Chapter 2, Part 1, Examples

Brooke Anderson

2/13/2020

Part 1

```
library(tidyverse)
library(purrr)
library(magrittr)
library(viridis)
library(Biostrings)
```

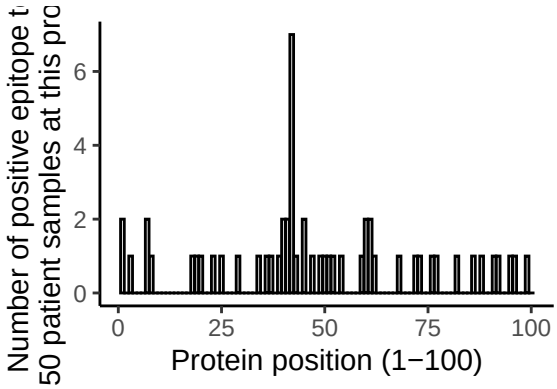
```
load("data/e100.RData")
e100
```

```
##   [1] 2 0 1 0 0 0 2 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 1
##  [38] 0 1 2 2 7 1 0 2 0 1 0 1 1 1 1 0 1 0 0 0 0 1 2 2 1 0 0 0 0 0 1 0 0 0 1 1 0
##  [75] 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0
```

```
e100_tidy <- tibble(protein_position = 1:100,
                    n_pos_epitope_tests = e100)
head(e100_tidy)
```

```
## # A tibble: 6 x 2
##   protein_position n_pos_epitope_tests
##              <int>               <dbl>
## 1                1                   2
## 2                2                   0
## 3                3                   1
## 4                4                   0
## 5                5                   0
## 6                6                   0
```

```
e100_tidy %>%
  ggplot(aes(x = protein_position, y = n_pos_epitope_tests)) +
  geom_col(fill = "white", color = "black") +
  theme_classic() +
  labs(x = "Protein position (1-100)",
       y = "Number of positive epitope tests\nacross 50 patient samples at this protein position")
```



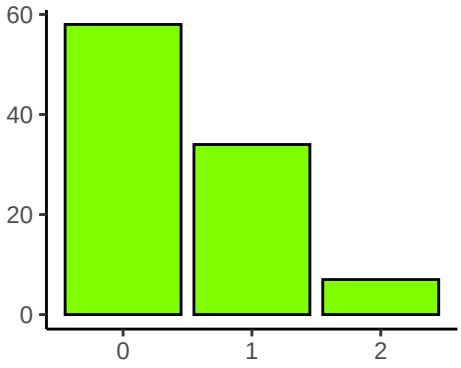

```
(e100_summary <- e100_tidy %>%
  group_by(n_pos_epitope_tests) %>%
  count())
```

```
## # A tibble: 4 x 2
## # Groups:   n_pos_epitope_tests [4]
##   n_pos_epitope_tests     n
##                 <dbl> <int>
## 1                   0    58
## 2                   1    34
## 3                   2     7
## 4                   7     1
```

