9 waters. We first need to do a new posterior simulation, using the existing samples for p, but with an experiment with 9 trials. Then calculate the fraction of draws with 6 out of 9 waters:

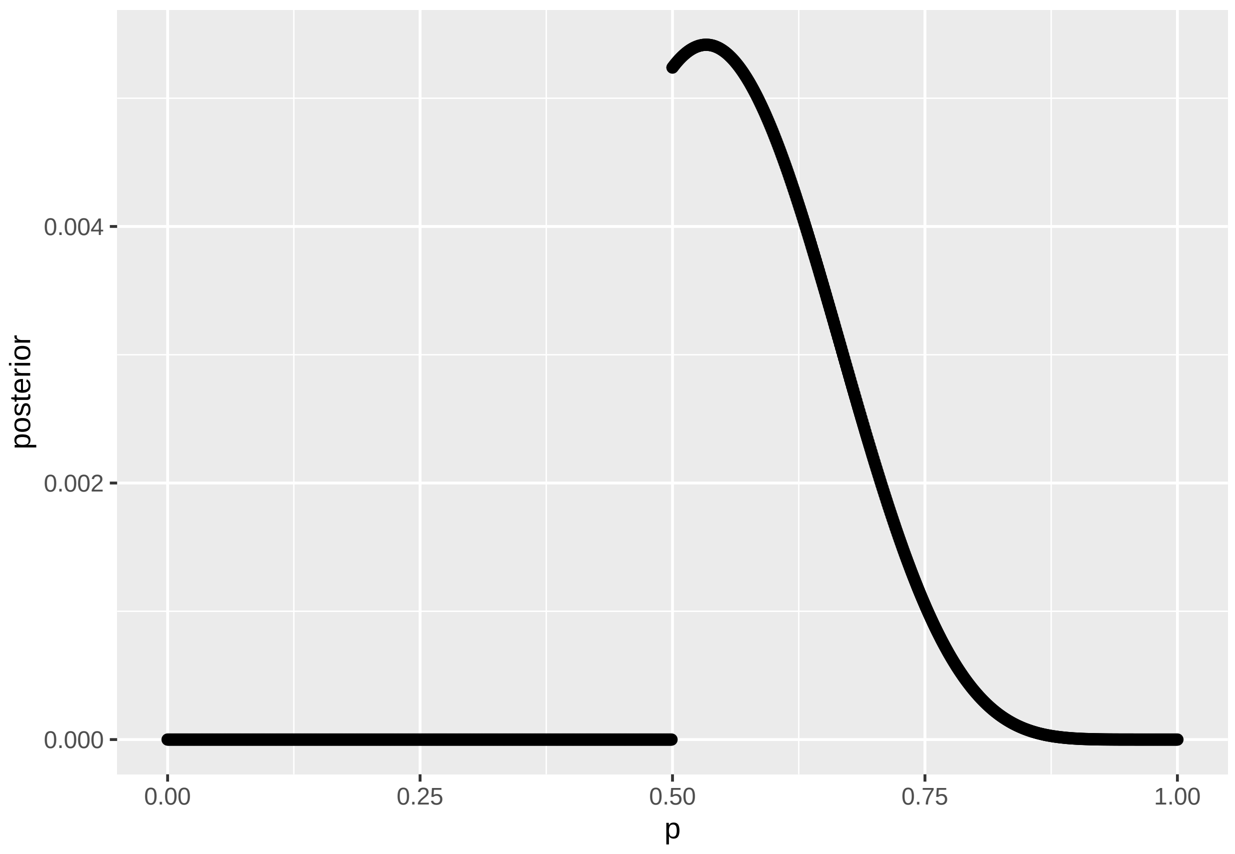
```r
# Answer: ~0.1746
new <- rbinom(1e4, size = 9, prob = samples3$p)
mean(new == 6)
```

```
## [1] 0.1807
```

