STATISTICAL RETHINKING

SECOND EDITION

PRACTICE PROBLEM SOLUTIONS

Contents

Date: April 6, 2020.

2. Chapter 2 Solutions

2E1. Both (2) and (4) are correct. (2) is a direct interpretation, and (4) is equivalent.

2E2. Only (3) is correct.

2E3. Both (1) and (4) are correct. For (4), the product $\Pr(\text{rain}|\text{Monday})\Pr(\text{Monday})$ is just the joint probability of rain and Monday, $\Pr(\text{rain}, \text{Monday})$. Then dividing by the probability of rain provides the conditional probability.

2E4. This problem is merely a prompt for readers to explore intuitions about probability. The goal is to help understand statements like "the probability of water is 0.7" as statements about partial knowledge, not as statements about physical processes. The physics of the globe toss are deterministic, not "random." But we are substantially ignorant of those physics when we toss the globe. So when someone states that a process is "random," this can mean nothing more than ignorance of the details that would permit predicting the outcome.

As a consequence, probabilities change when our information (or a model's information) changes. Frequencies, in contrast, are facts about particular empirical contexts. They do not depend upon our information (although our beliefs about frequencies do).

This gives a new meaning to words like "randomization," because it makes clear that when we shuffle a deck of playing cards, what we have done is merely remove our knowledge of the card order. A card is "random" because we cannot guess it.

2M1. Since the prior is uniform, it can be omitted from the calculations. But I'll show it here, for conceptual completeness. To compute the grid approximate posterior distribution for (1):

R code 2.1

```
p_grid <- seq( from=0 , to=1 , length.out=100 )
# likelihood of 3 water in 3 tosses
likelihood <- dbinom( 3 , size=3 , prob=p_grid )
prior <- rep(1,100) # uniform prior
posterior <- likelihood * prior
posterior <- posterior / sum(posterior) # standardize
```

And `plot(posterior)` will produce a simple and ugly plot. This will produce something with nicer labels and a line instead of individual points:

R code 2.2

```
plot( posterior ~ p_grid , type="l" )
```

The other two data vectors are completed the same way, but with different likelihood calculations. For (2):

R code 2.3

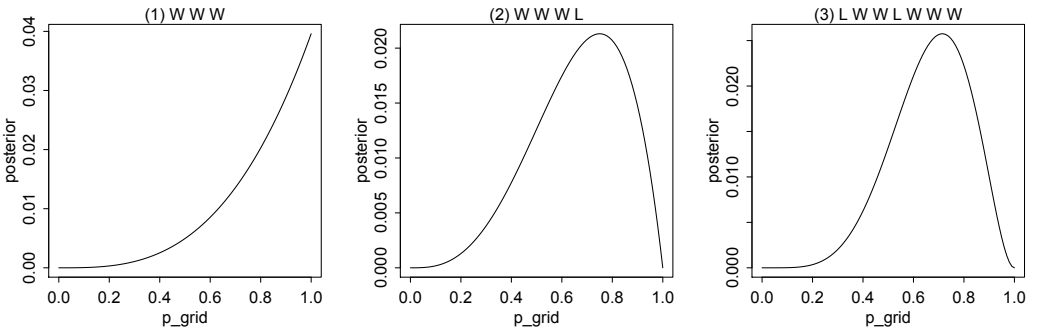
```
# likelihood of 3 water in 4 tosses
likelihood <- dbinom( 3 , size=4 , prob=p_grid )
```

And for (3):

R code 2.4

```
# likelihood of 5 water in 7 tosses
likelihood <- dbinom( 5 , size=7 , prob=p_grid )
```

And this is what each plot should look like:

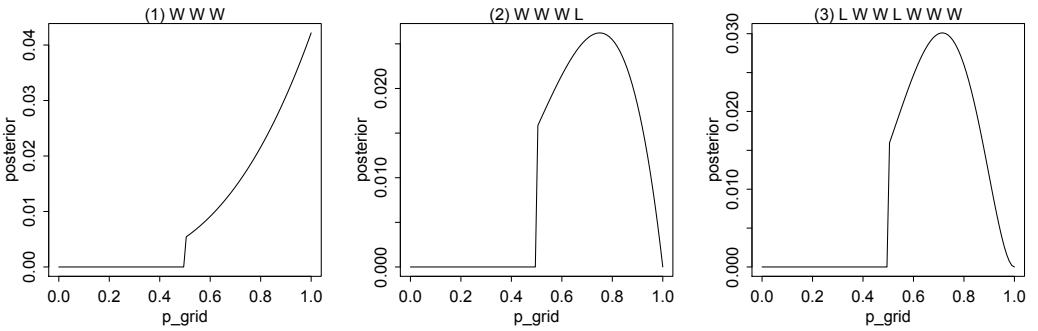


2M2. Only the prior has to be changed. For the first set of observations, W W W, this will complete the calculation and plot the result:

R code 2.5

```
p_grid <- seq( from=0 , to=1 , length.out=100 )
likelihood <- dbinom( 3 , size=3 , prob=p_grid )
prior <- ifelse( p_grid < 0.5 , 0 , 1 ) # new prior
posterior <- likelihood * prior
posterior <- posterior / sum(posterior) # standardize
plot( posterior ~ p_grid , type="l" )
```

The other two plots can be completed by changing the likelihood, just as in the previous problem. Here are the new plots, demonstrating that the prior merely truncates the posterior distribution below 0.5:



2M3. Here's what we know from the problem definition, restated as probabilities:

$$\Pr(\text{land}|\text{Earth}) = 1 - 0.7 = 0.3$$
$$\Pr(\text{land}|\text{Mars}) = 1$$

We also have, as stated in the problem, equal prior expectation of each globe. This means:

$$\Pr(\text{Earth}) = 0.5$$
$$\Pr(\text{Mars}) = 0.5$$

We want to calculate $\Pr(\text{Earth}|\text{land})$. By definition:

$$\Pr(\text{Earth}|\text{land}) = \frac{\Pr(\text{land}|\text{Earth})\Pr(\text{Earth})}{\Pr(\text{land})}$$

The $\Pr(\text{land})$ in the denominator is just the average probability of land, averaging over the two globes. So the above expands to:

$$\Pr(\text{Earth}|\text{land}) = \frac{\Pr(\text{land}|\text{Earth})\Pr(\text{Earth})}{\Pr(\text{land}|\text{Earth})\Pr(\text{Earth}) + \Pr(\text{land}|\text{Mars})\Pr(\text{Mars})}$$

Plugging in the numerical values:

$$\Pr(\text{Earth}|\text{land}) = \frac{(0.3)(0.5)}{(0.3)(0.5) + (1)(0.5)}$$

Let's compute the result using R:

R code 2.6

```
0.3*0.5 / ( 0.3*0.5 + 1*0.5 )
```

```
[1] 0.2307692
```

And there's the answer, $\Pr(\text{Earth}|\text{land}) \approx 0.23$. You can think of this posterior probability as an updated prior, of course. The prior probability was 0.5. Since there is more land coverage on Mars than on Earth, the posterior probability after observing land is smaller than the prior.

2M4. Label the three cards as (1) B/B, (2) B/W, and (3) W/W. Having observed a black (B) side face up on the table, the question is: How many ways could the other side also be black?

First, count up all the ways each card could produce the observed black side. The first card is B/B, and so there are 2 ways is could produce a black side face up on the table. The second card is B/W, so there is only 1 way it could show a black side up. The final card is W/W, so it has zero ways to produce a black side up.

Now in total, there are 3 ways to see a black side up. 2 of those ways come from the B/B card. The other comes from the B/W card. So 2 out of 3 ways are consistent with the other side of the card being black. The answer is 2/3.

2M5. With the extra B/B card, there are now 5 ways to see a black card face up: 2 from the first B/B card, 1 from the B/W card, and 2 more from the other B/B card. 4 of these ways are consistent with a B/B card, so the probability is now 4/5 that the other side of the card is also black.

2M6. This problem introduces uneven numbers of ways to draw each card from the bag. So while in the two previous problems we could treat each card as equally likely, prior to the observation, now we need to employ the prior odds explicitly.

There are still 2 ways for B/B to produce a black side up, 1 way for B/W, and zero ways for W/W. But now there is 1 way to get the B/B card, 2 ways to get the B/W card, and 3 ways to get the W/W card. So there are, in total, $1 \times 2 = 2$ ways for the B/B card to produce a black side up and $2 \times 1 = 2$ ways for the B/W card to produce a black side up. This means there are 4 ways total to see a black side up, and 2 of these are from the B/B card. 2/4 ways means probability 0.5.

2M7. The observation is now the sequence: black side up then white side up. We're still interested in the probability the other side of the first card is black. Let's take each possible card in turn.

First the B/B card. There are 2 ways for it to produce the first observation, the black side up. This leaves the B/W card and W/W card to produce the next observation. Each card is equally likely (has same number of ways to get drawn from the bag). But the B/W card has only 1 way to produce a white side up, while the W/W card has 2 ways. So 3 ways in total to get the second card to show white side up. All together, assuming the first card is B/B, there are $2 \times 3 = 6$ ways to see the BW sequence of sides up.

Now consider the B/W card being drawn first. There is 1 way for it to show black side up. This leaves the B/B and W/W cards to produce the second side up. B/B cannot show white up, so zero ways there. W/W has 2 ways to show white up. All together, that's $1 \times 2 = 2$ ways to see the sequence BW, when the first card is B/W.

The final card, W/W, cannot produce the sequence when drawn first. So zero ways.

Now let's bring it all together. Among all three cards, there are $6 + 2 = 8$ ways to produce the sequence BW. 6 of these are from the B/B being drawn first. So that's $6/8 = 0.75$ probability that the first card is B/B.

2H1. To solve this problem, realize first that it is asking for a conditional probability:

$$\Pr(\text{twins}_2|\text{twins}_1)$$

the probability the second birth is twins, conditional on the first birth being twins. Remember the definition of conditional probability:

$$\Pr(\text{twins}_2|\text{twins}_1) = \frac{\Pr(\text{twins}_1, \text{twins}_2)}{\Pr(\text{twins})}$$

So our job is to define $\Pr(\text{twins}_1, \text{twins}_2)$, the joint probability that both births are twins, and $\Pr(\text{twins})$, the unconditioned probability of twins.

$\Pr(\text{twins})$ is easier, so let's do that one first. The "unconditioned" probability just means that we have to average over the possibilities. In this case, that means the species have to averaged over. The problem implies that both species are equally common, so there's a half chance that any given panda is of either species. This gives us:

$$\Pr(\text{twins}) = \underbrace{\tfrac{1}{2}(0.1)}_{\text{Species A}} + \underbrace{\tfrac{1}{2}(0.2)}_{\text{Species B}}.$$

A little arithmetic tells us that $\Pr(\text{twins}) = 0.15$.

Now for $\Pr(\text{twins}_1, \text{twins}_2)$. The probability that a female from species A has two sets of twins is $0.1 \times 0.1 = 0.01$. The corresponding probability for species B is $0.2 \times 0.2 = 0.04$. Averaging over species identity:

$$\Pr(\text{twins}_1, \text{twins}_2) = \tfrac{1}{2}(0.01) + \tfrac{1}{2}(0.04) = 0.025.$$

Finally, we combine these probabilities to get the answer:

$$\Pr(\text{twins}_2|\text{twins}_1) = \frac{0.025}{0.15} = \frac{25}{150} = \frac{1}{6} \approx 0.17.$$

Note that this is higher than $\Pr(\text{twins})$. This is because the first set of twins provides some information about which species we have, and this information was automatically used in the calculation.

2H2. Our target now is $\Pr(\text{A}|\text{twins}_1)$, the posterior probability that the panda is species A, given that we observed twins. Bayes' theorem tells us:

$$\Pr(\text{A}|\text{twins}_1) = \frac{\Pr(\text{twins}_1|\text{A})\Pr(\text{A})}{\Pr(\text{twins})}.$$

We calculated $\Pr(\text{twins}) = 0.15$ in the previous problem, and we were given $\Pr(\text{twins}|\text{A}) = 0.1$ as well. The only stumbling block may be $\Pr(\text{A})$, the prior probability of species A. This was also given, implied by the equal abundance of both species. So prior to observing the birth, we have $\Pr(\text{A}) = 0.5$. So:

$$\Pr(\text{A}|\text{twins}_1) = \frac{(0.1)(0.5)}{0.15} = \frac{5}{15} = \frac{1}{3}.$$

So the posterior probability of species A, after observing twins, falls to 1/3, from a prior probability of 1/2. This also implies a posterior probability of 2/3 that our panda is species B, since we are assuming only two possible species. These are *small world* probabilities that trust the assumptions.

2H3. There are a few ways to arrive at the answer. The easiest is perhaps to recall that Bayes' theorem accumulates evidence, using Bayesian updating. So we can take the posterior probabilities from the previous problem and use them as prior probabilities in this problem. This implies $\Pr(\text{A}) = 1/3$. Now we can ignore the first observation, the twins, and concern ourselves with only the latest observation, the singleton birth. The previous observation is embodied in the prior, so there's no need to account for it again.

The formula is:

$$\Pr(\text{A}|\text{singleton}) = \frac{\Pr(\text{singleton}|\text{A})\Pr(\text{A})}{\Pr(\text{singleton})}$$

We already have the prior, $\Pr(\text{A})$. The other pieces are straightforward:

$$\begin{aligned}\Pr(\text{singleton}|\text{A}) &= 1 - 0.1 = 0.9\\ \Pr(\text{singleton}) &= \Pr(\text{singleton}|\text{A})\Pr(\text{A}) + \Pr(\text{singleton}|\text{B})\Pr(\text{B})\\ &= (0.9)\tfrac{1}{3} + (0.8)\tfrac{2}{3} = \frac{5}{6}\end{aligned}$$

Combining everything, we get:

$$\Pr(\text{A}|\text{singleton}) = \frac{(0.9)\frac{1}{3}}{\frac{5}{6}} = \frac{9}{25} = 0.36$$

This is a modest increase in posterior probability of species A, an increase from about 0.33 to 0.36.

The other way to proceed is to go back to the original prior, $\Pr(\text{A}) = 0.5$, before observed any births. Then you can treat both observations (twins, singleton) as data and update the original prior. I'm going to start abbreviating "twins" as T and "singleton" as S. The formula:

$$\Pr(\text{A}|\text{T, S}) = \frac{\Pr(\text{T, S}|\text{A})\Pr(\text{A})}{\Pr(\text{T, S})}$$

Let's start with the average likelihood, $\Pr(\text{T, S})$, because it will force us to define the likelihoods anyway.

$$\Pr(\text{T, S}) = \Pr(\text{T, S}|\text{A})\Pr(\text{A}) + \Pr(\text{T, S}|\text{B})\Pr(\text{B})$$

I'll go slowly through this, so I don't lose anyone along the way. The first likelihood is just the probability a species A mother has twins and then a singleton:

$$\Pr(\text{T, S}|\text{A}) = (0.1)(0.9) = 0.09$$

And the second likelihood is similar, but for species B:

$$\Pr(\text{T, S}|\text{B}) = (0.2)(0.8) = 0.16$$

The priors are both 0.5, so all together:

$$\Pr(\text{T, S}) = (0.09)(0.5) + (0.16)(0.5) = \frac{1}{8} = 0.125$$

We already have the likelihood needed for the numerator, so we can go to the final answer now:

$$\Pr(\text{A}|\text{T, S}) = \frac{(0.09)(0.5)}{0.125} = 0.36$$

Unsurprisingly, the same answer we got the other way.

2H4. First, ignoring the births. This is what we know about the test:

$$\Pr(\text{test A}|\text{A}) = 0.8$$
$$\Pr(\text{test A}|\text{B}) = 1 - 0.65 = 0.35$$

We use the original prior, $\Pr(\text{A}) = 0.5$. Plugging everything into Bayes' theorem:

$$\Pr(\text{A}|\text{test A}) = \frac{(0.8)(0.5)}{(0.8)(0.5) + (0.35)(0.5)} \approx 0.7$$

So the test has increased the confidence in species A from 0.5 to 0.7.

Now to use the birth information as well. The easiest way to do this is just to begin with the posterior from the previous problem. That posterior already used the birth data, so if we adopt it as our prior, we automatically use the birth data. And so our prior becomes $\Pr(\text{A}) = 0.36$. Then the approach is just the same as just above:

$$\Pr(\text{A}|\text{test A}) = \frac{\Pr(\text{test A}|\text{A})\Pr(\text{A})}{\Pr(\text{test A})}$$
$$\Pr(\text{A}|\text{test A}) = \frac{(0.8)(0.36)}{(0.36)(0.8) + (1 - 0.36)(0.35)} \approx 0.56$$

And since this posterior uses all of the data, the two births and the genetic test, it would be honest to label it as $\Pr(\text{A}|\text{test A, twins, singleton}) \approx 0.56$.

3. Chapter 3 Solutions

3E1. Let's begin by running the code that computes the samples (was given in the problem):

R code 3.1
```
p_grid <- seq( from=0 , to=1 , length.out=1000 )
prior <- rep( 1 , 1000 )
likelihood <- dbinom( 6 , size=9 , prob=p_grid )
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)
set.seed(100)
samples <- sample( p_grid , prob=posterior , size=1e4 , replace=TRUE )
```

Now to find out how much posterior probability lies below $p = 0.2$, just compute the proportion of samples that are below 0.2. I'll do this in two steps, to make the logic more transparent. First, count up how many samples are below 0.2:

R code 3.2
```
sum( samples < 0.2 )
```

```
5
```

Only 5 samples have a value less than 0.2. Now to compute the proportion, just divide by the number of samples:

R code 3.3
```
sum( samples < 0.2 ) / 1e4
```

```
[1] 5e-04
```

That's 5×10^{-4}. Not a lot of probability mass below 0.2.

3E2. A similar calculation as in **3E1**:

R code 3.4
```
sum( samples > 0.8 ) / 1e4
```

```
[1] 0.1117
```

The answer is that about 11% of the posterior probability lies above 0.8.

3E3. This one requires just a slightly more complicated condition inside the expression:

R code 3.5
```
sum( samples > 0.2 & samples < 0.8 ) / 1e4
```

```
[1] 0.8878
```

So about 89% of the posterior probability lies between 0.2 and 0.8.

3E4. This problem asks for an interval of defined mass, so we need to find the boundary. In the chapter, the `quantile` function was used for this purpose.

R code 3.6
```
quantile( samples , 0.2 )
```

```
      20%
0.5195195
```

So $p = 0.52$ (rounding for sanity) is the value that 20% of the posterior probability lies below. You can confirm this by going in the other direction:

```
sum( samples < 0.52  ) / 1e4
```

R code 3.7

```
[1] 0.201
```

3E5. A slight modification to the `quantile` code from **3E4** will do it. The trick here is to realize that finding the value of p above which 20% of the posterior probability lies means asking for the 80% quantile. Why? Because only 20% of the probability mass remains above 80%. Here's the code:

```
quantile( samples , 0.8 )
```

R code 3.8

```
      80%
0.7567568
```

Again, you can verify that this is correct by doing the calculation in reverse:

```
sum( samples > 0.75  ) / 1e4
```

R code 3.9

```
[1] 0.1905
```

Not exactly 0.2, but that's just because of the discrete nature of the grid we used, as well as the finite number of samples.

3E6. This problem is asking for a highest posterior density interval. You can compute this with the `HPDI` function:

```
HPDI( samples , prob=0.66 )
```

R code 3.10

```
    |0.66     0.66|
0.5205205 0.7847848
```

3E7. This problem is asking instead for a conventional percentile interval. `PI` will do the job:

```
PI( samples , prob=0.66 )
```

R code 3.11

```
      17%       83%
0.5005005 0.7687688
```

Note that this interval is, as expected, a little wider than the corresponding HPDI from the previous problem. If this isn't obvious at a glance, you can just compute the width of the intervals:

```
interval1 <- HPDI( samples , prob=0.66 )
interval2 <- PI( samples , prob=0.66 )
width1 <- interval1[2] - interval1[1]
width2 <- interval2[2] - interval2[1]
cbind(width1,width2)
```

R code 3.12

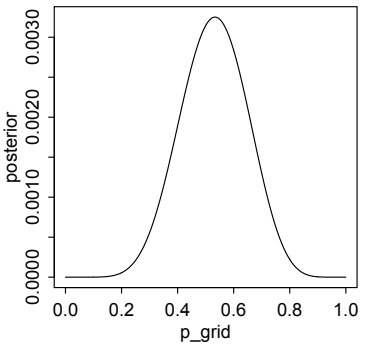
```
         width1    width2
0.66| 0.2642643 0.2682683
```

3M1. We can use the same code structure as before, but now with different data:

R code 3.13
```
p_grid <- seq( from=0 , to=1 , length.out=1000 )
prior <- rep( 1 , 1000 )
likelihood <- dbinom( 8 , size=15 , prob=p_grid )
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)
```

Note the 8 out of 15 in the `likelihood` calculation. When plotted, the posterior looks like this:

R code 3.14
```
plot( posterior ~ p_grid , type="l" )
```



3M2. To draw 10-thousand (`1e4`) samples:

R code 3.15
```
samples <- sample( p_grid , prob=posterior , size=1e4 , replace=TRUE )
```

And to compute the 90% HPDI:

R code 3.16
```
HPDI( samples , prob=0.9 )
```

```
     |0.9      0.9|
0.3383383 0.7317317
```

Your values will be slightly different, on account of sampling variation.

3M3. Following the example in the chapter, we just use `rbinom` to simulate samples, using samples from posterior distribution in place of the probability of water. This will do it:

R code 3.17
```
samples <- sample( p_grid , prob=posterior , size=1e4 , replace=TRUE )
w <- rbinom( 1e4 , size=15 , prob=samples )
```

And to compute the proportion of these simulated observations that match the data:

```
sum( w==8 ) / 1e4
```

R code 3.18

```
[1] 0.1473
```

What does this number mean? Not much. Any particular set of data can be unlikely, so discrete probabilities of data are hardly ever useful for summarize model fit. It's more common to summarize model fit using tail-area probabilities, for this reason. But those aren't always sensible either. In this case, it may be enough to confirm that the observed value 8 is right in the middle of the posterior predictive distribution. Plot it to confirm:

```
simplehist(w)
```

R code 3.19

This doesn't mean it is a good model. But it does mean that model fitting worked.

3M4. A minor change in the code is all that is needed:

```
w <- rbinom( 1e4 , size=9 , prob=samples )
sum( w==6 ) / 1e4
```

R code 3.20

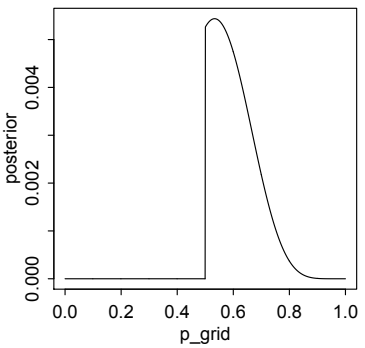
```
[1] 0.1802
```

Your answer will be slightly different, due to sampling variation.

3M5. Beginning again, but now with the new prior:

```
p_grid <- seq( from=0 , to=1 , length.out=1000 )
prior <- ifelse( p_grid < 0.5 , 0 , 1 )
likelihood <- dbinom( 8 , size=15 , prob=p_grid )
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)
samples <- sample( p_grid , prob=posterior , size=1e4 , replace=TRUE )
plot( posterior ~ p_grid , type="l" )
```

R code 3.21



The 90% HPDI will be a lot narrower now:

```
HPDI( samples , prob=0.9 )
```

R code 3.22

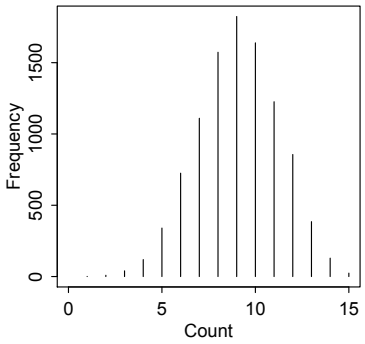
```
     |0.9      0.9|
0.5005005 0.7087087
```

It is narrower, just because the prior tells the model to ignore all values of p below 0.5. Prior information makes those values impossible causes of the data.

When re-simulating the posterior predictive distributions, note that the observed value of 8 water is no longer right in the center of the distribution. For example:

R code 3.23

```
w <- rbinom( 1e4 , size=15 , prob=samples )
simplehist(w)
```



The informative prior tells the model not to completely trust the data, so it shouldn't be surprising that the simulated posterior sampling distributions are not centered on the data. This is not an indication of anything wrong with the model. It's merely a consequence of the model.

When you reach multilevel models in the late chapters, you will have to expect mismatch between predictions and data in this way, because multilevel models often have informative priors. Informative priors are what make them good models.

3H1. First, define the parameter values to consider. As in the book, I'll use 1000 values of p here, but a value as low as 100 will do fine.

R code 3.24

```
p <- seq( from=0 , to=1 , length.out=1000 )
```

Now we need to define the uniform prior. An easy way to accomplish this is just to assign the same value to every model. There is no need to standardize the prior, because when you standardize the posterior, it will take care of it too.

R code 3.25

```
prior <- rep(1,length(p))
```

The value 1 in there is arbitrary. It could be anything, because only relative values matter, once the density is standardized.

We're ready to compute likelihoods, now. We want to compute the probability of the observed count of boy births, given all the probability values in `p`. So count up the boys and then compute the likelihoods:

R code 3.26

```
library(rethinking)
data(homeworkch3)
boys <- sum(birth1) + sum(birth2)
```

```
likelihood <- dbinom( boys , size=200 , prob=p )
```

Finally, the posterior is the standardized product of the prior and likelihoods, so we can compute it with:

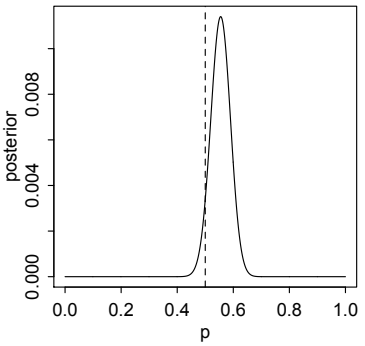
R code 3.27

```
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)
```

This is what your posterior distribution should look like:

R code 3.28

```
plot( posterior ~ p , type="l" )
abline( v=0.5 , lty=2 )
```



All that remains is to find the parameter value that maximizes posterior probability:

R code 3.29

```
p[ which.max(posterior) ]
```

```
[1] 0.5545546
```

3H2. You can just copy the code from the book, in order to draw samples from the posterior. Then it's just a matter of passing those samples to `HPDI`:

R code 3.30

```
p.samples <- sample( p , size=10000 , replace=TRUE , prob=posterior )
HPDI(p.samples,prob=0.50)
HPDI(p.samples,prob=0.89)
HPDI(p.samples,prob=0.97)
```

```
     |0.5      0.5|
0.5305305 0.5765766

    |0.89     0.89|
0.4964965 0.6076076

    |0.97     0.97|
0.4774775 0.6266266
```

Each of these intervals is the narrowest range of parameter values that contains the specified probability mass.

3H3. You can use the `rbinom` command to simulate binomial data, just like in the book. Here is code to regenerate the samples from the posterior and then use them to simulate 10-thousand samples of 200 predicted births:

R code 3.31

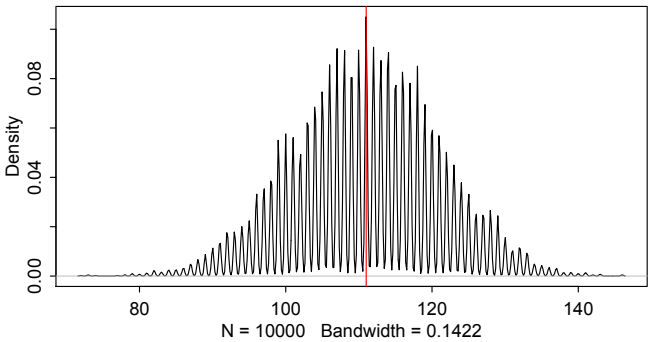
```
p.samples <- sample( p , size=10000 , replace=TRUE , prob=posterior )
bsim <- rbinom( 10000 , size=200 , prob=p.samples )
```

Now you want to compare the distribution of these predictions to the observed count of boys in all 200 observed births. Here is one way to do this, graphically:

R code 3.32

```
# adj value makes a strict histogram, with spikes at integers
dens( bsim , adj=0.1 )
abline( v=sum(birth1)+sum(birth2) , col="red" )
```

Here's what the plot looks like:



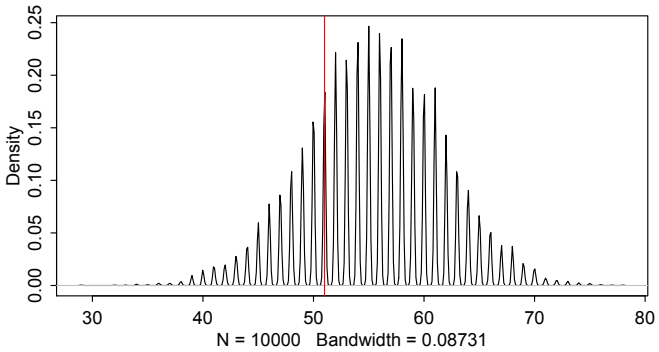
The black density is the simulated counts of male births, out of 200 maximum. The red vertical line is the observed count in the data. Looks like the model does a good job of predicting the data. In a sense, this isn't surprising, because the model was fit to this exact aspect of the data, the total count of boys.

3H4. The model can be threatened a little more by asking if it can predict other, subtler aspects of the data. How does it do predicting just first borns? The code looks very similar, but now there are only 100 births to predict. The posterior is the same as before.

R code 3.33

```
b1sim <- rbinom( 10000 , size=100 , prob=p.samples )
dens( b1sim , adj=0.1 )
abline( v=sum(birth1) , col="red" )
```

Here's what the plot looks like:



Not bad, but not right on center now. The frequency of boys among first borns is a little less than the model tends to predict. The model doesn't have a problem accounting for this variation though, as the red line is well within density of simulated data.

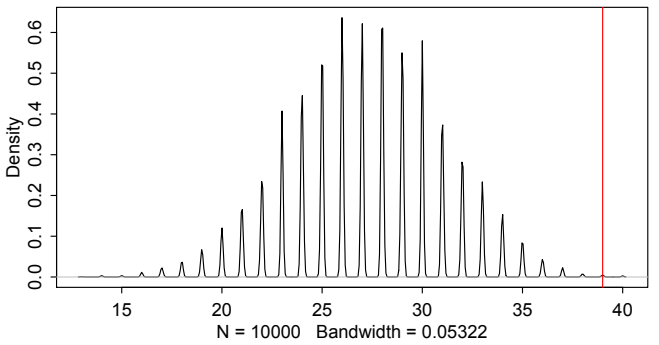
3H5. Now an even harder test of the model. How does it do predicting second births that follow girl births? The only tricky part of this is counting up boys born after girls. You can use R's subsetting to accomplish this:

R code 3.34

```
b01 <- birth2[birth1==0]
b01sim <- rbinom( 10000 , size=length(b01) , prob=p.samples )
dens(b01sim,adj=0.1)
abline( v=sum(b01) , col="red" )
```

The first line of code extracts those elements of `birth2` where `birth1` is equal to 0. So `b01` ends up holding all the second births that followed girls. The rest of the code is just as before, but now with a count of births to simulate equal to the length of `b01`.

Here's what the plot looks like:



That's pretty terrible. The observed number of boys who follow girls is far in excess of what the model predicts. Perhaps the first and second births are not independent?

4. Chapter 4 Solutions

4E1. The first line is the likelihood. The second line is very similar, but is instead the prior for the parameter μ. The third line is the prior for the parameter σ. Likelihoods and priors can look very similar, because a likelihood is effectively a prior for the residuals.

4E2. Two parameters in the posterior: μ and σ.

4E3. There are boxes in the chapter that provide examples. Here's the right form in this case, ignoring the specific distributions for the moment:

$$\Pr(\mu, \sigma|y) = \frac{\Pr(y|\mu, \sigma)\Pr(\mu)\Pr(\sigma)}{\int\int \Pr(y|\mu, \sigma)\Pr(\mu)\Pr(\sigma)d\mu d\sigma}$$

Now inserting the distributional assumptions:

$$\Pr(\mu, \sigma|y) = \frac{\text{Normal}(y|\mu, \sigma)\text{Normal}(\mu|0, 10)\text{Exponential}(\sigma|1)}{\int\int \text{Normal}(y|\mu, \sigma)\text{Normal}(\mu|0, 10)\text{Exponential}(\sigma|1)d\mu d\sigma}$$

4E4. The second line is the linear model.

4E5. There are 3 parameters in the posterior: α, β, and σ. The symbol μ is no longer a parameter in the posterior, because it is entirely determined by α, β, and x.

4M1. To sample from the prior distribution, we use `rnorm` to simulate, while averaging over the prior distributions of μ and σ. The easiest way to do this is to sample from the priors and then pass those samples to `rnorm` to simulate observations. This code will sample from the priors:

R code 4.1

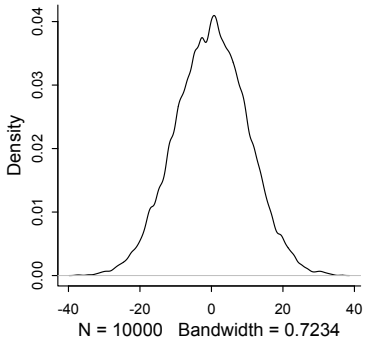
```
mu_prior <- rnorm( 1e4 , 0 , 10 )
sigma_prior <- rexp( 1e4 , 1 )
```

You may want to visualize these samples with `dens`, just to help school your intuition for the priors.

Now to simulate observations that average over these prior distributions of parameters:

R code 4.2

```
h_sim <- rnorm( 1e4 , mu_prior , sigma_prior )
dens( h_sim )
```



4M2. As a quap formula, the model in **4M1** is:

R code 4.3

```
f <- alist(
    y ~ dnorm( mu , sigma ),
    mu ~ dnorm( 0 , 10 ),
    sigma ~ dexp( 1 )
)
```

4M3. This is straightforward, but remember that mathematical notation makes use of index variables like *i*, while the R code is instead implicitly vectorized over observations.

$$
\begin{aligned}
y_i &\sim \text{Normal}(\mu, \sigma) \\
\mu &= \alpha + \beta x_i \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

4M4. This is an more open-ended problem than the others. But perhaps the simplest model structure that addresses the prompt would be:

$$
\begin{aligned}
h_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma) \\
\mu_{ij} &= \alpha + \beta(y_j - \bar{y}) \\
\alpha &\sim \text{Normal}(100, 10) \\
\beta &\sim \text{Normal}(0, 10) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

where h is height and y is year and $\bar{y}$ the average year in the sample. The index i indicates the student and the index j indicates the year.

The problem didn't say how old the students are, so you'll have to decide for yourself. The priors above assume the students are still growing, so the mean height α is set around 100 cm. The slope with year β is vague here—we'll do better in the next problem. For σ, this needs to express how variable students are in the same year

Let's simulate from these priors now and see what the model expects, before it sees the data. Begin by sampling from the priors.

R code 4.4

```
n <- 50
a <- rnorm( n , 100 , 10 )
b <- rnorm( n , 0 , 10 )
s <- rexp( n , 1 )
```

Now let's consider 50 students, each simulated with a different prior draw. For each student, we'll simulate three years.

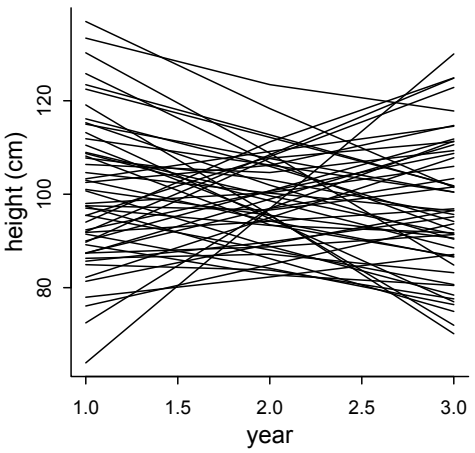
R code 4.5

```
y <- 1:3
ybar <- mean(y)
# matrix of heights, students in rows and years in columns
h <- matrix( NA , nrow=n , ncol=3 )
for ( i in 1:n ) for ( j in 1:3 )
    h[ i , j ] <- rnorm( 1 , a[i] + b[i]*( y[j] - ybar ) , s[i] )
```

Now let's plot them:

R code 4.6

```
plot( NULL , xlim=c(1,3) , ylim=range(h) , xlab="year" , ylab="height (cm)" )
for ( i in 1:n ) lines( 1:3 , h[i,] )
```



This is a rather ignorant prior, but at least the trends are within possibility. The biggest flaw is that many of these children are shrinking. That seems unlikely. Others are growing much too fast. We can address these issues in the next problem.

4M5. A simple way to force individuals to get taller from one year to the next is to constrain the slope β to be positive. A log-normal distribution makes this rather easy, but you do have to be careful in choosing its parameter values. Recall that the mean of a log-normal is $\exp(\mu + \sigma^2/2)$. Let's try $\mu = 1$

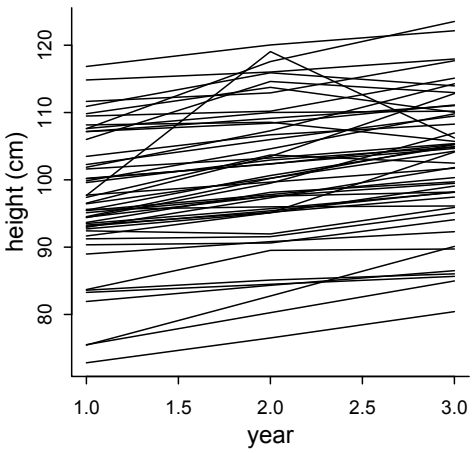
and $\sigma = 0.5$. This will give a mean $\exp(1 + 0.25/2) \approx 3$, or 3 cm of growth per year.

$$
\begin{aligned}
h_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma) \\
\mu_{ij} &= \alpha + \beta(y_j - \bar{y}) \\
\alpha &\sim \text{Normal}(100, 10) \\
\beta &\sim \text{Log-Normal}(1, 0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

Again simulate from the priors and plot:

R code 4.7

```
n <- 50
a <- rnorm( n , 100 , 10 )
b <- rlnorm( n , 1 , 0.5 )
s <- rexp( n , 1 )

y <- 1:3
ybar <- mean(y)
# matrix of heights, students in rows and years in columns
h <- matrix( NA , nrow=n , ncol=3 )
for ( i in 1:n ) for ( j in 1:3 )
    h[ i , j ] <- rnorm( 1 , a[i] + b[i]*( y[j] - ybar ) , s[i] )

plot( NULL , xlim=c(1,3) , ylim=range(h) , xlab="year" , ylab="height (cm)" )
for ( i in 1:n ) lines( 1:3 , h[i,] )
```



Aside from a few people, the lines are slope upwards. Why do some of them zig-zag? Because the variation around the expectation from σ allows it. If you think of this as measurement error, it's not necessarily bad. If measurement error is small however, you'd have to think harder. Much later in the book, you'll learn some tools to help with this kind of problem.

4M6. The simplest way to encode the information is to say:

$$\sigma \sim \text{Uniform}(0, 8)$$

Where $8 = \sqrt{64}$ because σ is the square-root of the variance. Okay, easy enough.

4M7. Once we take away the mean weight, the intercept α will no longer be the mean height. Instead it will be the mean height when weight is zero (which never happens). Let's see what happens, if we use the original priors:

R code 4.8
```
library(rethinking)
data(Howell1); d <- Howell1; d2 <- d[ d$age >= 18 , ]
m4.3b <- quap(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight ,
        a ~ dnorm( 178 , 20 ) ,
        b ~ dlnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=d2 )
precis( m4.3b )
```

```
        mean   sd   5.5%  94.5%
a     114.53 1.90 111.50 117.57
b       0.89 0.04   0.82   0.96
sigma   5.07 0.19   4.77   5.38
```

The original model has:

R code 4.9
```
precis( m4.3 )
```

```
        mean   sd   5.5%  94.5%
a     154.60 0.27 154.17 155.03
b       0.90 0.04   0.84   0.97
sigma   5.07 0.19   4.77   5.38
```

So the slope is the same, but the intercept has changed. We should expect this. You might want to confirm that the posterior predictions make sense, that the model fit correctly. When you do, you'll see that this model makes the same predictions as the original one.

The problem asked you to inspect the covariance among parameters. Let's do that:

R code 4.10
```
round( vcov( m4.3b ) , 2 )
```

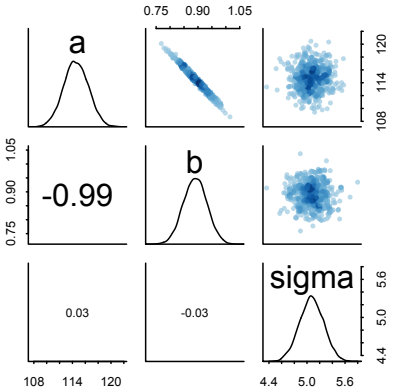
```
          a     b sigma
a      3.60 -0.08  0.01
b     -0.08  0.00  0.00
sigma  0.01  0.00  0.04
```

If you look at the same output for `m4.3`, you'll see that there is no covariance like there is above between `a` and `b`. That -0.08 covariance seems pretty small. But remember that this is on the scale of the variables. Let's convert to a correlation matrix instead:

R code 4.11
```
round( cov2cor(vcov( m4.3b )) , 2 )
```

```
          a     b sigma
a      1.00 -0.99  0.03
b     -0.99  1.00 -0.03
sigma  0.03 -0.03  1.00
```

The intercept `a` and slope `b` are almost perfectly negatively correlated. You can see this in the `pairs(m4.3b)` output:



We have the conclusion that while these models make the same posterior predictions, but the parameters have quite different meanings and relationships with one another. There is nothing wrong with this new version of the model. But usually it is much easier to set priors, when we center the predictor variables. But you can always use prior simulations to set sensible priors, when in doubt.

4M8. As the number of knots increases, the spline will wiggly more, but only up to a point. The width on the prior weights constrains how wiggly the spline can become. If you make the prior very wide, then then spline can wiggly a lot, if it has enough knots. The combination of number of knots and the width of the prior weights determines how tightly the spline can match the sample. In a later chapter, you'll see that having a perfect match to the sample is not really a good idea.

4H1. First, fit the model with `quap`, to provide a quadratic approximation of the posterior. This is just as in the chapter.

R code 4.12

```
library(rethinking)
data(Howell1)
d <- Howell1
d2 <- d[d$age>=18,]
xbar <- mean( d2$weight )

m <- quap(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*( weight  - xbar ) ,
        a ~ dnorm( 178 , 20 ) ,
        b ~ dlnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=d2 )
```

Then produce samples from the quadratic approximate posterior. You could use `mvrnorm` directly, or the convenient `extract.samples` function:

R code 4.13

```
post <- extract.samples( m )
str(post)
```

```
'data.frame': 10000 obs. of  3 variables:
 $ a    : num  155 155 155 155 155 ...
 $ b    : num  0.992 0.903 0.92 0.835 0.971 ...
 $ sigma: num  5.25 4.98 4.86 5.23 5.01 ...
 - attr(*, "source")= chr "quap posterior: 10000 samples from m"
```

Now we want to plug the weights in the table into this model, and then average over the posterior to compute predictions for each individual's height. The question is ambiguous as to whether it wants μ only or rather the distribution of individual height measurements (using σ). I'll show the harder approach, that uses σ and simulates individual heights. Also, I won't use `link` or `sim`, but it's fine if you did.

For the first individual, the code might look like this:

R code 4.14

```
y <- rnorm( 1e5 , post$a + post$b*( 46.95 - xbar ) , post$sigma )
mean(y)
PI(y,prob=0.89)
```

```
[1] 156.364

      5%      94%
148.2133 164.4904
```

How does the code work? The first line, which includes `rnorm`, simulates 100-thousand heights, using the samples from the posterior and an assumed weight of 46.95 kg. The second line then computes the average of these simulated heights. That gives the expected (mean) height. The third line then computes the 89% compatibility interval of height.

To do the above for each row in the table, you can just replace the weight each time. I'm going to write a function and use `sapply`:

R code 4.15

```
f <- function( weight ) {
    y <- rnorm( 1e5 , post$a + post$b*( weight - xbar ) , post$sigma )
    return( c( mean(y) , PI(y,prob=0.89) ) )
}
weight_list <- c(46.95,43.72,64.78,32.59,54.63)
result <- sapply( weight_list , f )
```

Now let's format into a results table:

R code 4.16

```
rtab <- cbind( weight_list , t( result ) )
colnames(rtab) <- c("weight","height","5%","94%")
rtab
```

```
     weight   height       5%      94%
[1,]  46.95 156.3754 148.2310 164.5441
[2,]  43.72 153.4612 145.3171 161.6234
[3,]  64.78 172.4761 164.1917 180.7644
[4,]  32.59 143.3843 135.2444 151.5570
[5,]  54.63 163.2732 155.1527 171.4127
```

You could have also computed the expected heights straight from the MAP. That approach is fine, and will give nearly the same answer.

4H2. (a) First make a new data frame with just the non-adults in it:

R code 4.17

```
library(rethinking)
data(Howell1)
d <- Howell1
d3 <- d[ d$age < 18 , ]
str(d3)
```

```
'data.frame': 192 obs. of  4 variables:
 $ height: num  121.9 105.4 86.4 129.5 109.2 ...
 $ weight: num  19.6 13.9 10.5 23.6 16 ...
 $ age   : num  12 8 6.5 13 7 17 16 11 17 8 ...
 $ male  : int  1 0 0 1 0 1 0 1 0 1 ...
```

Now find the quadratic approximate posterior. The code is the same as before. The data frame just changes.

R code 4.18

```
xbar <- mean( d3$weight )
m <- quap(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*( weight - xbar ) ,
        a ~ dnorm( 178 , 20 ),
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=d3 )
precis(m)
```

```
        mean   sd   5.5%  94.5%
a     108.38 0.61 107.41 109.36
b       2.72 0.07   2.61   2.83
sigma   8.44 0.43   7.75   9.13
```

The estimates suggest that the MAP coefficient for weight is 2.7. This implies that for a unit change of 1kg of weight, we predict an average of 2.7cm of increase in height.

(b) Now to plot the raw data and superimpose the model estimates, modify the code in Chapter 4. We will sample from the naive posterior, then compute 89% intervals for the mean and predicted heights. This is what the complete code looks like, if you opt not to use the convenience functions `link` and `sim`:

R code 4.19

```
post <- extract.samples( m )
w.seq <- seq(from=1,to=45,length.out=50)
mu <- sapply( w.seq , function(z) mean( post$a + post$b*(z-xbar) ) )
mu.ci <- sapply( w.seq , function(z)
    PI( post$a + post$b*(z-xbar) , prob=0.89 ) )
pred.ci <- sapply( w.seq , function(z)
    PI( rnorm( 10000 , post$a + post$b*(z-xbar) , post$sigma) , 0.89 ) )
```

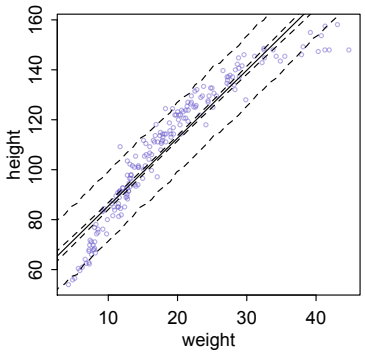
And then to plot everything:

R code 4.20

```
plot( height ~ weight , data=d3 ,
    col=col.alpha("slateblue",0.5) , cex=0.5 )
lines( w.seq , mu )
lines( w.seq , mu.ci[1,] , lty=2 )
```

```
lines( w.seq , mu.ci[2,] , lty=2 )
lines( w.seq , pred.ci[1,] , lty=2 )
lines( w.seq , pred.ci[2,] , lty=2 )
```

And the resulting plot looks like:



(c) The major problem with this model appears to be that the relationship between weight and height, for non-adults, isn't very linear. Instead it is curved. As a result, at low weight values, the predicted mean is above most of the actual heights. At middle weight values, the predicted mean is below most of the heights. Then again at high weight values, the mean is above the heights.

A parabolic model would likely fit these data much better. But that's not the only option. What we're after essentially is some way to model a reduction of the slope between height and weight, as weight increases. And onwards to the next solution, which does that, for the entire data.

4H3. (a) You can just use `log` inside the call to `quap`:

R code 4.21

```
library(rethinking)
data(Howell1)
d <- Howell1
logxbar <- mean( log(d$weight) )
mlw <- quap(
    alist(
        height ~ dnorm( mean=mu , sd=sigma ) ,
        mu <- a + b*( log(weight) - logxbar ) ,
        a ~ dnorm( 178 , 20 ) ,
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=d )
precis(mlw)
```

```
        mean   sd   5.5%  94.5%
a     138.27 0.22 137.92 138.62
b      47.02 0.38  46.41  47.63
sigma   5.13 0.16   4.89   5.38
```

Pretty hard to know what to make of these estimates, aside from the fact that the confidence intervals are quite narrow, owing to there being 544 rows. The estimate for `b` (β) is hard to understand, because it refers to log-kg, not raw kg. It means that for every increase of 1 log-kg of weight, you expect a

increase of 47 cm of height. But what's a log-kg? You want to know what the model predicts on the natural scale of measurement. So now to plotting...

(b) Begin by sampling from the naive posterior and computing the confidence intervals as per the examples in the book. Again, I'll not use the convenience functions, but it's fine if you did.

R code 4.22

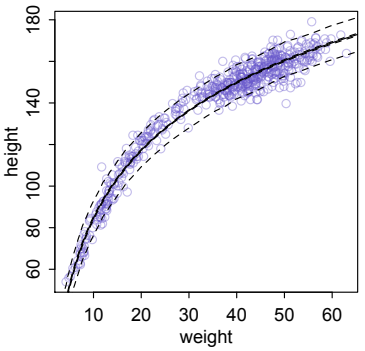
```
post <- extract.samples(mlw)
w.seq <- seq(from=4,to=63,length.out=50)
mu <- sapply( w.seq , function(z) mean( post$a+post$b*(log(z)-logxbar) ) )
mu.ci <- sapply( w.seq , function(z) PI( post$a+post$b*(log(z)-logxbar) ) )
h.ci <- sapply( w.seq , function(z)
    PI( rnorm(10000,post$a+post$b*(log(z)-logxbar),post$sigma) ) )
```

Now you have lists of numbers that can be plotted to produce the confidence curves.

R code 4.23

```
plot( height ~ weight , data=d , col=col.alpha("slateblue",0.4) )
lines( w.seq , mu )
lines( w.seq , mu.ci[1,] , lty=2 )
lines( w.seq , mu.ci[2,] , lty=2 )
lines( w.seq , h.ci[1,] , lty=2 )
lines( w.seq , h.ci[2,] , lty=2 )
```

This code yields:



The model may have been linear, but plotted on the raw scale of measurement, it is clearly non-linear. Not only is the trend for the mean curved, but the variance around the mean is not constant, on this scale. Instead, the variance around the mean increases with weight. On the scale you fit the model on, the variance was assumed to be constant. But once you transform the measurement scale, it usually won't be.

Notice also that the estimate for the mean is so precise that you can hardly even see the confidence interval for it. Don't get too confident about such results, though. Remember, all inferences of the model are conditional on the model. Even estimated trends that do a terrible job of prediction can have tight confidence intervals, when the data set is large.

4H4. Here is the model, just copied from the chapter:

R code 4.24

```
library(rethinking)
data(Howell1)
d <- Howell1
```

```
d$weight_s <- ( d$weight - mean(d$weight) )/sd(d$weight)
d$weight_s2 <- d$weight_s^2
m4.5 <- quap(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b1*weight_s + b2*weight_s2 ,
        a ~ dnorm( 178 , 20 ) ,
        b1 ~ dlnorm( 0 , 1 ) ,
        b2 ~ dnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=d )
```

Let's extract the prior:

R code 4.25

```
set.seed(45)
prior <- extract.prior( m4.5 )
precis( prior )
```

```
'data.frame': 1000 obs. of 4 variables:
        mean    sd   5.5%  94.5%     histogram
a     177.61 20.72 144.18 211.42
b1      1.61  1.88   0.19   4.43
b2     -0.05  0.97  -1.64   1.45
sigma  25.14 14.59   2.52  47.37
```

We want to simulate curves (parabolas) from this prior. One way is to use `link`. Then we won't have to write the linear model again.

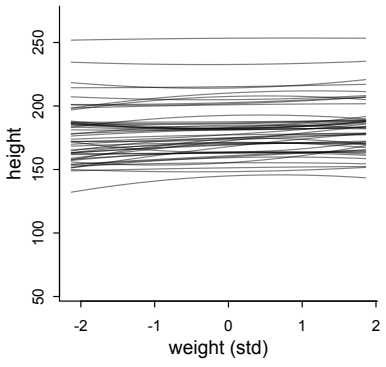
R code 4.26

```
w_seq <- seq( from=min(d$weight_s) , to=max(d$weight_s) ,
    length.out=50 )
w2_seq <- w_seq^2
mu <- link( m4.5 , post=prior ,
    data=list( weight_s=w_seq , weight_s2=w2_seq ) )
```

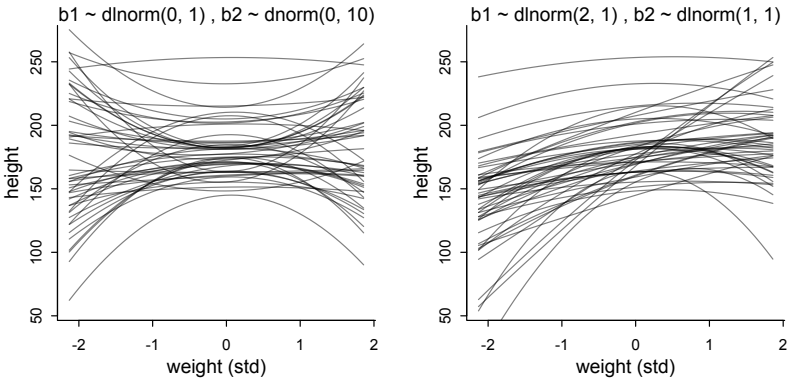
Now `mu` should contain 1000 parabolas. We'll plot just the first 50.

R code 4.27

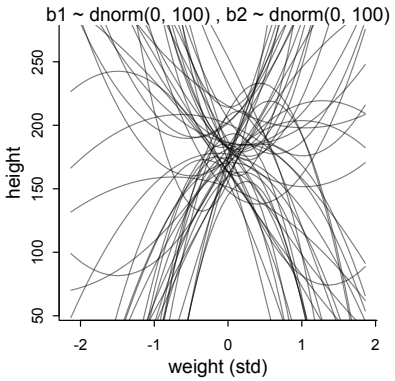
```
plot( NULL , xlim=range(w_seq) , ylim=c(55,270) ,
    xlab="weight (std)" , ylab="height" )
for ( i in 1:50 ) lines( w_seq , mu[i,] , col=col.alpha("black",0.5) )
```



Recall that the world's tallest person was 270cm tall. The tallest person in the sample is about 180cm. The prior curvature is not very strong. Those parabolas hardly bend at all. We can increase the standard deviation on the `b2` prior, but that will produce some silly shapes (left below), where either average weight is tallest or shortest. That can't be right. The basic problem is that `b2` needs to be negative to make the curve bend down, but `b1` has to also change in order to move the maximum height to the right. It's all a bit confusing, and is they key reason that working with polynomial models is so hard. The prior on the right below can only bend down, but I've made the linear model `a + b1*weight_s - b2*weight_s2` and given `b2` a log-Normal prior.



A key problem in getting reasonable curves here is that obviously `a` and `b1` and `b2` are correlated in the family of reasonable curves. But the priors are uncorrelated—they are independent of one another. Still, if you can get independent priors to at least live within some reasonably space of outcome values, that's a lot better than flat priors. What would flat priors look like here? Something like this:



These prior curves actually strongly favor explosive growth or shrinkage near the mean. This is a general phenomenon with "flat" priors: Once the predictor is at all complicated, "flat" does not imply "no relationship."

Do any of the priors above make a difference for inference in this sample? No. There is a lot of data and the model is quite simple, in terms of the way that parameters relate to predictions. This will not always be the case.

4H5. Load the data and look at the missing values:

R code 4.28

```
library(rethinking)
data(cherry_blossoms)
```

```
colSums( is.na(cherry_blossoms) )
```

```
      year        doy       temp temp_upper temp_lower
         0        388         91         91         91
```

So we need to select out complete cases for `doy` and `temp`.

R code 4.29

```
d <- cherry_blossoms
d2 <- d[ complete.cases( d$doy , d$temp ) , c("doy","temp") ]
```

You should be left with 787 rows.

If you just `plot(d2)` you'll see there is surely some relationship. But it's pretty noisy. I'm going to go ahead and build a spline here, for the sake of the example. But a linear fit wouldn't be awful, even though at the extremes this relationship cannot possibly be linear.

First build the knot list. We'll build the knots on the temperature record, thinking of `doy` as the outcome variable.

