STATISTICAL RETHINKING

PRACTICE PROBLEM

SOLUTIONS

Contents

Date: September 14, 2015.

1. 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 1.1

```
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 1.2

```
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 1.3

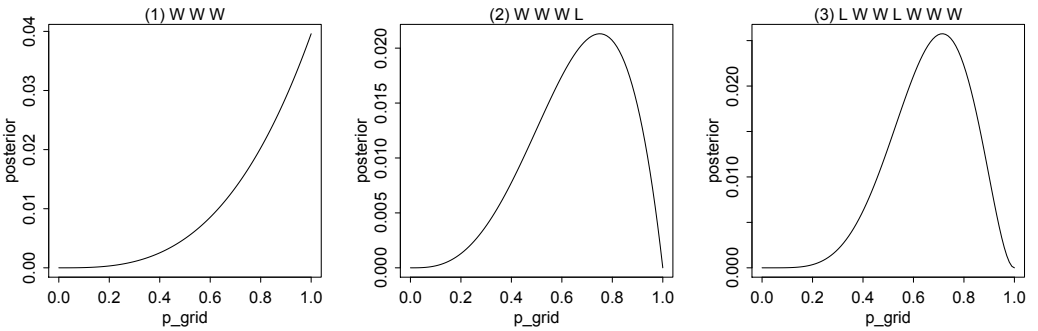
```
# likelihood of 3 water in 4 tosses
likelihood <- dbinom( 3 , size=4 , prob=p_grid )
```

And for (3):

R code 1.4

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

And this is what each plot should look like:

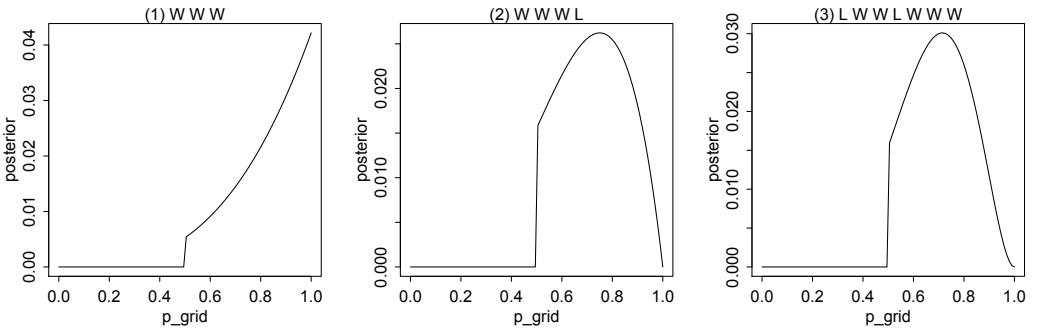


2M2. Only the prior has to be changed. For the first set of observations, W W W, this will complete the calculation and plot the result:

R code 1.5

```
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 1.6

```
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 $\Pr(\text{A}|\text{test A, twins, singleton}) \approx 0.56$.

2. Chapter 3 Solutions

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

R code 2.1
```
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 2.2
```
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 2.3
```
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 2.4
```
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 2.5
```
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 2.6
```
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 2.7

```
[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 2.8

```
      80%
0.7567568
```

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

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

R code 2.9

```
[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 2.10

```
    |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 2.11

```
      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 2.12

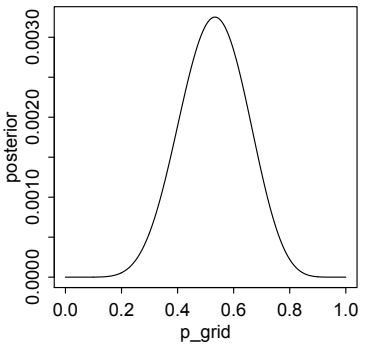
```
         width1    width2
0.66| 0.2642643 0.2682683
```

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

R code 2.13
```
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 2.14
```
plot( posterior ~ p_grid , type="l" )
```



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

R code 2.15
```
samples <- sample( p_grid , prob=posterior , size=1e4 , replace=TRUE )
```

And to compute the 90% HPDI:

R code 2.16
```
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 2.17
```
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 2.18

```
[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 2.19

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 2.20

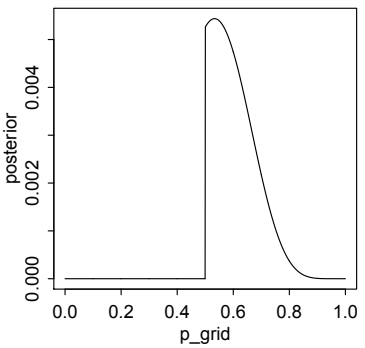
```
[1] 0.1802
```

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 2.21



The 90% HPDI will be a lot narrower now:

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

R code 2.22

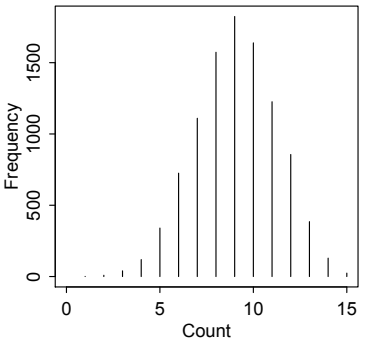
```
     |0.9      0.9|
0.5005005 0.7087087
```

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 2.23

```
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 2.24

```
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 2.25

```
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 2.26

```
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:

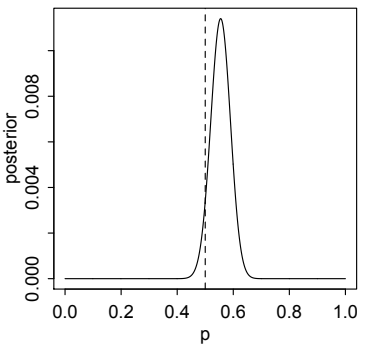
R code 2.27

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

This is what your posterior distribution should look like:

R code 2.28

```
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 2.29

```
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 2.30

```
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 2.31

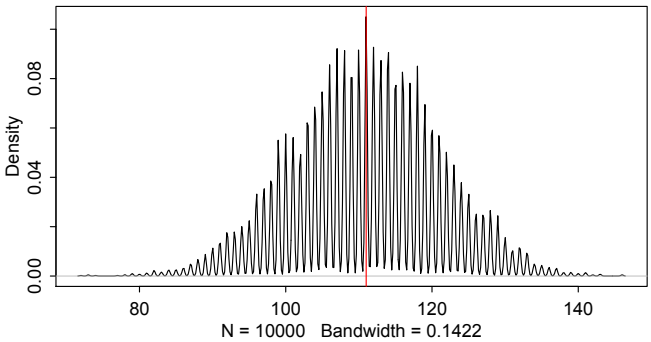
```
p.samples <- sample( p , size=10000 , replace=TRUE , prob=posterior )
bsim <- rbinom( 10000 , size=200 , prob=p.samples )
```

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 2.32

```
# 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:



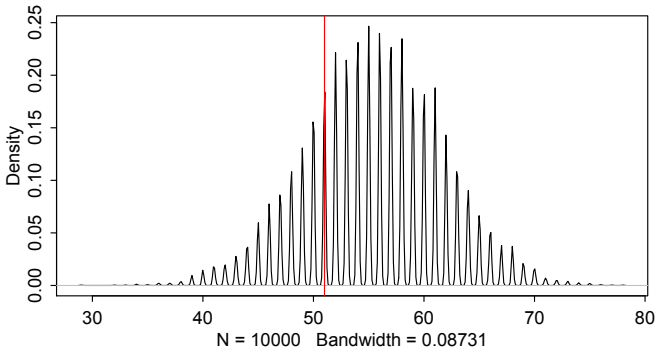
The black density is the simulated counts of male births, out of 200 maximum. The red vertical line is the observed count in the data. Looks like the model does a good job of predicting the data. In a sense, this isn't surprising, because the model was fit to this exact aspect of the data, the total 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 2.33

```
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.

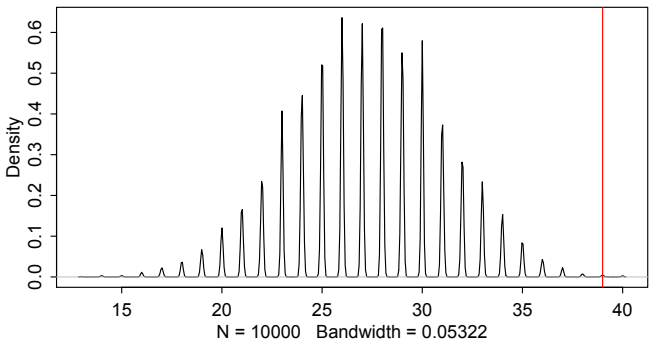
3H5. Now an even harder test of the model. How does it do predicting second births that follow girl births? The only tricky part of this is counting up boys born after girls. You can use R's subsetting to accomplish this:

R code 2.34

```
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?

3. 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:

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

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 of heights, 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 heights. This code will sample from the priors:

R code 3.1

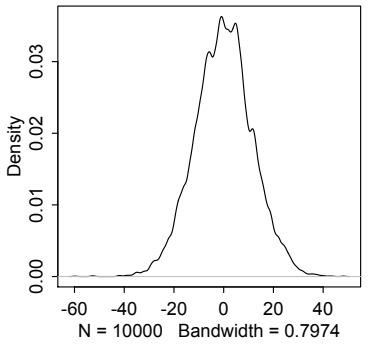
```
mu_prior <- rnorm( 1e4 , 0 , 10 )
sigma_prior <- runif( 1e4 , 0, 10 )
```

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

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

R code 3.2

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



Note that the prior distribution of heights is centered on zero. If that strikes you as odd, remember that a continuous variable like height can always be re-centered to any value you like, without changing any of the information in the data.

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

R code 3.3

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

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.

$$
\begin{aligned}
y_i &\sim \text{Normal}(\mu, \sigma) \\
\mu &= \alpha + \beta x_i \\
\alpha &\sim \text{Normal}(0, 50) \\
\beta &\sim \text{Normal}(0, 10) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

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

$$
\begin{aligned}
h_i &\sim \text{Normal}(\mu, \sigma) \\
\mu &= \alpha + \beta y_i \\
\alpha &\sim \text{Normal}(0, 100) \\
\beta &\sim \text{Normal}(0, 10) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

where h is height and y is year. These priors aren't great, but they'll do. The prior on the intercept α is effectively uninformative. The problem didn't say what scale height is measured on, so it's hard to do much else here. The prior on β is very weakly informative, centered on zero, which corresponds

to no impact of year. This doesn't seem like a great prior, because surely height increases each year or stays the same. So maybe a uniform prior above zero would be better?

$$
\begin{aligned}
h_i &\sim \text{Normal}(\mu, \sigma) \\
\mu &= \alpha + \beta y_i \\
\alpha &\sim \text{Normal}(0, 100) \\
\beta &\sim \text{Uniform}(0, 10) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

There are other options, including truncated distributions or log-Normal distributions, but the book hasn't introduced those yet.

4M5. This one is subtle. On the one hand, having information on measurement scale lets us set a more sensible prior for the intercept, α. You might center it on 120 now, for example. On the other hand, for the prior to really be "prior," it needs to be ignorance of the actual data values. Since the observed mean is a feature of the sample, not of the measurement scale, basing the prior on it could get you into trouble. What kind of trouble? It could lead to an illusionary fit to data, as essentially you've used the data twice: once in setting the prior and once in conditioning on the data (computing the posterior).

Much later in the book, you'll see how constraints on the data can be used to design models, without running the risk of using the data twice.

4M6. Again, this is a subtle issue. All the advice from just above applies. But it's not clear here whether "variance among heights for students of the same age is never more than 64cm" is a feature of the sample or of the population. If it's of the population, then setting a prior for it may be okay. In that case:

$$\sigma \sim \text{Uniform}(0, 64)$$

would make sense.

4H1. First, fit the model with `map`, to provide a quadratic approximation of the posterior. I'll use quite flat priors here, but if you used stronger priors, that's fine. What matters is the procedure be coherent.

R code 3.4

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

m <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight ,
        a ~ dnorm( 100 , 100 ) ,
        b ~ dnorm( 0 , 10 ) ,
        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 3.5

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

```
'data.frame': 10000 obs. of  3 variables:
 $ a    : num  114 115 114 115 113 ...
 $ b    : num  0.888 0.882 0.891 0.878 0.925 ...
 $ sigma: num  4.98 4.88 5.02 5.25 5.37 ...
```

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 3.6

```
y <- rnorm( 1e5 , post$a + post$b*46.95 , post$sigma )
mean(y)
HPDI(y,prob=0.90)
```

```
[1] 156.3685

    |0.9     0.9|
148.1271 164.7901
```

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 90% HPDI of height.

Do the above for each row in the table, and you should get numbers close to:

Individual	weight	expected height	90% confidence interval
1	46.95	156.3685	148.1271 – 164.7901
2	43.72	153.4653	144.7351 – 161.4463
3	64.78	172.5211	163.7667 – 180.8067
4	32.59	143.4007	135.0600 – 151.9068
5	54.63	163.3143	155.1877 – 171.9731

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 3.7

```
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 3.8

```
m <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight ,
        a ~ dnorm( 100 , 100 ),
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d3 )
precis(m)
```

```
       Mean StdDev  5.5% 94.5%
a     58.24   1.40 56.01 60.47
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 90% 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 3.9

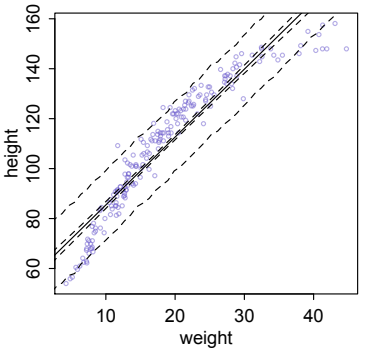
```
post <- extract.samples( m )
w.seq <- seq(from=1,to=45,length.out=50)
mu <- sapply( w.seq , function(z) mean( post$a + post$b*z ) )
mu.ci <- sapply( w.seq , function(z)
    HPDI( post$a + post$b*z , prob=0.9 ) )
pred.ci <- sapply( w.seq , function(z)
    HPDI( rnorm(10000,post$a + post$b*z,post$sigma) , 0.9 ) )
```

And then to plot everything:

R code 3.10

```
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 `map`:

R code 3.11

```
library(rethinking)
data(Howell1)
d <- Howell1
mlw <- map(
    alist(
        height ~ dnorm( mean=mu , sd=sigma ) ,
        mu <- a + b*log(weight) ,
        a ~ dnorm( 138 , 100 ) ,
        b ~ dnorm( 0 , 100 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d )
precis(mlw)
```

```
         Mean StdDev   5.5%  94.5%
a     -23.79   1.34 -25.93 -21.66
b      47.08   0.38  46.47  47.69
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 3.12

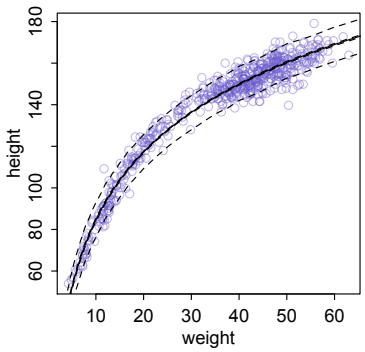
```
post <- extract.samples(mlw)
lw.seq <- seq(from=1.4,to=4.2,length.out=50)
mu <- sapply( lw.seq , function(z) mean( post$a+post$b*z ) )
mu.ci <- sapply( lw.seq , function(z) HPDI( post$a+post$b*z ) )
h.ci <- sapply( lw.seq , function(z)
    HPDI( rnorm(10000,post$a+post$b*z,post$sigma) ) )
```

Now you have lists of numbers that can be plotted to produce the confidence curves. But how to translate back to the non-log scale of measurement on the horizontal axis? Just exponentiate the x-axis coordinates in `lw.seq`, and the job is done:

R code 3.13

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

4. 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 4.1

```
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 )
```

A first regression model including this variable might be:

R code 4.2

```
m_5M4 <- map(
    alist(
        Divorce ~ dnorm(mu,sigma),
        mu <- a + bR*Marriage + bA*MedianAgeMarriage + bM*pct_LDS,
        a ~ dnorm(0,100),
        c(bA,bR,bM) ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ),
    data=d )
precis( m_5M4 )
```

```
       Mean StdDev  5.5% 94.5%
a     38.58   6.94 27.49 49.67
bA    -1.10   0.23 -1.46 -0.74
bR     0.00   0.08 -0.12  0.12
bM    -0.06   0.02 -0.10 -0.03
sigma  1.34   0.13  1.13  1.56
```

As expected, there is a negative association between percent LDS and divorce rate. This model assumes the relationship between divorce rate and percent LDF 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. The bivariate models are familiar to you:

R code 4.3

```
library(rethinking)
data(foxes)
d <- foxes
m1 <- map(
    alist(
        weight ~ dnorm( mu , sigma ) ,
        mu <- a + ba*area ,
        a ~ dnorm(0,100),
        ba ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) , data=d )
m2 <- map(
    alist(
        weight ~ dnorm( mu , sigma ) ,
        mu <- a + bg*groupsize ,
        a ~ dnorm(0,100),
        bg ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) , data=d )
precis(m1)
```

```
precis(m2)
```

```
      Mean StdDev  5.5% 94.5%
a     4.45   0.39  3.83  5.08
ba    0.02   0.12 -0.16  0.21
sigma 1.18   0.08  1.06  1.30

       Mean StdDev  5.5% 94.5%
a      5.07   0.32  4.55  5.59
bg    -0.12   0.07 -0.24 -0.01
sigma  1.16   0.08  1.04  1.29
```

From just these estimates, seems like area is rather unimportant. It's MAP is close to zero and its 89% interval covers a lot of territory on both sides of zero. Group size seems to have a weak negative effect on body weight.

As usual, it's easier to appreciate the importance of the estimates when you plot them against the data. First, the code for `m1`:

R code 4.4

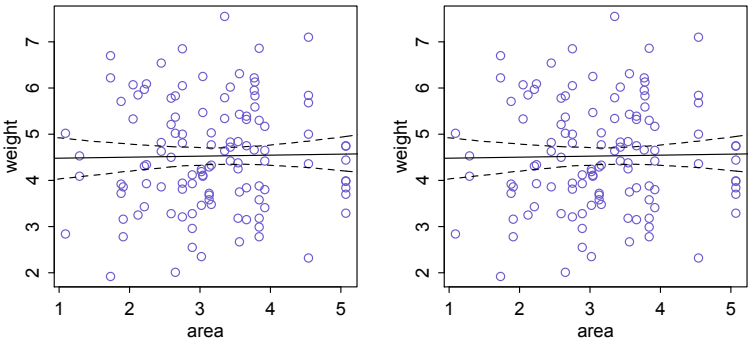
```
x.seq <- seq(from=0,to=6,by=0.025)
mu <- link( m1 , data=list(area=x.seq) )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
plot( weight ~ area , data=d , col="slateblue" )
lines( x.seq , mu.mean )
lines( x.seq , mu.ci[1,] , lty=2 )
lines( x.seq , mu.ci[2,] , lty=2 )
```

And now the very similar code for `m2`:

R code 4.5

```
x.seq <- seq(from=1,to=9,by=0.5)
mu <- link( m2 , data=list(groupsize=x.seq) )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
plot( weight ~ groupsize , data=d , col="slateblue" )
lines( x.seq , mu.mean )
lines( x.seq , mu.ci[1,] , lty=2 )
lines( x.seq , mu.ci[2,] , lty=2 )
```

And the plots look like this:



So weight on area is essentially a horizontal line—no trend at all of any importance—and the weight on group size effect is on the order of one unit of weight for a change in group size from 2 to 8 animals. That's a very modest effect, compared to the range of variation in weight within the data as a whole.

5H2. Here's the code for the multiple regression:

R code 4.6

```
m3 <- map(
    alist(
        weight ~ dnorm( mu , sigma ) ,
        mu <- a + ba*area + bg*groupsize ,
        a ~ dnorm(0,100),
        c(ba,bg) ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) , data=d )
precis(m3)
```

```
       Mean StdDev  5.5% 94.5%
a      4.45   0.37  3.86  5.04
ba     0.62   0.20  0.30  0.94
bg    -0.43   0.12 -0.63 -0.24
sigma  1.12   0.07  1.00  1.24
```

From the estimates alone, you can see that the importance of each predictor is increased, now. The MAP is further from zero, and the intervals in both cases are now reliably on one side of zero.

How big are the effects, relative to the range of variation in the data? Plots will help answer that. Here is the code for plotting the effect of area, holding groupsize constant at the mean:

R code 4.7

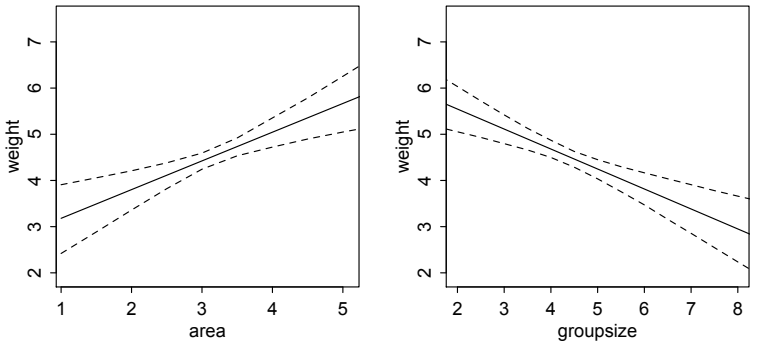
```
area.seq <- seq(from=1,to=6,by=0.5)
pred.dat <- data.frame( area=area.seq , groupsize=mean(d$groupsize) )
mu <- link( m3 , data=pred.dat )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
plot( weight ~ area , data=d , type="n" )
lines( area.seq , mu.mean )
lines( area.seq , mu.ci[1,] , lty=2 )
lines( area.seq , mu.ci[2,] , lty=2 )
```

And here is the code for the analogous plot for groupsize, holding area constant at the mean:

R code 4.8

```
gs.seq <- seq(from=1,to=9,by=0.5)
pred.dat <- data.frame( area=mean(d$area) , groupsize=gs.seq )
mu <- link( m3 , data=pred.dat )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
plot( weight ~ groupsize , data=d , type="n" )
lines( gs.seq , mu.mean )
lines( gs.seq , mu.ci[1,] , lty=2 )
lines( gs.seq , mu.ci[2,] , lty=2 )
```

And here are the plots:



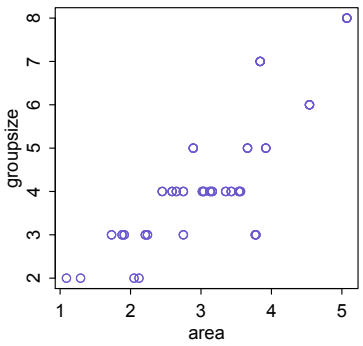
So from the smallest area territory to the largest, there is a change from 3kg to 6kg in expected weight. That's a lot. For a change from the smallest groups (2 animals) to the largest (8 animals), we expect a change in body weight from about 6kg down to 3kg. Again that's a big effect, given that body weight only varies between 2kg and 7kg in the entire data.

Remember, I've blanked out the data points here, because these are like experiments done with the model fit, to see how the predictions respond to changing one predictor, holding the other constant. Since in the real data, the predictors are correlated, you can't easily plot the predictions against the raw data.

You get different results with the multivariate model, because the predictor variables area and groupsize are correlated with one another but have opposite correlations with the outcome. Area is positively correlated with weight: more area means more resources means heavier foxes. Group size is negatively correlated with weight: bigger groups for a given territory size means less food per animal, means lighter foxes. But since in the natural fox groups, area and group size are correlated—more foxes go to the bigger territories, as you might expect from an ideal free perspective. You can see for yourself by plotting the two variables against one another:

R code 4.9

```
plot( groupsize ~ area , data=d , col="slateblue")
```



But since area and groupsize are not perfectly correlated, the multiple regression can focus on those groups that are most informative about the countervailing effects of each predictor. For example, those groups that have few animals for their area (you can see some in the plot above) help us figure out whether reducing group size leads to heavier foxes or not. The regression automatically compares such groups to groups with similar sized territories (similar area) but with higher densities of animals. If the animals in the smaller groups are heavier on average, that informs the estimate of the coefficient for groupsize.

It would of course be better to have a direct experiment. If you could experimentally alter group sizes, leaving territory sizes unchanged (might be hard, if more animals are needed to defend territory size!), then you could make stronger inferences about the direct causal effect of density on body weight.

5H3. First, fitting the new models:

R code 4.10

```
m4 <- map(
    alist(
        weight ~ dnorm( mu , sigma ) ,
        mu <- a + bf*avgfood + bg*groupsize ,
        a ~ dnorm(0,100),
        c(bf,bg) ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) , data=d )
m5 <- map(
    alist(
        weight ~ dnorm( mu , sigma ) ,
        mu <- a + bf*avgfood + bg*groupsize + ba*area ,
        a ~ dnorm(0,100),
        c(bf,bg,ba) ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) , data=d )
```

(a) Start by looking at `m4`:

R code 4.11

```
precis(m4)
```

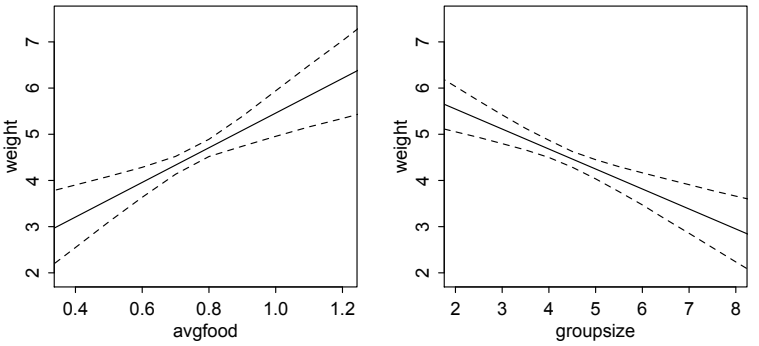
```
       Mean StdDev  5.5% 94.5%
a      4.14   0.43  3.45  4.83
bf     3.77   1.20  1.85  5.70
bg    -0.56   0.16 -0.81 -0.31
sigma  1.12   0.07  1.00  1.23
```

The estimate for avgfood in `m4` is pretty far from zero, and its 89% interval is reliably above zero too. So it seems avgfood is a good predictor, at least combined with groupsize. Is it more strongly related to body weight than area is? Hard to tell from the coefficient, because the range of avgfood is different than the range of area. Again, a plot helps.