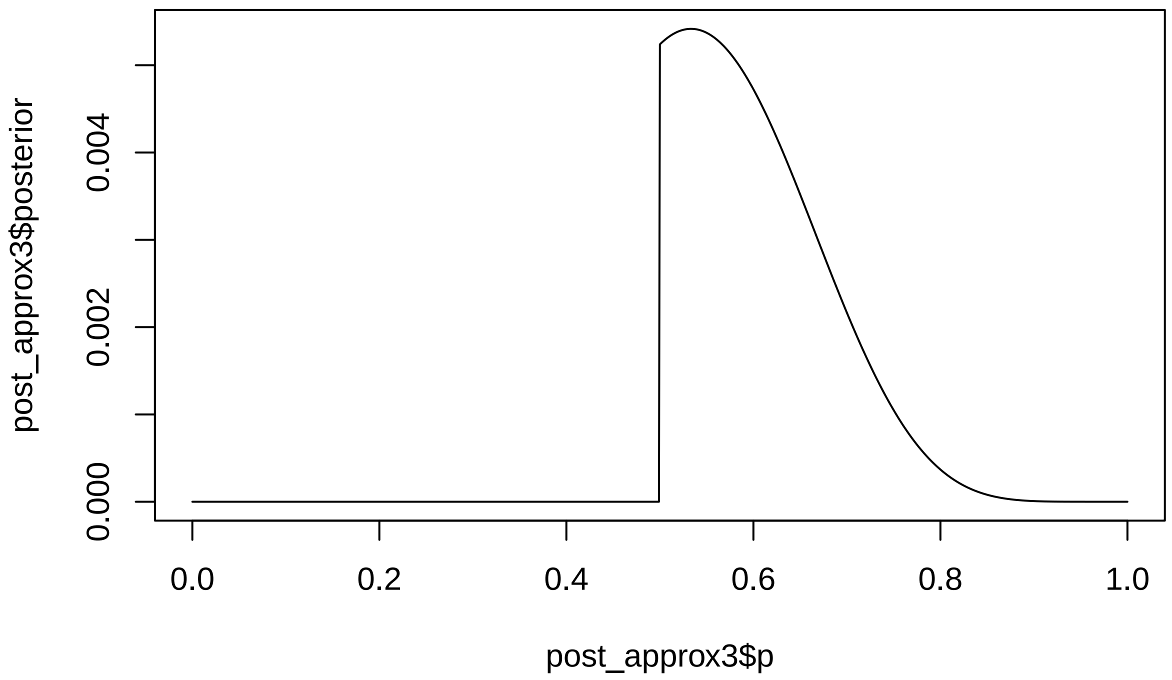
3M5

Repeat everything from 3M1 to here, but now alter the prior so that it is zero at and below $p = 0.5$ and a constant positive above. (I.e. we are certain that more than 50% of the globe is water.)

```
# Answers:
# Your posterior should look like a binomial distribution that's suddenly squashed to zero for p less t
# The 90% HPDI is between 0.50 and 0.71
# The probability of observing 8 waters in a sample of 15 is around 0.1551

# Creating our model
post_approx3 <- tibble(p = seq(from = 0, to = 1, by = stepsize)) %>%
  mutate(likelihood = dbinom(8, size = 15, prob = p),
         prior = ifelse(p < 0.5, 0, 1),
         raw.posterior = likelihood*prior,
         posterior = raw.posterior / sum(raw.posterior))

# Plotting the posterior using ggplot
ggplot(post_approx3, aes(x = p, y = posterior)) +
  geom_point()
```



```
# Plotting the posterior using base R
plot(post_approx3$p, post_approx3$posterior, type = "l")
```



```r
# Taking samples from our model and computing the 90% HPDI
samples4 <- tibble(p = sample(post_approx3$p, size = 10000, prob = post_approx3$posterior, replace = TRU

HPDI(samples4, p = 0.90)
```

```
##  |0.9  0.9|
## 0.500 0.712
```

```r
# Doing a posterior predictive simulation on our model and finding the likelihood of 8 waters in a samp

post.sims2 <-
  mutate(
    samples4,
    Wsampled = rbinom(n(), size = 15, prob = p),
    Wbinomial = rbinom(n(), size = 15, prob = 8 / 15)
  )

# Using outdated gather function
post.sims2.long <- gather(post.sims2, "type", "Ws", c("Wsampled", "Wbinomial"))

# Using newer pivot_longer function in dplyr
post.sims2.longer <- post.sims2 %>%
  pivot_longer(col = c("Wsampled", "Wbinomial"),
               names_to = "type",
               values_to = "Ws")
```

```r
# Posterior predictive check of our model:
new2 <- rbinom(10000, size = 15, prob = samples4$p )

# Finding the probability of 8 waters in 15 tosses
mean(new2 == 8)
```

```
## [1] 0.1524
```

```r
mean(post.sims2.long$Ws == 8)
```

```
## [1] 0.1775
```