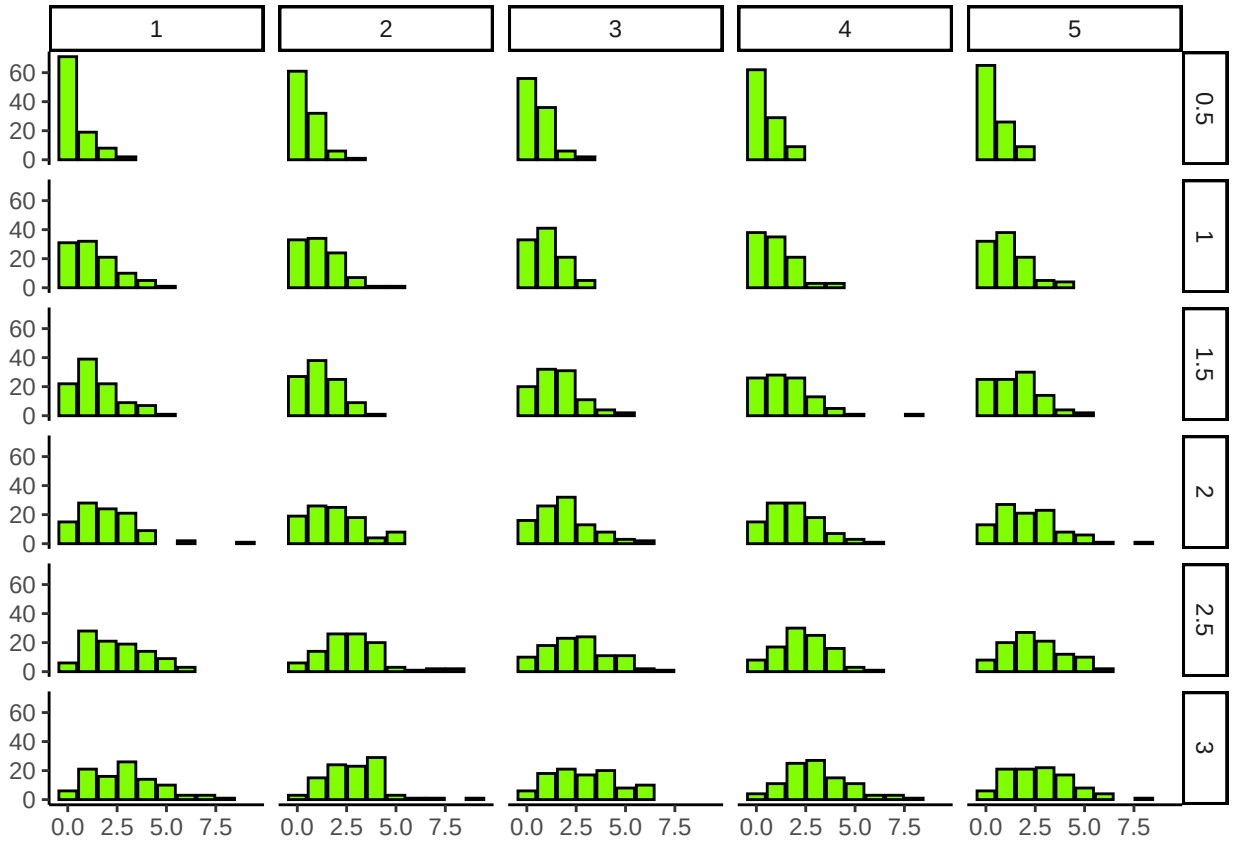
```
sim_poisson <- expand_grid(sample = 1:5,
                           lambda = seq(from = 0.5, to = 3, by = 0.5)) %>%
  mutate(sim_data = map(lambda, ~ rpois(n = 100, lambda = .x))) %>%
  unnest(sim_data) %>%
  group_by(sample, lambda, sim_data) %>%
  count()

head(sim_poisson)
```

```
## # A tibble: 6 x 4
## # Groups:   sample, lambda, sim_data [6]
##   sample lambda sim_data     n
##    <int>  <dbl>    <int> <int>
## 1      1    0.5        0    71
## 2      1    0.5        1    19
## 3      1    0.5        2     8
## 4      1    0.5        3     2
## 5      1    1          0    31
## 6      1    1          1    32
```

```
ggplot(filter(e100_summary, n_pos_epitope_tests != 7),
       aes(x = n_pos_epitope_tests, y = n)) +
  geom_col(fill = "chartreuse", color = "black") +
  theme_classic() +
  theme(axis.title = element_blank())
```




```
ggplot(sim_poisson, aes(x = sim_data, y = n)) +
  geom_col(fill = "chartreuse", color = "black") +
  facet_grid(lambda ~ sample) +
  theme_classic() +
  theme(axis.title = element_blank())
```



```r
tidy_loglike <- tibble(lambda = seq(from = 0.05, to = 0.95, length = 100)) %>%
  mutate(data_prob = map(lambda, ~ dpois(e100, .x))) %>%
  unnest(data_prob) %>%
  mutate(log_prob_data = log(data_prob)) %>%
  group_by(lambda) %>%
  summarize(loglikelihood = sum(log_prob_data))

head(tidy_loglike)
```

```
## # A tibble: 6 x 2
##   lambda loglikelihood
##    <dbl>         <dbl>
## 1 0.05           -183.
## 2 0.0591         -175.
## 3 0.0682         -168.
## 4 0.0773         -162.
## 5 0.0864         -157.
## 6 0.0955         -152.
```