R code 4.12

```
food.seq <- seq(from=0,to=1.5,by=0.1)
pred.dat <- data.frame( avgfood=food.seq , groupsize=mean(d$groupsize) )
mu <- link( m4 , data=pred.dat )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
plot( weight ~ avgfood , data=d , type="n" )
lines( food.seq , mu.mean )
lines( food.seq , mu.ci[1,] , lty=2 )
lines( food.seq , mu.ci[2,] , lty=2 )
```

I'll show this new plot next to the analogous plot from before, the one produced for area:



The estimated effect of avgfood is a little stronger than area, but not by much. Especially on the high end (the right side of the plot), the expected body weight goes above 6kg for the extreme values of avgfood, whereas the plot for area never quite gets that high. This comparison suggests avgfood is a little better as a predictor.

A more principled way to compare the predictive value of the two variables (still consider only models that have one or the other, not both!) is to compare some measure of fit. Lots of people use R^2, which is appropriate for Gaussian models, at least most of the time. A measure that works for any likelihood-based model is negative log-likelihood. You can extract the log-likelihood of each model to compare them:

R code 4.13

```
-logLik(m3)
-logLik(m4)
```

```
'log Lik.' 177.5824 (df=4)
'log Lik.' 177.3915 (df=4)
```

Smaller negative log-likelihood indicates a better fit. So model `m4` fits better, by a small amount. This suggests avgfood is a better predictor, consistent with the intuition arrived at from inspecting the plots above. But in Chapter 6, you'll see why fit measured this way can be misleading. In this case, it isn't, but you'll have to wait to verify that.

(b) But it would be nice to include both in a model and see which comes out on top. That's what model `m5` does.

R code 4.14

```
precis(m5)
```

```
       Mean StdDev  5.5% 94.5%
a      4.07   0.43  3.39  4.76
bf     2.46   1.44  0.16  4.75
bg    -0.60   0.16 -0.85 -0.35
ba     0.39   0.24  0.01  0.77
sigma  1.10   0.07  0.99  1.22
```

Is neither avgfood nor area important? While they both still have positive MAP estimates, their intervals are now spread much more widely.

This is a case of multicollinearity, although not an extreme one. Take a look at the scatter of avgfood against area (do this yourself). Bigger territories have more food in them. That accounts for the high correlation here. That in turn accounts for why adding both variables results in inflated standard errors for both.

Now why would avgfood be a slightly better predictor? Hard to know for sure, but it stands to reason that avgfood is the actual resource that affects body weight, and so it is a more reliable predictor.

If you think about it, area should only affect fox weight through food availability. To really test that speculation, we'd need some way to manipulate food availability in a habitat, keeping area the same. Then we could break up that correlation a bit and make stronger causal inferences.

But it might turn out that area does have some direct effect on fox weight, maybe because the larger territories are not really like the smaller ones. They could be in different places (on edges of towns, perhaps), and therefore introduce different factors, like different ecological communities and different disease risk.

5. Chapter 6 Solutions

6E1. 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.

6E2. This is a simple calculation in R, very much like the example on page 192:

R code 5.1

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

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

```
[1] 0.6108643
```

6E3. Similar to the problem above, but now with four types of events:

R code 5.2

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

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

```
[1] 1.376227
```

6E4. When events are impossible (have probability of zero), they just fall out of the calculation:

R code 5.3

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

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

```
[1] 1.098612
```

6M1. 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. DIC is instead:

$$\begin{aligned} \text{DIC} &= \bar{D} + (\bar{D} - \hat{D}) \\ &= \bar{D} + p_D \\ &= \hat{D} + 2p_D \end{aligned}$$

where $\bar{D}$ is the deviance on the training data, averaging over the posterior distribution of the parameters, and $\hat{D}$ is the deviance on the training data, plugging in the MAP estimates. Finally, there is WAIC. 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. To really grasp what WAIC means, it may be necessary to see the calculations in the Overthinking box on pages 206 and 207.

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 DIC, the fit term can be expressed either as $\bar{D}$ (the average deviance) or $\hat{D}$ (which is just D_{train} again). The penalty term is either the difference $\bar{D} - \hat{D}$ or rather twice that difference. See the alternative arrangements presented above. But in either case, the flexibility of the model is measured as the difference between the average deviance and the deviance at the posterior mean. 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 most general to least general: WAIC, DIC, AIC. When the posterior predictive mean is a good representation of the posterior predictive distribution, DIC and WAIC will tend to agree. When priors are effectively flat or overwhelmed by the amount of data, the DIC and AIC will tend to agree.

6M2. 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.

6M3. 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.

6M4. 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, as measured by both DIC and WAIC.

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 5.4

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

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

6M5. 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.

6M6. 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.

6H1. Let's fit each model. First, I'll define each formula list:

R code 5.5

```
f1 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age,
    c(a,b1) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
)
f2 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age + b2*age^2,
    c(a,b1,b2) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
```

```
)
f3 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age + b2*age^2 + b3*age^3,
    c(a,b1,b2,b3) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
)
f4 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age + b2*age^2 + b3*age^3 + b4*age^4,
    c(a,b1,b2,b3,b4) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
)
f5 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age + b2*age^2 + b3*age^3 + b4*age^4 + b5*age^5,
    c(a,b1,b2,b3,b4,b5) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
)
f6 <- alist(
    height ~ dnorm(mu,sigma),
    mu <- a + b1*age + b2*age^2 + b3*age^3 + b4*age^4 + b5*age^5 +
        b6*age^6,
    c(a,b1,b2,b3,b4,b5,b6) ~ dnorm(0,100),
    sigma ~ dunif(0,50)
)
```

You can fit these models without starting values, but you might have to try a few times. The reason is that with these very flat priors, a random starting value from the prior might be very far from the MAP. So now let's store some sensible starting values for `a` and `sigma`, since we'll use them over and over again:

R code 5.6

```
# compute starting values for a and sigma
a.start <- mean(d1$height)
sigma.start <- sd(d1$height)
```

Now to fit the models:

R code 5.7

```
# fit models
m1 <- map( f1 , data=d1 ,
    start=list(a=a.start,sigma=sigma.start,b1=0) )
m2 <- map( f2 , data=d1 ,
    start=list(a=a.start,sigma=sigma.start,b1=0,b2=0) )
m3 <- map( f3 , data=d1 ,
    start=list(a=a.start,sigma=sigma.start,b1=0,b2=0,
        b3=0) )
m4 <- map( f4 , data=d1 ,
    start=list(a=a.start,sigma=sigma.start,b1=0,b2=0,
        b3=0,b4=0) )
m5 <- map( f5 , data=d1 ,
    start=list(a=a.start,sigma=sigma.start,b1=0,b2=0,
        b3=0,b4=0,b5=0) )
m6 <- map( f6 , data=d1 ,
```

```
    start=list(a=a.start,sigma=sigma.start,b1=0,b2=0,
        b3=0,b4=0,b5=0,b6=0) )
```

And the easy way to get the WAIC values and weights for each model is to use `compare`:

R code 5.8

```
compare(m1,m2,m3,m4,m5,m6)
```

```
     WAIC pWAIC dWAIC weight    SE   dSE
m4 1926.1   5.7   0.0   0.58 25.51    NA
m5 1927.7   6.6   1.6   0.26 25.50  1.13
m6 1928.6   7.8   2.5   0.16 25.00  3.12
m3 1952.3   5.4  26.2   0.00 24.21 10.87
m2 2150.0   5.2 223.8   0.00 22.62 26.80
m1 2395.3   3.3 469.2   0.00 22.94 31.12
```

Your table may be slightly different, as WAIC is computed from samples, and so there is sampling variation. For example, `m5` and `m6` might swap positions. You can take more samples with `compare` by using the optional `n` argument. This will reduce sampling variation. See `?compare`.

So what's going on here? The models are sorted by WAIC, in ascending order. The order is ascending, from low to high values, because smaller WAIC values are better. They indicate smaller expected out-of-sample error (smaller expected KL-divergence).

The top three models are `m4`, `m5`, and `m6`. These models have quite similar WAIC values, and these three are far better than the bottom three models. You can see this most clearly by looking at the `dWAIC` column on the far right end of the table above. The top three models are all within 2 or 3 units of deviance of one another, while the next best model is 25 units away. From there, it gets even worse. So, we can be confident at least that the top three models should do a good job, out of sample.

The top three models also get nearly all of the Akaike weight. The top model gets about half the weight, suggesting about a half chance that it'll be best out of sample. The next two models divide the weight about equally between them, each having around a quarter of the weight. Note that in interpreting these values, I'm not being anal about the exact numbers. This is because WAIC is an estimate of the *average* deviance out of sample, but there can be substantial variation around these averages. So it won't pay to be too strict about small differences in weight.

You can get a sense of this from the standard error columns in the table. Notice that the standard error of the differences (last column) do lead the top three models to overlap a lot. That is, while `m4` is clearly the best in expectation, there's reason not to bet your life savings on its doing best in any particular test sample.

6H2. Here's the code to compute for `m4`, the best model, and then plot it.

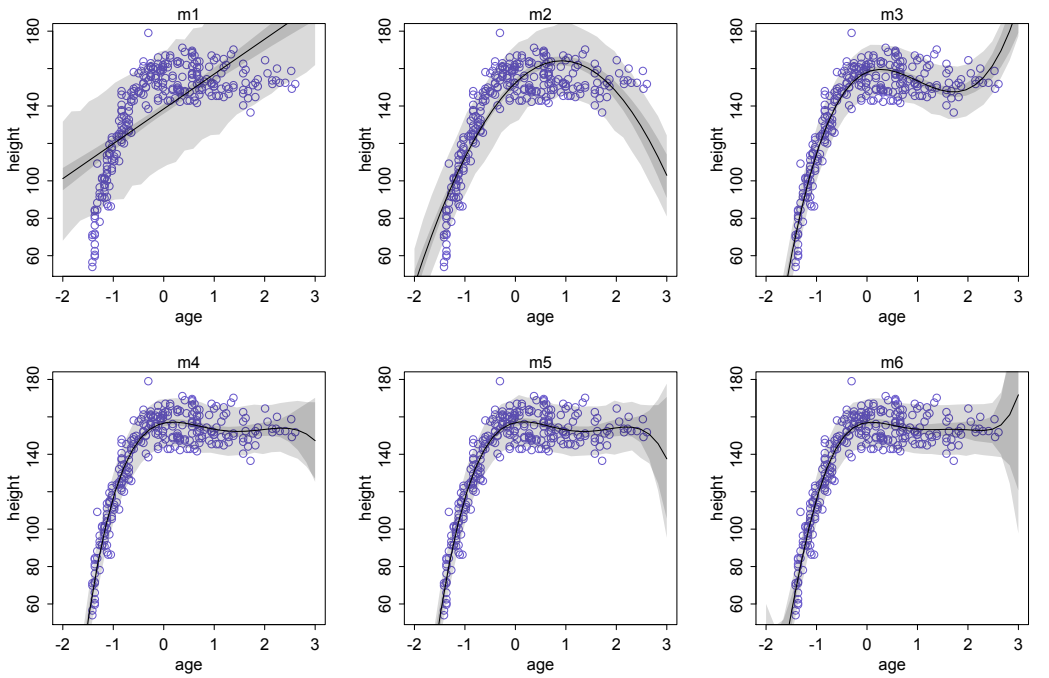
R code 5.9

```
# select out model to plot
m_plot <- m4

# define sequence of ages to compute values over
age.seq <- seq( from=-2 , to=3 , length.out=30 )

# compute posterior predictions for mu
mu <- link( m_plot , data=list(age=age.seq) )

# compute average prediction
mu.mean <- apply( mu , 2 , mean )
```

```
# compute 97% interval of average
mu.ci <- apply( mu , 2 , PI , prob=0.97 )

# compute interval of height
h <- sim( m_plot , data=list(age=age.seq) )
height.ci <- apply( h , 2 , PI )

# plot it all
plot( height ~ age , d1 , col="slateblue" , xlim=c(-2,3) )
lines( age.seq , mu.mean )
shade( mu.ci , age.seq )
shade( height.ci , age.seq )
```

The only thing to note here, as it differs from the text, is that I've extended the x-axis range, `xlim`, of the plot. I did this to show a little bit of the curve beyond the data range. Each plot for each model can be produced by just changing the line that defines `m_plot`. The functions `link` and `sim` are really doing the hard work here, extracting the model definitions from the fit model object and chugging through the posterior predictions for you.

Here are the plots for all six models, each labeled on top:



These predictive distributions reflect the information criteria quite well: the top three models, `m4` through `m6`, are barely different in prediction; the other three models are very different from one another and the top three. Among the top three models, there are very minor differences, particular beyond the range of the observed data. The most complex model, `m6`, turns wildly both just before and after the observed range of `age`. It's a very flexible model, and this kind of wild prediction at the extremes is a common problem with polynomial regressions. It is often called *Runge's phenomenon*, after Carl Runge (1856–1927), a German mathematician and scientist who described it. It's a well-understood reason for why complex functions can perform worse than simpler ones. In fitting the

sample, the fancy polynomial pays no cost for its flexibility. But in real prediction on new data, these wild swings at the extremes may ruin it.

6H3. For the model-averaged predictions, it's almost the same procedure. We just use `ensemble` instead of `link` and `sim`:

R code 5.10

```
h.ensemble <- ensemble( m4,m5,m6 , data=list(age=age.seq) )
```

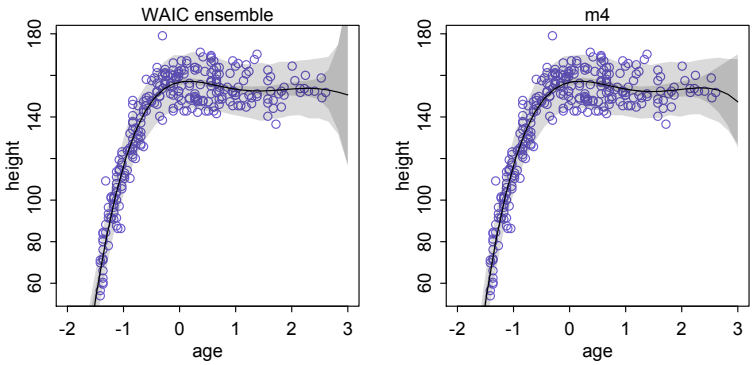
I included only the top three models, because none of the other models have even 1% of the model weight, so including them wouldn't produce any samples from them anyway. The result, `h.ensemble`, is a list containing a `link` matrix with samples for `mu` and a `sim` matrix with samples for `height`.

Here's all of the needed code, together:

R code 5.11

```
h.ensemble <- ensemble( m4,m5,m6 , data=list(age=age.seq) )

mu.mean <- apply( h.ensemble$link , 2 , mean )
mu.ci <- apply( h.ensemble$link , 2 , PI )
height.ci <- apply( h.ensemble$sim , 2 , PI )

plot( height ~ age , d1 , col="slateblue" , xlim=c(-2,3) )
lines( age.seq , mu.mean )
shade( mu.ci , age.seq )
shade( height.ci , age.seq )
```

And the here's the result, accompanied by the previous plot for ease of comparison:



The one on the left is the model averaged predictions. The one on the right is the predictions for `m4` only. There isn't much difference! Until you get outside the range of the data, that is. Then the model averaged predictions swing more wildly, on the far right edge. This is because these predictions include about a 25% influence of `m6`, which is capable of swinging around very rapidly. So out where there is no data to tame it, it's free to swing.

Keep in mind that with a smaller sample, the differences among the models and the ensemble would be more pronounced.

6H4. Computing each deviance, out of sample, is just a matter of extracting MAP estimates, plugging them into each linear model, and then summing log likelihoods over the heights in `d2`. You could also compute lppd, the fully Bayesian "deviance" that WAIC uses, but that would be harder. Using the

ordinary deviance will serve to teach the lesson, without making the calculations too difficult. Here's the code to do it for each model:

R code 5.12

```
k <- coef(m1)
mu <- k['a'] + k['b1']*d2$age
dev.m1 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )

k <- coef(m2)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2
dev.m2 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )

k <- coef(m3)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2 + k['b3']*d2$age^3
dev.m3 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )

k <- coef(m4)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2 + k['b3']*d2$age^3 +
    k['b4']*d2$age^4
dev.m4 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )

k <- coef(m5)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2 + k['b3']*d2$age^3 +
    k['b4']*d2$age^4 + k['b5']*d2$age^5
dev.m5 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )

k <- coef(m6)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2 + k['b3']*d2$age^3 +
    k['b4']*d2$age^4 + k['b5']*d2$age^5 + k['b6']*d2$age^6
dev.m6 <- (-2)*sum( dnorm( d2$height , mu , k['sigma'] , log=TRUE ) )
```

You end up with each out-of-sample deviance in the symbols `dev.m1` through `dev.m6`.

6H5. Here I've organized the out-of-sample deviance values in a table with the WAIC values (sorted by out-of-sample deviance, ascending):

R code 5.13

```
compare.tab <- compare(m1,m2,m3,m4,m5,m6,sort=FALSE)
tab <- data.frame(
        WAIC=compare.tab@output$WAIC,
        dev_out=c(dev.m1,dev.m2,dev.m3,dev.m4,dev.m5,dev.m6)
    )
rownames(tab) <- rownames(compare.tab@output)

# display, sorted by dev out
tab[ order(tab$dev_out) , ]
```

```
       WAIC  dev_out
m4 1926.586 1876.568
m6 1928.309 1877.628
m5 1927.964 1878.417
m3 1952.805 1932.251
m2 2149.982 2137.511
m1 2395.373 2422.093
```

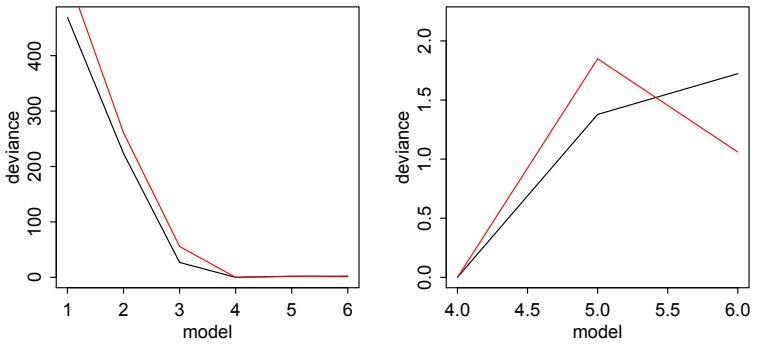
The order for WAIC and the actual out-of-sample deviance is nearly the same.

How about the relative differences? Here I'll subtract the smallest WAIC from each WAIC value and plot them all. Then I'll do the same for the deviance values, plotting them also.

R code 5.14

```
dWAIC <- tab$WAIC - min(tab$WAIC)
ddev_out <- tab$dev_out - min(tab$dev_out)

plot( 1:6 , dWAIC , type="l" , xlab="model" , ylab="deviance" )
lines( 1:6 , ddev_out , col="red" )

plot( 4:6 , dWAIC[4:6] , type="l" , xlab="model" ,
    ylab="deviance" , ylim=c(0,2.2) )
lines( 4:6 , ddev_out[4:6] , col="red" )
```

This is what you get:



The left plot shows all six models. The black line is WAIC. The red line is out-of-sample deviance. The righthand plot zooms in on the best three models, so you can see differences between them. Notice the tiny vertical scale in the righthand plot.

So WAIC did a reasonable job, but it also switched the orders of `m5` and `m6`. Note however that the weights for those models were very similar, so we shouldn't have been too confident about which would be better in any event, and we should have expected them to perform very similarly as well. Keep the scale on the vertical axis in mind: the difference between the out-of-sample accuracy of `m5` and `m6` is tiny.

6H6. The point of this problem is to drive home how regularization helps with out-of-sample prediction. Here's how we can fit this model:

R code 5.15

```
f <- alist(
    height ~ dnorm( mu , sigma ),
    mu <- a + b1*age + b2*age^2 + b3*age^3 + b4*age^4 +
         b5*age^5 + b6*age^6,
    c(b1,b2,b3,b4,b5,b6) ~ dnorm( 0 , 5 )
)

m <- map( f , data=d1 ,
    start=list(
        a=mean(d1$height),
        b1=0,b2=0,b3=0,b4=0,b5=0,b6=0,
        sigma=sd(d1$height)
```

```
    ) )
```

I pulled a trick here with `map` of using a vector of parameter names to specify the same prior for every beta-coefficient at once. If you used six lines of `dnorm(0,5)`, that's fine. But I thought I'd show off a little of the optional features here.

Here are the MAP estimates:

R code 5.16

```
precis(m)
```

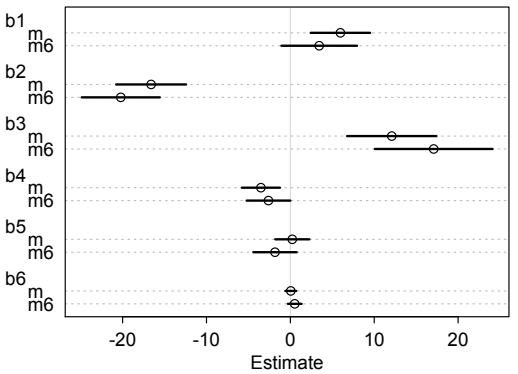
```
         Mean StdDev   5.5%  94.5%
a      155.86   0.88 154.46 157.26
b1       5.97   1.81   3.08   8.86
b2     -16.60   2.13 -20.01 -13.20
b3      12.09   2.72   7.75  16.44
b4      -3.51   1.16  -5.37  -1.65
b5       0.23   1.04  -1.43   1.89
b6       0.05   0.33  -0.48   0.58
sigma    8.20   0.36   7.63   8.77
```

What do these values mean? Not much, unless you can compute a sixth-order polynomial in your head. I cannot. It may be helpful to compare to the previous model `m6`, which used flat priors:

R code 5.17

```
plot( coeftab(m6,m) , pars=c("b1","b2","b3","b4","b5","b6") )
```

This is what you get:



I've plotted only the beta coefficients. Notice that in most cases the posterior distribution in model `m`, the regularized model, is closer to zero than in model `m6`. The regularizing priors move estimates towards zero. But as some parameters get closer to zero, others like `b1`, get farther from zero in an attempt to pick up the work that the other parameters were doing. So not every parameter in `m` is closer to zero. But most of them are.

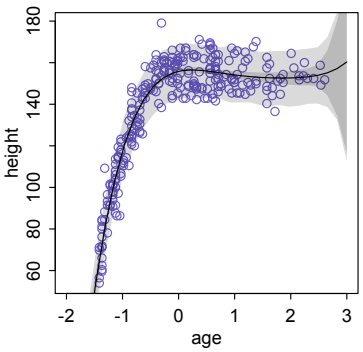
Now let's plot the implied posterior predictions.

R code 5.18

```
age.seq <- seq( from=-2 , to=3 , length.out=30 )
mu <- link( m , data=list(age=age.seq) )
h <- sim( m , data=list(age=age.seq) )

mu.mean <- apply( mu , 2 , mean)
mu.ci <- apply( mu , 2 , PI )
```

```
height.ci <- apply( h , 2 , PI )

plot( height ~ age , d1 , col="slateblue" , xlim=c(-2,3) )
lines( age.seq , mu.mean )
shade( mu.ci , age.seq )
shade( height.ci , age.seq )
```

This is the result:



Haven't we seen this plot already? Well, it's very similar to the previous plots, because it fits the data well. Being a pure sixth-degree polynomial, it does go more wild on the right edge. But otherwise very similar.

Now to compute the out-of-sample deviance. You've done this already. With slight code modifications:

R code 5.19

```
k <- coef(m)
mu <- k['a'] + k['b1']*d2$age + k['b2']*d2$age^2 + k['b3']*d2$age^3 +
    k['b4']*d2$age^4 + k['b5']*d2$age^5 + k['b6']*d2$age^6
dev <- (-2)*sum(
    dnorm(
        d2$height ,
        mean=mu ,
        sd=k['sigma'] ,
        log=TRUE ) )
dev
```

```
[1] 1875.433
```

To make it easier to compare this value to the previous models', I'll repeat the table from earlier, and tack our new regularized model onto the top of it, in the first row:

R code 5.20

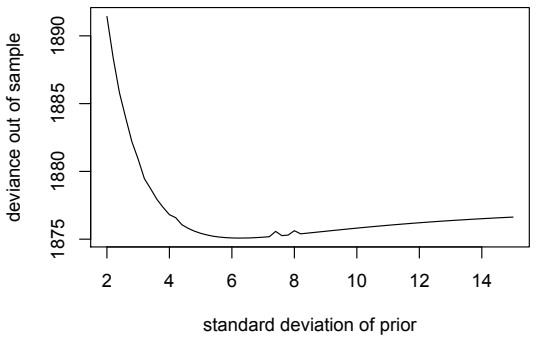
```
new_tab <- rbind( c(WAIC(m),dev) , tab )
rownames(new_tab)[1] <- "m"
new_tab[ order(new_tab$dev_out) , ]
```

```
       WAIC  dev_out
m  1930.978 1875.433
m4 1926.586 1876.568
m6 1928.309 1877.628
m5 1927.964 1878.417
m3 1952.805 1932.251
```

```
m2 2149.982 2137.511
m1 2395.373 2422.093
```

WAIC for the new model is worse than `m4`, but notice that it actually has a slightly better out-of-sample deviance than `m4`.

Note however that I just gave you the value 5 for the standard deviation of the prior. You might try other values and see how results vary. I did this already, and here's a plot showing the result of varying the standard deviation in the prior from 1 to 15, in increments of 0.2:



The best value is right around 6, actually. But there's a broad flat region in which many values are just about equally good. Smaller values get terrible quickly, because they prevent the model from learning enough. Larger values slowly get worse, as they let the model learn too much.

This cross-validation method of choosing a prior is quite common. It's common in machine learning as well, but there it's not usually described as choosing a prior. In ridge regression, for example, you'd be choosing instead a value for λ, which is a control parameter that governs regularization. But mathematically, it's equivalent to choosing the standard deviation of the prior.

Those little wiggles in the plot are just due to slightly imperfect convergence of the hill climbing algorithm that `map` uses by default. They could be cleared out with a little tweaking of some optional settings in `map`.

6. Chapter 7 Solutions

7E1. 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.).

7E2. 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.

7E3.

(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), $\mu_i = \alpha + \beta_S S_i + \beta_M M_i$.

7M1. 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.

7M2. 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.

7M3. 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.

7H1. Let's begin by loading the data and inspecting the `bed` variable.

R code 6.1

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

```
 [1] a a a a a a a a b b b b b b b b 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 6.2

```
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 `water` and `shade` variables, just like in the chapter. I'm not going to standardize the outcome, `blooms`, although that would make fitting this model a lot simpler. So instead I'll show you again how to use the `start` list to specify sensible start values. If we provided better priors, this wouldn't be necessary.