3M6

Go back to the situation where we have a flat prior. How many observations would it take to have the 99% percentile interval be only 1% wide? We'll assume that the observations are generated by a binomial probability distribution with a probability of water of 0.7. This means that the data are actually generated by the "small world" model we are using to fit the data.

A reasonable strategy for solving this problem is to automate our code so that we can change the number of observations and determine how wide the 99% PI is. To do this we will write a function to return the width of the 99% interval for a specific number of observations. In R, a variable name can have a function assigned to it using the `function` command.

Below is an outline of the function. Try to fill it in. What the function returns when you run it will be whatever its very last line is.

```r
# Make a function that takes in a number of observations N
# and outputs the 99% highest posterior density interval
int.width <- function(N) {

  # Set the local function parameters
  # (These parameters live only in this function, and take
  # priority over any global parameters with the same name.)
  p_true <- 0.7
  stepsize <- 0.001
  nsamp <- 1e4


  # Generate the sampled data
  # < write code here >
  W <- rbinom(1, size = N, prob = p_true)


  # do our grid approximation
  # < write code here >
  post_approx <- tibble(p = seq(from = 0, to = 1, by = stepsize)) %>%
    mutate(likelihood = dbinom(W, size = N, prob = p),
           prior = 1,
           raw.posterior = likelihood*prior,
           posterior = raw.posterior/sum(raw.posterior)
           )
  # Sample values of p from the posterior distribution
  # < write code here >
  samples <- sample(post_approx$p, size = 10000, prob = post_approx$posterior, replace = TRUE)

  # Calculate the 99% interval
  # < write code here >

  PI99 <- PI(samples, prob = 0.99)

  # Calculate the width of that interval
  # < write code here >
  as.numeric(PI99[2] - PI99[1])
}
```

Note that the function `int.width` will return different values when you call it because it uses randomly generated data, as well as random samples from the posterior. You can check that it works by calling it on a number of observations.

```
# 99% percentile interval width when there are two observations
int.width(2)
```

```
## [1] 0.83402
```

```
# The width gets smaller as we make more observations
int.width(200)
```

```
## [1] 0.157005
```

```
int.width(2000)
```

```
## [1] 0.053
```

We'd like to be able to give this function a list of observation numbers and have it apply itself to each element of that list. For example, *int.width(c(2, 5, 8))* would be equal to *c(int.width(2), int.width(5), int.width(8))*.

To do this we can "vectorize" the function. In R functions can be passed to other functions. The *Vectorize* function takes in a function and returns a version of the function that works on lists like we described.

```
# Make a vectorized version of the function. We'll call it int.width.v
int.width.v <- Vectorize(int.width)
```

Now *int.width.v(c(2, 5, 8))* will give us *c(int.width(2), int.width(5), int.width(8))*

```
int.width(2)
```

```
## [1] 0.828005
```

```r
int.width(5)
```

```
## [1] 0.787005
```

```r
int.width(8)
```

```
## [1] 0.695005
```

```r
int.width( c(2, 5, 8) )  # Not vectorized, only uses first element in vector
```

```
## [1] 0.922
```

```r
int.width.v( c(2, 5, 8) )  # Vectorized, uses all elements and outputs a list
```

```
## [1] 0.927005 0.787000 0.648010
```

Now that we have our function, we can use it repeatedly for different sample sizes (number of observations) to see how big our sample has to be to reduce the interval to 0.1. This will take a little bit of time to run.