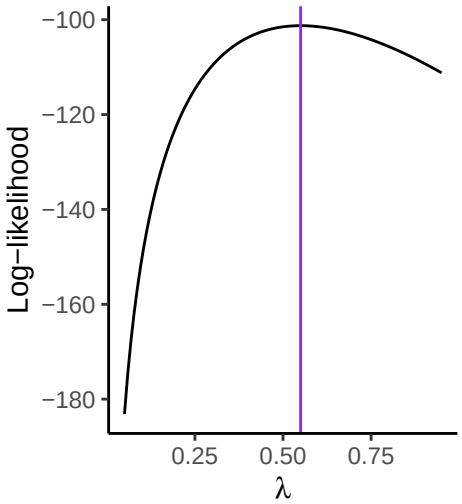
```r
tidy_loglike %>%
  filter(loglikelihood == max(loglikelihood))
```

```
## # A tibble: 1 x 2
##   lambda loglikelihood
##    <dbl>         <dbl>
## 1   0.55         -101.
```

```r
mean(e100)
```

```
## [1] 0.55
```

```r
ggplot(tidy_loglike, aes(x = lambda, y = loglikelihood)) +
  geom_line() +
  theme_classic() +
  labs(x = expression(lambda), y = "Log-likelihood") +
  geom_vline(xintercept = mean(e100), color = "purple")
```



```
staph <- readDNAStringSet("data/staphsequence.ffn.txt")
class(staph)
```

```
## [1] "DNAStringSet"
## attr(,"package")
## [1] "Biostrings"
```

```
staph
```

```
##   A DNAStringSet instance of length 2650
##        width seq                                                names
##    [1]  1362 ATGTCGGAAAAAGAAATTTGGG...AAAGAAATAAGAAATGTATAA lcl|NC_002952.2_c...
##    [2]  1134 ATGATGGAATTCACTATTAAAA...TTACCAATCAGAACTTACTAA lcl|NC_002952.2_c...
##    [3]   246 GTGATTATTTTGGTTCAAGAAG...ATTCATCAAGGTGAACAATGA lcl|NC_002952.2_c...
##    [4]  1113 ATGAAGTTAAATACACTCCAAT...CAAGGTGAAATTATAAAGTAA lcl|NC_002952.2_c...
##    [5]  1932 GTGACTGCATTGTCAGATGTAA...TATGCAAACTTAGACTTCTAA lcl|NC_002952.2_c...
##    ...   ... ...
## [2646]   720 ATGACTGTAGAATGGTTAGCAG...ACTCCTTTACTTGAAAAATAA lcl|NC_002952.2_c...
## [2647]  1878 GTGGTTCAAGAATATGATGTAA...CTCCAAAGGGTGAGTGACTAA lcl|NC_002952.2_c...
## [2648]  1380 ATGGATTTAGATACAATTACGA...CAATTCTGCTTAGGTAAATAG lcl|NC_002952.2_c...
## [2649]   348 TTGGAAAAAGCTTACCGAATTA...TTTAATAAAAAGATTAAGTAA lcl|NC_002952.2_c...
## [2650]   138 ATGGTAAAACGTACTTATCAAC...CGTAAAGTTTTATCTGCATAA lcl|NC_002952.2_c...
```

```
staph[1]
```

```
##   A DNAStringSet instance of length 1
##     width seq                                                names
## [1]  1362 ATGTCGGAAAAAGAAATTTGGGA...AAAAAGAAATAAGAAATGTATAA lcl|NC_002952.2_c...
```

```
class(staph[1])
```

```
## [1] "DNAStringSet"
## attr(,"package")
## [1] "Biostrings"
```

```
staph[[1]]
```

```
##   1362-letter "DNAString" instance
## seq: ATGTCGGAAAAAGAAATTTGGGAAAAAGTGCTTGAA...GTAGAGAATCTTGAAAAAGAAATAAGAAATGTATAA
```

```
class(staph[[1]])
```

```
## [1] "DNAString"
## attr(,"package")
## [1] "Biostrings"
```

```
letterFrequency(staph[[1]], letters = c("A", "C", "G", "T"))
```

```
##   A   C   G   T
## 522 219 229 392
```

Part 2

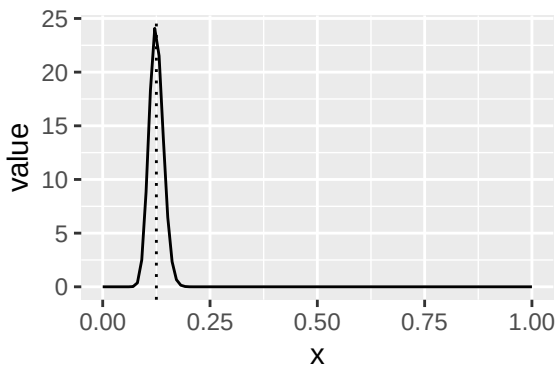
Say you think that the **parameter** for your probability distribution isn't certain. Instead, you think it is a random variable from a beta distribution with parameters: $\alpha = 50$, $\beta = 350$.