R code 4.30

```
num_knots <- 30
knot_list <- quantile( d2$temp , probs=seq(0,1,length.out=num_knots) )
library(splines)
B <- bs(d2$temp,
    knots=knot_list[-c(1,num_knots)] ,
    degree=3 , intercept=TRUE )
```

And now the model:

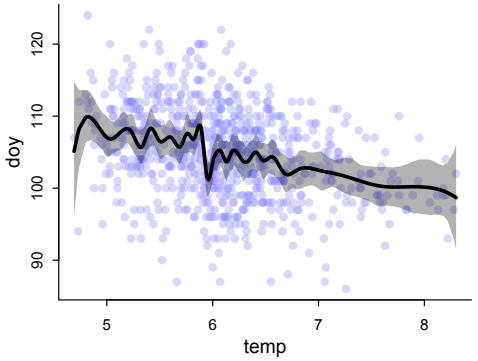
R code 4.31

```
m4H5 <- quap(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- a + B %*% w ,
        a ~ dnorm(100,10),
        w ~ dnorm(0,10),
        sigma ~ dexp(1)
    ), data=list( D=d2$doy , B=B ) ,
    start=list( w=rep( 0 , ncol(B) ) ) )
```

You can inspect the `precis` output, if you like. The weights aren't going to be meaningful to you. Let's plot. The only trick here is to get the order of the temperature values right when we plot, since they are not ordered in the data or in the basis functions. We can do this with `order` to get the index values for the proper order and then index everything else by this:

R code 4.32

```
mu <- link( m4H5 )
mu_mean <- apply( mu , 2 , mean )
mu_PI <- apply( mu , 2 , PI, 0.97 )
plot( d2$temp , d2$doy , col=col.alpha(rangi2,0.3) , pch=16 ,
     xlab="temp" , ylab="doy" )
o <- order( d2$temp )
lines( d2$temp[o] , mu_mean[o] , lwd=3 )
shade( mu_PI[,o] , d2$temp[o] , col=grau(0.3) )
```



There is a silly amount of wiggle in this spline. I used 30 knots and quite loose prior weights, so this wiggle isn't unexpected. It also probably isn't telling us anything causal. Overall the trend is quite linear, aside from the odd drop just before 6 degrees. This could be real, or it could be an artifact of changes in the record keeping. The colder dates are also older and the temperatures for older dates were estimated differently.

4H6. The code from the chapter is enough to define the model again:

R code 4.33

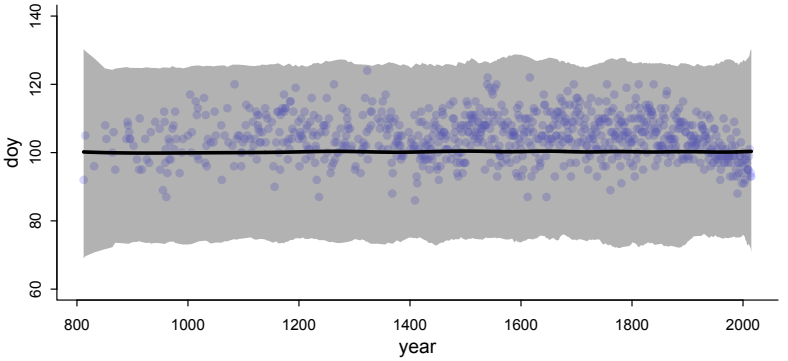
```
library(rethinking)
data(cherry_blossoms)
d <- cherry_blossoms
d2 <- d[ complete.cases(d$doy) , ] # complete cases on doy
num_knots <- 15
knot_list <- quantile( d2$year , probs=seq(0,1,length.out=num_knots) )
library(splines)
B <- bs(d2$year,
    knots=knot_list[-c(1,num_knots)] ,
    degree=3 , intercept=TRUE )

m4.7 <- quap(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- a + B %*% w ,
        a ~ dnorm(100,10),
        w ~ dnorm(0,10),
        sigma ~ dexp(1)
    ), data=list( D=d2$doy , B=B ) ,
    start=list( w=rep( 0 , ncol(B) ) ) )
```

Now let's extract the prior and use it to simulate observations:

R code 4.34

```
p <- extract.prior( m4.7 )
mu <- link( m4.7 , post=p )
mu_mean <- apply( mu , 2 , mean )
mu_PI <- apply( mu , 2 , PI, 0.97 )
plot( d2$year , d2$doy , col=col.alpha(rangi2,0.3) , pch=16 ,
     xlab="year" , ylab="doy" , ylim=c(60,140) )
lines( d2$year , mu_mean , lwd=3 )
```

```
shade( mu_PI , d2$year , col=grau(0.3) )
```

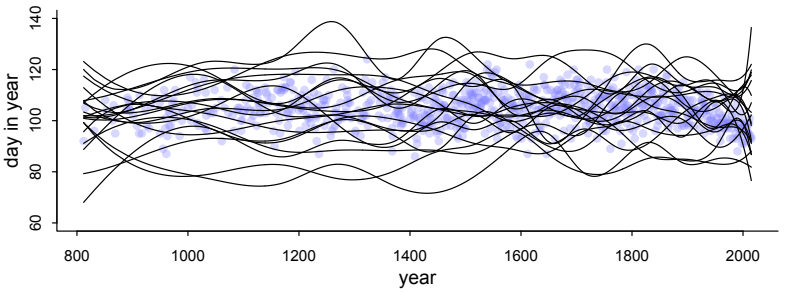


These priors imply no specific trend and very wide range of possibilities inside the gray region, including unrealistically early blossom dates in the low range and unrealistically late dates in the high range.

This is more satisfying if instead of a compatibility region we plot individual splines sampled from the prior. Here are 20 splines sampled from the prior:

R code 4.35

```
p <- extract.prior( m4.7 )
mu <- link( m4.7 , post=p )
plot( d2$year , d2$doy , col=col.alpha(rangi2,0.3) , pch=16 ,
     xlab="year" , ylab="day in year" , ylim=c(60,140) )
for ( i in 1:20 ) lines( d2$year , mu[i,] , lwd=1 )
```

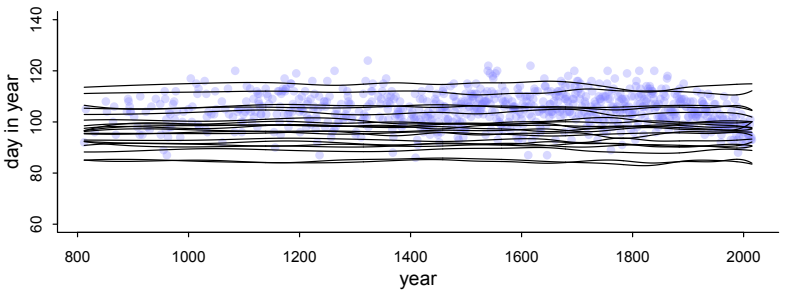


What happens when we tighten the priors?

R code 4.36

```
m4.7b <- quap(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- a + B %*% w ,
        a ~ dnorm(100,10),
        w ~ dnorm(0,1),
        sigma ~ dexp(1)
    ), data=list( D=d2$doy , B=B ) ,
    start=list( w=rep( 0 , ncol(B) ) ) )

p <- extract.prior( m4.7b )
```

```
mu <- link( m4.7b , post=p )
plot( d2$year , d2$doy , col=col.alpha(rangi2,0.3) , pch=16 ,
     xlab="year" , ylab="day in year" , ylim=c(60,140) )
for ( i in 1:20 ) lines( d2$year , mu[i,] , lwd=1 )
```



Perhaps obvious in hindsight, the gray region gets tighter.

The prior weights control how flexible the spline is in changing the local expectation. If you make the prior flat, then the spline is free to wiggle. The tighter the prior, the more evidence is needed to induce local wiggle.

4H7. It is easy enough to just remove the intercept a from the model. But if that's all we do, the prior won't be anywhere near the data, since the prior weights are offsets from the intercept. So we need to either center the weights someplace else or just put a fixed intercept in the linear model. I'll do the first:

R code 4.37

```
library(rethinking)
data(cherry_blossoms)
d <- cherry_blossoms
d2 <- d[ complete.cases(d$doy) , ] # complete cases on doy
num_knots <- 15
knot_list <- quantile( d2$year , probs=seq(0,1,length.out=num_knots) )
library(splines)
B <- bs(d2$year,
    knots=knot_list[-c(1,num_knots)] ,
    degree=3 , intercept=TRUE )

m4H7 <- quap(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- 0 + B %*% w ,
        w ~ dnorm(100,10),
        sigma ~ dexp(1)
    ), data=list( D=d2$doy , B=B ) ,
    start=list( w=rep( 0 , ncol(B) ) ) )
```

Note the 100 inside the prior weights. If you plot the posterior predictions, you'll see they are essentially the same as before. If you don't include the 100, you'll see that the ends of the spline drift downwards, towards the prior. But there is so much data here that the core of the spline should look very similar. the prior predictions however would be absolutely bonkers.

5. Chapter 5 Solutions

5E1. Only (2) and (4) are multiple linear regressions. Both have more than one predictor variable and corresponding coefficients in the linear model. The model (1) has only a single predictor variable, x. The model (3) has two predictor variables, but only their difference for each case enters the model, so effectively this is a uni-variate regression, with a single slope parameter.

5E2. A verbal model statement like this will always be somewhat ambiguous. That is why mathematical notation is needed in scientific communication. However, the conventional interpretation of the statement would be:

$$A_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta_L L_i + \beta_P P_i$$

where A is animal diversity, L is latitude, and P is plant diversity. This linear model "controls" for plant diversity, while estimating a linear relationship between latitude and animal diversity.

5E3. Define T as time to PhD degree, the outcome variable implied by the problem. Define F as amount of funding and S as size of laboratory, the implied predictor variables. Then the model (ignoring priors) might be:

$$T_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta_F F_i + \beta_S S_i$$

The slopes β_F and β_S should both be positive.

How can both be positively associated with the outcome in a multiple regression, but neither by itself? If they are negatively correlated with one another, then considering each alone may miss the positive relationships with the outcome. For example, large labs have less funding per student. Small labs have more funding per student, but poorer intellectual environments. So both could be positive influences on time to degree, but be negatively associated in nature.

5E4. This question is tricky. First, the answer will actually depend upon the priors, which aren't mentioned in the problem. But assuming weakly informative or flat priors, the answer is that (1), (3), (4), and (5) are inferentially equivalent. They'll make the same predictions, and you can convert among them after model fitting. (2) stands out because it has a redundant parameter, the intercept α.

5M1. The are many good answers to this question. The easiest approach is to think of some context that follows the divorce rate pattern in the chapter: one predictor influences both the outcome and the other predictor. For example, we might consider predicting whether or not a scientific study replicates. Two predictor variables are available: (1) sample size and (2) statistical significance. Sample size influences both statistical significance and reliability of a finding. This induces a correlation between significance and successful replication, even though significance is not associated with replication, once sample size is taken into account.

5M2. Again, many good answers are possible. The pattern from the milk energy example in the chapter is the simplest. Consider for example the influences of income and drug use on health. Income is positively associated, in reality, with health. Drug use is, for the sake of the example, negatively associated with health. But wealthy people consume more drugs than the poor, simply because the wealthy can afford them. So income and drug use are positively associated in the population. If this positive association is strong enough, examining either income or drug use alone will show only a weak relationship with health, because each works on health in opposite directions.

5M3. Divorce might lead to, or be in expectation of, remarriage. Thus divorce could cause marriage rate to rise. In order to examine this idea, or another like it, the data would need to be structured into more categories, such as remarriage rate versus first marriage rate. Better yet would be longitudinal data. In many real empirical contexts, causation involves feedback loops that can render regression fairly useless, unless some kind of time series framework is used.

5M4. It is worth finding and entering the values yourself, for the practice at data management. But here are the values I found, scraped from Wikipedia and merged into the original data:

R code 5.1

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
d$pct_LDS <- c(0.75, 4.53, 6.18, 1, 2.01, 2.82, 0.43, 0.55, 0.38,
  0.75, 0.82, 5.18, 26.35, 0.44, 0.66, 0.87, 1.25, 0.77, 0.64, 0.81,
  0.72, 0.39, 0.44, 0.58, 0.72, 1.14, 4.78, 1.29, 0.61, 0.37, 3.34,
  0.41, 0.82, 1.48, 0.52, 1.2, 3.85, 0.4, 0.37, 0.83, 1.27, 0.75,
  1.21, 67.97, 0.74, 1.13, 3.99, 0.92, 0.44, 11.5 )
d$L <- standardize( d$pct_LDS )
d$A <- standardize( d$MedianAgeMarriage )
d$M <- standardize( d$Marriage )
d$D <- standardize( d$Divorce )
```

A first regression model including this variable might be:

R code 5.2

```
m_5M4 <- quap(
    alist(
        D ~ dnorm(mu,sigma),
        mu <- a + bM*M + bA*A + bL*L,
        a ~ dnorm(0,0.2),
        c(bA,bM,bL) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
precis( m_5M4 )
```

```
       mean   sd  5.5% 94.5%
a      0.00 0.09 -0.15  0.15
bA    -0.69 0.14 -0.92 -0.46
bM     0.04 0.15 -0.20  0.27
bL    -0.31 0.12 -0.50 -0.12
sigma  0.73 0.07  0.62  0.85
```

As expected, there is a negative association between percent LDS and divorce rate. This model assumes the relationship between divorce rate and percent LDS is linear. This makes sense if the LDS

community has a lower divorce rate within itself only, and so as it makes up more of a State's population, that State's divorce rate declines. This is to say that the expected divorce rate of State i is a "convex" mix of two average divorce rates:

$$D_i = (1 - P)D_{\text{G}} + PD_{\text{LDS}}$$

where D_i is the divorce rate for State i, P is the proportion of the State's population that is LDS, and the two divorce rates D_{G} and D_{LDS} are the divorce rates for gentiles (non-LDS) and LDS, respectively. If $D_{\text{G}} > D_{\text{LDS}}$, then as P increases, the value of D_i increases linearly as well.

But maybe the percent LDS in the population has a secondary impact as a marker of a State-level cultural environment that has lower divorce in more demographic groups than just LDS. In that case, this model will miss that impact. Can you think of a way to address this?

5M5. This is an open-ended question with many good, expanding answers. Here's the basic outline of an approach. The first two implied variables are the rate of obesity O and the price of gasoline P. The first proposed mechanism suggests that higher price P reduces driving D, which in turn increases exercise X, which then reduces obesity O. As a set of regressions, this mechanism implies:

(1) D as a declining function of P
(2) X as a declining function of D
(3) O as a declining function of X

In other words, for the mechanism to work, each predictor above needs be negatively associated with each outcome. Note that each outcome becomes a predictor. That's just how these causal chains look. A bunch of reasonable control variables could be added to each of the regressions above. Consider for example that a very wealthy person will be more insensitive to changes in price, so we might think to interact P with income. The second proposed mechanism suggests that price P reduces driving D which reduces eating out E which reduces obesity O. A similar chain of regressions is implied:

(1) D as a declining function of P
(2) E as an increasing function of D
(3) O as an increasing function of E

5H1. For the graph $M \rightarrow A \rightarrow D$, the implications are that M is independent of D when conditioning on A. You can check this with `dagitty` if you aren't sure:

R code 5.3

```
library(dagitty)
dag_5H1 <- dagitty("dag{M->A->D}")
impliedConditionalIndependencies(dag_5H1)
```

```
D _||_ M | A
```

We can check these with the data, provided we are willing to make some additional statistical assumptions about the functions that relate each variable to the others. The only functions we've used so far in the book are linear (additive) functions. The implication above suggests that a regression of D on both M and A should show little association between D and M. You know from the chapter that this is true in the divorce data sample.

So the data are consistent with this graph. But can you think of a way that marriage rate M would causally influence median age of marriage A? If you cannot, then maybe this graph fails on basic scientific grounds. No data are required.

5H2. The first thing to do is outline the full statistical model. If the DAG is $M \rightarrow A \rightarrow D$, then this implies two regressions. The first regresses A on M and the second D on A. That is the model we need to program, in order to compute the counterfactual prediction of intervening on M. The model contains both regressions, and the estimates from both regressions are used to compute the counterfactual prediction, because any intervention on M first influences A which then influences D.

Start by loading the data:

R code 5.4

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
d$D <- standardize( d$Divorce )
d$M <- standardize( d$Marriage )
d$A <- standardize( d$MedianAgeMarriage )
```

Now the model:

R code 5.5

```
m5H2 <- quap(
    alist(
       # A -> D
        D ~ dnorm( muD , sigmaD ),
        muD <- aD + bAD*A,
       # M -> A
        A ~ dnorm( muA , sigmaA ),
        muA <- aA + bMA*M,
       # priors
        c(aD,aA) ~ dnorm(0,0.2),
        c(bAD,bMA) ~ dnorm(0,0.5),
        c(sigmaD,sigmaA) ~ dexp(1)
    ) , data=d )
precis(m5H2)
```

```
         mean   sd  5.5% 94.5%
aD       0.00 0.10 -0.16  0.16
aA       0.00 0.09 -0.14  0.14
bAD     -0.57 0.11 -0.74 -0.39
bMA     -0.69 0.10 -0.85 -0.54
sigmaD   0.79 0.08  0.66  0.91
sigmaA   0.68 0.07  0.57  0.79
```

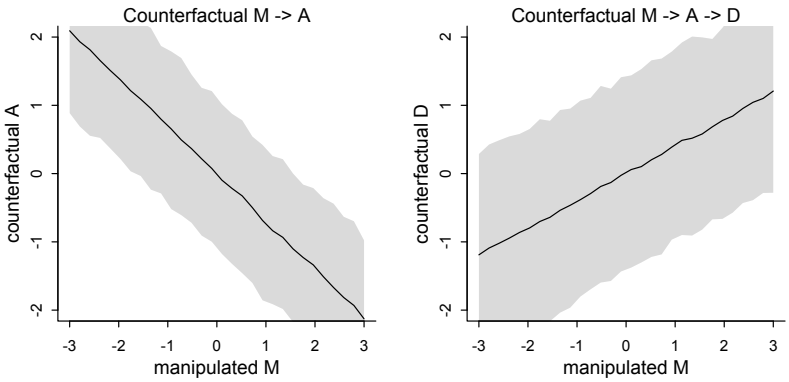
This just shows what you already know from the chapter: A and M are negatively associated and A and D are also negatively associated. It's interpreting these associations as having causal directions that makes counterfactual prediction.

Now the counterfactual prediction is accomplished by building an input range of values for the interventions on M and then simulating both A and then D. We can use `sim` for this. Or you could do it in raw code just with `rnorm` and samples from the posterior. Here is the `sim` approach, just like in the chapter:

R code 5.6

```
M_seq <- seq( from=-3 , to=3 , length.out=30 )

sim_dat <- data.frame( M=M_seq )

s <- sim( m5H2 , data=sim_dat , vars=c("A","D") )

plot( sim_dat$M , colMeans(s$A) , ylim=c(-2,2) , type="l" ,
```

```
    xlab="manipulated M" , ylab="counterfactual A"  )
shade( apply(s$A,2,PI) , sim_dat$M )
mtext( "Counterfactual M -> A" )

plot( sim_dat$M , colMeans(s$D) , ylim=c(-2,2) , type="l" ,
    xlab="manipulated M" , ylab="counterfactual D"  )
shade( apply(s$D,2,PI) , sim_dat$M )
mtext( "Counterfactual M -> A -> D" )
```



The left plot shows the intermediate $M \rightarrow A$ effect, which comes directly from the coefficient `bMA`. This effect is negative, with one unit change in M moving A by about -0.7 on average. The right plot shows the full counterfactual effect of M on D. This effect depends upon both coefficients `bMA` and `bAD`. It is not negative, but rather positive. Why? Because both coefficients are negative. If increasing A decreases D, then decreasing A increases D. Since increasing M decreases A, it follows that increasing M increases D. If that is confusing, that's natural. Luckily the model can get this right without even understanding it.

In simple linear models like these, the total causal effect along a path like $M \rightarrow M \rightarrow D$ is given by multiplying the coefficients along the path. In this case that means the product of `bMA` and `bAD`. So that's $(-0.57) \times (-0.69) \approx 0.4$ at the posterior mean.

Okay, but the question asked what would happen if we halved a State's marriage rate. And this is the hard part of the problem. We have to decide what the original marriage rate was, to know what halving it means. And then we have to convert to the standardized scale that the model uses.

The average marriage rate in the sample is about 20. So if we halve the rate of an average State, it ends up at 10. What is 10 in the standardized units of the model? We find out by standardizing it!

R code 5.7

```
(10 - mean(d$Marriage))/sd(d$Marriage)
```

```
[1] -2.663047
```

That would be a huge effect! Looking at the righthand plot above, going from $M = 0$ to $M = -2.7$ would take us from about $D = 0$ to $D = -1$. Or we could do this calculation more directly:

R code 5.8

```
M_seq <- c( 0 , -2.67 )
sim_dat <- data.frame( M=M_seq )
s <- sim( m5H2 , data=sim_dat , vars=c("A","D") )
diff <- s$D[,2] - s$D[,1]
mean( diff )
```

```
[1] -1.050188
```

So the causal effect of halving an average State's marriage rate is to decrease divorce by a full standard deviation. Well, if this causal model is right.

5H3. This is very similar to the previous problem. The model is only more complicated, but the solution steps are the same. Load the data again:

R code 5.9

```
library(rethinking)
data(milk)
d <- milk
d$K <- standardize( d$kcal.per.g )
d$N <- standardize( d$neocortex.perc )
d$M <- standardize( log(d$mass) )
d2 <- d[ complete.cases( d$K , d$N , d$M ) , ]
```

The causal model is $K \leftarrow M \rightarrow N \rightarrow K$. It'll be easier if we break this down into the component functions: (1) K is a function of M and N, (2) N is a function of M. So again this is just two regression models, run simultaneously.

R code 5.10

```
m5H3 <- quap(
    alist(
       # M -> K <- N
        K ~ dnorm( muK , sigmaK ),
        muK <- aK + bMK*M + bNK*N,
       # M -> N
        N ~ dnorm( muN , sigmaN ),
        muN <- aN + bMN*M,
       # priors
        c(aK,aN) ~ dnorm( 0 , 0.2 ),
        c(bMK,bNK,bMN) ~ dnorm( 0 , 0.5 ),
        c(sigmaK,sigmaN) ~ dexp( 1 )
    ) , data=d2 )
precis( m5H3 )
```

```
         mean   sd  5.5% 94.5%
aK       0.07 0.13 -0.15  0.28
aN      -0.01 0.12 -0.21  0.18
bMK     -0.70 0.22 -1.06 -0.35
bNK      0.68 0.25  0.28  1.07
bMN      0.61 0.13  0.40  0.83
sigmaK   0.74 0.13  0.53  0.95
sigmaN   0.63 0.11  0.46  0.80
```

This problem also asks the awkward question to find out what would happen if we double M. So we need to pick some value to double. So let's pick a species, double its mass, then covert that the standardized log mass. Sounds complicated, but take it one step at a time.

Body mass in this sample rages from 0.12 kg to 97.72 kg, with a mean of 15 kg. Let's consider a species of average mass and double it. In standardized log units:

R code 5.11

```
(log(c(15,30)) - mean(log(d$mass)))/sd(log(d$mass))
```

```
[1] 0.7457136 1.1539295
```

The first value above is the standardized value for 1 kg. The second is for 2 kg. Now the calculation:

R code 5.12

```
M_seq <- c( 0.75 , 1.15 )
sim_dat <- data.frame( M=M_seq )
s <- sim( m5H3 , data=sim_dat , vars=c("N","K") )
diff <- s$K[,2] - s$K[,1]
quantile( diff , probs=c( 0.05 , 0.5 , 0.94 ) )

M_seq <- seq( from=-3 , to=3 , length.out=30 )
sim_dat <- data.frame( M=M_seq )
s <- sim( m5H3 , data=sim_dat , vars=c("N","K") )
```

```
        5%         50%         94%
-2.2706577 -0.1297385  1.8274152
```

So doubling mass from 15 kg to 30 kg would (according to the model) have no strong effect on K. It is expected to decline a bit. But the compatibility interval above is the entire range of the variable, basically.

This makes sense if you just think about the paths again. In a linear model, we can get the causal effect by multiplying the coefficients on each path and then adding all the paths together. In this case, we need:

R code 5.13

```
0.61*0.68 - 0.7
```

```
[1] -0.2852
```

But the effect we computed is only for a change of $1.15 - 0.75 = 0.4$, so we want 0.4 of the above:

R code 5.14

```
( 0.61*0.68 - 0.7 )*0.4
```

```
[1] -0.11408
```

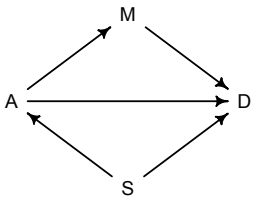
And that's almost what we got from the simulation as the median effect. This only works for linear models, remember. The simulation approach will work for any model.

5H4. The important thing in this problem is not which DAG you draw, but how you analyze it. That is, for whatever DAG you produce, you need to correctly infer its testable implications. Then you can use the divorce data to evaluate these implications (at least under the rubric of linear models).

For the sake of the example, let's suppose southernness S influences both age of marriage A and divorce rate D. So this is the DAG:

R code 5.15

```
library(dagitty)
dag_5H4 <- dagitty("dag{ M <- A -> D; A <- S -> D; M -> D }")
coordinates(dag_5H4) <- list(
    x=c(A=0,M=1,D=2,S=1),
    y=c(A=1,M=0,D=1,S=2) )
drawdag( dag_5H4 )
```



And the testable implications:

R code 5.16

```
impliedConditionalIndependencies( dag_5H4 )
```

```
M _||_ S | A
```

So M and S should be independent, after conditioning on A. Inspect the graph and make sure you can explain why this is the only testable implication.

Let's make a simple linear model to see whether this implication holds. We just need a linear regression with M as the outcome and both S and A as predictors. There is already a variable in the data, `South`, that indicates which States are in the south.

R code 5.17

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
d$D <- standardize( d$Divorce )
d$M <- standardize( d$Marriage )
d$A <- standardize( d$MedianAgeMarriage )
d$S <- d$South

m_5H4 <- quap(
    alist(
        M ~ dnorm( mu , sigma ),
        mu <- a + bS*S + bA*A,
        a ~ dnorm( 0 , 0.2 ),
        c(bS,bA) ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp( 1 )
    ) , data=d )
precis( m_5H4 )
```

```
       mean   sd  5.5% 94.5%
a      0.04 0.10 -0.12  0.19
bS    -0.17 0.19 -0.48  0.14
bA    -0.71 0.10 -0.87 -0.56
sigma  0.68 0.07  0.57  0.78
```

The coefficient on S is small and has plenty of mass on both sides of zero. So the implication isn't obviously false.

6. Chapter 6 Solutions

6E1. The mechanisms outlined in the chapter (and the previous) were: (1) confounding (a third variable that influences the exposure and outcome) (2) multicollinearity, (3) conditioning on post-treatment variables, (4) collider bias.

6E2. The example provided needs to be consistent with the named mechanism. From my own research interests, the influence of grandparents on grandchildren is confounded by how much in need the grandchildren are. For example, if grandparents choose to go where they are most needed, then presence of grandparents could be associated with worse grandchild outcomes. But it would be a mistake in that case to infer that grandparents hurt their grandkids.

6E3. They are, with their conditional independencies:

(1) The fork, $X \leftarrow Z \rightarrow Y$, $Y \perp\!\!\!\perp X|Z$
(2) The pipe, $X \rightarrow Z \rightarrow Y$, $Y \perp\!\!\!\perp X|Z$
(3) The collider, $X \rightarrow Z \leftarrow Y$, $Y \not\perp\!\!\!\perp X|Z$
(4) The descendant, which is a variable influenced by another. When we condition on a descendent, we weakly condition on its parent, and so what happens depends upon the parent's place in the graph.

6E4. The biased sample in the introduction to the chapter are those proposals that were good enough to pass a threshold. A proposal can pass a threshold if either criterion is good enough. Therefore among those selected, there tends to be a negative association between the criteria. The association is negative also in those that aren't selected, but we don't get to see those.

In conditioning on a collider, the sample is similarly divided into sub-samples. And the association in the sub-samples can be different than in the total sample. But we do it ourselves and it can produce similar statistical consequences as selection bias.

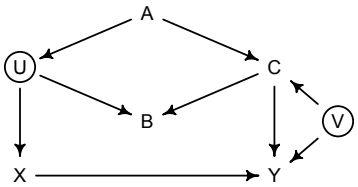
Think about a lamp. It can be either on or off. The lamp is caused by the switch being on and the electricity working. It is a collider. If we condition on the lamp's state, then there are two sub-samples: those lamps that are on and those that are off. Among that that are on, both the switch and the electricity must be on. Among those that are off, at least one is off. Therefore there can be a negative association between the switch and the electricity in the sub-sample of lamps that are off, even when there is no association between the switch and electricity in the entire sample.

6M1. The DAG is now:

R code 6.1

```
library(dagitty)
dag_6M1 <- dagitty("dag{
    U [unobserved]
    V [unobserved]
    X -> Y
    X <- U -> B <- C -> Y
    U <- A -> C
    C <- V -> Y }")
coordinates(dag_6M1) <- list(
```

```
    x=c(X=0,Y=2,U=0,A=1,B=1,C=2,V=2.5),
    y=c(X=2,Y=2,U=1,A=0.5,B=1.5,C=1,V=1.5) )
drawdag(dag_6M1)
```



Now there are 5 paths connecting X and Y:

(1) $X \rightarrow Y$ (the causal path)
(2) $X \leftarrow U \rightarrow B \leftarrow C \rightarrow Y$
(3) $X \leftarrow U \rightarrow B \leftarrow C \leftarrow V \rightarrow Y$
(4) $X \leftarrow U \leftarrow A \rightarrow C \rightarrow Y$
(5) $X \leftarrow U \leftarrow A \rightarrow C \leftarrow V \rightarrow Y$

We want to leave path 1 open and make sure all of the others are closed, because they are non-causal paths that will confound inference. As before, all the paths through B are already closed, since B is a collider. So we don't condition on B. Similarly, the new paths through both A and V are closed, because C is a collider on those paths. So it's enough to condition on A to close all non-causal paths.

If you like, you can check using `dagitty`:

R code 6.2

```
adjustmentSets( dag_6M1 , exposure="X" , outcome="Y" )
```

```
 { A }
```

Note that in the chapter, where V is absent, it is also fine to condition on C. It isn't now.

6M2. To simulate:

R code 6.3

```
N <- 1000
X <- rnorm(N)
Z <- rnorm(N,X,0.1)
Y <- rnorm(N,Z)
cor(X,Z)
```

```
[1] 0.995054
```

That's a high correlation. X and Z are almost the same variable. Now let's regress:

R code 6.4

```
m_6M2 <- quap(
    alist(
        Y ~ dnorm( mu , sigma ),
        mu <- a + bX*X + bZ*Z,
        c(a,bX,bZ) ~ dnorm(0,1),
        sigma ~ dexp(1)
    ) , data=list(X=X,Y=Y,Z=Z) )
precis( m_6M2 )
```

```
       mean   sd  5.5% 94.5%
a     -0.06 0.03 -0.10 -0.01
```

```
bX     0.15 0.28 -0.30  0.60
bZ     0.83 0.28  0.39  1.28
sigma  0.97 0.02  0.94  1.01
```

The standard deviations are larger than we might expect, for such a large sample. But the model has no trouble finding the truth here. This is an example of conditioning on a post-treatment variable—it knocks out X because Z mediates entirely the effect of X.

Bottom line: We can't interpret multicollinearity outside of a specific model. It isn't a property of the data alone.

6M3. In each case, the procedure is the same: List all paths between X and Y and figure out which non-causal paths are open and which variables you can use to close them. We'll consider each in turn.

Top left. There are three paths between X and Y: (1) $X \rightarrow Y$, (2) $X \leftarrow Z \rightarrow Y$, (3) $X \leftarrow Z \leftarrow A \rightarrow Y$. Both (2) and (3) are open, non-causal paths. Conditioning on Z is sufficient to close both.

Top right. Again three paths: (1) $X \rightarrow Y$, (2) $X \rightarrow Z \rightarrow Y$, (3) $X \rightarrow Z \leftarrow A \rightarrow Y$. Paths (1) and (2) are both causal. We want both open. Path (3) is non-causal, but it is also closed already because Z is a collider on that path.

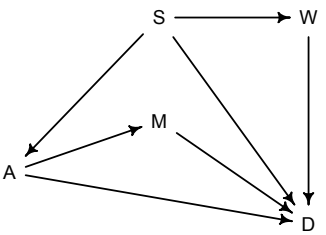
Bottom left. Paths: (1) $X \rightarrow Y$, (2) $X \rightarrow Z \leftarrow Y$, (3) $X \leftarrow A \rightarrow Z \leftarrow Y$. Only path (1) is causal. The other paths are both closed, because both contain a collider at Z. Conditioning on Z would be a disaster.

Bottom right. Paths: (1) $X \rightarrow Y$, (2) $X \rightarrow Z \rightarrow Y$, (3) $X \leftarrow A \rightarrow Z \rightarrow Y$. Paths (1) and (2) are causal. Path (3) is an open backdoor path. To close it, we must condition on A or Z, but if condition on Z, that would close a causal path.

6H1. How hard this is depends upon the DAG you draw. Here's one idea:

R code 6.5

```
library(dagitty)
dag_6H1 <- dagitty("dag{A -> D; A -> M -> D; A <- S -> D; S -> W -> D}")
coordinates(dag_6H1) <- list(
    x=c(A=0,M=1,D=2,S=1,W=2),
    y=c(A=0.75,M=0.5,D=1,S=0,W=0) )
drawdag(dag_6H1)
```



Given this DAG, you can probably see that all we need to control for is S, an indicator for southern States. This is needed to block the backdoor paths through the other variables.

It's very likely that there are unobserved variables that confound this inference. For example, if Waffle House tends to build in places where it is cheap to operate, then there are economic confounds that might influence both the presence of Waffle Houses and divorce rate. In that case, we'd need to measure those confounds to get an inference. Lots of social scientists believe that it is usually impossible to measure all the probably confounds in these studies. They look for natural or quasi-experiments instead.

6H2. The DAG from the previous answer has several testable implications.

R code 6.6

```
impliedConditionalIndependencies( dag_6H1 )
```

```
A _||_ W | S
M _||_ S | A
M _||_ W | S
M _||_ W | A
```

Now find the `WaffleHouses` variable in the divorce data. You build a series of regression models to evaluate the above implications. For example, the first implication needs a regression of A on W and S. For all the implications, you should find them credibly: (1) true, (2) true, (3) true, (4) true.

6H3. Because there are no back-door paths from `area` to `weight`, we only need to include `area`. No other variables are needed. Here is a model using standardized versions of the variables and those standardized priors from the book:

R code 6.7

```
library(rethinking)
data(foxes)
d <- foxes
d$W <- standardize(d$weight)
d$A <- standardize(d$area)
m1 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bA*A,
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
precis(m1)
```

```
      mean   sd  5.5% 94.5%
a     0.00 0.08 -0.13  0.13
bA    0.02 0.09 -0.13  0.16
sigma 0.99 0.06  0.89  1.09
```

Territory size seems to have no total causal influence on weight, at least not in this sample.

6H4. To infer the causal influence of `avgfood` on `weight`, we need to close any back-door paths. There are no back-door paths in the DAG. So again, just use a model with a single predictor. If you include `groupsize`, to block the indirect path, then you won't get the total causal influence of food. You'll just get the direct influence. But I asked for the effect of adding food, and that would mean through all forward paths.

R code 6.8

```
d$F <- standardize(d$avgfood)
m2 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bF*F,
        a ~ dnorm(0,0.2),
        bF ~ dnorm(0,0.5),
```

```
        sigma ~ dexp(1)
    ), data=d )
precis(m2)
```

```
       mean   sd  5.5% 94.5%
a      0.00 0.08 -0.13  0.13
bF    -0.02 0.09 -0.17  0.12
sigma  0.99 0.06  0.89  1.09
```

Again nothing. Adding food does not change weight. This shouldn't surprise you, if the DAG is correct, because `area` is upstream of `avgfood`.

6H5. The variable `groupsize` does have a back-door path, passing through `avgfood`. So to infer the causal influence of `groupsize`, we need to close that path. This implies a model with both `groupsize` and `avgfood` as predictors.

R code 6.9

```
d$G <- standardize(d$groupsize)
m3 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bF*F + bG*G,
        a ~ dnorm(0,0.2),
        c(bF,bG) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
precis(m3)
```

```
       mean   sd  5.5% 94.5%
a      0.00 0.08 -0.13  0.13
bF     0.48 0.18  0.19  0.76
bG    -0.57 0.18 -0.86 -0.29
sigma  0.94 0.06  0.84  1.04
```

It looks like group size is negatively associated with weight, controlling for food. Similarly, food is positively associated with weight, controlling for group size. So the causal influence of group size is to reduce weight—less food for each fox. And the direct causal influence of food is positive, of course. But the total causal influence of food is still nothing, since it causes larger groups. This is a masking effect, like in the milk energy example. But the causal explanation here is that more foxes move into a territory until the food available to each is no better than the food in a neighboring territory. Every territory ends up equally good/bad on average. This is known in behavioral ecology as an *ideal free distribution*.

Problems **6H6** and **6H7** have no specific solutions, given that they depend upon individual questions and topics.

7. Chapter 7 Solutions

7E1. The criteria for this measure of "uncertainty" are:

(1) Continuity. The measure is not discrete, but it may be bounded. So it can have a minimum and maximum, but it must be continuous in between.
(2) Increasing with the number of events. When more distinct things can happen, and all else is equal, there is more uncertainty than when fewer things can happen.
(3) Additivity. This is hardest one to understand, for most people. Additivity is desirable because it means we can redefine event categories without changing the amount of uncertainty.

7E2. This is a simple calculation in R, very much like the example on page 192:

R code 7.1

```
# define probabilities of heads and tails
p <- c( 0.7 , 0.3 )

# compute entropy
-sum( p*log(p) )
```

```
[1] 0.6108643
```

7E3. Similar to the problem above, but now with four types of events:

R code 7.2

```
# define probabilities of sides
p <- c( 0.2 , 0.25 , 0.25 , 0.3 )

# compute entropy
-sum( p*log(p) )
```

```
[1] 1.376227
```

7E4. When events are impossible (have probability of zero), they just fall out of the calculation:

R code 7.3

```
# define probabilities of sides
p <- c( 1/3 , 1/3 , 1/3 )

# compute entropy
-sum( p*log(p) )
```

```
[1] 1.098612
```

7M1. The definition of AIC is:

$$\begin{aligned} \text{AIC} &= -2 \log \Pr(\text{data}|\text{MAP estimates}) + 2p \\ &= D_{\text{train}} + 2p \end{aligned}$$

where p is the number of parameters in the model. WAIC is a notational mess, but once you understand each part, then the notation makes sense:

$$\begin{aligned} \text{WAIC} &= -2(\text{lppd} - p_{\text{WAIC}}) \\ &= -2\left(\sum_i \log \Pr(y_i) - \sum_i V(y_i)\right) \end{aligned}$$

where $\Pr(y_i)$ is the likelihood of observation i, averaging over the posterior distribution, and $V(y_i)$ is the variance in the log-likelihood of observation i, taking the variance over the posterior distribution.

Comparing these definitions, each has a term that expresses the fit to the training sample, as well as a term that expresses some penalty for flexibility in fitting to the sample. For AIC, the fit term is simply the "plug-in deviance," D_{train}, and the penalty term is twice the number of parameters, p. For WAIC, the fit term is a fully Bayesian log-likelihood that averages over the posterior prediction for each observation separately. This is lppd. The penalty term is the sum of the variances of each log-likelihood.

From more general to less general: WAIC, AIC. When the posterior predictive mean is a good representation of the posterior predictive distribution, and the priors are effectively flat or overwhelmed by the amount of data, then WAIC and AIC will tend to agree.

7M2. Model selection means ranking models by information criteria (or any other criteria) and choosing the highest ranked model. Model averaging is instead weighting each model by its relative distance from the best model and creating a posterior predictive distribution comprising predictions from each model in proportion to these weights, a prediction "ensemble." Model selection discards information about model uncertainty. Model averaging retains some of that information, but it still discards some information about model uncertainty, because it compresses the full information about the set of models into a single predictive distribution. Since the full set of ranks and information criteria values cannot be recovered from the averaged predictive distribution, some of that information has been lost. In other words, different model comparison sets with different ranks and information criteria values can produce nearly identical prediction ensembles. But those different model comparison sets may be different in other important ways. For example, inspecting the full model comparison set, with the structure of each model, often reveals why some models fit better than others.

7M3. When one model is fit to fewer (or different) observations, it is being judged on a different target than the other models. If fewer observations are used, the model will usually appear to perform better, because the deviance will smaller due to having less to predict. Less to predict means less prediction error, and deviance (as well as information criteria) is a kind of accumulated error. We have to be careful about this, because most of R's black-box regression functions like `lm`, `glm`, and `glmer` will automatically and silently drop incomplete cases, reducing the number of observations the model is fit to. This is bad behavior for scientific software, but unfortunately it is the norm.

7M4. As a prior becomes more concentrated around particular parameter values, the model becomes less flexible in fitting the sample. One way to remember this is to think of the prior as representing previous learning from previous observations. So a more concentrated, or peaked, prior represents more previous data. As the model becomes less flexible, the effective number of parameters declines.

To perform some simple experiments to demonstrate this, try out this code, changing the value for `sigma` in the data list. The smaller `sigma`, the more concentrated the prior and the smaller the effective number of parameters should be.

R code 7.4

```
y <- rnorm(10) # execute just once, to get data

# repeat this, changing sigma each time
m <- quap(
    alist(
        y ~ dnorm(mu,1),
        mu ~ dnorm(0,sigma)
    ), data=list(y=y,sigma=10) )
WAIC(m)
```

7M5. Informative priors reduce overfitting by reducing the sensitivity of a model to a sample. Some of the information in a sample is *irregular*, not a recurring feature of the process of interest.

7M6. If a prior is overly informative, then even regular features of a sample will not be learned by a model. In this case, the model may underfit the sample. In case this sounds like a terrifying balancing act, in practice there is usually a broad family of priors which achieve practically indistinguishable estimates from the same sample. The last "hard" practice problem for this chapter provides an example.

7H1/7H2. It's probably obvious from the prompt that the original curve was hand-drawn and not the result of any fitting procedure. And you can see there is a high outlier point, while the rest of the points show a general increase. But let's take this seriously and polish our modeling skills.

Lots of combinations of models will produce the same general inference. I'll start with comparing linear to polynomial models. Then I'll try a spline. Then I'll consider a robust regression to cope better with the outlier.

Here is a linear fit, together with a quadratic and cubic fit:

R code 7.5

```
library(rethinking)
data(Laffer)
d <- Laffer
d$T <- standardize( d$tax_rate )
d$R <- standardize( d$tax_revenue )

# linear model
m7H1a <- quap(
    alist(
        R ~ dnorm( mu , sigma ),
        mu <- a + b*T,
        a ~ dnorm( 0 , 0.2 ),
        b ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ) , data=d )

# quadratic model
m7H1b <- quap(
    alist(
        R ~ dnorm( mu , sigma ),
```

```
        mu <- a + b*T + b2*T^2,
        a ~ dnorm( 0 , 0.2 ),
        c(b,b2) ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ) , data=d )

# cubic model
m7H1c <- quap(
    alist(
        R ~ dnorm( mu , sigma ),
        mu <- a + b*T + b2*T^2 + b3*T^3,
        a ~ dnorm( 0 , 0.2 ),
        c(b,b2,b3) ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ) , data=d )
```

As a first check, let's see how these models compare on purely predictive criteria, like PSIS:

R code 7.6

```
compare( m7H1a , m7H1b , m7H1c , func=PSIS )
```

```
Some Pareto k values are very high (>1)
      PSIS    SE dPSIS  dSE pPSIS weight
m7H1a 93.3 26.88   0.0   NA   8.3   0.78
m7H1b 96.5 32.37   3.2 5.75  10.9   0.16
m7H1c 98.3 31.82   5.0 5.25  11.8   0.06
```

Not much support for the wiggly models over the linear one. But PSIS also warns about high leverage points. You can probably guess from scatterplot which point is causing the problem. If you peek at the pointwise k values, you'll see:

R code 7.7

```
PSISk(m7H1a)
```

```
 [1]  0.61  0.46  0.44  0.07 -0.03  0.09  0.28  0.29 -0.07  0.04  0.43  1.85
[13]  0.34  0.34 -0.11 -0.07 -0.13 -0.08  0.31 -0.01 -0.02 -0.02  0.05  0.04
[25]  0.07  0.26  0.28  0.23  0.10
```

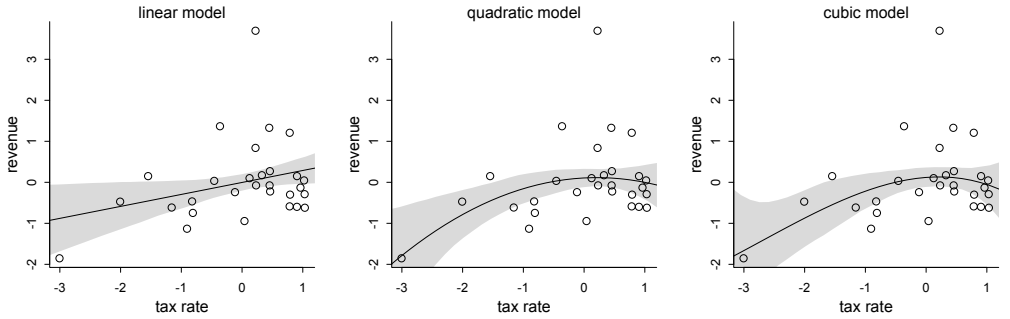
Point 12 is much over 1. That's the nation with the very high tax revenue value. I think it's Norway, actually, and it results from an accounting trick involving oil revenue.

It'll be useful to plot the posterior predictions of each model.

R code 7.8

```
T_seq <- seq( from=-3.2 , to=1.2 , length.out=30 )
la <- link( m7H1a , data=list(T=T_seq) )
lb <- link( m7H1b , data=list(T=T_seq) )
lc <- link( m7H1c , data=list(T=T_seq) )

plot( d$T , d$R , xlab="tax rate" , ylab="revenue" )
mtext( "linear model" )
lines( T_seq , colMeans(la) )
shade( apply( la , 2 , PI ) , T_seq )

plot( d$T , d$R , xlab="tax rate" , ylab="revenue" )
mtext( "quadratic model" )
lines( T_seq , colMeans(lb) )
shade( apply( lb , 2 , PI ) , T_seq )
```

```
plot( d$T , d$R , xlab="tax rate" , ylab="revenue" )
mtext( "cubic model" )
lines( T_seq , colMeans(lc) )
shade( apply( lc , 2 , PI ) , T_seq )
```



The polynomial models do bend the right way, but they aren't bending much. If anything, they offer stronger evidence that more tax produces more revenue, not that very high tax reduces revenue.

For the sake of the exercise, let's consider a basis spline. Refer back to Chapter 4, the last section, if you've forgotten splines.

R code 7.9

```
num_knots <- 15
knot_list <- quantile( d$T , probs=seq(0,1,length.out=num_knots) )

library(splines)
B <- bs(d$T,
    knots=knot_list[-c(1,num_knots)] ,
    degree=3 , intercept=TRUE )

m7H1s <- quap(
    alist(
        R ~ dnorm( mu , sigma ) ,
        mu <- a + B %*% w ,
        a ~ dnorm(0,1),
        w ~ dnorm(0,1),
        sigma ~ dexp(1)
    ), data=list( R=d$R , B=B ) ,
    start=list( w=rep( 0 , ncol(B) ) ) )
```

The coefficients aren't interpretable, but let's compare to the previous models:

R code 7.10

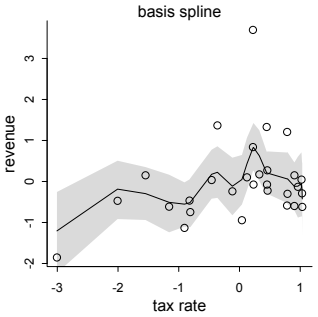
```
compare( m7H1a , m7H1b , m7H1c , m7H1s , func=PSIS )
```

```
Some Pareto k values are very high (>1)
       PSIS    SE dPSIS  dSE pPSIS weight
m7H1a  93.1 26.81   0.0   NA   8.2   0.60
m7H1c  95.1 28.45   2.1 2.57  10.2   0.21
m7H1b  95.4 30.30   2.4 3.74  10.3   0.18
m7H1s 102.4 31.75   9.3 6.06  17.7   0.01
```

Terrible. What do the predictions look like?

R code 7.11

```
mu <- link( m7H1s )
mu_PI <- apply( mu , 2 , PI )
plot( d$T , d$R , xlab="tax rate" , ylab="revenue" )
mtext( "basis spline" )
o <- order( d$T )
lines( d$T[o] , colMeans(mu)[o] )
shade( mu_PI[,o] , d$T[o] )
```



If you really want to interpret that as support for the curved relationship, good luck. You might try more rigid splines, with lower polynomial order, fewer knots, and tighter weights, to see if this can be made more sensible. But clearly this is a curve-fitting exercise, not science, at this point.

For the Student-t model, we just need to replace the normal likelihood. I'll use $\nu = 2$, so the tails are satisfyingly thick.

R code 7.12

```
# linear model with student-t
m7H1d <- quap(
    alist(
        R ~ dstudent( 2 , mu , sigma ),
        mu <- a + b*T,
        a ~ dnorm( 0 , 0.2 ),
        b ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ) , data=d )
```

Comparing the previous models:

R code 7.13

```
compare( m7H1a , m7H1b , m7H1c , m7H1s , m7H1d , func=PSIS )
```

```
Some Pareto k values are very high (>1)
       PSIS    SE dPSIS   dSE pPSIS weight
m7H1d  75.0 13.70   0.0    NA   4.0      1
m7H1b  89.1 25.23  14.0 16.78   7.1      0
m7H1a  98.8 32.32  23.8 23.44  11.0      0
m7H1c 103.2 36.99  28.1 27.91  14.2      0
m7H1s 105.5 33.20  30.5 23.88  19.3      0
```

Oh PSIS likes this one. It also doesn't itself throw any Pareto-k warnings. If you like, try to polynomial models with the Student-t likelihood. Overall, there is no obvious support for a strongly increasing-then-decreasing relationship between tax rate and tax revenue.

7H3. To compute the entropies, we just need a function to compute the entropy. Information entropy, as defined in lecture and the book, is simply:

$$H(p) = -\sum_i p_i \log(p_i)$$

where p is a vector of probabilities summing to 1. In R code this would look like:

R code 7.14

```
H <- function(p) -sum(p*log(p))
```

I'll make a list of the birb distributions and then push each through the function above.

R code 7.15

```
IB <- list()
IB[[1]] <- c( 0.2 , 0.2 , 0.2 , 0.2 , 0.2 )
IB[[2]] <- c( 0.8 , 0.1 , 0.05 , 0.025 , 0.025 )
IB[[3]] <- c( 0.05 , 0.15 , 0.7 , 0.05 , 0.05 )
sapply( IB , H )
```

```
[1] 1.6094379 0.7430039 0.9836003
```

The first island has the largest entropy, followed by the third, and then the second in last place. Why is this? Entropy is a measure of the evenness of a distribution. The first islands has the most even distribution of birbs. This means you wouldn't be very surprised by any particular birb. The second island, in contrast, has a very uneven distribution of birbs. If you saw any birb other than the first species, it would be surprising.

Now we need K-L distance, so let's write a function for it:

R code 7.16

```
DKL <- function(p,q) sum( p*(log(p)-log(q)) )
```

This is the distance from q to p, regarding p as true and q as the model. Now to use each island as a model of the others, we need to consider the different ordered pairings. I'll just make a matrix and loop over rows and columns:

R code 7.17

```
Dm <- matrix( NA , nrow=3 , ncol=3 )
for ( i in 1:3 ) for ( j in 1:3 ) Dm[i,j] <- DKL( IB[[j]] , IB[[i]] )
round( Dm , 2 )
```

```
     [,1] [,2] [,3]
[1,] 0.00 0.87 0.63
[2,] 0.97 0.00 1.84
[3,] 0.64 2.01 0.00
```

The way to read this is each row as a model and each column as a true distribution. So the first island, the first row, has the smaller distances to the other islands. This makes sense, since it has the highest entropy. Why does that give it a shorter distance to the other islands? Because it is less surprised by the other islands, due to its high entropy.

7H4. I won't repeat the models here. They are in the text. Model `m6.9` contains both marriage status and age. Model `m6.10` contains only age. Model `m6.9` produces a confounded inference about the relationship between age and happiness, due to opening a collider path. To compare these models using WAIC:

R code 7.18

```
compare( m6.9 , m6.10 )
```

```
        WAIC pWAIC dWAIC weight    SE  dSE
m6.9  2714.0   3.7   0.0      1 37.54   NA
m6.10 3101.9   2.3 387.9      0 27.74 35.4
```

The model that produces the invalid inference, `m6.9`, is expected to predict much better. And it would. This is because the collider path does convey actual association. We simply end up mistaken about the causal inference. We should not use WAIC (or LOO) to choose among models, unless we have some clear sense of the causal model. These criteria will happily favor confounded models.

7H5. These are the models:

R code 7.19

```
library(rethinking)
data(foxes)
d <- foxes
d$W <- standardize(d$weight)
d$A <- standardize(d$area)
d$F <- standardize(d$avgfood)
d$G <- standardize(d$groupsize)

m1 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bF*F + bG*G + bA*A,
        a ~ dnorm(0,0.2),
        c(bF,bG,bA) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
m2 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bF*F + bG*G,
        a ~ dnorm(0,0.2),
        c(bF,bG) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
m3 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bG*G + bA*A,
        a ~ dnorm(0,0.2),
        c(bG,bA) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
m4 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bF*F,
        a ~ dnorm(0,0.2),
        bF ~ dnorm(0,0.5),
        sigma ~ dexp(1)
```

```
    ), data=d )
m5 <- quap(
    alist(
        W ~ dnorm( mu , sigma ),
        mu <- a + bA*A,
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=d )
```

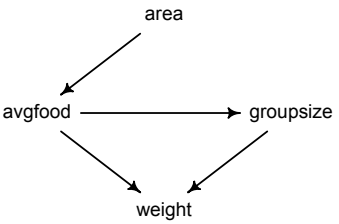
Comparing with WAIC:

```
compare( m1 , m2 , m3 , m4 , m5 )
```

R code 7.20

```
    WAIC pWAIC dWAIC weight    SE  dSE
m1 322.9   4.7   0.0   0.47 16.28   NA
m3 323.9   3.7   1.0   0.28 15.68 2.90
m2 324.1   3.9   1.2   0.25 16.14 3.60
m4 333.4   2.4  10.6   0.00 13.79 7.19
m5 333.7   2.7  10.8   0.00 13.79 7.24
```

To remind you, the DAG from the original problem is:



Notice that the top three models are `m1`, `m3`, and `m2`. They have very similar WAIC values. The differences are small and smaller in all cases than the standard error of the difference. WAIC sees these models are tied. This makes sense, given the DAG, because as long as a model has `groupsize` in it, we can include either `avgfood` or `area` or both and get the same inferences. Another way to think of this is that the influence of good, adjusting for group size, is (according to the DAG) the same as the influence of area, adjusting for group size, because the influence of area is routed entirely through food and group size. There are no backdoor paths.

What about the other two models, `m4` and `m5`? These models are tied with one another, and both omit group size. Again, the influence of area passes entirely through food. So including only food or only area should produce the same inference—the total causal influence of area (or food) is just about zero. That's indeed what the posterior distributions suggest:

```
coeftab(m4,m5)
```

R code 7.21

```
      m4      m5
a      0       0
bF    -0.02     NA
sigma  0.99    0.99
bA       NA    0.02
nobs     116   116
```

8. Chapter 8 Solutions

8E1. There are many good responses. Here are some examples.

(1) Bread dough rises because of yeast, conditional on the presence of sugar. This is an interaction between yeast and sugar.
(2) Education leads to higher income, conditional on field of study. Some types of degrees lead to higher income than others.
(3) Gasoline makes a car go, unless it has no spark plugs (or belts, or wheels, etc.).

8E2. Only number (1) is a strict interaction. The others all imply additive influences. For (1), you could express it as "cooking with low heat leads to caramelizing, conditional on the onions not drying out." That's an interaction, as the effect of heat depends upon moisture. For (2), a car with many cylinders can go fast, whether or not it also has a good fuel injector. The reverse is also implied: either a fuel injector or more cylinders is sufficient to make a car go faster. For (3), the statement does not imply that the influence of parents depends upon the beliefs of friends. It just implies that one may be influenced by either parents or friends. For (4), the word "or" gives away that this is another case in which either factor, sociality or manipulative appendages (hands, tentacles), is sufficient to predict intelligence.

8E3.

(1) For outcome "extent caramelized", $\mu_i = \alpha + \beta_H H_i + \beta_M M_i + \beta_{HM} H_i M_i$.
(2) For outcome "maximum speed", $\mu_i = \alpha + \beta_C C_i + \beta_F F_i$.
(3) For outcome "extent conservative", $\mu_i = \alpha + \beta_P P_i + \beta_F F_i$.
(4) For outcome "intelligence" (whatever that means when comparing an octopus to a monkey), $\mu_i = \alpha + \beta_S S_i + \beta_M M_i$.

8M1. Tulips are a winter flower in much of their natural range. High temperatures do frustrate them. This is not a *linear* interaction, because by raising temperature the effect was to prevent all blooms. A linear effect would be an additive change. But it is still correct to say that there is a three-way interaction here: the influence of water and shade depend upon one another, and both and their interaction depend upon temperature. In later chapters, you'll see how to model non-linear responses of this kind.

8M2. Here is one idea. Let L_i stand for the ordinary linear model from the chapter. Then let H_i be a 0/1 indicator of whether or not the temperature was hot. Then:

$$\mu_i = L_i(1 - H_i)$$

When $H_i = 1$, the entire model above is zero, regardless of the value of L_i.

8M3. The implied relationship between ravens and wolves is one in which wolves do not need ravens, but ravens very much do benefit from wolves (at least in some places). Here's an example set of data in which this might be the case:

Region	Wolves	Ravens
1	12	43
2	15	46
3	7	28
4	30	99
5	17	60
6	70	212

Really, this "interaction" is not a statical interaction effect at all, because just stating that ravens depend upon wolves implies that we can partially predict raven density with wolf density. A statistical interaction requires instead that some other third variable regulate the dependency of ravens on wolves. For example, in regions in which there is plenty of small prey for ravens to kill and consume on their own, the presence of wolves may not matter at all.

8M4. The direct approach is to use log-normal or exponential distributions. Both of these distributions are strictly positive. We can force the shade effect to be negative by using a minus in the linear model.

Thinking about the interaction is harder. But our prior information tells us that more water increases the effect of light. So similarly more water decreases the impact of shade. So the interaction should be negative. So we can force the interaction effect to also have a log-normal or exponential, but make it negative by adding a minus in the linear model.

Here's the code with log-normal. Remember (or look up) that the mean of a log-normal is $\exp(\mu + \sigma^2/2)$. So we can't make these values very large at all, if we want to prior mean to be reasonable. Let's load the data and try a first guess:

R code 8.1

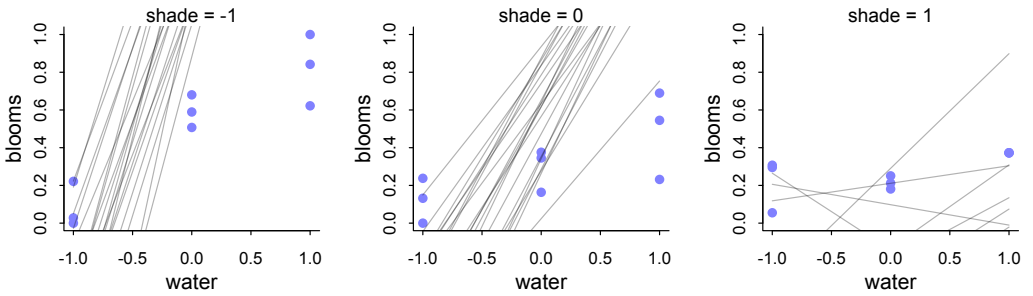
```
library(rethinking)
data(tulips)
d <- tulips
d$blooms_std <- d$blooms / max(d$blooms)
d$water_cent <- d$water - mean(d$water)
d$shade_cent <- d$shade - mean(d$shade)

m8M4a <- quap(
    alist(
        blooms_std ~ dnorm( mu , sigma ) ,
        mu <- a + bw*water_cent - bs*shade_cent - bws*water_cent*shade_cent ,
        a ~ dnorm( 0.5 , 0.25 ) ,
        bw ~ dlnorm( 0 , 0.25 ) ,
        bs ~ dlnorm( 0 , 0.25 ) ,
        bws ~ dlnorm( 0 , 0.25 ) ,
        sigma ~ dexp( 1 )
    ) , data=d )
```

Before looking at the fit, let's do prior simulation to see if these priors make sense.

R code 8.2