R code 6.3

```
d$water.c <- d$water - mean(d$water)
d$shade.c <- d$shade - mean(d$shade)
m1 <- map(
    alist(
        blooms ~ dnorm(mu,sigma),
        mu <- a + bW*water.c + bS*shade.c +
              bWS*water.c*shade.c +
              b_bed_b*bed_b + b_bed_c*bed_c ,
        a ~ dnorm(0,100),
```

```
        c(bW,bS,bWS,b_bed_b,b_bed_c) ~ dnorm(0,100),
        sigma ~ dunif(0,100)
    ),
    start=list(a=130,bW=0,bS=0,bWS=0,b_bed_b=0,b_bed_c=0,sigma=90),
    data=d )
precis(m1)
```

```
           Mean StdDev   5.5%  94.5%
a         97.34  12.74  76.98 117.70
bW        75.17   9.20  60.47  89.86
bS       -41.25   9.19 -55.95 -26.56
bWS      -52.23  11.24 -70.19 -34.27
b_bed_b   44.42  18.02  15.63  73.22
b_bed_c   49.03  18.02  20.24  77.83
sigma     39.17   5.33  30.65  47.69
```

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 6.4

```
( 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 6.5

```
m2 <- map(
    alist(
        blooms ~ dnorm(mu,sigma),
        mu <- a[bed_idx] +
              bW*water.c + bS*shade.c +
              bWS*water.c*shade.c ,
        a[bed_idx] ~ dnorm(130,100),
        c(bW,bS,bWS) ~ dnorm(0,100),
        sigma ~ dunif(0,100)
    ),
    start=list(a=rep(130,3),bW=0,bS=0,bWS=0,sigma=90),
    data=d )
precis(m2,depth=2)
```

```
          Mean StdDev   5.5%  94.5%
a[1]     97.67  12.95  76.97 118.37
a[2]    142.37  12.95 121.67 163.07
a[3]    146.99  12.95 126.29 167.69
bW       75.16   9.20  60.46  89.86
bS      -41.25   9.20 -55.95 -26.55
bWS     -52.18  11.24 -70.14 -34.21
sigma    39.18   5.33  30.66  47.71
```

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 6.6

```
post <- extract.samples(m2)
diff_b_c <- post$a[,2] - post$a[,3]
HPDI( diff_b_c )
```

```
    |0.89     0.89|
-34.55393  24.53934
```

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.

7H2. Now let's compare the model from the previous problem with the interaction model from the chapter.

R code 6.7

```
m3 <- map(
    alist(
        blooms ~ dnorm(mu,sigma),
        mu <- a + bW*water.c + bS*shade.c +
              bWS*water.c*shade.c ,
        a ~ dnorm(130,100),
        c(bW,bS,bWS) ~ dnorm(0,100),
        sigma ~ dunif(0,100)
    ),
    start=list(a=130,bW=0,bS=0,bWS=0,sigma=90),
    data=d )
compare(m2,m3)
```

```
    WAIC pWAIC dWAIC weight    SE  dSE
m2 294.3   9.5   0.0   0.67  9.56   NA
m3 295.7   6.4   1.4   0.33 10.03 7.99
```

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.

7H3. (a) The first thing you need to do is make the new data frame, missing Seychelles. Here's the code to do so:

R code 6.8

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

```
d2 <- dd[ dd$country!="Seychelles" , ]
```

So now `d2` contains all the complete cases, excepting Seychelles. Fitting the interaction model is straightforward, despite how ugly the code looks:

R code 6.9

```
m3 <- map(
    alist(
        log_gdp ~ dnorm( mu , sigma ) ,
        mu <- a + bA*cont_africa + br*rugged +
            bAr*cont_africa*rugged,
        a ~ dnorm(0,100),
        c(bA,br,bAr) ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) ,
    data=d2 )
```

To ease comparison, I'll also refit the model using all of the data in `dd`, which is in the book and lecture. Call this model `m3all` now. Then to compare the estimates:

R code 6.10

```
coeftab(m3,m3all)
```

```
      m3      m3all
a        9.22    9.22
bA      -1.88   -1.95
br       -0.2    -0.2
bAr      0.30    0.39
sigma    0.93    0.93
nobs      169     170
```

So the estimates are very similar, but the magnitude of the interaction parameter, `bAr`, has gone down, after removing Seychelles. Let's take a look at the standard deviations and intervals of the marginal posterior distributions:

R code 6.11

```
precis(m3)
```

```
       Mean StdDev  5.5% 94.5%
a      9.22   0.14  9.00  9.44
bA    -1.88   0.23 -2.24 -1.52
br    -0.20   0.08 -0.32 -0.08
bAr    0.30   0.14  0.08  0.52
sigma  0.93   0.05  0.84  1.01
```

Still reliably above zero, so seems like removing Seychelles didn't kill the interaction entirely, even though it did diminish its strength. Computing the expected slope in Africa:

$$\beta_r + \beta_{Ar}(1) \approx -0.2 + 0.3 = 0.1.$$

And for the original fit, including Seychelles, we got:

$$\beta_r + \beta_{Ar}(1) \approx -0.2 + 0.39 = 0.19.$$

So the expected slope in Africa has been cut approximately in half, compared to the model that used all of the data.

(b) I'm going to plot the predicted relationship for African nations, with a blue regression line for the old fit also on the same plot. Here are the calculations you are accustomed to by now:

R code 6.12

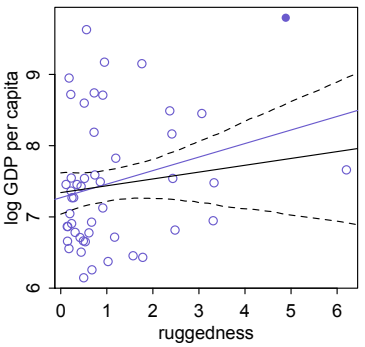
```
post <- extract.samples( m3 )
rugged.seq <- seq(from=0,to=8,by=0.1)
pred.dat <- data.frame(rugged=rugged.seq,cont_africa=1)
mu <- link( m3 , data=pred.dat )
mu.mean <- apply( mu , 2 , mean )
mu.ci <- apply( mu , 2 , PI )
```

And here is the code to plot it all:

R code 6.13

```
dplot <- dd[dd$cont_africa==1,]
plot( log_gdp ~ rugged , data=dplot , col="slateblue" ,
    pch=ifelse(dplot$country=="Seychelles",16,1) ,
    ylab="log GDP per capita" , xlab="ruggedness" )
abline( a=9.22-1.95 , b=0.19 , col="slateblue" )
lines( rugged.seq , mu.mean )
lines( rugged.seq , mu.ci[1,] , lty=2 )
lines( rugged.seq , mu.ci[2,] , lty=2 )
```

And here's the plot that results:



Seychelles is shown by the filled point. Just like the estimates suggested, the slope has been cut approximately in half. Notice too that the uncertainty around the line has grown, compared to the plots in lecture and the book chapter. The predicted relationship outside of Africa (not shown) has not changed: it is still reliably negative.

(c) To fit the other two models now:

R code 6.14

```
m1 <- map(
    alist(
        log_gdp ~ dnorm( mu , sigma ) ,
        mu <- a + br*rugged,
        a ~ dnorm(0,100),
        br ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) ,
    data=d2 )
m2 <- map(
    alist(
        log_gdp ~ dnorm( mu , sigma ) ,
        mu <- a + bA*cont_africa + br*rugged,
```

```
        a ~ dnorm(0,100),
        c(br,bA) ~ dnorm(0,10),
        sigma ~ dunif(0,50)
    ) ,
    data=d2 )
```

I've made the priors rather flat, so you might get some bad start values and have to try a few times before the models converge. Once they do, here is the comparison:

R code 6.15

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

```
    WAIC pWAIC dWAIC weight    SE   dSE
m3 463.5   4.7   0.0    0.8 15.30    NA
m2 466.2   4.0   2.8    0.2 14.36  3.86
m1 536.0   2.6  72.6    0.0 13.25 15.79
```

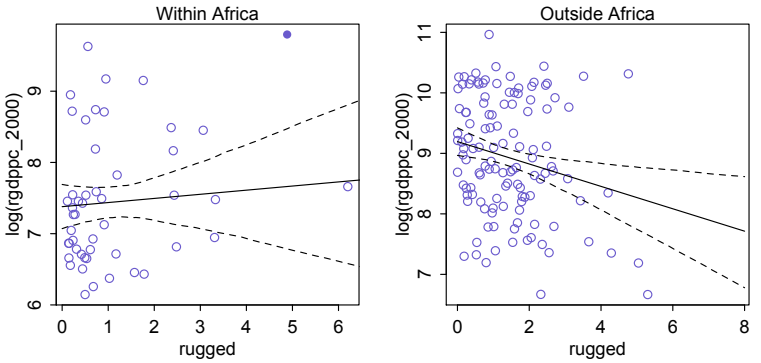
To generate the WAIC-averaged predictions:

R code 6.16

```
r.seq <- seq(from=0,to=8,by=0.1)
pred.dat <- data.frame(rugged=r.seq,cont_africa=1)
mu_ensemble <- ensemble(m1,m2,m3,data=pred.dat)
mu.mean <- apply( mu_ensemble$link , 2 , mean )
mu.ci <- apply( mu_ensemble$link , 2 , PI )
dPlot <- d[ d$cont_africa==1 , ]
plot( log(rgdppc_2000) ~ rugged , data=dPlot ,
    pch=ifelse(dPlot$country=="Seychelles",16,1) ,
    col="slateblue" )
lines( r.seq , mu.mean )
lines( r.seq , mu.ci[1,] , lty=2 )
lines( r.seq , mu.ci[2,] , lty=2 )
```

You can change the 1 values for the Africa dummy variable inside of `pred.dat`, to make a similar plot for nations outside of Africa. Also be sure to change the "1" in the `dPlot` line that pulls out the raw data, as well.

Here are the plots, showing model-averaged predictions for the data omitting Seychelles:



What are we to make of these results? I think the model comparison analysis suggests that the relationship between ruggedness and GDP is different inside and outside of Africa. However, there is substantial uncertainty about the magnitude of the difference. The most plausible difference (conditional on model/data) is shown in the plots above: about flat within Africa and about -0.2 outside of it. But the data are consistent with both larger and smaller differences.

And it's worth remembering that the *difference* in slope is not the overlap in the marginal distributions of each slope. If you want the contrast, compute the contrast. We did that in lecture and in the book.

7H4. (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.

R code 6.17

```
library(rethinking)
data(nettle)
d <- nettle
d$log_langpc <- log( d$num.lang / d$k.pop )
d$log_area <- log(d$area)

m0 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a + 0,
        a ~ dnorm(0,100),
        sigma ~ dunif(0,10)
    ) , data=d )
m1 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a + bg*mean.growing.season,
        a ~ dnorm(0,100),
        bg ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ) , data=d )
m2 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a + bg*mean.growing.season + ba*log_area,
        a ~ dnorm(0,100),
        c(bg,ba) ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ) , data=d )

compare(m0,m1,m2)
```

```
    WAIC pWAIC dWAIC weight    SE  dSE
m1 268.2   3.8   0.0   0.51 15.33   NA
m2 268.3   5.0   0.1   0.48 15.78 3.71
m0 276.1   2.8   8.0   0.01 16.88 6.02
```

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 6.18

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

```
                  m1      m2
(Intercept)     -6.68   -3.86
```

```
mean.growing.season    0.17    0.14
log_area                 NA    -0.2
nobs                     74      74
```

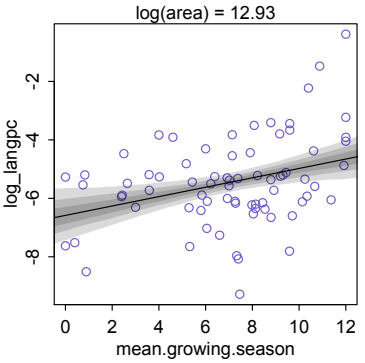
You can check the standard errors, too, to ensure that the estimate for `mean.growing.season` is reliably positive in both cases.

Plotting the predicted relationship, averaging across models:

R code 6.19

```
mu.area <- mean(d$log_area)
x.seq <- seq(from=-1,to=13,by=0.25)
new.dat <- data.frame(log_area=mu.area,mean.growing.season=x.seq)

mu <- ensemble(m0,m1,m2,data=new.dat)
mu.mean <- apply( mu$link , 2 , mean )

plot( log_langpc ~ mean.growing.season , data=d ,
    col="slateblue" )
mtext( paste("log(area) =",round(mu.area,2)) , 3 )
lines( x.seq , mu.mean )

# plot several intervals with shading
for ( p in c(0.5,0.79,0.95) ) {
mu.PI <- apply( mu$link , 2 , PI , prob=p )
shade( mu.PI , x.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 6.20

```
m1 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a + bs*sd.growing.season,
```

```
        a ~ dnorm(0,100),
        bs ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ) , data=d )
m2 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a + bs*sd.growing.season + ba*log_area,
        a ~ dnorm(0,100),
        c(bs,ba) ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ) , data=d )

compare(m0,m1,m2)
```

```
    WAIC pWAIC dWAIC weight    SE  dSE
m1 273.7   4.0   0.0   0.52 17.23   NA
m2 274.6   5.7   0.9   0.33 17.34 3.94
m0 276.1   2.8   2.4   0.15 17.06 4.65
```

A very similar story, although not as strong a relationship. Take a look at the marginal posterior from `m2`:

```
precis(m2)
```

R code 6.21

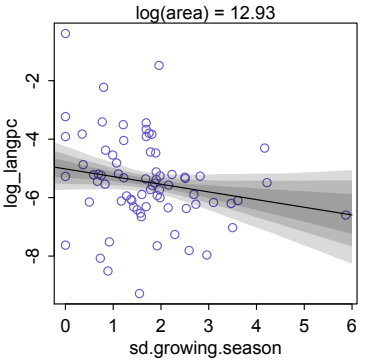
```
       Mean StdDev  5.5% 94.5%
a     -2.02   1.88 -5.03  0.98
bs    -0.21   0.19 -0.51  0.09
ba    -0.24   0.16 -0.49  0.01
sigma  1.44   0.12  1.25  1.63
```

Including log(area) generates considerable uncertainty about the direction of the effect of `sd.growing.season`. Still, the MAP is negative, as predicted.

Plotting the model-averaged relationship:

```
mu.area <- mean(d$log_area)
x.seq <- seq(from=-1,to=6,length.out=30)
new.dat <- data.frame(log_area=mu.area,sd.growing.season=x.seq)

mu <- ensemble(m0,m1,m2,data=new.dat)
mu.mean <- apply( mu$link , 2 , mean )

plot( log_langpc ~ sd.growing.season , data=d ,
    col="slateblue" )
mtext( paste("log(area) =",round(mu.area,2)) , 3 )
lines( x.seq , mu.mean )

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

R code 6.22



So the relationship isn't likely flat or positive, but many slightly negative slopes are consistent with the data (and model structure).

(c) You can do a bunch of model comparison here. What you'll find is that the model that interacts mean and sd of growing season does best, with some indication that including `log(area)` helps, and some weight reserved to the model with only main effects of mean and sd of growing season. I'll go ahead and just work with the interaction model, because it's the trickiest. But modifying the later plotting code to use `ensemble` should be straightforward: just replace any calls to `link` with calls to `ensemble`, and be sure to extract the `link` slot, as in the examples above.

Here's the model:

R code 6.23

```
m3 <- map(
    alist(
        log_langpc ~ dnorm(mu,sigma),
        mu <- a +
              bg*mean.growing.season +
              bs*sd.growing.season +
              bgs*mean.growing.season*sd.growing.season,
        a ~ dnorm(0,100),
        c(bg,bs,bgs) ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ) , data=d )
```

Okay, so let's inspect the marginal posterior:

R code 6.24

```
precis(m3)
```

```
       Mean StdDev  5.5% 94.5%
a     -6.97   0.59 -7.92 -6.03
bg     0.30   0.07  0.18  0.41
bs     0.42   0.37 -0.17  1.02
bgs   -0.11   0.05 -0.18 -0.03
sigma  1.31   0.11  1.13  1.48
```

Now keep in mind that we haven't centered the predictor variables here, so interpreting the parameters is hazardous. Well, interpreting coefficients in an interaction model is almost always hazardous, centered predictors or not. But 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 6.25

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

# now loop over the three plots
# draw languages against mean.growing.season in each
mean.seq <- seq(from=-1,to=13,length.out=30)
par(mfrow=c(1,3)) # set up plot window for row of 3 plots
for ( i in 1:3 ) {
    sd.val <- sd.seq[i] # select out value for this plot
    new.dat <- data.frame(
        mean.growing.season = mean.seq,
        sd.growing.season = sd.val )
    mu <- link( m3 , 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$sd.growing.season , sd.val , 2 , "slateblue" )

    plot( log_langpc ~ mean.growing.season , data=d , col=cols )
    mtext( paste("sd.growing.season =",round(sd.val,2)) , 3 , cex=0.75 )
    lines( mean.seq , mu.mean )
    shade( mu.PI , mean.seq )
}
```

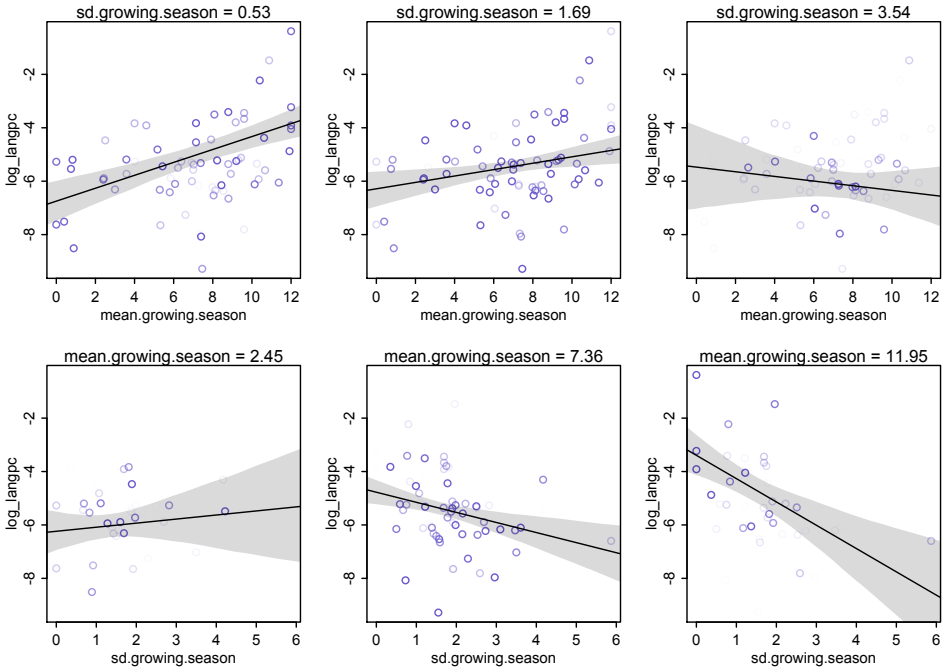
And this code will produce the analogous triptych in which `sd.growing.season` is varied on the horizontal axis.

R code 6.26

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

# now loop over the three plots
x.seq <- seq(from=-1,to=6.5,length.out=30)
par(mfrow=c(1,3)) # set up plot window for row of 3 plots
for ( i in 1:3 ) {
    top.val <- top.seq[i] # select out value for this plot
    new.dat <- data.frame(
        mean.growing.season = top.val ,
        sd.growing.season = x.seq )
```

```
    mu <- link( m3 , 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$mean.growing.season , top.val , 5 , "slateblue" )

    plot( log_langpc ~ sd.growing.season , data=d , col=cols )
    mtext( paste("mean.growing.season =",round(top.val,2)) , 3 , cex=0.75 )
    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).

7. Chapter 8 Solutions

8E1. Only (3) is required.

8E2. 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.

8E3. 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.

8E4. 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.

8E5. `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.

8E6. 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.

8M1. Here is the model again, now using a uniform prior for `sigma`:

R code 7.1

```
# 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.trim <- dd[ , c("log_gdp","rugged","cont_africa") ]

# new model with uniform prior on sigma
m8.1_unif <- map2stan(
    alist(
        log_gdp ~ dnorm(mu,sigma),
        mu <- a + bR*rugged + bA*cont_africa + bAR*rugged*cont_africa,
        a ~ dnorm(0,100),
        bR ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bAR ~ dnorm(0,10),
        sigma ~ dunif(0,10)
    ),
    data=dd.trim , chains=2 )
```

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

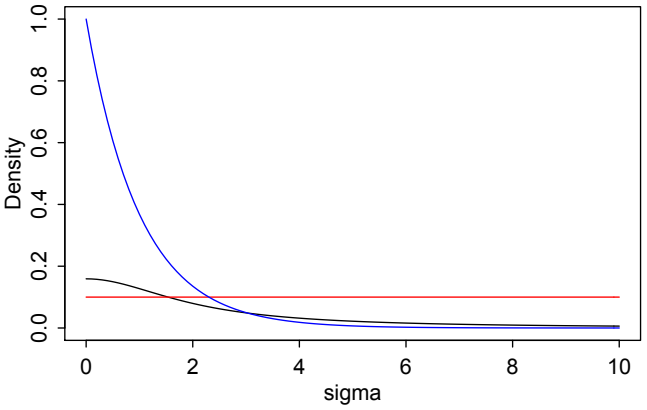
R code 7.2

```
m8.1_exp <- map2stan(
    alist(
        log_gdp ~ dnorm(mu,sigma),
        mu <- a + bR*rugged + bA*cont_africa + bAR*rugged*cont_africa,
        a ~ dnorm(0,100),
        bR ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bAR ~ dnorm(0,10),
        sigma ~ dexp(1)
    ),
    data=dd.trim , chains=2 )
```

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 7.3

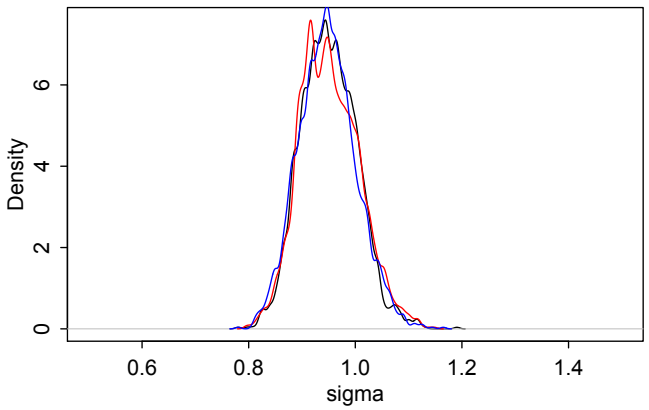
```
curve( dcauchy(x,0,2) , from=0 , to=10 ,
    xlab="sigma" , ylab="Density" , ylim=c(0,1) )
curve( dunif(x,0,10) , add=TRUE , col="red" )
curve( dexp(x,1) , add=TRUE , col="blue" )
```



Now let's compare the posterior distributions of `sigma` for all three models. If you need to re-run model `m8.1stan` from the chapter, do so. What I'll do here is extract all three sets of samples and just plot them over one another.

R code 7.4

```
sigma_cauchy <- extract.samples(m8.1stan,pars="sigma")
sigma_unif <- extract.samples(m8.1_unif,pars="sigma")
sigma_exp <- extract.samples(m8.1_exp,pars="sigma")
dens( sigma_cauchy[[1]] , xlab="sigma" , xlim=c(0.5,1.5) )
dens( sigma_unif[[1]] , add=TRUE , col="red" )
dens( sigma_exp[[1]] , add=TRUE , col="blue" )
```



All three posterior distributions are very similar. Why? There's a lot of data, relative to the difference in priors. The next problem asks you to explore what happens when you make them stronger.

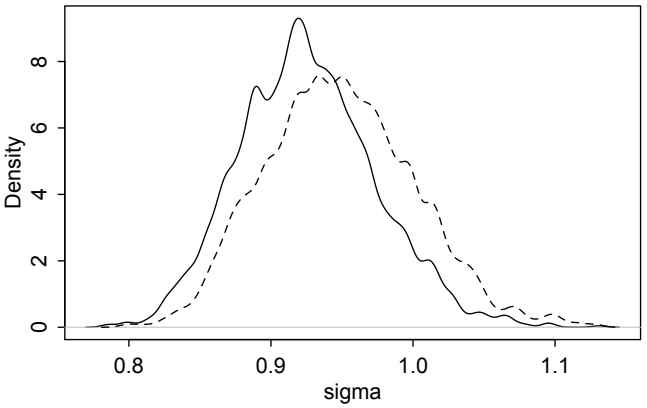
8M2. To address this problem, merely change the `dcauchy(0,2)` and `dexp(1)` priors in the previous models. The trick in this problem is that the "2" in the Cauchy is *scale*. As it gets bigger, the prior gets weaker. But the "1" in the exponential distribution is an *inverse* scale. So as you make it bigger, the prior gets stronger. Once you realize that, the rest is straightforward.

If you make the priors strong enough, they can easily pull the posterior distribution closer to zero. But you'll have to make them very strong, for this data. For example, let's make the exponential prior quite strong, with an expected value of $1/10$:

R code 7.5

```
m8.1_exp_new <- map2stan(
    alist(
        log_gdp ~ dnorm(mu,sigma),
        mu <- a + bR*rugged + bA*cont_africa + bAR*rugged*cont_africa,
        a ~ dnorm(0,100),
        bR ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bAR ~ dnorm(0,10),
        sigma ~ dexp(10)
    ),
    data=dd.trim , chains=2 )
sigma_old <- extract.samples(m8.1_exp,pars="sigma")
sigma_new <- extract.samples(m8.1_exp_new,pars="sigma")
dens( sigma_new[[1]] , xlab="sigma" )
```

```
dens( sigma_old[[1]] , add=TRUE , lty=2 )
```

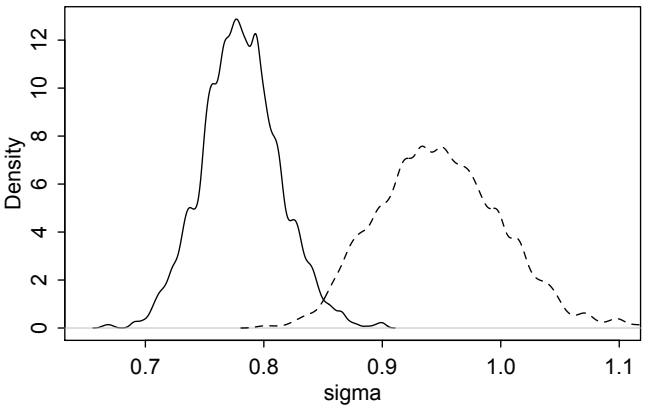


The dashed density is the original one, using the `dexp(1)` prior from the previous problem. The prior has hardly made a difference. Why? Because the likelihood is dominating, and it doesn't put hardly any probably over the small values of `sigma` that the prior likes.