First, here's that beta distribution:

```
beta_dist <- tibble(x = seq(from = 0, to = 1, length = 100)) %>%
  mutate(value = map_dbl(x, ~ dbeta(x = .x, shape1 = 50, shape2 = 350)))
beta_dist %>%
  head()
```

```
## # A tibble: 6 x 2
##        x    value
##    <dbl>    <dbl>
## 1 0      0.
## 2 0.0101 3.53e-34
## 3 0.0202 5.54e-21
## 4 0.0303 6.33e-14
## 5 0.0404 2.17e- 9
## 6 0.0505 3.02e- 6
```

```
ggplot(beta_dist, aes(x = x, y = value)) +
  geom_line() +
  geom_vline(xintercept = 50 / (50 + 350),
             linetype = 3) # Theoretical mean of this distribution is alpha / alpha + beta
```



Now, if we want to try to simulate what **outcomes** from this process might look like, we'll do a two-step simulation:

1. Simulate different values of the parameter for the binomial distribution, *since we now think that this parameter is itself a random variable* (this is the part 1 of the heart of the Bayesian paradigm for this example). We think this parameter follows a **beta distribution**, with parameters of $\alpha = 50$ and $\beta = 350$, so we'll draw random samples from that distribution for this step.
2. Once we have a lot of random samples of this parameter value from step 1, we can take random draws from binomial distributions to simulate the actual data. We will simulate in this case from *lots* of examples of the binomial distribution family, using the parameters from step 1.

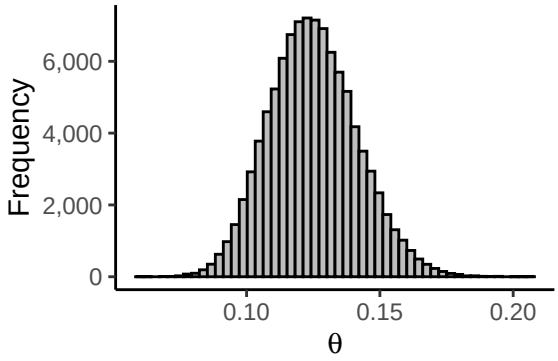
Step 1:

```
bayesian_simulation <- tibble(index = 1:100000,
                              rtheta = rbeta(100000, 50, 350))

bayesian_simulation %>%
  head()
```

```
## # A tibble: 6 x 2
##   index rtheta
##   <int>  <dbl>
## 1     1  0.144
## 2     2  0.124
## 3     3  0.142
## 4     4  0.124
## 5     5  0.125
## 6     6  0.125
```

Here's a histogram of the θ values for that draw:

```
ggplot(bayesian_simulation, aes(x = rtheta)) +
  geom_histogram(bins = 50, fill = "gray", color = "black") +
  theme_classic() +
  scale_x_continuous(name = expression(theta)) +
  scale_y_continuous(name = "Frequency", labels = scales::comma)
```



Step 2:

Next, for *each* of these simulated θ values (which, again, represent the parameter of a binomial distribution), we'll simulate the count of successes from a binomial experiment with 300 trials.

Remember that, if we were pretty sure that the data came from a binomial distribution with p = 0.10 (i.e., a 10% chance of success), here's how we would simulate the result of an experiment with 300 trials:

```
rbinom(n = 1, size = 300, prob = 0.10)
```

```
## [1] 20
```

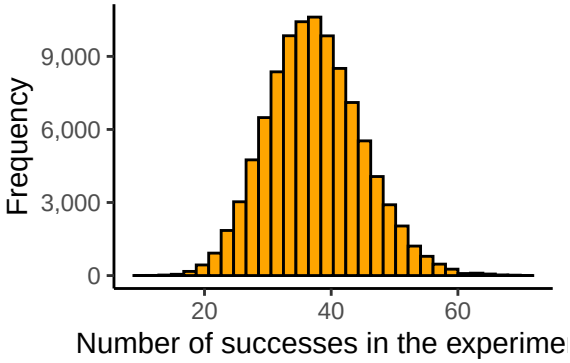
Now we'll just do this process for every value of θ that we got from sampling the beta distribution in step 1:

```
bayesian_simulation <- bayesian_simulation %>%
  mutate(sim_successes = map_dbl(rtheta, ~ rbinom(n = 1, prob = .x, size = 300)))
bayesian_simulation %>%
  head()
```

```
## # A tibble: 6 x 3
##   index rtheta sim_successes
##   <int>  <dbl>         <dbl>
## 1     1  0.144            44
## 2     2  0.124            41
## 3     3  0.142            45
## 4     4  0.124            38
## 5     5  0.125            45
## 6     6  0.125            39
```

Now we can look at a histogram of these simulated number of successes from step 2:

```
ggplot(bayesian_simulation, aes(x = sim_successes)) +
  geom_histogram(bins = 32, fill = "orange", color = "black") +
  theme_classic() +
  scale_x_continuous(name = "Number of successes in the experiment") +
  scale_y_continuous(name = "Frequency", labels = scales::comma)
```



This is an example of a simulation where we think that both the model parameter (θ) and the outcomes are random variables.