```
p <- extract.prior( m8M4a )
par(mfrow=c(1,3),cex=1.1) # 3 plots in 1 row
for ( s in -1:1 ) {
```

```
    idx <- which( d$shade_cent==s )
    plot( d$water_cent[idx] , d$blooms_std[idx] , xlim=c(-1,1) , ylim=c(0,1) ,
        xlab="water" , ylab="blooms" , pch=16 , col=rangi2 )
    mtext( concat( "shade = " , s ) )
    mu <- link( m8M4 , post=p , data=data.frame( shade_cent=s , water_cent=-1:1 ) )
    for ( i in 1:20 ) lines( -1:1 , mu[i,] , col=col.alpha("black",0.3) )
}
```

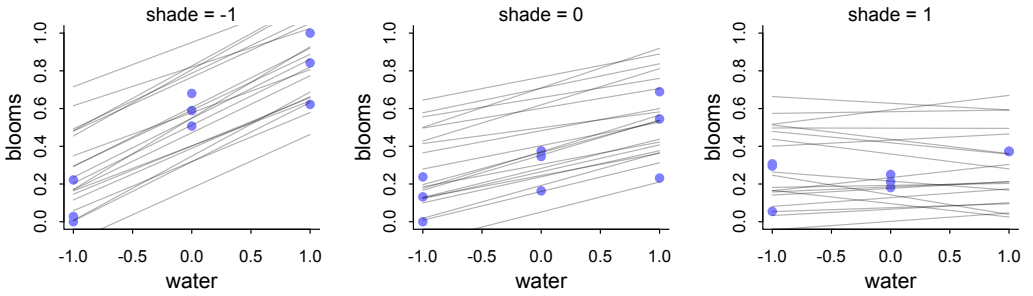


Wow, that's terrible. First guess was no good. We don't want to fit the data visually here, but we do want prior associations within the realm of possibility. So we need to make the prior means closer to zero. Let's try:

R code 8.3

```
m8M4b <- quap(
    alist(
        blooms_std ~ dnorm( mu , sigma ) ,
        mu <- a + bw*water_cent - bs*shade_cent - bws*water_cent*shade_cent ,
        a ~ dnorm( 0.5 , 0.25 ) ,
        bw ~ dlnorm( -2 , 0.25 ) ,
        bs ~ dlnorm( -2 , 0.25 ) ,
        bws ~ dlnorm( -2 , 0.25 ) ,
        sigma ~ dexp( 1 )
    ) , data=d )
```

Extract the prior and repeat the plotting code:



That's a lot better, mostly within the empirically possible range of the outcome at least. There is still plenty of uncertainty for the data to hone down, however.

8H1. Let's begin by loading the data and inspecting the `bed` variable.

R code 8.4

```
library(rethinking)
data(tulips)
d <- tulips
d$bed
```

```
 [1] a a a a a a a a a b b b b b b b b b c c c c c c c c c c
Levels: a b c
```

This is a factor with three levels. So we could either code two dummy variables to contain the same information or rather one index variable. Both approaches were introduced in Chapter 5. I'll show both approaches here, to provide additional examples.

First, the dummy variable approach. We'll need two dummy variables, one less than the number of categories. This will construct them:

R code 8.5

```
d$bed_b <- ifelse( d$bed=="b" , 1 , 0 )
d$bed_c <- ifelse( d$bed=="c" , 1 , 0 )
```

For bed `a`, it will be absorbed into the intercept, and then the coefficients for each dummy variable will contain *contrasts* with bed `a`.

And here is the model using these dummy variables. I'm also going to center the variables, just like in the chapter.

R code 8.6

```
d$blooms_std <- d$blooms / max(d$blooms)
d$water_cent <- d$water - mean(d$water)
d$shade_cent <- d$shade - mean(d$shade)
m1 <- quap(
    alist(
        blooms_std ~ dnorm( mu , sigma ),
        mu <- a + bw*water_cent + bs*shade_cent +
              bws*water_cent*shade_cent +
              b_bed_b*bed_b + b_bed_c*bed_c ,
        a ~ dnorm( 0 , 0.25 ),
        c(bw,bs,bws,b_bed_b,b_bed_c) ~ dnorm(0,0.25),
        sigma ~ dexp( 1 )
    ) , data=d )
precis(m1)
```

```
          mean   sd  5.5% 94.5%
a         0.27 0.03  0.21  0.32
bw        0.21 0.03  0.17  0.25
bs       -0.11 0.03 -0.15 -0.07
bws      -0.14 0.03 -0.19 -0.09
b_bed_b   0.12 0.05  0.04  0.20
b_bed_c   0.14 0.05  0.06  0.22
sigma     0.11 0.01  0.08  0.13
```

This table is a bit monstrous to look at, but comparing to the interaction model fit in the chapter reveals that everything is basically the same, except for the presence now of the bed parameters. Both beds "b" and "c" appear to have done better than bed "a" did. How can I tell? Because both `b_bed_b` and `b_bed_c` are reliably positive.

Now let's do the model over again, using an index variable instead. This approach estimates a unique intercept for each category. To construct the index variable, you can use the convenient `coerce_index` function:

R code 8.7
```
( d$bed_idx <- coerce_index( d$bed ) )
```

```
 [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
```

So now bed "a" has index 1, bed "b" index 2, and bed "c" index 3. To fit the model:

R code 8.8
```
m2 <- quap(
    alist(
        blooms_std ~ dnorm( mu , sigma ),
        mu <- a[bed_idx] + bw*water_cent + bs*shade_cent +
              bws*water_cent*shade_cent ,
        a[bed_idx] ~ dnorm( 0 , 0.25 ),
        c(bw,bs,bws) ~ dnorm( 0 , 0.25 ),
        sigma ~ dexp( 1 )
    ) , data=d )
precis(m2,depth=2)
```

```
       mean   sd  5.5% 94.5%
a[1]   0.26 0.04  0.21  0.32
a[2]   0.39 0.04  0.33  0.44
a[3]   0.40 0.04  0.34  0.46
bw     0.21 0.03  0.17  0.25
bs    -0.11 0.03 -0.15 -0.07
bws   -0.14 0.03 -0.19 -0.09
sigma  0.11 0.01  0.08  0.13
```

These are (nearly) the same estimates. Again we see that beds "b" and "c" did better than bed "a". Bed "c" did appear to grow a little better than bed "b". But what's the posterior distribution of that difference? We can calculate it with samples, just like the examples in Chapter 5:

R code 8.9
```
post <- extract.samples(m2)
diff_b_c <- post$a[,2] - post$a[,3]
PI( diff_b_c )
```

```
         5%         94%
-0.09631392  0.06708758
```

So while the expected difference is there, the posterior distribution of the difference has a lot of probability on both sides of zero.

So what to make of all of this? Including bed in the analysis doesn't change any qualitative inference about the experiment, even though there probably were differences between the beds.

8H2. Now let's compare the model from the previous problem with the interaction model from the chapter.

R code 8.10
```
m3 <- quap(
    alist(
        blooms_std ~ dnorm( mu , sigma ),
        mu <- a + bw*water_cent + bs*shade_cent +
              bws*water_cent*shade_cent ,
        a ~ dnorm( 0 , 0.25 ),
        c(bw,bs,bws) ~ dnorm( 0 , 0.25 ),
        sigma ~ dexp( 1 )
```

```
    ) , data=d )
compare(m2,m3)
```

```
    WAIC    SE dWAIC  dSE pWAIC weight
m2 -22.8 10.16   0.0   NA  10.0   0.57
m3 -22.3 10.18   0.6 8.35   6.4   0.43
```

The model with `bed`, `m2`, does a little bit better in the WAIC comparison. But the difference is very small. Why? Because while there is some evidence that bed mattered, the treatments mattered a lot more. So including `bed` in the model doesn't help prediction much, as least as far as WAIC can estimate. This is all as it should be: this was a factorial experiment. In the experimental design, there is no correlation between bed and treatment. So even when there are effects of bed, we can still get good measures of the treatments. In observational studies, trying to control for common confounds like bed is much more important, typically.

8H3. (a) Let's load the data and fit the model, as in the chapter:

R code 8.11

```
library(rethinking)
data(rugged)
d <- rugged
d$log_gdp <- log(d$rgdppc_2000)
dd <- d[ complete.cases(d$rgdppc_2000) , ]
dd$log_gdp_std <- dd$log_gdp / mean(dd$log_gdp)
dd$rugged_std <- dd$rugged / max(dd$rugged)
dd$cid <- ifelse( dd$cont_africa==1 , 1 , 2 )

m8.3 <- quap(
    alist(
        log_gdp_std ~ dnorm( mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dnorm( 0 , 0.3 ) ,
        sigma ~ dexp( 1 )
    ) , data=dd )
```

Now let's look at the Pareto *k* values, sorted and displayed with the country names:

R code 8.12

```
k <- PSISk( m8.3 )
o <- order( k , decreasing=TRUE )
data.frame( country=as.character( dd$isocode ) ,
    k=k , dd$rugged_std , dd$log_gdp_std )[o,]
```

```
    country     k dd.rugged_std dd.log_gdp_std
145     SYC  0.64  0.7876491454      1.1501264
150     TJK  0.49  0.8547242825      0.7826922
24      BWA  0.40  0.0291841342      1.0507430
118     NPL  0.35  0.8131247985      0.8438960
62      GNQ  0.34  0.0901322154      1.1304719
84      KGZ  0.34  0.6912286359      0.8632546
93      LSO  0.34  1.0000000000      0.8994088
27      CHE  0.28  0.7676555950      1.2110100
167     YEM  0.26  0.3745565946      0.7830325
```

```
55      GAB  0.25  0.0351499516      1.0237147
57      GEO  0.23  0.5899709771      0.8851899
108     MUS  0.22  0.1530151564      1.0768738
144     SWZ  0.22  0.4938729442      0.9922796
...
```

Note that the Pareto *k* values depend upon samples, so the values you see will be a little different. But the basic order should be similar. Seychelles (SYC) does have the largest Pareto *k* value. But several other countries also have pretty high values. What do these countries have in common? That are poorly fit the trend, but in different ways. Seychelles has an unusually high GDP. Tajikstan (TJK) has an unusually low GDP. Botswana (BWA) has an unusually high GDP for such a flat African country.

(b) Simply change the likelihood to Student-t:

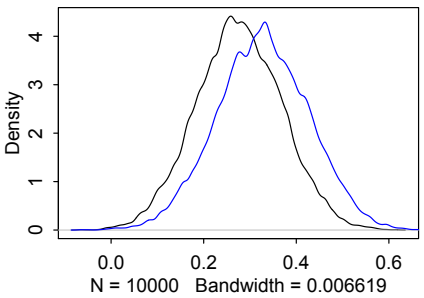
R code 8.13

```
m8.3t <- quap(
    alist(
        log_gdp_std ~ dstudent( 2 , mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dnorm( 0 , 0.3 ) ,
        sigma ~ dexp( 1 )
    ) , data=dd )
```

We want to know how the posterior distribution of the difference between `b[1]` and `b[2]` has changed by using the Student-t distribution. So let's do that comparison directly:

R code 8.14

```
postN <- extract.samples( m8.3 )
postT <- extract.samples( m8.3t )
diffN <- postN$b[,1] - postN$b[,2]
diffT <- postT$b[,1] - postT$b[,2]
dens( diffN )
dens( diffT , add=TRUE , col="blue" )
```



Blue is the Student-t model. The difference has actually grown a little. The thing about the Student-t model is that it treats all the of the extreme points, not just the ones you notice with your eyes. So it can sometimes surprise you.

8H4. (a) You could have chosen to fit a number of different models. I'm going to fit three models that represent a basic analysis of the hypothesis: (1) A model predicting log-lang-per-capita with only a constant, (2) a model with only mean growing season as a predictor, and (3) a model that includes log(area) as a covariate. Then I'll compare them, using WAIC.

But keep in mind that WAIC is not choosing a model for us. The causal theory chooses the model. In this case that means both area and growing season influence the number of languages. These two variables may be associated with one another though other processes. For example, as a consequence of world history, larger national territories (area) may be found in particular places and therefore be associated with particular growing seasons. Since larger nations (area) tend to have fewer languages, there are clearly historical processes at work here. Can you make a DAG that expresses this?

Let's load the data and build some models:

R code 8.15

```
library(rethinking)
data(nettle)
d <- nettle
d$L <- standardize( log( d$num.lang / d$k.pop ) )
d$A <- standardize( log(d$area) )
d$G <- standardize( d$mean.growing.season )

m0 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + 0,
        a ~ dnorm(0,0.2),
        sigma ~ dexp(1)
    ) , data=d )

m1 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + bG*G,
        a ~ dnorm(0,0.2),
        bG ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=d )

m2 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + bG*G + bA*A,
        a ~ dnorm(0,0.2),
        c(bG,bA) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=d )

compare(m0,m1,m2)
```

```
    WAIC    SE dWAIC  dSE pWAIC weight
m2 205.6 16.10   0.0   NA   4.7   0.53
m1 206.0 15.57   0.3 3.90   3.7   0.46
m0 213.7 17.22   8.1 8.09   2.7   0.01
```

This is very strong support for using mean growing season as a predictor of language diversity. Controlling for log(area) has little effect, as can be seen by comparing the estimates from models `m1` and `m2`:

R code 8.16

```
coeftab(m1,m2)
```

```
      m1      m2
```

```
a           0       0
bG       0.34    0.29
sigma    0.92    0.91
bA         NA   -0.17
nobs       74      74
```

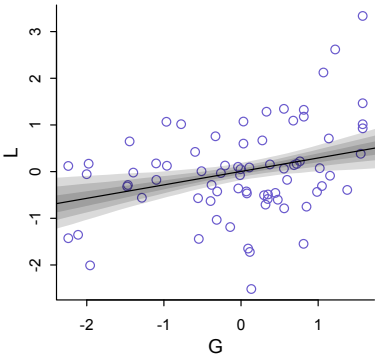
You can check the standard errors, too, to ensure that the estimate for `mean.growing.season` is reliably positive in both cases.

Plotting the predicted relationship for `m2`:

R code 8.17

```
G_seq <- seq( from=-2.5 , to=2 , length.out=30 )
new_dat <- data.frame( A=0 , G=G_seq )

mu <- link( m2 , data=new_dat )

plot( L ~ G , data=d , col="slateblue" )
lines( G_seq , colMeans(mu) )

# plot several intervals with shading
for ( p in c(0.5,0.79,0.95) ) {
mu_PI <- apply( mu , 2 , PI , prob=p )
shade( mu_PI , G_seq )
}
```

And here's the plot:



Hm, that relationship doesn't actually look very linear. Most of the error is above the line, at long growing season values on the right side. In any event, there does seem to be some positive relationship between the two variables, as predicted. Maybe we shouldn't be surprised by how poor the linear relationship represents the data. After all, there are a ton of unmeasured variables that influence language density. For example, imperial expansions have mainly reduced language diversity, but have done so unequally in different regions.

(b) Fitting analogous models and comparing them (`m0` is same as before):

R code 8.18

```
d$S <- standardize( d$sd.growing.season )
m4 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + bS*S,
        a ~ dnorm(0,0.2),
```

```
        bS ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=d )
m5 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + bS*S + bA*A,
        a ~ dnorm(0,0.2),
        c(bS,bA) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=d )
compare(m0,m4,m5)
```

```
    WAIC    SE dWAIC  dSE pWAIC weight
m4 211.0 17.36   0.0   NA   3.7   0.51
m5 211.8 17.21   0.8 3.86   5.4   0.34
m0 213.5 17.11   2.6 4.37   2.5   0.14
```

A very similar story, although not as strong a relationship. Take a look at the marginal posterior from `m5`:

```
precis(m5)
```

R code 8.19

```
       mean   sd  5.5% 94.5%
a      0.00 0.10 -0.15  0.15
bS    -0.14 0.12 -0.34  0.06
bA    -0.19 0.12 -0.39  0.01
sigma  0.94 0.08  0.82  1.06
```

Including log(area) generates considerable uncertainty about the direction of the effect of `sd.growing.season`. Still, the MAP is negative, as predicted. Of course, there is no reason to think the effect of S should be linear, as this model assumes.

(c) Here's the interaction model:

```
m6 <- quap(
    alist(
        L ~ dnorm(mu,sigma),
        mu <- a + bG*G + bS*S + bGS*G*S,
        a ~ dnorm(0,0.2),
        c(bG,bS,bGS) ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=d )
```

R code 8.20

Okay, so let's inspect the marginal posterior:

```
precis(m6)
```

R code 8.21

```
       mean   sd  5.5% 94.5%
a      0.00 0.09 -0.14  0.15
bG     0.23 0.11  0.05  0.41
bS    -0.23 0.10 -0.39 -0.07
bGS   -0.24 0.10 -0.40 -0.08
sigma  0.85 0.07  0.74  0.97
```

You can see above that the interaction coefficient is reliably negative. What that means exactly will require some more work.

Let's see what the predictions look like. Let's show the interaction in a panel of six plots. The top row will show the relationship between log-lang-per-capita and mean.growing.season, across values of sd.growing.season. The bottom row will show log-lang-per-capita on sd.growing.season, across values of mean.growing.season. This is just to show the two-way interaction from both perspectives.

I'm also going to use transparency, as a function of distance from the value on the top of each plot, to show how to data change through the 3rd, un-plotted, dimension. The function `col.dist` in the `rethinking` package handles this for you. I didn't use it in the book, so here is an example of how it can be helpful. You can get more details about how it works from the help `?col.dist`. The key issue is the standard deviation value, which determines how quickly color fades as individual points move away from some reference value. The reference value in these plots will be the value of the third variable displayed on the top margin. You'll want to play with the standard deviation value to get a sense of how it works. Larger values mean less fading.

This code will draw a triptych of interactions, varying `mean.growing.season` along the horizontal axis and `sd.growing.season` across the plots.

R code 8.22

```
# pull out 10%, 50%, and 95% quantiles of sd.growing.season
# these values will be used to make the three plots
S_seq <- quantile(d$S,c(0.1,0.5,0.95))

# now loop over the three plots
# draw languages against mean.growing.season in each
G_seq <- seq(from=-2.5,to=2,length.out=30)
par(mfrow=c(1,3),cex=1.1) # set up plot window for row of 3 plots
for ( i in 1:3 ) {
    S_val <- S_seq[i] # select out value for this plot
    new.dat <- data.frame(
        G = G_seq,
        S = S_val )
    mu <- link( m6 , data=new.dat )
    mu.mean <- apply( mu , 2 , mean )
    mu.PI <- apply( mu , 2 , PI )

    # fade point color as function of distance from sd.val
    cols <- col.dist( d$S , S_val , 2 , "slateblue" )

    plot( L ~ G , data=d , col=cols , lwd=2 )
    mtext( paste("S =",round(S_val,2)) , 3 )
    lines( G_seq , mu.mean )
    shade( mu.PI , G_seq )
}
```

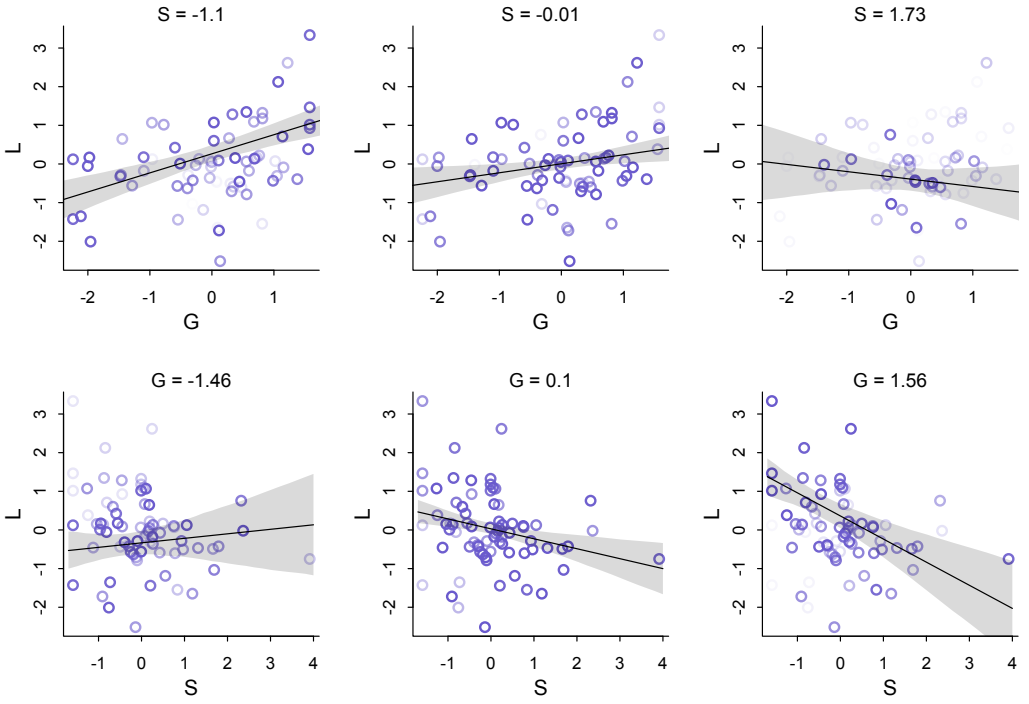
And this code will produce the analogous triptych in which `sd.growing.season` is varied on the horizontal axis.

R code 8.23

```
# pull out 10%, 50%, and 95% quantiles of mean.growing.season
G_seq <- quantile(d$G,c(0.1,0.5,0.95))

# now loop over the three plots
x.seq <- seq(from=-1.7,to=4,length.out=30)
par(mfrow=c(1,3),cex=1.1) # set up plot window for row of 3 plots
for ( i in 1:3 ) {
```

```
    G_val <- G_seq[i] # select out value for this plot
    new.dat <- data.frame(
        G = G_val ,
        S = x.seq )
    mu <- link( m6 , data=new.dat )
    mu.mean <- apply( mu , 2 , mean )
    mu.PI <- apply( mu , 2 , PI )

    # fade point color as function of distance from sd.val
    cols <- col.dist( d$G , G_val , 5 , "slateblue" )

    plot( L ~ S , data=d , col=cols , lwd=2 )
    mtext( paste("G =",round(G_val,2)) , 3 )
    lines( x.seq , mu.mean )
    shade( mu.PI , x.seq )
}
```

And here are both triptych constructions:



So the models suggest that mean growing season increases language diversity, unless the variance in growing season is also high (top row). Simultaneously, variance in growing season decreases language diversity, unless the mean growing season is very short (bottom row).

8H5. Load the data and construct the variables we'll need:

R code 8.24

```
library(rethinking)
data(Wines2012)
```

```
d <- Wines2012

dat_list <- list(
    S = standardize(d$score),
    jid = as.integer(d$judge),
    wid = as.integer(d$wine)
)
```

The model is straightforward. The only issue is the priors. Since I've standardized the outcome, we can use the ordinary N(0,0.5) prior from the examples in the text with standardized outcomes. Then the prior outcomes will stay largely within the possible outcome space. A bit more regularization than that wouldn't be a bad idea either.

R code 8.25

```
m1 <- quap(
    alist(
        S ~ dnorm( mu , sigma ),
        mu <- a[jid] + w[wid],
        a[jid] ~ dnorm(0,0.5),
        w[wid] ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ), data=dat_list )
```

Since this is your first MCMC homework, we'll spend some time inspecting the chains to ensure they worked. First, the diagnostics that `precis` provides:

R code 8.26

```
precis( m1 , 2 )
```

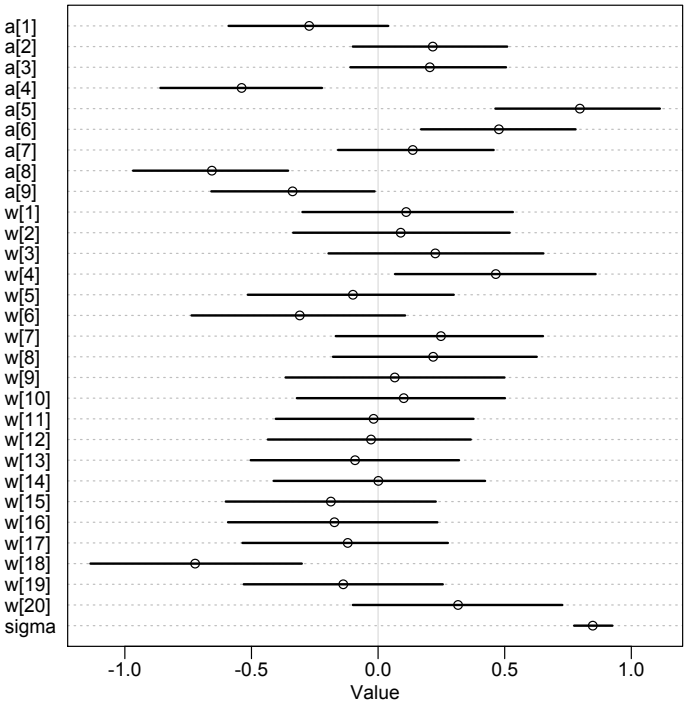
```
       mean   sd  5.5% 94.5%
a[1]  -0.28 0.19 -0.58  0.01
a[2]   0.22 0.19 -0.08  0.51
a[3]   0.21 0.19 -0.09  0.51
a[4]  -0.55 0.19 -0.85 -0.25
a[5]   0.81 0.19  0.51  1.11
a[6]   0.48 0.19  0.19  0.78
a[7]   0.13 0.19 -0.16  0.43
a[8]  -0.67 0.19 -0.97 -0.37
a[9]  -0.35 0.19 -0.65 -0.05
w[1]   0.12 0.25 -0.27  0.51
w[2]   0.09 0.25 -0.30  0.48
w[3]   0.24 0.25 -0.16  0.63
w[4]   0.48 0.25  0.09  0.87
w[5]  -0.11 0.25 -0.50  0.28
w[6]  -0.32 0.25 -0.71  0.07
w[7]   0.25 0.25 -0.14  0.64
w[8]   0.24 0.25 -0.16  0.63
w[9]   0.07 0.25 -0.32  0.46
w[10]  0.10 0.25 -0.29  0.50
w[11] -0.01 0.25 -0.40  0.38
w[12] -0.03 0.25 -0.42  0.37
w[13] -0.09 0.25 -0.48  0.30
w[14]  0.01 0.25 -0.39  0.40
w[15] -0.19 0.25 -0.58  0.20
w[16] -0.17 0.25 -0.57  0.22
```

```
w[17] -0.12 0.25 -0.52  0.27
w[18] -0.75 0.25 -1.14 -0.35
w[19] -0.14 0.25 -0.53  0.25
w[20]  0.33 0.25 -0.06  0.73
sigma  0.79 0.04  0.72  0.85
```

Now let's plot these parameters so they are easier to interpret:

R code 8.27

```
plot( precis( m1 , 2 ) )
```



The a parameters are the judges. Each represents an average deviation of the scores. So judges with lower values are harsher on average. Judges with higher values liked the wines more on average. There is some noticeable variation here. It is fairly easy to tell the judges apart.

The w parameters are the wines. Each represents an average score across all judges. Except for wine 18 (a New Jersey red I think), there isn't that much variation. These are good wines, after all. Overall, there is more variation from judge than from wine.

8H6. The easiest way to code the data is to use indicator variables. Let's look at that approach first. I'll do an index variable version next. I'll use the three indicator variables W (NJ wine), J (American NJ), and R (red wine).

R code 8.28

```
dat_list2 <- list(
    S = standardize(d$score),
    W = d$wine.amer,
    J = d$judge.amer,
    R = ifelse(d$flight=="red",1L,0L)
)
```

The model structure is just a linear model with an ordinary intercept. I'll put a relatively tight prior on the intercept, since it must be near zero (centered outcome). What about the coefficients for the indicator variables? Let's pretend we haven't already seen the results from Problem 1—there aren't any big wine differences to find there. Without that cheating foresight, we should consider what the most extreme effect could be. How big could the difference between NJ and French wines be? Could it be a full standard deviation? If so, then maybe a Normal(0,0.5) prior makes sense, since they place a full standard deviation difference out in the tails of the prior. I'd personally be inclined to something even tighter, so that it regularizes more. But let's go with these wide priors, which nevertheless stay within the outcome space. It would make even more sense to put a tighter prior on the difference between red and white wines—on average they should be the no different, because judges only compare within flights. Here's the model:

R code 8.29

```
m2a <- quap(
    alist(
        S ~ dnorm( mu , sigma ),
        mu <- a + bW*W + bJ*J + bR*R,
        a ~ dnorm( 0 , 0.2 ),
        c(bW,bJ,bR) ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ), data=dat_list2 )
precis( m2a )
```

```
       mean   sd  5.5% 94.5%
a     -0.01 0.12 -0.21  0.18
bW    -0.18 0.13 -0.40  0.04
bJ     0.23 0.13  0.01  0.44
bR    -0.01 0.14 -0.22  0.21
sigma  0.98 0.05  0.90  1.06
```

As expected, red and wines are on average the same—`bR` is right on top of zero. American judges seem to be more on average slightly more generous with ratings—`bJ` is slightly but reliably above zero. American wines have slightly lower average ratings than French wines—`bW` is mostly below zero, but not very large in absolute size.

Okay, now for an index variable version. The thing about index variables is that you can easily end up with more parameters than in an equivalent indicator variable model. But it's still the same posterior distribution. You can convert from one to the other (if the priors are also equivalent). We'll need three index variables:

R code 8.30

```
dat_list2b <- list(
    S = standardize(d$score),
    wid = d$wine.amer + 1L,
    jid = d$judge.amer + 1L,
    fid = ifelse(d$flight=="red",1L,2L)
)
```

Now `wid` is 1 for a French wine and 2 for a NJ wine, `jid` is 1 for a French judge and 2 for an American judge, and `fid` is 1 for red and 2 for white. Those `1L` numbers are just the R way to type the number as an integer—"`1L`" is the integer 1, while "`1`" is the real number 1. We want integers for an index variable.

Now let's think about priors for the parameters that correspond to each index value. Now the question isn't how big the difference could be, but rather how far from the mean an indexed category could be. If we use Normal(0,0.5) priors, that would make a full standard deviation difference from the global mean rare. It will also match what we had above, in a crude sense. Again, I'd be tempted

to something narrow, for the sake of regularization. But certainly something like Normal(0,10) is flat out silly, because it makes impossible values routine. Let's see what we get:

R code 8.31

```
m2b <- quap(
    alist(
        S ~ dnorm( mu , sigma ),
        mu <- w[wid] + j[jid] + f[fid],
        w[wid] ~ dnorm( 0 , 0.5 ),
        j[wid] ~ dnorm( 0 , 0.5 ),
        f[wid] ~ dnorm( 0 , 0.5 ),
        sigma ~ dexp(1)
    ), data=dat_list2b )
precis( m2b , 2 )
```

```
       mean   sd  5.5% 94.5%
w[1]   0.09 0.30 -0.39  0.57
w[2]  -0.09 0.30 -0.57  0.39
j[1]  -0.12 0.30 -0.60  0.36
j[2]   0.12 0.30 -0.36  0.60
f[1]   0.00 0.30 -0.48  0.48
f[2]   0.00 0.30 -0.47  0.48
sigma  0.98 0.05  0.90  1.06
```

To see that this model is the same as the previous, let's compute contrasts. The contrast between American and French wines is:

R code 8.32

```
post <- extract.samples(m2b)
diff_w <- post$w[,2] - post$w[,1]
precis( diff_w )
```

```
'data.frame': 10000 obs. of 1 variables:
        mean   sd  5.5% 94.5%    histogram
diff_w -0.18 0.15 -0.42  0.05 ▁▁▂▅▇▂▁▁
```

That's almost exactly the same mean and standard deviation as `bW` in the first model. The other contrasts match as well.

8H7. I'll use the indicator variable approach here, because it'll be much easier. Once you start using MCMC in the next chapter, it'll be possible to define very flexible parameter structures. Then the index approach will be easy again.

For the indicator approach, we can use the same predictor variables as before:

R code 8.33

```
dat_list2 <- list(
    S = standardize(d$score),
    W = d$wine.amer,
    J = d$judge.amer,
    R = ifelse(d$flight=="red",1L,0L)
)
```

It's the model that is different.

R code 8.34

```
m3 <- quap(
    alist(
```

```
        S ~ dnorm( mu , sigma ),
        mu <- a + bW*W + bJ*J + bR*R +
              bWJ*W*J + bWR*W*R + bJR*J*R,
        a ~ dnorm(0,0.2),
        c(bW,bJ,bR) ~ dnorm(0,0.5),
        c(bWJ,bWR,bJR) ~ dnorm(0,0.25),
        sigma ~ dexp(1)
    ), data=dat_list2 )
```

I used the same priors as before for the main effects. I used tighter priors for the interactions. Why? Because interactions represent sub-categories of data, and if we keep slicing up the sample, differences can't keep getting bigger. Again, the most important thing is not to use flat priors like Normal(0,10) that produce impossible outcomes.

R code 8.35

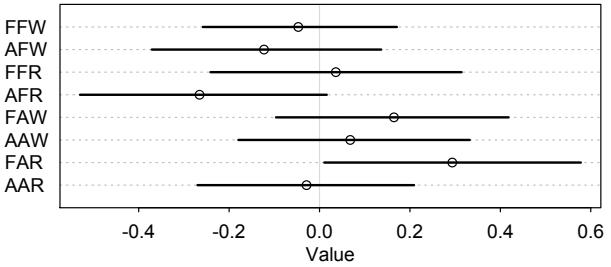
```
precis(m3)
```

```
       mean   sd  5.5% 94.5%
a     -0.05 0.13 -0.25  0.16
bW    -0.07 0.17 -0.35  0.20
bJ     0.21 0.18 -0.08  0.49
bR     0.09 0.18 -0.21  0.38
bWJ   -0.02 0.18 -0.31  0.27
bWR   -0.23 0.18 -0.52  0.06
bJR    0.05 0.18 -0.24  0.34
sigma  0.98 0.05  0.89  1.06
```

Reading the parameters this way is not easy. But right away you might notice that `bW` is now close to zero and overlaps it a lot on both sides. NJ wines are no longer on average worse. So the interactions did something. Glancing at the interaction parameters, you can see that only one of them has much mass away from zero, `bWR`, the interaction between NJ wines and red flight, so red NJ wines. To get the predicted scores for red and white wines from both NJ and France, for both types of judges, we can use `link`:

R code 8.36

```
pred_dat <- data.frame(
    W = rep( 0:1 , times=4 ),
    J = rep( 0:1 , each=4 ),
    R = rep( c(0,0,1,1) , times=2 )
)
mu <- link( m3 , data=pred_dat )
row_labels <- paste( ifelse(pred_dat$W==1,"A","F") ,
                 ifelse(pred_dat$J==1,"A","F") ,
                 ifelse(pred_dat$R==1,"R","W") , sep="" )
plot( precis( list(mu=mu) , 2 ) , labels=row_labels )
```



I've added informative labels. FFW means: French wine, French judge, White wine. So the first four rows are as judged by French judges. The last four are as judged by American judges. The two rows that jump out are the 4th and the 2nd-to-last, AFR and FAR. Those are NJ red wines as judged by French judges and French red wines as judged by American judges. French judges didn't like NJ reds so much (really only one NJ red, if you look back at Problem 1). And American judges liked French reds more. Besides these two interactions, notice that it is very hard to figure this out from the table of coefficients.

9. Chapter 9 Solutions

9E1. Only (3) is required.

9E2. Gibbs sampling requires that we use special priors that are *conjugate* with the likelihood. This means that holding all the other parameters constant, it is possible to derive analytical solutions for the posterior distribution of each parameter. These conditional distributions are used to make smart proposals for jumps in the Markov chain. Gibbs sampling is limited both by the necessity to use conjugate priors, as well as its tendency to get stuck in small regions of the posterior when the posterior distribution has either highly correlated parameters or high dimension.

9E3. Hamiltonian Monte Carlo cannot handle discrete parameters. This is because it requires a smooth surface to glide its imaginary particle over while sampling from the posterior distribution.

9E4. The effect number of samples `n_eff` is an estimate of the number of completely independent samples that would hold equivalent information about the posterior distribution. It is always smaller than the actual number of samples, because samples from a Markov chain tend to sequentially correlated or *autocorrelated*. As autocorrelation rises, `n_eff` gets smaller. At the limit of perfect autocorrelation, for example, all samples would have the same value and `n_eff` would be equal to 1, no matter the actual number of samples drawn.

9E5. `Rhat` should approach 1. How close should it get? People disagree, but it is common to judge that any value less than 1.1 indicates convergence. But like all heuristic indicators, `Rhat` can be fooled.

9E6. A healthy Markov chain should be both *stationary* and *well-mixing*. The first is necessary for inference. The second is desirable, because it means the chain is more efficient. A chain that is both of these things should resemble horizontal noise.

A chain that is malfunctioning, as the problem asks, would not be stationary. THis means it is not converging to the target distribution, the posterior distribution. Examples were provided in the chapter. A virtue of Hamiltonian Monte Carlo is that it makes such chains very obvious: they tend to be rather flat wandering trends. Sometimes they are perfectly flat.

The best test of convergence is always to compare multiple chains. So the best sketch of a malfunctioning trace plot would be one that shows multiple chains wandering into different regions of the parameter space. Figure 8.7 in the chapter, left side, provides an example.

9M1. Here is the model again, now using a uniform prior for `sigma`:

R code 9.1

```
# load and rep data
library(rethinking)
data(rugged)
d <- rugged
d$log_gdp <- log(d$rgdppc_2000)
dd <- d[ complete.cases(d$rgdppc_2000) , ]
```

```
dd$log_gdp_std <- dd$log_gdp / mean(dd$log_gdp)
dd$rugged_std <- dd$rugged / max(dd$rugged)
dd$cid <- ifelse( dd$cont_africa==1 , 1 , 2 )

dat_slim <- list(
    log_gdp_std = dd$log_gdp_std,
    rugged_std = dd$rugged_std,
    cid = as.integer( dd$cid ) )

# new model with uniform prior on sigma
m9.1_unif <- ulam(
    alist(
        log_gdp_std ~ dnorm( mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dnorm( 0 , 0.3 ) ,
        sigma ~ dunif( 0 , 1 )
    ) , data=dat_slim , chains=4 , cores=4 )
```

And here's the model with an exponential prior on `sigma`:

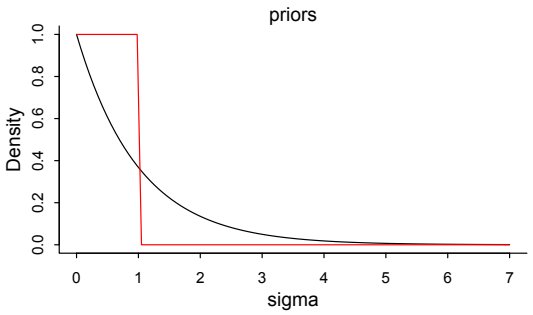
R code 9.2

```
m9.1_exp <- ulam(
    alist(
        log_gdp_std ~ dnorm( mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dnorm( 0 , 0.3 ) ,
        sigma ~ dexp( 1 )
    ) , data=dat_slim , chains=4 , cores=4 )
```

Before we look at the posterior distributions for each model, it may help to visualize each prior. So let's do that first. Maybe the easiest way is to draw random samples from each prior distribution and plot densities. But I'll use the `curve` function instead, just to provide an example of its use.

R code 9.3

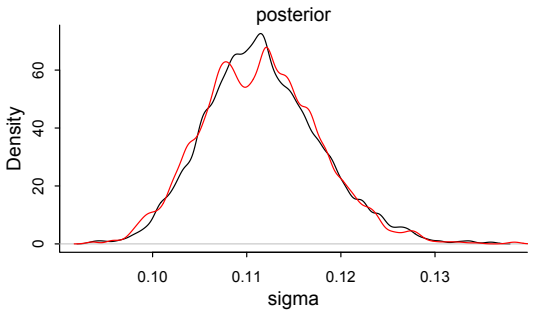
```
curve( dexp(x,1) , from=0 , to=7 ,
    xlab="sigma" , ylab="Density" , ylim=c(0,1) )
curve( dunif(x,0,1) , add=TRUE , col="red" )
mtext( "priors" )
```



Now let's compare the posterior distributions of `sigma` for both models.

R code 9.4

```
post <- extract.samples( m9.1_exp )
dens( post$sigma , xlab="sigma" )
post <- extract.samples( m9.1_unif )
dens( post$sigma , add=TRUE , col="red" )
mtext( "posterior" )
```



The posterior distributions are almost identical. Why? Because there is a lot of data to inform `sigma`. Can you find a prior that won't wash out?

9M2. The model code is easy enough (using the same data as in the previous problem):

R code 9.5

```
m9M2 <- ulam(
    alist(
        log_gdp_std ~ dnorm( mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dexp( 0.3 ) ,
        sigma ~ dexp( 1 )
    ) , data=dat_slim , chains=4 , cores=4 )
precis( m9M2 , 2 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
a[1]  0.89 0.02 0.86  0.91  1519     1
a[2]  1.05 0.01 1.03  1.07  1435     1
b[1]  0.15 0.07 0.03  0.27  1201     1
b[2]  0.02 0.02 0.00  0.05  1779     1
sigma 0.11 0.01 0.10  0.12  1344     1
```

The big difference here is `b[2]`. In the original model, the mass of this parameter is almost entirely below zero. Now it cannot go below zero, because the prior is not defined below zero. So instead the mass presses up against zero tightly.

9M3. You could use almost any model from the chapter. But I'll use the terrain ruggedness model again, since it was used in the other problems so far. I'll fix it at 1000 post-warmup samples for the sake of comparison. Then I'll compare a range of warmup values.

Here's some code to automate it all. You can do the same thing the hard way, just recoding the model each time. But it's a lot faster to compile the model once and reuse it.

R code 9.6

```
# compile model
# use fixed start values for comparability of runs
start <- list(a=c(1,1),b=c(0,0),sigma=1)
m9.1 <- ulam(
    alist(
        log_gdp_std ~ dnorm( mu , sigma ) ,
        mu <- a[cid] + b[cid]*( rugged_std - 0.215 ) ,
        a[cid] ~ dnorm( 1 , 0.1 ) ,
        b[cid] ~ dnorm( 0 , 0.3 ) ,
        sigma ~ dexp( 1 )
    ) , start=start , data=dat_slim , chains=1 , iter=1 )

# define warmup values to run through
warm_list <- c(5,10,100,500,1000)

# first make matrix to hold n_eff results
n_eff <- matrix( NA , nrow=length(warm_list) , ncol=5 )

# loop over warm_list and collect n_eff
for ( i in 1:length(warm_list) ) {
    w <- warm_list[i]
    m_temp <- ulam( m9.1 , chains=1 ,
        iter=1000+w , warmup=w , refresh=-1 , start=start )
    n_eff[i,] <- precis(m_temp,2)$n_eff
}

# add some names to rows and cols of result
# just to make it pretty
# there's always time to make data pretty
colnames(n_eff) <- rownames(precis(m_temp,2))
rownames(n_eff) <- warm_list
```

If you inspect the matrix `n_eff` now, you'll see parameters in columns and warmup values in rows:

```
       a[1]   a[2]   b[1]   b[2]  sigma
5     101.8   59.2    7.4   23.8   30.0
10   1247.2 1224.3  675.5  901.8  781.0
100  1404.6 1094.2  511.9  752.9  761.5
500  1940.5  962.6 1288.4  901.1 1199.9
1000 1072.3 1689.1 1905.9 1340.3 1295.6
```

So you can see that for this model and this data, very little warmup is needed. Basically anything more than 10 is the same. This isn't always going to be true, however. For the more complicated non-linear models that your'll meet in later chapters, and especially multilevel models, more warmup is often helpful. How much? It depends, unfortunately. Models and data are just too diverse to give useful general advice here.

But I can show you a common situation that benefits from more warmup. Let's go back to the leg data example from Chapter 6:

R code 9.7

```
N <- 100                          # number of individuals
height <- rnorm(N,10,2)           # sim total height of each
leg_prop <- runif(N,0.4,0.5)      # leg as proportion of height
leg_left <- leg_prop*height +     # sim left leg as proportion + error
    rnorm( N , 0 , 0.02 )
```

```
leg_right <- leg_prop*height +    # sim right leg as proportion + error
    rnorm( N , 0 , 0.02 )
                                  # combine into data frame
d <- data.frame(height,leg_left,leg_right)
```

And now to run these data through the same process, using the model from Chapter 5 as well:

R code 9.8

```
m <- ulam(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + bl*leg_left + br*leg_right ,
        a ~ dnorm( 10 , 100 ) ,
        bl ~ dnorm( 2 , 10 ) ,
        br ~ dnorm( 2 , 10 ) ,
        sigma ~ dunif( 0 , 10 )
    ) , data=d , iter=1 )

warm_list <- c(5,10,100,500,1000)
n_eff <- matrix( NA , nrow=length(warm_list) , ncol=4 )
for ( i in 1:length(warm_list) ) {
    w <- warm_list[i]
    m_temp <- ulam( m , chains=1 ,
        iter=1000+w , warmup=w , refresh=-1 )
    n_eff[i,] <- precis(m_temp,2)$n_eff
}
colnames(n_eff) <- rownames(precis(m_temp,2))
rownames(n_eff) <- warm_list
round(n_eff,1)
```

```
         a    bl    br sigma
5      8.3   3.2   3.0  54.9
10    65.7   6.7   6.7  70.0
100  515.9  92.2  92.5 676.3
500  538.6 292.8 295.0 552.4
1000 488.4 306.5 307.4 567.7
```

This time, even with a simpler model and fewer observations, it took longer for the benefits of more warmup to taper off. Efficiency improved all the way up to 1000 warmup steps, at least for the parameters `bl` and `br`. Why? Because in this posterior, the parameters `bl` and `br` are highly correlated, recall. Take a look at `pairs(m_temp)` for example. This makes it harder to sample efficiently from the posterior.

In complex multilevel models, there are nearly always batches of highly correlated parameters. So being conservative about warmup often pays off.

9H1. What this code does is sample from the priors. There is no likelihood, and that's okay. The posterior distribution is then just a merger of the priors. What is tricky about this problem though is that the Cauchy prior for the parameter `b` will not produce the kind of trace plot you might expect from a good Markov chain. This is because Cauchy is a very long tailed distribution, so it'll occasionally make distance leaps out into the tail. We'll look at the `precis` output, then the trace plot, so you can see what I mean.

Run the provided code, then inspect the marginal posterior:

R code 9.9

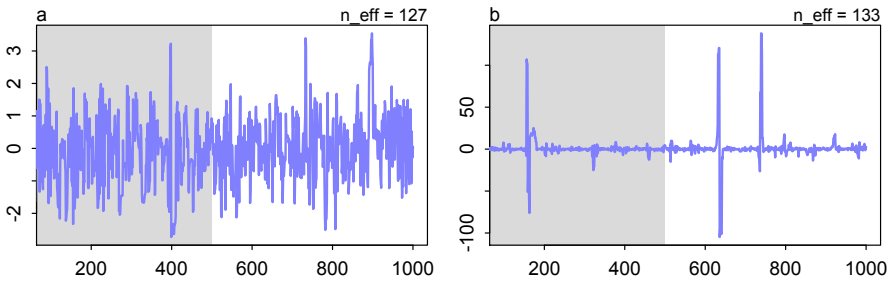
```
precis(mp)
```

```
  mean    sd  5.5% 94.5% n_eff Rhat4
a 0.05  0.97 -1.38  1.50   127  1.01
b 1.01 19.09 -5.83  8.03   133  1.00
```

Now let's look at the trace plot, which is what has alarmed students in the past, when I have assigned this problem as homework:

R code 9.10

```
traceplot( mp , n_col=2 , lwd=2 )
```



The trace plot might look a little weird to you, because the trace for `b` has some big spikes in it. That's how a Cauchy behaves, though. It has thick tails, so needs to occasionally sample way out. The trace plot for `a` is typical Gaussian in shape. Since the posterior distribution does often tend towards Gaussian for many parameters, it's possible to get too used to expecting every trace to look like the one on the left. But you have to think about the influence of priors in this case. The trace on the right is just fine.

9H2. First, load and prepare the data.

R code 9.11

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
d$D <- standardize( d$Divorce )
d$M <- standardize( d$Marriage )
d$A <- standardize( d$MedianAgeMarriage )
d_trim <- list(D=d$D,M=d$M,A=d$A)
```

Now to fit the models over again, this time using `ulam`. Note that we need to add `log_lik=TRUE` to get the terms needed to compute PSIS or WAIC.

R code 9.12

```
m5.1_stan <- ulam(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- a + bA * A ,
        a ~ dnorm( 0 , 0.2 ) ,
        bA ~ dnorm( 0 , 0.5 ) ,
        sigma ~ dexp( 1 )
    ) , data=d_trim , chains=4 , cores=4 , log_lik=TRUE )
m5.2_stan <- ulam(
    alist(
```

```
        D ~ dnorm( mu , sigma ) ,
        mu <- a + bM * M ,
        a ~ dnorm( 0 , 0.2 ) ,
        bM ~ dnorm( 0 , 0.5 ) ,
        sigma ~ dexp( 1 )
    ) , data=d_trim , chains=4 , cores=4 , log_lik=TRUE )
m5.3_stan <- ulam(
    alist(
        D ~ dnorm( mu , sigma ) ,
        mu <- a + bM*M + bA*A ,
        a ~ dnorm( 0 , 0.2 ) ,
        bM ~ dnorm( 0 , 0.5 ) ,
        bA ~ dnorm( 0 , 0.5 ) ,
        sigma ~ dexp( 1 )
    ) , data=d_trim , chains=4 , cores=4 , log_lik=TRUE )
```

Since you are possibly still getting used to Markov chains, I recommend checking the trace and trank plots for each model, as well as inspecting the `n_eff` and `Rhat` values for each parameter in each model.

Now to compare the models:

R code 9.13

```
compare( m5.1_stan , m5.2_stan , m5.3_stan , func=PSIS )
```

```
Some Pareto k values are high (>0.5)
           PSIS    SE dPSIS  dSE pPSIS weight
m5.1_stan 125.9 12.89   0.0   NA   3.7   0.72
m5.3_stan 127.8 13.00   1.8 0.70   4.8   0.28
m5.2_stan 139.3  9.94  13.4 9.33   3.0   0.00
```

And WAIC is quite similar:

R code 9.14

```
compare( m5.1_stan , m5.2_stan , m5.3_stan , func=WAIC )
```

```
           WAIC    SE dWAIC  dSE pWAIC weight
m5.1_stan 125.7 12.61   0.0   NA   3.6   0.71
m5.3_stan 127.5 12.73   1.8 0.70   4.7   0.29
m5.2_stan 139.2  9.81  13.5 9.17   2.9   0.00
```

The model with only age-at-marriage comes out on top, although the model with both predictors does nearly as well. In fact, the PSIS/WAIC of both models is nearly identical. I'd call this is a tie, because even though one model does a bit better than the other, the difference between them is of no consequence. How can we explain this? Well, look at the marginal posterior for `m5.3_stan`:

R code 9.15

```
precis(m5.3_stan)
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
a      0.00 0.10 -0.16  0.16  1561     1
bM    -0.06 0.15 -0.30  0.18  1012     1
bA    -0.61 0.15 -0.87 -0.36  1072     1
sigma  0.83 0.09  0.70  0.98  1563     1
```

While this model includes marriage rate as a predictor, it estimates very little expected influence for it, as well as substantial uncertainty about the direction of any influence it might have. So models `m5.3_stan` and `m5.1_stan` make practically the same predictions. After accounting for the larger penalty for `m5.3_stan`—4.7 instead of 3.7—the two models rank almost the same. This makes sense,

because you already learned back in Chapter 5 that marriage rate probably gets its correlation with divorce rate through a correlation with age at marriage. So even though including marriage rate in a model doesn't really aid in prediction, there is enough evidence here that the parameter `bR` can be estimated well enough, and including marriage rate doesn't hurt prediction either.

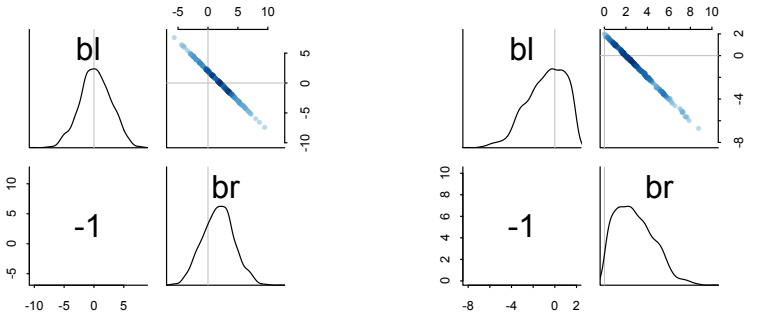
Or at least that's what PSIS/WAIC expects. Only the future will tell which model is actually better for forecasting. PSIS/WAIC is not an oracle. It's a golem.

9H3. Here is the simulation code again, just copied from the chapter:

R code 9.16

```
N <- 100                          # number of individuals
set.seed(909)
height <- rnorm(N,10,2)           # sim total height of each
leg_prop <- runif(N,0.4,0.5)      # leg as proportion of height
leg_left <- leg_prop*height +     # sim left leg as proportion + error
    rnorm( N , 0 , 0.02 )
leg_right <- leg_prop*height +    # sim right leg as proportion + error
    rnorm( N , 0 , 0.02 )
                                  # combine into data frame
d <- data.frame(height,leg_left,leg_right)
```

Run the `ulam` models in the prompt, and then we can look at the `pairs` plot for each model, which is probably the quickest way to see the impact of the change in prior. The posterior distributions for bl and br are symmetric, and perfectly negatively correlated, in `m5.8s`. In the other model, with the truncated prior, they are still perfectly negatively correlated, but now they look like mirror images, one being skewed left and the other right. See below (left: `m5.8s`; right: `m5.8s2`).



What has happened is that the posterior distributions are necessarily negatively correlated with one another. That arises because the left and right legs contain the same information. So this is the lack of identifiability thing returning. So when we change the prior on one of the parameters, that information has to cascade into the posterior distribution of the other parameter as well. Otherwise the negative posterior correlation wouldn't be maintained.

9H4. Inspecting WAIC for the two models, you get:

R code 9.17

```
compare( m5.8s , m5.8s2 )
```

```
        WAIC    SE dWAIC  dSE pWAIC weight
m5.8s2 194.2 11.27     0   NA   2.7   0.62
m5.8s  195.2 11.16     1 0.72   3.2   0.38
```

This is a good tie. You'll get slightly different numbers, on account of simulation variance. Note that the model with the truncated prior is less flexible, as indicated by `pWAIC`. Why? Because the prior is more informative, the variance in the posterior distribution is smaller. But the same model, `m5.8s2`, also fits the sample worse, as a consequence of the more informative prior.

9H5. To do as the problem asks, we'll need a new vector for the population sizes of the islands. Let's call it `pop_size` and give it randomly shuffled values from 1 to 10:

R code 9.18

```
pop_size <- sample( 1:10 )
```

Now most of the same code will work, but we'll need to extract two elements of `pop_size` when we construct the probability of moving. Why? Because its population size that matters when deciding to move, not each island's position. Now that size and position are not the same numbers, we just have use indexing to translate from position to population size. Here's how it works. The new code is at the end, where `prob_move` is calculated. And I've added the line above to the start, as well.