Let's try something radically strong:

R code 7.6

```
m8.1_exp_new2 <- map2stan(
    alist(
        log_gdp ~ dnorm(mu,sigma),
        mu <- a + bR*rugged + bA*cont_africa + bAR*rugged*cont_africa,
        a ~ dnorm(0,100),
        bR ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bAR ~ dnorm(0,10),
        sigma ~ dexp(100)
    ),
    data=dd.trim , chains=2 )
sigma_new2 <- extract.samples(m8.1_exp_new2,pars="sigma")
dens( sigma_new2[[1]] , xlab="sigma" , xlim=c(0.65,1.1) )
dens( sigma_old[[1]] , add=TRUE , lty=2 )
```



Now it has moved a little.

In general it will be harder to get the posterior to move when using the Cauchy prior, because the Cauchy has more mass in its tail. Try it out.

8M3. 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 7.7

```
# compile model
# use fixed start values for comparability of runs
m <- map2stan(
    alist(
        log_gdp ~ dnorm(mu,sigma),
        mu <- a + bR*rugged + bA*cont_africa + bAR*rugged*cont_africa,
        a ~ dnorm(0,100),
        bR ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bAR ~ dnorm(0,10),
        sigma ~ dcauchy(0,2)
    ),
    start=list(a=8,bR=0,bA=0,bAR=0,sigma=1) ,
    data=dd.trim , chains=1 , iter=1 )

# define warmup values to run through
warm_list <- c(1,5,10,100,500,1000)

# 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 <- resample( m , chains=1 ,
        iter=1000+w , warmup=w , refresh=-1 , WAIC=FALSE )
    n_eff[i,] <- precis(m_temp)@output$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)@output)
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          bR          bA         bAR        sigma
1       0.5007511   0.5007511   0.5007511   0.5007511    0.5007511
5       4.3256825   4.0200142   3.9558464   2.8122120   13.3467754
10    304.3737104 370.7338882 189.3232743 262.6432755 1000.0000000
100   321.0008576 394.6673949 173.1020514 244.2395800  869.4751451
500   241.1348599 268.6110577 277.8579786 283.0203281  370.2711484
1000  219.7749135 190.9539593 259.8739707 315.2980722  680.2881654
```

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 5:

R code 7.8

```
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 7.9

```
m <- map2stan(
    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(1,5,10,100,500,1000)
n_eff <- matrix( NA , nrow=length(warm_list) , ncol=4 )
for ( i in 1:length(warm_list) ) {
    w <- warm_list[i]
    m_temp <- resample( m , chains=1 ,
        iter=1000+w , warmup=w , refresh=-1 , WAIC=FALSE )
    n_eff[i,] <- precis(m_temp)@output$n_eff
}
colnames(n_eff) <- rownames(precis(m_temp)@output)
rownames(n_eff) <- warm_list
show(n_eff)
```

```
                a          bl          br       sigma
1      0.5007511   0.5007511   0.5007511   0.5007511
5      4.5461377   6.7509053   3.4498516  10.9574569
10     2.9052710   4.5635038   5.6987064   6.5577471
100  406.5540250 109.5477783 109.7709377 590.3716127
500  415.5212711 294.6830846 295.2950901 449.1176021
1000 441.6867022 336.5340548 336.3034810 346.3409875
```

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 correlation,

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.

8H1. 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:

```
precis(mp)
```

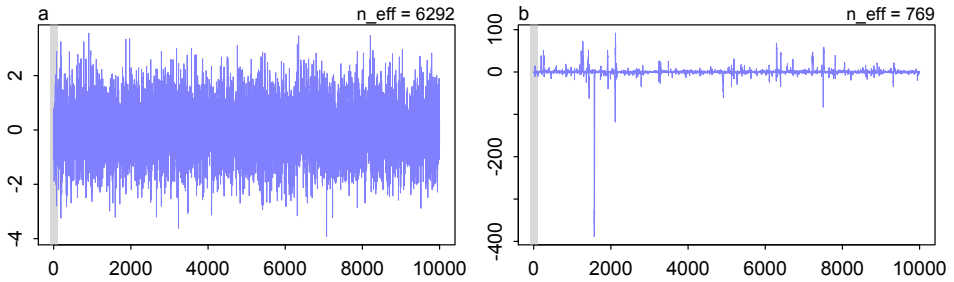
R code 7.10

```
  Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a 0.00   0.99      -1.60       1.58  6292    1
b 0.15  13.70      -5.45       5.76   769    1
```

The first fact that might make you worry that something went wrong is that sampling for `b` has a much lower effective sample size than `a`. But that's okay. 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:

```
plot( mp , n_col=2 )
```

R code 7.11



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.

8H2. First, load and prepare the data. Note that while Chapter 5 uses dot-notation for names like `Marriage.s`, while switch to using underscore-notation now, because Stan hates dots. This notation issue can be annoying, but it's a common fact of scientific computing that software cares about this stuff, so best to get used to it.

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

R code 7.12

```
d$MedianAgeMarriage_s <- (d$MedianAgeMarriage-mean(d$MedianAgeMarriage))/
    sd(d$MedianAgeMarriage)
d$Marriage_s <- (d$Marriage - mean(d$Marriage))/sd(d$Marriage)
```

Now to fit the models over again, this time using `map2stan`:

R code 7.13

```
m5.1_stan <- map2stan(
    alist(
        Divorce ~ dnorm( mu , sigma ) ,
        mu <- a + bA * MedianAgeMarriage_s ,
        a ~ dnorm( 10 , 10 ) ,
        bA ~ dnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 10 )
    ) ,
    data = d , chains=4 )
m5.2_stan <- map2stan(
    alist(
        Divorce ~ dnorm( mu , sigma ) ,
        mu <- a + bR * Marriage_s ,
        a ~ dnorm( 10 , 10 ) ,
        bR ~ dnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 10 )
    ) ,
    data = d , chains=4 )
m5.3_stan <- map2stan(
    alist(
        Divorce ~ dnorm( mu , sigma ) ,
        mu <- a + bR*Marriage_s + bA*MedianAgeMarriage_s ,
        a ~ dnorm( 10 , 10 ) ,
        bR ~ dnorm( 0 , 1 ) ,
        bA ~ dnorm( 0 , 1 ) ,
        sigma ~ dunif( 0 , 10 )
    ) ,
    data = d , chains=4 )
```

Since you are possibly still getting used to Markov chains, I recommend checking the trace 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 7.14

```
compare(m5.1_stan,m5.2_stan,m5.3_stan)
```

```
           WAIC pWAIC dWAIC weight    SE  dSE
m5.1_stan 186.2   3.8   0.0   0.67 12.48   NA
m5.3_stan 187.6   4.7   1.4   0.33 12.36 0.82
m5.2_stan 199.6   3.1  13.5   0.00  9.60 9.13
```

The model with only age-at-marriage comes out on top, although the model with both predictors does nearly as well. In fact, the 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 7.15

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

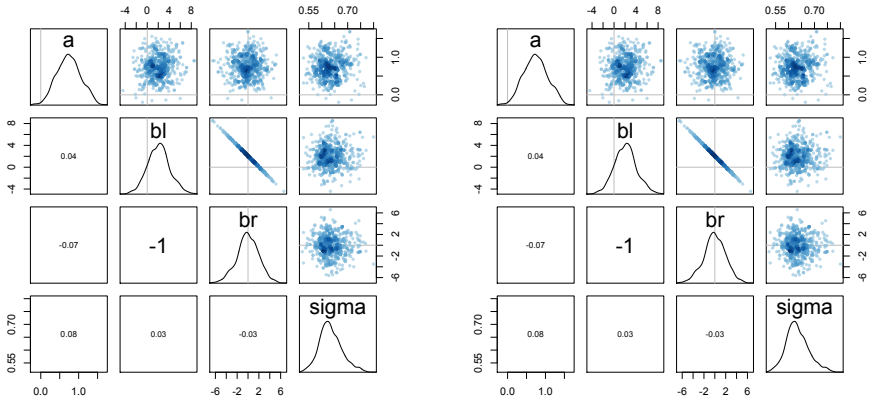
```
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
```

```
a      9.68   0.21       9.34      10.03  1872     1
bR    -0.12   0.28      -0.59       0.32  1949     1
bA    -1.12   0.28      -1.56      -0.67  2072     1
sigma  1.52   0.16       1.28       1.79  1530     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 3.8—the two models rank almost the same. This makes sense, because you already learned back in Chapter 5 that marriage rate 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 WAIC expects. Only the future will tell which model is actually better for forecasting. WAIC is not an oracle. It's a golem.

8H3. Looking at the `pairs` plot for each model 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.

8H4. Inspecting WAIC for the two models, you get:

R code 7.16

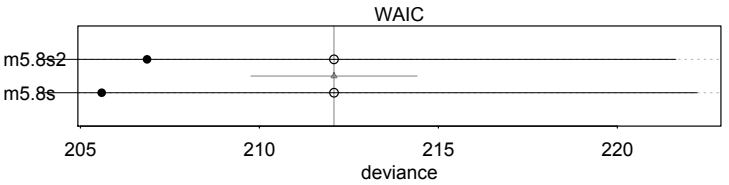
```
compare(m5.8s,m5.8s2)
```

```
        WAIC pWAIC dWAIC weight    SE  dSE
m5.8s2 212.1   2.6     0    0.5  9.55   NA
m5.8s  212.1   3.2     0    0.5 10.14 2.32
```

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. You can see this by plotting:

R code 7.17

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



The black dots show in-sample deviance (well, −2lppd here, the fully Bayesian deviance that WAIC uses). Notice that `m5.8s2` has a higher (worse) deviance in-sample. It's less flexible, though, so the distance from it's in-sample deviance (filled point) to its out-of-sample deviance (open circle) is smaller than for model `m5.8s`. The two models end up tied, right on the same estimated out-of-sample values.

8H5. 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 7.18

```
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.

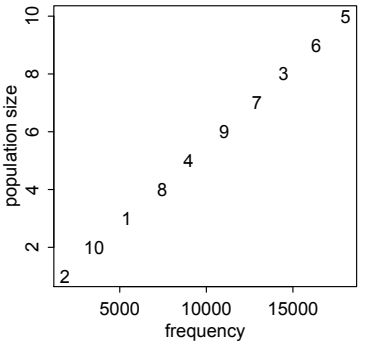
R code 7.19

```
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 7.20

```
# 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.

8H6. Okay this is most likely the hardest problem in the book, in any chapter. 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:

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.

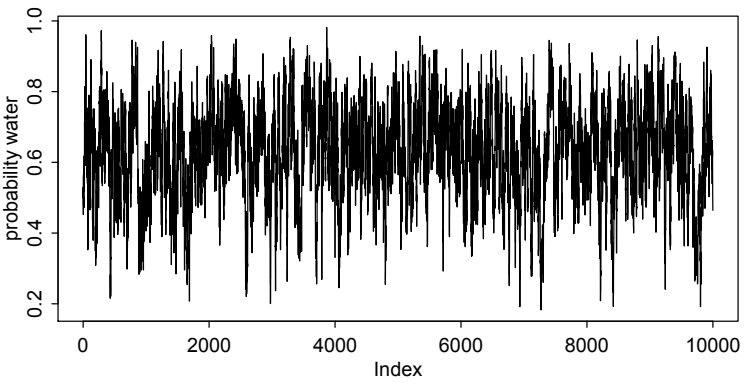
R code 7.21

```
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 7.22

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

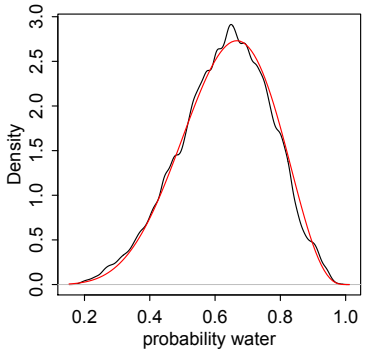


This chain is fine, but it is more autocorrelated than a typical Stan chain. That's why it tends to wander around a bit. It is stationary, but it isn't mixing very well. This is common of such a simple Metropolis chain. But it is working.

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 7.23

```
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 7.24

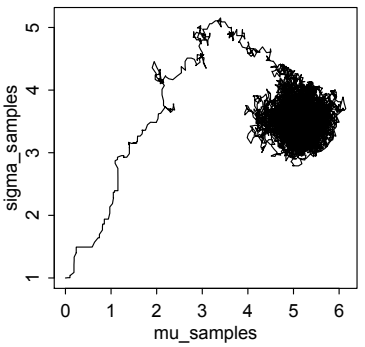
```
# simulate some data
# 100 observations with mean 5 and sd 3
y <- rnorm( 100 , 5 , 3 )

# 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 7.25

```
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.

8. Chapter 10 Solutions

10E1. 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 8.1

```
[1] -0.6190392
```

10E2. 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 8.2

```
[1] 0.9608343
```

10E3. Proportional odds is calculated by exponentiating the coefficient. So:

```
exp( 1.7 )
```

R code 8.3

```
[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.

10E4. 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.

10M1. 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.

10M2. 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.

10M3. 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.

10M4. 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.

10M5. 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.

10M6. 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.

10H1. The necessary model is found on page 313. You just need to use `map` instead of `map2stan`. This code will fit the model:

R code 8.4

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d2 <- d
d2$recipient <- NULL
m10.4_map <- map(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + (bp + bpC*condition)*prosoc_left ,
        a[actor] ~ dnorm(0,10),
        bp ~ dnorm(0,10),
```

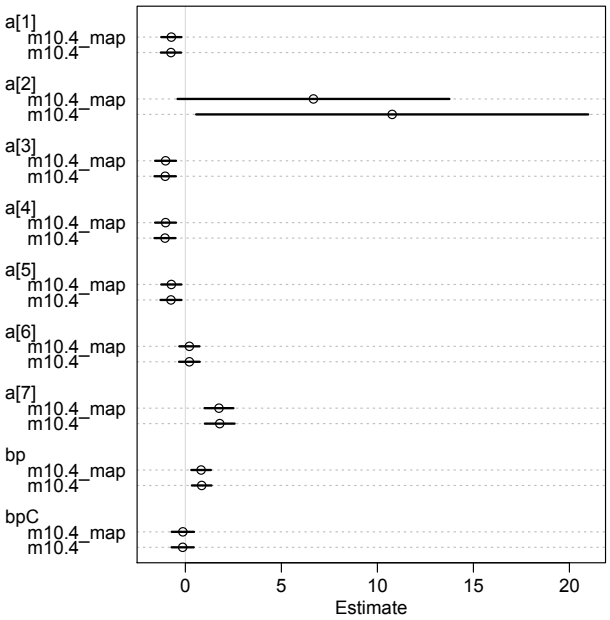
```
        bpC ~ dnorm(0,10)
    ) ,
    data=d2 )
```

Be sure to fit the original model, as well, so you can compare them.

The most obvious difference between the quadratic approximation (`m10.4_map`) and the fully Bayesian fit (`m10.4`) is the posterior for the second intercept, the one for actor number 2. Let's compare:

R code 8.5

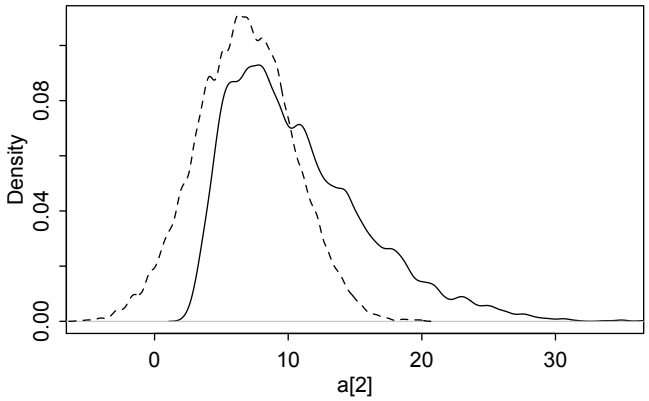
```
plot(coeftab(m10.4,m10.4_map))
```



While it makes essentially no difference for estimating the treatment effects, the quadratic approximation underestimates the uncertainty for actor 2's handedness. Why? Because the quadratic approximation enforces symmetric uncertainty around the posterior mode, but the nature of a logistic regression often leads to asymmetric uncertainty. In this case, there is more uncertainty on the up side than the down side. Comparing the marginal posterior distributions helps to emphasize this:

R code 8.6

```
post_map <- extract.samples(m10.4_map)
post_map2stan <- extract.samples(m10.4)
dens( post_map2stan$a[,2] , xlab="a[2]" ,
    ylim=c(0,0.11) , xlim=c(-5,35) )
dens( post_map$a[,2] , lty=2 , add=TRUE )
```



The solid density is the fully Bayesian posterior produced by `map2stan`. The dashed density is the quadratic approximation produced by `map`. Note the long tail on the righthand side, for the fully Bayesian posterior. This arises from the fact that basically any sufficiently large value will guarantee a probability of 1 for the event. So an infinite number of parameter values are in principle consistent with the data, since actor 2 *never* pulled the righthand lever. Both ways of fitting the model appreciate the strong preference, but the quadratic approximation cannot appreciate the asymmetry in uncertainty, so it ends up assigning too little on the upwards side and too much on the downwards side. The fully Bayesian posterior, in contrast, has no trouble recognizing the imbalance in uncertainty.

10H2. Begin by fitting all of the models, but this time using `map2stan` for all of them. Why? Because it is usually unwise to compare models that are estimated by different methods. In this case, it won't alter the inferences. But in principle a method like quadratic approximation could produce a better fit to sample, but be less representative of the true uncertainty implied by the model and data. In particular, since WAIC is computed using posterior variance, and quadratic estimation often misrepresents that variance, you have to be careful. So let's do them all the same way, so we don't have to wonder.

R code 8.7

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d2 <- d
d2$recipient <- NULL

m10.1 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a ,
        a ~ dnorm(0,10)
    ) ,
    data=d2 , chains=2 )
m10.2 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a + bp*prosoc_left ,
        a ~ dnorm(0,10) ,
        bp ~ dnorm(0,10)
```

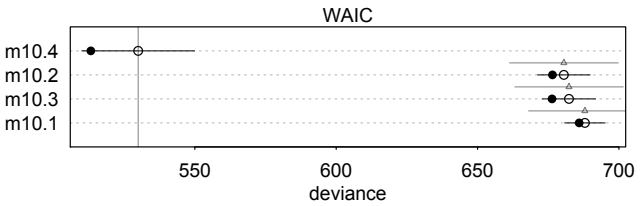
```
    ) ,
    data=d2 , chains=2 )
m10.3 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a + (bp + bpC*condition)*prosoc_left ,
        a ~ dnorm(0,10) ,
        bp ~ dnorm(0,10) ,
        bpC ~ dnorm(0,10)
    ) ,
    data=d2 , chains=2 )
m10.4 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + (bp + bpC*condition)*prosoc_left ,
        a[actor] ~ dnorm(0,10),
        bp ~ dnorm(0,10),
        bpC ~ dnorm(0,10)
    ) ,
    data=d2 , chains=2 , iter=2500 , warmup=500 )
```

Now compare all four models:

R code 8.8

```
compare(m10.1,m10.2,m10.3,m10.4)
plot(compare(m10.1,m10.2,m10.3,m10.4))
```

```
       WAIC pWAIC dWAIC weight    SE   dSE
m10.4 530.0   8.4   0.0      1 19.95    NA
m10.2 680.5   2.0 150.6      0  9.32 19.23
m10.3 682.3   3.0 152.4      0  9.40 19.17
m10.1 688.0   1.0 158.0      0  7.15 19.94
```



Model `m10.4` easily dominates the others. This is a result of the variation among actors, which dwarfs the variation arising from treatment. Since only model `m10.4` recognizes the variation among actors, it does much better in sample. It does so much better in sample, that even after accounting for the expected penalty out-of-sample, it still is expected to out-perform the other, simpler models.

The caveat here is that this comparison applies only to predicting new observations for the same chimpanzees. For new chimpanzees, we couldn't use their individual intercepts to make predictions. We could use the variation among chimpanzees to calibrate our predictions, though. But to do that we'll need to actually estimate the variation itself. That will wait until Chapter 12. Near the end of that chapter, we'll return to this kind of comparison problem.

10H3. (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 8.9

```
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:

R code 8.10

```
f <- alist(
        y ~ dbinom( n , p ),
        logit(p) <- a + bP*pirateL + bV*victimL + bA*pirateA ,
        a ~ dnorm(0,10),
        bP ~ dnorm(0,5),
        bV ~ dnorm(0,5),
        bA ~ dnorm(0,5)
    )

m1 <- map( f , data=d )

m1.stan <- map2stan( f , data=d , warmup=1000 , iter=1e4 )

precis(m1)
precis(m1.stan)
```

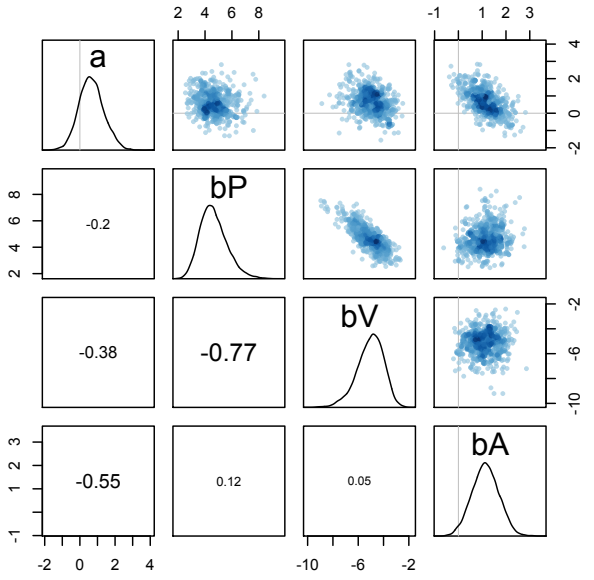
```
    Mean StdDev  5.5% 94.5%
a   0.59   0.66 -0.47  1.65
bP  4.24   0.90  2.81  5.67
bV -4.59   0.96 -6.13 -3.06
bA  1.08   0.53  0.23  1.94

    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a   0.66   0.70      -0.51       1.75  3431    1
bP  4.64   1.01       3.07       6.21  2740    1
bV -5.06   1.06      -6.63      -3.37  2699    1
bA  1.14   0.56       0.23       2.02  3428    1
```

Hm, those estimates don't look the same. Checking the trace plot for `m1.stan` confirms that it fit correctly. So something is wrong with the quadratic approximation here. Taking a look at the pairs plot confirms that there is some serious skew here:

R code 8.11

```
pairs(m1.stan)
```

And below is what it should look like:



What has happened here is that both `bP` and `bV` are hitting ceiling and floor effects, due to the binomial outcome. The parameter `bP` is hitting the ceiling, as a large pirate has so big an advantage that many extreme values are all consistent with the data. Basically any value of `bP` that ensures nearly 100% success will do. So there's a long tail of large values on the right side of the posterior distribution for `bP`. The same thing is going on with `bV`, but in the opposite direction, hitting the floor.

(b) Now for interpreting these estimates. We'll use the `map2stan` 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 8.12

```
logistic( 0.66 )
```

```
[1] 0.6592604
```

66% of attempts by immature small pirates on small victims are expected to succeed.

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 8.13

```
logistic( 0.66 + 4.64 )
```

```
[1] 0.9950332
```

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.

But what about the proportional odds (odds ratio) interpretations? You can exponentiate each β estimate to get the proportional change in odds that arising from changing the corresponding dummy

variable from 0 to 1. Ignoring the confidence intervals for the moment (they'll appear in the plots to come), you can quickly exponentiate the β estimates with:

R code 8.14

```
round( exp( coef(m1.stan)[2:4] ) , 3 )
```

```
     bP      bV      bA
103.608   0.006   3.122
```

So this says that the odds $(p/(1-p))$ of a successful attempt are multiplied by 104, when the pirate becomes large. That's a huge effect. Similarly, the odds of success are multiplied by 0.006, when the victim becomes large. So if the pirate is large and the victim is large, the proportional change in odds is $104 \times 0.006 \approx 0.62$ or about 40% reduction in odds. Finally, the effect of being adult is to improve the odds, multiplying them by about 3, tripling the odds of a successful attempt. These are all *relative* risk measures. For the absolute measures, it'll be nice to plot them.

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 8.15

```
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" ) )
```

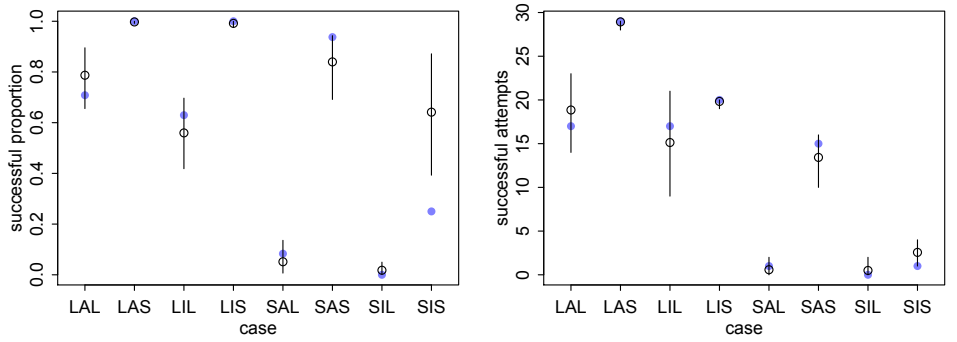
And here's the second plot, showing number of successes on the vertical:

R code 8.16

```
# display posterior predicted probability of success
points( 1:8 , p.mean )
for ( i in 1:8 ) lines( c(i,i) , p.PI[,i] )

# 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 probability of success
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 8.17

```
m2.stan <- map2stan(
    alist(
        y ~ dbinom( n , p ),
        logit(p) <- a + bP*pirateL + bV*victimL +
            bA*pirateA + bPA*pirateL*pirateA ,
        a ~ dnorm(0,10),
        bP ~ dnorm(0,5),
        bV ~ dnorm(0,5),
        bA ~ dnorm(0,5),
        bPA ~ dnorm(0,5)
    ) , data=d , warmup=1000 , iter=1e4 )

compare(m1.stan,m2.stan)
```

```
        WAIC pWAIC dWAIC weight    SE  dSE
m2.stan 93.9   4.7   0.0   0.92 12.76   NA
m1.stan 98.7   4.1   4.9   0.08 13.33 4.68
```

That's strong support for the interaction model, but the main effect model doesn't completely vanish. The model with the interaction gets 92% of the weight. Note that we can't be too slavish to these WAIC values and their weights, because the standard error of the difference is of the same magnitude as the difference in WAIC. It still makes sense to bet on `m2.stan`, if you must bet. In this case, I

personally feel more secure appealing to actual *biology* to justify the interaction model. There are good behavioral ecological reasons to expect an interaction, and there is good evidence of one. WAIC is not necessarily the most important thing.