Adding in observed data

Now, we can try to update our belief about the probability distribution (in this case, by updating our belief about the distribution of the θ parameter) by *adding what we can learn from new data* (this is the second part of the heart of the Bayesian paradigm in this example). For example, the expected value of θ is, we think before we see any new data, about 0.125 (this is $\frac{\alpha}{\alpha+\beta}$ from the values we used for the prior beta distribution for θ).

If we have data where the probability of success is closer to 20%, we want our **updated** distribution for θ to have a higher expected value. If we observe in new data a probability of success that is 10%, on the other hand, we want the update distribution to have a lower expected value. If we have observed data with a probability of success right around 12.5%, then that should give us more confidence that this is a good expected value of θ, so we won't want the expected value to change, but we would like the distribution to get "sharper"—we're becoming more confident that the value is close to its expected value in that case.

In the example, we **observe data** where, out of 300 trials, there are 40 successes. This is a proportion of successes of $\frac{40}{300} = 0.133$. This is a little higher than the expected value in our prior distribution, so we should already know that when we add in the information from this new data, our beta distribution for θ should shift a little to the left.

Here's how we can "add" this data based on our simulated data: we can look *just* at the simulated values where the number of successes reflect our observed data (that is, where there are 40 simulated successes):

```
post_data <- bayesian_simulation %>%
  filter(sim_successes == 40)
dim(post_data)
```

```
## [1] 4757    3
```

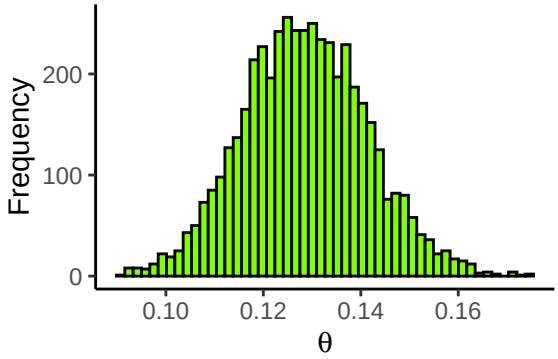
```
head(post_data)
```

```
## # A tibble: 6 x 3
##   index rtheta sim_successes
##   <int>  <dbl>         <dbl>
## 1    17  0.123            40
```

```
## 2     55  0.119            40
## 3     61  0.126            40
## 4     74  0.166            40
## 5     77  0.134            40
## 6     98  0.115            40
```

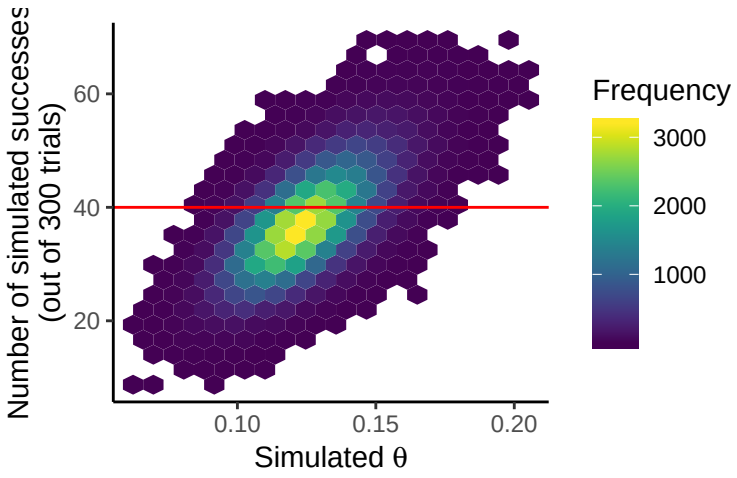
Here is a histogram of the values of θ in our simulation that resulted in 40 successes in the second step of the simulation:

```
ggplot(post_data, aes(x = rtheta)) +
  geom_histogram(bins = 50, fill = "chartreuse", color = "black") +
  theme_classic() +
  scale_x_continuous(name = expression(theta)) +
  scale_y_continuous(name = "Frequency", labels = scales::comma)
```