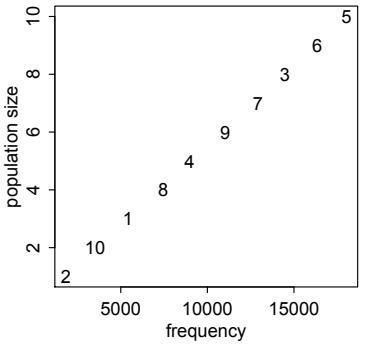
R code 9.19

```
num_weeks <- 1e5
positions <- rep(0,num_weeks)
pop_size <- sample( 1:10 )
current <- 10
for ( i in 1:num_weeks ) {
    # record current position
    positions[i] <- current

    # flip coin to generate proposal
    proposal <- current + sample( c(-1,1) , size=1 )
    # now make sure he loops around the archipelago
    if ( proposal < 1 ) proposal <- 10
    if ( proposal > 10 ) proposal <- 1

    # move?
    prob_move <- pop_size[proposal]/pop_size[current]
    current <- ifelse( runif(1) < prob_move , proposal , current )
}
```

To verify this is working as intended, compare the frequency in the samples of each island to each island's population size:

R code 9.20

```
# compute frequencies
f <- table( positions )

# plot frequencies against relative population sizes
# label each point with island index
plot( as.vector(f) , pop_size , type="n" ,
    xlab="frequency" , ylab="population size" ) # empty plot
text(  , x=f , y=pop_size )
```



Each island appears in the samples in direct proportion to its population size. Run the code a few times and watch the index values shuffle around.

In a typical Metropolis context, remember, the island index values are parameter values and the population sizes are posterior probabilities.

9H6. Okay this is one of the hardest problems in the book. Why? Because it is asking a lot. There wasn't an example in the chapter of actually using the Metropolis algorithm with data and a model. So converting the silly island hopping example to a practical model fitting example is a huge challenge.

Really this problem is a way to sneak in this extra content, as actually writing Markov chains is beyond the scope of the book. But if you are curious, here's one solution.

Here's the raw data for the globe tossing example:

$$\text{W L W W W L W L W}$$

That's 6 waters in 9 tosses. The likelihood is binomial, and we'll use a uniform prior, as in Chapter 2. So this is the model:

$$w \sim \text{Binomial}(n, p)$$
$$p \sim \text{Uniform}(0, 1)$$

where w is the observed number of water and n is the number of globe tosses. The parameter p is the target of inference. We need a posterior distribution for it. The prior distribution is the given uniform density form zero to one.

To get a working Metropolis algorithm for this model, think of the different values of p as the island indexes and the product of the likelihood and prior as the population sizes. What is tricky here is that there an infinite number of islands: every continuous value that p can take from zero to one. That's hardly a problem, though. We just need a different way of proposing moves, so that we can land on a range of islands. It'll make more sense, once you see the code.

R code 9.21

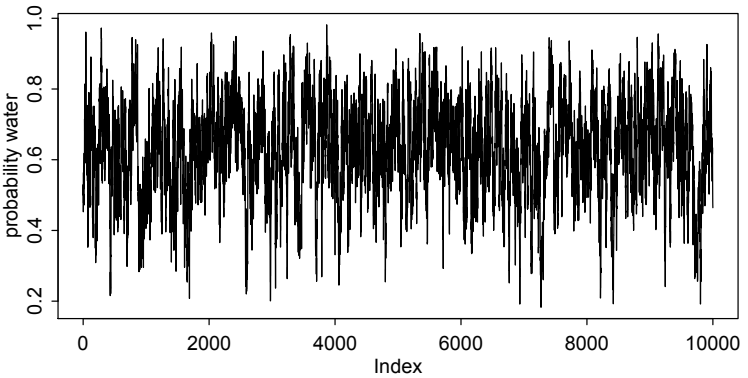
```
num_samples <- 1e4
p_samples <- rep(NA,num_samples)
p <- 0.5 # initialize chain with p=0.5
for ( i in 1:num_samples ) {
    # record current parameter value
    p_samples[i] <- p

    # generate a uniform proposal from -0.1 to +0.1
    proposal <- p + runif(1,-0.1,0.1)
    # now reflect off boundaries at 0 and 1
    # this is needed so proposals are symmetric
```

```
    if ( proposal < 0 ) proposal <- abs(proposal)
    if ( proposal > 1 ) proposal <- 1-(proposal-1)

    # compute posterior prob of current and proposal
    prob_current <- dbinom(6,size=9,prob=p) * dunif(p,0,1)
    prob_proposal <- dbinom(6,size=9,prob=proposal) * dunif(proposal,0,1)

    # move?
    prob_move <- prob_proposal/prob_current
    p <- ifelse( runif(1) < prob_move , proposal , p )
}
```

That's really all there is to it. Once the loop finishes—it'll be very very fast—take a look at the trace plot:

R code 9.22

```
plot( p_samples , type="l" , ylab="probability water" )
```

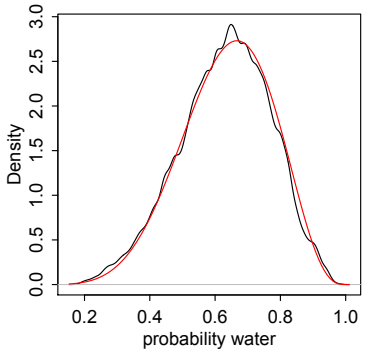


This chain is fine, but it is more autocorrelated than a typical Stan chain. That's why it tends to wander around a bit. It is stationary, but it isn't mixing very well. This is common of such a simple Metropolis chain. But it is working.

Now look at the posterior distribution implied by these samples. I'll also compare it to the analytically derived posterior for this model:

R code 9.23

```
dens( p_samples , xlab="probability water" )
curve( dbeta(x,7,4) , add=TRUE , col="red" )
```



Not bad.

So what would you do if the model had more than one parameter? You just add another proposal for each parameter, accepting or rejecting each proposal independent of the others. Suppose for example we want to do a simple linear regression with a Metropolis chain. Let's simulate some simple Gaussian data and then compute the posterior distribution of the mean and standard deviation using a custom Metropolis algorithm. The model is:

$$y_i \sim \text{Normal}(\mu, \sigma)$$
$$\mu \sim \text{Normal}(0, 10)$$
$$\sigma \sim \text{Uniform}(0, 10)$$

This is the code:

R code 9.24

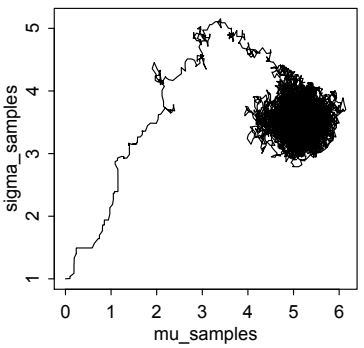
```
# simulate some data
# 100 observations with mean 5 and sd 3
y <- rnorm( 100 , 5 , 3 )

# now chain to sample from posterior
num_samples <- 1e4
mu_samples <- rep(NA,num_samples)
sigma_samples <- rep(NA,num_samples)
mu <- 0
sigma <- 1
for ( i in 1:num_samples ) {
    # record current parameter values
    mu_samples[i] <- mu
    sigma_samples[i] <- sigma

    # proposal for mu
    mu_prop <- mu + runif(1,-0.1,0.1)

    # compute posterior prob of mu and mu_prop
    # this is done treating sigma like a constant
    # will do calculations on log scale, as we should
    # so log priors get added to log likelihood
    log_prob_current <- sum(dnorm(y,mu,sigma,TRUE)) +
                        dnorm(mu,0,10,TRUE) + dunif(sigma,0,10,TRUE)
    log_prob_proposal <- sum(dnorm(y,mu_prop,sigma,TRUE)) +
                        dnorm(mu_prop,0,10,TRUE) + dunif(sigma,0,10,TRUE)
    # move?
    prob_move <- exp( log_prob_proposal - log_prob_current )
```

```
    mu <- ifelse( runif(1) < prob_move , mu_prop , mu )

    # proposal for sigma
    sigma_prop <- sigma + runif(1,-0.1,0.1)
    # reflect off boundary at zero
    if ( sigma_prop < 0 ) sigma_prop <- abs(sigma_prop)

    # compute posterior probabilities
    log_prob_current <- sum(dnorm(y,mu,sigma,TRUE)) +
                        dnorm(mu,0,10,TRUE) + dunif(sigma,0,10,TRUE)
    log_prob_proposal <- sum(dnorm(y,mu,sigma_prop,TRUE)) +
                        dnorm(mu,0,10,TRUE) + dunif(sigma_prop,0,10,TRUE)
    # move?
    prob_move <- exp( log_prob_proposal - log_prob_current )
    sigma <- ifelse( runif(1) < prob_move , sigma_prop , sigma )
}
```

This code is more complex, but it is really the same strategy, just with two internal steps, one for each parameter. You can process `mu_samples` and `sigma_samples` as usual. I'll show the sequential samples plotted together now, so you can see how the chain wanders into the high probability region of the posterior as the chain evolves:

R code 9.25

```
plot( mu_samples , sigma_samples , type="l" )
```



We initialized the chain at $(0, 1)$. It then quickly wandered up and to the right, eventually falling into the orbit of the high density region of the posterior. That long trail into the high density region is often called "burn in" and usually trimmed off before analysis. You don't have to do such trimming with Stan chains, because Stan's warm-up phase takes care of a similar task, and Stan only returns post-warmup samples to you.

9H7. Following the example in the box starting on page 276, we need to write functions for both the log-probability of the data, the `U` function, and its gradient, the `U_gradient` function.

The log-probability is nothing new. We did it in the previous problem. Now we'll build it into its own function:

R code 9.26

```
# y is successes, n is trials
U_globe <- function( q ) {
```

```
    U <- dbinom(y,size=n,prob=q,log=TRUE) + dunif(q,0,1,log=TRUE)
    return( -U )
}
```

The gradient requires some thought. We need to compute:

$$\frac{\partial U(y|n,p)}{\partial p}$$

where y is the observed count 6 and $n = 9$. This is made a bit easier by the fact that we used a uniform prior, which has a constant gradient. So we can omit the prior in this case. To get the derivative above, you can either aggressively apply the chain rule, or plug the binomial log-probability expression into a symbolic system like Mathematica and get the answer:

$$\frac{\partial \log U(y|n,p)}{\partial p} = \frac{y - np}{p(1-p)}$$

And this gives us the gradient function:

R code 9.27

```
U_globe_gradient <- function( q ) {
    G <- ( y - n*q )/( q*(1-q) )
    return( -G )
}
```

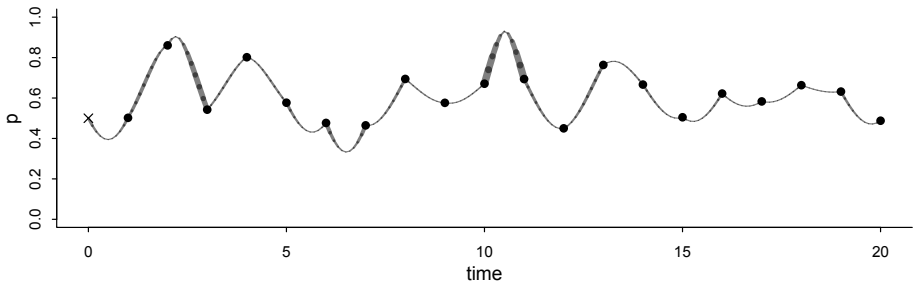
Now we can modify the code in Rcode 9.7 on page 277 to run the HMC simulation. This example has only one dimension, p, so instead of the fancy 2D plot of trajectories, I'll plot time on the horizontal axis and the position on the vertical. All this code does really is loop and call `HMC2`, feeding the result back in each time to get a new trajectory. The samples are stored in `samples`.

R code 9.28

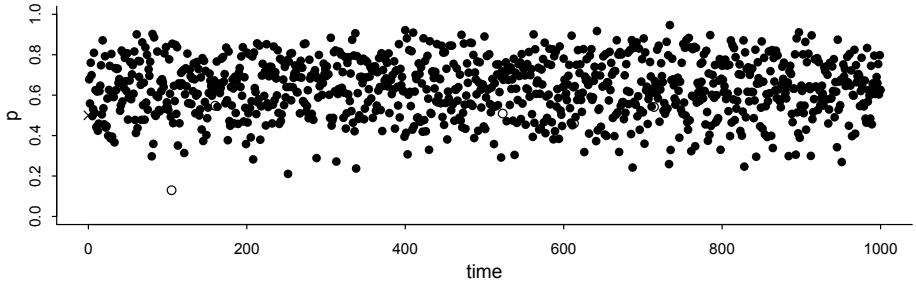
```
# data
y <- 6
n <- 9

# initialize everything
Q <- list()
Q$q <- 0.5
n_samples <- 20
plot( NULL , xlab="time" , ylab="p" , ylim=c(0,1) , xlim=c(0,n_samples) )
path_col <- col.alpha("black",0.5)
points( 0 , Q$q , pch=4 , col="black" )

step <- 0.03
L <- 10

samples <- rep(NA,n_samples)

show_trajectory <- TRUE
for ( i in 1:n_samples ) {
    Q <- HMC2( U_globe , U_globe_gradient , step , L , Q$q )
    # plot trajectory
    if ( show_trajectory==TRUE )
      for ( j in 1:L ) {
        K0 <- sum(Q$ptraj[j,]^2)/2 # kinetic energy
        tx <- (i-1) + c( 1/L*(j-1) , 1/L*j )
        lines( tx , Q$traj[j:(j+1),1] , col=path_col , lwd=1+2*K0 )
```

```
      }
    points( i , Q$traj[L+1,1] , pch=ifelse( Q$accept==1 , 16 , 1 ) )
    if ( Q$accept==1 ) samples[i] <- Q$q
}
```

Running this, you should get something like:



Increase the number of samples to, say, 1000 (you might want to disable plotting the trajectories):



The open points are divergent trajectories. They aren't stored in `samples`, but they do imply that we could tune the step size a bit to do better. See if you can tune the divergent trajectories out.

11. Chapter 11 Solutions

11E1. If $p = 0.35$, then log-odds are:

$$\log \frac{0.35}{1 - 0.35}$$

Calculating in R:

```
log( 0.35 / (1-0.35) )
```

R code 11.1

```
[1] -0.6190392
```

11E2. This one is asking for the reverse operation, going from log-odds to probability. The `logistic` and `inv_logit` functions accomplish this. These are just two names for the same function.

```
logistic( 3.2 )
```

R code 11.2

```
[1] 0.9608343
```

11E3. Proportional odds is calculated by exponentiating the coefficient. So:

```
exp( 1.7 )
```

R code 11.3

```
[1] 5.473947
```

This means that each unit change in the predictor variable multiplies the odds of the event by 5.5.

To demystify this relationship a little, if the linear model L is the log-odds of the event, then the odds of the event are just $\exp(L)$. Now we want to compare the odds before and after increasing a predictor by one unit. We want to know how much the odds increase, as a result of the unit increase in the predictor. We can use our dear friend algebra to solve this problem:

$$\exp(\alpha + \beta x)Z = \exp(\alpha + \beta(x + 1))$$

The left side is the odds of the event, before increasing x. The Z represents the proportional change in odds that we're going to solve for. It's unknown value will make the left side equal to the right side. The right side is the odds of the event, after increasing x by 1 unit. So we just solve for Z now. The answer is $Z = \exp(\beta)$. And that's where the formula comes from.

11E4. An *offset* specifies the relative duration or distance that a count was accumulated over. It may also be called an *exposure*. So if all observed counts were from uniform observation periods, then there is no need to use an offset. But if instead some observations came from longer periods than others, you can use an offset to adjust estimates for the fact that we expect counts from longer periods to be bigger.

11M1. The most basic reason is that aggregated binomial counts have to average over all of the orders, or permutations, that are consistent with the observed count. Disaggregated binomial counts, in 0/1 form, do not have to cope with order. So for example, if we flip 2 coins and observe one head and one tail, this is a count of 1 head in 2 trials. As aggregated data, the probability is:

$$\frac{2!}{1!1!}p(1-p) = 2p(1-p)$$

where p is the probability of a head on each trial. The fraction in front is the multiplicity (same as what was used in Chapter 9 to derive maximum entropy). It just says how many ways to get 1 head from 2 coins, in any order. But as disaggregated data, we instead just predict each coin separately and then multiply them together to get the joint probability of the data. So:

$$p(1-p)$$

is the entire likelihood. So the aggregated data has an extra constant in front to handle all the permutations. This doesn't influence inference, because the multiplicity constant isn't a function of the parameter p. But it does influence the magnitude of the likelihood and log-likelihood.

11M2. This problem is a prompt to realize that coefficients in Poisson models are not so easy to interpret. But we can figure out the change in the mean count with a little algebra. A Poisson model typically has a log link. So in a linear model, the mean count λ is related to the linear model by:

$$\begin{aligned}\log(\lambda) &= \alpha + \beta x \\ \lambda &= \exp(\alpha + \beta x)\end{aligned}$$

So suppose that $\beta = 1.7$, as the problem states. If x increases by a unit, what happens to λ? We can compute the change in λ:

$$\begin{aligned}\Delta\lambda &= \exp(\alpha + \beta(x+1)) - \exp(\alpha + \beta x) \\ &= \exp(\alpha + \beta x)(\exp(\beta) - 1)\end{aligned}$$

Hm, not so simple. The change depends upon the entire linear model, still. This is the rule in non-linear models.

What about the ratio of means, though? So define λ_x as the mean before the change and λ_{x+1} as the mean after x increases by a unit. Then the ratio of the means is:

$$\frac{\lambda_{x+1}}{\lambda_x} = \frac{\exp(\alpha + \beta(x+1))}{\exp(\alpha + \beta x)} = \exp(\beta)$$

This is just like the proportional change in odds we found with logistic regression. The proportional change in the expectation for a Poisson model is given by exponentiating a coefficient.

So finally we can arrive at a firm, if not always useful, way to address the question of what a coefficient of 1.7 means. It means that the proportional change in the expected count will be $\exp(1.7) = 5.5$, when the corresponding predictor increases by one unit.

11M3. It is conventional to use a logit link for a binomial GLM because we need to map the continuous linear model value to a probability parameter that is bounded between zero and one. The inverse-logit function, often known as the *logistic*, is one way to do this.

There are deeper reasons for using the logistic. It arises naturally when working with multinomial probability densities. There was a hint of this in one of the Overthinking boxes in Chapter 9, in which you saw how to derive when the binomial distribution has maximum entropy.

11M4. It is conventional to use a log link for a Poisson GLM because we need to map the continuous linear model value to a mean that must be positive. In other words, the mean of a Poisson is bounded at zero. The inverse function of log is exp, so exponentiating the linear model guarantees it will be positive.

The log link for a Poisson model can also be justified by factoring the Poisson probability density formula. This is the way so-called *canonical* link functions were sometimes defined. But the log link is also used routinely with exponential and gamma GLMs, for which there is nothing "canonical" about the log link at all. The canonical link for the gamma and exponential is the inverse function, which is bad news because it fails to constrain the mean to be positive.

11M5. The problem suggests using a logit link in a Poisson model, like this:

$$y_i \sim \text{Poisson}(\mu_i)$$
$$\text{logit}(\mu_i) = \alpha + \beta x_i$$

This would bound the mean μ to lie between zero and one. With the addition of one more parameter however, it could bound it to lie between zero and any arbitrary maximum:

$$\log \frac{\mu_i}{M - \mu_i} = \alpha + \beta x_i$$

where M is a parameter that determines the maximum expected count.

This sort of link looks funny. In practice you never see it, because if a count variable can reach a maximum, it is usually more appropriate to use a binomial likelihood together with the logit link. Remember, the premise with a Poisson likelihood is that it is really binomial, but the probability is very low and the number of trials very large. So any theoretical maximum count is never reached in the data.

Using the logit link with a Poisson could make sense if you have reason to think that the influence of predictors on the mean diminishes eventually. That is, if you want to stop the exponential growth.

11M6. The constraints that make both binomial and Poisson maximum entropy distributions are: (1) discrete binary outcomes, (2) constant probability of each event across trials (or constant expected value). These two distributions have the same constraints, because the Poisson is just a simplified form of the binomial that applies when the probability of the focal event is very low and the number of trials is very large.

11M7. Here is the `quap` model:

R code 11.4

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$treatment <- 1 + d$prosoc_left + 2*d$condition
dat_list <- list(
    pulled_left = d$pulled_left,
    actor = d$actor,
    treatment = as.integer(d$treatment) )

m11.4q <- quap(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + b[treatment] ,
```

```
        a[actor] ~ dnorm( 0 , 1.5 ),
        b[treatment] ~ dnorm( 0 , 0.5 )
    ) , data=dat_list )
```

Let's put the posterior summaries from both models into the same table, so they can be easily compared:

R code 11.5

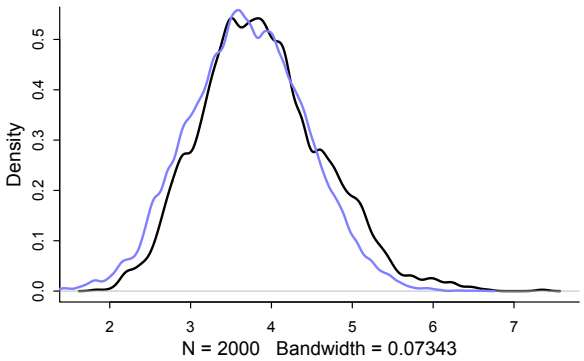
```
pr <- precis( m11.4 , 2 )[,1:4]
prq <- precis( m11.4q , 2 )
round( cbind( pr , prq ) , 2 )
```

```
      mean   sd  5.5% 94.5%  mean   sd  5.5% 94.5%
a[1] -0.46 0.32 -0.97  0.05 -0.44 0.33 -0.96  0.08
a[2]  3.89 0.76  2.76  5.15  3.71 0.72  2.55  4.86
a[3] -0.76 0.34 -1.32 -0.24 -0.73 0.33 -1.26 -0.20
a[4] -0.75 0.33 -1.27 -0.26 -0.73 0.33 -1.26 -0.20
a[5] -0.45 0.33 -0.96  0.05 -0.44 0.33 -0.96  0.08
a[6]  0.46 0.32 -0.06  0.98  0.47 0.33 -0.06  1.00
a[7]  1.94 0.43  1.28  2.68  1.91 0.41  1.24  2.57
b[1] -0.03 0.28 -0.48  0.45 -0.04 0.28 -0.49  0.41
b[2]  0.49 0.28  0.05  0.94  0.47 0.28  0.02  0.93
b[3] -0.37 0.28 -0.82  0.09 -0.38 0.29 -0.83  0.08
b[4]  0.37 0.28 -0.06  0.82  0.36 0.28 -0.09  0.82
```

The first 4 columns are the original `ulam` model. The last 4 columns are the new `quap` model. THe only noticeable difference is with `a[2]`. The `ulam` model has a higher mean and a clearly higher upper range. Let's plot these densities to see what's going on:

R code 11.6

```
post <- extract.samples( m11.4 )
postq <- extract.samples( m11.4q )
dens( post$a[,2] , lwd=2 )
dens( postq$a[,2] , add=TRUE , lwd=2 , col=rangi2 )
```



The black density is the `ulam` model. You can see how it places more mass in the upper tail. What is happening here is that the `quap` model, in blue, has to be Gaussian, so it ends up putting too little mass in the upper tail and too much in the lower tail. Why is there supposed to be more mass in the upper tail? Because of the ceiling effect of the logit link. Recall that these parameters are on the log-odds (logit) scale. A value above 4 means "almost always". It's not possible for data to decide between

a value of 5 and a value of 6 on this scale. This means there are many high values that will produce the same probability of the data. The prior stops values like 10 or 20 from being plausible.

When we relax the prior on these parameters, it makes this point much clearer. Here's the revised code for both models:

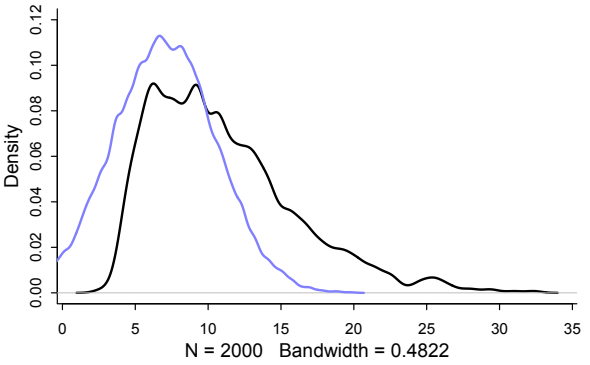
R code 11.7

```
m11M7q <- quap(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + b[treatment] ,
        a[actor] ~ dnorm( 0 , 10 ),
        b[treatment] ~ dnorm( 0 , 0.5 )
    ) , data=dat_list )
m11M7u <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + b[treatment] ,
        a[actor] ~ dnorm( 0 , 10 ),
        b[treatment] ~ dnorm( 0 , 0.5 )
    ) , data=dat_list , cores=4 , chains=4 )
```

Again let's focus on actor 2's intercepts:

R code 11.8

```
post <- extract.samples( m11M7u )
postq <- extract.samples( m11M7q )
dens( post$a[,2] , lwd=2 , ylim=c(0,0.12) )
dens( postq$a[,2] , add=TRUE , lwd=2 , col=rangi2 )
```



Now with the flatter prior, the very large log-odds values are not excluded and the posterior includes them. The quadratic approximation in this case (blue) is very bad. Note that this is the kind of nonsense that you can expect from most non-Bayesian binomial models. You can see for yourself. Here is R's built-in GLM fitting function doing the same model:

R code 11.9

```
precis( glm( pulled_left ~ as.factor(actor) + as.factor(treatment) ,
    data=dat_list , family=binomial ) )
```

```
                      mean      sd      5.5%   94.5%
(Intercept)          -0.52    0.31    -1.01   -0.04
as.factor(actor)2    18.96  752.72 -1184.03 1221.96
as.factor(actor)3    -0.31    0.35    -0.87    0.25
as.factor(actor)4    -0.31    0.35    -0.87    0.25
```

```
as.factor(actor)5      0.00   0.34    -0.55    0.55
as.factor(actor)6      0.94   0.35     0.39    1.50
as.factor(actor)7      2.50   0.45     1.78    3.22
as.factor(treatment)2  0.62   0.30     0.14    1.10
as.factor(treatment)3 -0.42   0.31    -0.91    0.07
as.factor(treatment)4  0.49   0.30     0.01    0.96
```

The intercept for actor 2 has quite a confidence region.

11M8. Let's load the data, strip out Hawaii, and fit the model again:

R code 11.10

```
library(rethinking)
data(Kline)
d <- Kline
d$P <- standardize( log(d$population) )
d$contact_id <- ifelse( d$contact=="high" , 2 , 1 )
d2 <- d[ d$culture!="Hawaii" , ]

dat2 <- list(
    T = d2$total_tools ,
    P = d2$P ,
    cid = d2$contact_id )

m11.10b <- ulam(
    alist(
        T ~ dpois( lambda ),
        log(lambda) <- a[cid] + b[cid]*P,
        a[cid] ~ dnorm( 3 , 0.5 ),
        b[cid] ~ dnorm( 0 , 0.2 )
    ), data=dat2 , chains=4 )
```

R code 11.11

```
precis(m11.10b,2)
```

```
     mean   sd  5.5% 94.5% n_eff Rhat4
a[1] 3.17 0.13  2.96  3.38  1517     1
a[2] 3.61 0.07  3.49  3.73  1666     1
b[1] 0.19 0.13 -0.02  0.39  1463     1
b[2] 0.19 0.16 -0.07  0.45  2047     1
```

Compared to the model fit to the complete sample, there is now no difference between the two slopes. Hawaii was really the only point that made the contact categories different. The relationship with log-population remains.

11H1. Let's prep the data first (same code as in the chapter):

R code 11.12

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$treatment <- 1 + d$prosoc_left + 2*d$condition
dat_list <- list(
    pulled_left = d$pulled_left,
    actor = d$actor,
```

```
    treatment = as.integer(d$treatment) )
```

Now there are four models to fit: `m11.1` through `m11.4`. Let's fit all of them with `ulam`:

R code 11.13

```
m11.1 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a ,
        a ~ dnorm( 0 , 10 )
    ) , data=dat_list , chains=4 , log_lik=TRUE )
m11.2 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a + b[treatment] ,
        a ~ dnorm( 0 , 1.5 ),
        b[treatment] ~ dnorm( 0 , 10 )
    ) , data=dat_list , chains=4 , log_lik=TRUE  )
m11.3 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a + b[treatment] ,
        a ~ dnorm( 0 , 1.5 ),
        b[treatment] ~ dnorm( 0 , 0.5 )
    ) , data=dat_list , chains=4 , log_lik=TRUE )
m11.4 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + b[treatment] ,
        a[actor] ~ dnorm( 0 , 1.5 ),
        b[treatment] ~ dnorm( 0 , 0.5 )
    ) , data=dat_list , chains=4 , log_lik=TRUE )
```

Now we can compare:

R code 11.14

```
compare( m11.1 , m11.2 , m11.3 , m11.4 , func=PSIS )
```

```
       PSIS    SE dPSIS   dSE pPSIS weight
m11.4 532.4 18.97   0.0    NA   8.6      1
m11.3 682.7  9.19 150.3 18.43   3.7      0
m11.2 683.4  9.73 151.0 18.50   4.2      0
m11.1 688.1  7.08 155.7 18.95   1.1      0
```

That is pretty solid support for the predictive superiority of the actor-specific intercept model. This isn't surprising, since so much variation is among actors and not so much among treatments.

11H2. (a) First, we need to make some 0/1 dummy variables for the categorical variables `P`, `A`, and `V` in the data frame. Name them whatever you like, but here are my choices:

R code 11.15

```
library(rethinking)
library(MASS)
data(eagles)
d <- eagles
```

```
d$pirateL <- ifelse( d$P=="L" , 1 , 0 )
d$victimL <- ifelse( d$V=="L" , 1 , 0 )
d$pirateA <- ifelse( d$A=="A" , 1 , 0 )
```

Now to fit the model both ways. I'm going use the standard intercept prior here, and a only slighter narrow prior for each coefficient. This allows for large effects.

R code 11.16

```
f <- alist(
        y ~ dbinom( n , p ),
        logit(p) <- a + bP*pirateL + bV*victimL + bA*pirateA ,
        a ~ dnorm(0,1.5),
        bP ~ dnorm(0,1),
        bV ~ dnorm(0,1),
        bA ~ dnorm(0,1) )

m1 <- quap( f , data=d )

m1_stan <- ulam( f , data=d , chains=4 , log_lik=TRUE )

precis(m1)
precis(m1_stan)
```

```
    mean   sd  5.5% 94.5%
a   0.35 0.48 -0.42  1.12
bP  2.58 0.44  1.88  3.28
bV -2.71 0.47 -3.46 -1.96
bA  0.89 0.41  0.24  1.54

    mean   sd  5.5% 94.5% n_eff Rhat4
a   0.36 0.48 -0.42  1.12  1245     1
bP  2.64 0.45  1.95  3.36  1371     1
bV -2.78 0.47 -3.55 -2.04  1333     1
bA  0.91 0.42  0.27  1.61  1325     1
```

Those are reasonably similar. Note that the `ulam` model does have more extreme coefficient values for `bP` and `bV`. So likely the quadratic approximation is having a little trouble with ceiling/floor effects. Note that if you make the priors weaker, these models will not be so similar. Try Normal(0,10) for example:

R code 11.17

```
f <- alist(
        y ~ dbinom( n , p ),
        logit(p) <- a + bP*pirateL + bV*victimL + bA*pirateA ,
        a ~ dnorm(0,10),
        bP ~ dnorm(0,10),
        bV ~ dnorm(0,10),
        bA ~ dnorm(0,10) )
```

Refit the models and then you should see:

R code 11.18

```
precis(m1)
precis(m1_stan)
```

```
    mean   sd  5.5% 94.5%
a   0.61 0.67 -0.47  1.69
```

```
bP  4.47 1.00  2.86  6.07
bV -4.83 1.07 -6.54 -3.12
bA  1.09 0.54  0.23  1.96

    mean   sd  5.5% 94.5% n_eff Rhat4
a   0.66 0.72 -0.47  1.85   900  1.01
bP  5.01 1.21  3.43  7.16   649  1.00
bV -5.41 1.28 -7.66 -3.67   694  1.00
bA  1.15 0.57  0.27  2.10   803  1.01
```

Again, with GLMs the existence of ceiling and floor effects makes quadratic approximation risky.

(b) Now for interpreting these estimates. We'll use the `ulam` estimates here, because the quadratic approximation was not entirely satisfactory. Remember, your estimates will be slightly different, due to Monte Carlo error. But the effective predictions should be indistinguishable.

First, the intercept log-odds, `a`, indicates the probability of a successful attempt for a pirate when all of the predictors are at zero. So that means a small immature pirate against a small victim has probability:

R code 11.19

```
post <- extract.samples( m1_stan )
quantile( logistic( post$a ) , probs=c(0.055,0.5,0.945) )
```

```
     5.5%       50%     94.5%
0.3960993 0.5859281 0.7544786
```

A little under 60% of attempts by immature small pirates on small victims are expected to succeed. Note the way I did the calculation above. First the logistic transform is performed for every sample, and then the summary is made. Always summarize last. The mean of a function is not the same as the function of the mean!

So what about the slope (β) estimates? These estimates just change the intercept for the log-odds. So for example, the probability that a large immature pirate succeeds against a small victim would be:

R code 11.20

```
quantile( logistic( post$a + post$bP ) , probs=c(0.055,0.5,0.945) )
```

```
     5.5%       50%     94.5%
0.8993965 0.9518775 0.9786352
```

That is to say, the large pirate is almost certain to succeed, according to the model. You can generate such predictions for all of the combinations, if you wish. And indeed, that's how the plotting of predictions works.

There are many ways to plot the posterior predictions. I'll use a straightforward format with cases on the horizontal and probability/count on the vertical. Here's the code for the first plot, showing proportion success on the vertical axis:

R code 11.21

```
d$psuccess <- d$y / d$n

p <- link(m1_stan)
y <- sim(m1_stan)

p.mean <- apply( p , 2 , mean )
p.PI <- apply( p , 2 , PI )
y.mean <- apply( y , 2 , mean )
y.PI <- apply( y , 2 , PI )
```

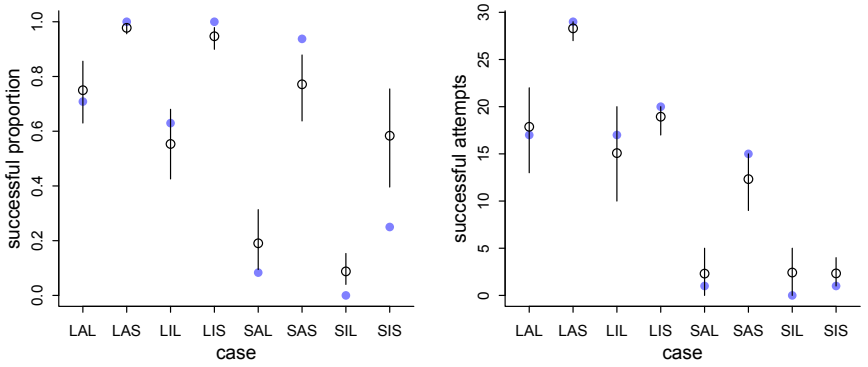
```
# plot raw proportions success for each case
plot( d$psuccess , col=rangi2 ,
    ylab="successful proportion" , xlab="case" , xaxt="n" ,
    xlim=c(0.75,8.25) , pch=16 )

# label cases on horizontal axis
axis( 1 , at=1:8 ,
    labels=c( "LAL","LAS","LIL","LIS","SAL","SAS","SIL","SIS" ) )

# display posterior predicted proportions successful
points( 1:8 , p.mean )
for ( i in 1:8 ) lines( c(i,i) , p.PI[,i] )
```

And here's the second plot, showing number of successes on the vertical:

R code 11.22

```
# plot raw counts success for each case
plot( d$y , col=rangi2 ,
    ylab="successful attempts" , xlab="case" , xaxt="n" ,
    xlim=c(0.75,8.25) , pch=16 )

# label cases on horizontal axis
axis( 1 , at=1:8 ,
    labels=c( "LAL","LAS","LIL","LIS","SAL","SAS","SIL","SIS" ) )

# display posterior predicted successes
points( 1:8 , y.mean )
for ( i in 1:8 ) lines( c(i,i) , y.PI[,i] )
```

And here are the resulting plots:



So what's different? The biggest difference is probably that the left plot (proportions) makes the probabilities more comparable, because it ignores the sample size for each case on the horizontal axis. This has the advantage of showing, for example, that SIS attempts are predicted to be somewhat successful, even though only 4 of them were observed. The right plot (counts), in contrast, makes it hard to see the differing probabilities, because samples size varies so much across cases.

On the other hand, the count plot (right) has the advantage of showing additional uncertainty that arises from the binomial process. Inspect the SIS case again, for example. The observed proportion of SIS successes is outside and below the probability interval (left plot). However, in the bottom plot, the model can accommodate the SIS cases, due to additional uncertainty arising from the binomial

process. In other words, an additional level of stochasticity is factored into the bottom plot, and so in total looking also at counts might be necessary to seriously critique the model.

(c) The two models to compare are the one you fit before, with the three main effects, and the new one, including the interaction $P \times A$. Here is the code to fit the models and compare them:

R code 11.23

```
m2 <- ulam(
    alist(
        y ~ dbinom( n , p ),
        logit(p) <- a + bP*pirateL + bV*victimL +
            bA*pirateA + bPA*pirateL*pirateA ,
        a ~ dnorm(0,1.5),
        bP ~ dnorm(0,1),
        bV ~ dnorm(0,1),
        bA ~ dnorm(0,1),
        bPA ~ dnorm(0,1)
    ) , data=d , chains=4 , log_lik=TRUE )

compare( m1_stan , m2 )
```

```
          WAIC   SE dWAIC  dSE pWAIC weight
m1_stan 37.1 5.63   0.0   NA   5.2   0.53
m2      37.3 5.17   0.2 0.76   5.3   0.47
```

This seems to be a tie. Let's look at the estimates for the interaction model:

R code 11.24

```
precis(m2)
```

```
     mean   sd  5.5% 94.5% n_eff Rhat4
a    0.32 0.53 -0.52  1.18   971     1
bP   2.70 0.51  1.92  3.54  1291     1
bV  -2.77 0.50 -3.63 -1.98  1083     1
bA   0.98 0.50  0.19  1.80   988     1
bPA -0.14 0.61 -1.12  0.82  1138     1
```

So when the pirate is both large and adult, the log-odds are smaller. This seems like a weird result. Shouldn't being large and adult help? It does. But the main effects for size and age are also turned on, when an individual is both large and adult. So it's hard to understand what the model is saying, without computing predictions case by case. However, the predictions are essentially the same as the previous model. There just isn't enough data here to support any confident inference about an interaction.

11H3. (a) This is a classic case in which the raw predictor variables have an inconveniently large scale. Standardizing the predictors before fitting will help both.

R code 11.25

```
data(salamanders)
d <- salamanders
d$C <- standardize(d$PCTCOVER)
d$A <- standardize(d$FORESTAGE)
```

The Poisson model is easy enough. But getting the priors to be sensible will take some simulation. Consider this model formula:

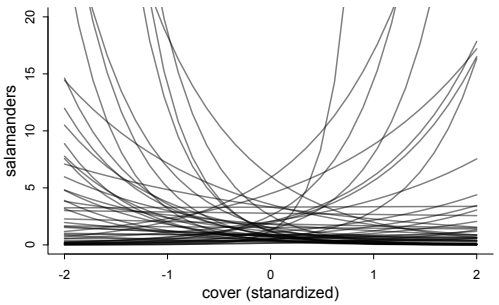
R code 11.26

```
f <- alist(
    SALAMAN ~ dpois( lambda ),
    log(lambda) <- a + bC*C,
    a ~ dnorm(0,1),
    bC ~ dnorm(0,1) )
```

Let's simulate directly from the prior and see what the implied observations look like:

R code 11.27

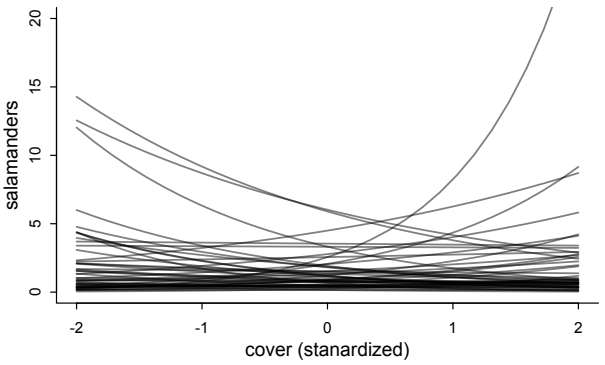
```
N <- 50 # 50 samples from prior
a <- rnorm( N , 0 , 1 )
bC <- rnorm( N , 0 , 1 )
C_seq <- seq( from=-2 , to=2 , length.out=30 )
plot( NULL , xlim=c(-2,2) , ylim=c(0,20) ,
    xlab="cover (stanardized)" , ylab="salamanders" )
for ( i in 1:N )
    lines( C_seq , exp( a[i] + bC[i]*C_seq ) , col=grau() , lwd=1.5 )
```



You may not have a good prior for how large the average salamander count should be. But it should not be huge. They are hard to find! This prior isn't awful—it allows some rare explosive trends, but is mainly staying within the plausible observation range. Let's try to calm it down a bit by shrinking `bC`:

R code 11.28

```
bC <- rnorm( N , 0 , 0.5 )
plot( NULL , xlim=c(-2,2) , ylim=c(0,20) ,
    xlab="cover (stanardized)" , ylab="salamanders" )
for ( i in 1:N )
    lines( C_seq , exp( a[i] + bC[i]*C_seq ) , col=grau() , lwd=1.5 )
```



That's better—fewer explosive increases in salamanders.

Now let's update the prior with the data:

R code 11.29

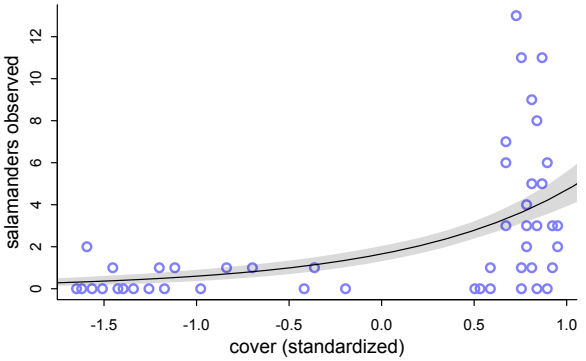
```
f <- alist(
    SALAMAN ~ dpois( lambda ),
    log(lambda) <- a + bC*C,
    a ~ dnorm(0,1),
    bC ~ dnorm(0,0.5) )
m1 <- ulam( f , data=d , chains=4 )
precis(m1)
```

```
   mean   sd 5.5% 94.5% n_eff Rhat4
a  0.49 0.14 0.26  0.71   549  1.01
bC 1.05 0.17 0.79  1.32   548  1.00
```

There seems to be a positive association with percent cover. Let's plot the posterior predictions:

R code 11.30

```
plot( d$C , d$SALAMAN , col=rangi2 , lwd=2 ,
    xlab="cover (standardized)" , ylab="salamanders observed" )
C_seq <- seq( from=-2 , to=2 , length.out=30 )
l <- link( m1 , data=list(C=C_seq) )
lines( C_seq , colMeans( l ) )
shade( apply( l , 2 , PI ) , C_seq )
```



Let's does seem like a case in which the variance is much greater than the mean—an over-dispersion situation that we'll discuss the in the next chapter. Still, it is the very high end of forest cover that shows some high counts.

(b) The motivation for this second model is the possibility that forest cover is associated causally with forest age—older forests are less disturbed and have more cover. In that case, it could be that forage age is a confound. Let's inspect a model with both cover and forest age.

R code 11.31

```
f2 <- alist(
    SALAMAN ~ dpois( lambda ),
    log(lambda) <- a + bC*C + bA*A,
    a ~ dnorm(0,1),
    c(bC,bA) ~ dnorm(0,0.5) )
m2 <- ulam( f2 , data=d , chains=4 )
precis(m2)
```

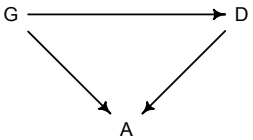
```
   mean   sd  5.5% 94.5% n_eff Rhat4
```

```
a  0.48 0.14  0.25  0.71   908     1
bA 0.02 0.09 -0.13  0.17  1208     1
bC 1.04 0.18  0.76  1.33   868     1
```

Notice that the estimate for `bA` is now nearly zero, with small interval around it. There isn't much association between forest age and salamander density, while also conditioning on percent cover.

Why doesn't forest age help much? It certainly does improve predictions, in the absence of percent cover—check for yourself by fitting a model that includes only `FORESTAGE` as a predictor. If all we knew was forest age, it would be a good predictor (try it!). But compared to percent cover, forest age doesn't help us at all.

11H4. The implied DAG is:



where G is gender, D is discipline, and A is award. The direct causal effect of gender is the path $G \rightarrow A$. The total effect includes that path and the indirect path $G \rightarrow D \rightarrow A$. We can estimate the total causal influence (assuming this DAG is correct) with a model that conditions only on gender. I'll use a N(-1,1) prior for the intercepts, because we know from domain knowledge that less than half of applicants get awards.

R code 11.32

```
dat_list <- list(
    awards = as.integer(d$awards),
    apps = as.integer(d$applications),
    gid = ifelse( d$gender=="m" , 1L , 2L )
)
m1_total <- ulam(
    alist(
        awards ~ binomial( apps , p ),
        logit(p) <- a[gid],
        a[gid] ~ normal(-1,1)
    ), data=dat_list , chains=4 )
precis(m1_total,2)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat
a[1] -1.53 0.06 -1.64 -1.43  1371    1
a[2] -1.74 0.08 -1.88 -1.61  1291    1
```

Gender 1 here is male and 2 is female. So males have higher rates of award, on average. How big is the difference? Let's look at the contrast on absolute (penguin) scale:

R code 11.33

```
post <- extract.samples(m1_total)
diff <- inv_logit( post$a[,1] ) - inv_logit( post$a[,2] )
precis( list( diff=diff ) )
```

```
'data.frame': 2000 obs. of 1 variables:
     mean   sd 5.5% 94.5%    histogram
diff 0.03 0.01 0.01  0.05 ▁▁▁▂▅▇▃▂▁▁▁
```

So a small 3% difference on average. Still, with such low funding rates (in some disciplines), 3% is a big advantage.

Now for the direct influence of gender, we condition on discipline as well:

R code 11.34

```
dat_list$disc <- as.integer(d$discipline)
m1_direct <- ulam(
    alist(
        awards ~ binomial( apps , p ),
        logit(p) <- a[gid] + d[disc],
        a[gid] ~ normal(-1,1),
        d[disc] ~ normal(0,1)
    ), data=dat_list , chains=4 , cores=4 , iter=3000 )
precis(m1_direct,2)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat
a[1] -1.33 0.31 -1.84 -0.85   615 1.01
a[2] -1.47 0.31 -1.98 -0.98   636 1.01
d[1]  0.31 0.36 -0.26  0.90   848 1.01
d[2] -0.01 0.33 -0.54  0.54   722 1.01
d[3] -0.24 0.33 -0.75  0.30   694 1.01
d[4] -0.28 0.36 -0.85  0.31   791 1.00
d[5] -0.35 0.33 -0.86  0.19   691 1.01
d[6] -0.03 0.35 -0.58  0.53   789 1.00
d[7]  0.28 0.39 -0.33  0.91   968 1.00
d[8] -0.46 0.32 -0.96  0.07   669 1.01
d[9] -0.21 0.34 -0.74  0.34   758 1.01
```

Those chains didn't sample very efficiently. This likely because the model is over-parameterized—it has more parameters than absolutely necessary. This doesn't break it. It just makes the sampling less efficient. Anyway, now we can compute the gender difference again. On the relative scale:

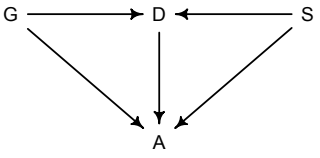
R code 11.35

```
post <- extract.samples(m1_direct)
diff_a <- post$a[,1] - post$a[,2]
precis( list( diff_a=diff_a ) )
```

```
'data.frame': 6000 obs. of 1 variables:
       mean   sd  5.5% 94.5% histogram
diff_a 0.14 0.11 -0.03  0.31 ▁▂▅▇▃▁▁
```

Still an advantage for the males, but reduced and overlapping zero a bit. To see this difference on the absolute scale, we need to account for the base rates in each discipline as well. If you look at the `postcheck(m1_direct)` display, you'll see the predictive difference is very small. There are also several disciplines that reverse the advantage. If there is a direct influence of gender here, it is small, much smaller than before we accounted for discipline. Why? Because again the disciplines have different funding rates and women apply more to the disciplines with lower funding rates. But it would be hasty, I think, to conclude there are no other influences. There are after all lots of unmeasured confounds...

11H5. The implied DAG is:



where S is stage of career (unobserved). This DAG has the same structure as the grandparents-parents-children-neighborhoods example from earlier in the course. When we condition on discipline D it opens a backdoor path through S to A. It is not possible here to get an unconfounded estimate of gender on awards.

Here's a simulation to demonstrate the potential issue.

R code 11.36

```
set.seed(1913)
N <- 1000
G <- rbern(N)
S <- rbern(N)
D <- rbern( N , p=inv_logit( G + S ) )
A <- rbern( N , p=inv_logit( 0.25*G + D + 2*S - 2 ) )
dat_sim <- list( G=G , D=D , A=A )
```

This code simulates 1000 applicants. There are 2 genders (G 0/1), 2 stages of career (S 0/1), and 2 disciplines (D 0/1). Discipline 1 is chosen more by gender 1 and career stage 1. So that could mean more by males and later stage of career. Then awards A have a consistent bias towards gender 1, and discipline 1 has a higher award rate, and stage 1 also a higher award rate. If we analyze these data:

R code 11.37

```
m2_sim <- ulam(
    alist(
        A ~ bernoulli(p),
        logit(p) <- a + d*D + g*G,
        c(a,d,g) ~ normal(0,1)
    ), data=dat , chains=4 , cores=4 )
precis(m2_sim)
```

```
   mean   sd  5.5% 94.5% n_eff Rhat
g  0.09 0.13 -0.13  0.30   984    1
d  1.21 0.15  0.98  1.46   927    1
a -0.90 0.13 -1.12 -0.70   983    1
```

The parameter g is the advantage of gender 1. It is smaller than the true advantage and the estimate straddles zero quite a lot, even with 1000 applicants. It is also possible to have no gender influence and infer it by accident. Try these settings:

R code 11.38

```
set.seed(1913)
N <- 1000
G <- rbern(N)
S <- rbern(N)
D <- rbern( N , p=inv_logit( 2*G - S ) )
A <- rbern( N , p=inv_logit( 0*G + D + S - 2 ) )
dat_sim2 <- list( G=G , D=D , A=A )
m2_sim_2 <- ulam( m2_sim , data=dat_sim2 , chains=4 , cores=4 )
precis(m2_sim_2,2)
```

```
   mean   sd  5.5% 94.5% n_eff Rhat
g  0.25 0.15  0.00  0.48  1153    1
d  0.28 0.15  0.03  0.52  1036    1
a -1.12 0.12 -1.31 -0.94  1166    1
```

Now it looks like gender 1 has a consistent advantage, but in fact there is no advantage in the simulation.

11H6. First let's load the data and set it up for use:

R code 11.39

```
library(rethinking)
data(Primates301)
d <- Primates301
d2 <- d[ complete.cases( d$social_learning , d$brain , d$research_effort ) , ]
dat <- list(
    soc_learn = d2$social_learning,
    log_brain = standardize( log(d2$brain) ),
    log_effort = log(d2$research_effort) )
```

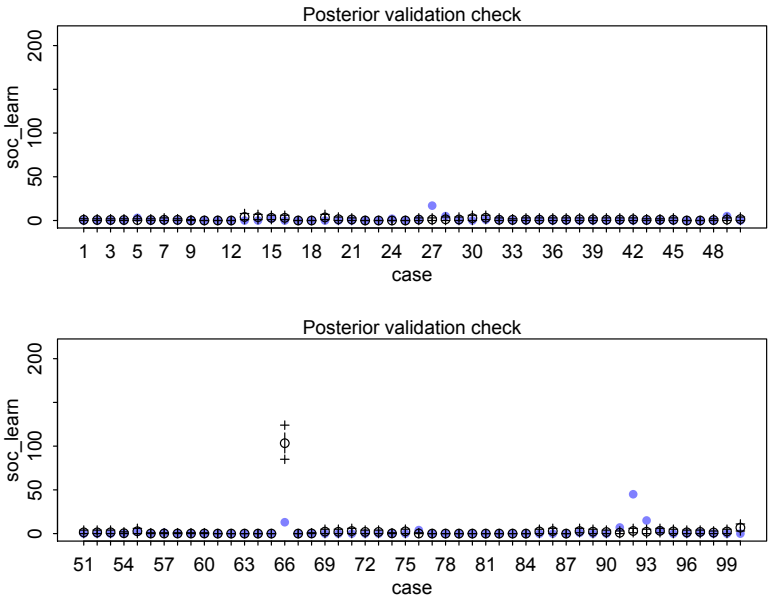
Now we first want a model with social learning as the outcome and brain size as a predictor. For this Poisson GLM, I'm going to use a N(0,1) prior on the intercept, since we know the counts should be small.

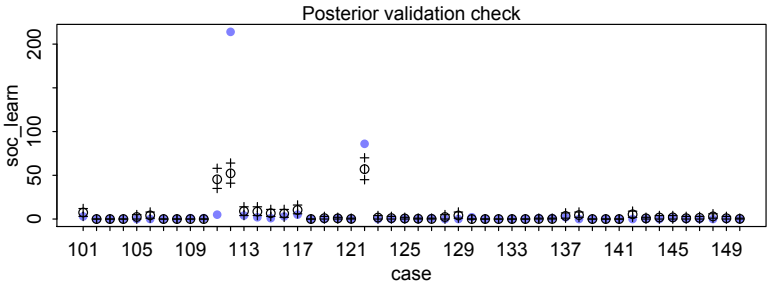
R code 11.40

```
m3_1 <- ulam(
    alist(
        soc_learn ~ poisson( lambda ),
        log(lambda) <- a + bb*log_brain,
        a ~ normal(0,1),
        bb ~ normal(0,0.5)
    ), data=dat , chains=4 , cores=4 )
precis( m3_1 )
```

```
    mean   sd  5.5% 94.5% n_eff Rhat
a  -1.18 0.12 -1.36 -0.99   423    1
bb  2.76 0.08  2.64  2.88   445    1
```

Brain size seems to be strongly associated with social learning observations. Let's look at the posterior predictions. I'll use `postcheck(m3_1,window=50)`:





The blue points are the raw data, recall. These are not great posterior predictions. Clearly other factors are in play.

Let's try the research effort variable now:

R code 11.41

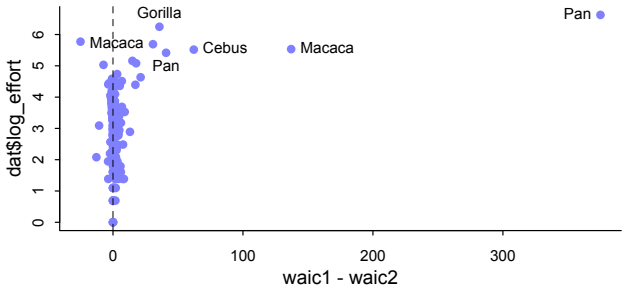
```
m3_2 <- ulam(
    alist(
        soc_learn ~ poisson( lambda ),
        log(lambda) <- a + be*log_effort + bb*log_brain,
        a ~ normal(0,1),
        c(bb,be) ~ normal(0,0.5)
    ), data=dat , chains=4 , cores=4 )
precis( m3_2 )
```

```
    mean   sd  5.5% 94.5% n_eff Rhat
a  -5.97 0.33 -6.49 -5.45   479 1.01
be  1.53 0.07  1.42  1.64   456 1.01
bb  0.46 0.08  0.33  0.59   599 1.01
```

Brain size `bb` is still positively associated, but much less. Research effort `be` is strongly associated. To see how these models disagree, let's use pointwise WAIC to see which cases each predicts well.

R code 11.42

```
waic1 <- WAIC( m3_1 , pointwise=TRUE )
waic2 <- WAIC( m3_2 , pointwise=TRUE )
plot( waic1 - waic2 , dat$log_effort , col=rangi2 , pch=16 )
identify( waic1-waic2 , dat$log_effort , d2$genus , cex=0.8 )
abline(v=0,lty=2,lwd=0.5)
```

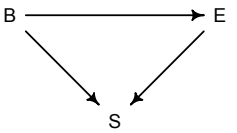


Species on the right of the vertical line fit better for model `m3_2`, the model with research effort. These are mostly species that are studied a lot, like chimpanzees (*Pan*) and macaques (*Macaca*). The genus *Pan* especially has been a focus on social learning research, and its counts are inflated by this.

This is a good example of how the nature of measurement influences inference. There are likely a lot of false zeros in these data, species that are not studied often enough to get a good idea of their

learning tendencies. Meanwhile every time a chimpanzee sneezes, someone writes a social learning paper.

Okay, finally I asked for a DAG. This is my guess:



B is brain size, E is research effort, and S is social learning. Research effort doesn't actually influence social learning, but it does influence the value of the variable. The model results above are consistent with this DAG in the sense that including E reduced the association with B, which we would expect when we close the indirect path through E. If researchers choose to look for social learning in species with large brains, this leads to an exaggerated estimate of the association between brains and social learning.

12. Chapter 12 Solutions

12E1. An ordered categorical variable is constrained to discrete values, such as {1,2,3,4,5}, and these values are also constrained to a fixed order of magnitudes. The distances between the values are not however necessarily the same. So for example the amount of change required to move an a value from 1 to 2 may be different from the amount of change required to move from 2 to 3. Examples include subjective ratings, "Likert" scales, and relative distances or durations.