Let's look at the estimates for the interaction model:

R code 8.18

```
precis(m2.stan)
```

```
     Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a   -0.79   1.06      -2.44       0.92  2279    1
bP   6.60   1.47       4.23       8.83  2123    1
bV  -5.30   1.19      -7.10      -3.41  2383    1
bA   3.44   1.26       1.35       5.30  2149    1
bPA -2.97   1.38      -5.11      -0.78  2163    1
```

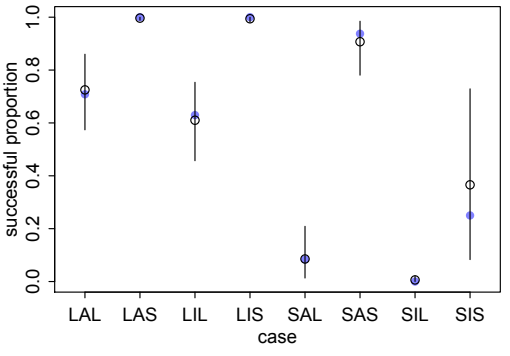
So when the pirate is both large and adult, the log-odds are smaller. This seems like a weird result. Shouldn't being large and adult help? It does. But the main effects for size and age are also turned on, when an individual is both large and adult. So it's hard to understand what the model is saying, without computing predictions case by case.

So now for some plots. We'll use a prediction ensemble. I'll show the probability scale here. You can easily modify the code, following the example from the previous section, to plot predicted counts instead.

R code 8.19

```
p <- ensemble(m2.stan,m1.stan)$link

p.mean <- apply( p , 2 , mean )
p.PI <- apply( p , 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 probability of success
points( 1:8 , p.mean )
for ( i in 1:8 ) lines( c(i,i) , p.PI[,i] )
```

And here are the results.



Predictions, as indicated by the empty circle in each case, have gotten closer to the observed. This is especially true for LAL, LIL, and SIS. The interaction effect allows the model to cope with the ceiling effect better, meaning it doesn't have to use the main effect estimates to predict both middle-level success and topped-out success.

10H4. (a) Before we start fitting, this is a classic case in which the raw predictor variables have an inconveniently large scale. This tends to mess up `map` more than it does `map2stan`. Still, standardizing the predictors before fitting will help both.

R code 8.20

```
data(salamanders)
d <- salamanders

# function to make this more convenient
stdz <- function(x) (x-mean(x))/sd(x)

d$PCTCOVER <- stdz(d$PCTCOVER)
d$FORESTAGE <- stdz(d$FORESTAGE)
```

To fit the Poisson regression of salamander counts against percent cover:

R code 8.21

```
f <- alist(
    SALAMAN ~ dpois( lambda ),
    log(lambda) <- a + bc*PCTCOVER,
    a ~ dnorm(0,10),
    bc ~ dnorm(0,1)
)
m1 <- map( f , data=d )
m1.stan <- map2stan( f , data=d , warmup=1000 , iter=1e4 )

precis(m1)
precis(m1.stan)
```

```
   Mean StdDev 5.5% 94.5%
a  0.46   0.15 0.21  0.70
bc 1.12   0.18 0.83  1.41

   Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  0.43   0.16       0.18       0.68  2405    1
bc 1.15   0.19       0.84       1.44  2404    1
```

Looks like there's a tiny skew in this posterior. But not so much as to cause any differences in effective inference. If you don't believe, inspect posterior predictions for both models.

(You could also fit the intercept-only model and compare. It'll do far worse than this model.)

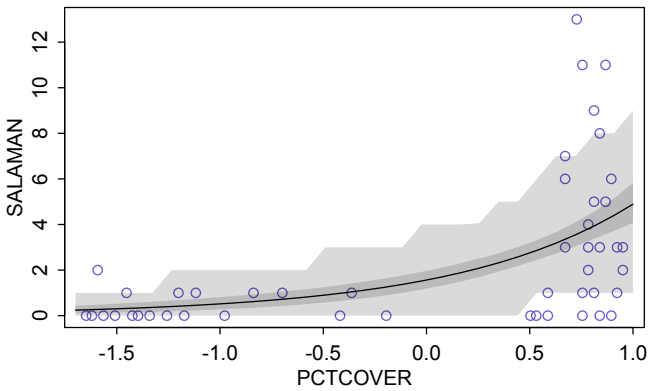
So what about those estimates? The estimates for `bc` is positive, which means counts are predicted to increase with cover. But by how much? Hard to say, until you convert back to the count scale. So let's plot the predictions, now. Here's the code to do it. The approach is, as far scripting goes, similar to the binomial model, because `link` and `sim` automatically detect the link function you specified in the model fitting. But if you had to do it yourself, you would use `exp` now instead of `logistic` as the inverse link.

R code 8.22

```
# define values to compute over
pctcover.seq <- seq(from=-1.7,to=1,length.out=30)
```

```
# lambda calcs
lambda <- link( m1.stan , data=list(PCTCOVER=pctcover.seq) )
lambda.mean <- apply(lambda,2,mean)
lambda.PI <- apply(lambda,2,PI)

# outcome calcs
n <- sim( m1.stan , data=list(PCTCOVER=pctcover.seq) )
n.PI <- apply(n,2,PI)

# plotten sie!
plot( SALAMAN ~ PCTCOVER , data=d , col="slateblue" )
lines( pctcover.seq , lambda.mean )
shade( lambda.PI , pctcover.seq )
shade( n.PI , pctcover.seq )
```

And here's the resulting plot:



There is certainty a relationship between cover and density: the more cover, the more salamanders on average. But the observed data actually seem to show two clusters of responses, with plots with cover below 60% having similar low counts, with no obvious increasing trend, and those above 70% cover having much higher counts. The model doesn't really predict this observation. There also may be more dispersion in the high-cover sites.

Probably a good place for a gamma-Poisson or multilevel model, really. Better yet, some theory would help nominate over variables to explain the dispersion.

(b) Let's inspect a model with both cover and forest age.

R code 8.23

```
f2 <- alist(
    SALAMAN ~ dpois( lambda ),
    log(lambda) ~ a + bc*PCTCOVER + bf*FORESTAGE,
    a ~ dnorm(0,10),
    bc ~ dnorm(0,1),
    bf ~ dnorm(0,1)
)
m2.stan <- map2stan( f2 , data=d , warmup=1000 , iter=1e4 )
precis(m2.stan)
```

```
   Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
```

```
a  0.43   0.16       0.19       0.68  2733    1
bc 1.14   0.20       0.82       1.46  2650    1
bf 0.00   0.10      -0.16       0.16  3259    1
```

Notice that the estimate for `bf` is nearly zero, with small interval around it. There isn't much association between forest age and salamander density, while also controlling for 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. This is probably because the two predictors are correlated, but the predictive ability of forest age arises just through its correlation with percent cover. That is to say, older forests also tend to have more cover. But it's cover that the salamanders are seeking, and so they are found more in places with more cover. This is a misleading correlation, much like the Waffle House and divorce rate thing from Chapter 5.

9. Chapter 11 Solutions

11E1. 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.

11E2. 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*.

11E3. 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.

11E4. 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.

11M1. 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 9.1

```
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 9.2
```
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 9.3
```
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:

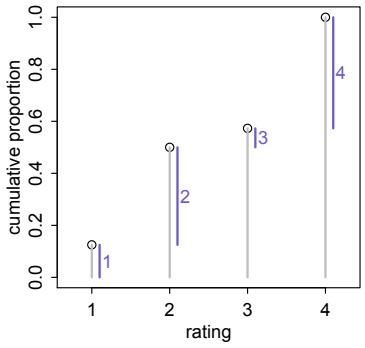
R code 9.4
```
log(p/(1-p))
```

```
[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.

11M2. Using the vectors `p` and `q` from the previous problem:

R code 9.5
```
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" )
```



11M3. 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.

11H1. The problem left some freedom to specify your own priors. So if you used different priors than those in the solution below, don't panic. Your priors may be fine. If you get different inferences however, try to figure out if it was due to the different choice of priors. If you prefer your priors, be sure to figure out a principled argument for them.

Here's a very simple Poisson model predicting `deaths` using only `femininity` as a predictor. I'm going to standardize `femininity`, for the usual reasons.

R code 9.6

```
library(rethinking)
data(Hurricanes)
d <- Hurricanes
d$fmnnty_std <- ( d$femininity - mean(d$femininity) )/sd(d$femininity)

m1 <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty,
        a ~ dnorm(0,10),
        bf ~ dnorm(0,1)
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std
    ) , chains=4 )
```

Before fitting the intercept-only model to compare, let's peek at the parameters:

R code 9.7

```
precis(m1)
```

```
   Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  3.00   0.02       2.96       3.04  1812    1
bf 0.24   0.03       0.20       0.28  2090    1
```

So this model seems to think there is a reliably positive association between femininity of the hurricane names and deaths.

Here's the intercept only model:

R code 9.8

```
m0 <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a,
        a ~ dnorm(0,10)
    ),
    data=list(deaths=d$deaths) , chains=4 )
```

Let's compare the two models:

R code 9.9

```
compare(m0,m1)
```

```
     WAIC pWAIC dWAIC weight      SE    dSE
m1 4403.2 135.2   0.0      1  993.05     NA
m0 4439.8  80.8  36.6      0 1070.71 143.15
```

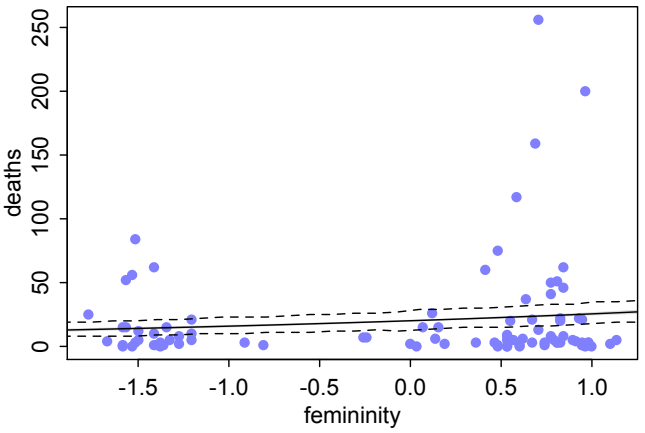
That's pretty strong support for the model that includes femininity of names.

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 9.10

```
# plot raw data
plot( d$fmnnty_std , d$deaths , pch=16 ,
    col=rangi2 , xlab="femininity" , ylab="deaths" )

# compute model-based trend
pred_dat <- list( fmnnty=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$fmnnty , lambda.mu )
shade( lambda.PI , pred_dat$fmnnty )

# 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$fmnnty , deaths_sim.PI[1,] , lty=2 )
lines( pred_dat$fmnnty , 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).

11H2. 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 9.11

```
library(rethinking)
data(Hurricanes)
d <- Hurricanes
d$fmnnty_std <- ( d$femininity - mean(d$femininity) )/sd(d$femininity)

m3 <- map2stan(
    alist(
        deaths ~ dgampois(lambda,scale),
        log(lambda) <- a + bf*fmnnty,
        a ~ dnorm(0,10),
        bf ~ dnorm(0,1),
        scale ~ dcauchy(0,1) # dexp would also be fine here
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std
    ) , chains=4 )
```

Inspect the marginal posterior distributions of the parameters:

R code 9.12

```
precis(m3)
```

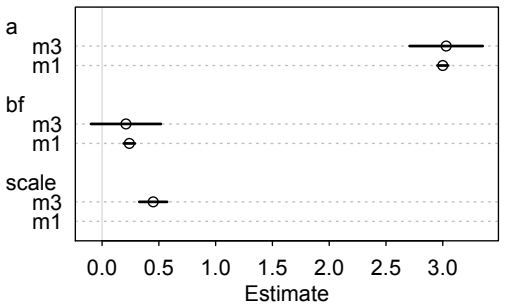
```
      Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     3.03   0.16       2.76       3.29  2396    1
bf    0.21   0.16      -0.03       0.47  2526    1
```

```
scale 0.45   0.06       0.36       0.55  2653    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`:

R code 9.13

```
plot(coeftab(m1,m3))
```



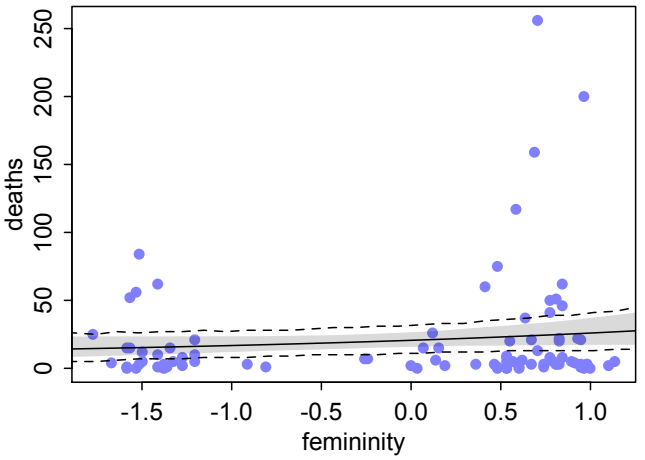
Model `m3` has nearly the same posterior means for the intercept and slope `bf`, but much more uncertainty.

How does this translate into predictions? To find out, let's do the plotting from the previous problem over again, using model `m3` now:

R code 9.14

```
# plot raw data
plot( d$fmnnty_std , d$deaths , pch=16 ,
    col=rangi2 , xlab="femininity" , ylab="deaths" )

# compute model-based trend
pred_dat <- list( fmnnty=seq(from=-2,to=1.5,length.out=30) )
lambda <- link(m3,data=pred_dat)
lambda.mu <- apply(lambda,2,mean)
lambda.PI <- apply(lambda,2,PI)

# superimpose trend
lines( pred_dat$fmnnty , lambda.mu )
shade( lambda.PI , pred_dat$fmnnty )

# compute sampling distribution
deaths_sim <- sim(m3,data=pred_dat)
deaths_sim.PI <- apply(deaths_sim,2,PI)

# superimpose sampling interval as dashed lines
lines( pred_dat$fmnnty , deaths_sim.PI[1,] , lty=2 )
lines( pred_dat$fmnnty , deaths_sim.PI[2,] , lty=2 )
```

This is the new result:



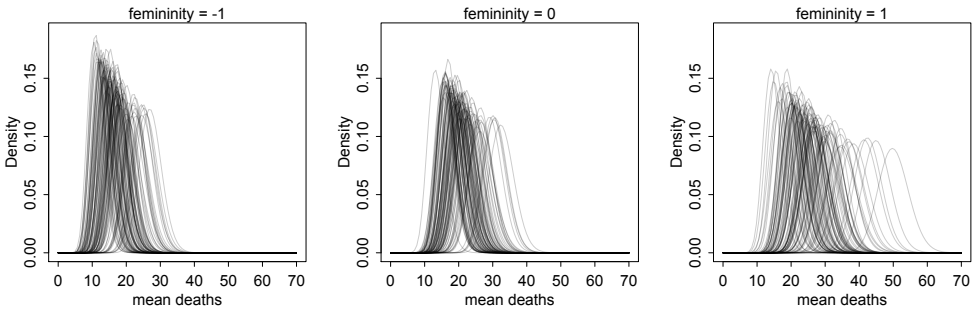
There is more uncertainty now about the relationship, and the prediction interval is wider. But the predictions are still terrible.

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 9.15

```
post <- extract.samples(m3)

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 12.

You might be curious how `m3` compares to `m1`, in terms of WAIC. If so, take a look. You'll find that the effective number of parameters for `m3` is very very large. It does fit the data better. But it's also very prone to overfitting. This is also a context in which DIC and WAIC very much disagree about model ranking. In my experience, Poisson models exhibit common disagreement between DIC and WAIC. Unlike DIC, WAIC is more data dependent. Exactly where the observations lie can make a big difference for WAIC but not for DIC. As a result, it is possible to get large disagreement between them, as in this example.

11H3. 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.

For example, here's a model that interacts `min_pressure` with `femininity`. I will use homogenous Poisson model here, for the moment, but you could go ahead and use gamma-Poisson models as well.

R code 9.16

```
# standardize new predictor
d$min_pressure_std <- (d$min_pressure - mean(d$min_pressure))/sd(d$min_pressure)

# interaction model
m_f_p_fp <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty + bp*minpress +
                        bfp*fmnnty*minpress,
        a ~ dnorm(0,10),
        c(bf,bp,bfp) ~ dnorm(0,1)
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std,
        minpress=d$min_pressure_std
    ) , chains=4 )
```

And let's go ahead and fit the same model without the interaction, with with the two main effects, for comparison:

R code 9.17

```
m_f_p <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty + bp*minpress,
        a ~ dnorm(0,10),
        c(bf,bp) ~ dnorm(0,1)
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std,
        minpress=d$min_pressure_std
    ) , chains=4 )
```

Be sure to check that both models converged and sampled correctly. Now comparing the two with WAIC:

R code 9.18

```
compare(m_f_p,m_f_p_fp)
```

```
           WAIC pWAIC dWAIC weight      SE   dSE
m_f_p    3490.0 211.3   0.0      1 1112.58    NA
m_f_p_fp 3571.8 270.9  81.8      0 1136.81 46.12
```

Well the model without the interaction seems to have a lot more support. Let's also look at the interaction estimate itself:

R code 9.19

```
precis(m_f_p_fp)
```

```
     Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a    2.72   0.03       2.68       2.77  1512    1
bf   0.26   0.03       0.21       0.31  1469    1
bp  -0.74   0.02      -0.77      -0.70  1430    1
bfp  0.07   0.03       0.03       0.11  1583    1
```

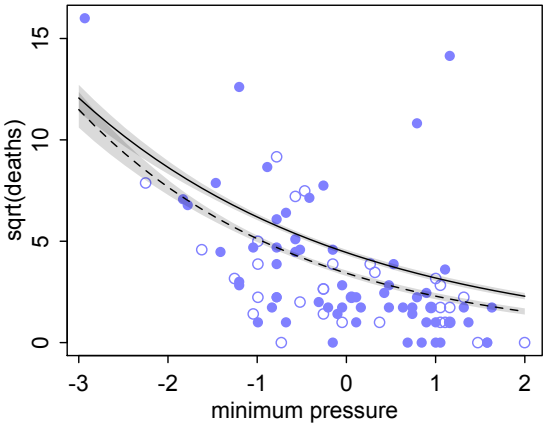
The interaction coefficient is positive. Interpreting this is a bit tricky, because hurricanes get stronger as their minimum pressure gets *lower*. So it makes sense for the main effect to be negative: storms with *larger* minimum pressure cause fewer deaths. Does it also make sense for the interaction 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 9.20

```
minpress_seq <- seq(from=-3,to=2,length.out=30)

# 'masculine' storms
d_pred <- data.frame(
    fmnnty=-1,
    minpress=minpress_seq )
lambda_m <- link(m_f_p_fp,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(
```

```
    fmnnty=1,
    minpress=minpress_seq )
lambda_f <- link(m_f_p_fp,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( d$min_pressure_std , sqrt(d$deaths) ,
    pch=ifelse(d$fmnnty_std>0,16,1) , col=rangi2 ,
    xlab="minimum pressure" , ylab="sqrt(deaths)" )
lines( minpress_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , minpress_seq )
lines( minpress_seq , sqrt(lambda_f.mu) , lty=1 )
shade( sqrt(lambda_f.PI) , minpress_seq )
```



The dashed trend is "masculine" storms, and the solid trend is "feminine" storms. So you can see clearly here that the interaction model expects feminine storms to be a little more deadly, but the difference between masculine and feminine storms actually decreases as pressure drops. This is not in agreement with the hypothesis.

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. Here are the new models and their WAIC comparison:

R code 9.21

```
# standardize predictor
d$damage_std <- (d$damage_norm - mean(d$damage_norm))/sd(d$damage_norm)

# interaction model
m_f_d_fd <- map2stan(
    alist(
        deaths ~ dpois(lambda),
```

```
        log(lambda) <- a + bf*fmnnty + bd*damage +
                        bfd*fmnnty*damage,
        a ~ dnorm(0,10),
        c(bf,bd,bfd) ~ dnorm(0,1)
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std,
        damage=d$damage_std
    ) , chains=4 )
m_f_d <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty + bd*damage,
        a ~ dnorm(0,10),
        c(bf,bd) ~ dnorm(0,1)
    ),
    data=list(
        deaths=d$deaths,
        fmnnty=d$fmnnty_std,
        damage=d$damage_std
    ) , chains=4 )
```

Be sure to check that both models converged and sampled correctly. Now comparing the two with WAIC:

```
compare( m_f_d , m_f_d_fd )
```

R code 9.22

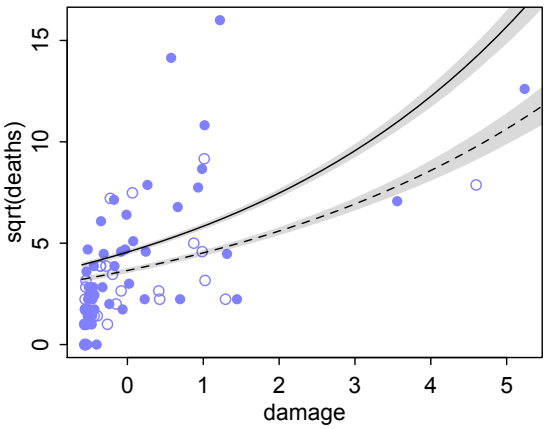
```
           WAIC pWAIC dWAIC weight     SE   dSE
m_f_d    3216.7 130.4     0      1 794.23    NA
m_f_d_fd 3234.8 147.3    18      0 796.87 29.57
```

Hm, still not much support for the interaction. Let's look at the counterfactual predictions again:

```
damage_seq <- seq(from=-0.6,to=5.5,length.out=30)

# 'masculine' storms
d_pred <- data.frame(
    fmnnty=-1,
    damage=damage_seq )
lambda_m <- link(m_f_d_fd,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(
    fmnnty=1,
    damage=damage_seq )
lambda_f <- link(m_f_d_fd,data=d_pred)
lambda_f.mu <- apply(lambda_f,2,mean)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( d$damage_std , sqrt(d$deaths) ,
```

R code 9.23

```
    pch=ifelse(d$fmnnty_std>0,16,1) , col=rangi2 ,
    xlab="damage" , ylab="sqrt(deaths)" )
lines( damage_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , damage_seq )
lines( damage_seq , sqrt(lambda_f.mu) , lty=1 )
shade( sqrt(lambda_f.PI) , damage_seq )
```



Now we see the anticipated relationship: feminine storms (solid trend) are both stronger at all damage values, and the difference from masculine storms increases with damage. But the effect is very small. Look at the coefficients to see why:

R code 9.24

```
precis(m_f_d_fd)
```

```
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a   2.81   0.03       2.77       2.86  2070    1
bf  0.22   0.03       0.18       0.27  1716    1
bd  0.46   0.01       0.44       0.48  1981    1
bfd 0.03   0.01       0.01       0.05  1560    1
```

The interaction coefficient is quite small. It is reliably positive, but tiny. So it doesn't make much difference, until storm damage grows massive.

The models including `damage_norm` do better than those with `min_pressure`. Check the WAIC comparison for yourself. So for a final task here (you could have done and still do much more though), let's consider a gamma-Poisson versions of the damage model:

R code 9.25

```
dat <- list(
    deaths=d$deaths,
    fmnnty=d$fmnnty_std,
    damage=d$damage_std )
mgP_f_d_fd <- map2stan(
    alist(
        deaths ~ dgampois(lambda,scale),
        log(lambda) <- a + bf*fmnnty + bd*damage +
                       bfd*fmnnty*damage,
        a ~ dnorm(0,10),
        c(bf,bd,bfd) ~ dnorm(0,1),
        scale ~ dcauchy(0,1)
```

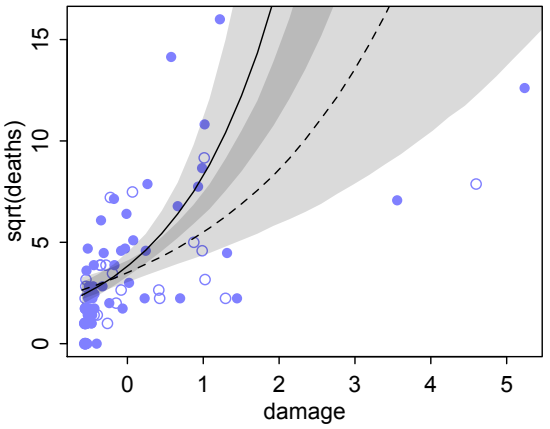
```
    ),
    data=dat , chains=4 )
precis(mgP_f_d_fd)
```

```
      Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     2.60   0.13       2.39       2.80  2819    1
bf    0.09   0.13      -0.11       0.30  2380    1
bd    1.25   0.22       0.91       1.59  2436    1
bfd   0.31   0.21      -0.01       0.65  2093    1
scale 0.69   0.10       0.51       0.83  2145    1
```

This model finds a stronger relationship with damage and a stronger interaction, but both estimates are now much more variable. This is what we should expect from a gamma-Poisson model. The gamma-distributed rates allow a wider range of parameter values to produce similar predictions. What do the predictions look like now?

R code 9.26

```
damage_seq <- seq(from=-0.6,to=5.5,length.out=30)

# 'masculine' storms
d_pred <- data.frame(
    fmnnty=-1,
    damage=damage_seq )
lambda_m <- link(mgP_f_d_fd,data=d_pred)
lambda_m.mu <- apply(lambda_m,2,median)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame(
    fmnnty=1,
    damage=damage_seq )
lambda_f <- link(mgP_f_d_fd,data=d_pred)
lambda_f.mu <- apply(lambda_f,2,median)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( d$damage_std , sqrt(d$deaths) ,
    pch=ifelse(d$fmnnty_std>0,16,1) , col=rangi2 ,
    xlab="damage" , ylab="sqrt(deaths)" )
lines( damage_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , damage_seq )
lines( damage_seq , sqrt(lambda_f.mu) , lty=1 )
shade( sqrt(lambda_f.PI) , damage_seq )
```



This plot looks a little strange, because the overlapping prediction regions for masculine and feminine storms create that dark wedge in the middle. But again, feminine storms are more deadly at all damage amounts, and the difference grows with damage. But note that now predictions are all over the place, especially for masculine storms. This is a result of allowing the gamma-distributed variation.

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, almost anything could be creating the correlation, weak as it is, between femininity and deaths. Why? Because there are many unmeasured confounds. So analyses like this are heavily dependent upon how plausible you find the hypothesis. I personally find it highly implausible that people refuse to evacuate because they imagine "Andrea" is not dangerous while "Andrew" is. If you've even been in a hurricane evacuation, you might agree with me. But other people, including the authors of the study, apparently do find this idea plausible.

Regardless, the data aren't going to resolve the issue.

11H4. All that is needed here is to pre-transform the predictor `damage_norm` to log scale and then replace the original version in the model or models from the previous problem. I'll fit both a main effects model and interaction model, comparing them to the analogous models that use `damage_norm` on the original scale.

One subtle issue with log transforming a predictor like this is whether or not standardize it after the transformation. I like standardized predictors, because they make fitting and interpretation a little easier. More importantly, it makes setting informative (regularizing) priors easier. But it's rarely essential with these simple models. Here, I'll transform the original `damage_norm` and then standardize it, but if you didn't standardize, that's fine. Just be cautious about the scale of your prior.

R code 9.27