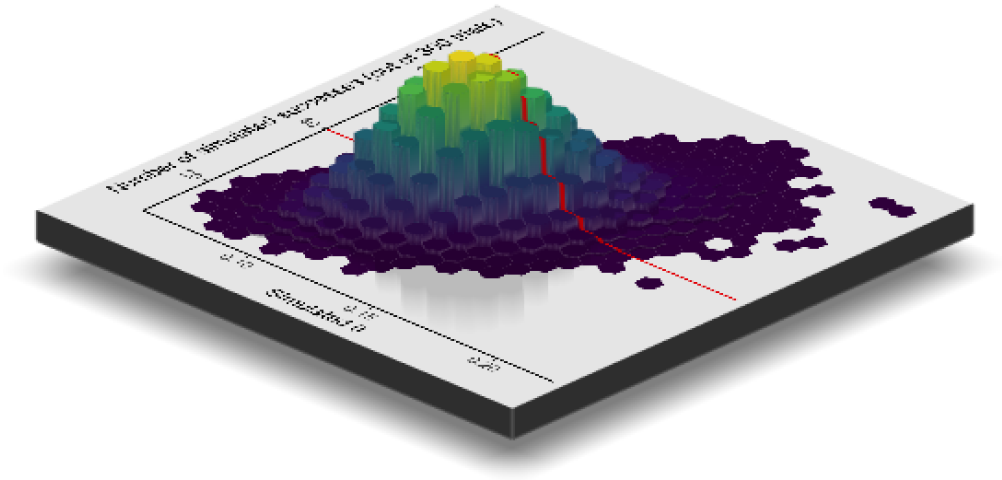
Here's another way to think of this: if we looked at the **bivariate distribution** of the simulations, looking at how the frequency of observed values vary by both the simulated θ value and the simulated count of successes based on that θ, then we now want to take a "slice" along that bivariate distribution right at the point where the number of successes is 40:

```
biv_dist <- ggplot(bayesian_simulation, aes(x = rtheta, y = sim_successes)) +
  geom_hex(bins = 20, size = 0) +
  scale_fill_viridis(name = "Frequency") +
  theme_classic() +
  labs(x = expression(paste("Simulated ", theta)),
       y = "Number of simulated successes\n(out of 300 trials)") +
  geom_hline(yintercept = 40, color = "red")
biv_dist
```



(If you'd like to try this in 3-D, you can do that now with the `rayshader` package:)

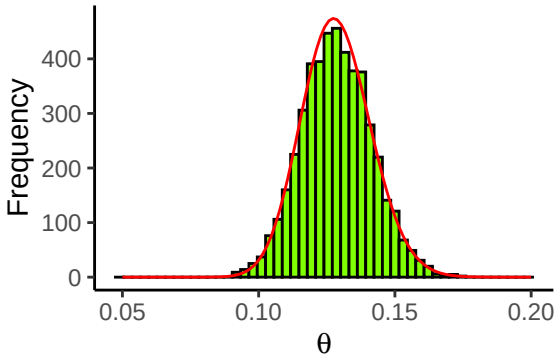
```
library(rayshader)
plot_gg(biv_dist + theme(legend.position = "none"),
        width = 4, height = 4, scale = 300, multicore = TRUE)
```



It turns out that there's a really nice theoretical solution for the posterior distribution if the prior distribution for the parameter was a beta distribution. It turns out that, if in your data you observe y successes out of n trials, then your posterior distribution for θ is also a beta distribution, this time with the first size parameter of $\alpha + \beta$ and the second size parameter of $\beta + (n - y)$.

We can create values from that distribution and add this as a line to the results from our simulation to see that the theoretical and simulated results agree pretty well:

```
beta_dist_post <- tibble(x = seq(from = 0.05, to = 0.2, length = 100)) %>%
  mutate(value = map_dbl(x, ~ dbeta(x = .x,
                                    shape1 = 50 + 40, shape2 = 350 + (300 - 40))))

ggplot(post_data, aes(x = rtheta)) +
  geom_histogram(bins = 50, fill = "chartreuse", color = "black") +
  theme_classic() +
  scale_x_continuous(name = expression(theta)) +
  scale_y_continuous(name = "Frequency", labels = scales::comma) +
  geom_line(data = beta_dist_post, aes(x = x, y = 15 * value), color = "red")
```