An unordered categorical variable, in contrast, is not constrained to any order among the values. The different values merely represent different discrete outcomes, without any implied ordering. These variables are the natural extension of binary outcomes to more than two categories. Examples include choices among colors, individual identities, and individual words.

12E2. Ordered logistic regression typically uses a *cumulative logit* link function. This is similar to an ordinary logit link, but in which the probability is a cumulative probability instead of a discrete probability of a single event. As a consequence, the cumulative logit link states that the linear model is the log-odds of the specified event *or any event of lower ordered value*.

12E3. Ignoring zero-inflation will tend to underestimate the rate of events. Why? Because a count distribution with extra zeros added to it will have a lower mean. So treating such data as single-process count data will result in a lower estimate for the mean rate.

12E4. Over-dispersion can arise simply from variation in underlying rates across units. For example, if we count the number of ice creams sold by various ice cream shops for each day over an entire month, the aggregated counts will likely be over-dispersed. This is because some shops sell more ice cream than others—they do not all have the same average rate of sales across days.

Under-dispersion is considered less often. Under-dispersed count data has less variation than expected. One common process that might produce under-dispersed counts is when sequential observations are directly correlated with one another, *autocorrelation*. This is the premise of Conway-Maxwell-Poisson (aka COM-Poisson) distributions, which arise from one model of this kind, the *state-dependent queuing model*, commonplace in the study of servers and production systems of many kinds. Simply stated, when the rate at which jobs are completed depends upon how many jobs are waiting to be completed, then counts may be highly autocorrelated. This reduces variation in the observed counts, resulting in under-dispersion.

12M1. Log cumulative odds is defined as:

$$\log \frac{p_k}{1 - p_k}$$

where p_k is the probability of value k or any value less than k. So first we compute the proportion of the sample that is at each unique value. We'll call these values q:

R code 12.1

```
n <- c( 12, 36 , 7 , 41 )
q <- n / sum(n)
q
```

```
[1] 0.12500000 0.37500000 0.07291667 0.42708333
```

Note that these values sum to 1:

R code 12.2

```
sum(q)
```

```
[1] 1
```

Now compute cumulative probability of each value. The algorithm is to take each value and add to it all of the values to its left. The R function `cumsum` actually does this for us:

R code 12.3

```
p <- cumsum(q)
p
```

```
[1] 0.1250000 0.5000000 0.5729167 1.0000000
```

There are the p_k values we need. All that remains is to take the log of the odds of each:

R code 12.4

```
log(p/(1-p))
```

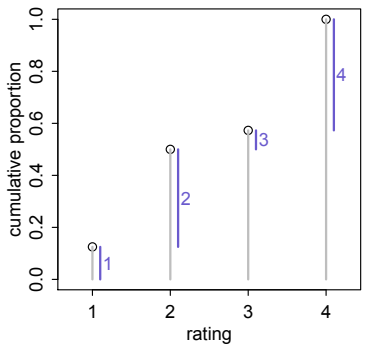
```
[1] -1.9459101  0.0000000  0.2937611        Inf
```

Note that the log cumulative odds of the highest value is infinity, just as in the chapter. It is always known, because of how the data are scaled.

12M2. Using the vectors `p` and `q` from the previous problem:

R code 12.5

```
plot( 1:4 , p , xlab="rating" , ylab="cumulative proportion" ,
    xlim=c(0.7,4.3) , ylim=c(0,1) , xaxt="n" )
axis( 1 , at=1:4 , labels=1:4 )

# plot gray cumulative probability lines
for ( x in 1:4 ) lines( c(x,x) , c(0,p[x]) , col="gray" , lwd=2 )

# plot blue discrete probability segments
for ( x in 1:4 )
    lines( c(x,x)+0.1 , c(p[x]-q[x],p[x]) , col="slateblue" , lwd=2 )

# add number labels
text( 1:4+0.2 , p-q/2 , labels=1:4 , col="slateblue" )
```



12M3. As with the zero-inflated Poisson, the zero-inflated binomial mixes some extra zeros into another distribution. The structure is very much the same as the ZI-Poisson. First, a single probability determines whether or not a zero is observed. Call this probability p_0. When a zero is not observed from this first process, the binomial distribution takes over. It may also generate a zero. Call the probability of a success from the binomial process q, and let it have n trials. Then the probability of a zero, mixing together both processes, is:

$$\Pr(0|p_0, q, n) = p_0 + (1 - p_0)(1 - q)^n$$

The logic is that either we get a zero from the first process, p_0 of the time, or we got a zero from the binomial, which happens only when all trials fail, $(1 - q)^n$ of the time. The probability of any particular non-zero observation y is similarly:

$$\Pr(y|p_0, q, n) = (1 - p_0)\frac{n!}{y!(n - y)!}q^y(1 - q)^{n-y}$$

Compare this expression to the ZI-Poisson expression in the chapter.

12H1. The problem left you freedom to specify your own priors. We'll need to do some prior simulations again, of course. This means thinking about typical ranges of hurricane deaths (the intercept) and how much of an effect the name could possibly have (the slope).

Let's start with a very simple Poisson model predicting `deaths` using only `femininity` as a predictor. I'm going to standardize `femininity`, for the usual reasons. Then we'll simulate from the priors and see how not to make the prior predictions silly.

R code 12.6

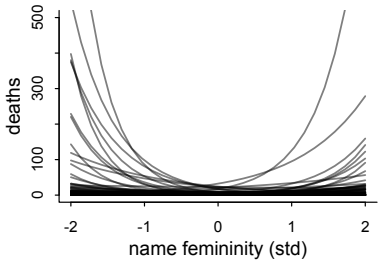
```
library(rethinking)
data(Hurricanes)
d <- Hurricanes
d$fmnnty_std <- ( d$femininity - mean(d$femininity) )/sd(d$femininity)
dat <- list( D=d$deaths , F=d$fmnnty_std )

# model formula - no fitting yet
f <- alist(
        D ~ dpois(lambda),
        log(lambda) <- a + bF*F,
        a ~ dnorm(1,1),
        bF ~ dnorm(0,1) )
```

Now if you aren't an expert on hurricanes, you won't have a sense of what the plausible range of D should be. The most deadly hurricane in recorded history is the 1900 "Great Galveston hurricane," which killed somewhere between 6000 and 10000 people. Note that this storm doesn't have a personal name, and it doesn't appear in the sample, because storms didn't get names until after 1950. Most storms since 1950 are also much less deadly than this, killing fewer than a dozen people. But occasionally there are storms that kill a hundred or more. Hurricane mortality is a thick-tailed phenomenon. So you might have a hunch already that a simple Poisson model is not going to do a good job. But let's do out best. What do the priors above produce?

R code 12.7

```
N <- 100
a <- rnorm(N,1,1)
bF <- rnorm(N,0,1)
F_seq <- seq( from=-2 , to=2 , length.out=30 )
plot( NULL , xlim=c(-2,2) , ylim=c(0,500) ,
    xlab="name femininity (std)" , ylab="deaths" )
```

```
for ( i in 1:N ) lines( F_seq , exp( a[i] + bF[i]*F_seq ) , col=grau() , lwd=1.5 )
```



This allows for some rather implausible extreme trends. But the typical trend is very modest, only a very small increase (or decrease) in deaths as femininity of the name changes. I'd rather use a tighter prior on the slope, because it makes no sense to think that the femininity of a storm's name could increase the deaths by an order of magnitude. But let's proceed with these priors and see what happens.

To fit the model:

R code 12.8

```
m1 <- ulam( f , data=dat , chains=4 , log_lik=TRUE )
precis( m1 )
```

```
   mean   sd 5.5% 94.5% n_eff Rhat4
a  3.00 0.02 2.96  3.04  1219     1
bF 0.24 0.03 0.20  0.28  1069     1
```

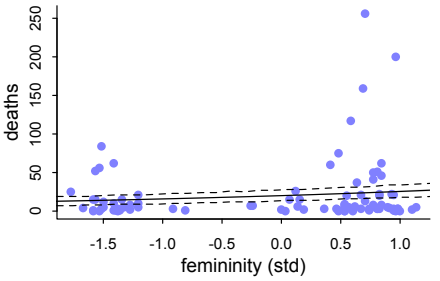
The model seems to think there is a reliable association between femininity of name and deaths.

Now in order to see which hurricanes the model retrodicts well, we can plot the implied trend over the raw data points. I'll compute and plot the expected death count, 89% interval of the expectation, and 89% interval of the expected distribution of deaths (using Poisson sampling).

R code 12.9

```
# plot raw data
plot( dat$F , dat$D , pch=16 , lwd=2 ,
    col=rangi2 , xlab="femininity (std)" , ylab="deaths" )

# compute model-based trend
pred_dat <- list( F=seq(from=-2,to=1.5,length.out=30) )
lambda <- link( m1 , data=pred_dat )
lambda.mu <- apply(lambda,2,mean)
lambda.PI <- apply(lambda,2,PI)

# superimpose trend
lines( pred_dat$F , lambda.mu )
shade( lambda.PI , pred_dat$F )

# compute sampling distribution
deaths_sim <- sim(m1,data=pred_dat)
deaths_sim.PI <- apply(deaths_sim,2,PI)

# superimpose sampling interval as dashed lines
lines( pred_dat$F , deaths_sim.PI[1,] , lty=2 )
lines( pred_dat$F , deaths_sim.PI[2,] , lty=2 )
```

This is the result:



We can't even see the 89% interval of the expected value, because it is so narrow. The sampling distribution isn't much wider itself. What you can see here is that femininity accounts for very little of the variation in deaths, especially at the high end. There's a lot of over-dispersion, which is very common in Poisson models. As a consequence, this homogenous Poisson model does a poor job for most of the hurricanes in the sample, as most of them lie outside the prediction envelop (the dashed boundaries).

Any trend also seems to be driven by a small number of extreme storms with feminine names. As you might expect, PSIS does not like this:

R code 12.10

```
stem( PSISk( m1 ) )
```

```
Some Pareto k values are very high (>1)

  The decimal point is 1 digit(s) to the left of the |

 -0 | 8521
  0 | 000112233334555667899990000011111111122233444455556666666777778899
  2 | 000011122289
  4 | 1
  6 | 87
  8 | 59
 10 |
 12 | 9
 14 | 9
 16 | 8
 18 |
 20 |
 22 | 3
```

There are four storms with k values above 1, and one above 2!

12H2. To deal with the over-dispersion seen in the previous problem, now we'll fit a Poisson model with varying rates, a gamma-Poisson model. This code will set up the data (just as in the previous problem) and fit the gamma-Poisson model:

R code 12.11

```
library(rethinking)
data(Hurricanes)
d <- Hurricanes
d$fmnnty_std <- ( d$femininity - mean(d$femininity) )/sd(d$femininity)
dat <- list( D=d$deaths , F=d$fmnnty_std )
```

```
m2 <- ulam(
    alist(
        D ~ dgampois( lambda , scale ),
        log(lambda) <- a + bF*F,
        a ~ dnorm(1,1),
        bF ~ dnorm(0,1),
        scale ~ dexp(1)
    ), data=dat , chains=4 , log_lik=TRUE )
```

Inspect the marginal posterior distributions of the parameters:

```
precis(m2)
```

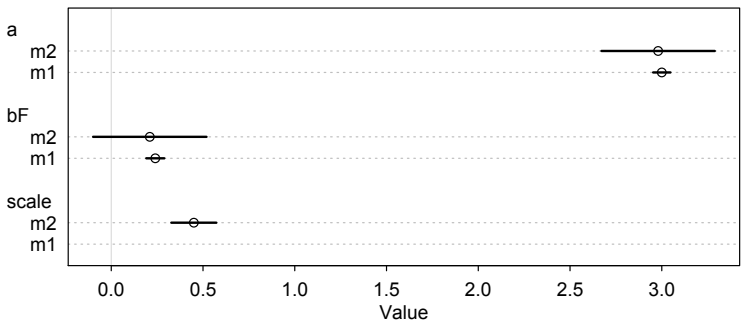
R code 12.12

```
      mean   sd  5.5% 94.5% n_eff Rhat4
a     2.98 0.16  2.73  3.23  1597     1
bF    0.21 0.16 -0.05  0.46  1435     1
scale 0.45 0.06  0.35  0.56  1652     1
```

Note that the 89% interval for `bF` now overlaps zero, because it has more than 5 times the standard deviation as the analogous parameter in model `m1`. For a head-to-head comparison, let's do a graphical `coeftab`:

```
plot(coeftab(m1,m2))
```

R code 12.13



Model `m2` has nearly the same posterior means for the intercept and slope `bF`, but much more uncertainty.

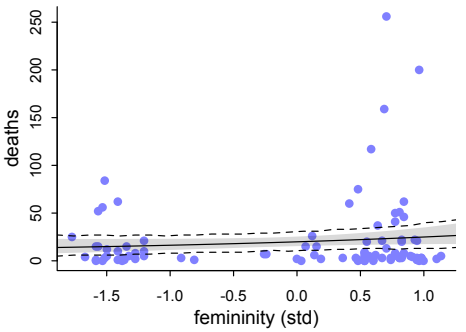
How does this translate into predictions? To find out, let's do the plotting from the previous problem over again, using model `m3` now:

```
# plot raw data
plot( dat$F , dat$D , pch=16 , lwd=2 ,
    col=rangi2 , xlab="femininity (std)" , ylab="deaths" )

# compute model-based trend
pred_dat <- list( F=seq(from=-2,to=1.5,length.out=30) )
lambda <- link(m2,data=pred_dat)
lambda.mu <- apply(lambda,2,mean)
lambda.PI <- apply(lambda,2,PI)

# superimpose trend
```

R code 12.14

```
lines( pred_dat$F , lambda.mu )
shade( lambda.PI , pred_dat$F )

# compute sampling distribution
deaths_sim <- sim(m2,data=pred_dat)
deaths_sim.PI <- apply(deaths_sim,2,PI)

# superimpose sampling interval as dashed lines
lines( pred_dat$F , deaths_sim.PI[1,] , lty=2 )
lines( pred_dat$F , deaths_sim.PI[2,] , lty=2 )
```

This is the new result:



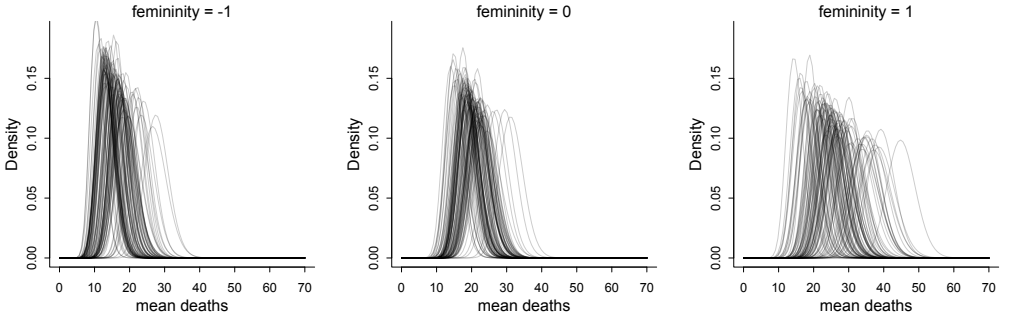
There is more uncertainty now about the relationship, and the prediction interval is wider. But the predictions are still terrible.

Now for the conceptual part of this problem: Why does including varying rates, via the gamma distribution, result in greater uncertainty in the relationship? The gamma-Poisson model allows each hurricane to have its own unique expected death rate, sampled from a common distribution that is a function of the femininity of hurricane names. We can actually plot this distribution from the posterior distribution, for any given femininity value. I'll produce three examples, plotting 100 randomly sampled gamma distributions of the rate of deaths for three different femininity values:

R code 12.15

```
post <- extract.samples(m2)

fem <- (-1) # 1 stddev below mean
for ( i in 1:100 )
    curve( dgamma2(x,exp(post$a[i]+post$bF[i]*fem),post$scale[i]) ,
        from=0 , to=70 , xlab="mean deaths" , ylab="Density" ,
        ylim=c(0,0.19) , col=col.alpha("black",0.2) ,
        add=ifelse(i==1,FALSE,TRUE) )
mtext( concat("femininity = ",fem) )
```

And just change the value assigned to `fem` above to make the other plots. Here are the plots:



Each gray curve above is a gamma distribution of mean death rates, sampled from the posterior distribution of the model. This is an inherently confusing thing: a distribution sampled from a distribution. So let's take it again, slowly. A gamma distribution is defined by two parameters: a mean and a scale. The mean in this model is controlled by the linear model and its two parameters, `a` and `bf`. The code above takes single values of `a` and `bf` from the posterior distribution and builds a single linear model. It's exponentiated in the code, because this model uses a log link. And the parameter `scale` is the scale, so you can see that in the code as well. 100 gamma distributions are drawn for each given value of femininity. This visualizes the uncertainty in the posterior about the variation in death rates. Read that again, slowly. It is a bit weird, but with a little time, it makes plenty of sense. Just like a simple parameter like an intercept has uncertainty, and the posterior distribution measures it (given a model and data), a function of parameters like a gamma distribution will also have uncertainty. Essentially there are an infinite number of gamma distributions that are possible, the the Bayesian model has considered all of them and ranked them by their plausibility. Each plot above shows 100 such gamma distributions, sampled from the posterior distribution in proportion to their plausibilities.

So now back to the explanation of why the parameters `a` and `bF` have wider posterior distributions. Once we allow any given values of `a` and `bF` to produce many different death rates, because they feed into a gamma distribution that produces variation, then many more distinct values of `a` and `bF` can be consistent with the data. This results in wider posterior distributions. The same phenomenon will reappear when we arrive at multilevel models in Chapter 13.

You might be curious how `m2` compares to `m1`, in terms of PSIS/WAIC. If so, take a look. You'll find that the effective number of parameters for `m1` is very very large. Adding a parameter to `m2` actually makes the model less prone to overfitting. Can you explain why?

12H3. This was the hypothesis in the original paper: people tend to take storms with feminine names less seriously, and so such storms are potentially more deadly, given equivalent strength. Our mission is to explore models that include both `min_pressure` and `damage_norm` as measures of storm strength. We'll interact `femininity` with each, following the notion that a storm's potential to cause death depends upon the femininity of its name.

If we make a DAG that expresses this idea, it might be $F \rightarrow D \leftarrow S$, where F is femininity of the storm's name, D is deaths, and S is storm's strength (potential to cause death). Since storm names are taken off a list in alphabetical order, and gender alternates, presumably there is no association between F and S. However to get the causal estimate right, we have to get the function relating D to F and S right. The hypothesis says it should be an interaction of some kind.

For example, here's a model that interacts `min_pressure` with `femininity`. I'll use gamma-Poisson models, given what we've seen in the previous problems.

R code 12.16

```
# standardize new predictor
# add to dat from previous problem
dat$P <- standardize(d$min_pressure)
```

```
# interaction model
m12H3a <- ulam(
    alist(
        D ~ dgampois( lambda , scale ),
        log(lambda) <- a + bF*F + bP*P + bFP*F*P,
        a ~ dnorm(1,1),
        c(bF,bP,bFP) ~ dnorm(0,1),
        scale ~ dexp(1)
    ), data=dat , chains=4 )
precis( m12H3a )
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
a      2.76 0.15  2.53  3.00  2417     1
bFP    0.30 0.14  0.07  0.53  1603     1
bP    -0.67 0.14 -0.89 -0.45  1797     1
bF     0.30 0.14  0.07  0.52  2547     1
scale  0.55 0.08  0.44  0.68  2804     1
```

The interaction coefficient is positive. Interpreting this is a bit tricky, because hurricanes get stronger as their minimum pressure gets *lower*. So it makes sense for the main effect to be negative: storms with *larger* minimum pressure cause fewer deaths. Does it also make sense for the interaction to be *positive*? As usual, I advise caution with interpreting interactions from the coefficient table. Better to construct some counterfactual predictions and figure it out that way.

Let's do that. I'll plot deaths against `min_pressure`, for both masculine and feminine storms:

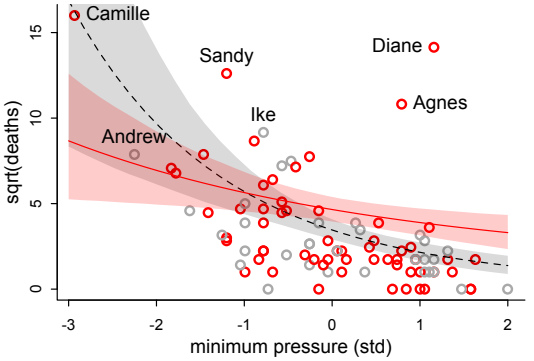
R code 12.17

```
P_seq <- seq(from=-3,to=2,length.out=30)

# 'masculine' storms
d_pred <- data.frame( F=-1 , P=P_seq )
lambda_m <- link( m12H3a , data=d_pred )
lambda_m.mu <- apply(lambda_m,2,mean)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame( F=1, P=P_seq )
lambda_f <- link( m12H3a , data=d_pred )
lambda_f.mu <- apply(lambda_f,2,mean)
lambda_f.PI <- apply(lambda_f,2,PI)

# now try plotting together
# will use sqrt scale for deaths,
#   to make differences easier to see
#   cannot use log scale, bc of zeros in data
#   note uses of sqrt() throughout code
plot( dat$P , sqrt(dat$D) ,
    pch=1 , lwd=2 , col=ifelse(dat$F>0,"red","dark gray") ,
    xlab="minimum pressure (std)" , ylab="sqrt(deaths)" )
lines( P_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , P_seq )
lines( P_seq , sqrt(lambda_f.mu) , lty=1 , col="red" )
shade( sqrt(lambda_f.PI) , P_seq , col=col.alpha("red",0.2) )
```

It'll be useful to label some of the storms:

R code 12.18

```
identify( dat$P , sqrt(dat$D) , labels=d$name )
```



The gray trend is "masculine" storms, and the red trend is "feminine" storms. So you can see clearly here that the interaction model expects feminine storms to be a little more deadly on average, but the difference between masculine and feminine storms actually decreases as pressure drops. This is not in agreement with the hypothesis. It's clear again the trend is driven by a few storms.

There's another storm damage variable, however. Let's repeat the above analysis, using `damage_norm` this time. The variable `damage_norm` is an estimate of the property damage of a storm. The notion is that this might serve as a proxy for potential to cause deaths, a better one than `min_pressure` because it may account for settlement patterns and population density. We'll use the logarithm of damage norm, because even if we expect a one-to-one relationship between damage norm and deaths, then we need to log damage norm inside the linear model, as it will be exponentiated through the link function.

R code 12.19

```
# standardize predictor
dat$S <- standardize(log(d$damage_norm))

m12H3b <- ulam(
    alist(
        D ~ dgampois( lambda , scale ),
        log(lambda) <- a + bF*F + bS*S + bFS*F*S,
        a ~ dnorm(1,1),
        c(bF,bS,bFS) ~ dnorm(0,1),
        scale ~ dexp(1)
    ), data=dat , chains=4 , log_lik=TRUE )
precis( m12H3b )
```

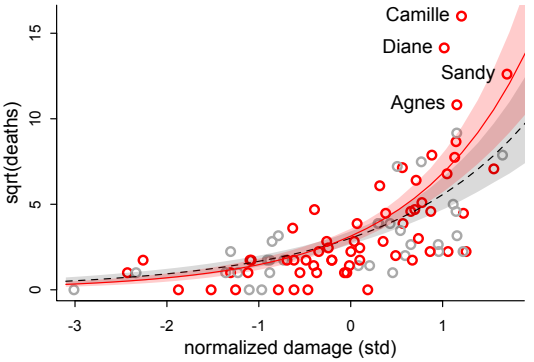
```
      mean   sd  5.5% 94.5% n_eff Rhat4
a     2.24 0.12  2.05  2.43  1834     1
bFS   0.17 0.12 -0.03  0.36  2296     1
bS    1.38 0.12  1.18  1.56  1945     1
bF    0.04 0.12 -0.15  0.22  2624     1
scale 1.06 0.18  0.80  1.36  2081     1
```

Let's look at the counterfactual predictions again:

R code 12.20

```
S_seq <- seq(from=-3.1,to=1.9,length.out=30)

# 'masculine' storms
```

```
d_pred <- data.frame( F=-1 , S=S_seq )
lambda_m <- link( m12H3b , data=d_pred )
lambda_m.mu <- apply(lambda_m,2,mean)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame( F=1 , S=S_seq )
lambda_f <- link( m12H3b , data=d_pred )
lambda_f.mu <- apply(lambda_f,2,mean)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( dat$S , sqrt(dat$D) ,
    pch=1 , lwd=2 , col=ifelse(dat$F>0,"red","dark gray") ,
    xlab="normalized damage (std)" , ylab="sqrt(deaths)" )
lines( S_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , S_seq )
lines( S_seq , sqrt(lambda_f.mu) , lty=1 , col="red" )
shade( sqrt(lambda_f.PI) , S_seq , col=col.alpha("red",0.2) )

# label some points
identify( dat$S , sqrt(dat$D) , labels=d$name )
```



Now we see the anticipated relationship: feminine storms (red trend) are associated with more deaths at higher damage norm values. But the effect is very small, and it's driven by those same few storms.

So what's going on in these data? No one really knows. But I think that there just isn't much statistical evidence for the interaction. It's driven by few storms, and most of the variation in the data have little to do with damage or femininity of names. Given that this is an observational, rather than experimental, study, what else could explain the association between femininity and deaths? The easiest explanation is that the difference is driven by a small number of storms, essentially four storms, and they happened to have feminine names. In any given time series, there will be many spurious patterns, and if we keep fishing for something with an association, we'll find it.

In this case, there is no plausible mechanism for doubling deaths just because a storm is named Andrea instead of Andrew.

12H4. I used the log transformation in the code solution to the previous problem. We can compare to the model using damage norm directly:

R code 12.21

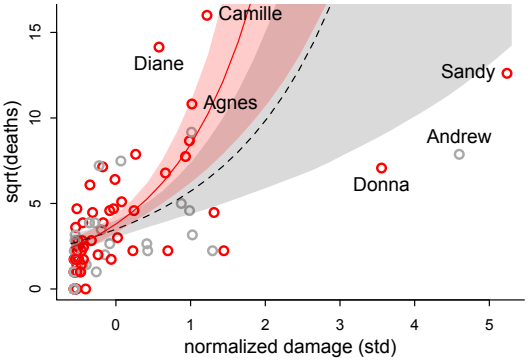
```
dat$S2 <- standardize(d$damage_norm)

m12H3c <- ulam(
    alist(
        D ~ dgampois( lambda , scale ),
        log(lambda) <- a + bF*F + bS*S2 + bFS*F*S2,
        a ~ dnorm(1,1),
        c(bF,bS,bFS) ~ dnorm(0,1),
        scale ~ dexp(1)
    ), data=dat , chains=4 , log_lik=TRUE )

compare( m12H3b , m12H3c , func=PSIS )
```

```
Some Pareto k values are high (>0.5)
        PSIS    SE dPSIS   dSE pPSIS weight
m12H3b 630.5 31.20   0.0    NA   5.3      1
m12H3c 670.9 34.13  40.4 13.58   6.9      0
```

The model using damage norm directly doesn't fit so well. Let's take a look at the posterior predictions:

R code 12.22

```
S2_seq <- seq(from=-0.6,to=5.3,length.out=30)

# 'masculine' storms
d_pred <- data.frame( F=-1 , S2=S2_seq )
lambda_m <- link( m12H3c , data=d_pred )
lambda_m.mu <- apply(lambda_m,2,mean)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame( F=1 , S2=S2_seq )
lambda_f <- link( m12H3c , data=d_pred )
lambda_f.mu <- apply(lambda_f,2,mean)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( dat$S2 , sqrt(dat$D) ,
    pch=1 , lwd=2 , col=ifelse(dat$F>0,"red","dark gray") ,
    xlab="normalized damage (std)" , ylab="sqrt(deaths)" )
lines( S2_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , S2_seq )
lines( S2_seq , sqrt(lambda_f.mu) , lty=1 , col="red" )
shade( sqrt(lambda_f.PI) , S2_seq , col=col.alpha("red",0.2) )

# label some points
identify( dat$S2 , sqrt(dat$D) , labels=d$name )
```



Since damage norm scales multiplicatively on this scale, the distances grow fast as we move right on the horizontal axis. This makes it very hard for the model to fit the data. So it misses completely the most damaging storms, so it can fit the others.

12H5. To see whether females in these data are more bothered (rate as less permissible) scenarios involving the contact principle (`contact==1`), we need an interaction effect. The reason is that just including a main effect of `male` only addresses whether men are likely to rate any scenario as more or less permissible. It doesn't address a gender difference in any particular principle.

So we need an interaction `male*contact`, at least. I'll use an index variable, and then we can easily let each parameter have a different value for men and for women. This means everything interacts with gender.

R code 12.23

```
library(rethinking)
data(Trolley)
d <- Trolley
dat <- list(
    R = d$response,
    A = d$action,
    I = d$intention,
    C = d$contact )
dat$Gid <- ifelse( d$male==1 , 1L , 2L )
```

To fit the model, we also want to let the cutpoints vary by gender. The best way to do this would be to let each gender have its own vector of cutpoints. We could do that easily in pure Stan code. It's awkward with `ulam`.

R code 12.24

```
dat$F <- 1L - d$male
m12H5 <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- a*F + bA[Gid]*A + bC[Gid]*C + BI*I ,
        BI <- bI[Gid] + bIA[Gid]*A + bIC[Gid]*C ,
        c(bA,bI,bC,bIA,bIC)[Gid] ~ dnorm( 0 , 0.5 ),
        a ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 )
    ) , data=dat , chains=4 , cores=4 )
```

Let's glance at the estimates:

R code 12.25

```
precis( m12H5 )
```

```
               mean   sd  5.5% 94.5% n_eff Rhat4
bIC[1]       -1.28 0.13 -1.50 -1.07  1534     1
bIC[2]       -1.15 0.14 -1.37 -0.93  1272     1
bIA[1]       -0.43 0.11 -0.61 -0.26  1328     1
bIA[2]       -0.42 0.12 -0.61 -0.23  1271     1
bC[1]        -0.48 0.09 -0.63 -0.34  1444     1
bC[2]        -0.21 0.10 -0.36 -0.05  1282     1
bI[1]        -0.34 0.08 -0.47 -0.22  1081     1
bI[2]        -0.26 0.09 -0.40 -0.12  1182     1
bA[1]        -0.60 0.07 -0.71 -0.48  1122     1
bA[2]        -0.34 0.08 -0.47 -0.21  1240     1
a            -0.79 0.08 -0.92 -0.65   976     1
cutpoints[1] -3.03 0.06 -3.13 -2.93  1086     1
cutpoints[2] -2.33 0.06 -2.43 -2.24  1071     1
cutpoints[3] -1.73 0.06 -1.83 -1.63  1065     1
cutpoints[4] -0.68 0.06 -0.77 -0.58  1022     1
cutpoints[5]  0.01 0.06 -0.08  0.11  1074     1
cutpoints[6]  0.94 0.06  0.85  1.04  1209     1
```

Inspect the estimates for `a` (the main effect of being female on cumulative log-odds) and `bC[2]` (the interaction effect of being both female and in a contact scenario). The main effect is negative, -0.79, but `bC[2]` is actually less negative than `bC[1]`. So in terms of the pure contact effect, women in the sample were less bothered than men. The other effects, like `bIC`, won't change this inference, because the difference between the genders is still in the wrong direction.

This isn't a causal inference. Women in the sample differ from men in may ways other than gender. A later problem gets at that.

12H6. Read in the data:

R code 12.26

```
library(rethinking)
data(Fish)
d <- Fish
str(d)
```

```
'data.frame': 250 obs. of  6 variables:
 $ fish_caught: int  0 0 0 0 1 0 0 0 0 1 ...
 $ livebait   : int  0 1 1 1 1 1 1 1 0 1 ...
 $ camper     : int  0 1 0 1 0 1 0 0 1 1 ...
 $ persons    : int  1 1 1 2 1 4 3 4 3 1 ...
 $ child      : int  0 0 0 1 0 2 1 3 2 0 ...
 $ hours      : num  21.124 5.732 1.323 0.548 1.695 ...
```

I gave you to have a lot of freedom here. So I'm not expecting any particular model. But I do expect you to have constructed a correct zero-inflated Poisson (ZIPoisson) model and use the exposure/offset correctly. So here's an example, using a subset of the predictors:

R code 12.27

```
d$loghours <- log(d$hours)
m12H6a <- ulam(
    alist(
        fish_caught ~ dzipois( p , mu ),
        logit(p) <- a0 + bp0*persons + bc0*child,
```

```
        log(mu) <- a + bp*persons + bc*child + loghours,
        c(a0,a) ~ dnorm(0,1),
        c(bp0,bc0,bp,bc) ~ dnorm(0,0.5)
    ) , data=d , chains=4 )
```

There are two key things to note. First, the parameters are different in the two linear models. No parameters are shared among them. This allows the same predictors to potentially influence each part of the process in different ways. Second, the offset `loghours` is added onto the end of the linear model for `log(mu)`, the mean of the Poisson process producing fish. This corrects for the fact that some visitors stayed longer, so had more opportunity to catch fish.

Here are the marginal posterior distributions from the model above:

R code 12.28

```
precis(m12H6a)
```

```
     mean   sd  5.5% 94.5% n_eff Rhat4
a   -2.25 0.14 -2.49 -2.04  1159     1
a0   0.41 0.46 -0.34  1.12  1455     1
bc   0.53 0.09  0.38  0.67  1327     1
bp   0.66 0.04  0.60  0.73  1152     1
bc0  0.50 0.46 -0.28  1.20  1472     1
bp0 -0.76 0.21 -1.10 -0.43  1460     1
```

Note that the parameters `a0`, `bp0`, and `bc0` are on the logit (log-odds) scale, while the parameters `a`, `bp`, and `bc` are on the log scale.

Here's an example of generating predictions. We need to consider each component of the process separately, and actually `link` will respect this, because it computes the inverse-link values for all linear models in the formula list you provided:

R code 12.29

```
zip_link <- link(m12H6a)
str(zip_link)
```

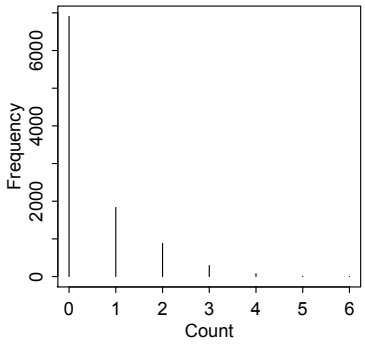
```
List of 2
 $ p : num [1:2000, 1:250] 0.441 0.345 0.459 0.503 0.351 ...
 $ mu: num [1:2000, 1:250] 4.38 4.46 4.01 4.37 4.34 ...
```

The `p` matrix is probabilities of not fishing for each sample/case, and the `mu` matrix is the expected number of fish caught (conditional on fishing) for each sample/case.

So what do these matrices imply about posterior predictions? You have to use the model, as always. The `p` values provide predictions for excess zeros, and the `mu` values for the Poisson process (which can also produce zeros). The an observation requires using both. For example, if $p = 0.5$ and $\mu = 1$ (fixed right now for ease of understand), then the implied predictive distribution is:

R code 12.30

```
zeros <- rbinom(1e4,1,0.5)
obs_fish <- (1-zeros)*rpois(1e4,1)
simplehist(obs_fish)
```



Now luckily `sim` understands all of this, because it knows the model already. So you can just:

R code 12.31

```
fish_sim <- sim(m12H6a)
str(fish_sim)
```

```
 num [1:1000, 1:250] 0 0 0 0 0 0 4 1 1 0 ...
```

And these simulations integrate over the posterior uncertainty, so there is one simulation for each sample.

You could go on to summarize and plot these simulations. But what we actually want is counterfactual posterior predictions. So let's assume for example a party of 1 person spending 1 hour in the park. Let's do that now:

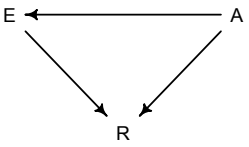
R code 12.32

```
# new data
pred_dat <- list(
    loghours=log(1), # note that this is zero, the baseline rate
    persons=1,
    child=0 )

# sim predictions - want expected number of fish, but must use both processes
fish_link <- link( m12H6a , data=pred_dat )

# summarize
p <- fish_link$p
mu <- fish_link$mu
( expected_fish_mean <- mean( (1-p)*mu ) )
( expected_fish_PI <- PI( (1-p)*mu ) )
```

```
[1] 0.1191649

       5%       94%
0.0925886 0.1471911
```

What this means is that one person is expected to extract on average 0.12 fish per hour, with the interval shown. This estimate allows for the zero-inflation, so it accounts for the probability that the person is not fishing at all, depressing the expected value.

12H7. This is my DAG:



Age could influence both response R and education E. It could influence R, because people at different ages could have different attitudes. Age could influence education, because the longer you have lived, the more education you could have completed (up to a point). It's like the age causing marriage example from earlier in the course.

To evaluate the causal influence of E on R, we need to block the back-door from E through A to R. So we need a model that conditions on both E and A. Then the estimate for E should be the causal influence of E.

Let's set up the data:

R code 12.33

```
data(Trolley)
d <- Trolley

# recode these in order
edu_levels <- c( 6 , 1 , 8 , 4 , 7 , 2 , 5 , 3 )
d$edu_new <- edu_levels[ d$edu ]

idx <- 1:nrow(d)
dat <- list(
    y = d$response[idx] ,
    A = d$action[idx],
    I = d$intention[idx],
    C = d$contact[idx],
    E = as.integer( d$edu_new[idx] ),
    edu_norm = normalize( d$edu_new[idx] ),
    age = standardize( d$age[idx] ),
    alpha = rep(2,7) # g prior
)
```

Note that I standardized age above. And now here is the model with both E and A. Really all we need to do is add age to the linear model. The rest you can copy from the model in the chapter.

R code 12.34

```
m12H7 <- ulam(
    alist(
        y ~ ordered_logistic( phi , cutpoints ),
        phi <- bE*sum( delta_shell[1:E] ) + bA*A + bC*C + BI*I + bAge*age,
        BI <- bI + bIA*A + bIC*C ,
        c(bA,bI,bC,bIA,bIC,bE,bAge) ~ normal( 0 , 0.5 ),
        cutpoints ~ normal( 0 , 1.5 ),
        vector[8]: delta_shell <<- append_row( 0 , delta ),
        simplex[7]: delta ~ dirichlet( alpha )
    ), data=dat , chains=4 , cores=4 )
precis(m12H7)
```

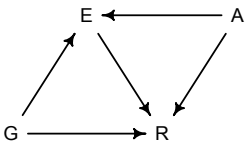
```
      mean   sd  5.5% 94.5% n_eff Rhat4
bAge -0.10 0.02 -0.13 -0.07   464  1.00
bE    0.21 0.13 -0.05  0.36   152  1.02
bIC  -1.24 0.09 -1.39 -1.09  1235  1.00
bIA  -0.43 0.08 -0.56 -0.31  1186  1.00
```

```
bC   -0.34 0.07 -0.46 -0.23  1342  1.00
bI   -0.29 0.06 -0.38 -0.20  1140  1.00
bA   -0.47 0.05 -0.56 -0.39  1166  1.00
```

You may recall from the chapter that education has a negative effect in the model without age. Now that we include age, education has a positive influence (with some overlap with zero). So age has indeed soaked up some of the previous influence assigned to education. The back-door may be real.

I'd summarize this model, assuming this DAG is true, as saying that age causes people to give slightly lower responses. This could be a cohort effect, and not a causal influence of age. Either way, it is small. Education seems to cause higher responses (more approval). This suggests that education trains people to see some or all of the features A,I,C as more permissible. A model that interacted education with each might shed more light on things. Remember: A DAG doesn't say whether you need an interaction effect or not. That is a separate problem.

12H8. This is my DAG:



The only new part is the fork coming from gender G. It influences both R and E. If this DAG is correct, then is E confounded? Yes. Let's see the minimum adjustment set necessary to measure the causal influence of E:

R code 12.35

```
library(dagitty)
dag2 <- dagitty("dag{
    E -> R <- A
    A -> E
    G -> E
    G -> R
}")
adjustmentSets( dag2 , exposure="E" , outcome="R" , effect="total" )
```

```
 { A, G }
```

This means we'd need to condition on both A and G to get an un-confounded estimate for E. Why? There is a back-door from E to G to R. It is just like the back-door through A.

Here's the model we need, which includes education, age, and gender (female dummy variable):

R code 12.36

```
dat$female <- ifelse( d$male==1 , 0L , 1L )
m12H8 <- ulam(
    alist(
        y ~ ordered_logistic( phi , cutpoints ),
        phi <- bE*sum( delta_shell[1:E] ) + bA*A + bC*C + BI*I +
               bAge*age + bF*female,
        BI <- bI + bIA*A + bIC*C ,
        c(bA,bI,bC,bIA,bIC,bE,bAge,bF) ~ normal( 0 , 0.5 ),
        cutpoints ~ normal( 0 , 1.5 ),
        vector[8]: delta_shell <<- append_row( 0 , delta ),
        simplex[7]: delta ~ dirichlet( alpha )
    ), data=dat , chains=4 , cores=4 )
precis(m12H8)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
bF   -0.56 0.04 -0.62 -0.51  2069  1.00
bAge -0.07 0.02 -0.11 -0.04   889  1.00
bE    0.02 0.17 -0.26  0.25   483  1.01
bIC  -1.27 0.10 -1.42 -1.12  1224  1.00
bIA  -0.44 0.08 -0.56 -0.31   895  1.00
bC   -0.34 0.07 -0.45 -0.24  1224  1.00
bI   -0.29 0.06 -0.38 -0.20   863  1.00
bA   -0.48 0.05 -0.56 -0.40   943  1.00
```

Age is still negative (and weak), while education is right near zero and straddles both sides. Gender seems to have accounted for all of the previous influenced assigned to education. It looks like female respondents gave lower average responses—indicating less approval.

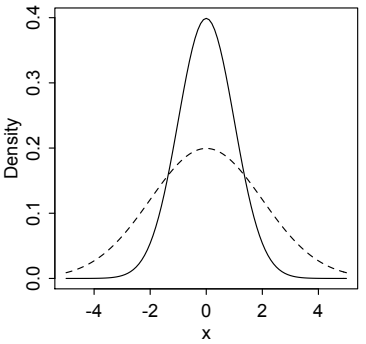
It would be worth figuring out how gender is associated with education in this sample. It could be true for example that some education levels under-sampled men or women, and this leads to another kind of confound. Consider for example if older men are less likely to respond, so the sample becomes increasingly female with age. Then education level will also be increasingly female with age. Since the sample is not a representative sample of the population, there are probably some biases of this sort.

13. Chapter 13 Solutions

13E1. Option (a) will produce more shrinkage, because the prior is more concentrated. If this isn't obvious from the smaller standard deviation, you can always plot the two priors and compare:

R code 13.1

```
curve( dnorm(x,0,1) , from=-5 , to=5 , ylab="Density" )
curve( dnorm(x,0,2) , add=TRUE , lty=2 )
```



Since option (a), shown by the solid density, piles up more mass around zero, it will pull extreme values closer to zero.

13E2. All that is really required to convert the model to a multilevel model is to take the prior for the vector of intercepts, α_{GROUP}, and make it adaptive. This means we define parameters for its mean and standard deviation. Then we assign these two new parameters their own priors, *hyperpriors*. This is what it looks like:

$$\begin{aligned} y_i &\sim \text{Binomial}(1, p_i) \\ \text{logit}(p_i) &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 1.5) \\ \sigma_\alpha &\sim \text{Exponential}(1) \end{aligned}$$

The exact hyperpriors you assign don't matter here. Since this problem has no data context, it isn't really possible to say what sensible priors would be. Note also that an exponential prior on σ_α is just as sensible, absent context, as the half-Cauchy prior.

13E3. This is very similar to the previous problem. The only trick here is to notice that there is already a standard deviation parameter, σ. But that standard deviation is for the *residuals*, at the top

level. We'll need yet another standard deviation for the varying intercepts:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma &\sim \text{Exponential}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 5) \\ \sigma_\alpha &\sim \text{Exponential}(0, 1) \end{aligned}$$

13E4. You can just copy the answer from problem **13E2** and swap out the binomial likelihood for a Poisson, taking care to change the link function from logit to log. Of course you'd need to rethink the priors, and those will depend upon what the outcome is. But here is the right structure:

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i) \\ \log(\lambda_i) &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 1) \\ \sigma_\alpha &\sim \text{Exponential}(0, 1) \end{aligned}$$

Under the hood, all multilevel models are alike. It doesn't matter which likelihood function rests at the top. Take care, however, to reconsider priors. The scale of the data and parameters is likely quite different for a Poisson model. Absent any particular context in this problem, you can't recommend better priors. But in real work, it's good to think about reasonable values and provide regularizing priors on the relevant scale.

13E5. The cross-classified model adds another varying intercept type. This is no harder than duplicating the original varying intercepts structure. But you have to take care now not to over-parameterize the model by having a hyperprior mean for both intercept types. You can do this by just assigning one of the adaptive priors a mean of zero. Suppose for example that the second cluster type is DAY:

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i) \\ \log(\lambda_i) &= \alpha_{\text{GROUP}[i]} + \alpha_{\text{DAY}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_{\text{GROUP}}) \\ \alpha_{\text{DAY}} &\sim \text{Normal}(0, \sigma_{\text{DAY}}) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 1) \\ \sigma_{\text{GROUP}} &\sim \text{Exponential}(0, 1) \\ \sigma_{\text{DAY}} &\sim \text{Exponential}(0, 1) \end{aligned}$$

Or you can just pull the mean intercept out of both priors and put it in the linear model:

$$
\begin{aligned}
y_i &\sim \text{Poisson}(\lambda_i) \\
\log(\lambda_i) &= \bar{\alpha} + \alpha_{\text{GROUP}[i]} + \alpha_{\text{DAY}[i]} + \beta x_i \\
\alpha_{\text{GROUP}} &\sim \text{Normal}(0, \sigma_{\text{GROUP}}) \\
\alpha_{\text{DAY}} &\sim \text{Normal}(0, \sigma_{\text{DAY}}) \\
\beta &\sim \text{Normal}(0, 1) \\
\bar{\alpha} &\sim \text{Normal}(0, 1) \\
\sigma_{\text{GROUP}} &\sim \text{Exponential}(0, 1) \\
\sigma_{\text{DAY}} &\sim \text{Exponential}(0, 1)
\end{aligned}
$$

These are exactly the same model. Although as you'll see later in Chapter 14, these different forms might be more or less efficient in sampling.

13M1. First, let's set up the data list:

R code 13.2

```
library(rethinking)
data(reedfrogs)
d <- reedfrogs

dat <- list(
    S = d$surv,
    n = d$density,
    tank = 1:nrow(d),
    pred = ifelse( d$pred=="no" , 0L , 1L ),
    size_ = ifelse( d$size=="small" , 1L , 2L )
)
```

Now to define a series of models. The first is just the varying intercepts model from the text:

R code 13.3

```
m1.1 <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank],
        a[tank] ~ normal( a_bar , sigma ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

The other models just incorporate the predictors, as ordinary regression terms.

R code 13.4

```
# pred
m1.2 <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank] + bp*pred,
        a[tank] ~ normal( a_bar , sigma ),
        bp ~ normal( -0.5 , 1 ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

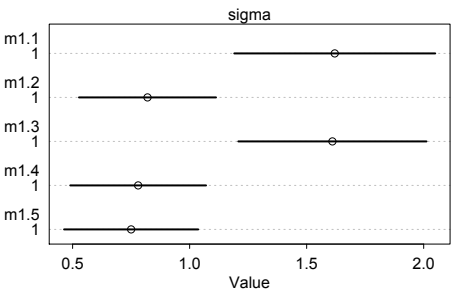
```
# size
m1.3 <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank] + s[size_],
        a[tank] ~ normal( a_bar , sigma ),
        s[size_] ~ normal( 0 , 0.5 ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 , log_lik=TRUE )

# pred + size
m1.4 <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank] + bp*pred + s[size_],
        a[tank] ~ normal( a_bar , sigma ),
        bp ~ normal( -0.5 , 1 ),
        s[size_] ~ normal( 0 , 0.5 ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 , log_lik=TRUE )

# pred + size + interaction
m1.5 <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a_bar + z[tank]*sigma + bp[size_]*pred + s[size_],
        z[tank] ~ normal( 0 , 1 ),
        bp[size_] ~ normal( -0.5 , 1 ),
        s[size_] ~ normal( 0 , 0.5 ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

I coded the interaction model using a non-centered parameterization. The interaction itself is done by creating a `bp` parameter for each size value. In this way, the effect of pred depends upon size.

Let's look at all the `sigma` posterior distributions:

R code 13.5

```
plot( coeftab( m1.1 , m1.2 , m1.3 , m1.4 , m1.5 ), pars="sigma" )
```



The two models that omit predation, `m1.1` and `m1.3`, have larger values of sigma. This is because predation explains some of the variation among tanks. So when you add it to the model, the variation in the tank intercepts gets smaller. We'll examine this in more detail in a the last problem of this chapter.

The general point here is that the model with only intercepts measures the variation among tanks, but does nothing to explain it. As we add treatment variables, the variation should shrink, even though the total variation in the data of course stays the same.

13M2. The WAIC scores:

R code 13.6

```
compare( m1.1 , m1.2 , m1.3 , m1.4 , m1.5 )
```

```
      WAIC   SE dWAIC  dSE pWAIC weight
m1.2 198.8 8.97   0.0   NA  19.1   0.31
m1.1 199.2 7.10   0.4 5.65  20.5   0.25
m1.5 199.7 8.96   0.9 3.22  19.1   0.20
m1.4 200.3 8.83   1.4 2.13  19.2   0.15
m1.3 201.1 7.28   2.2 5.61  21.4   0.10
```

These models are really very similar in expected out-of-sample accuracy. The tank variation is huge. But take a look at the posterior distributions for predation and size. You'll see that predation does seem to matter, as you'd expect. Size matters a lot less. So while predation doesn't explain much of the total variation, there is plenty of evidence that it is a real effect. Remember: We don't select a model using WAIC (or PSIS). A predictor can make little difference in total accuracy but still be a real causal effect.

If you inspect the posterior distributions, you'll see that the coefficients for predation are further from zero than are the coefficients for size. This is consistent with the model rankings.

The fact that the tank-only model does so well does not mean that predation and size do not matter. The posterior distributions clearly suggest that predation and size do matter. It only means that much more variation exists among tanks for other reasons. As always, prediction and inference are just different tasks.

13M3. Now we want a slightly modified version of model `m1.1` from problem **13M1**. We just replace the Gaussian adaptive prior with a similar Cauchy prior. The

R code 13.7

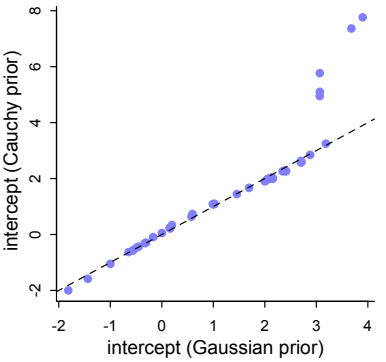
```
m1.1c <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank],
        a[tank] ~ dcauchy( a_bar , sigma ),
        a_bar ~ normal( 0 , 1 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 ,
    log_lik=TRUE , control=list(adapt_delta=0.99) )
```

You might have some trouble sampling efficiently from this posterior, on account of the long tails of the Cauchy. These result in the intercepts being poorly identified. You saw a simple example of this problem in Chapter 9, when you met MCMC and learned about diagnosing bad chains. This topic will come up in more detail in Chapter 13. In any event, be sure to check the chains carefully and sample more if you need to.

The problem asked you to compare the posterior means of the a parameters. Plotting the posterior means will be a lot more meaningful than just looking at the values.

R code 13.8

```
post1 <- extract.samples(m1.1)
a_tank1 <- apply(post1$a,2,mean)
post2 <- extract.samples(m1.1c)
a_tank2 <- apply(post2$a,2,mean)
plot( a_tank1 , a_tank2 , pch=16 , col=rangi2 ,
    xlab="intercept (Gaussian prior)" , ylab=" intercept (Cauchy prior)" )
abline(a=0,b=1,lty=2)
```



The dashed line show the values for which the intercepts are equal in the two models. You can see that for the majority of tank intercepts, the Cauchy model actually produces posterior means that are essentially the same as those from the Gaussian model. But the extremely large intercepts, under the Gaussian prior, are very much more extreme under the Cauchy prior. For those tanks, on the righthand side of the plot, all of the tadpoles survived. So using only the data from each tank alone, the log-odds of survival are infinite. The adaptive prior applies pooling that shrinks those log-odds inwards from infinity, thankfully. But the Gaussian prior causes more shrinkage of the extreme values than the Cauchy prior does. That is what accounts for those 5 extreme points on the right of the plot above.

13M4. To sample this posterior efficiently, we'll want to us a non-centered parameterization. The centered parameterization will work, but Stan will complain a lot. I'll use `transpars` to include the centered intercepts in the posterior:

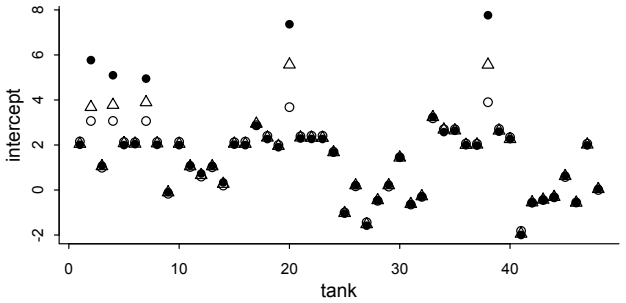
R code 13.9

```
m1.1t <- ulam(
    alist(
        S ~ binomial( n , p ),
        logit(p) <- a[tank],
        transpars> vector[tank]:a <<- a_bar + z*sigma,
        z[tank] ~ dstudent( 2 , 0 , 1 ),
        a_bar ~ normal( 0 , 1 ),
        sigma ~ exponential( 1 )
    ), data=dat , chains=4 , cores=4 ,
    log_lik=TRUE , control=list(adapt_delta=0.99) )
```

To compare all three posterior distributions:

R code 13.10

```
postn <- extract.samples( m1.1 )
an <- colMeans( postn$a )
postc <- extract.samples( m1.1c )
ac <- colMeans( postc$a )
postt <- extract.samples( m1.1t )
at <- colMeans( postt$a )
plot( NULL , xlim=c(1,48) , ylim=range(c(an,ac,at)) ,
    xlab="tank" , ylab="intercept" )
points( 1:48 , an )
points( 1:48 , ac , pch=16 )
points( 1:48 , at , pch=2 )
```



In most tanks, the posterior means are essentially the same. There are 5 tanks however where they differ. In each of these, there is the same ordering: Cauchy (filled) is most extreme, followed by Student-t (triangle), and then by Gaussian (open circle). This results from Cauchy having the thickest tails, followed by Student-t and then Gaussian. Thicker tails produce less shrinkage.

13M5. This is much like the model in the chapter, just with the two varying intercept means inside the two priors, instead of one mean outside both priors (inside the linear model). Since there are two parameters for the means, one inside each adaptive prior, this model is over-parameterized: an infinite number of different values of α and γ will produce the same sum $\alpha + \gamma$. The parameter γ is redundant, in other words. This will produce a poorly-identified posterior. It's best to avoid specifying a model like this. Now you'll see why.

Here's the code to prepare the data and fit the model:

R code 13.11

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$treatment <- 1 + d$prosoc_left + 2*d$condition
dat_list <- list(
    pulled_left = d$pulled_left,
    actor = d$actor,
    block_id = d$block,
    treatment = as.integer(d$treatment) )

m13M5 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + g[block_id] + b[treatment] ,
```

```
        b[treatment] ~ dnorm( 0 , 0.5 ),
      ## adaptive priors
        a[actor] ~ dnorm( a_bar , sigma_a ),
        g[block_id] ~ dnorm( g_bar , sigma_g ),
      ## hyper-priors
        a_bar ~ dnorm( 0 , 1.5 ),
        g_bar ~ dnorm( 0 , 1.5 ),
        sigma_a ~ dexp(1),
        sigma_g ~ dexp(1)
    ) , data=dat_list , chains=4 , cores=4 , log_lik=TRUE )
```

And just to make life easier, here's the code to re-fit the model from the chapter:

R code 13.12

```
m13.4 <- ulam(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + g[block_id] + b[treatment] ,
        b[treatment] ~ dnorm( 0 , 0.5 ),
      ## adaptive priors
        a[actor] ~ dnorm( a_bar , sigma_a ),
        g[block_id] ~ dnorm( 0 , sigma_g ),
      ## hyper-priors
        a_bar ~ dnorm( 0 , 1.5 ),
        sigma_a ~ dexp(1),
        sigma_g ~ dexp(1)
    ) , data=dat_list , chains=4 , cores=4 , log_lik=TRUE )
```

Now lets look at the `precis` output of each model:

R code 13.13

```
precis(m13.4 , 2 , pars=c("a_bar","b") )
precis(m13M5 , 2 , pars=c("a_bar","b","g_bar") )
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
a_bar  0.61 0.72 -0.58  1.75   949  1.01
b[1]  -0.15 0.32 -0.65  0.36   449  1.01
b[2]   0.38 0.32 -0.14  0.89   445  1.01
b[3]  -0.49 0.32 -1.00  0.01   440  1.01
b[4]   0.26 0.31 -0.22  0.77   439  1.01

       mean   sd  5.5% 94.5% n_eff Rhat4
a_bar  0.62 1.23 -1.35  2.55    22  1.11
b[1]  -0.16 0.29 -0.58  0.32   160  1.03
b[2]   0.39 0.29 -0.06  0.88   394  1.02
b[3]  -0.48 0.30 -0.96  0.02   492  1.01
b[4]   0.24 0.31 -0.21  0.75    40  1.07
g_bar  0.23 1.06 -1.34  2.01   150  1.02
```

The new model, `m13M5`, samples quite poorly. The `n_eff` values are much lower, and the `Rhat` values are larger. You may also have noticed that it samples slowly. This is what happens when you over-parameterize the intercept. Notice however that the inferences about the slopes are practically identical. So even though the over-parameterized model is inefficient, it has identified the slope parameters.