```
d$log_damage <- log(d$damage_norm)
d$log_damage_std <- (d$log_damage - mean(d$log_damage))/sd(d$log_damage)

dat <- list(
    deaths=d$deaths,
    fmnnty=d$fmnnty_std,
    log_damage=d$log_damage_std )

# capital 'D' here denotes log scale
m_f_D_fD <- map2stan(
```

```
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty + bD*log_damage +
                       bfD*fmnnty*log_damage,
        a ~ dnorm(0,10),
        c(bf,bD,bfD) ~ dnorm(0,1)
    ),
    data=dat , chains=4 )

m_f_D <- map2stan(
    alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf*fmnnty + bD*log_damage,
        a ~ dnorm(0,10),
        c(bf,bD) ~ dnorm(0,1)
    ),
    data=dat , chains=4 )
```

Now if you still have the models fit from the previous problem, we can compare the four models:

```
compare( m_f_d , m_f_d_fd , m_f_D , m_f_D_fD )
```

R code 9.28

```
           WAIC pWAIC  dWAIC weight     SE    dSE
m_f_D_fD 2095.6 108.0    0.0      1 441.44     NA
m_f_D    2137.1  98.4   41.5      0 451.55  47.82
m_f_d    3216.7 130.4 1121.1      0 794.23 432.13
m_f_d_fd 3234.8 147.3 1139.2      0 796.87 434.92
```

The log-damage models are kickin' it. And the interaction model comes out ahead of the main effect model. Let's look at it's coefficients, before plotting some counterfactual predictions:

```
precis(m_f_D_fD)
```

R code 9.29

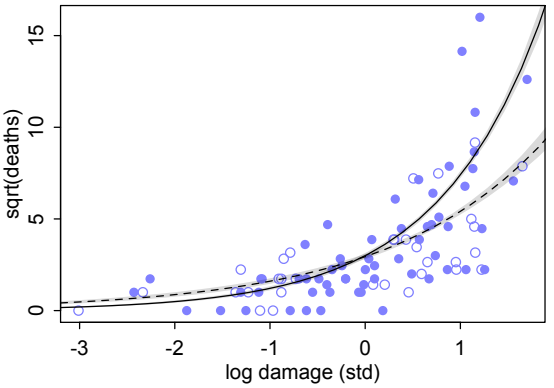
```
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a   2.18   0.04       2.11       2.24  1382    1
bf  0.01   0.04      -0.06       0.08  1193    1
bD  1.51   0.04       1.45       1.57  1211    1
bfD 0.30   0.04       0.23       0.36  1300    1
```

The main effect of femininity now straddles zero, but that interaction is reliably positive. As always, some plotting is really needed to understand the implications:

```
log_damage_seq <- seq(from=-3.2,to=2,length.out=30)

# 'masculine' storms
d_pred <- data.frame(
    fmnnty=-1,
    log_damage=log_damage_seq )
lambda_m <- link( m_f_D_fD , data=d_pred )
lambda_m.mu <- apply(lambda_m,2,median)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame(
```

R code 9.30

```
    fmnnty=1,
    log_damage=log_damage_seq )
lambda_f <- link( m_f_D_fD , data=d_pred )
lambda_f.mu <- apply(lambda_f,2,median)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( d$log_damage_std , sqrt(d$deaths) ,
    pch=ifelse(d$fmnnty_std>0,16,1) , col=rangi2 ,
    xlab="log damage (std)" , ylab="sqrt(deaths)" )
lines( log_damage_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , log_damage_seq )
lines( log_damage_seq , sqrt(lambda_f.mu) , lty=1 )
shade( sqrt(lambda_f.PI) , log_damage_seq )
```



As before, solid trend is "feminine" storms and dashed is "masculine" storms. In this perspective, it's only at the highest damage levels that there is a difference between masculine and feminine storms. The feminine storms did do more damage, on average. But there's not much difference for the majority of storms.

You might wonder what the gamma-Poisson interaction model is like. So let's try it:

R code 9.31

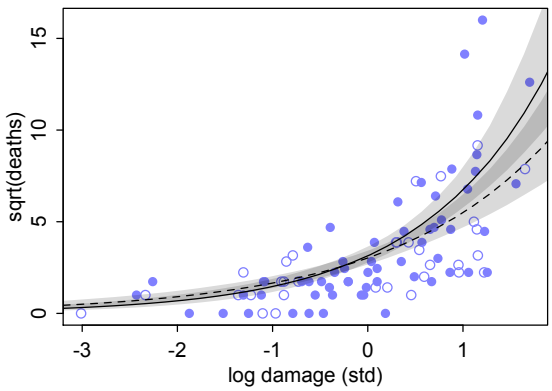
```
mgP_f_D_fD <- map2stan(
    alist(
        deaths ~ dgampois(lambda,scale),
        log(lambda) <- a + bf*fmnnty + bD*log_damage +
                       bfD*fmnnty*log_damage,
        a ~ dnorm(0,10),
        c(bf,bD,bfD) ~ dnorm(0,1),
        scale ~ dcauchy(0,1)
    ),
    data=dat , chains=4 )
precis(mgP_f_D_fD)
```

```
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a      2.25   0.11       2.07       2.43  2852    1
bf     0.04   0.12      -0.15       0.23  2623    1
bD     1.37   0.12       1.18       1.56  2471    1
bfD    0.17   0.13      -0.02       0.38  2852    1
scale  1.06   0.17       0.78       1.32  2772    1
```

As expected, the standard deviations of the coefficients have all increased, for the usual reason: allowing gamma-distributed heterogeneity makes many more parameter values plausible. What do the predictions look like? Let's see:

R code 9.32

```
# 'masculine' storms
d_pred <- data.frame(
    fmnnty=-1,
    log_damage=log_damage_seq )
lambda_m <- link( mgP_f_D_fD , data=d_pred )
lambda_m.mu <- apply(lambda_m,2,median)
lambda_m.PI <- apply(lambda_m,2,PI)

# 'feminine' storms
d_pred <- data.frame(
    fmnnty=1,
    log_damage=log_damage_seq )
lambda_f <- link( mgP_f_D_fD , data=d_pred )
lambda_f.mu <- apply(lambda_f,2,median)
lambda_f.PI <- apply(lambda_f,2,PI)

# plot
plot( d$log_damage_std , sqrt(d$deaths) ,
    pch=ifelse(d$fmnnty_std>0,16,1) , col=rangi2 ,
    xlab="log damage (std)" , ylab="sqrt(deaths)" )
lines( log_damage_seq , sqrt(lambda_m.mu) , lty=2 )
shade( sqrt(lambda_m.PI) , log_damage_seq )
lines( log_damage_seq , sqrt(lambda_f.mu) , lty=1 )
shade( sqrt(lambda_f.PI) , log_damage_seq )
```



Those four feminine super-storms on the righthand side are still pulling the feminine trend upwards, but the gamma-distributed variation accounts for some of their distance from the mean, allowing the median prediction (the solid trend) to rest lower than in the pure (homogenous) Poisson model.

11H5. 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. But I'm going to make a new variable `female`, so we can focus on females, like the question asks. You can do it either way.

R code 9.33

```
library(rethinking)
data(Trolley)
d <- Trolley
d$female <- 1 - d$male
```

And we can add a main effect of `female` and an interaction with `contact`. To fit the model, using `map`:

R code 9.34

```
m1a <- map(
    alist(
        response ~ dordlogit( phi , c(a1,a2,a3,a4,a5,a6) ),
        phi <- bA*action + bI*intention + bC*contact +
            bAI*action*intention + bCI*contact*intention +
            bf*female + bfC*female*contact,
        c(a1,a2,a3,a4,a5,a6) ~ dnorm(0,10),
        c(bA,bI,bC,bAI,bCI,bf,bfC) ~ dnorm(0,10)
    ) ,
    data=d ,
    start=list(a1=-2,a2=-1.5,a3=-0.5,a4=0,a5=1,a6=1.5) )
```

And now I'll show the same model via `map2stan`. The only trick to be concerned with here is that `Stan` does not allow a variable or parameter with the name `case`, so we'll need to pass in a trimmed data list. Also be prepared for this to take more time than previous models in the book. There's a lot of data, and the likelihood is more complicated. Still, Stan does a great job and can handle vastly more complex models than this one.

R code 9.35

```
dat <- list(
    response=d$response,
    action=d$action,
    intention=d$intention,
    contact=d$contact,
    female=d$female
)
m1a_stan <- map2stan(
    alist(
        response ~ dordlogit( phi , cutpoints ),
        phi <- bA*action + bI*intention + bC*contact +
            bAI*action*intention + bCI*contact*intention +
            bf*female + bfC*female*contact,
        cutpoints ~ dnorm(0,10),
        c(bA,bI,bC,bAI,bCI,bf,bfC) ~ dnorm(0,10)
    ) ,
    data=dat , chains=3 , cores=3 ,
    start=list(cutpoints=c(-2,-1.5,-0.5,0,1,1.5)) )
```

I'll work with the `map` fit, but it makes no difference in this example. Let's glance at the estimates:

R code 9.36

```
precis( m1a )
```

```
     Mean StdDev  5.5% 94.5%
a1  -2.95   0.06 -3.04 -2.86
```

```
a2  -2.25   0.05 -2.34 -2.17
a3  -1.65   0.05 -1.73 -1.57
a4  -0.59   0.05 -0.67 -0.52
a5   0.10   0.05  0.02  0.17
a6   1.02   0.05  0.94  1.10
bA  -0.48   0.05 -0.56 -0.39
bI  -0.28   0.06 -0.38 -0.19
bC  -0.43   0.08 -0.56 -0.30
bAI -0.45   0.08 -0.58 -0.32
bCI -1.29   0.10 -1.45 -1.14
bf  -0.62   0.04 -0.68 -0.55
bfC  0.21   0.09  0.07  0.35
```

Inspect the estimates for `bf` (the main effect of being female on cumulative log-odds) and `bfC` (the interaction effect of being both female and in a contact scenario). The main effect is negative, -0.62, and the interaction is positive, 0.21.

It will be easier to make the comparison between females and males, if we compute expected log-odds for both females and males, both with and without contact treatment. Let's just write our own link function for the linear model `phi`. You could also use `link()` as usual, but this way I get to show you another example of doing something for yourself.

R code 9.37

```
post <- extract.samples(m1a)
phi.link <- function(A,I,C,f) {
    post$bA*A + post$bI*I + post$bC*C + post$bAI*A*I +
        post$bCI*C*I + post$bf*f + post$bfC*f*C
}
```

Now we compute posterior distributions for females and males, both with and without contact:

R code 9.38

```
female.noC <- phi.link(0,0,0,1)
female.C <- phi.link(0,0,1,1)
male.noC <- phi.link(0,0,0,0)
male.C <- phi.link(0,0,1,0)
```

And then we can take the difference between contact and no-contact to see how much both females and males *change* when contact is added:

R code 9.39

```
female.diff <- female.C - female.noC
male.diff <- male.C - male.noC
```

For a quick comparison, use `precis`:

R code 9.40

```
precis( data.frame( female.diff , male.diff ) )
```

```
             Mean StdDev |0.89 0.89|
female.diff -0.22   0.09 -0.35 -0.08
male.diff   -0.43   0.08 -0.56 -0.29
```

So females are actually *less* bothered by contact, in the sense that their log-odds decline less than that of males.

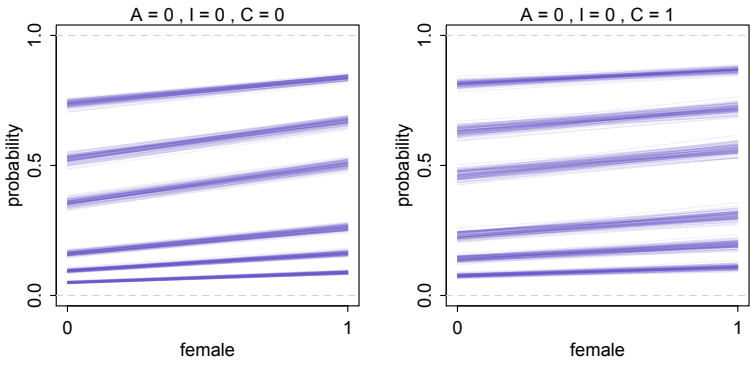
So we have to conclude that females are less bothered by contact than are males. But on the other hand, females are more bothered by all of the principles than are males, on average. So it may be that females are just hitting a floor effect here, such they can't be as bothered as males are, because males start out rating everything as more permissible, on average.

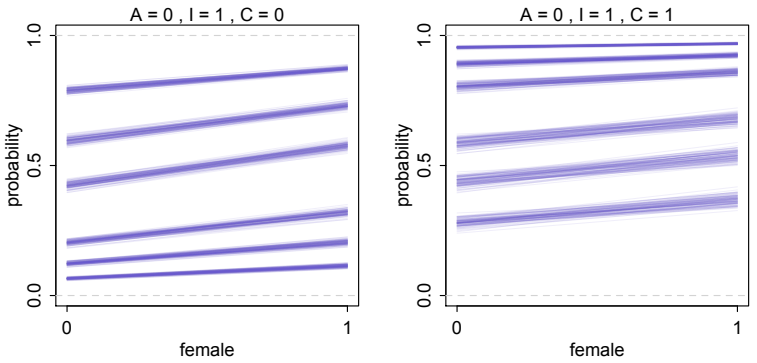
It'd also help to plot the predictions, like the example in Chapter 12. Here's the code to produce a prediction plot for `action==0`, `intention==0`, and `contact==0`. You can change the `kA`, `kI`, and `kC` values to produce other plots.

R code 9.41

```
post <- extract.samples(m1a)
kA <- 0
kC <- 1
kI <- 1
kf <- 0:1
plot( 1 , 1 , type="n" , xlab="female" , ylab="probability" ,
    xlim=c(0,1) , ylim=c(0,1) , xaxp=c(0,1,1) , yaxp=c(0,1,2) )
for ( s in 1:100 ) {
    p <- post[s,]
    ak <- as.numeric( c(p$a1,p$a2,p$a3,p$a4,p$a5,p$a6) )
    phi <- p$bf*kf + p$bfC*kf*kC + p$bA*kA + p$bI*kI +
        p$bC*kC + p$bAI*kA*kI + p$bCI*kC*kI
    pk <- pordlogit( 1:6 , phi=phi , a=ak )
    for ( i in 1:6 )
        lines( 0:1 , pk[,i] ,
            col=col.alpha("slateblue",0.1) , lwd=0.5 )
}
abline( h=0 , lty=2 , col="lightgray" )
abline( h=1 , lty=2 , col="lightgray" )
mtext( paste("A =",kA,", I =",kI,", C =",kC ) , 3 )
```

Below, I show some plots produced in this way. You can see that, overall, males (0 on the horizontal axis) rate everything as more permissible. But the difference is smaller for the cases in which contact is equal to 1 (righthand plots). That is, if you compare the lefthand plot to the righthand plot in each row, the slopes of the lines becomes more flat, indicating a smaller difference between females and males.





You could fit more models, if you are concerned about overfitting here. But after doing all of that, and model-averaging the predictions, you'll end up at the same destination as above. But you don't have to take my word for it. It's entirely possible that I missed something. But even if I didn't, you'll learn something from the exercise.

11H6. Read in the data:

R code 9.42

```
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 9.43

```
d$loghours <- log(d$hours)
m2a <- map2stan(
    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(0,10),
        c(bp0,bc0,bp,bc) ~ dnorm(0,1)
    ) ,
    data=d , iter=1e4 , chains=2 )
```

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 9.44
```
precis(m2a)
```

```
     Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a0   0.77   0.56      -0.08       1.71  3765    1
a   -2.30   0.15      -2.54      -2.07  3705    1
bp0 -1.01   0.27      -1.44      -0.58  3655    1
bc0  1.01   0.56       0.19       1.90  4082    1
bp   0.67   0.04       0.61       0.74  3720    1
bc   0.56   0.09       0.42       0.72  4630    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:

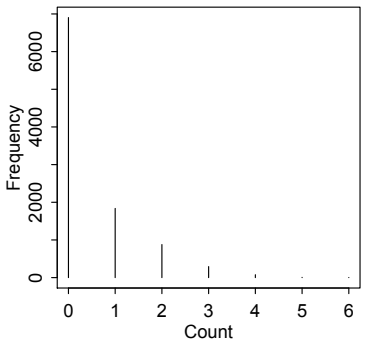
R code 9.45
```
zip_link <- link(m2a)
str(zip_link)
```

```
List of 2
 $ p : num [1:1000, 1:250] 0.506 0.709 0.333 0.589 0.419 ...
 $ mu: num [1:1000, 1:250] 4.1 4.26 3.56 4.02 3.92 ...
```

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 9.46
```
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 9.47
```
fish_sim <- sim(m2a)
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 9.48

```
# 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( m2a , 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] 0.1099065

        5%        94%
0.08122458 0.13897464
```

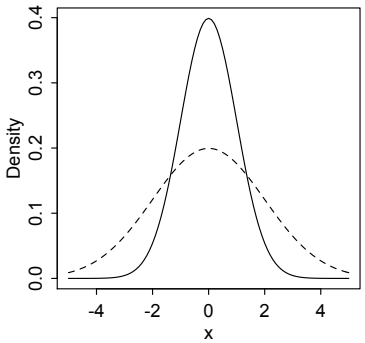
What this means is that one person is expected to extract on average 0.11 fish per hour, with the interval shown. This estimate allows for the zero-inflation, so it accounts for the probability that the person is not fishing at all, depressing the expected value.

10. Chapter 12 Solutions

12E1. 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 10.1

```
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.

12E2. 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:

$$\begin{aligned} y_i &\sim \text{Binomial}(1, p_i) \\ \text{logit}(p_i) &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 10) \\ \sigma_\alpha &\sim \text{HalfCauchy}(0, 1) \end{aligned}$$

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.

12E3. 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:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma &\sim \text{HalfCauchy}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 10) \\ \sigma_\alpha &\sim \text{HalfCauchy}(0, 1) \end{aligned}$$

12E4. You can just copy the answer from problem **12E2** and swap out the binomial likelihood for a Poisson, taking care to change the link function from logit to log:

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i) \\ \log(\lambda_i) &= \alpha_{\text{GROUP}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_\alpha) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 10) \\ \sigma_\alpha &\sim \text{HalfCauchy}(0, 1) \end{aligned}$$

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.

12E5. 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:

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i) \\ \log(\lambda_i) &= \alpha_{\text{GROUP}[i]} + \alpha_{\text{DAY}[i]} + \beta x_i \\ \alpha_{\text{GROUP}} &\sim \text{Normal}(\bar{\alpha}, \sigma_{\text{GROUP}}) \\ \alpha_{\text{DAY}} &\sim \text{Normal}(0, \sigma_{\text{DAY}}) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\alpha} &\sim \text{Normal}(0, 10) \\ \sigma_{\text{GROUP}} &\sim \text{HalfCauchy}(0, 1) \\ \sigma_{\text{DAY}} &\sim \text{HalfCauchy}(0, 1) \end{aligned}$$

Or you can just pull the mean intercept out of both priors and put it in the linear model:

$$
\begin{aligned}
y_i &\sim \text{Poisson}(\lambda_i) \\
\log(\lambda_i) &= \bar{\alpha} + \alpha_{\text{GROUP}[i]} + \alpha_{\text{DAY}[i]} + \beta x_i \\
\alpha_{\text{GROUP}} &\sim \text{Normal}(0, \sigma_{\text{GROUP}}) \\
\alpha_{\text{DAY}} &\sim \text{Normal}(0, \sigma_{\text{DAY}}) \\
\beta &\sim \text{Normal}(0, 1) \\
\bar{\alpha} &\sim \text{Normal}(0, 10) \\
\sigma_{\text{GROUP}} &\sim \text{HalfCauchy}(0, 1) \\
\sigma_{\text{DAY}} &\sim \text{HalfCauchy}(0, 1)
\end{aligned}
$$

These are exactly the same model. Although as you'll see later in Chapter 13, these different forms might be more or less efficient in sampling.

12M1. Let's load the reed frog data and look at the relevant variables:

R code 10.2

```
library(rethinking)
data(reedfrogs)
d <- reedfrogs
str(d)
```

```
'data.frame': 48 obs. of  5 variables:
 $ density : int  10 10 10 10 10 10 10 10 10 10 ...
 $ pred    : Factor w/ 2 levels "no","pred": 1 1 1 1 1 1 1 2 2 ...
 $ size    : Factor w/ 2 levels "big","small": 1 1 1 1 2 2 2 1 1 ...
 $ surv    : int  9 10 7 10 9 9 10 9 4 9 ...
 $ propsurv: num  0.9 1 0.7 1 0.9 0.9 1 0.9 0.4 0.9 ...
```

The problem says to use `pred` and `size`. These are binary factors, so we'll need to recode them as dummy variables.

R code 10.3

```
d$pred <- ifelse( d$pred=="no" , 0 , 1 )
d$big <- ifelse( d$size=="big" , 1 , 0 )
```

Now we can define and fit the models. Just to review, let's begin again with the basic varying intercept model from the chapter. It contains no predictors, just an intercept for each tank:

R code 10.4

```
d$tank <- 1:nrow(d)

m0 <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank],
        a_tank[tank] ~ dnorm(a,sigma_tank),
        a ~ dnorm(0,10),
        sigma_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 )
```

To this model skeleton we add predictors. Nothing will change otherwise. The varying intercepts and their priors remain the same throughout. First a model containing only the dummy variable indicating presence of predators:

R code 10.5

```
m_p <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank] + bp*pred,
        bp ~ dnorm(0,1),
        a_tank[tank] ~ dnorm(a,sigma_tank),
        a ~ dnorm(0,10),
        sigma_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 )
```

Next the model containing only the dummy variable indicating big tadpoles:

R code 10.6

```
m_b <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank] + bb*big,
        bb ~ dnorm(0,1),
        a_tank[tank] ~ dnorm(a,sigma_tank),
        a ~ dnorm(0,10),
        sigma_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 )
```

Finally, a model containing both predictors and a model that contains their interaction:

R code 10.7

```
m_p_b <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank] + bp*pred + bb*big,
        c(bp,bb) ~ dnorm(0,1),
        a_tank[tank] ~ dnorm(a,sigma_tank),
        a ~ dnorm(0,10),
        sigma_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 )
m_p_b_pb <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank] + bp*pred + bb*big + bpb*pred*big,
        c(bp,bb,bpb) ~ dnorm(0,1),
        a_tank[tank] ~ dnorm(a,sigma_tank),
        a ~ dnorm(0,10),
        sigma_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 )
```

Now we'd like to inspect how the estimated variation across tanks changes from model to model. This means comparing posterior distributions for σ_{TANK} across the models. The `coeftab` function is probably the quickest way to compare posterior means, at least:

R code 10.8
```
coeftab(m0,m_p,m_b,m_p_b,m_p_b_pb)
```

```
           m0      m_p     m_b     m_p_b   m_p_b_pb
...
sigma_tank    1.64    0.83    1.63    0.79    0.75
...
nobs            48      48      48      48      48
```

You'll get a full table with all of the parameters in it, including the varying intercepts (all 48 of them). I've just abbreviated the output above to make it easier to focus on `sigma_tank`. Note that adding a predictor always decreased the posterior mean variation across tanks. Why? Because the predictors are, well, predicting variation. This leaves less variation for the varying intercepts to mop up. In theory, if we had in the form of predictor variables all of the relevant information that determined the survival outcomes, there would be zero variation across tanks.

You might also notice that `big` reduces the variation much less than does `pred`. The predictor `big`, in these models, doesn't help prediction very much, so accounting for it has minimal impact on the estimated variation across tanks.

12M2. Here is the WAIC comparison table:

R code 10.9
```
compare(m0,m_p,m_b,m_p_b,m_p_b_pb)
```

```
           WAIC pWAIC dWAIC weight    SE  dSE
m_p       1000.8  28.9   0.0   0.36 37.41   NA
m_p_b     1001.0  28.2   0.2   0.32 37.52 1.94
m_p_b_pb  1001.1  28.0   0.3   0.31 37.66 2.73
m0        1009.9  38.2   9.1   0.00 38.20 6.49
m_b       1010.2  38.2   9.4   0.00 38.22 6.58
```

The top three models are almost perfectly tied, suggesting that `big` accounts for very little. Can we see this is the coefficients?

R code 10.10
```
coeftab(m0,m_p,m_b,m_p_b,m_p_b_pb)
```

```
           m0      m_p     m_b     m_p_b   m_p_b_pb
...
sigma_tank    1.64    0.83    1.63    0.79    0.75
bp              NA   -2.44      NA   -2.46   -1.94
bb              NA      NA   -0.31   -0.46    0.14
bpb             NA      NA      NA      NA   -1.09
nobs            48      48      48      48      48
```

I abbreviated the above by clipping out the intercept parameters. Let's focus on the beta coefficients. Note that the posterior means for `bb` are smaller in absolute value than those for `bp`. This is consistent with the WAIC comparison. In fact, the standard deviations on these coefficients are big enough that the `bb` posterior distributions overlap zero quite a bit. Consider for example the model `m_b`:

R code 10.11
```
precis(m_b)
```

```
            Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
bb         -0.31   0.46      -1.03       0.40   438    1
a           1.53   0.34       0.97       2.07   755    1
sigma_tank  1.63   0.22       1.26       1.94  2225    1
```

But before you conclude that tadpole size doesn't matter, remember that other models, perhaps including additional predictors, might find new life for `big`. Inference is always conditional on the model.

12M3. Now we want a slightly modified version of model `m0` from problem **12M1**. We just replace the Gaussian adaptive prior with a similar Cauchy prior. The

R code 10.12

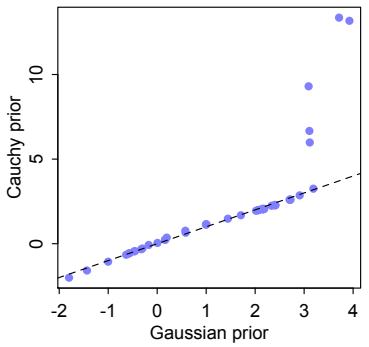
```
m0_Cauchy <- map2stan(
    alist(
        surv ~ dbinom(density,p),
        logit(p) <- a_tank[tank],
        a_tank[tank] ~ dcauchy(a,scale_tank),
        a ~ dnorm(0,10),
        scale_tank ~ dcauchy(0,1)
    ),
    data=d , chains=4 , warmup=1000 , iter=3000 ,
    control=list(adapt_delta=0.99) , cores=4 )
```

You might have some trouble sampling efficiently from this posterior, on account of the long tails of the Cauchy. These result in the intercepts `a_tank` being poorly identified. You saw a simple example of this problem in Chapter 8, when you met MCMC and learned about diagnosing bad chains. To help with sampling, I've added the `control` list. 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_tank` parameters. Plotting the posterior means will be a lot more meaningful than just looking at the values.