For the maximum a posteriori (MAP) estimate, based on simulation, it's just the most common value of the θs from the simulation that resulted in 40 successes in the simulation:

```
post_data %>%
  mutate(round_rtheta = round(rtheta, digits = 3)) %>%
  group_by(round_rtheta) %>%
  count() %>%
  arrange(desc(n)) %>%
  head()
```

```
## # A tibble: 6 x 2
## # Groups:   round_rtheta [6]
##   round_rtheta     n
##          <dbl> <int>
## 1        0.13    161
## 2        0.129   153
## 3        0.124   152
## 4        0.126   152
## 5        0.127   148
## 6        0.12    146
```

You can use quantiles of the simulations to get the 95% posterior credibility interval:

```
post_data %>%
  summarize(lower_pi = quantile(rtheta, 0.025),
            upper_pi = quantile(rtheta, 0.975))
```

```
## # A tibble: 1 x 2
##   lower_pi upper_pi
##      <dbl>    <dbl>
## 1    0.105    0.154
```

Example: nucleotide patterns

```
library("Biostrings")
library("BSgenome")
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: rtracklayer
```

```
available.genomes() %>%
  str_subset("Hsapiens") # You can use regular expressions to pull out human genome data
```

```
##  [1] "BSgenome.Hsapiens.1000genomes.hs37d5"
##  [2] "BSgenome.Hsapiens.NCBI.GRCh38"
##  [3] "BSgenome.Hsapiens.UCSC.hg17"
##  [4] "BSgenome.Hsapiens.UCSC.hg17.masked"
##  [5] "BSgenome.Hsapiens.UCSC.hg18"
##  [6] "BSgenome.Hsapiens.UCSC.hg18.masked"
##  [7] "BSgenome.Hsapiens.UCSC.hg19"
##  [8] "BSgenome.Hsapiens.UCSC.hg19.masked"
##  [9] "BSgenome.Hsapiens.UCSC.hg38"
## [10] "BSgenome.Hsapiens.UCSC.hg38.masked"
```

If you don't have the package with E. coli data yet, you'll need to install it from Bioconductor (uncomment

the first line below). The library loads an object called `Ecoli` with the E. coli genome data. You can use `str` to check this out.

```
# BiocManager::install("BSgenome.Ecoli.NCBI.20080805")
library("BSgenome.Ecoli.NCBI.20080805")
class(Ecoli)
```

```
## [1] "BSgenome"
## attr(,"package")
## [1] "BSgenome"
```

```
str(Ecoli)
```

```
## Formal class 'BSgenome' [package "BSgenome"] with 17 slots
##   ..@ pkgname           : chr "BSgenome.Ecoli.NCBI.20080805"
##   ..@ single_sequences  :Formal class 'RdaNamedSequences' [package "BSgenome"] with 3 slots
##   .. .. ..@ seqlengths:Formal class 'RdaCollection' [package "XVector"] with 2 slots
##   .. .. .. .. ..@ dirpath : chr "/Library/Frameworks/R.framework/Versions/3.6/Resources/library/BSge
##   .. .. .. .. ..@ objnames: chr "seqlengths"
##   .. .. ..@ dirpath   : chr "/Library/Frameworks/R.framework/Versions/3.6/Resources/library/BSgenome
##   .. .. ..@ objnames  : chr [1:13] "NC_008253" "NC_008563" "NC_010468" "NC_004431" ...
##   ..@ multiple_sequences:Formal class 'RdaCollection' [package "XVector"] with 2 slots
##   .. .. ..@ dirpath : chr "/Library/Frameworks/R.framework/Versions/3.6/Resources/library/BSgenome.E
##   .. .. ..@ objnames: chr(0)
##   ..@ source_url        : chr "ftp://ftp.ncbi.nih.gov/genomes/Bacteria/"
##   ..@ user_seqnames     : Named chr [1:13] "NC_008253" "NC_008563" "NC_010468" "NC_004431" ...
##   .. ..- attr(*, "names")= chr [1:13] "NC_008253" "NC_008563" "NC_010468" "NC_004431" ...
##   ..@ injectSNPs_handler:Formal class 'InjectSNPsHandler' [package "BSgenome"] with 4 slots
##   .. .. ..@ SNPlocs_pkgname          : chr(0)
##   .. .. ..@ getSNPcount              :function ()
##   .. .. ..@ getSNPlocs               :function ()
##   .. .. ..@ seqname_translation_table: chr(0)
##   ..@ .seqs_cache       :<environment: 0x7f8a0759b998>
##   ..@ .link_counts      :<environment: 0x7f8a0759b6c0>
##   ..@ nmask_per_seq     : int 0
##   ..@ masks             :Formal class 'RdaCollection' [package "XVector"] with 2 slots
##   .. .. ..@ dirpath : chr NA
##   .. .. ..@ objnames: chr(0)
##   ..@ organism          : chr "Escherichia coli"
##   ..@ common_name       : chr "E. coli"
##   ..@ provider          : chr "NCBI"
##   ..@ provider_version  : chr "2008/08/05"
##   ..@ release_date      : chr NA
##   ..@ release_name      : chr NA
##   ..@ seqinfo           :Formal class 'Seqinfo' [package "GenomeInfoDb"] with 4 slots
##   .. .. ..@ seqnames   : chr [1:13] "NC_008253" "NC_008563" "NC_010468" "NC_004431" ...
##   .. .. ..@ seqlengths : int [1:13] 4938920 5082025 4746218 5231428 4979619 4643538 5528445 5498450 5
##   .. .. ..@ is_circular: logi [1:13] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   .. .. ..@ genome     : chr [1:13] "2008/08/05" "2008/08/05" "2008/08/05" "2008/08/05" ...
```