13M6. We can compute the posterior distributions with 4 simple `ulam` models:

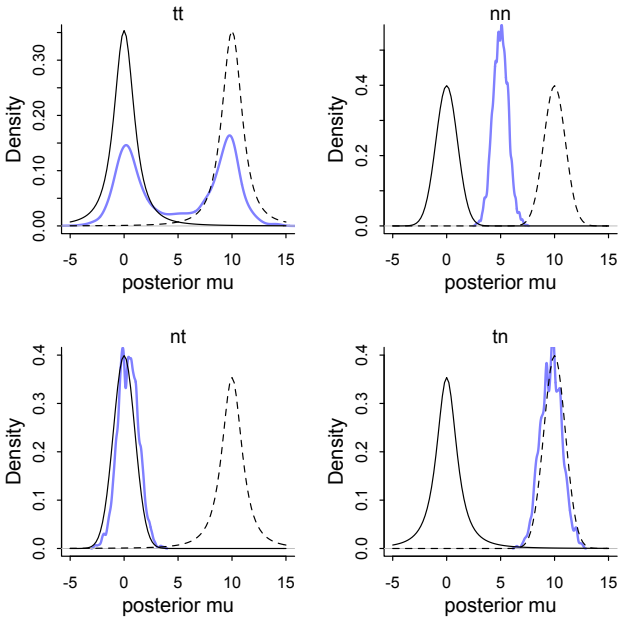
R code 13.14

```
mtt <- ulam(
    alist(
        y ~ dstudent(2,mu,1),
        mu ~ dstudent(2,10,1)
    ), data=list(y=0) , chains=4 )

mnn <- ulam(
    alist(
        y ~ dnorm(mu,1),
        mu ~ dnorm(10,1)
    ), data=list(y=0) , chains=4 )

mtn <- ulam(
    alist(
        y ~ dstudent(2,mu,1),
        mu ~ dnorm(10,1)
    ), data=list(y=0) , chains=4 )

mnt <- ulam(
    alist(
        y ~ dnorm(mu,1),
        mu ~ dstudent(2,10,1)
    ), data=list(y=0) , chains=4 )
```

Now to plot each posterior (blue) against the corresponding likelihoods (solid black) and priors (dashed):

R code 13.15

```
par(mfrow=c(2,2),cex=1.05)

p <- extract.samples(mtt)
dens(p$mu , xlim=c(-5,15) , ylim=c(0,0.35) , lwd=2 , col=rangi2 , xlab="posterior mu" )
mtext("tt")
curve( dstudent(0,2,x,1) , add=TRUE , lty=1 ) # lik
curve( dstudent(x,2,10,1) , add=TRUE , lty=2 ) # prior

p <- extract.samples(mnn)
dens(p$mu, xlim=c(-5,15), ylim=c(0,0.55), lwd=2 , col=rangi2, xlab="posterior mu")
mtext("nn")
curve( dnorm(0,x,1) , add=TRUE , lty=1 ) # lik
curve( dnorm(x,10,1) , add=TRUE , lty=2 ) # prior

p <- extract.samples(mnt)
dens(p$mu, xlim=c(-5,15), ylim=c(0,0.4), lwd=2 , col=rangi2, xlab="posterior mu")
mtext("nt")
curve( dnorm(0,x,1) , add=TRUE , lty=1 ) # lik
curve( dstudent(x,2,10,1) , add=TRUE , lty=2 ) # prior

p <- extract.samples(mtn)
dens(p$mu, xlim=c(-5,15), ylim=c(0,0.4), lwd=2 , col=rangi2, xlab="posterior mu")
mtext("tn")
curve( dstudent(0,2,x,1) , add=TRUE , lty=1 ) # lik
curve( dnorm(x,10,1) , add=TRUE , lty=2 ) # prior
```



The tt model is perhaps the most surprising—it has two peaks, one at the likelihood peak and another at the prior peak. The other three models have one posterior mode, but in different places. The nn model is a compromise between the likelihood and prior. The nt model prefers the likelihood and the tn model prefers the prior.

The explanation for this pattern is that the Student-t distribution, with its thick tails, does not aggressively pull posterior mass towards itself like the Gaussian, with its thin tails, does. So with both a Student-t likelihood and prior, neither dominates, and we end up with two modes. With two Gaussians, they tug each other to the middle. When either the likelihood or the prior is Student-t, the Student-t loses to the Gaussian.

There are two important lessons here. The first is that it is not easy to guess how prior and likelihood combine to form the posterior. Things like thickness of tails matter. The second is that we can use this fact to control what happens when the data (likelihood) and prior are incompatible. If we use thick-tailed priors, then the data will dominate. If instead we trust the prior more than the data (because of for example poor data quality), we might instead want to use a thick-tailed likelihood and thin-tailed prior. With enough data, the likelihood will still dominate.

13H1. Loading the data and prepping the data list:

R code 13.16

```
library(rethinking)
data(bangladesh)
d <- bangladesh
d$district_id <- as.integer(as.factor(d$district))

dat_list <- list(
    C = d$use.contraception,
    did = d$district_id )
```

Now for the ordinary fixed effect model:

R code 13.17

```
m13H1.1 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did],
        a[did] ~ normal( 0 , 1.5 )
    ) , data=dat_list , chains=4 , cores=4 , log_lik=TRUE )
```

And the varying intercepts model:

R code 13.18

```
m13H1.2 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did],
        a[did] ~ normal( a_bar , sigma ),
        a_bar ~ normal( 0 , 1.5 ),
        sigma ~ exponential( 1 )
    ) , data=dat_list , chains=4 , cores=4 , log_lik=TRUE )
```

Now let's extract the samples, compute posterior mean probabilities in each district, and plot it all. I'll plot fixed effects in blue, varying effects as open circles, and the raw empirical proportion in each district as a plus symbol.

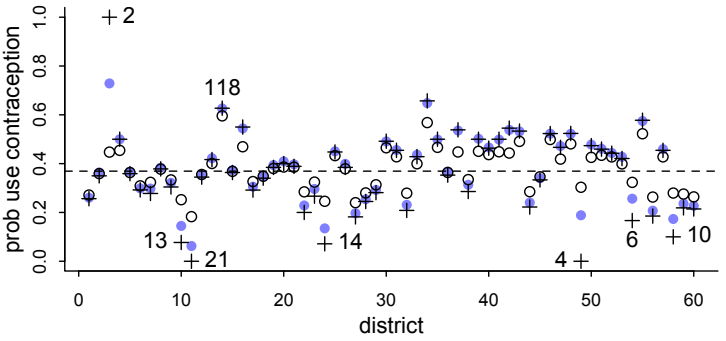
R code 13.19

```
post1 <- extract.samples( m13H1.1 )
post2 <- extract.samples( m13H1.2 )

p1 <- apply( inv_logit(post1$a) , 2 , mean )
p2 <- apply( inv_logit(post2$a) , 2 , mean )

# compute raw estimate from data in each district
t3 <- table( d$use.contraception , d$district_id )
n_per_district <- colSums( t3 )
p_raw <- as.numeric( t3[2,]/n_per_district )

nd <- max(dat_list$did)
plot( NULL , xlim=c(1,nd) , ylim=c(0,1) , ylab="prob use contraception" ,
    xlab="district" )
points( 1:nd , p1 , pch=16 , col=rangi2 )
points( 1:nd , p2 )
points( 1:nd , p_raw , pch=3 )
abline( h=mean(inv_logit(post2$a_bar)) , lty=2 )
```

Now I'll label a few points with the sample size in the district:

R code 13.20

```
identify( 1:nd , p_raw , labels=n_per_district )
```



As you'd expect, the varying intercepts (open circles) are shrunk towards the mean (the dashed line) relative to both the fixed intercepts (blue circles) and the raw proportions (plus symbols). Some are shrunk more than others. The third district from the left shrunk a lot. Let's look at the sample size in each district:

R code 13.21

```
table(d$district_id)
```

```
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
117  20   2  30  39  65  18  37  23  13  21  29  24 118  22  20  24  47  26  15

 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
 18  20  15  14  67  13  44  49  32  61  33  24  14  35  48  17  13  14  26  41

 41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
 26  11  45  27  39  86  15  42   4  19  37  61  19   6  45  27  33  10  32  42
```

District 3 has only 2 women sampled. So it shrinks a lot. There are couple of other districts, like 49 and 54, that also have very few women sampled. But their fixed estimates aren't as extreme, so they don't shrink as much as district 3 does.

All of this is explained by partial pooling, of course.

13H2. First, let's load the data and re-run the old model from Chapter 12:

R code 13.22

```
data(Trolley)
d <- Trolley

dat <- list(
    R = d$response,
    A = d$action,
    I = d$intention,
    C = d$contact )

m13H2.1 <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- bA*A + bC*C + BI*I ,
        BI <- bI + bIA*A + bIC*C ,
        c(bA,bI,bC,bIA,bIC) ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 )
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

Now to run the varying intercept model, we need to build a valid individual ID variable. The IDs in the data are long tags, so we can coerce them to integers in many ways. What is important is that the index values go from 1 to the number of individuals.

R code 13.23

```
dat$id <- coerce_index( d$id )
```

Now we can run the model. The only additions here are the `a[id]` in the linear model and the adaptive prior for it. But I'll show the code for both the centered and non-centered parameterizations. The non-centered version should sample better. But both work.

R code 13.24

```
m13H2.2 <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- a[id] + bA*A + bC*C + BI*I ,
        BI <- bI + bIA*A + bIC*C ,
        a[id] ~ normal( 0 , sigma ),
        c(bA,bI,bC,bIA,bIC) ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 ),
        sigma ~ exponential(1)
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )

m13H2.2z <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- z[id]*sigma + bA*A + bC*C + BI*I ,
        BI <- bI + bIA*A + bIC*C ,
        z[id] ~ normal( 0 , 1 ),
        c(bA,bI,bC,bIA,bIC) ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 ),
        sigma ~ exponential(1)
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

We can begin by comparing the posterior distributions. The original coefficients are:

R code 13.25

```
precis(m13H2.1)
```

```
     mean   sd  5.5% 94.5% n_eff Rhat4
bIC -1.24 0.10 -1.39 -1.07   997     1
bIA -0.43 0.08 -0.56 -0.31  1022     1
bC  -0.34 0.07 -0.45 -0.23   902     1
bI  -0.29 0.05 -0.38 -0.20   817     1
bA  -0.47 0.05 -0.55 -0.39  1013     1
```

And the new ones, having added the individual IDs, are:

R code 13.26

```
precis(m13H2.2z)
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
bIC   -1.67 0.10 -1.82 -1.51  1007  1.00
bIA   -0.56 0.08 -0.69 -0.43   870  1.00
bC    -0.45 0.07 -0.57 -0.34   927  1.00
bI    -0.39 0.06 -0.48 -0.29   837  1.00
```

```
bA    -0.65 0.06 -0.74 -0.56   840  1.00
sigma  1.90 0.08  1.78  2.04   159  1.01
```

Everything has gotten more negative. This is because there is a lot of individual variation in average rating—look at the distribution for `sigma`. That is on the logit scale, so that's a lot of variation on the probability scale. That variation in average rating was hiding some of the effect of the treatments. We get more precision by conditioning on individual.

The WAIC comparison can also help show how much variation comes from individual differences in average rating:

R code 13.27

```
compare( m13H2.1 , m13H2.2z )
```

```
           WAIC     SE  dWAIC    dSE pWAIC weight
m13H2.2z 31055.5 179.35    0.0     NA 355.1      1
m13H2.1  36928.4  80.70 5872.9 173.49  10.5      0
```

The WAIC difference is massive. This is consistent with individual variation in average rating being a major effect in this sample.

This is all quite typical of likert-scale data, in my experience. Individuals anchor on different points and this adds noise. When we have repeat samples from the same individual, we can condition away some of that noise and get more precise estimates of the treatment effects.

13H3. The cross-classified model will add additional varying intercepts for each `story` in the data. There are 12 different stories, which are repeated across individuals. So we have repeat measures on `story` just as we have repeat measures on individual `id`.

So let's load the data again and build the index variable for `story`:

R code 13.28

```
library(rethinking)
data(Trolley)
d <- Trolley
dat <- list(
    R = d$response,
    A = d$action,
    I = d$intention,
    C = d$contact,
    Sid = coerce_index(d$story),
    id = coerce_index(d$id) )
```

The cross-classified model just needs another set of varying intercepts clustered on story. Let's try these with a centered parameterization first. I'll call the story intercepts `s[Sid]` and their standard deviation `tau`. I'll also show the code for the non-centered version, which actually samples more efficiently. Here's the code:

R code 13.29

```
m13H3 <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- z[id]*sigma + s[Sid] + bA*A + bC*C + BI*I ,
        BI <- bI + bIA*A + bIC*C ,
        z[id] ~ normal( 0 , 1 ),
        s[Sid] ~ normal( 0 , tau ),
        c(bA,bI,bC,bIA,bIC) ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 ),
        sigma ~ exponential(1),
```

```
        tau ~ exponential(1)
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )

m13H3z <- ulam(
    alist(
        R ~ dordlogit( phi , cutpoints ),
        phi <- z[id]*sigma + sz[Sid]*tau + bA*A + bC*C + BI*I ,
        BI <- bI + bIA*A + bIC*C ,
        z[id] ~ normal( 0 , 1 ),
        sz[Sid] ~ normal( 0 , 1 ),
        c(bA,bI,bC,bIA,bIC) ~ dnorm( 0 , 0.5 ),
        cutpoints ~ dnorm( 0 , 1.5 ),
        sigma ~ exponential(1),
        tau ~ exponential(1)
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

Let's look first at the marginal posterior distribution for `m13H3z`:

```
precis(m13H3z)
```

R code 13.30

```
       mean   sd  5.5% 94.5% n_eff Rhat4
bIC   -1.29 0.11 -1.47 -1.11  1721  1.00
bIA   -0.53 0.09 -0.67 -0.39  1708  1.00
bC    -1.08 0.10 -1.24 -0.92  1506  1.00
bI    -0.46 0.07 -0.57 -0.34  1697  1.00
bA    -0.89 0.07 -1.00 -0.78  1434  1.00
sigma  1.97 0.08  1.85  2.11   343  1.01
tau    0.55 0.15  0.38  0.80   396  1.01
```

The standard deviation among individuals, `sigma`, is similar to what it was in `m13H2z`. The posterior mean standard deviation among stories, `tau`, is about a third as large. So there's more variation among individuals than among stories.

Including varying intercepts on stories has always had a noticeable impact on estimates for the treatment variables. So again, variation across clusters in the presence of repeat measures plausibly biased the treatment estimate. But the qualitative story stays the same. This model improves precision, but it isn't telling a different causal story.

13H4. This problem is very similar to **13M1**, but it asks for interpretation of the posterior distribution. The same code is needed. Run the models from that solution again. Then let's inspect the posterior distributions of the coefficients. First the model with only predation:

```
precis( m1.2 )
```

R code 13.31

```
       mean   sd  5.5% 94.5% n_eff Rhat4
bp    -2.43 0.29 -2.90 -1.95   264  1.01
a_bar  2.53 0.23  2.17  2.91   286  1.00
sigma  0.82 0.14  0.61  1.07   694  1.00
```

Predation has a very strong negative effect on survival, which makes sense. Now consider the model that omits predation but includes size:

R code 13.32
```
precis( m1.3 , 2 , pars="s" )
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
s[1]  0.24 0.37 -0.35  0.82   237  1.01
s[2] -0.06 0.38 -0.64  0.55   239  1.01
```

Not such a clear effect of size. Now let's consider the models that include both. First the model without an interaction:

R code 13.33
```
precis( m1.4 , 2 , pars=c("bp","s") )
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
bp   -2.47 0.28 -2.92 -2.02   625  1.00
s[1]  0.32 0.37 -0.26  0.91   184  1.02
s[2] -0.11 0.37 -0.70  0.47   151  1.03
```

The agrees with the previous models. Predation has a clear and large impact. Size not so much. Now the interaction model:

R code 13.34
```
precis( m1.5 , 2 , pars=c("bp","s") )
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
bp[1] -1.86 0.38 -2.45 -1.24   906  1.01
bp[2] -2.77 0.38 -3.36 -2.17   899  1.00
s[1]   0.12 0.39 -0.52  0.73  1117  1.00
s[2]   0.16 0.40 -0.46  0.80  1295  1.00
```

The effect of predation does seem to vary by size. Let's compute the contrast:

R code 13.35
```
post <- extract.samples( m1.5 )
quantile( post$bp[,2] - post$bp[,1] , c(0.055,0.5,0.945) )
```

```
       5.5%         50%       94.5%
-1.68585188 -0.93005107 -0.08262948
```

So the contrast is reliably negative. Seems like size does matter, but only as it influences predation.

14. Chapter 14 Solutions

14E1. To add a varying slope on x, we have to add a dimension to the adaptive prior. This will mean that the β parameter becomes the average slope, and then we'll need a new standard deviation parameter for the slopes and a correlation parameter to estimate the correlation between intercepts and slopes. The way this was done in the chapter, it'll look like the following. Keep in mind that the precise notation varies among statisticians: sometimes (as below) vectors are in square brackets and matrices in parentheses, but others mix and match otherwise, or always use one or the other. As long as it is clear in context, it'll be fine. And if anyone gives you grief about your notation, just remind (or inform) them that notational conventions vary and ask them which part requires clarification.

$$
\begin{aligned}
y_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha_{\text{GROUP}[i]} + \beta_{\text{GROUP}[i]} x_i \\
\begin{bmatrix} \alpha_{\text{GROUP}} \\ \beta_{\text{GROUP}} \end{bmatrix} &\sim \text{MVNormal}\left(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S} \right) \\
\mathbf{S} &= \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{HalfCauchy}(0, 1) \\
\sigma_\alpha &\sim \text{Exponential}(0, 1) \\
\sigma_\beta &\sim \text{Exponential}(0, 1) \\
\mathbf{R} &\sim \text{LKJcorr}(2)
\end{aligned}
$$

If you used different priors for β and σ_β and $\mathbf{R}$, that's fine. Just be sure your choices make sense to you.

14E2. This is an open, imaginative problem. So instead of providing a precise answer, let me provide the structure that is being asked for. Then I'll follow it was some brief examples.

When intercepts and slopes are correlated, it means that clusters in the data that have high baselines also have stronger positive associations with a predictor variable. This is merely descriptive. So the mechanistic explanations that are consistent with it are highly diverse.

A very common example comes from educational testing. In this context, clusters are individual schools and observations are individual student test scores. Schools with high average test scores also tend to show greater differences (a larger slope) between poor and rich students within the school.

In growth data, large individuals also tend to grow faster. So for any interval of measurement, the repeat measures of size for individuals show a positive correlation between beginning height (intercept) and the slope across time.

In financial data, for very similar reasons, large investments tend to grow faster. So again, intercepts tend to be positively associated with slopes.

14E3. It possible for a varying effects model to actually have fewer effective parameters than a corresponding fixed effect model when there is very little variation among clusters. This will create strong shrinkage of the estimates, constraining the individual varying effect parameters. So even though the varying effects model must have more actual parameters in the posterior distribution, it can be

less flexible in fitting the data, because it adaptively regularizes. There's really nothing special about varying slopes in this respect. Varying intercepts work the same way.

Here's an example. To answer this problem, you did not need to provide a computational example. I'm just going to provide one for additional clarity.

Let's simulate some data in which there are clusters, but they aren't very different from one another. For the sake of comprehension, let's suppose the clusters are individuals and the observations are test scores. Each individual will have a unique "ability" that influences each test score. Our goal in estimation will be to recover these abilities.

R code 14.1

```
N_individuals <- 100
N_scores_per_individual <- 10

# simulate abilities
ability <- rnorm(N_individuals,0,0.1)

# simulate observed test scores
# sigma here large relative to sigma of ability
N <- N_scores_per_individual * N_individuals
id <- rep(1:N_individuals,each=N_scores_per_individual)
score <- round( rnorm(N,ability[id],1) , 2 )

# put observable variables in a data frame
d <- data.frame(
    id = id,
    score = score )
```

Now let's fit a fixed effect model. This is the "unpooled" model with an intercept for each individual, but with a fixed prior.

R code 14.2

```
m_nopool <- ulam(
    alist(
        score ~ dnorm(mu,sigma),
        mu <- a_id[id],
        a_id[id] ~ dnorm(0,10),
        sigma ~ dexp(1)
    ), data=d , chains=4 , log_lik=TRUE )
```

And this is the corresponding "partial pooling" model with an adaptive prior, the varying effects model. I'm going to use the non-centered parameterization here, because this is exactly a common situation in which it is needed to efficiently sample.

R code 14.3

```
m_partpool <- ulam(
    alist(
        score ~ dnorm(mu,sigma),
        mu <- a + z_id[id]*sigma_id,
        z_id[id] ~ dnorm(0,1),
        a ~ dnorm(0,10),
        sigma ~ dexp(1),
        sigma_id ~ dexp(1)
    ), data=d , chains=4 , log_lik=TRUE )
```

Now let's compare their effective parameters, as computed by WAIC. But first, let's count the actual parameters in each model. The model `m_nopool` has 100 intercepts (one for each individual) and

one standard deviation parameter, for 101 parameters total. The model `m_partpool` has 100 varying intercepts, one average intercept `a`, and two standard deviation parameters, for 103 parameters total. Now let's compare:

R code 14.4

```
compare( m_nopool , m_partpool )
```

```
             WAIC    SE dWAIC   dSE pWAIC weight
m_partpool 2838.8 44.57   0.0    NA   8.3      1
m_nopool   2943.1 44.65 104.4 18.91  96.2      0
```

The partial pooling model only has 8 effective parameters (in this simulation—your values will differ a bit due to simulation variance). The no-pool model has about 96 effective parameters. That's a big difference. You can uncover why `m_partpool` has so many fewer effective parameters than actual parameters by looking at the posterior for `sigma_id`:

R code 14.5

```
precis(m_partpool)
```

```
           mean   sd  5.5% 94.5% n_eff Rhat4
a         -0.02 0.03 -0.07  0.03  4002  1.00
sigma      1.00 0.02  0.96  1.03  2560  1.00
sigma_id   0.07 0.05  0.01  0.15   722  1.01
```

The posterior mean is just 0.07, so that induces a lot of shrinkage. Those 100 parameters are only as flexible as 6 or so parameters.

This whole ordeal emphasizes something very important about model "complexity": the number of dimensions in a model is not a relevant measure of complexity in terms of judging overfitting risk. This should warn us off using any intuitive measure of complexity when employing Occam's Razor style reasoning. Just because one model makes more assumptions than another (has more parameters or distributions), that does not always mean that it overfits more.

In statistics, "simple" just doesn't mean what in means in plain English.

14M1. Here's a compact form of the café robot simulation code, with the correlation `rho` changed to zero:

R code 14.6

```
# set up parameters of population
a <- 3.5            # average morning wait time
b <- (-1)           # average difference afternoon wait time
sigma_a <- 1        # std dev in intercepts
sigma_b <- 0.5      # std dev in slopes
rho <- (0)          # correlation between intercepts and slopes
Mu <- c( a , b )
cov_ab <- sigma_a*sigma_b*rho
Sigma <- matrix( c(sigma_a^2,cov_ab,cov_ab,sigma_b^2) , ncol=2 )

# simulate observations
N_cafes <- 20
library(MASS)
set.seed(6) # used to replicate example
vary_effects <- mvrnorm( N_cafes , Mu , Sigma )
a_cafe <- vary_effects[,1]
b_cafe <- vary_effects[,2]
N_visits <- 10
afternoon <- rep(0:1,N_visits*N_cafes/2)
```

```
cafe_id <- rep( 1:N_cafes , each=N_visits )
mu <- a_cafe[cafe_id] + b_cafe[cafe_id]*afternoon
sigma <- 0.5  # std dev within cafes
wait <- rnorm( N_visits*N_cafes , mu , sigma )

# package into  data frame
d <- data.frame( cafe=cafe_id , afternoon=afternoon , wait=wait )
```

And this code will sample from the posterior distribution, using the same model as in the chapter:

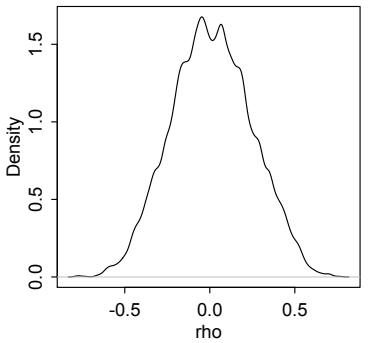
R code 14.7

```
m14.1 <- ulam(
    alist(
        wait ~ normal( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
        c(a_cafe,b_cafe)[cafe] ~ multi_normal( c(a,b) , Rho , sigma_cafe ),
        a ~ normal(5,2),
        b ~ normal(-1,0.5),
        sigma_cafe ~ exponential(1),
        sigma ~ exponential(1),
        Rho ~ lkj_corr(2)
    ) , data=d , chains=4 , cores=4 )
```

Now to visualize the posterior distribution of `rho`:

R code 14.8

```
post <- extract.samples(m14.1)
dens( post$Rho[,1,2] , xlab="rho" )
```



There's the zero correlation.

I set the random number seed in the code above to 6, so you can replicate the figure above. But for fun, go ahead and do the simulation and sampling again with the seed set to `set.seed(5)`. You should then see that the posterior distribution of `rho` is massed up above zero. This won't happen with most random simulations, but it's good to remember that any particular sample may be misleading. The posterior distribution is not magic: it cannot necessarily recover the data-generating mechanism. And with real data, these models will never accurately recover the data generating mechanism. At best, they describe it in a scientifically useful way.

14M2. The model presented in the problem uses completely independent priors for the intercepts, $\alpha_{\text{CAFÉ}}$, and slopes, $\beta_{\text{CAFÉ}}$. The consequence of this is that the model implicitly assumes no correlation

between the intercepts and slopes. It's missing the correlation parameter. Let's see what happens then when in truth the the intercepts and slopes are indeed correlated but the model ignores the correlation.

First, let's run the data simulation again, repeated here in compact form for convenience:

R code 14.9

```
# set up parameters of population
a <- 3.5            # average morning wait time
b <- (-1)           # average difference afternoon wait time
sigma_a <- 1        # std dev in intercepts
sigma_b <- 0.5      # std dev in slopes
rho <- (-0.7)          # correlation between intercepts and slopes
Mu <- c( a , b )
cov_ab <- sigma_a*sigma_b*rho
Sigma <- matrix( c(sigma_a^2,cov_ab,cov_ab,sigma_b^2) , ncol=2 )

# simulate observations
N_cafes <- 20
library(MASS)
set.seed(5) # used to replicate example
vary_effects <- mvrnorm( N_cafes , Mu , Sigma )
a_cafe <- vary_effects[,1]
b_cafe <- vary_effects[,2]
N_visits <- 10
afternoon <- rep(0:1,N_visits*N_cafes/2)
cafe_id <- rep( 1:N_cafes , each=N_visits )
mu <- a_cafe[cafe_id] + b_cafe[cafe_id]*afternoon
sigma <- 0.5  # std dev within cafes
wait <- rnorm( N_visits*N_cafes , mu , sigma )

# package into  data frame
d <- data.frame( cafe=cafe_id , afternoon=afternoon , wait=wait )
```

And here is the code to sample from the new model's posterior distribution:

R code 14.10

```
m14M2 <- ulam(
    alist(
        wait ~ dnorm( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
        a_cafe[cafe] ~ dnorm(a,sigma_alpha),
        b_cafe[cafe] ~ dnorm(b,sigma_beta),
        a ~ dnorm(0,10),
        b ~ dnorm(0,10),
        sigma ~ dexp(1),
        sigma_alpha ~ dexp(1),
        sigma_beta ~ dexp(1)
    ) , data=d , chains=4 , cores=4 )
```

And let's sample again from the model in the chapter, so we can directly compare estimates.

R code 14.11

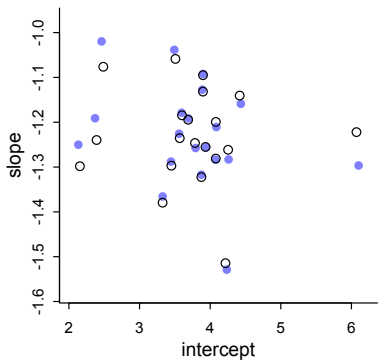
```
m14.1 <- ulam(
    alist(
        wait ~ normal( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
        c(a_cafe,b_cafe)[cafe] ~ multi_normal( c(a,b) , Rho , sigma_cafe ),
        a ~ normal(5,2),
```

```
        b ~ normal(-1,0.5),
        sigma_cafe ~ exponential(1),
        sigma ~ exponential(1),
        Rho ~ lkj_corr(2)
    ) , data=d , chains=4 , cores=4 )
```

We're primarily interested in differences in the varying effects estimates. So let's plot them together and see if there are any differences (in posterior means):

R code 14.12

```
post1 <- extract.samples(m14.1)
a1 <- apply( post1$a_cafe , 2 , mean )
b1 <- apply( post1$b_cafe , 2 , mean )

post2 <- extract.samples(m14M2)
a2 <- apply( post2$a_cafe , 2 , mean )
b2 <- apply( post2$b_cafe , 2 , mean )

plot( a1 , b1 , xlab="intercept" , ylab="slope" ,
    pch=16 , col=rangi2 , ylim=c( min(b1)-0.05 , max(b1)+0.05 ) ,
    xlim=c( min(a1)-0.1 , max(a1)+0.1 ) )
points( a2 , b2 , pch=1 )
```



The blue filled points are the posterior means from `m14.1`, the original mode in the chapter that includes the correlation between intercepts and slopes. The open points are the posterior means from `m14M2`, the model that assumes no correlation between intercepts and slopes.

First, notice that there is a lot of agreement here. Assuming no correlation didn't completely change inference. And if you look at the posterior distributions of the average effects, α and β and the standard deviation parameters, they are essentially identical.

Second, notice that there are differences in the above plot, and they tend to occur at extreme intercept and slope values. In the middle of the cloud of points, the blue and open circles almost perfectly overlap. At the edges, there is more separation. This is especially true for extreme intercepts. What's going on here?

Model `m14.1` estimated the correlation between intercepts and slopes and it used that correlated at the same time to adjust the intercepts and slopes. You learned this much in the chapter. The correlation in these data is negative, so large intercepts and associated (on average) with small slopes. So in the plot above, right of the center the blue points are displaced *below* the open points, because the model used the fact that those intercepts were *larger* than average to push the slopes to be *smaller*.

Likewise, to the left of the center the intercepts are *smaller* than average. So the model pushes the slopes up to become *larger*.

The model that includes the correlation will be, on average, more accurate. It exploits additional information about the population in order to shrink in both dimensions. But a model that assumes zero correlation in the prior does not prevent estimates from having a non-zero posterior correlation.

14M3. Okay, so there isn't a UCBadmit example in this chapter anymore (there was in the 1st edition). But we can build a varying slopes model of the UCBadmit example from chapter 11. The easiest way to do this is to use an indicator for gender (male) and then we can use the same varying slopes structure as in the chapter.

R code 14.13

```
library(rethinking)
data(UCBadmit)
d <- UCBadmit
dat_list <- list(
    admit = d$admit,
    applications = d$applications,
    male = ifelse( d$applicant.gender=="male" , 1 , 0 ),
    dept_id = rep(1:6,each=2)
)

m14M3 <- ulam(
    alist(
        admit ~ dbinom( applications , p ) ,
        logit(p) <- delta[dept_id] + bm[dept_id]*male,
        a ~ dnorm( 0 , 1.5 ) ,
        c(delta,bm)[dept_id] ~ multi_normal( c(a,bm_bar) , Rho , sigma_dept ),
        bm_bar ~ dnorm(0,1),
        sigma_dept ~ dexp(1),
        Rho ~ dlkjcorr(2)
    ) , data=dat_list , chains=4 , cores=4 )
precis( m14M3 , 3 )
```

```
                mean   sd  5.5% 94.5% n_eff Rhat4
a              -0.45 0.59 -1.39  0.48  1314     1
bm[1]          -0.78 0.27 -1.20 -0.37   978     1
bm[2]          -0.20 0.31 -0.71  0.28  1084     1
bm[3]           0.08 0.14 -0.14  0.30  1754     1
bm[4]          -0.09 0.14 -0.32  0.13  1462     1
bm[5]           0.11 0.19 -0.18  0.42  1723     1
bm[6]          -0.12 0.27 -0.55  0.30  1352     1
delta[1]        1.29 0.25  0.91  1.70  1009     1
delta[2]        0.73 0.31  0.24  1.23  1076     1
delta[3]       -0.65 0.09 -0.78 -0.51  1808     1
delta[4]       -0.62 0.10 -0.78 -0.45  1475     1
delta[5]       -1.13 0.12 -1.32 -0.95  1871     1
delta[6]       -2.60 0.20 -2.93 -2.28  1580     1
bm_bar         -0.16 0.23 -0.51  0.17  1209     1
sigma_dept[1]  1.50 0.48  0.92  2.37  1785     1
sigma_dept[2]  0.46 0.23  0.19  0.85   824     1
Rho[1,1]        1.00 0.00  1.00  1.00   NaN   NaN
Rho[1,2]       -0.30 0.35 -0.79  0.32  1226     1
Rho[2,1]       -0.30 0.35 -0.79  0.32  1226     1
```

```
Rho[2,2]       1.00 0.00  1.00  1.00  1977     1
```

The posterior distribution for the correlation between intercept (female admission rate) and slopes (male admission rate) to be negative. It's not very strong, but it makes sense given the varying effects: Department 1 has the highest rate of admitting women and the smallest slope (change in admissions of men), while Department 6 has the opposite pattern (male/female admission rates about the same, but very low intercept).

Now let's compare the non-centered version of the model.

R code 14.14

```
m14M3nc <- ulam(
    alist(
        admit ~ dbinom( applications , p ) ,
        logit(p) <- a + v[dept_id,1] + (bm_bar + v[dept_id,2])*male,
        transpars> matrix[dept_id,2]:v <-
            compose_noncentered( sigma_dept , L_Rho , z ),
        matrix[2,dept_id]:z ~ dnorm( 0 , 1 ),
        a ~ dnorm( 0 , 1.5 ) ,
        bm_bar ~ dnorm(0,1),
        vector[2]:sigma_dept ~ dexp(1),
        cholesky_factor_corr[2]:L_Rho ~ lkj_corr_cholesky( 2 )
    ) , data=dat_list , chains=4 , cores=4 )
```

First let's compare running times. On my machine, each chain in the centered model `m14M3` took about 0.8 seconds. In the new model `m14M3nc`, each chain took about 1 second to finish sampling. So the centered version of the model sampled faster.

Now let's compare effective sample sizes. I'll just pull out the `n_eff` values from the `precis` output and bind them together in a matrix. This is a little tricky, because the non-centered model returns more "parameters" in the posterior distribution, on account of also returning the internal z-scores and Cholesky factor that was used in sampling. So we need to exclude those. And we need the same parameters in the same order from the first model. So I'll build a parameter name list first and pass it to both:

R code 14.15

```
precis( m14M3nc , 3 )
```

```
                mean   sd  5.5% 94.5% n_eff Rhat4
z[1,1]          1.25 0.52  0.43  2.11   715  1.00
z[1,2]          0.85 0.48  0.09  1.63   599  1.00
z[1,3]         -0.16 0.37 -0.76  0.43   540  1.00
z[1,4]         -0.14 0.37 -0.73  0.45   559  1.00
z[1,5]         -0.52 0.39 -1.15  0.10   558  1.00
z[1,6]         -1.60 0.56 -2.53 -0.74   682  1.01
z[2,1]         -1.21 0.76 -2.45 -0.03  1335  1.00
z[2,2]          0.21 0.80 -1.03  1.55  1475  1.00
z[2,3]          0.64 0.62 -0.32  1.68  1148  1.00
z[2,4]          0.15 0.60 -0.80  1.11  1198  1.01
z[2,5]          0.60 0.66 -0.41  1.71  1380  1.00
z[2,6]         -0.45 0.85 -1.84  0.93  1295  1.00
a              -0.42 0.56 -1.26  0.49   552  1.00
bm_bar         -0.16 0.20 -0.49  0.14   911  1.00
sigma_dept[1]  1.46 0.44  0.92  2.29   775  1.00
sigma_dept[2]  0.44 0.21  0.17  0.80   852  1.01
L_Rho[1,1]      1.00 0.00  1.00  1.00   NaN   NaN
L_Rho[1,2]      0.00 0.00  0.00  0.00   NaN   NaN
L_Rho[2,1]     -0.31 0.34 -0.79  0.31  1274  1.00
```

```
L_Rho[2,2]     0.88 0.13  0.61  1.00  1413  1.00
v[1,1]         1.70 0.60  0.75  2.63   657  1.00
v[1,2]        -0.60 0.30 -1.12 -0.16  1179  1.00
v[2,1]         1.15 0.63  0.16  2.12   602  1.00
v[2,2]        -0.04 0.32 -0.54  0.47  1578  1.00
v[3,1]        -0.22 0.56 -1.13  0.62   564  1.00
v[3,2]         0.23 0.23 -0.13  0.62  1030  1.00
v[4,1]        -0.20 0.57 -1.10  0.65   568  1.00
v[4,2]         0.07 0.23 -0.28  0.45  1109  1.01
v[5,1]        -0.71 0.57 -1.61  0.15   545  1.00
v[5,2]         0.27 0.25 -0.10  0.69  1300  1.00
v[6,1]        -2.18 0.58 -3.11 -1.29   628  1.00
v[6,2]         0.04 0.30 -0.44  0.51  1582  1.00
```

You'll have to compare carefully to the previous precis output, because the parameters aren't named the same thing, but the centered version actually has higher effective sample size in almost every case. What is going on here?

The chapter says that *sometimes* non-centered parameterizations are better. Other times, the centered form is better. This is such a time. How can you guess which situation you are in, before trying both parameterizations? Typically you can't. But the non-centered parameterization tends to be better when the variation across clusters is either very small—close to zero—or poorly identified by the data.

14M4. The first thing that is required is to sampling from the Gaussian process model, as well as the simpler Poisson models from Chapter 11. Here's the code to sampling from the Gaussian process model. Fit model `m14.8` from this chapter (you will need to add `log_lik=TRUE` to the code in the chapter) and model `m11.11` from Chapter 11. Then compare:

R code 14.16

```
compare( m11.11 , m14.8 )
```

```
       WAIC    SE dWAIC   dSE pWAIC weight
m14.8  68.0  2.43   0.0    NA   4.3      1
m11.11 80.1 11.26  12.1 11.16   4.9      0
```

The Gaussian process has fewer effective parameters, even though it has more actual parameters. The GP model has 10 intercepts, one for each society, and 5 other parameters for a total of 15. The ordinary model has 5 total parameters, which WAIC counts pretty accurately. GP models typically have many fewer effective parameters than actual parameters, because they aggressively regularize.

14M5. Using the code in the chapter, we can prepare all the data structures and then use the same models, but with the roles of B and G reversed.

R code 14.17

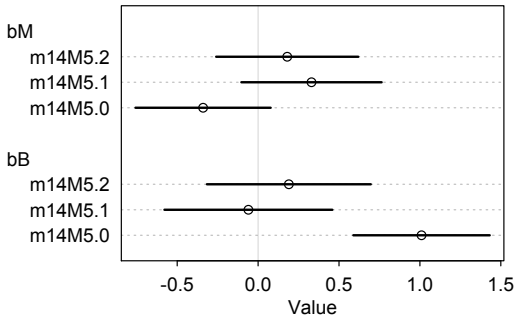
```
library(rethinking)
data(Primates301)
data(Primates301_nex)
d <- Primates301
d$name <- as.character(d$name)
dstan <- d[ complete.cases( d$group_size , d$body , d$brain ) , ]
spp_obs <- dstan$name

dat_list <- list(
    N_spp = nrow(dstan),
```

```
    M = standardize(log(dstan$body)),
    B = standardize(log(dstan$brain)),
    G = standardize(log(dstan$group_size)),
    Imat = diag(nrow(dstan)) )

library(ape)
tree_trimmed <- keep.tip( Primates301_nex, spp_obs )
Rbm <- corBrownian( phy=tree_trimmed )
V <- vcv(Rbm)
Dmat <- cophenetic( tree_trimmed )

dat_list$V <- V[ spp_obs , spp_obs ]
dat_list$R <- dat_list$V / max(V)

# ordinary regression
m14M5.0 <- ulam(
    alist(
        G ~ multi_normal( mu , SIGMA ),
        mu <- a + bM*M + bB*B,
        matrix[N_spp,N_spp]: SIGMA <- Imat * sigma_sq,
        a ~ normal( 0 , 1 ),
        c(bM,bB) ~ normal( 0 , 0.5 ),
        sigma_sq ~ exponential( 1 )
    ), data=dat_list , chains=4 , cores=4 )

# Brownian motion model
m14M5.1 <- ulam(
    alist(
        G ~ multi_normal( mu , SIGMA ),
        mu <- a + bM*M + bB*B,
        matrix[N_spp,N_spp]: SIGMA <- R * sigma_sq,
        a ~ normal( 0 , 1 ),
        c(bM,bB) ~ normal( 0 , 0.5 ),
        sigma_sq ~ exponential( 1 )
    ), data=dat_list , chains=4 , cores=4 )

# OU model
dat_list$Dmat <- Dmat[ spp_obs , spp_obs ] / max(Dmat)
m14M5.2 <- ulam(
    alist(
        G ~ multi_normal( mu , SIGMA ),
        mu <- a + bM*M + bB*B,
        matrix[N_spp,N_spp]: SIGMA <- cov_GPL1( Dmat , etasq , rhosq , 0.01 ),
        a ~ normal(0,1),
        c(bM,bB) ~ normal(0,0.5),
        etasq ~ half_normal(1,0.25),
        rhosq ~ half_normal(3,0.25)
    ), data=dat_list , chains=4 , cores=4 )
```

Now let's compare the estimates:

R code 14.18

```
plot( coeftab(m14M5.0,m14M5.1,m14M5.2) , pars=c("bM","bB") )
```

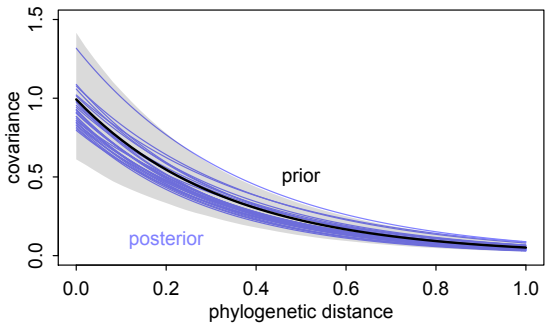


The model ignoring phylogeny (0) finds a strong association between brain size and group size, while conditioning on body mass. The two models that include phylogeny (1 and 2) are less enthusiastic about brain size. So it seems that including phylogeny, as least in this way, produces less evidence of a causal association between brain size and group size.

Something we can do in addition is inspect the covariance function from the OU/GP model, `m14M5.2`. Just like in the chapter:

R code 14.19

```
post <- extract.samples(m14M5.2)
plot( NULL , xlim=c(0,max(dat_list$Dmat)) , ylim=c(0,1.5) ,
    xlab="phylogenetic distance" , ylab="covariance" )

# posterior
for ( i in 1:30 )
    curve( post$etasq[i]*exp(-post$rhosq[i]*x) , add=TRUE , col=rangi2 )

# prior mean and 89% interval
eta <- abs(rnorm(1e3,1,0.25))
rho <- abs(rnorm(1e3,3,0.25))
d_seq <- seq(from=0,to=1,length.out=50)
K <- sapply( d_seq , function(x) eta*exp(-rho*x) )
lines( d_seq , colMeans(K) , lwd=2 )
shade( apply(K,2,PI) , d_seq )
text( 0.5 , 0.5 , "prior" )
text( 0.2 , 0.1 , "posterior" , col=rangi2 )
```

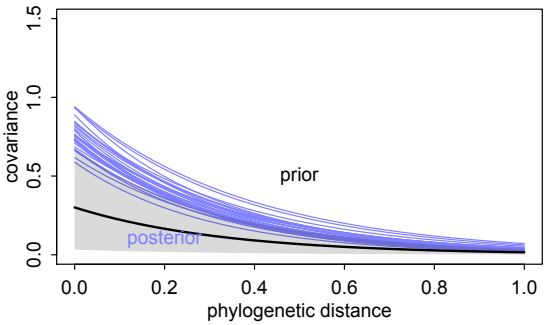


Hm, looks like we got the prior back. This is always suspicious. Let's try changing the prior and looking again, to make sure:

R code 14.20

```
# OU model with different GP prior
m14M5.3 <- ulam(
    alist(
        G ~ multi_normal( mu , SIGMA ),
        mu <- a + bM*M + bB*B,
        matrix[N_spp,N_spp]: SIGMA <- cov_GPL1( Dmat , etasq , rhosq , 0.01 ),
        a ~ normal(0,1),
        c(bM,bB) ~ normal(0,0.5),
        etasq ~ half_normal(0.25,0.25),
        rhosq ~ half_normal(3,0.25)
    ), data=dat_list , chains=4 , cores=4 )
```

Extract the samples and re-run the plotting code:



Okay, the posterior is different from the prior here, so it seems there is some information about phylogenetic signature in the data. And the model uses it to infer that the association between brain size and group size is confounded by phylogeny.

14H1. Loading the data and running the model:

R code 14.21

```
library(rethinking)
data(bangladesh)
d <- bangladesh

dat_list <- list(
    C = d$use.contraception,
    did = as.integer( as.factor(d$district) ),
    urban = d$urban )

m14H1.1 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did] + b[did]*urban,
        c(a,b)[did] ~ multi_normal( c(abar,bbar) , Rho , Sigma ),
        abar ~ normal(0,1),
        bbar ~ normal(0,0.5),
        Rho ~ lkj_corr(2),
```

```
        Sigma ~ exponential(1)
    ) , data=dat_list , chains=4 , cores=4 , iter=4000 )
```

This is a conventional varying slopes model, with a centered parameterization. No surprises. If you peek at the posterior distributions for the average effects, you'll see that the average slope is positive:

R code 14.22

```
precis(m14H1.1)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
abar -0.68 0.10 -0.84 -0.53  5722     1
bbar  0.64 0.16  0.38  0.90  3861     1
```

This implies that urban areas use contraception more. Not surprising. Now consider the distribution of varying effects:

R code 14.23

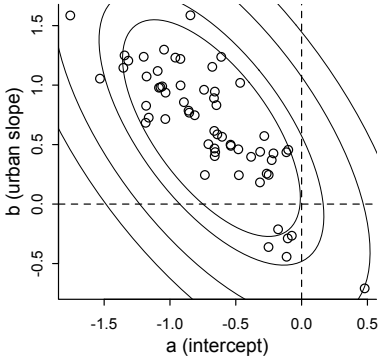
```
precis( m14H1.1 , depth=3 , pars=c("Rho","Sigma") )
```

```
          mean   sd  5.5% 94.5% n_eff Rhat4
Rho[1,1]  1.00 0.00  1.00  1.00   NaN   NaN
Rho[1,2] -0.65 0.17 -0.86 -0.35  1609     1
Rho[2,1] -0.65 0.17 -0.86 -0.35  1609     1
Rho[2,2]  1.00 0.00  1.00  1.00  7482     1
Sigma[1]  0.58 0.10  0.43  0.74  1755     1
Sigma[2]  0.79 0.20  0.48  1.11   920     1
```

The correlation between the intercepts and slopes is quite negative. Let's plot the individual effects to appreciate this:

R code 14.24

```
post <- extract.samples(m14H1.1)

a <- apply( post$a , 2 , mean )
b <- apply( post$b , 2 , mean )

plot( a, b , xlab="a (intercept)" , ylab="b (urban slope)" )
abline( h=0 , lty=2 )
abline( v=0 , lty=2 )

library(ellipse)
R <- apply( post$Rho , 2:3 , mean )
s <- apply( post$Sigma , 2 , mean )
S <- diag(s) %*% R %*% diag(s)
ll <- c( 0.5 , 0.67 , 0.89 , 0.97 )
for ( l in ll ) {
    el <- ellipse( S , centre=c( mean(post$abar) , mean(post$bbar) ) , level=l )
    lines( el , col="black" , lwd=0.5 )
}
```

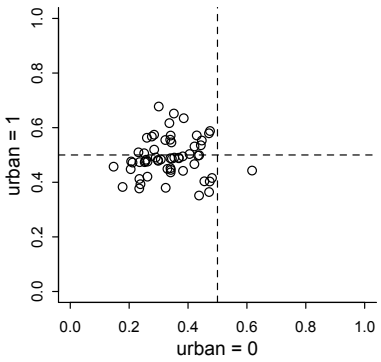


There's the negative correlation—districts with higher use outside urban areas (a values) have smaller slopes. Since the slope is the difference between urban and non-urban areas, you can see this as saying that districts with high use in rural areas have urban areas that aren't as different.

On the outcome scale, what this ends up meaning is that urban places are much the same in all districts, but rural areas vary a lot. Plotting now in the outcome scale:

R code 14.25

```
u0 <- inv_logit( a )
u1 <- inv_logit( a + b )

plot( u0 , u1 , xlim=c(0,1) , ylim=c(0,1) , xlab="urban = 0" , ylab="urban = 1" )
abline( h=0.5 , lty=2 )
abline( v=0.5 , lty=2 )
```



This plot is on the probability scale. The horizontal axis is probability of contraceptive use in rural area of a district. The vertical is the probability in urban area of same district. The urban areas all straddle 0.5. Most the of the rural areas are below 0.5. The negative correlation between the intercepts and slopes is necessary to encode this pattern.

In fact, if we fit the model so it instead has two intercepts, one for rural and one for urban, there is no strong correlation between those intercepts. Here's such a model:

R code 14.26

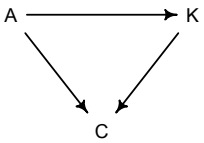
```
# version with matrix instead of slopes
dat_list$uid <- dat_list$urban + 1L

m14H1.2 <- ulam(
    alist(
```

```
        C ~ bernoulli( p ),
        logit(p) <- a[did,uid],
        vector[2]:a[did] ~ multi_normal( c(abar,bbar) , Rho , Sigma ),
        abar ~ normal(0,1),
        bbar ~ normal(0,1),
        Rho ~ lkj_corr(2),
        Sigma ~ exponential(1)
    ) , data=dat_list , chains=4 , cores=4 , iter=4000 )
precis( m14H1.2 , depth=3 , pars="Rho" )
```

```
          mean   sd  5.5% 94.5% n_eff Rhat4
Rho[1,1]  1.00 0.00  1.00  1.00   NaN   NaN
Rho[1,2] -0.09 0.28 -0.52  0.38  1664     1
Rho[2,1] -0.09 0.28 -0.52  0.38  1664     1
Rho[2,2]  1.00 0.00  1.00  1.00  8049     1
```

Correlation all gone.

14H2. Here's my DAG:



A is age, K is number of children, and C is contraception use. To study this DAG, we should estimate both the total causal influence of A and then condition also on K and see if the direct influence of A is smaller. Here's the model for the total influence of A:

R code 14.27

```
dat_list$children <- standardize( d$living.children )
dat_list$age <- standardize( d$age.centered )

m14H2.1 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did] + b[did]*urban + bA*age,
        c(a,b)[did] ~ multi_normal( c(abar,bbar) , Rho , Sigma ),
        abar ~ normal(0,1),
        c(bbar,bA) ~ normal(0,0.5),
        Rho ~ lkj_corr(2),
        Sigma ~ exponential(1)
    ) , data=dat_list , chains=4 , cores=4 , iter=4000 )

precis(m14H2.1)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
abar -0.69 0.10 -0.85 -0.53  6121     1
bA    0.08 0.05  0.00  0.16 14876     1
bbar  0.64 0.16  0.38  0.90  4300     1
```

In this model, the total causal effect of age is positive and very small. Older individuals use slightly more contraception.

And now the model with both K and A:

R code 14.28

```
m14H2.2 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did] + b[did]*urban + bK*children + bA*age,
        c(a,b)[did] ~ multi_normal( c(abar,bbar) , Rho , Sigma ),
        abar ~ normal(0,1),
        c(bbar,bK,bA) ~ normal(0,0.5),
        Rho ~ lkj_corr(2),
        Sigma ~ exponential(1)
    ) , data=dat_list , chains=4 , cores=4 , iter=4000 )

precis(m14H2.2)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
abar -0.72 0.10 -0.88 -0.56  6341     1
bA   -0.26 0.07 -0.38 -0.15  9281     1
bK    0.51 0.07  0.40  0.63  8945     1
bbar  0.69 0.16  0.43  0.95  4050     1
```

In this model, the direct effect of age is negative, and much farther from zero than before. The effect of number of children is strong and positive. These results are consistent with the DAG, because they imply that the reason the total effect of age, from `m14H2.1`, is positive is that older individuals also have more kids. Having more kids increases contraception. Being older, controlling for kids, actually makes contraception less likely.

14H3. To build this model, you need the ordered categorical predictor machinery from the book example. The maximum observed number of kids in the sample is 4. So that means we need three parameters, for three transitions in number of kids. We'll set up the alpha prior that way:

R code 14.29

```
dat_list$K <- d$living.children
dat_list$alpha <- rep(2,3)

m14H3.1 <- ulam(
    alist(
        C ~ bernoulli( p ),
        logit(p) <- a[did] + b[did]*urban + bK*sum( delta_shell[1:K] ) + bA*age,
        c(a,b)[did] ~ multi_normal( c(abar,bbar) , Rho , Sigma ),
        abar ~ normal(0,1),
        c(bbar,bK,bA) ~ normal(0,0.5),
        Rho ~ lkj_corr(2),
        Sigma ~ exponential(1),
        vector[4]: delta_shell <<- append_row( 0 , delta ),
        simplex[3]: delta ~ dirichlet( alpha )
    ) , data=dat_list , chains=4 , cores=4 , iter=4000 )

precis(m14H3.1)
```

```
      mean   sd  5.5% 94.5% n_eff Rhat4
abar -1.55 0.15 -1.78 -1.32  2260     1
bA   -0.22 0.06 -0.33 -0.12  4675     1
bK    1.26 0.15  1.02  1.50  2234     1
bbar  0.69 0.16  0.44  0.95  3706     1
```

So the general effects are the same—age reduces use and kids increase it. Let's look at the individual kid parameters now:

R code 14.30

```
precis( m14H3.1 , 3 , pars="delta" )
```

```
         mean   sd 5.5% 94.5% n_eff Rhat4
delta[1] 0.74 0.08 0.60  0.87 10494     1
delta[2] 0.16 0.08 0.05  0.31 10688     1
delta[3] 0.09 0.05 0.02  0.19 11863     1
```

`delta[1]` is the transition from 1 to 2 kids. It is much larger than the other two parameters. So most of the influence of kids on contraception comes from having a second child.

14H4. Here's the code to fit the model with varying intercepts and slopes for age, clustered by Subject:

R code 14.31

```
library(rethinking)
data(Oxboys)
d <- Oxboys
d$A <- standardize( d$age )
d$id <- coerce_index( d$Subject )

m14H4.1 <- ulam(
    alist(
        height ~ dnorm( mu , sigma ),
        mu <- a_bar + a[id] + (b_bar + b[id])*A,
        a_bar ~ dnorm(150,10),
        b_bar ~ dnorm(0,10),
        c(a,b)[id] ~ multi_normal( 0 , Rho_id , sigma_id ),
        sigma_id ~ dexp(1),
        Rho_id ~ dlkjcorr(2),
        sigma ~ dexp(1)
    ), data=d , chains=4 , cores=4 , iter=4000 )
```

Let's look at the estimates (omitting the varying effects and correlation for the moment):

R code 14.32

```
precis( m14H4.1 , depth=2 , pars=c("a_bar","b_bar","sigma_id") )
```

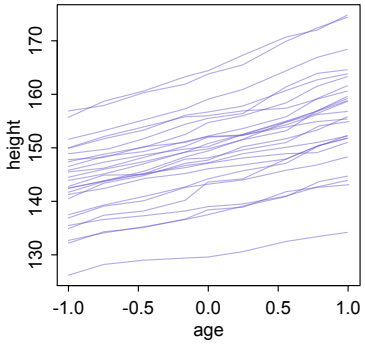
```
              mean   sd   5.5%  94.5% n_eff Rhat4
a_bar       149.48 1.41 147.21 151.74   258  1.01
b_bar         4.22 0.21   3.89   4.56   424  1.01
sigma_id[1]   7.32 0.89   6.05   8.84  5338  1.00
sigma_id[2]   1.06 0.15   0.85   1.33  4548  1.00
```

Let's interpret the intercept `a_bar` first. Since the predictor `age` is standardized, the intercept is the average height at the average age. Then the average slope `b_bar` is average change in height for unit change in standard age. So over the whole sample, which is about 2 units of standard age, the average boy grew about $2 \times 4.22 = 8.5$cm. That's not so easy to understand. Plotting the raw data might help:

R code 14.33

```
plot( height ~ age , type="n" , data=d )
for ( i in 1:26 ) {
    h <- d$height[ d$Subject==i ]
```