R code 10.13

```
post_m0 <- extract.samples(m0)
a_tank_m0 <- apply(post_m0$a_tank,2,mean)
post_m0C <- extract.samples(m0_Cauchy)
a_tank_m0C <- apply(post_m0C$a_tank,2,mean)
plot( a_tank_m0 , a_tank_m0C , pch=16 , col=rangi2 ,
    xlab="Gaussian prior" , ylab="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.

12M4. 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 10.14

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$recipient <- NULL
d$block_id <- d$block

m12M4 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ),
        logit(p) <- a_actor[actor] + a_block[block_id] +
                    (bp + bpc*condition)*prosoc_left,
        a_actor[actor] ~ dnorm( a , sigma_actor ),
        a_block[block_id] ~ dnorm( g , sigma_block ),
        c(a,g,bp,bpc) ~ dnorm(0,10),
        sigma_actor ~ dcauchy(0,1),
        sigma_block ~ dcauchy(0,1)
    ) ,
    data=d, warmup=1000 , iter=6000 , chains=4 , cores=3 )
```

And just to make life easier, here's the code to re-fit the model from the chapter:

R code 10.15

```
m12.5 <- map2stan(
    alist(
        pulled_left ~ dbinom( 1 , p ),
        logit(p) <- a + a_actor[actor] + a_block[block_id] +
                    (bp + bpc*condition)*prosoc_left,
        a_actor[actor] ~ dnorm( 0 , sigma_actor ),
        a_block[block_id] ~ dnorm( 0 , sigma_block ),
        c(a,bp,bpc) ~ dnorm(0,10),
        sigma_actor ~ dcauchy(0,1),
        sigma_block ~ dcauchy(0,1)
    ) ,
    data=d, warmup=1000 , iter=6000 , chains=4 , cores=3 )
```

Now lets look at the `precis` output of each model:

R code 10.16

```
precis(m12.5)

precis(m12M4)
```

```
              Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a             0.44   0.96      -1.00       1.94  2857    1
bp            0.83   0.26       0.42       1.24  9090    1
bpc          -0.14   0.30      -0.61       0.34  9395    1
sigma_actor   2.28   0.95       1.04       3.45  4088    1
sigma_block   0.22   0.18       0.01       0.43  1894    1

              Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a            -0.06   7.06     -11.66      11.15   657 1.01
g             0.47   7.05     -10.32      12.35   662 1.01
bp            0.84   0.25       0.43       1.23  1188 1.01
bpc          -0.13   0.29      -0.62       0.30  2613 1.00
sigma_actor   2.37   0.99       1.05       3.94    74 1.05
sigma_block   0.28   0.17       0.08       0.48   149 1.03
```

The new model, `m12M4`, 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.

12H1. To fit the fixed-effects and varying-effects models:

R code 10.17

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

# fix index
d$district_id <- as.integer(as.factor(d$district))

# prep trimmed data list
dlist <- list(
    use_contraception = d$use.contraception,
    district = d$district_id )

# fixed effects model
m12H1f <- map2stan(
    alist(
        use_contraception ~ dbinom( 1 , p ),
        logit(p) <- a_district[district],
        a_district[district] ~ dnorm(0,10)
    ),
    data=dlist )

# varying effects model
m12H1v <- map2stan(
    alist(
        use_contraception ~ dbinom( 1 , p ),
        logit(p) <- a + a_district[district],
        a ~ dnorm(0,10),
        a_district[district] ~ dnorm(0,sigma),
        sigma ~ dcauchy(0,1)
    ),
    data=dlist )
```

Our job now is to get predicted probabilities for each district for each model. You could do this several ways. I'm going to show you how to use `link` to do it. First, set up a new data list to compute predictions for. In this list, each case is just a unique district.

R code 10.18
```
pred.dat <- list(district=1:60)
```

The values inside `use_contraception` don't matter. It just matters that they exist, so that `link` knows you want predictions for any linear models associated with that outcome variable.

And now we just call `link` for each model, passing it the new cases to compute predicted probabilities for:

R code 10.19
```
pred1 <- link(m12H1f,data=pred.dat)
pred2 <- link(m12H1v,data=pred.dat)
```

It's worth peeking inside these to see that what they hold are 1000 proabilities for each district. That is, the prediction for each district was computed for each of 1000 samples from the posterior distribution.

R code 10.20
```
str(pred1)
```

```
 num [1:1000, 1:60] 0.184 0.31 0.316 0.285 0.273 ...
```

So that's 1000 probabilities for each of the 60 districts. So to get average probabilities in each district:

R code 10.21
```
p1.mean <- apply( pred1 , 2 , mean )
p2.mean <- apply( pred2 , 2 , mean )
round(p1.mean,2)
```

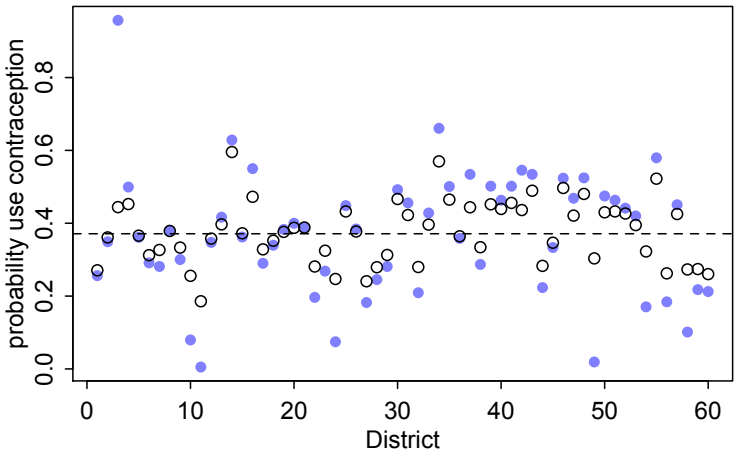
```
 [1] 0.26 0.35 0.96 0.50 0.36 0.29 0.28 0.38 0.30 0.08 0.00 0.34 0.42 0.63 0.37
[16] 0.55 0.29 0.34 0.39 0.40 0.39 0.20 0.27 0.07 0.45 0.38 0.18 0.25 0.28 0.49
[31] 0.45 0.21 0.43 0.66 0.50 0.35 0.54 0.29 0.50 0.46 0.50 0.54 0.53 0.22 0.33
[46] 0.52 0.47 0.53 0.02 0.47 0.46 0.44 0.42 0.16 0.58 0.19 0.46 0.10 0.22 0.22
```

Those are posterior average probabilities of using contraception for each district. These are the points to plot.

Now to plot the estimated proportions using contraception. This means we take the estimated log-odds lists and use the logistic transform on them, to convert to probability scale. This code will do this, as they are plotted:

R code 10.22
```
plot( 1:60 , p1.mean , col=rangi2 , pch=16 , xlab="District" ,
    ylab="probability use contraception" )
points( 1:60 , p2.mean )
abline( h=logistic(coef(m12H1v)[1]) , lty=2 ) # plot line for 'a'
```

And here's the plot:



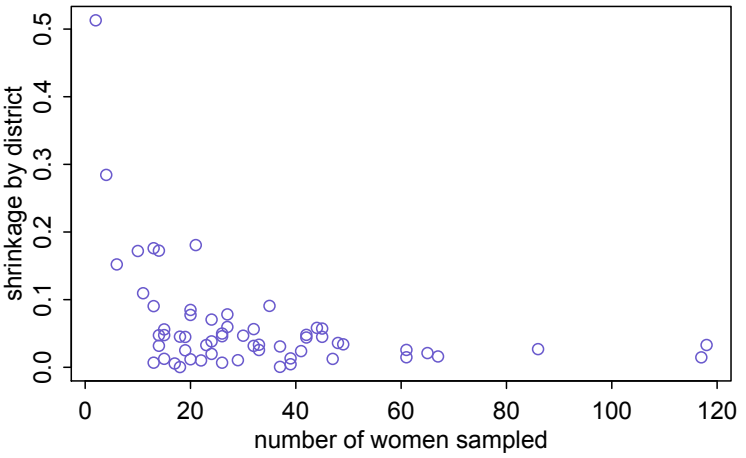
The blue points are the fixed-effects estimates, and the open black ones are the varying effects. The dashed line is the average proportion of women using contraception, in the entire sample (as estimated by the varying effects model). Notice first that the black points are always closer to the dashed line, as was the case with the tadpole example in lecture. This results from *shrinkage*, which results from pooling information.

There are cases with rather extreme disagreements, though. The most obvious is district 3, which has a fixed (blue) estimate of 1 but a varying (black) estimate of only 0.44. There are also two districts (11 and 49) for which the fixed estimates are zero, but the varying estimates are 0.18 and 0.30. What's going on here? In these districts, either all sampled women used contraception or none did. As a result, the fixed effects estimates were silly—the parameter estimates are far from zero with very large standard errors. The varying effects model was able to produce more rational estimates, because it pooled information from other districts. Depending upon how many women were sampled in each district, there was more or less shrinkage (pooling) towards to grand mean. So for example in the case of district 3, there were only 2 women in the sample, and so there is a lot of distance between the blue and black points. In contrast, district 11 had 21 women in the sample, and so while pooling pulls the estimate off of zero to 0.18, it doesn't pull it nearly as far as district 3.

Another way to think of this phenomenon is to view the same estimates arranged by number of women in the district sample, on the horizontal axis. Then on the vertical we can plot the distance (absolute value of the difference) between the fixed and varying estimates. Here's what that looks like:

R code 10.23

```
# compute number of women sampled in each district
n_by_district <- sapply( 1:60 ,
    function(did) length(d$district_id[d$district_id==did]) )
# compute shrinkage
shrinkage <- abs( p1.mean - p2.mean )
plot( n_by_district , shrinkage , col="slateblue" ,
    xlab="number of women sampled" , ylab="shrinkage by district" )
```



You can think of the vertical axis as being the amount of shrinkage. The districts with fewer women sampled show a lot more shrinkage, because there is less information in them. As a result, they are expected to overfit more, and so they are shrunk more towards the overall mean.

12H2. First, load the data and construct a trimmed data list to use:

R code 10.24

```
library(rethinking)
data(Trolley)
d <- Trolley
dat <- list(
    response=d$response,
    action=d$action,
    intention=d$intention,
    contact=d$contact,
    id=coerce_index(d$id) )
```

We want to define two models. They will be identical aside from one having varying intercepts on individuals. There are repeat measures on individuals in these data, because each individual evaluated a range of scenarios.

Here's the ordinary model without intercepts on individuals:

R code 10.25

```
m12H2a <- map2stan(
     alist(
         response ~ dordlogit(phi,cutpoints),
         phi <- bA*action + bI*intention + bC*contact,
         c(bA,bI,bC) ~ dnorm(0,1),
         cutpoints ~ dnorm(0,10)
     ),
     start=list( cutpoints=seq(from=-2.5,to=1.5,length.out=6) ),
     data=dat , chains=3 , cores=3 )
```

Note the trick of using `seq` to define the ordered start values for the `cutpoints`.

Now for the model with varying intercepts on individuals:

R code 10.26

```
m12H2b <- map2stan(
     alist(
         response ~ dordlogit(phi,cutpoints),
         phi <- a_id[id] + bA*action + bI*intention + bC*contact,
         a_id[id] ~ dnorm(0,sigma_id),
         c(bA,bI,bC) ~ dnorm(0,1),
         cutpoints ~ dnorm(0,10),
         sigma_id ~ dcauchy(0,1)
     ),
     start=list( cutpoints=seq(from=-2.5,to=1.5,length.out=6) ),
     data=dat , chains=3 , cores=3 )
```

The subtle issue here is to realize not to include a new parameter for the mean of the varying intercepts. Why? Because the `cutpoints` are the means of each level. These varying intercepts are adjustments to those means. So we just need the standard deviation, `sigma_id`.

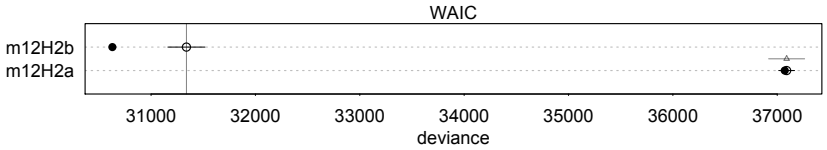
Let's do the quick WAIC comparison of the two models:

R code 10.27

```
compare(m12H2a,m12H2b)
```

```
          WAIC pWAIC  dWAIC weight     SE    dSE
m12H2b 31338.7 354.8    0.0      1 177.86     NA
m12H2a 37090.0   9.1 5751.3      0  76.17 172.54
```

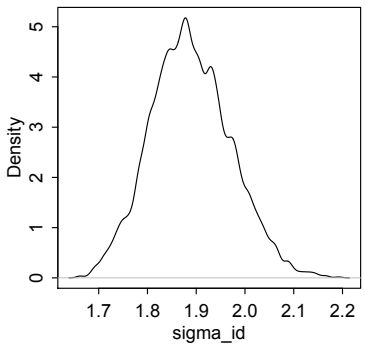
That's a big difference. Plotted, it is even more striking:



This suggests there is a lot of variation among individuals. Let's take a look at the posterior distribution of `sigma_id`:

R code 10.28

```
sigma_id <- extract.samples(m12H2b)$sigma_id
dens(sigma_id,xlab="sigma_id")
```



With a mode around 1.9, that's a lot of individual variation. How can I tell? You have to keep in mind the logit (log-odds) scale these parameters are on. A change on one unit of log-odds is a big change in probability. To visualize how big a difference this variation makes, we can simply compute

the predicted average response for each individual in the sample, under a similar treatment. For this, we'll use `sim`. Note the trick below of including a dummy max-value response in the prediction data frame. This is needed so that `sim` (as well as `link`) knows how many levels are present in the outcome variable.

R code 10.29

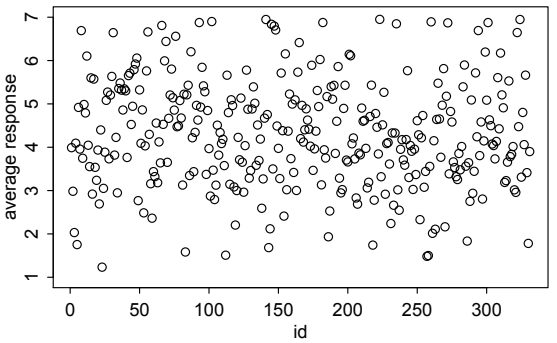
```
d_pred <- data.frame(
    response=7,  #### needed so sim knows num levels
    action=0,
    intention=1,
    contact=0,
    id=1:331     #### 331 unique individuals
)
response <- sim(m12H2b,data=d_pred)
str(response)
```

```
 int [1:1000, 1:331] 4 7 7 5 6 4 5 3 6 6 ...
```

The result is 1000 simulated responses for each of the 331 individuals. Now let's average over the samples and plot the points:

R code 10.30

```
response.mu <- apply(response,2,mean)
plot( response.mu , xlab="id" , ylab="average response" , ylim=c(1,7) )
```



That's a lot of variation in average response to the same type of scenario.

Now it's natural to ask what accounting for this individual variation has done to our estimates of the treatment effects. Let's compare the posterior means of the two models:

R code 10.31

```
coeftab(m12H2a,m12H2b)
```

```
             m12H2a  m12H2b
cutpoints[1]   -2.84   -3.97
cutpoints[2]   -2.15   -3.05
cutpoints[3]   -1.57   -2.27
cutpoints[4]   -0.55   -0.79
cutpoints[5]    0.12    0.25
cutpoints[6]    1.03    1.65
bA             -0.71   -0.96
bI             -0.72   -0.96
bC             -0.96   -1.27
...
sigma_id          NA    1.89
```

```
nobs            9930    9930
```

You'll get a bunch of varying intercepts in your display as well, but I've cut those out of the above to save space. Note that the `bA`, `bI`, and `bC` coefficients have all gotten further from zero, indicating stronger effects. The variation among individuals masked some of the treatment effect, perhaps. You should worry that the precision has changed as well, so go ahead and compare the standard deviations, using `precis` perhaps. You'll see that the `m12H2b` estimates are just as precise. They are merely further from zero.

12H3. 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 10.32

```
library(rethinking)
data(Trolley)
d <- Trolley
dat <- list(
    response=d$response,
    action=d$action,
    intention=d$intention,
    contact=d$contact,
    id=coerce_index(d$id),
    story=coerce_index(d$story) )
```

The following code will fit the cross-classified model. Be patient. It may take up to 10 or 15 minutes per chain, depending upon your computer. So you definitely want to use multiple `cores` here, so you can sampling from all of the chains at the same time.

R code 10.33

```
m12H3 <- map2stan(
     alist(
         response ~ dordlogit(phi,cutpoints),
         phi <- a_id[id] + a_story[story] +
                bA*action + bI*intention + bC*contact,
         a_id[id] ~ dnorm(0,sigma_id),
         a_story[story] ~ dnorm(0,sigma_story),
         c(bA,bI,bC) ~ dnorm(0,1),
         cutpoints ~ dnorm(0,10),
         sigma_id ~ dcauchy(0,1),
         sigma_story ~ dcauchy(0,1)
     ),
     start=list( cutpoints=seq(from=-2.5,to=1.5,length.out=6) ),
     data=dat , chains=3 , cores=3 )
```

The first thing to do is compare this model to the two models from the previous problem. If you need to re-fit those models (`m12H2a` and `m12H2b`), do so now. Then:

R code 10.34

```
compare( m12H2a , m12H2b , m12H3 )
```

```
          WAIC pWAIC  dWAIC weight     SE    dSE
m12H3  30686.7 364.4    0.0      1 179.71     NA
m12H2b 31338.6 354.9  652.0      0 177.80  48.31
m12H2a 37090.1   9.1 6403.4      0  76.10 175.06
```

That looks like strong support for including varying intercepts on `story`, implying there is important variation among stories with the same treatment (`action`, `intention`, `contact`) values.

Let's look at the marginal posterior distributions for `m12H3`:

R code 10.35

```
precis(m12H3)
```

```
             Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
bA          -1.24   0.06      -1.32      -1.14  3000    1
bI          -0.93   0.05      -1.02      -0.85  3000    1
bC          -1.85   0.08      -1.97      -1.72  3000    1
sigma_id     1.96   0.08       1.84       2.10  3000    1
sigma_story  0.61   0.15       0.39       0.80  3000    1
```

The standard deviation among individuals, `sigma_id`, is similar to what it was in `m12H2b`. The posterior mean standard deviation among stories, `sigma_story`, 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. Both `bA` and `bC` have gotten further from zero, and their 89% intervals don't even include the previous 89% lower bounds from `m12H2b`. So again, variation across clusters in the presence of repeat measures hid some of the treatment effect from us, it seems.

11. Chapter 13 Solutions

13E1. 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. You can find an annotated example on page 407. 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.

$$
\begin{aligned}
y_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha_{\text{GROUP}[i]} + \beta_{\text{GROUP}[i]} x_i \\
\begin{bmatrix} \alpha_{\text{GROUP}} \\ \beta_{\text{GROUP}} \end{bmatrix} &\sim \text{MVNormal}\left(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S} \right) \\
\mathbf{S} &= \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta &\sim \text{Normal}(0, 10) \\
\sigma &\sim \text{HalfCauchy}(0, 1) \\
\sigma_\alpha &\sim \text{HalfCauchy}(0, 1) \\
\sigma_\beta &\sim \text{HalfCauchy}(0, 1) \\
\mathbf{R} &\sim \text{LKJcorr}(2)
\end{aligned}
$$

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.

13E2. 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.

13E3. 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 11.1

```
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 11.2

```
m_nopool <- map2stan(
    alist(
        score ~ dnorm(mu,sigma),
        mu <- a_id[id],
        a_id[id] ~ dnorm(0,10),
        sigma ~ dcauchy(0,1)
    ),
    data=d , chains=2 )
```

And this is the corresponding "partial pooling" model with an adaptive prior, the varying effects model. I'm going to use the non-centered (page 419, see Overthinking box on page 422) parameterization here, because this is exactly a common situation in which it is needed to efficiently sample.

R code 11.3

```
m_partpool <- map2stan(
    alist(
        score ~ dnorm(mu,sigma),
        mu <- a + z_id[id]*sigma_id,
        z_id[id] ~ dnorm(0,1),
        a ~ dnorm(0,10),
        sigma ~ dcauchy(0,1),
        sigma_id ~ dcauchy(0,1)
    ),
    constraints=list(sigma_id="lower=0"),
```

```
    data=d , chains=4 , cores=4 )
```

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 11.4

```
compare( m_nopool , m_partpool )
```

```
             WAIC pWAIC dWAIC weight    SE   dSE
m_partpool 2917.3   8.2   0.0      1 44.45    NA
m_nopool   3024.0  95.7 106.8      0 45.72 17.29
```

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 11.5

```
precis(m_partpool)
```

```
         Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a        0.04   0.03      -0.01       0.10  4000    1
sigma    1.04   0.02       1.00       1.08  4000    1
sigma_id 0.07   0.05       0.00       0.14  1272    1
```

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.

13M1. Here's a compact form of the café robot simulation code, with the correlation `rho` changed to zero:

R code 11.6

```
# 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:

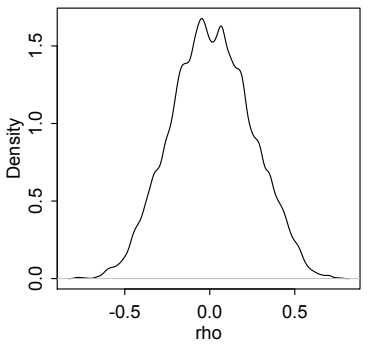
R code 11.7

```
m13.1 <- map2stan(
    alist(
        wait ~ dnorm( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
        c(a_cafe,b_cafe)[cafe] ~ dmvnorm2(c(a,b),sigma_cafe,Rho),
        a ~ dnorm(0,10),
        b ~ dnorm(0,10),
        sigma_cafe ~ dcauchy(0,2),
        sigma ~ dcauchy(0,2),
        Rho ~ dlkjcorr(2)
    ) ,
    data=d ,
    iter=5000 , warmup=2000 , chains=2 , cores=2 )
```

Now to visualize the posterior distribution of `rho`:

R code 11.8

```
post <- extract.samples(m13.1)
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.

13M2. 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 11.9

```
# 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 11.10

```
m13M2 <- map2stan(
    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 ~ dcauchy(0,1),
        sigma_alpha ~ dcauchy(0,1),
        sigma_beta ~ dcauchy(0,1)
    ) ,
    data=d ,
    iter=5000 , warmup=2000 , chains=3 , cores=3 )
```

And let's sample again from the model in the chapter, so we can directly compare estimates. In the main text, the Cauchy priors were given a scale of 2, while in the model above they are set to 1. So I'll adjust the priors to match here. But it doesn't make much difference.

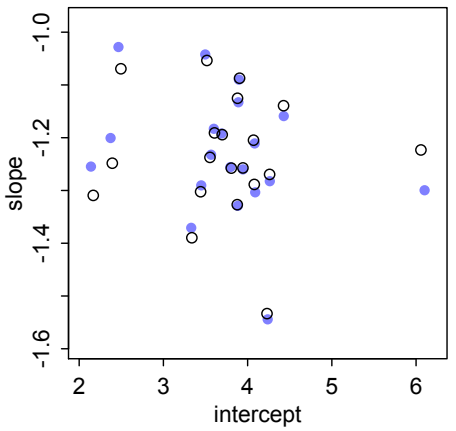
R code 11.11

```
m13.1 <- map2stan(
    alist(
        wait ~ dnorm( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
        c(a_cafe,b_cafe)[cafe] ~ dmvnorm2(c(a,b),sigma_cafe,Rho),
        a ~ dnorm(0,10),
        b ~ dnorm(0,10),
        sigma_cafe ~ dcauchy(0,1),
        sigma ~ dcauchy(0,1),
        Rho ~ dlkjcorr(2)
    ) ,
    data=d ,
    iter=5000 , warmup=2000 , chains=3 , cores=3 )
```

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 11.12

```
post1 <- extract.samples(m13.1)
a1 <- apply( post1$a_cafe , 2 , mean )
b1 <- apply( post1$b_cafe , 2 , mean )

post2 <- extract.samples(m13M2)
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 `m13.1`, the original mode in the chapter that includes the correlation between intercepts and slopes. The open points are the posterior means from `m13M2`, 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 `m13.1` 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.

13M3. For convenience, first let's fit the varying slopes model from the chapter. To compare total running times, you'll have to run this model on your machine, anyway.

R code 11.13

```
library(rethinking)
data(UCBadmit)
d <- UCBadmit
d$male <- ifelse( d$applicant.gender=="male" , 1 , 0 )
d$dept_id <- coerce_index( d$dept )

m13.3 <- map2stan(
    alist(
        admit ~ dbinom( applications , p ),
        logit(p) <- a_dept[dept_id] +
                    bm_dept[dept_id]*male,
        c(a_dept,bm_dept)[dept_id] ~ dmvnorm2( c(a,bm) , sigma_dept , Rho ),
        a ~ dnorm(0,10),
        bm ~ dnorm(0,1),
        sigma_dept ~ dcauchy(0,2),
        Rho ~ dlkjcorr(2)
    ) ,
    data=d , warmup=1000 , iter=5000 , chains=4 )
```

And this is the easy way to specify the non-centered form:

R code 11.14

```
m13M3 <- map2stan(
    alist(
        admit ~ dbinom( applications , p ),
        logit(p) <- a + a_dept[dept_id] +
                    bm + bm_dept[dept_id]*male,
        c(a_dept,bm_dept)[dept_id] ~ dmvnormNC( sigma_dept , Rho ),
        a ~ dnorm(0,10),
        bm ~ dnorm(0,1),
        sigma_dept ~ dcauchy(0,2),
        Rho ~ dlkjcorr(2)
    ) ,
```

```
    data=d , warmup=1000 , iter=5000 , chains=4 )
```

First let's compare running times. On my machine, each chain in the original model `m13.3` took about 1.7 seconds. In the new model `m13M3`, each chain took about 10 seconds to finish sampling. So the centered version of the mode, `m13.3`, 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 11.15

```
par_list <- c('a','bm','a_dept','b_dept','sigma_dept','Rho')
n_eff1 <- precis(m13.3,2,pars=par_list)@output[,'n_eff']
n_eff2 <- precis(m13M3,2,pars=par_list)@output[,'n_eff']
cbind( n_eff1 , n_eff2 )
```

```
        n_eff1    n_eff2
 [1,]  5143.729  5659.551
 [2,]  6783.107  9580.778
 [3,] 10010.004  3372.168
 [4,]  9098.312  4840.550
 [5,]  9928.490  3805.146
 [6,]  8934.476  3929.227
 [7,]  8921.542  3586.782
 [8,]  7374.170  3621.472
 [9,]  8038.578  3629.475
[10,]  5116.746  3824.117
[11,] 16000.000 16000.000
[12,]  8602.085  7408.192
[13,]  8602.085  7408.192
[14,] 15534.758 15369.117
```

Wow, the centered version actually has higher effective sample size in almost every case. But in the text, I said that non-centered parameterizations were good for getting more efficient sampling. What is going on here?