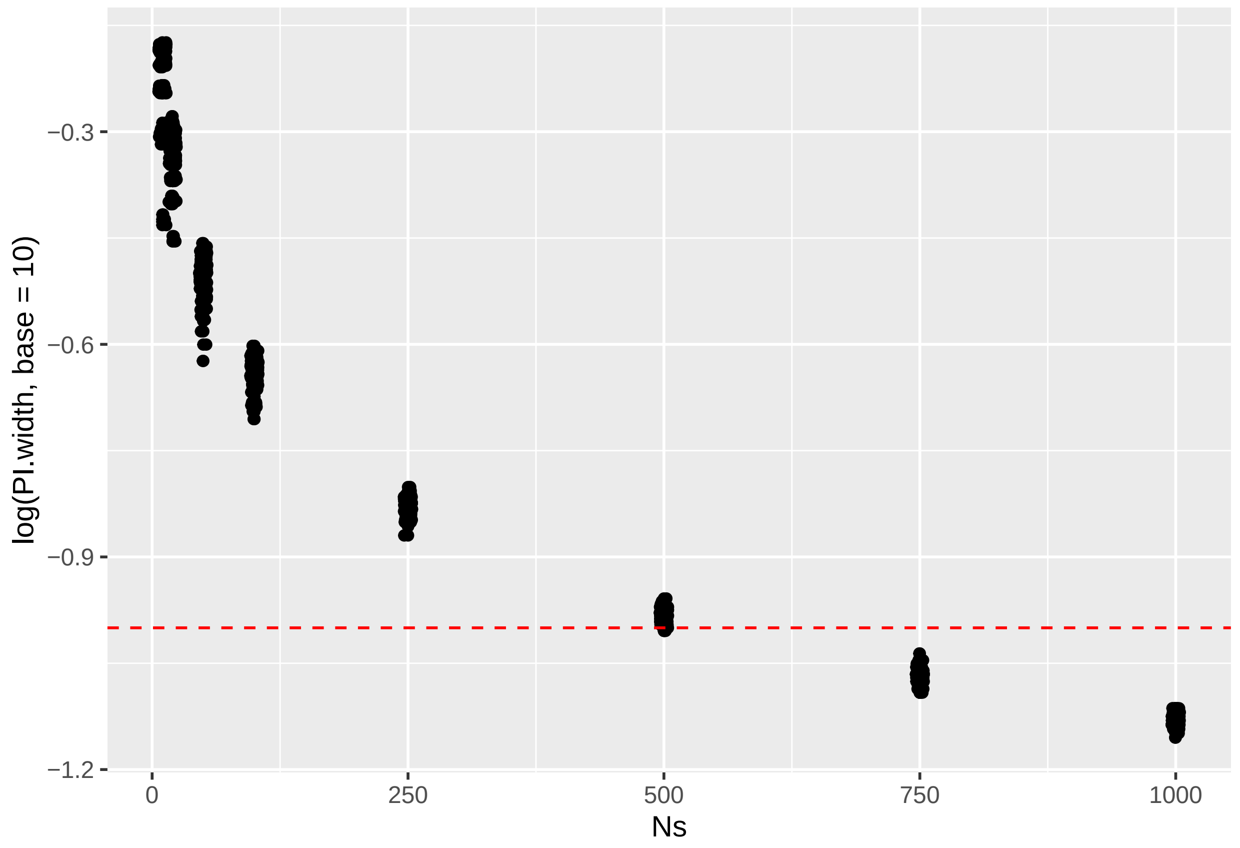
```r
Nlist <- tibble(Ns = rep(c(10, 20, 50, 100, 250, 500, 750, 1000), each = 100))

Nlist <- mutate(Nlist, PI.width = int.width.v(Ns))
```

Have a look at your list and you will see that it has slightly different values when the function is applied to the same sample size.

```r
view(Nlist)
```

Finally we'll plot it and see the PI changes with sample size. We'll put the PI width on a log scale to make it easier to see when it gets below 0.1 (note that $log(0.1, 10) = -1$) and put a red horizontal line in.

```r
ggplot(data = Nlist, aes(x = Ns, y = log(PI.width, base = 10), group = Ns)) +
  geom_point() +
  geom_jitter() +
  geom_hline(aes(yintercept = log(0.1, base = 10)), color = "red", linetype = "dashed")
```



A little after how many observations do we see that our 99% percentage interval goes below a width of 0.1?

After about 500 observations we see that our 0.99 Percentage Interval drops below a width of about 0.1.

(You should get about 500.)