```
    a <- d$age[ d$Subject==i ]
    lines( a , h , col=col.alpha("slateblue",0.5) , lwd=2 )
}
```



You can see here perhaps that while some boys grew more and others grew less, the average growth was a little more than 10cm.

Let's consider the variation in intercepts and slopes now, as the problem asks. There is substantial variation among both intercepts and slopes. But which contributes more to variation in the data? You can't really say without knowing the predictor values that multiply the slopes. If for example the `age` values have a very large range in the data, then a smaller standard deviation for slopes could manifest as more variation in the data attributable to variation in slopes. But in this case, you can probably appreciate from the plot just above that the intercepts are contributing more to differences among boys in the total data.

14H5. Now let's consider the correlation between intercepts and slopes, using the model fit in the previous problem.

R code 14.34

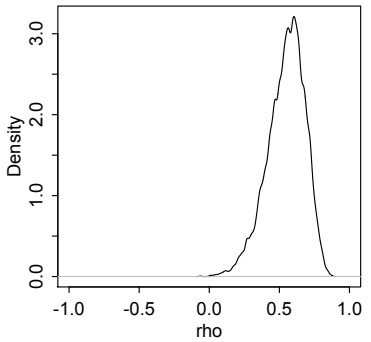
```
precis( m14H4.1 , depth=3 , pars="Rho_id" )
```

```
             mean   sd 5.5% 94.5% n_eff Rhat4
Rho_id[1,1] 1.00 0.00  1.0  1.00   NaN   NaN
Rho_id[1,2] 0.53 0.13  0.3  0.72  4699     1
Rho_id[2,1] 0.53 0.13  0.3  0.72  4699     1
Rho_id[2,2] 1.00 0.00  1.0  1.00  7596     1
```

So the posterior distribution of the correlation between intercepts and slopes has a mean of 0.55 and an 89% interval from 0.3 to 0.7. This is what it looks like:

R code 14.35

```
rho <- extract.samples(m14H4.1)$Rho_id[,1,2]
dens( rho , xlab="rho" , xlim=c(-1,1) )
```



This positive correlation suggests that larger intercepts are associated with larger slopes. In more meaningful terms, this means that boys who are bigger also grow faster. You might be able to see this in the data plot from the previously problem. The boys who were tallest at the start also grew the fastest. The boys who were shortest at the start also grew the slowest. As a result, the difference between the tallest and shortest boys grew over time.

To appreciate the value of this inference, consider a new sample of boys that is purely cross-sectional. No time series has yet been observed. But on the basis of this correlation, you might predict that the tallest boys in the new sample would grow the fastest. This would let you make better predictions, assuming of course that this result generalizes to another sample.

14H6. To simulate, you just run the model forwards. So we extract the estimates of the parameters and use them to define a distribution. Then we sample random values from that distribution. I'll walk through each step.

First, it will help to write down the varying intercepts and slopes model in math form, for clarity:

$$\begin{aligned} H_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \alpha_{\text{SUBJECT}[i]} + (\beta + \beta_{\text{SUBJECT}[i]})A_i, \\ \begin{pmatrix} \alpha_{\text{SUBJECT}} \\ \beta_{\text{SUBJECT}} \end{pmatrix} &\sim \text{Normal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_\beta \\ \rho\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix} \right). \end{aligned}$$

So H_i is the height for observation i, A_i is the corresponding age for that case. σ defines the standard deviation of heights for a particular Subject and age combination (this is mostly going to be measurement error, but could also be changes across ages that don't correspond to the linear trend that is being modeled). α is the average height at age zero across all boys, and β is the average slope on age, across all boys. The varying effects α_{SUBJECT} and β_{SUBJECT} are the individual Subject deviations from those averages, and finally we assume those varying effects are sampled from a bivariate normal distribution (last line of the model above), with a variance-covariance matrix:

$$\mathbf{S} = \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_\beta \\ \rho\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix}$$

I've labeled this thing $\mathbf{S}$ just so I can talk about it later, and you'll know exactly what I'm referring to. Three parameters define the bivariate distribution: the standard deviation of intercepts (σ_α), the standard deviation of slopes (σ_β), and the correlation between intercepts and slopes (ρ).

So to simulate boys from the fit model, we need estimates for α, β, σ, σ_α, σ_β, and ρ. We don't need to pay attention to the α_{SUBJECT} and β_{SUBJECT} estimates, because we are simulating new boys, not plotting predictions for the boys in the sample. So make some symbols to hold the posterior means (ignoring uncertainty for the moment):

R code 14.36
```
post <- extract.samples(m14H4.1)
rho <- mean( post$Rho_id[,1,2] )
sb <- mean( post$sigma_id[,2] )
sa <- mean( post$sigma_id[,1] )
sigma <- mean( post$sigma )
a <- mean( post$a_bar )
b <- mean( post$b_bar )
```

Those are ρ, σ_β, σ_α, σ, α, and β, respectively. So now we can define the variance-covariance matrix S:

R code 14.37
```
S <- matrix( c( sa^2 , sa*sb*rho , sa*sb*rho , sb^2 ) , nrow=2 )
round( S , 2 )
```

```
      [,1] [,2]
[1,] 53.58 4.10
[2,]  4.10 1.13
```

And now to sample varying intercepts and slopes from the bivariate distribution of them:

R code 14.38
```
library(MASS)
ve <- mvrnorm( 10 , c(0,0) , Sigma=S )
ve
```

```
          [,1]        [,2]
 [1,] 2.434950  0.57284532
 [2,] 4.782632  0.96182774
 [3,] 3.253491  1.04498283
 [4,] 7.560079 -1.90747544
 [5,] 1.435269 -1.39754684
 [6,] 2.920339 -0.81308012
 [7,] 2.823925  0.01066704
 [8,] 6.887322  1.09897341
 [9,] 4.477458  0.73169761
[10,] 5.080821  1.14740189
```

Those are 10 random boys, with their own varying intercepts (first column) and slopes (second column). Remember, these values will be added to the mean α and β values to make predicted heights.

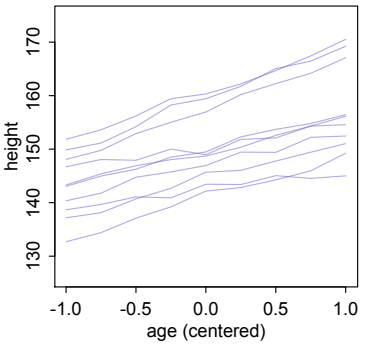
Now the last simulation step is to top it all off with a simulated trend for each boy. So now we make use of σ (`sigma`) and produce random normal heights across ages. I'll just simulate each boy's trend as I plot them. First, define the sequence of ages to simulate over, and then make an empty plot:

R code 14.39
```
age.seq <- seq(from=-1,to=1,length.out=9)
plot( 0 , 0 , type="n" , xlim=range(d$age) , ylim=range(d$height) ,
    xlab="age (centered)" , ylab="height" )
```

Now to loop over the rows in `ve` and simulate 9 heights for each pair of parameters:

R code 14.40
```
for ( i in 1:nrow(ve) ) {
    h <- rnorm( 9 ,
        mean=a + ve[i,1] + (b + ve[i,2])*age.seq ,
        sd=sigma )
    lines( age.seq , h , col=col.alpha("slateblue",0.5) )
}
```

Finally, here's what it looks like:



As always, your plot will look a little different, on account of simulation variance. Run the simulation a few times to get a sense for this.

15. Chapter 15 Solutions

15E1. To add measurement error on a predictor variable, just add a distributional assumption for the observed values. In this case, we want to allow each observed log-population, $\log P_i$, to be a draw from some distribution with an unknown true value plus error. In the chapter, the example used a Gaussian distribution. So I'll use that again here. Specifically, assume that each observed $\log P_i$ is defined by:

$$\log P_i \sim \text{Normal}(\phi_i, \sigma_P)$$

where each ϕ_i is an unobserved true log-population for each society i and σ_P is the standard error of measurement of log-population size.

To complete the model, we just add the above into the original model and replace the $\log P_i$ in the linear model with the unobserved ϕ_i values:

$$\begin{aligned} T_i &\sim \text{Poisson}(\mu_i) \\ \log \mu_i &= \alpha + \beta\phi_i \\ \log P_i &\sim \text{Normal}(\phi_i, \sigma_P) \\ \alpha &\sim \text{Normal}(0, 1) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma_P &\sim \text{Exponential}(1) \end{aligned}$$

I added a default prior for σ_P above. In a real analysis, you'd have information about the error that would help you either set an informative prior or (as in the chapter) use precise data for the standard error.

15E2. Imputation is almost the same trick as measurement error. When there is no measurement at all for a particular case in the data, the other cases which are measured provide information to define an adaptive prior for the variable. This prior then informs the missing values. This is exactly what was done in the chapter. The details depend upon the causal model, as with measurement error. But the simplest case is very simple. Here's what it might look like for the Oceanic societies model:

$$\begin{aligned} T_i &\sim \text{Poisson}(\mu_i) \\ \log \mu_i &= \alpha + \beta\phi_i \\ \phi_i &\sim \text{Normal}(\bar{\phi}, \sigma_P) \\ \alpha &\sim \text{Normal}(0, 1) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\phi} &\sim \text{Normal}(0, 1) \\ \sigma_P &\sim \text{Exponential}(1) \end{aligned}$$

Now each ϕ_i value is either an observed log-population value or otherwise a parameter that stands in place of a missing value. This is just like the example in the chapter, in which the vector N was a mix of observed neocortex percents and parameters that stood in place of missing values.

15M1. This is a subtle question. The key is recognize that the distributional assumption about the predictor that contains missing values does not contain any information about each case. As a consequence, it implicitly assumes that missing values are *randomly* located among the cases. Now keep

in mind that "random" only ever means that we do not know why the data turned out the way it did. It isn't a claim about causation, just a claim about information.

15M2. Let's consider just the simplest model, `m15.5`, but modify it to constrain B to a 0-1 interval. This also means we don't want to standardize *B*. I'll call if *P*, so there is no confusion. This is a subtle coding problem actually, because the chain will have a hard time initializing the imputed values, unless you assign them the right constraints. You can put an explicit prior on `P_impute` in the formula, using Uniform(0,1), to impose the right constraints. If you work directly with the Stan code, it's easier to appreciate what's going on.

The bigger issue is the priors. Once you use neocortex proportion directly, you'll have to assign another prior for the coefficient. It'll need to be much broader, because the variation in neocortex is small. And it'll be useful to manually center the values inside the linear model, so that the other priors still make sense.

R code 15.1

```
library(rethinking)
data(milk)
d <- milk
d$neocortex.prop <- d$neocortex.perc / 100
d$logmass <- log(d$mass)
dat_list2 <- list(
    K = standardize( d$kcal.per.g ),
    P = d$neocortex.prop ,
    M = standardize( d$logmass ) )

m15M2.1 <- ulam(
    alist(
        K ~ dnorm( mu , sigma ),
        mu <- a + bP*(P-0.67) + bM*M,
        P ~ dbeta2( nu , theta ),
        nu ~ dbeta( 2 , 2 ),
        a ~ dnorm( 0 , 0.5 ),
        bM ~ dnorm( 0, 0.5 ),
        bP ~ dnorm( 0 , 10 ),
        theta ~ dexp( 1 ),
        sigma ~ dexp( 1 ),
        vector[12]:P_impute ~ uniform(0,1)
    ) , data=dat_list2 , chains=4 , cores=4 , iter=2000 )
```

Now if you look at the posterior, you'll see that the imputed values are in the proper 0-1 interval. And if you extract them and plot them, you'll see they correlate almost perfectly with the previous imputed values.

15M3. This problem is as easy as modifying the code from the chapter to have double values for the standard error variable, `Divorce.SE`.

R code 15.2

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
dlist <- list(
    D_obs = standardize( d$Divorce ),
```

```
    D_sd = d$Divorce.SE / sd( d$Divorce ),
    M = standardize( d$Marriage ),
    A = standardize( d$MedianAgeMarriage ),
    N = nrow(d) )

m15M3.1 <- ulam(
    alist(
        D_obs ~ dnorm( D_true , D_sd*2.0 ),
        vector[N]:D_true ~ dnorm( mu , sigma ),
        mu <- a + bA*A + bM*M,
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        bM ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=dlist , chains=4 , cores=4 , iter=4000 )
```

You'll notice that this model does not sample very efficiently. Increasing the standard errors on the outcome variable has made the posterior less well identified, and this has made sampling harder in this case. But it does work. We can do better by using a non-centered parameterization:

R code 15.3

```
m15M3.2 <- ulam(
    alist(
        D_obs ~ dnorm( mu + z_true*sigma , D_sd*2.0 ),
        vector[N]:z_true ~ dnorm( 0 , 1 ),
        mu <- a + bA*A + bM*M,
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        bM ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=dlist , chains=4 , cores=4 , iter=4000 ,
    control=list(max_treedepth=14) )
```

That should sample much better for you.

Now compare the estimates produced. I'll omit the `div_est` estimates here.

R code 15.4

```
precis(m15.1) # original
precis(m15M3.2) # double standard error
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
a     -0.05 0.10 -0.21  0.11  1321     1
bA    -0.61 0.16 -0.87 -0.35  1184     1
bM     0.06 0.17 -0.20  0.32   981     1
sigma  0.58 0.11  0.42  0.77   596     1

       mean   sd  5.5% 94.5% n_eff Rhat4
a     -0.12 0.10 -0.28  0.05 11193     1
bA    -0.65 0.17 -0.91 -0.38  7490     1
bM     0.20 0.19 -0.11  0.49  7928     1
sigma  0.15 0.11  0.01  0.35  4411     1
```

The marginal posterior for `bA` is unchanged. But the posterior for `bR`, the coefficient for marriage rate, has gotten larger. Why? Increasing the standard errors has allowed different States to exert influence

on the regression. All of the States have less certain divorce rates now, but the States that were previously quite precisely estimated—usually very large States—are now substantially less precise. This shifts the balance of information among the States and alters the results.

15M4. Let's do a simple Gaussian simulation of the DAG:

R code 15.5

```
N <- 500
X <- rnorm(N)
Y <- rnorm(N,X)
Z <- rnorm(N,Y)
d <- list(X=X,Y=Y,Z=Z)
```

Now a simple linear regression of Y on X and Z:

R code 15.6

```
m15M4 <- ulam(
    alist(
        Y ~ dnorm( mu , sigma ),
        mu <- a + bX*X + bZ*Z,
        c(a,bX,bZ) ~ dnorm(0,1),
        sigma ~ dexp(1)
    ) , data=d , chains=4 )
precis(m15M4)
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
bZ     0.51 0.02  0.47  0.54  1563     1
bX     0.50 0.04  0.43  0.56  1550     1
a     -0.05 0.03 -0.10  0.00  2029     1
sigma  0.71 0.02  0.68  0.75  2149     1
```

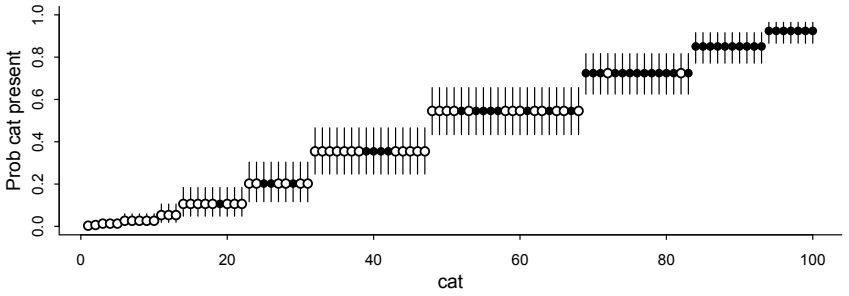
The coefficients suggest both X and Z contribute equally to Y. In terms of association, that is correct. But you know that isn't the right causal story. Only X causally influences Y. If you wanted the causal influence of X, you have gotten only half of it. That's a large bias.

This sort of "confound" doesn't have a specific name. But what is happening is that we are weakly conditioning on the outcome itself. Remember, Z is a descendant of Y. If we condition on a descendent, it's like conditioning on the parent, but weakly. You may have heard that conditioning on the outcome (selecting on the outcome) is bad news. It's the kind of design bias that cannot be recovered from statistically. In this case, we induce it statistically by including Z. It is strongly related to design problems like selection bias and case-control bias.

15M5. Run model `m15.9` and then let's compare the posterior probabilities `PrC1` to the true values in `cat`. There are lots of ways to do this comparison. I'm going to plot the posterior distribution of each cat and color the posterior mean by the true value: filled for present and open for absent.

R code 15.7

```
post <- extract.samples(m15.9)
PrC1_mean <- colMeans( post$PrC1 )
PrC1_PI <- apply( post$PrC1 , 2 , PI )
o <- order( PrC1_mean )
plot( NULL , xlim=c(1,100) , ylim=c(0,1) , xlab="cat" , ylab="Prob cat present" )
for ( i in 1:100 ) lines( rep(i,2) , PrC1_PI[,o[i]] )
for ( i in 1:100 ) points( i , PrC1_mean[o][i] , pch=ifelse( cat[o][i]==1 , 16 , 21 ) ,
    lwd=1.5 , bg="white" )
```



That's not too bad, in the sense that the posterior probability is a good guide to the truth.

How could we do better than this? We could have more data. In a Poisson model we can do this easily by adding an exposure to the generative model. For example swap out the `notes` line with:

R code 15.8

```
notes <- rpois( N_houses , 10*( alpha + beta*cat ) )
```

Run the new simulation and fit the model and you'll find that the model almost perfectly categorizes each cat. But that's only because there is so much evidence and there are no zeros in the data. We need models like this most when the data aren't enough. And in those cases, what the model does for us is calibrate our uncertainty. It doesn't do magic.

15M6. Let's start with the R code 15.8 on page 500 and move forward. The first model of missingness is where missing values in H are random with respect to everything else.

R code 15.9

```
N <- 100
S <- rnorm( N )
H <- rbinom( N , size=10 , inv_logit(S) )
D <- rbern( N ) # dogs completely random
Hm <- H
Hm[D==1] <- NA
```

Now a simple binomial GLM to estimate the causal influence of S on H.

R code 15.10

```
obs <- which( !is.na( Hm ) )
m15M6.1 <- ulam(
    alist(
        H ~ binomial( 10 , p ),
        logit(p) <- a + bS*S,
        a ~ normal( 0 , 1.5 ),
        bS ~ normal( 0 , 0.5 )
    ) , data=list( H=Hm[obs] , S=S[obs] ) , chains=4 )
precis( m15M6.1 )
```

```
   mean   sd  5.5% 94.5% n_eff Rhat4
a  0.06 0.10 -0.10  0.22  1280     1
bS 1.01 0.11  0.83  1.18  1209     1
```

The true effect is 1, so this is a pretty good estimate.

In the next simulation, studying causes dogs to eat homework:

R code 15.11

```
D <- ifelse( S > 0 , 1 , 0 )
Hm <- H
Hm[D==1] <- NA
```

You don't need a new statistical model here, because what we want to do is condition on S and we are doing that already.

R code 15.12

```
obs <- which( !is.na( Hm ) )
m15M6.2 <- ulam(
    alist(
        H ~ binomial( 10 , p ),
        logit(p) <- a + bS*S,
        a ~ normal( 0 , 1.5 ),
        bS ~ normal( 0 , 0.5 )
    ) , data=list( H=Hm[obs] , S=S[obs] ) , chains=4 )
precis( m15M6.2 )
```

```
   mean   sd  5.5% 94.5% n_eff Rhat4
a  0.06 0.10 -0.10  0.21  1431     1
bS 1.01 0.11  0.83  1.19   945     1
```

Again good.

The third example is completed for you in the chapter.

The fourth is not done for you. Starting R code 15.15 on page 503:

R code 15.13

```
N <- 100
S <- rnorm(N)
H <- rbinom( N , size=10 , inv_logit(S) )
D <- ifelse( H < 5 , 1 , 0 )
Hm <- H; Hm[D==1] <- NA
```

You know from the chapter that this example is a problem. The missingness is a function of the variable H itself. Let's see what the model says when we just drop the missing cases:

R code 15.14

```
obs <- which( !is.na( Hm ) )
m15M6.3 <- ulam(
    alist(
        H ~ binomial( 10 , p ),
        logit(p) <- a + bS*S,
        a ~ normal( 0 , 1.5 ),
        bS ~ normal( 0 , 0.5 )
    ) , data=list( H=Hm[obs] , S=S[obs] ) , chains=4 )
precis( m15M6.3 )
```

```
   mean   sd 5.5% 94.5% n_eff Rhat4
a  0.41 0.11 0.24  0.59   925     1
bS 0.80 0.14 0.59  1.02  1003     1
```

The estimate is biased downwards by the fact that the the worst home homework is missing from the sample. The qualitative conclusion is correct, however. Simulations like this one can help us to interpret estimates that come from biased and confounded models. Sometimes the bias can be bounded.

The question didn't ask you to try to impute the missing values. That's a much more advanced type of question. In this case, you'd need to use discrete imputation. I'm going to do this, using raw Stan

code, where it'll be easier. Let's start with just the problem of marginalizing over the missing values. Suppose we know the process that generates missing homework: dogs eat any homework with a score less than 5. So the probability of a missing value is the probability that the score was 4 or less. Does give us information about S? Let's make the model and see.

R code 15.15

```
Hmm <- Hm
Hmm[ is.na(Hmm) ] <- (-9)

c15M6.4 <- "
data{
    int H[100];
    vector[100] S;
}
parameters{
    real a;
    real bS;
}
model{
    vector[100] p;
    bS ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 1.5 );
    for ( i in 1:100 ) {
        p[i] = a + bS * S[i];
        p[i] = inv_logit(p[i]);
    }
    for ( i in 1:100 ) {
        if ( H[i] > -1 ) H[i] ~ binomial( 10 , p[i] );
        if ( H[i] < 0 ) {
            vector[5] pv;
            for ( j in 0:4 ) pv[j+1] = binomial_lpmf( j | 10 , p[i] );
            target += log_sum_exp( pv );
        }
    }
}
"

m15M6.4 <- stan( model_code=c15M6.4 , data=list( H=Hmm , S=S ) , chains=4 )
precis( m15M6.4 )
```

```
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -0.05 0.09 -0.19  0.09  2209     1
bS  1.10 0.11  0.92  1.29  2588     1
```

And there's our less biased estimate of the influence of S on H. There is information in the missing values, because once we know the process that generates missingness, we can predict it through the model and update the posterior distribution.

Now for the very advanced part: Let's also impute these unobserved homework scores. This means using the generated quantities block to compute the inverse. This is the new code we need to tack on the end:

R code 15.16

```
gq_code <- "
generated quantities{
    int H_impute[100];
    for ( i in 1:100 ) {
        real p = inv_logit(a + bS * S[i]);
```

```
        if ( H[i] > -1 ) H_impute[i] = H[i];
        if ( H[i] < 0 ) {
           // compute Pr( H==j | p , H < 5 )
           vector[5] lbp;
           real Z;
           for ( j in 0:4 ) lbp[j+1] = binomial_lpmf( j | 10 , p );
           // convert to probabilities by normalizing
           Z = log_sum_exp( lbp );
           for ( j in 1:5 ) lbp[j] = exp( lbp[j] - Z );
           // generate random sample from posterior
           H_impute[i] = categorical_rng( lbp ) - 1;
        }
    }
}
"
```

Now we paste this on the end of the previous code and fire it up.

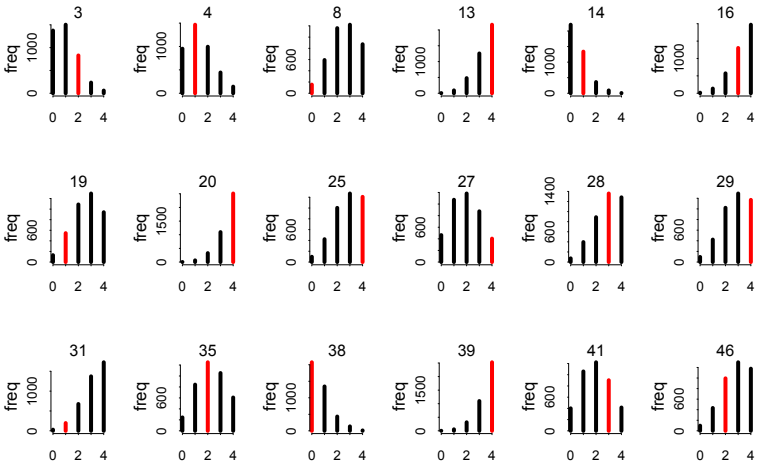
R code 15.17

```
code_new <- concat( c15M6.4 , gq_code )
m15M6.5 <- stan( model_code=code_new , data=list( H=Hmm , S=S ) , chains=4 )
post <- extract.samples( m15M6.5 )
```

Now we need to summarize the posterior distribution of each missing value. Instead of just showing the posterior mean, let's show the entire distribution. Here are the first 18 missing values, with the true value in red:

R code 15.18

```
par(mfrow=c(3,6),cex=1.05)
j <- which( is.na(Hm) )
for ( i in 1:18 ) {
    k <- H[j[i]] + 1
    col_vec <- rep( "black" , 5 )
    col_vec[k] <- "red"
    plot( table( post$H_impute[,j[i]] ) , col=col_vec , ylab="freq" , lwd=4 )
    mtext( j[i] )
}
```



There is a lot of posterior uncertainty, so the mean doesn't summarize these distributions well. But the model does manage to assign more mass to true values on average. It can do this, because high S values are more likely to have higher H values.

15H1. To prepare for the first model, lead the data and take a look at each variable:

R code 15.19

```
library(rethinking)
data(elephants)
d <- elephants
str(d)
```

```
'data.frame': 41 obs. of  2 variables:
 $ AGE    : int  27 28 28 28 28 29 29 29 29 29 ...
 $ MATINGS: int  0 1 1 1 3 0 0 0 2 2 ...
```

This is a very simple set of data. `AGE` contains ages in years of individual male elephants. `MATINGS` contains counts of matings for those individuals.

Before we jump into the model, let's think about how age should influence the number of matings. You could enter it as it is, in which case you are assuming an exponential relationship between age and matings, because the log link makes it so:

$$\log \lambda_i = \alpha + \beta A_i \implies \lambda_i = \exp(\alpha + \beta A_i) = \exp(\alpha)\exp(\beta A_i)$$

Maybe there is an exponential relationship with age. But there are other options. For example it could be that matings scale with the logarithm of age. Like this:

$$\log \lambda_i = \alpha + \beta \log A_i \implies \lambda_i = \exp(\alpha) A_i^{\beta}$$

This allows age to have increasing or diminishing returns on matings, depending up whether β is less than or greater than 1.

Let's use the logarithm approach. But using age directly as in a conventional GLM isn't wrong. Unlike in most examples, I'll keep age on the natural scale. This means that the intercept is now the expected matings at the start of sexual maturity (about 20 years old). So let's convert age to years after that by subtracting 20. The implied Poisson model predicting `MATINGS` with `AGE` is:

R code 15.20

```
m15H1.1 <- ulam(
    alist(
        MATINGS ~ dpois(lambda),
        lambda <- exp(a)*(AGE-20)^bA,
        a ~ dnorm(0,1),
        bA ~ dnorm(0,1)
    ), data=d , chains=4 )
precis( m15H1.1 )
```

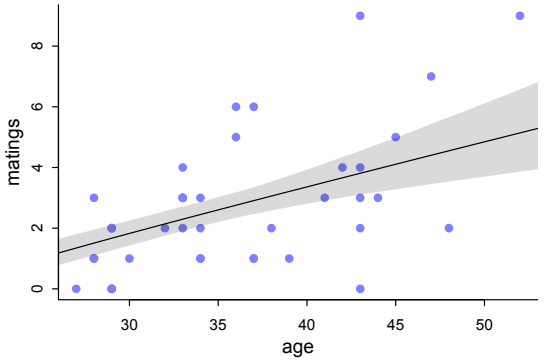
```
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -1.45 0.59 -2.41 -0.53   276  1.01
bA  0.89 0.21  0.55  1.22   270  1.01
```

These chains are not very efficient, but they did converge. Take a look at the trace plot, too. Let's plot the implied relationship:

R code 15.21

```
A_seq <- seq( from=25 , to=55 , length.out=30 )
lambda <- link( m15H1.1 , data=list(AGE=A_seq) )
lambda_mu <- apply(lambda,2,mean)
lambda_PI <- apply(lambda,2,PI)
```

```
plot( d$AGE , d$MATINGS , pch=16 , col=rangi2 ,
    xlab="age" , ylab="matings" )
lines( A_seq , lambda_mu )
shade( lambda_PI , A_seq )
```



That looks like a reliably positive relationship. Older elephants get more matings, on average.

Now let's assume each AGE value was measured with Gaussian error with standard deviation 5, as the problem suggests. Here's the new model. The only trick to observe, as in the chapter, is to manually add the [i] index to the predictor. I'm going to subtract 20 from the ages first, to make the coding simpler. We can add it back later, after sampling.

R code 15.22

```
d$AGE0 <- d$AGE - 20
m15H1.2 <- ulam(
    alist(
        MATINGS ~ dpois(lambda),
        lambda <- exp(a)*AGE_est[i]^bA,
        AGE0 ~ dnorm( AGE_est , 5 ),
        vector[41]:AGE_est ~ dunif( 0 , 50 ),
        a ~ dnorm(0,1),
        bA ~ dnorm(0,1)
    ), data=d , chains=4 )
precis( m15H1.2 )
```

```
41 vector or matrix parameters hidden. Use depth=2 to show them.
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -1.27 0.55 -2.18 -0.42   770     1
bA  0.83 0.19  0.53  1.13   808     1
```

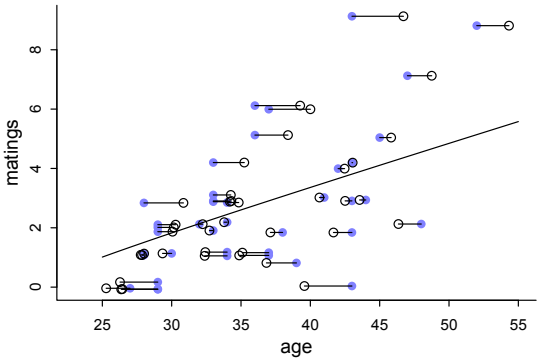
Ignoring the AGE_est parameters for the moment, the average association between MATINGS and AGE has not changed. Why not? The measurement error is both symmetric and the same for all ages. So this is unlike the divorce rate example in the chapter in the sense that the error is uniform. So adding equal error to all of the predictor values doesn't have much impact. Here, it has had essentially no impact on inference. Chances are that real measurement error would not be uniform across all ages. It would be much easier to distinguish among young ages than older ages, for example.

We can learn something more in this example by inspecting the posterior distributions of the AGE values, the AGE_est parameters. Let's extract them and compare the posterior means to the observed values. The plot I'll construct will have AGE on the horizontal and MATINGS on the vertical, with open points for the inferred posterior means and filled blue points for the observed, as usual. I'll connect each pair of points for the same animal with a line segment. Since so many points overlap, I'll also

add a little jitter to the vertical scale, so we can tell individuals apart more easily. Finally, I'll plot the mean regression trend.

R code 15.23

```
post <- extract.samples(m15H1.2)
AGE_est <- apply(post$AGE_est,2,mean) + 20

# make jittered MATINGS variable
MATINGS_j <- jitter(d$MATINGS)

# observed
plot( d$AGE , MATINGS_j , pch=16 , col=rangi2 ,
    xlab="age" , ylab="matings" , xlim=c(23,55) )

# posterior means
points( AGE_est , MATINGS_j )

# line segments
for ( i in 1:nrow(d) )
    lines( c(d$AGE[i],AGE_est[i]) , rep(MATINGS_j[i],2) )

# regression trend - computed earlier
lines( A_seq , lambda_mu )
```



The key thing to notice here is that the observations above the regression trend have been adjusted upwards in the posterior distribution, while the observations below the trend have been adjusted downwards. Why? Consider an observed number of matings that is above the expectation for a given observed age. This puts the blue point above the regression trend. For this observed age, measured with error, the matings exceed what is expected for that age. So the model "realizes" that the actual age of that individual is probably above what was measured and entered into the data table. So the open points (inferred) above the trend are to the right of the blue points (measured). Likewise, a blue point below the regression trend has a number of matings below what is expected for the measured age. So the model realizes that the actual age is probably below the measured age. So the open points (inferred) below the trend are to the left of the blue points (measured). Notice also that blue points farther from the regression trend shrink farther towards it. This is pooling, as usual.

15H2. All that's required to fit the new models is to adjust the `5` inside the distribution assigned to `AGE`. For example, let's begin by doubling the standard error to 10.

R code 15.24

```
m15H2.1 <- ulam(
    alist(
        MATINGS ~ dpois(lambda),
        lambda <- exp(a)*AGE_est[i]^bA,
        AGE0 ~ dnorm( AGE_est , 10 ),
        vector[41]:AGE_est ~ dunif( 0 , 50 ),
        a ~ dnorm(0,1),
        bA ~ dnorm(0,1)
    ), data=d , chains=4 )
precis( m15H2.1 )
```

```
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -1.12 0.54 -2.01 -0.30   635  1.01
bA  0.75 0.18  0.49  1.05   665  1.01
```

Smaller, but not close to zero. Double again and we get:

```
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -0.93 0.57 -1.90 -0.04   717  1.01
bA  0.65 0.18  0.37  0.94   705  1.01
```

At this rate, it might take a while. Let's jump to a standard deviation of 100:

```
    mean   sd  5.5% 94.5% n_eff Rhat4
a  -0.46 1.06 -1.76  1.92   170  1.02
bA  0.46 0.35 -0.36  0.86   161  1.02
```

Still not zero, and the chains sample very badly. But there is a lot of mass below zero for `bA`. The basic lesson is that as the error grows, it is increasingly hard to detect any association between age and mating.

15H3. Run the provided code to generate the data. It should look like this:

R code 15.25

```
set.seed(100)
x <- c( rnorm(10) , NA )
y <- c( rnorm(10,x) , 100 )
d <- list(x=x,y=y)
show(d)
```

```
$x
 [1] -0.50219235  0.13153117 -0.07891709  0.88678481  0.11697127  0.31863009
 [7] -0.58179068  0.71453271 -0.82525943 -0.35986213          NA

$y
 [1]  -0.4123062   0.2278056  -0.2805510   1.6266253   0.2403508   0.2893134
 [7]  -0.9706449   1.2253890  -1.7390736   1.9504347 100.0000000
```

Ignoring the last case, with the missing x value, these two variables have a strong positive association:

R code 15.26

```
precis( lm(y~x,d) )
```

```
            mean   sd  5.5% 94.5%
(Intercept) 0.24 0.28 -0.20  0.68
x           1.42 0.52  0.59  2.26
```

But what happens when we impute, using the known distribution for x. Here's the model given in the problem:

R code 15.27

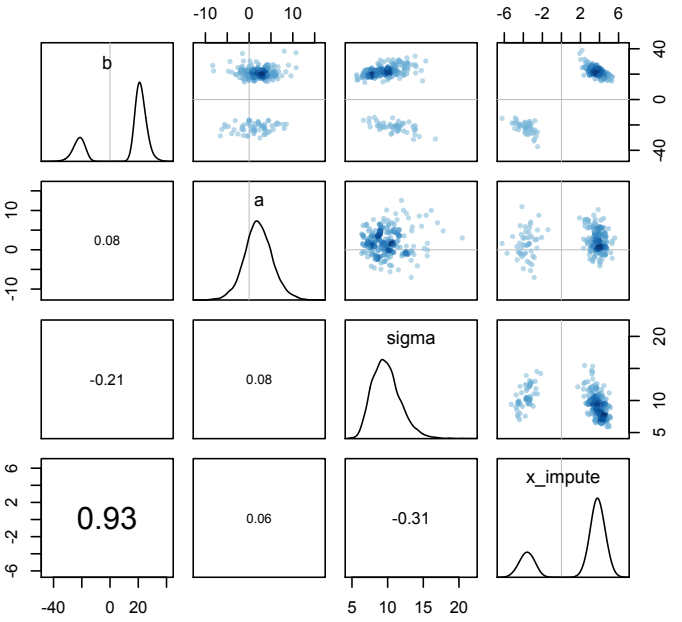
```
m15H3 <- ulam(
    alist(
        y ~ dnorm(mu,sigma),
        mu <- a + b*x,
        x ~ dnorm(0,1),
        c(a,b) ~ dnorm(0,100),
        sigma ~ dexp(1)
    ), data=d , chains=4 , iter=4000 ,
    control=list(adapt_delta=0.99) )
precis(m15H3)
```

```
              mean    sd   5.5% 94.5% n_eff Rhat4
b            10.86 19.49 -25.03 27.43     2  5.45
a             2.14  3.24  -2.91  7.42  3900  1.00
sigma         9.85  2.01   7.03 13.31    54  1.04
x_impute[1]   1.95  3.30  -4.19  4.83     2  4.98
```

Something weird is going on with both the imputed value, x_impute, and b. Look at the standard deviations and intervals. Let's look at the posterior distribution, focusing on the joint distribution of b and x_impute:

R code 15.28

```
pairs(m15H3)
```



The posterior indicates that the plausible values of the b coefficient, conditional on the data and model, are either strongly positive or equally strongly negative. And the posterior distribution for the missing *x* value, x_impute, is also bimodal. And jointly, the large b values go with the large x_impute values.

What has happened here? How has imputing a single missing x changed inference so much? And why is the posterior bimodal? Since a y value of 100 is an extremely large value, only an extremely large x value would be consistent with both that y and the original posterior mean for β, around 1.4. So for $y = 100$, the x consistent with the parameters (inferred ignoring the missing value) would be:

$$\begin{aligned} y &= \alpha + \beta x \\ 100 &= 0.24 + 1.4x \\ x &= (100 - 0.24)/1.4 \end{aligned}$$

And the answer is about $x = 71$. But since the prior assigned to x is relatively narrow, Normal$(0, 1)$, it is too implausible that the missing x is that large. So what are the alternatives? The posterior distribution has nominated them. Let's plot each of the modes, in terms of the implied regression relationship.

First, let's take the positive `b` samples and see what they imply. There are lots of ways to do this in the code. I'm just going to loop over the parameters and keep just the samples where `b` is positive. We'll need the samples where `b` is negative later, so I'll extract those into another list at the same time.

R code 15.29

```
post <- extract.samples(m15H3)
post_pos <- post
post_neg <- post
for ( i in 1:length(post) ) {
    post_pos[[i]] <- post[[i]][post$b>0]
    post_neg[[i]] <- post[[i]][post$b<0]
}
```

Now let's compute the regression line for the positive samples. Since we don't want to use the imputation parameter in constructing the line and its confidence region, we can't easily use `link` here. So we'll just do it ourselves:

R code 15.30

```
x_seq <- seq(from=-2.6,to=4,length.out=30)
mu_link <- function(x,post) post$a + post$b*x
mu <- sapply( x_seq , mu_link , post=post_pos )
mu_mu <- apply(mu,2,mean)
mu_PI <- apply(mu,2,PI)
```

Now to plot the data, including the imputed x value, but only for the positive samples still:

R code 15.31

```
x_impute <- mean(post_pos$x_impute)
plot( y ~ x , d , pch=16 , col=rangi2 , xlim=c(-0.85,x_impute) )
points( x_impute , 100 )
lines( x_seq , mu_mu )
shade( mu_PI , x_seq )
```

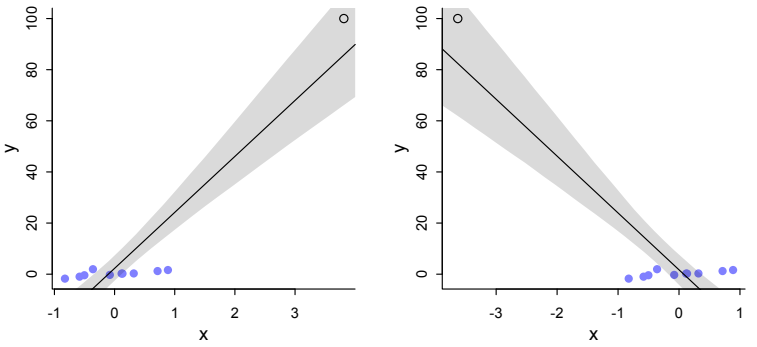
And before showing the result, let's also compute the negative relationship, so we can easily compare them:

R code 15.32

```
x_seq <- seq(from=-4,to=4,length.out=50)
mu <- sapply( x_seq , mu_link , post=post_neg )
mu_mu <- apply(mu,2,mean)
mu_PI <- apply(mu,2,PI)
x_impute <- mean(post_neg$x_impute)
plot( y ~ x , d , pch=16 , col=rangi2 , xlim=c(-3.7,0.9) )
```

```
points( x_impute , 100 )
lines( x_seq , mu_mu )
shade( mu_PI , x_seq )
```

Here are both plots, with the positive relationship on the left and the negative relationship on the right:



The only ways to make values of the missing x that are consistent with the model and data is to have a very strong positive or negative slope. Why? Because the missing x has to be much closer to zero than the original β value implied. This forces a steep slope on any regression relationship that will include the case with the missing value, shown by the open point in both plots above. The positive relationship remains more plausible than the negative one, because the other points (shown in blue) demonstrate a positive relationship between y and x.

You may want to change the assumed distribution on x in the model, to see what impact it has. You might even replace the mean and standard deviation of x with parameters, like the imputation example in the chapter.

15H4. Running the model without measurement error (the code was given in the problem), we get:

R code 15.33

```
precis( m15H4 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
a     0.42 0.06 0.33  0.51   267  1.02
b     0.78 0.01 0.76  0.80   232  1.03
sigma 0.29 0.02 0.27  0.32   261  1.00
```

We'll plot this implied relationship in a moment, after fitting the measurement error model too. To build the measurement error model, all we really need to do is add the observation process. This means that the observed values arise from their own distribution, each having a true value as the mean. We use these unknown true values in the regression. It looks like this:

R code 15.34

```
dat_list$N_spp <- length(dat_list$B)
dat_list$Mse <- Mse
dat_list$Bse <- Bse

m15H4.2 <- ulam(
    alist(
    # B model
        B ~ normal( B_true , Bse ),
```

```
        vector[N_spp]:B_true ~ dlnorm( mu , sigma ),
        mu <- a + b*log( M_true[i] ),

    # M model
        M ~ normal( M_true , Mse ),
        vector[N_spp]:M_true ~ normal( 0.5 , 1 ),

    # priors
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list ,
    start=list( M_true=dat_list$M , B_true=dat_list$B ) ,
    chains=4 , cores=4 , control=list(max_treedepth=15) )
```

The top chunk is the model for the B values. The first line is the measurement process. Then the next two lines are the same regression as before, but with `B_true` replacing the observed `B` values. Likewise `M_true` replaces the observed `M` in the linear model.

The second chunk is the measurement model for `M`. The prior for `M_true` covers the entire range of the normalize variable—it ranges from 0 to 1 now, recall, because we scaled it that way to start by dividing by the maximum observed value.

The last chunk holds the same priors as before.

Note the `control` list at the bottom. If you run this model without that, it will work, but be inefficient and warn about exceeding maximum "treedepth." This is not a concern for the validity of the chains, just how well they run. Treedepth is a control parameter for NUTS algorithm. The Stan manual contains more detail, if you want it.

Once it finishes:

```
precis( m15H4.2 )
```

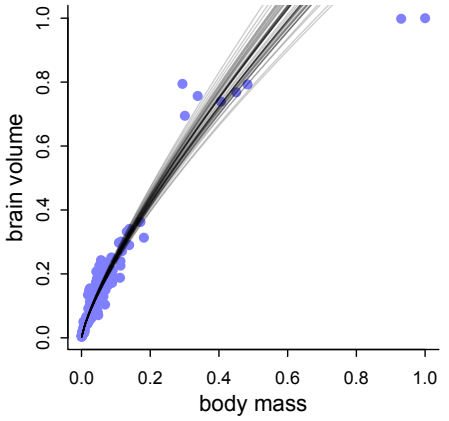
R code 15.35

```
364 vector or matrix parameters hidden. Use depth=2 to show them.
      mean   sd 5.5% 94.5% n_eff Rhat4
a     0.42 0.06 0.33  0.51   947     1
b     0.78 0.01 0.76  0.81   970     1
sigma 0.26 0.02 0.24  0.29  1262     1
```

Those 364 hidden parameters are the estimated true values. We can look at those later on. For now, notice that the posterior distributions of `a` and `b` are nearly identical to `m1.1`. Adding measurement error hasn't changed a thing! Plotting the regression against the observed values:

```
plot( B ~ M , xlab="body mass" , ylab="brain volume" , col=rangi2 , pch=16 )
post <- extract.samples(m15H4.2)
for ( i in 1:50 ) curve( exp(post$a[i])*x^(post$b[i]) , add=TRUE , col=grau(0.2) )
```

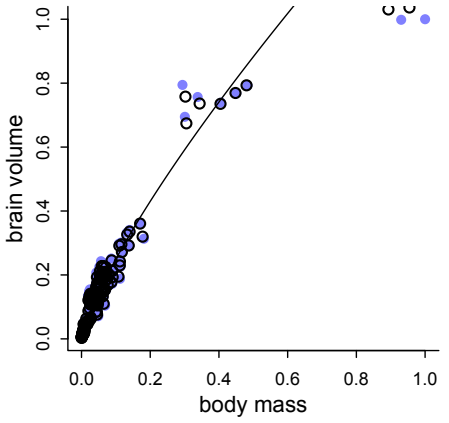
R code 15.36



The two points in the upper right are gorillas. Most primates are small, and obviously gorillas have something special going on. Now let's plot the estimated values on this:

R code 15.37

```
B_est <- apply( post$B_true , 2 , mean )
M_est <- apply( post$M_true , 2 , mean )

plot( B ~ M , xlab="body mass" , ylab="brain volume" , col=rangi2 , pch=16 )
points( M_est , B_est , pch=1 , lwd=1.5 )

x_seq <- seq( from=0 , to=1 , length.out=100 )
EB <- sapply( x_seq , function(x) mean( exp(post$a)*x^(post$b) ) )
lines( x_seq , EB )
```



The open points are the posterior mean estimates. Notice that they have moved towards the regression line, as you'd expect. But even the outlier gorillas haven't moved much. The assumed error just isn't big enough to get them any closer.

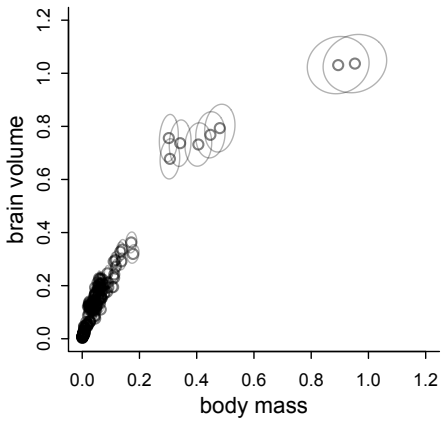
If you increase the amount of error, you can get all of the species to fall right on the regression line. Try for example 30% error. The model will mix poorly, but take a look at the inferred true values.

The truth of this example is that there are just so many small primates that they dominate the relationship. And their measurement errors are also smaller (in absolute terms). So adding plausible amounts of measurement error here doesn't make a big difference. We still don't have a good explanation for gorillas.

Before moving on, I'll also plot the estimated species values with 50% compatibility ellipses.

R code 15.38

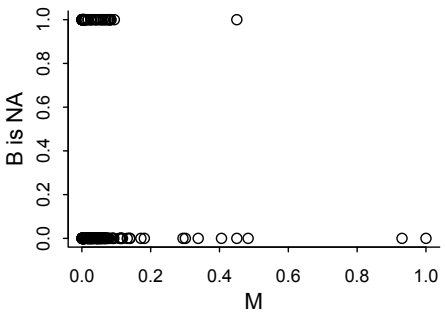
```
library(ellipse)
plot( B_est ~ M_est , xlab="body mass" , ylab="brain volume" , lwd=1.5 ,
    col=grau() , xlim=c(0,1.2) , ylim=c(0,1.2) )
for ( i in 1:length(B_est) ) {
    SIGMA <- cov( cbind( post$M_true[,i] , post$B_true[,i] ) )
    el <- ellipse( SIGMA , centre=c(M_est[i],B_est[i]) , level=0.5 )
    lines( el , col=grau(0.3) )
}
```



15H5. First, let's see where the missing values are, to get some idea about the missingness mechanism. If missing brain sizes are associated with certain ranges of body sizes, then it isn't plausibly MCAR (dog eats any homework). Let's plot body size against missingness:

R code 15.39

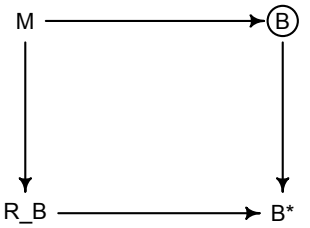
```
Bna <- is.na(d$brain[cc])
plot( Bna ~ M , ylab="B is NA" )
```



Looks like the missing brain values are almost all for small bodied species. This implies at least a MAR (dog eats students' homework) mechanism. Let's try a DAG to express it:

R code 15.40

```
library(dagitty)
dag2 <- dagitty('dag{
    M -> R_B -> "B*" <- B
    M -> B
}')
coordinates(dag2) <- list( x=c(M=0,B=1,R_B=0,"B*"=1),
                           y=c(M=0,B=0,R_B=1,"B*"=1) )
drawdag( dag2 , shapes=list(B="c") )
```



M here is body mass, B (unobserved, suggested by the circle) is brain size, R_B is the missingness mechanism, and B* is the observed brain sizes (with missing values). The arrow from M to R_B indicates that body size influences missingness. In this case, it would imply that small body size makes a missing brain value more likely.

Now let's do some imputation. Remember that the model for imputation is really no different than an ordinary model. It just needs a prior for any variable with missing values. In this case, the missing values are in the outcome, so the likelihood is the prior we need. So the model doesn't change at all. In `ulam`:

R code 15.41

```
dat_list <- list(
    B = B,
    M = M )

m15H5.1 <- ulam(
    alist(
        B ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) ,
    data=dat_list , chains=4 , cores=4 ,
    start=list( B_impute = rep(0.5,56) ) )
```

`ulam` figures out how to do the imputation. But an equivalent model that is more explicit would be:

R code 15.42

```
m15H5.1b <- ulam(
    alist(
        B_merge ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        B_merge <- merge_missing( B , B_impute ),
        a ~ normal(0,1),
        b ~ normal(0,1),
```

```
        sigma ~ exponential(1)
    ) , data=dat_list ,  chains=4 , cores=4  ,
    start=list( B_impute = rep(0.5,56) ) )
```

It's a little more obvious now what `ulam` is doing. It constructs the merged vector of observed and imputed values, `B_merge`, and then uses that merged vector as the outcome. The outcome distribution at the top of the model is the prior for each `B_impute` parameter. That prior is adaptive—it has parameters inside it. Hence, shrinkage happens.

Here is the posterior summary for the imputation model:

R code 15.43

```
precis( m15H5.1 )
```

```
56 vector or matrix parameters hidden. Use depth=2 to show them.
      mean   sd 5.5% 94.5% n_eff Rhat4
a     0.43 0.06 0.33  0.52   969     1
b     0.78 0.01 0.76  0.81   992     1
sigma 0.29 0.02 0.27  0.32  1343     1
```

The same analysis on only complete cases:

R code 15.44

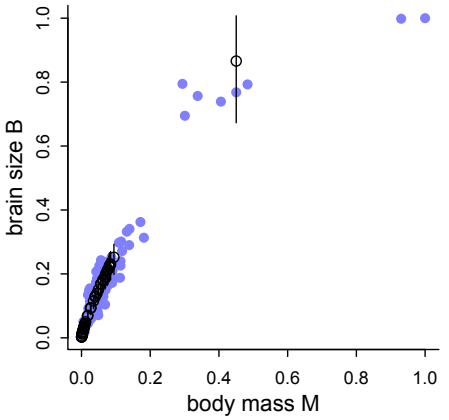
```
cc2 <- complete.cases( B )

dat_list2 <- list(
    B = B[cc2],
    M = M[cc2] )

m15H5.2 <- ulam(
    alist(
        B ~ dlnorm( mu , sigma ),
        mu <- a + b*log(M),
        a ~ normal(0,1),
        b ~ normal(0,1),
        sigma ~ exponential(1)
    ) , data=dat_list2 ,  chains=4 , cores=4 )

precis( m15H5.2 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
a     0.43 0.06 0.34  0.52   671     1
b     0.78 0.01 0.76  0.81   680     1
sigma 0.29 0.02 0.27  0.32   979     1
```

Really no difference from before. Let's plot the imputed values:

R code 15.45

```
post <- extract.samples(m15H5.1)
Bi <- apply( post$B_impute , 2 , mean )
miss_idx <- which( is.na(B) )

plot( M[-miss_idx] , B[-miss_idx] , col=rangi2 , pch=16 ,
    xlab="body mass M" , ylab="brain size B" )
points( M[miss_idx] , Bi )
Bi_ci <- apply( post$B_impute , 2 , PI , 0.5 )
for ( i in 1:length(Bi) ) lines( rep(M[miss_idx][i],2) , Bi_ci[,i] )
```



Black open points are the imputed values, with 50% compatibility intervals. Imputation hasn't done much, apparently because all but one of the missing values are in a very dense region of the body size range. So almost no information was lost—the missing info is redundant.

It would be useful to perform some analysis of this sort when deciding which new measurements to make. We should focus empirical work on the species that will provide the most information gain from the perspective of specific questions.