It looks like we're going to pull out the sequence for one strain based on its NCB accession number:

```
ecoli <- Ecoli$NC_010473
class(ecoli)
```

```
## [1] "DNAString"
## attr(,"package")
```

```
## [1] "Biostrings"
```

```
str(ecoli)
```

```
## Formal class 'DNAString' [package "Biostrings"] with 5 slots
##   ..@ shared         :Formal class 'SharedRaw' [package "XVector"] with 2 slots
##   .. .. ..@ xp                    :<externalptr>
##   .. .. ..@ .link_to_cached_object:<environment: 0x7f8a052cf740>
##   ..@ offset         : int 0
##   ..@ length         : int 4686137
##   ..@ elementMetadata: NULL
##   ..@ metadata       : list()
```

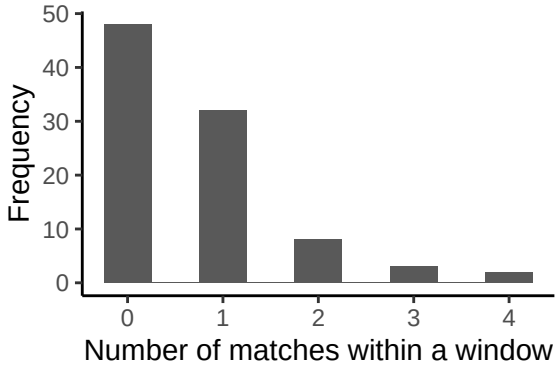
```
head(ecoli)
```

```
##   6-letter "DNAString" instance
## seq: AGCTTT
```

```
shineDalgarno <- "AGGAGGT"
window <- 50000

num_matches <- tibble(start = seq(from = 1, to = length(ecoli) - window, by = window)) %>%
  mutate(end = start + window - 1) %>%
  mutate(n_matches = map2_dbl(start, end, ~ countPattern(shineDalgarno,
                                                         ecoli[.x:.y],
                                                         max.mismatch = 0)))
head(num_matches)
```

```
## # A tibble: 6 x 3
##    start    end n_matches
##    <dbl>  <dbl>     <dbl>
## 1      1  50000         0
## 2  50001 100000         1
## 3 100001 150000         0
## 4 150001 200000         1
## 5 200001 250000         4
## 6 250001 300000         0
```

```
ggplot(num_matches, aes(x = n_matches)) +
  geom_histogram(binwidth = 0.5) +
  theme_classic() +
  labs(x = "Number of matches within a window",
       y = "Frequency")
```



As an alternative to using `table`, you can `group_by` and `count`:

```
num_matches %>%
  group_by(n_matches) %>%
  count()
```

```
## # A tibble: 5 x 2
## # Groups:   n_matches [5]
##   n_matches     n
##       <dbl> <int>
## 1         0    48
## 2         1    32
## 3         2     8
## 4         3     3
## 5         4     2
```

Here's another way of plotting the data that might help you look for patterns in where the matches to the motif are most common:

```
ggplot(num_matches, aes(x = start, y = 1)) +
  geom_tile(aes(fill = n_matches)) +
  theme_classic() +
  theme(legend.position = "top",
        axis.title.y = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank()) +
  scale_fill_viridis(name = "# of matches to Shine-\nDalgarno motif in the window") +
  scale_x_continuous(name = "Position of window start", labels = scales::comma)
```