15H6. From the code in the chapter, load and prepare the data:

R code 15.46

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
dlist <- list(
    D_obs = standardize( d$Divorce ),
    D_sd = d$Divorce.SE / sd( d$Divorce ),
    M_obs = standardize( d$Marriage ),
    M_sd = d$Marriage.SE / sd( d$Marriage ),
    A = standardize( d$MedianAgeMarriage ),
    N = nrow(d) )
```

The new model needs to replace the fixed prior for the true M values with a prior that contains a linear model with A. This is because M is a function of A in the causal model. This code will work, and I've highlighted the new lines by spacing around them:

R code 15.47

```
m15H6.1 <- ulam(
    alist(
        D_obs ~ dnorm( D_true , D_sd ),
        vector[N]:D_true ~ dnorm( mu , sigma ),
        mu <- a + bA*A + bM*M_true[i],
        M_obs ~ dnorm( M_true , M_sd ),

        vector[N]:M_true ~ dnorm( muM , sigmaM ),
        muM <- aM + bAM*A,

        c(a,aM) ~ dnorm(0,0.2),
        c(bA,bM,bAM) ~ dnorm(0,0.5),
```

```
        sigma ~ dexp( 1 ),
        sigmaM ~ dexp( 1 )
    ) , data=dlist , chains=4 , cores=4 )
```

The question didn't ask for any specific diagnostics from the model. But let's briefly consider the posterior distribution:

```
precis( m15H6.1 )
```

R code 15.48

```
100 vector or matrix parameters hidden. Use depth=2 to show them.
        mean   sd  5.5% 94.5% n_eff Rhat4
aM     -0.11 0.07 -0.23  0.01  2047     1
a      -0.02 0.10 -0.18  0.13  2302     1
bAM    -0.67 0.08 -0.80 -0.53  2613     1
bM      0.29 0.26 -0.13  0.68  1521     1
bA     -0.47 0.19 -0.78 -0.17  1778     1
sigma   0.56 0.11  0.39  0.74   714     1
sigmaM  0.44 0.07  0.34  0.56  1093     1
```

Let's compare this to model `m15.2` in the chapter:

```
precis( m15.2 )
```

R code 15.49

```
100 vector or matrix parameters hidden. Use depth=2 to show them.
       mean   sd  5.5% 94.5% n_eff Rhat4
a     -0.04 0.10 -0.19  0.12  1629  1.01
bA    -0.54 0.16 -0.80 -0.29   912  1.00
bM     0.19 0.20 -0.14  0.52   677  1.00
sigma  0.57 0.10  0.41  0.74   760  1.00
```

There are some small differences, which suggests the imputed true values differ a little. Let's look at the details, comparing the `M_true` values, so we can understand what has happened.

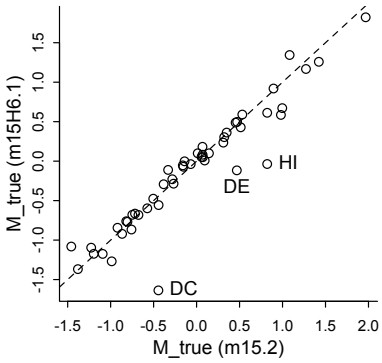
```
post1 <- extract.samples(m15H6.1)
post2 <- extract.samples(m15.2)
Mtrue1 <- colMeans( post1$M_true )
Mtrue2 <- colMeans( post2$M_true )
plot( Mtrue2 , Mtrue1 , xlab="M_true (m15.2)" , ylab="M_true (m15H6.1)" )
abline( a=0 , b=1 , lty=2 )
```

R code 15.50

And I'll label a few interesting points:

```
identify( Mtrue2 , Mtrue1 , d$Loc )
```

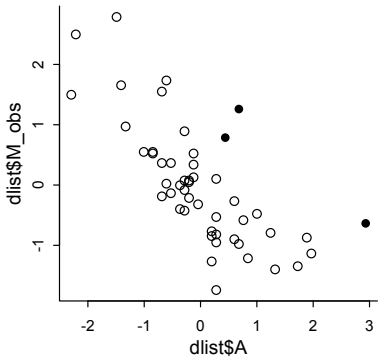
R code 15.51



For the most part, the posterior means are the same. But there are three States below the equality line. In these States, the estimated *M* value is lower in the new model than it was in the previous model. Let's look at the *A* and *M* values for the States, to see where these three (DC, DE and HI) lie:

R code 15.52

```
lstates <- c("DC","DE","HI")
plot( dlist$A , dlist$M_obs , pch=ifelse(d$Loc %in% lstates,16,1) )
```



These States have unusually large observed *M* values for their observed *A* values. So when we included *A* in the model for *M*, it corrects the true estimates to be more in line with the trend. This means reducing the estimated true *M* values.

15H7. Let's begin by keying in the data, the counts of the outcomes from 1 to 8:

R code 15.53

```
y <- c( 18 , 19 , 22 , NA , NA , 19 , 20 , 22 )
```

You might just look at these numbers and intuit that 20 4s and 20 5s is the best guess. And you wouldn't be wrong. But we're going to justify that guess with probability theory now.

Let's begin by listing the unknown variables. Then we'll assign priors and update with the observations. The unobserved variables are (1) the number of spins, (2) the vector of probabilities of each number 1-8, and (3) the counts of 4s and 5s. Let's consider each in turn.

Let *S* be the number of spins. We know that *S* is at least:

R code 15.54

```
sum(y,na.rm=TRUE)
```

```
[1] 120
```

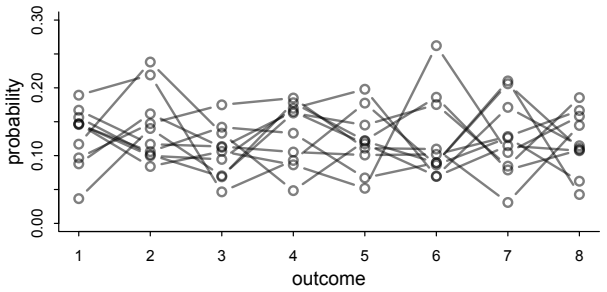
So that's the lower bound. There is no obvious upper bound. With only those facts to go on, one choice would be a Poisson prior for the number of spins. We also need an expected value for this Poisson prior. We could make that a parameter as well, with its own prior, or we could fix it. To keep things (relatively) simple, let's fix it at $8 \times 20 = 160$. So the prior is Poisson, but truncated below so that the minimum is 120. The easy way to approach this is to parameterize the number of spins above 120 and give that a Poisson distribution with a mean of 40. So the distribution of S is:

$$S \sim \text{Poisson}(40) + 120$$

Now we need a prior for the vector of probabilities. Let's call that vector p. As the problem suggested, we'll use a Dirichlet prior. Dirichlet was introduced in an earlier chapter, where we used it as a prior for categorical outcomes like this. The problem says we want the prior to encode the knowledge that none of the outcomes is twice as likely as the others. We can't express that directly in a Dirichlet, but we can find a parameterization that gets it qualitatively right. Let's try:

R code 15.55

```
library(gtools)
p <- rdirichlet( 1e3 , alpha=rep(4,8) )
plot( NULL , xlim=c(1,8) , ylim=c(0,0.3) , xlab="outcome" , ylab="probability" )
for ( i in 1:10 ) lines( 1:8 , p[i,] , type="b" , col=grau() , lwd=2 )
```



Hard to tell exactly what is going on from that plot. But we can count the number of simulated draws in which any probability is more than twice as large as another. One way to do this is to sort each vector of probabilities and ask whether the largest is more than twice as big as the smallest.

R code 15.56

```
twicer <- function( p ) {
    o <- order( p )
    if ( p[o][8]/p[o][1] > 2 ) return( TRUE ) else return( FALSE )
}
sum( apply( p , 1 , twicer ) )
```

```
[1] 978
```

978 of 1000 simulations violate the criterion. So we need a tighter prior.

R code 15.57

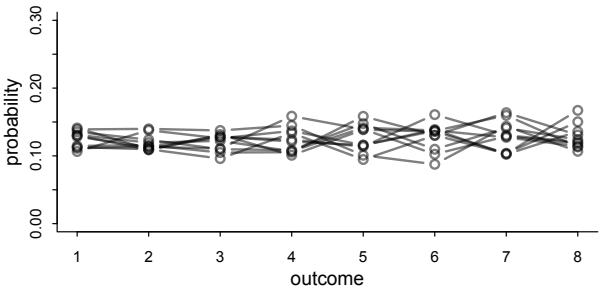
```
p <- rdirichlet( 1e3 , alpha=rep(50,8) )
sum( apply( p , 1 , twicer ) )
```

```
[1] 13
```

That's pretty good. It's also a very tight prior. Let's look at it:

R code 15.58

```
plot( NULL , xlim=c(1,8) , ylim=c(0,0.3) , xlab="outcome" , ylab="probability" )
for ( i in 1:10 ) lines( 1:8 , p[i,] , type="b" , col=grau() , lwd=2 )
```



Now both the total number of spins S and the counts of 4s and 5s are both discrete variables. So we can't declare them in Stan as parameters. We'll instead have to marginalize over them in the model block and then use the generated quantities to invert the likelihoods to get posterior probabilities. Luckily, we can focus on the 4s and 5s, because that's all we were asked for.

I'll do this is raw Stan code, because we'll need some loops and fancier data structures. This book hasn't taught these coding techniques. But any attempt that gets the approach right—the notion that we need to marginalize over all combinations of 4s and 5s—is gold. Here's the code, and then I'll explain.

R code 15.59

```
code15H7 <- '
data{
    int N;
    int y[N];
    int y_max; // consider at most this many spins for y4 and y5
    int S_mean;
}
parameters{
    simplex[N] p;   // probabilities of each outcome
}
model{
    vector[(1+y_max)*(1+y_max)] terms;
    int k = 1;

    p ~ dirichlet( rep_vector(50,N) );

    // loop over possible values for unknown cells 4 and 5
    // this code updates posterior of p
    for ( y4 in 0:y_max ) {
        for ( y5 in 0:y_max ) {
            int Y[N] = y;
            Y[4] = y4;
            Y[5] = y5;
            terms[k] = poisson_lpmf(y4+y5|S_mean-120) + multinomial_lpmf(Y|p);
            k = k + 1;
        }//y5
    }//y4
    target += log_sum_exp(terms);
}
```

```
generated quantities{
    matrix[y_max+1,y_max+1] P45; // prob y4,y5 takes joint values
    // now compute Prob(y4,y5|p)
    {
        matrix[(1+y_max),(1+y_max)] terms;
        int k = 1;
        real Z;
        for ( y4 in 0:y_max ) {
            for ( y5 in 0:y_max ) {
              int Y[N] = y;
              Y[4] = y4;
              Y[5] = y5;
              terms[y4+1,y5+1] = poisson_lpmf(y4+y5|S_mean-120) + multinomial_lpmf(Y|p);
            }//y5
        }//y4
        Z = log_sum_exp( to_vector(terms) );
        for ( y4 in 0:y_max )
            for ( y5 in 0:y_max )
                P45[y4+1,y5+1] = exp( terms[y4+1,y5+1] - Z );
    }
}
'
```

At the top, the data block declares the observed variables and fixed parts of the priors. We feed these values in with this:

R code 15.60

```
y <- c(18,19,22,-1,-1,19,20,22)
dat <- list(
    N = length(y),
    y = y,
    S_mean = 160,
    y_max = 40 )
```

Notice that I've replace the `NA` values in `y` with -1. This is because Stan can't accept `NA`.

The parameters block just declares the vector of outcome probabilities p. We'll assign the prior for this in the next block.

The model block is big. We'll take it one step at a time. The first line declares a vector to accumulate the marginalization terms for the probabilities of each count of 4s and 5s. We'll consider every possible combination of 4s and 5s from 0 to `y_max` (40 here) for each, for a total of `(y_max+1)*(y_max+1)` terms. When `y_max` is 40, this is 1681 terms. Stan can handle it. The counter `k` is there to help us manage this vector.

After assigning the Dirichlet prior to p, there is a big loop over all combinations of 4s and 5s. For each combination, we compute the log-probability of the entire vector `Y` (with the hypothetical 4s and 5s inserted) and that many total spins. The total spins have a Poisson probability and the vector of counts `Y` has a multinomial probability conditional on that number of spins and the vector `p`. This is the same strategy as at the end of the chapter where we marginalized over the unobserved cats. Now however there are many many (1681) terms to marginalize over. This code computes the likelihood of the observed counts, marginalizing over the unobserved 4s and 5s.

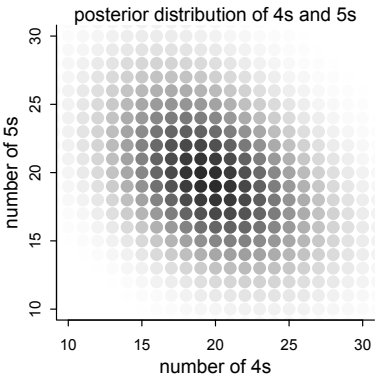
The generated quantities block repeats most of this code, in order to compute the posterior probability of each combination of 4s and 5s. It begins by declaring a matrix `P45` of these combinations.

Then the loops come again. But now we store the individual log-probabilities in a big matrix of combinations of 4s and 5s. Each of these terms is $\log \mathrm{P}(Y, S|p)$. At the bottom, we normalize these terms so that they all sum to 1 and we have a proper joint distribution over all combinations of 4s and 5s.

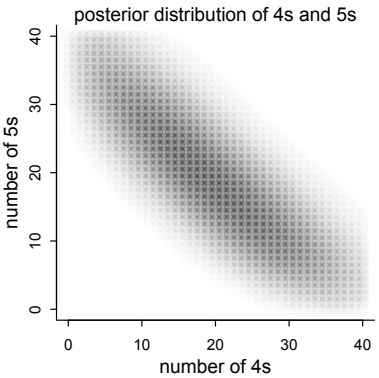
Now let's run the Stan model, extract the samples, and plot the joint distribution of 4s and 5s:

R code 15.61

```
m15H7 <- stan( model_code=code15H7 , data=dat , chains=4 , cores=4 )
post <- extract.samples(m15H7)
y_max <- dat$y_max
plot( NULL , xlim=c(10,y_max-10) , ylim=c(10,y_max-10) ,
    xlab="number of 4s" , ylab="number of 5s" )
mtext( "posterior distribution of 4s and 5s" )
for ( y4 in 0:y_max ) for ( y5 in 0:y_max ) {
    k <- grau( mean( post$P45[,y4+1,y5+1] )/0.01 )
    points( y4 , y5 , col=k , pch=16 , cex=1.5 )
}
```



So as you may have intuited, 20 4s and 20 5s is most plausible. But combinations of counts near there are almost equally plausible. Notice also the negative correlation in the counts of 4s and 5s. This is because more 4s makes more 5s less plausible, given that we have a prior on the total number of spins that keeps it from ballooning to infinity and that the prior on p vector keeps the counts of each outcome relatively close to one another. If you change the Dirichlet prior to a vector of 2s, for example, you'll get instead:



In the middle, 20 4s and 20 5s is still most plausible, but there is a lot of mass along the diagonal now.

16. Chapter 16 Solutions

16E1. A lot could be said here. GLMs are generalized devices for measuring associations. While they are very flexible, they are not specialized to any particular scientific problem. This means that the parameters may not have clear scientific meaning. This makes the priors harder to build and the inferences harder to translate across scales and measurements. GLMs can also be so easy to extend to include new variables that their ease may risk silly overfitting and confounding. More mechanistic models are harder to extend, and this can at times be an ironic advantage.

16E2. Many examples are possible here. Any example from the chapter is sufficient. Simple physics models provide a number of examples. For example, the stopping distance of a vehicle can be approximated as:

$$d = \frac{v^2}{2mg}$$

where v is the velocity when the brake is applied, m is the "coefficient of friction," and g is gravitational acceleration (9.8 m/s^2 on Earth). In this equation, v and d and m are observable/measurable variables. We might consider a regression of d on v and m, for example, in the GLM framework. But there is no linear combination of v and m that will give us the expression. If m is constant and we ignore it, however, then the function can be approximated by:

$$d = \beta v^2$$

where β to be estimated is $1/(2mg)$. This is a typical thing about GLMs: Under suitable approximations or by making some things constant, almost everything is a GLM. That doesn't mean we won't do better by attending the a more scientific model.

16E3. There was an example in an earlier chapter, the Poisson model of Oceanic tools. The expected number of tools was:

$$\lambda_i = \alpha P^{\beta}/\gamma$$

Taking the log of both sides:

$$\log \lambda_i = \log(\alpha) + \beta \log(P) - \log(\gamma)$$

And that is a linear model in which the difference $\log(\alpha) - \log(\gamma)$ is the intercept. Many models with purely multiplicative relations will begin additive on the log scale. The growth model in the chapter is another example.

16M1. All we need is to replace the 3 with a new parameter with some appropriate prior. I'll use an exponential prior, so that the parameter is constrained to be positive. Nothing else makes any sense.

R code 16.1

```
m16M1.1 <- ulam(
    alist(
        w ~ dlnorm( mu , sigma ),
        exp(mu) <- 3.141593 * k * p^2 * h^a,
        p ~ beta( 2 , 18 ),
        k ~ exponential( 0.5 ),
        sigma ~ exponential( 1 ),
        a ~ exponential( 1 )
```

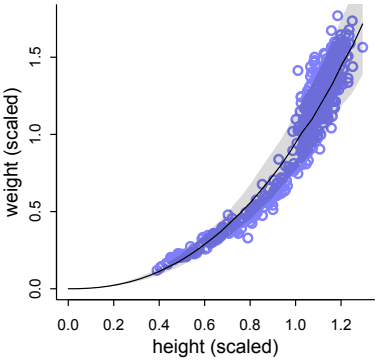
```
    ), data=d , chains=4 , cores=4 )
precis( m16M1.1 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
p     0.24 0.05 0.17  0.34   688  1.01
k     5.80 2.69 2.52 10.85   649  1.00
sigma 0.13 0.00 0.12  0.13  1143  1.00
a     2.32 0.02 2.29  2.36  1140  1.00
```

The new parameter is a. Instead of 3, we get 2.3. Let's look at the posterior predictions:

R code 16.2

```
h_seq <- seq( from=0 , to=max(d$h) , length.out=30 )
w_sim <- sim( m16M1.1 , data=list(h=h_seq) )
mu_mean <- apply( w_sim , 2 , mean )
w_CI <- apply( w_sim , 2 , PI )
plot( d$h , d$w , xlim=c(0,max(d$h)) , ylim=c(0,max(d$w)) , col=rangi2 ,
    lwd=2 , xlab="height (scaled)" , ylab="weight (scaled)" )
lines( h_seq , mu_mean )
shade( w_CI , h_seq )
```



Compared to the original model, this model is fitting the shorter individuals better but the taller individuals worse.

16M2. Let's simulate directly from the prior:

R code 16.3

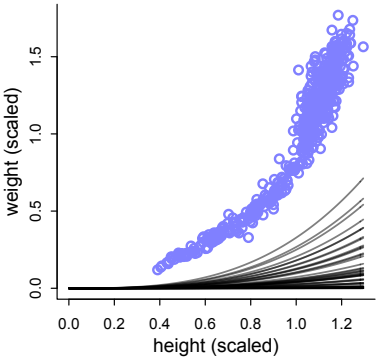
```
N <- 50
p <- rbeta( N , 2 , 18 )
k <- rexp( N , 0.5 )
sigma <- rexp( N , 1 )
prior <- list( p=p , k=k , sigma=sigma )
```

And then we can simulate observations for a range of heights. I'll plot prior trends over the observed range of height in the sample:

R code 16.4

```
h_seq <- seq( from=0 , to=max(d$h) , length.out=30 )
plot( d$h , d$w , xlim=c(0,max(d$h)) , ylim=c(0,max(d$w)) , col=rangi2 ,
    lwd=2 , xlab="height (scaled)" , ylab="weight (scaled)" )
for ( i in 1:N ) {
```

```
    with( prior ,
        curve( 3.141593 * k[i] * p[i]^2 * x^3 ,
            add=TRUE , lwd=1.5 , col=grau() ) )
}
```

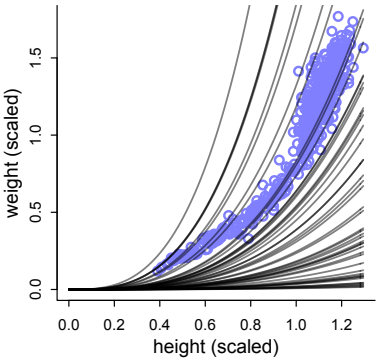


That prior is really not growing fast enough for a human population. Either p or k or both needs to be larger on average. We don't want to fit the data right now. But we do want to capture reasonable adult weight, which this prior does not.

Let's try again:

R code 16.5

```
N <- 50
p <- rbeta( N , 4 , 18 )
k <- rexp( N , 1/4 )
sigma <- rexp( N , 1 )
prior <- list( p=p , k=k , sigma=sigma )
```

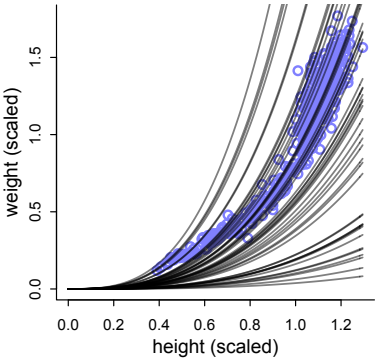


That's better, but it still masses up on the horizontal axis. This is because the prior for k is exponential, which puts a lot of mass near zero. We could try log-Normal instead. Remember, the mean of a log-Normal is very sensitive to the variance, $\exp(\mu + \sigma^2/2)$. And the median is $\exp(\mu)$. So let's try $\mu = \log(7)$ and $\sigma = 0.1$:

R code 16.6

```
N <- 50
p <- rbeta( N , 4 , 18 )
k <- rlnorm( N , log(7) , 0.2 )
sigma <- rexp( N , 1 )
```

```
prior <- list( p=p , k=k , sigma=sigma )
```



This is a lot better. It's still implausibly spread out. But at least it doesn't exclude realistic growth trajectories now, and it doesn't mass up on the horizontal axis either.

There is so much data here that these priors do almost no work, aside from enforcing constraints. But you won't always be so lucky.

16M3. Let's start by drawing from the priors:

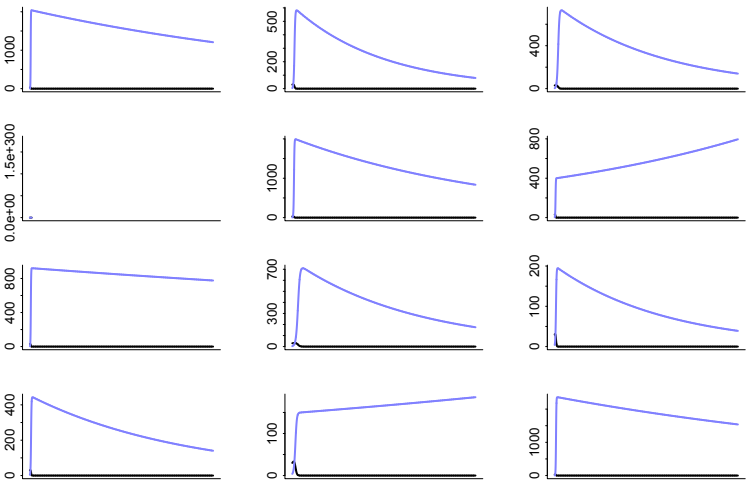
R code 16.7

```
N <- 12
theta <- matrix( NA , nrow=N , ncol=4 )
theta[,1] <- rnorm( N , 1 , 0.5 )
theta[,3] <- rnorm( N , 1 , 0.5 )
theta[,2] <- rnorm( N , 0.05 , 0.05 )
theta[,4] <- rnorm( N , 0.05 , 0.05 )

theta[,1] <- rnorm( N , 0.5 , 0.05 )
theta[,3] <- rnorm( N , 0.025 , 0.05 )
theta[,2] <- rnorm( N , 0.05 , 0.05 )
theta[,4] <- rnorm( N , 0.5 , 0.05 )
```

And then we can use the simulation code in the chapter (the `sim_lynx_hare` function) to simulate from these priors:

R code 16.8

```
par(mfrow=c(4,3),cex=1.05)
par(mgp = c(1, 0.2, 0), mar = c(1, 2, 1, 1) + 0.1,
        tck = -0.02, cex.axis = 0.8, bty = "l" )

for ( s in 1:N ) {
    z <- sim_lynx_hare( 1e4 , as.numeric(Lynx_Hare[1,2:3]) , theta[s,] )
    ymax <- max(z)
    if ( ymax == Inf ) ymax <- 1.79e+300
    plot( NULL , ylim=c(0,ymax) , xlim=c(0,1e4) , xaxt="n" , ylab="" , xlab="" )
    lines( z[,2] , col="black" , lwd=2 )
    lines( z[,1] , col=rangi2 , lwd=2 )
}
```

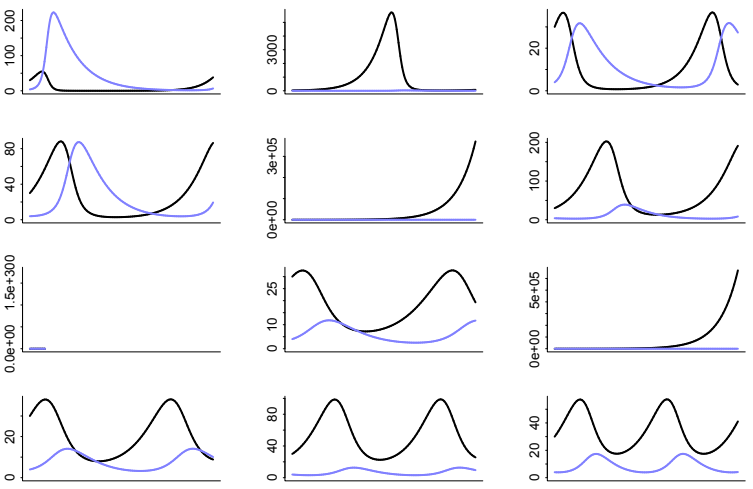


That's terrible. None of these cycle at all, and most of them just have explosive Lynx growth (blue trends).

Let's try to do better. Try these, nudging the priors towards values that can at least sometimes cycle:

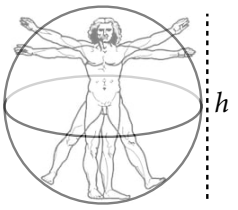
R code 16.9

```
N <- 12
theta <- matrix( NA , nrow=N , ncol=4 )
theta[,1] <- rnorm( N , 0.5 , 0.1 )
theta[,3] <- rnorm( N , 0.025 , 0.05 )
theta[,2] <- rnorm( N , 0.05 , 0.05 )
theta[,4] <- rnorm( N , 0.5 , 0.1 )
```



Some of these still explode in growth, but at least it is the hares now. And some do cycle.

16M4. The equation for the volume of a sphere, as derived by Archimedes, is $V = (4/3)\pi r^3$, where r is the sphere's radius. So height $h = 2r$ in our context. This gives us $V = (4/3)\pi(h/2)^3 = (1/6)\pi h^3$. Or as Leonardo da Vinci would have drawn it:



$V = (1/6)\pi h^3$

There is no longer a proportionality parameter p for the base, but we still need a conversion factor for volume to weight, k. This gives us:

$$W = k\frac{1}{6}\pi h^3$$

Now before we fit this to the data, notice that it is already very similar to the cylinder model. Both have the structure:

$$W = k[\text{stuff}]h^3$$

So they should produce very similar fits. The parameter k is this model will act like the product of p and k in the original model. But the constant factor $1/6$ in this model means k won't exactly be the produce of the two original parameters.

Now let's fit this beast to the data:

R code 16.10

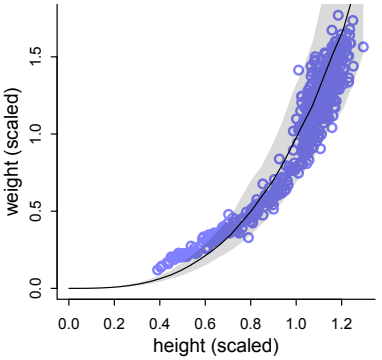
```
m16M4.1 <- ulam(
    alist(
        w ~ dlnorm( mu , sigma ),
        exp(mu) <- k * 3.141593/6 * h^3,
        k ~ exponential( 0.5 ),
        sigma ~ exponential( 1 )
    ), data=d , chains=4 , cores=4 )
precis( m16M4.1 )
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
k     1.81 0.02 1.78  1.84  1408     1
sigma 0.21 0.01 0.20  0.22  1309     1
```

Alright, let's look at the posterior predictions:

R code 16.11

```
h_seq <- seq( from=0 , to=max(d$h) , length.out=30 )
w_sim <- sim( m16M4.1 , data=list(h=h_seq) )
mu_mean <- apply( w_sim , 2 , mean )
w_CI <- apply( w_sim , 2 , PI )
plot( d$h , d$w , xlim=c(0,max(d$h)) , ylim=c(0,max(d$w)) , col=rangi2 ,
    lwd=2 , xlab="height (scaled)" , ylab="weight (scaled)" )
lines( h_seq , mu_mean )
shade( w_CI , h_seq )
```



Yeah, that looks almost identical. People are obviously spheres (not).

16H1. This problem requires giving each sex a different maximum, and for that we need to get either an index variable or dummy variable for sex. Let's begin by loading the data as in the chapter.

R code 16.12

```
library(rethinking)
data(Panda_nuts)
dat_list <- list(
    n = as.integer( Panda_nuts$nuts_opened ),
    age = Panda_nuts$age / max(Panda_nuts$age),
    seconds = Panda_nuts$seconds )
```

Now we want to add an index (or dummy) for sex. I'll use a dummy variable in this case, because we want to make sure males are on average larger, and this is easier to code with a dummy variable.

R code 16.13

```
dat_list$male_id <- ifelse( Panda_nuts$sex=="m" , 1L , 0L )
```

Now the model will just need a small tweak. The parameter ϕ incorporates adult mass and its conversion to strength. So we can add the male-female contrast to this when the actor is male. I'll make the effect multiplicative, so that the new parameter is the extra proportion of female mass. Adult male chimpanzees are on average about 40% heavier than females. You could do it as an additive effect, but then you'd interpret it differently.

I'll use a positively constrained prior, to enforce males being larger. Here's the modified model:

R code 16.14

```
m16H1.1 <- ulam(
    alist(
        n ~ poisson( lambda ),
        lambda <- seconds*(1+pm*male_id)*phi*(1-exp(-k*age))^theta,
        phi ~ lognormal( log(1) , 0.1 ),
        pm ~ exponential( 2 ),
        k ~ lognormal( log(2) , 0.25 ),
        theta ~ lognormal( log(5) , 0.25 )
    ), data=dat_list , chains=4 )
precis(m16H1.1)
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
phi   0.60 0.05 0.52  0.68   695  1.01
pm    0.67 0.14 0.46  0.92   764  1.00
```

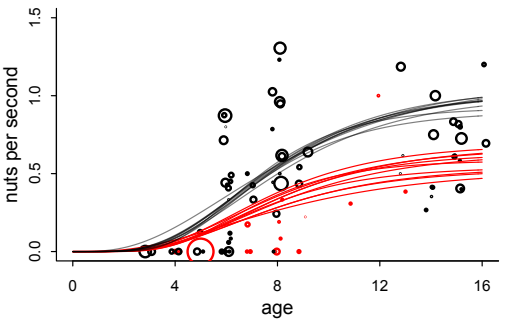
```
k     5.21 0.69 4.05  6.27   690  1.01
theta 7.73 1.86 5.01 11.02   730  1.01
```

So the posterior mean of `pm` is 0.67, which indicates males are 1.67 times stronger at maximum. Note the wide compatibility interval, though. There are possible substantial individual differences.

Let's plot the posterior predictions now, separating males and females. I'll plot 10 posterior curves for females (red) and males.

R code 16.15

```
post <- extract.samples(m16H1.1)
plot( NULL , xlim=c(0,1) , ylim=c(0,1.5) , xlab="age" ,
    ylab="nuts per second" , xaxt="n" )
at <- c(0,0.25,0.5,0.75,1,1.25,1.5)
axis( 1 , at=at , labels=round(at*max(Panda_nuts$age)) )

pts <- dat_list$n / dat_list$seconds
point_size <- normalize( dat_list$seconds )
points( jitter(dat_list$age) , pts , lwd=2 , cex=point_size*3 ,
    col=ifelse(dat_list$male_id==1,"black","red") )

# 10 female curves
for ( i in 1:10 ) with( post ,
    curve( phi[i]*(1-exp(-k[i]*x))^theta[i] , add=TRUE , col="red" ) )

# 10 male curves
for ( i in 1:10 ) with( post ,
    curve( (1+pm[i])*phi[i]*(1-exp(-k[i]*x))^theta[i] , add=TRUE , col=grau() ) )
```



That is a very clear difference. Since this is a strength-dependent skill, this difference isn't surprising. Notice also that the red points are smaller, which indicates taht female chimpanzees were observed less often performing the task.

16H2. The parameter ϕ is maximum size/strength, and the parameter k is essentially growth rate. Let's allow ϕ to vary by individual. We will need a variable for individual ID, to link together observations from the same individual. Let's code that:

R code 16.16

```
dat_list$id <- Panda_nuts$chimpanzee
```

The model needs a different ϕ offset for each individual. The trick is to keep the total rate `lambda` positive. If we use ordinary Gaussian varying intercepts, we can't guarantee that, because the Gaussian can be negative. But there's no reason we can't use another distribution for varying effects. Let's use an exponential. This means each effect will be a multiplicative factor of ϕ, from 0 upwards. We'll set

the prior mean at 1, meaning no difference from the average. You could also try a log-normal or a beta distribution, letting the scale parameter stretch the distribution.

I'll use a non-centered parameterization. I'll also add the rescaled versions to the posterior using generated quantities. This will let us interpret them more easily after sampling. Here's the model code.

R code 16.17

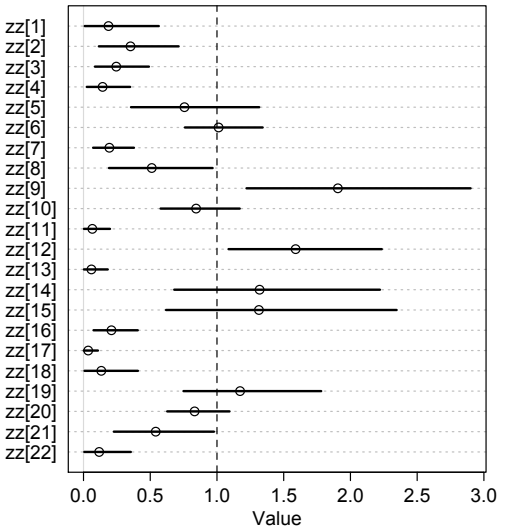
```
m16H2.1 <- ulam(
    alist(
        n ~ poisson( lambda ),
        lambda <- seconds*(phi*z[id]*tau)*(1-exp(-k*age))^theta,
        phi ~ lognormal( log(1) , 0.1 ),
        z[id] ~ exponential( 1 ),
        tau ~ exponential(1),
        k ~ lognormal( log(2) , 0.25 ),
        theta ~ lognormal( log(5) , 0.25 ),
        gq> vector[id]:zz <<- z*tau # rescaled
    ), data=dat_list , chains=4 , cores=4 ,
    control=list(adapt_delta=0.99) , iter=4000 )
precis(m16H2.1)
```

```
      mean   sd 5.5% 94.5% n_eff Rhat4
phi   1.00 0.10 0.85  1.17  8657     1
tau   0.65 0.21 0.39  1.03  2127     1
k     3.06 0.74 1.98  4.34  4426     1
theta 3.15 0.66 2.26  4.31  5614     1
```

The parameter `tau` indicates individual differences. It's firmly away from zero. To get an idea of the absolute effect of these differences, let's plot the varying effects.

R code 16.18

```
plot( precis( m16H2.1 , 2 , pars="zz" ) )
abline( v=1 , lty=2 )
```



Each of these is the proportion of ϕ for each individual. Values below 1 are smaller/weaker than average. Values above 1 are larger/stronger than average. There is considerable variation here, although some of it may be confounded with age, as individuals aren't observed across many years.

One problem with this model is that the scale parameter `tau` doesn't act like a typical scale for varying effects. As it shrinks to zero, the average adult mass shrinks too. In a usual varying effect model, as the scale shrinks to zero, the mean doesn't change, only the difference among individuals vanish. One way to get `tau` to behave in the usual way would be to exponentiate a sum like `exp(phi + z[id]*tau)`, but this would require reinterpreting the priors and constraints on these parameters, as now they can be negative on the latent scale. Like this:

R code 16.19

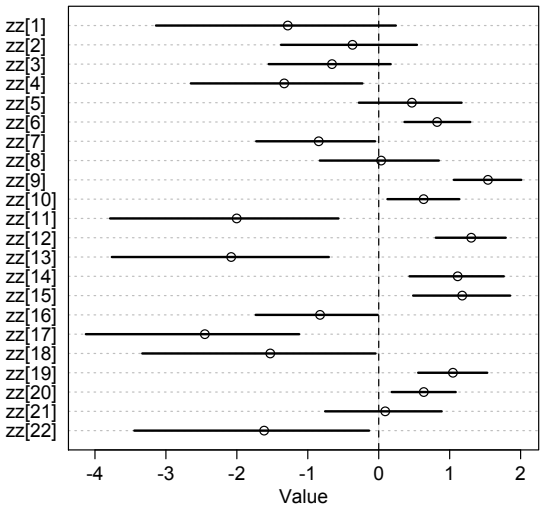
```
m16H2.2 <- ulam(
    alist(
        n ~ poisson( lambda ),
        lambda <- seconds*exp(phi+z[id]*tau)*(1-exp(-k*age))^theta,
        phi ~ normal( 0 , 0.5 ),
        z[id] ~ normal( 0 , 1 ),
        tau ~ exponential(1),
        k ~ lognormal( log(2) , 0.25 ),
        theta ~ lognormal( log(5) , 0.25 ),
        gq> vector[id]:zz <<- z*tau # rescaled
    ), data=dat_list , chains=4 , cores=4 ,
    control=list(adapt_delta=0.99) , iter=4000 )
precis(m16H2.2)
```

```
       mean   sd  5.5% 94.5% n_eff Rhat4
phi   -0.74 0.32 -1.25 -0.22  2589     1
tau    1.42 0.34  0.97  2.02  1583     1
k      2.66 0.67  1.71  3.82  4276     1
theta  3.07 0.59  2.25  4.14  5765     1
```

Having `phi` negative here just means that it is below 1 on the exponentiated scale. The new varying effects:

R code 16.20

```
plot( precis( m16H2.2 , 2 , pars="zz" ) )
abline( v=0 , lty=2 )
```



If you compare the previous plot, you'll see the high and low values are in the same places.

16H3. Let's construct the lag terms, as given in the problem prompt, and build the autoregressive model.

R code 16.21

```
data(Lynx_Hare)
dat_ar1 <- list(
    lynx = Lynx_Hare$Lynx[2:21],
    lynx_lag1 = Lynx_Hare$Lynx[1:20],
    hare = Lynx_Hare$Hare[2:21],
    hare_lag1 = Lynx_Hare$Hare[1:20] )

m16H3.1 <- ulam(
    alist(
        hare ~ lognormal( log(muh) , sigmah ),
        lynx ~ lognormal( log(mul) , sigmal ),
        muh <- ah + phi_hh*hare_lag1 + phi_hl*lynx_lag1,
        mul <- al + phi_ll*lynx_lag1 + phi_lh*hare_lag1,
        c(ah,al) ~ normal(0,1),
        phi_hh ~ normal(1,0.5),
        phi_hl ~ normal(-1,0.5),
        phi_ll ~ normal(1,0.5),
        phi_lh ~ normal(1,0.5),
        c(sigmah,sigmal) ~ exponential(1)
    ) , data=dat_ar1 , chains=4 , cores=4 )
precis( m16H3.1 )
```

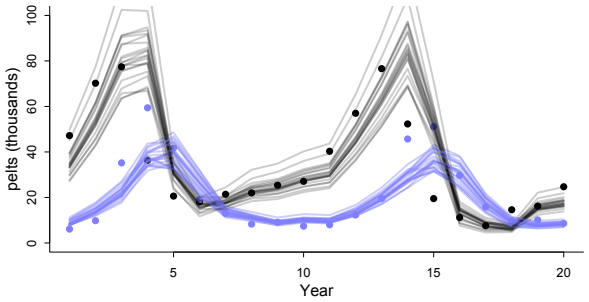
```
        mean   sd  5.5% 94.5% n_eff Rhat4
al     -0.93 0.91 -2.32  0.57  1865     1
ah      0.78 0.95 -0.69  2.30  2231     1
phi_hh  1.16 0.16  0.92  1.42  1327     1
phi_hl -0.19 0.10 -0.35 -0.03  1345     1
phi_ll  0.54 0.09  0.40  0.68  1828     1
phi_lh  0.25 0.05  0.18  0.33  1399     1
sigmal  0.31 0.06  0.23  0.40  1549     1
sigmah  0.45 0.08  0.34  0.60  1818     1
```

It's not easy to tell what is going on from these coefficients, but let's try. All of the `phi` parameters are positive, except for `phi_hl`, which is the effect on hares of lynx. So more lynx means fewer hare. That makes sense. The other parameters are positive, indicating that more hare means more lynx. That makes sense too. So these parameters make some sense.

Let's look at the implied time series.

R code 16.22

```
post <- extract.samples(m16H3.1)
plot( dat_ar1$hare , pch=16 , xlab="Year" , ylab="pelts (thousands)" , ylim=c(0,100) )
points( dat_ar1$lynx , pch=16 , col=rangi2 )
mu <- link(m16H3.1)
for ( s in 1:21 ) {
    lines( 1:20 , mu$muh[s,] , col=col.alpha("black",0.2) , lwd=2 )
    lines( 1:20 , mu$mul[s,] , col=col.alpha(rangi2,0.4) , lwd=2 )
}
```



Blue are lynx. If you compare this plot to the one in the chapter, you'll see that this model has a hard time with the lynx series in particular, which doesn't rise fast enough. But it also can't follow the hare series, as it doesn't drop in time.

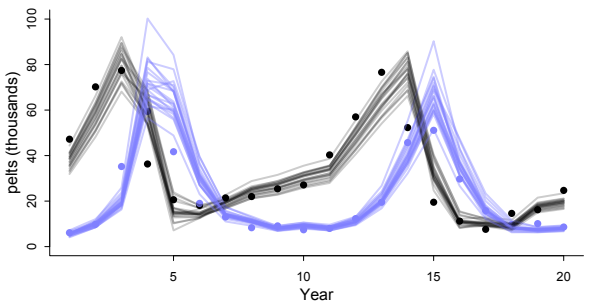
This model has a couple of structural problems. The first is that it treats the observed values as being measured without error. The second is that is treats effects as linear, when they are not. This is why it can't turn fast enough, while the ODE model can. Or another way to think of this is that the ODE model encodes interactions between the species, while this one just has main effects of each on the other.

Let's try a lagged interaction model. If you look at the ODE model in the chapter, you'll see that hare effect lynx only through the product of lynx and hare. Likewise, hare are impacted by lynx through the product of the two. So let's try an autoregressive model with similar structure.

R code 16.23

```
m16H3.2 <- ulam(
    alist(
        hare ~ lognormal( log(muh) , sigmah ),
        lynx ~ lognormal( log(mul) , sigmal ),
        muh <- ah + phi_hh*hare_lag1 + phi_hl*lynx_lag1*hare_lag1,
        mul <- al + phi_ll*lynx_lag1 + phi_lh*hare_lag1*lynx_lag1,
        c(ah,al) ~ normal(0,1),
        phi_hh ~ normal(1,0.5),
        phi_hl ~ normal(-1,0.5),
        phi_ll ~ normal(1,0.5),
        phi_lh ~ normal(1,0.5),
        c(sigmah,sigmal) ~ exponential(1)
    ) , data=dat_ar1 , chains=4 , cores=4 )
```

Plotting again:



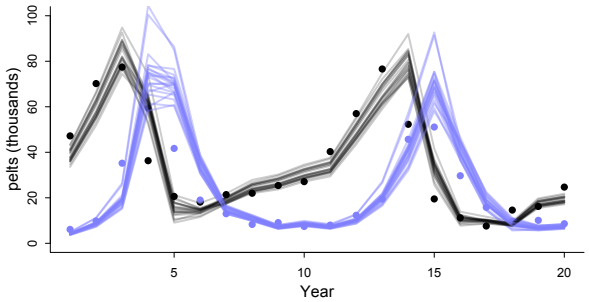
That's a lot better, but the lynx are dramatically overshooting the data.

Another problem with this model may be that the intercepts don't make any sense. If there are no hare, then there will be no hare in the future (absent immigration). What happens if we remove the intercepts?

R code 16.24

```
m16H3.3 <- ulam(
    alist(
        hare ~ lognormal( log(muh) , sigmah ),
        lynx ~ lognormal( log(mul) , sigmal ),
        muh <- phi_hh*hare_lag1 + phi_hl*lynx_lag1*hare_lag1,
        mul <- phi_ll*lynx_lag1 + phi_lh*hare_lag1*lynx_lag1,
        phi_hh ~ normal(1,0.5),
        phi_hl ~ normal(-1,0.5),
        phi_ll ~ normal(1,0.5),
        phi_lh ~ normal(1,0.5),
        c(sigmah,sigmal) ~ exponential(1)
    ) , data=dat_ar1 , chains=4 , cores=4 )
```

Plotting yet again:



This seems to be slightly worse than before. We're chasing epicycles here.

16H4. We need to construct the new lag variables. Then the 2-step autoregressive model has a similar structure as before. We'll start with the basic linear model with no interactions, as in the previous problem.

R code 16.25

```
dat_ar2 <- list(
    lynx = Lynx_Hare$Lynx[3:21],
    lynx_lag1 = Lynx_Hare$Lynx[2:20],
    lynx_lag2 = Lynx_Hare$Lynx[1:19],
    hare = Lynx_Hare$Hare[3:21],
    hare_lag1 = Lynx_Hare$Hare[2:20],
    hare_lag2 = Lynx_Hare$Hare[1:19] )

m16H4.1 <- ulam(
    alist(
        hare ~ lognormal( log(muh) , sigmah ),
        lynx ~ lognormal( log(mul) , sigmal ),
        muh <- ah + phi_hh*hare_lag1 + phi_hl*lynx_lag1 +
                    phi2_hh*hare_lag2 + phi2_hl*lynx_lag2,
        mul <- al + phi_ll*lynx_lag1 + phi_lh*hare_lag1 +
                    phi2_ll*lynx_lag2 + phi2_lh*hare_lag2,
```

```
        c(ah,al) ~ normal(0,1),
        phi_hh ~ normal(1,0.5),
        phi_hl ~ normal(-1,0.5),
        phi_ll ~ normal(1,0.5),
        phi_lh ~ normal(1,0.5),
        phi2_hh ~ normal(0,0.5),
        phi2_hl ~ normal(0,0.5),
        phi2_ll ~ normal(0,0.5),
        phi2_lh ~ normal(0,0.5),
        c(sigmah,sigmal) ~ exponential(1)
    ) , data=dat_ar2 , chains=4 , cores=4 )
precis( m16H4.1 )
```

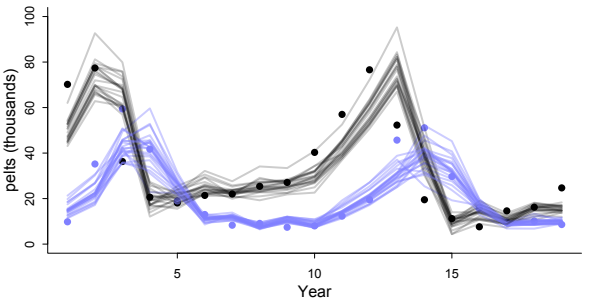
```
         mean   sd  5.5% 94.5% n_eff Rhat4
al      -0.48 0.96 -1.97  1.08  1944     1
ah       0.38 1.01 -1.25  1.97  1972     1
phi_hh   1.02 0.20  0.70  1.35   988     1
phi_hl  -0.74 0.34 -1.29 -0.20   822     1
phi_ll   0.91 0.24  0.52  1.28   930     1
phi_lh   0.39 0.13  0.18  0.60  1045     1
phi2_hh  0.18 0.28 -0.27  0.65   806     1
phi2_hl  0.40 0.16  0.14  0.66  1044     1
phi2_ll -0.19 0.11 -0.35 -0.01  1021     1
phi2_lh -0.23 0.20 -0.55  0.09   927     1
sigmal   0.30 0.06  0.22  0.40  1390     1
sigmah   0.39 0.07  0.30  0.52  1548     1
```

These parameters are even harder to understand. So the effect of lynx on hard after 1 time step, `phi_hl`, is negative, but the effect after 2 time steps, `phi2_hl`, is positive. If you inspect the pairs of 1-lag and 2-lag coefficients, you'll see several sign flips.

Let's look at the predictions:

R code 16.26

```
plot( dat_ar2$hare , pch=16 , xlab="Year" , ylab="pelts (thousands)" , ylim=c(0,100) )
points( dat_ar2$lynx , pch=16 , col=rangi2 )
mu <- link(m16H4.1)
for ( s in 1:21 ) {
    lines( 1:19 , mu$muh[s,] , col=col.alpha("black",0.2) , lwd=2 )
    lines( 1:19 , mu$mul[s,] , col=col.alpha(rangi2,0.4) , lwd=2 )
}
```

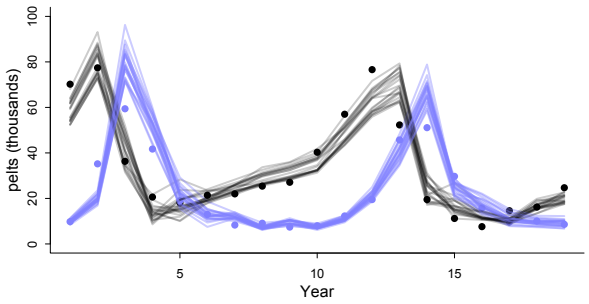


As before, the hare series has trouble dropping fast enough, and the lynx don't rise fast enough.

Let's try the interaction model. This is going to get messy:

R code 16.27

```
m16H4.2 <- ulam(
    alist(
        hare ~ lognormal( log(muh) , sigmah ),
        lynx ~ lognormal( log(mul) , sigmal ),
        muh <- ah + phi_hh*hare_lag1 + phi_hl*lynx_lag1*hare_lag1 +
                    phi2_hh*hare_lag2 + phi2_hl*lynx_lag2*hare_lag2,
        mul <- al + phi_ll*lynx_lag1 + phi_lh*hare_lag1*lynx_lag1 +
                    phi2_ll*lynx_lag2 + phi2_lh*hare_lag2*lynx_lag2,
        c(ah,al) ~ normal(0,1),
        phi_hh ~ normal(1,0.5),
        phi_hl ~ normal(-1,0.5),
        phi_ll ~ normal(1,0.5),
        phi_lh ~ normal(1,0.5),
        phi2_hh ~ normal(0,0.5),
        phi2_hl ~ normal(0,0.5),
        phi2_ll ~ normal(0,0.5),
        phi2_lh ~ normal(0,0.5),
        c(sigmah,sigmal) ~ exponential(1)
    ) , data=dat_ar2 , chains=4 , cores=4 )
```

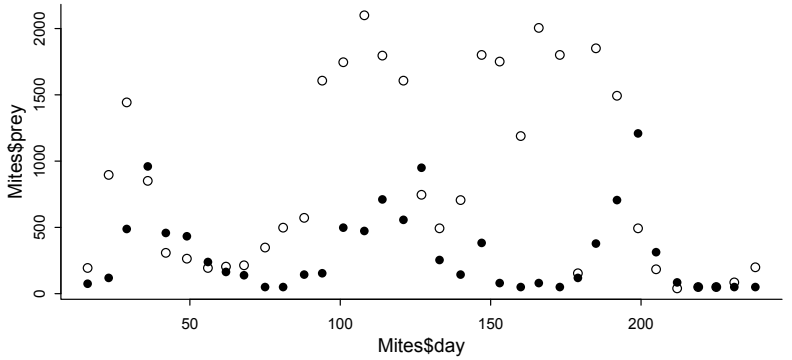


And the lynx now overshoot. This is very similar to the lag-1 model. The additional lag terms don't help much, in this case. In a strictly causal interpretation, they don't make any sense. Only the recent past can influence the present. If the distant past influences the present, it must be through the recent past. But models with multiple lag terms are not rare.

16H5. Let's begin by just loading the data and plotting it:

R code 16.28

```
library(rethinking)
data(Mites)

plot( Mites$day , Mites$prey )
points( Mites$day , Mites$predator , pch=16 )
```



The horizontal axis is day in the experiment and the vertical axis is count of individuals. The open circles are prey and the filled circles are predators. There does seem to be some kind of cycle there, maybe.

We need to build the model and we need priors. The model is actually simpler than the one in the book, because there won't be a measurement layer. We have actual mite counts here, not some error-prone proxy like pelts. Let's start with the priors, simulating the true counts as we did in the chapter. If you play around, you can find some priors that will produce cycles. We need to the simulation function from the chapter:

R code 16.29

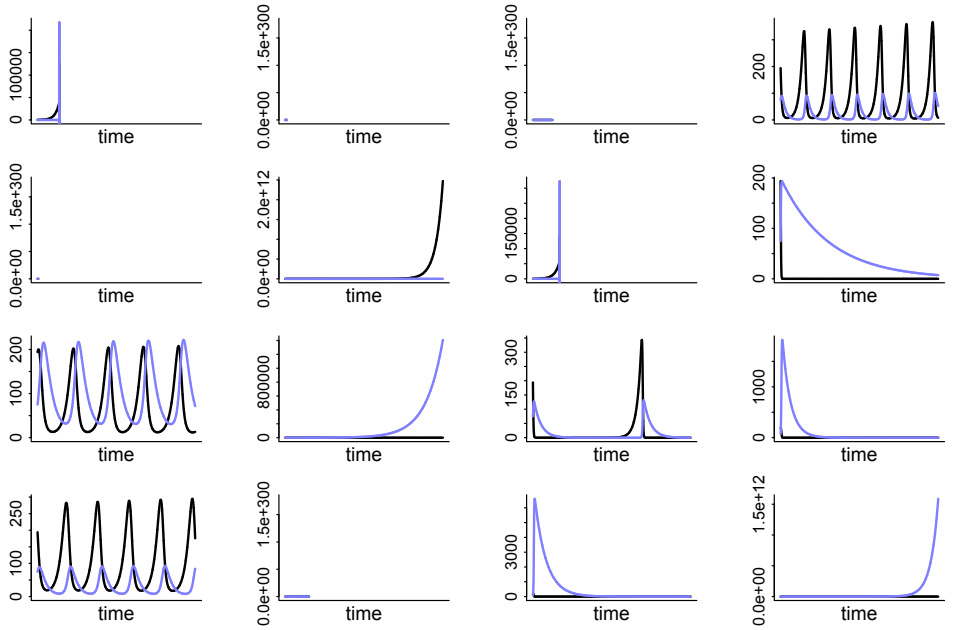
```
sim_mites <- function( n_steps , init , theta , dt=0.002 ) {
    L <- rep(NA,n_steps)
    H <- rep(NA,n_steps)
    L[1] <- init[1]
    H[1] <- init[2]
    for ( i in 2:n_steps ) {
        L[i] <- L[i-1] + dt*L[i-1]*( theta[3]*H[i-1] - theta[4] )
        H[i] <- H[i-1] + dt*H[i-1]*( theta[1] - theta[2]*L[i-1] )
    }
    return( cbind(L,H) )
}
```

And here are the priors I eventually settled on, with code to simulate and plot:

R code 16.30

```
N <- 16
theta <- matrix( NA , N , 4 )
theta[,1] <- rnorm( N , 1.5 , 1 )
theta[,2] <- rnorm( N , 0.005 , 0.1 )
theta[,3] <- rnorm( N , 0.0005 , 0.1 )
theta[,4] <- rnorm( N , 0.5 , 1 )

par(mfrow=c(4,4),cex=1.05)
par(mgp = c(1, 0.2, 0), mar = c(1, 2, 1, 1) + 0.1,
        tck = -0.02, cex.axis = 0.8, bty = "l" )

for ( i in 1:N ) {
    z <- sim_mites( n_steps=1e4 , init=as.numeric(Mites[1,3:2]) , theta=theta[i,] )
    # preds in blue, prey in black
    ymax <- max(z)
    if ( ymax == Inf ) ymax <- 1.79e+300
    plot( NULL , ylim=c(0,ymax) , xlim=c(0,1e4) , xaxt="n" , ylab="" , xlab="" )
```

```
    lines( z[,2] , col="black" , lwd=2 )
    lines( z[,1] , col=rangi2 , lwd=2 )
    mtext( "time" , 1 )
}
```



These priors can produce cycles, but they very often produce unregulated growth of either the prey or predator. Try repeating the the code above a few times, to get a sense of the range of behavior.

The subtle trick is going to be getting the time units in the data and implied by these parameters on the same scale. I'll do this by dividing the days by 7 to make them into fractions of a week. But you could also rescale the parameters.

Now we need the model. We need the lynx-hare model, but without the measurement error part. If you use the measurement error part, it'll work very similarly however. I'm going to display the model code here. I made a separate text file, named `Mites.stan`, with this model code, instead of storing it in an R character vector.

```
// Mites model
functions {
  real[] dpop_dt( real t,                 // time
                  real[] pop_init,        // initial state {lynx, hares}
                  real[] theta,           // parameters
                  real[] x_r, int[] x_i) {  // unused
    real L = pop_init[1];
    real H = pop_init[2];
    real bh = theta[1];
    real mh = theta[2];
    real ml = theta[3];
    real bl = theta[4];
    // differential equations
    real dH_dt = (bh - mh * L) * H;
    real dL_dt = (bl * H - ml) * L;
    return { dL_dt , dH_dt };
  }
}
```

```
data {
  int<lower=0> N;              // number of measurement times
  int<lower=0> mites[N,2];    // measured populations
  real<lower=0> days[N];      // days from start of experiment
}
parameters {
  real<lower=0> theta[4];      // { bh, mh, ml, bl }
  real<lower=0> pop_init[2];   // initial population state
  real<lower=0> sigma[2];      // measurement errors
}
transformed parameters {
  real pop[N, 2];
  pop[1,1] = pop_init[1];
  pop[1,2] = pop_init[2];
  pop[2:N,1:2] = integrate_ode_rk45(
    dpop_dt, pop_init, 0, days[2:N], theta,
    rep_array(0.0, 0), rep_array(0, 0),
    1e-5, 1e-3, 5e2);
}
model {
  // priors
  theta[1] ~ normal( 3*0.5 , 1 );       //bh
  theta[2] ~ normal( 0.01*0.5 , 0.1 );  //mh
  theta[3] ~ normal( 0.001*0.5 , 0.1 ); //bl
  theta[4] ~ normal( 1*0.5 , 1 );       //ml
  sigma ~ exponential( 1 );
  pop_init[1] ~ normal( mites[1,1] , 50 );
  pop_init[2] ~ normal( mites[1,2] , 50 );
  // observation model
  // connect latent population state to observed pelts
  for ( t in 1:N )
    for ( k in 1:2 )
      mites[t,k] ~ lognormal( log(pop[t,k]) , sigma[k] );
}
generated quantities {
  real mites_pred[N,2];
  for ( t in 1:N )
    for ( k in 1:2 )
      mites_pred[t,k] = lognormal_rng( log(pop[t,k]) , sigma[k] );
}
```

Now we can prepare the data and run the model:

R code 16.31

```
dat_mites <- list(
    N = nrow(Mites),
    mites = as.matrix( Mites[,3:2] ),
    days = Mites[,1]/7 )

m16H5.1 <- stan( file="Mites.stan" , data=dat_mites , chains=4 , cores=4 , iter=2000 ,
    control=list( adapt_delta=0.99 ) )

precis(m16H5.1,2)
```

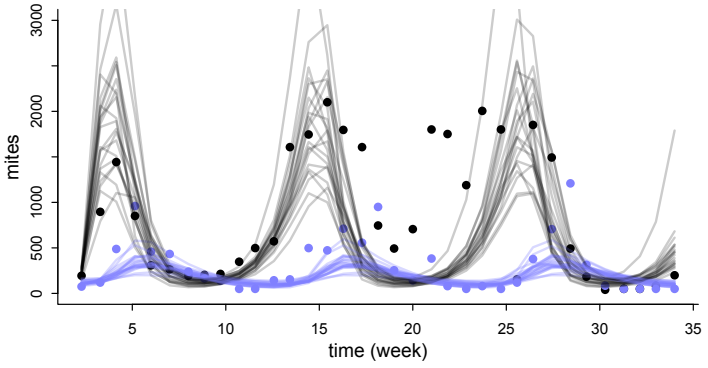
```
140 matrix parameters hidden. Use depth=3 to show them.
              mean    sd   5.5%  94.5% n_eff Rhat4
theta[1]      1.27  0.30   0.91   1.80   763     1
```

```
theta[2]      0.01  0.00   0.00   0.01   776     1
theta[3]      0.33  0.07   0.21   0.44   934     1
theta[4]      0.00  0.00   0.00   0.00  1224     1
pop_init[1] 115.90 19.08  87.85 149.01  1770     1
pop_init[2] 249.54 39.32 188.23 312.68  2235     1
sigma[1]      0.73  0.12   0.56   0.93  1507     1
sigma[2]      1.07  0.15   0.87   1.33  2500     1
```

Some of those parameters are on such a small scale, they look to be equal to zero. Let's look at the posterior predictions:

R code 16.32

```
post <- extract.samples(m16H5.1)

mites <- dat_mites$mites
plot( dat_mites$days , mites[,2] , pch=16 , ylim=c(0,3000) ,
    xlab="time (week)" , ylab="mites" )
points( dat_mites$days , mites[,1] , col=rangi2 , pch=16 )

for ( s in 1:21 ) {
    lines( dat_mites$days , post$pop[s,,2] , col=col.alpha("black",0.2) , lwd=2 )
    lines( dat_mites$days , post$pop[s,,1] , col=col.alpha(rangi2,0.3) , lwd=2 )
}
```



There are some clear problems here. The cycles don't look to be of the same width each time, and the model cannot handle that. And the predator cycles (blue) are not steep enough. Ecologists know that plain Lotka-Volterra models have trouble with realistic data. And these data are even an experiment.

Better models have stochasticity in them, such that stochasticity in birth/death processes propagates through the dynamics. These models are deterministic, and so in small populations and spatially divided populations (like these mite experiments—go read the original paper), they can have clear problems. The important thing is to understand the structural features of each model that produce different dynamics. Real biology is too complicated to ever get the model right. But models are still essential for their study.