The truth is that 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.

13M4. The first thing that is required is to sampling from the Gaussian process model, as well as the simpler Poisson models from Chapter 10. Here's the code to sampling from the Gaussian process model:

R code 11.16

```
library(rethinking)
data(islandsDistMatrix)
Dmat <- islandsDistMatrix
data(Kline2) # load the ordinary data, now with coordinates
d <- Kline2
```

```
d$society <- 1:10 # index observations
m13.7 <- map2stan(
    alist(
        total_tools ~ dpois(lambda),
        log(lambda) <- a + g[society] + bp*logpop,
        g[society] ~ GPL2( Dmat , etasq , rhosq , 0.01 ),
        a ~ dnorm(0,10),
        bp ~ dnorm(0,1),
        etasq ~ dcauchy(0,1),
        rhosq ~ dcauchy(0,1)
    ),
    data=list(
        total_tools=d$total_tools,
        logpop=d$logpop,
        society=d$society,
        Dmat=islandsDistMatrix),
    warmup=2000 , iter=1e4 , chains=4 )
```

And here are the models from Chapter 10, but with similar priors on the intercept a:

R code 11.17

```
d$contact_high <- ifelse( d$contact=="high" , 1 , 0 )
m10.10 <- map2stan(
    alist(
        total_tools ~ dpois( lambda ),
        log(lambda) <- a + bp*logpop +
            bc*contact_high + bpc*contact_high*logpop,
        a ~ dnorm(0,10),
        c(bp,bc,bpc) ~ dnorm(0,1)
    ), data=d , chains=4 )
m10.11 <- map2stan(
    alist(
        total_tools ~ dpois( lambda ),
        log(lambda) <- a + bp*logpop + bc*contact_high,
        a ~ dnorm(0,10),
        c(bp,bc) ~ dnorm( 0 , 1 )
    ), data=d , chains=4 )
m10.12 <- map2stan(
    alist(
        total_tools ~ dpois( lambda ),
        log(lambda) <- a + bp*logpop,
        a ~ dnorm(0,10),
        bp ~ dnorm( 0 , 1 )
    ), data=d , chains=4 )
m10.13 <- map2stan(
    alist(
        total_tools ~ dpois( lambda ),
        log(lambda) <- a + bc*contact_high,
        a ~ dnorm(0,10),
        bc ~ dnorm( 0 , 1 )
    ), data=d , chains=4 )
m10.14 <- map2stan(
    alist(
        total_tools ~ dpois( lambda ),
        log(lambda) <- a,
```

```
        a ~ dnorm(0,10)
    ), data=d , chains=4 )
```

Now let's compare them all using WAIC:

R code 11.18

```
compare(m10.10,m10.11,m10.12,m10.13,m10.14,m13.7)
```

```
        WAIC pWAIC dWAIC weight    SE   dSE
m13.7   67.5   4.1   0.0   0.99  2.19    NA
m10.11  78.4   3.9  11.0   0.00 10.94 11.32
m10.10  80.3   4.9  12.8   0.00 11.21 11.46
m10.12  84.7   3.9  17.2   0.00  8.99  8.21
m10.14 140.8   7.6  73.3   0.00 31.48 32.33
m10.13 151.2  17.2  83.7   0.00 45.23 46.72
```

The Gaussian process model, `m13.7`, gets nearly all of the weight. However, the standard error of the difference between it and the next highest ranked model, `m10.11`, is the same as the difference itself. And the next ranked model, `m10.10`, is basically tied with `m10.11`. So we can't be too cocky here. There isn't a lot of data, after all.

But we can still learn from attempting to understand the effective number of parameter, `pWAIC`, as the problem asks. Model `m13.7` has 4 parameters, and WAIC estimates its effective parameter count at almost exactly 4. You might think this is to be expected, but look at the other models. Here's a table comparing each models actual parameter count and the effective parameter count:

Model	parameters	p_{WAIC}
`m13.7`	4	4.1
`m10.11`	3	3.9
`m10.10`	4	4.9
`m10.12`	2	3.9
`m10.14`	1	7.6
`m10.13`	2	17.2

There's a tendency for the other Poisson models to be more flexible than you might expect from the parameter count alone. Model `10.14`, the intercept-only model, actually ends up with almost 7 more effective parameters than actual parameters. And model `m10.13` ends up with a whopping 17 effective parameters.

This is in fact a very common occurrence with Poisson GLMs. So do not be alarmed. For now, just note that the Gaussian process model is the only model here with any form of adaptive regularization. It is a very complex model, based upon its structure. But is actually less flexible than most of the other models, and it is the only model that has about the same effective number of parameters as actual parameters.

13H1. If you haven't completed the problem from Chapter 12 that uses this data set, do so first. Then, let's begin by reloading the data and preparing the index variable for district and building a trimmed data list:

R code 11.19

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

# fix index - can also use coerce_index() here
d$district_id <- as.integer(as.factor(d$district))
```

```
# rename outcome so it doesn't have a dot in it
dlist <- list(
    use_contraception = d$use.contraception,
    urban = d$urban,
    district = d$district_id )
```

And to sample from the varying slopes model:

R code 11.20

```
m13H1 <- map2stan(
    alist(
        use_contraception ~ dbinom( 1 , p ),
        logit(p) <- a + a_district[district] +
                    (b + b_district)*urban,
        c(a,b) ~ dnorm(0,10),
        c(a_district,b_district)[district] ~ dmvnorm2(0,sigma,Rho),
        sigma ~ dcauchy(0,1),
        Rho ~ dlkjcorr(2)
    ),
    data=dlist,
    warmup=1000 , iter=4000 , chains=4 , cores=3 )
```

Let's look at the marginal posterior, omitting the varying effects:

R code 11.21

```
precis( m13H1 , pars=c("a","b","sigma","Rho") , depth=2 )
```

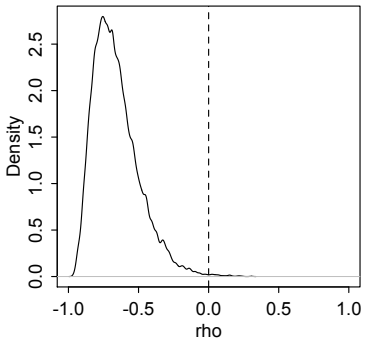
```
           Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a         -0.72   0.10      -0.88      -0.55  4499    1
b          0.72   0.17       0.45       1.00  5070    1
sigma[1]   0.58   0.10       0.42       0.73  2264    1
sigma[2]   0.79   0.21       0.47       1.12   938    1
Rho[1,1]   1.00   0.00       1.00       1.00 12000  NaN
Rho[1,2] -0.66   0.16      -0.90      -0.43  1752    1
Rho[2,1] -0.66   0.16      -0.90      -0.43  1752    1
Rho[2,2]   1.00   0.00       1.00       1.00 12000    1
```

So the main effect of urban residence appears to be to increase contraceptive use. Not too surprising. From the variance components, `sigma[1]` is the standard deviation for varying intercepts and `sigma[2]` is the standard deviation for slopes. We see that there is more variation in the slope of `urban` across districts than there is in the intercepts. We do have to be careful interpreting differences here, because a varying slope is multiplied by data, so the scale isn't the same as the intercepts. But it's safe to say that there's a lot of variation in slopes—urbanity has a different relationship with contraceptive use in different districts.

Now let's inspect the correlation between intercepts and slopes across districts. The posterior mean of ρ is -0.66, with an 89% interval well below zero. Here's the marginal posterior for ρ, the correlation between intercepts and slopes in the population of districts:

R code 11.22

```
post <- extract.samples(m13H1)
dens( post$Rho[,1,2] , xlab="rho" , xlim=c(-1,1) )
abline( v=0 , lty=2 ) # add dashed marker at zero correlation
```



I added a dashed marker at the zero correlation mark, to aid in interpretation.

All of this means that intercepts and slopes are negatively correlated. So larger intercepts are paired with smaller slopes. The higher the contraceptive use in rural areas (determined entirely by the varying intercept), the smaller the difference with the urban area of the same district. This is easier to see with two more plots. First, let's show probability of rural use against probability of urban use for each district:

R code 11.23

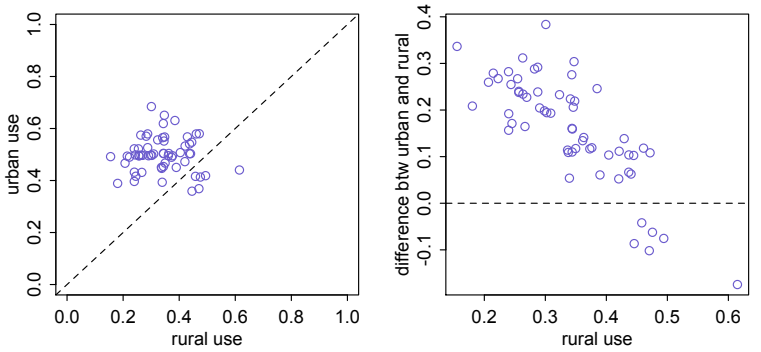
```
pred.dat.rural <- list(
    urban=rep(0,60),
    district=1:60 )
pred.dat.urban <- list(
    urban=rep(1,60),
    district=1:60 )
pred.rural <- link( m13H1 , data=pred.dat.rural )
pred.urban <- link( m13H1 , data=pred.dat.urban )

means.rural <- apply( pred.rural , 2 , mean )
means.urban <- apply( pred.urban , 2 , mean )

plot( means.rural , means.urban , col="slateblue" ,
    xlim=c(0,1) , ylim=c(0,1) ,
    xlab="rural use" , ylab="urban use" )
abline(a=0,b=1,lty=2)
```

And we will also plot the difference between urban and rural, as a function of rural use:

R code 11.24

```
plot( means.rural , means.urban-means.rural , col="slateblue" ,
    xlab="rural use" , ylab="difference btw urban and rural" )
abline(h=0,lty=2)
```

This results in (showing both plots):



So we might say that districts in which more rural women use contraception have urban areas more similar to the rural areas. To say it another way, districts in which rural women use more contraception are places in which the urban and rural women are more similar (closer to the dashed lines). Districts in which urban use is highest have low rural use, but districts with the lowest urban use rates have some of the highest rural use rates.

13H2. Here's the code to fit the model with varying intercepts and slopes for age, clustered by Subject:

R code 11.25

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

m13H2 <- map2stan(
    alist(
        height ~ dnorm( mu , sigma ),
        mu <- a + a_subject +
                 (b + b_subject)*age,
        a ~ dnorm(0,100),
        b ~ dnorm(0,10),
        c(a_subject,b_subject)[Subject] ~
                       dmvnorm2(0,sigma_subject,Rho_subject),
        sigma_subject ~ dcauchy(0,1),
        Rho_subject ~ dlkjcorr(2),
        sigma ~ dcauchy(0,1)
    ),
    data=d , warmup=1000 , iter=3000 , chains=4 , cores=3 )
```

Let's look at the estimates (omitting the varying effects and correlation for the moment):

R code 11.26

```
precis( m13H2 , depth=2 , pars=c("a","b","sigma_subject") )
```

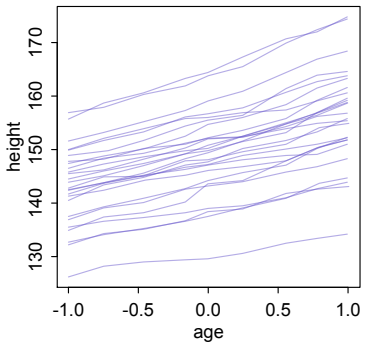
```
                  Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a                149.30   1.66     146.57     151.88   412 1.01
b                  6.50   0.35       5.98       7.06   384 1.01
sigma_subject[1]   8.08   1.15       6.30       9.86  4772 1.00
sigma_subject[2]   1.69   0.25       1.28       2.05  3700 1.00
```

Let's interpret the intercept a first. Since the predictor age is standardized, the intercept is the average height at the average age. Then the average slope b 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 $2 \times 6.5 = 13$cm. That's not so easy to understand. Plotting the raw data might help:

R code 11.27

```
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) )
}
```



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.

13H3. Now let's consider the correlation between intercepts and slopes, using the model fit in the previous problem.

R code 11.28

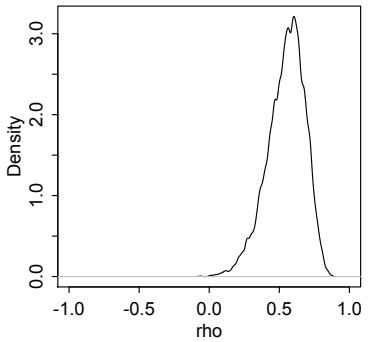
```
precis( m13H2 , depth=2 , pars="Rho_subject" )
```

```
                Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
Rho_subject[1,1] 1.00   0.00       1.00       1.00  8000  NaN
Rho_subject[1,2] 0.55   0.13       0.35       0.76  5269    1
Rho_subject[2,1] 0.55   0.13       0.35       0.76  5269    1
Rho_subject[2,2] 1.00   0.00       1.00       1.00  7100    1
```

So the posterior distribution of the correlation between intercepts and slopes has a mean of 0.55 and an 89% interval from 0.35 to 0.76. This is what it looks like:

R code 11.29

```
rho <- extract.samples(m13H2)$Rho_subject[,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.

13H4. 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 11.30

```
post <- extract.samples(m13H2)
rho <- mean( post$Rho[,1,2] )
sb <- mean( post$sigma_subject[,2] )
sa <- mean( post$sigma_subject[,1] )
sigma <- mean( post$sigma )
a <- mean( post$a )
b <- mean( post$b )
```

Those are ρ, σ_β, σ_α, σ, α, and β, respectively. So now we can define the variance-covariance matrix S:

R code 11.31

```
S <- matrix( c( sa^2 , sa*sb*rho , sa*sb*rho , sb^2 ) , nrow=2 )
round( S , 2 )
```

```
      [,1] [,2]
[1,] 64.34 7.35
[2,]  7.35 2.81
```

And now to sample varying intercepts and slopes from the bivariate distribution of them:

R code 11.32

```
library(MASS)
ve <- mvrnorm( 10 , c(0,0) , Sigma=S )
ve
```

```
             [,1]        [,2]
 [1,] 10.26365234  3.36897685
 [2,]  0.53672770  0.55468957
 [3,] -2.72012193 -0.08724327
 [4,] -8.13864370  1.44000700
 [5,]  1.11519674 -2.37703476
 [6,]  7.83903002  2.76541405
 [7,] -5.09026624  0.64593482
 [8,] -0.01181788 -0.41395942
 [9,] -7.00732191 -2.80120314
[10,] 11.07659433  2.86093515
```

Those are 10 random boys, with their own varying intercepts (first column) and slopes (second column). Remember, these values will be added to the plain 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 11.33

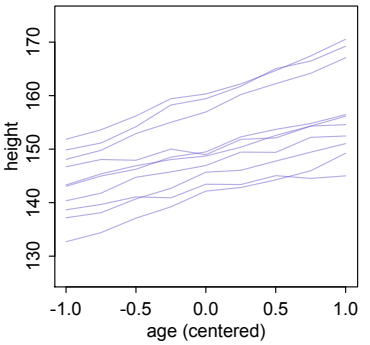
```
age.seq <- seq(from=-1,to=1,length.out=9)
plot( 0 , 0 , type="n" , xlim=range(d$age) , ylim=range(d$height) ,
    xlab="age (centered)" , ylab="height" )
```

Now to loop over the rows in `ve` and simulate 9 heights for each pair of parameters:

R code 11.34

```
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.

12. Chapter 14 Solutions

14E1. 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:

$$\begin{aligned} T_i &\sim \text{Poisson}(\mu_i) \\ \log \mu_i &= \alpha + \beta\phi_i \\ \log P_i &\sim \text{Normal}(\phi_i, \sigma_P) \\ \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma_P &\sim \text{Exponential}(1) \end{aligned}$$

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.

14E2. 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. Here's what it might look like for the Oceanic societies model:

$$\begin{aligned} T_i &\sim \text{Poisson}(\mu_i) \\ \log \mu_i &= \alpha + \beta\phi_i \\ \phi_i &\sim \text{Normal}(\bar{\phi}, \sigma_P) \\ \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 1) \\ \bar{\phi} &\sim \text{Normal}(0, 10) \\ \sigma_P &\sim \text{Exponential}(1) \end{aligned}$$

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.

14M1. 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.

14M2. First, let's repeat fitting the models from Chapter 6, but this time we'll use weakly regularizing priors, since they are superior. We'll also fit these models with `map2stan`. Remember, when using `map2stan` or Stan itself, it's good form to rename variables that contain dots. Otherwise you might run into problems with the R side of the code interfacing with the Stan side of the code. Stan abhors dots like nature abhors a vacuum.

R code 12.1

```
library(rethinking)
data(milk)
d <- milk
dcc <- d[ complete.cases(d$neocortex.perc) , ]
dat <- list(
    log_mass = log(dcc$mass),
    kcal = dcc$kcal.per.g,
    neocortex = dcc$neocortex.perc/100 )

m6.11 <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a,
        a ~ dnorm(0,100),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat , chains=4 )
m6.12 <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a + bn*neocortex,
        a ~ dnorm(0,100),
        bn ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat , chains=4 )
m6.13 <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a + bm*log_mass,
        a ~ dnorm(0,100),
        bm ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat , chains=4 )
m6.14 <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a + bn*neocortex + bm*log_mass,
        a ~ dnorm(0,100),
        c(bn,bm) ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat , chains=4 )
```

These models with the reduced data yield these WAIC rankings:

R code 12.2
```
compare( m6.11 , m6.12 , m6.13 , m6.14 )
```

```
       WAIC pWAIC dWAIC weight   SE  dSE
m6.14 -15.5   2.6   0.0   0.91 4.95   NA
m6.13  -8.9   2.1   6.6   0.03 4.20 2.20
m6.11  -8.9   1.3   6.6   0.03 3.58 3.90
m6.12  -8.1   1.7   7.4   0.02 3.30 4.12
```

This is just as before: The model with both predictors does best by far, because of the masking relationship between body mass and neocortex percent.

Now let's repeat this analysis using all of the cases with imputation on neocortex. First prepare another trimmed data list, this time using all of the cases. We'll two different lists, because when we fit the models that don't have neocortex in them, Stan will still complain about the `NA`s in the data list.

R code 12.3
```
dat_full <- list(
    log_mass = log(d$mass),
    kcal = d$kcal.per.g,
    neocortex = d$neocortex.perc/100 )
dat_full_small <- list(
    log_mass = log(d$mass),
    kcal = d$kcal.per.g )
```

To include imputation, just like in the chapter, all that's required is to assign a distribution to the `neocortex` variable. So for example here is model `m6.12` again with imputation added:

R code 12.4
```
m6.12_full <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a + bn*neocortex,
        neocortex ~ dnorm( mu_nc , sigma_nc ),
        mu_nc ~ dnorm(0.5,1),
        sigma_nc ~ dcauchy(0,1),
        a ~ dnorm(0,100),
        bn ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat_full , chains=4 )
```

The other models follow the same pattern. Here are the other three models, modify to use the full data and impute neocortex values:

R code 12.5
```
m6.11_full <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a,
        a ~ dnorm(0,100),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat_full_small , chains=4 )
m6.13_full <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
```

```
        mu <- a + bm*log_mass,
        a ~ dnorm(0,100),
        bm ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat_full_small , chains=4 )
m6.14_full <- map2stan(
    alist(
        kcal ~ dnorm( mu , sigma ),
        mu <- a + bn*neocortex + bm*log_mass,
        neocortex ~ dnorm( mu_nc , sigma_nc ),
        mu_nc ~ dnorm(0.5,1),
        sigma_nc ~ dcauchy(0,1),
        a ~ dnorm(0,100),
        c(bn,bm) ~ dnorm(0,1),
        sigma ~ dcauchy(0,1)
    ) ,
    data=dat_full , chains=4 )
```

And let's compare these models. Remember, it makes no sense to compare the four models above to the previous four models. Why? Because they are fit to different data. We can however inspect the two different WAIC tables and ask what impact including the additional data has had.

```
compare( m6.11_full , m6.12_full , m6.13_full , m6.14_full )
```

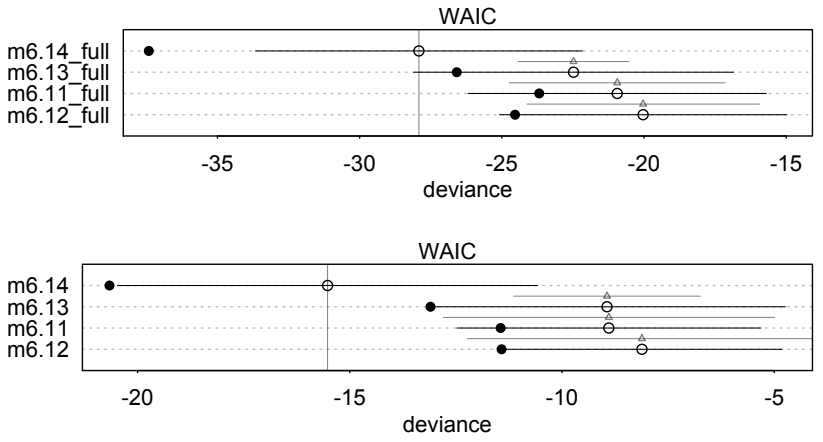
R code 12.6

```
            WAIC pWAIC dWAIC weight   SE  dSE
m6.14_full -27.9   4.7   0.0   0.90 5.75   NA
m6.13_full -22.5   2.1   5.4   0.06 5.62 1.95
m6.11_full -20.9   1.4   7.0   0.03 5.24 3.80
m6.12_full -20.0   2.3   7.9   0.02 5.06 4.09
```

The rank order is the same, and the weights are very similar. It may easier to do a head-to-head comparison by just plotting both tables as dot charts:

```
plot(compare(m6.11_full,m6.12_full,m6.13_full,m6.14_full))
plot(compare(m6.11,m6.12,m6.13,m6.14))
```

R code 12.7



Among all the differences, a striking one is that all of the standard errors are relatively smaller when we use the full data. This is to be expected, as there is more information. You can see this in the coefficients as well. Consider for example the standard deviations on the regression parameters in `m6.14_full` and `m6.14`:

R code 12.8

```
precis(m6.14)
precis(m6.14_full,pars=c("a","bn","bm","sigma"))
```

```
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     -0.27   0.47      -1.03       0.43   741 1.00
bn     1.53   0.72       0.44       2.75   721 1.00
bm    -0.07   0.03      -0.11      -0.03   794 1.00
sigma  0.15   0.03       0.10       0.19   994 1.01

       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     -0.10   0.39      -0.72       0.52   902 1.01
bn     1.22   0.61       0.26       2.19   870 1.01
bm    -0.05   0.02      -0.09      -0.02  1384 1.00
sigma  0.14   0.02       0.10       0.18  1467 1.00
```

The width of the marginal posterior is narrower in every case, and this translates into narrower intervals as well.

For extra practice, I recommend repeating the above with a better imputation model, such as the second imputation model from the examples in the chapter.

14M3. 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 12.9

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce
dlist <- list(
    div_obs=d$Divorce,
    div_sd=d$Divorce.SE * 2 , # note times 2
    R=d$Marriage,
    A=d$MedianAgeMarriage )

m14M3 <- map2stan(
    alist(
        div_est ~ dnorm(mu,sigma),
        mu <- a + bA*A + bR*R,
        div_obs ~ dnorm(div_est,div_sd),
        a ~ dnorm(0,10),
        bA ~ dnorm(0,10),
        bR ~ dnorm(0,10),
        sigma ~ dcauchy(0,2.5)
    ) ,
    data=dlist ,
    start=list(div_est=dlist$div_obs) ,
    WAIC=FALSE , iter=5000 , warmup=1000 , chains=4 , cores=3 ,
    control=list(adapt_delta=0.99) )
```

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. Draw more samples, if you have any doubts. And as always, be sure to check the trace plot and diagnostic statistics.

Now compare the estimates produced. I'll omit the `div_est` estimates here.

R code 12.10

```
precis(m14.1) # original
precis(m14M3) # double standard error
```

```
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     21.37   6.58      11.45      32.41  2784    1
bA    -0.55   0.21      -0.90      -0.22  2857    1
bR     0.13   0.08       0.01       0.25  3267    1
sigma  1.13   0.21       0.78       1.44  1707    1

       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     19.71   6.50       9.17      29.68   445 1.00
bA    -0.56   0.21      -0.89      -0.23   418 1.00
bR     0.22   0.08       0.09       0.35   410 1.01
sigma  0.36   0.22       0.05       0.68   171 1.04
```

The marginal posterior for `bA` is unchanged. But the posterior for `bR`, the coefficient for marriage rate, has almost doubled. 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.

14H1. To prepare for the first model, lead the data and take a look at each variable:

R code 12.11

```
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.

The implied Poisson model predicting `MATINGS` with `AGE` is:

R code 12.12

```
m14H1_1 <- map2stan(
    alist(
        MATINGS ~ dpois(lambda),
        log(lambda) <- a + bA*AGE,
        a ~ dnorm(0,10),
        bA ~ dnorm(0,1)
    ),
    data=d , chains=4 )
precis(m14H1_1)
```

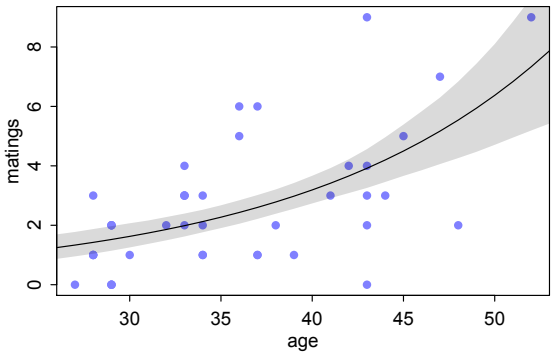
```
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  -1.59   0.55      -2.42      -0.65   380    1
```

```
bA  0.07   0.01       0.05       0.09   384    1
```

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 12.13

```
AGE_seq <- seq(from=20,to=60,by=1)
lambda <- link(m14H1_1,data=list(AGE=AGE_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( AGE_seq , lambda_mu )
shade( lambda_PI , AGE_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.

R code 12.14

```
m14H1_2 <- map2stan(
    alist(
        MATINGS ~ dpois(lambda),
        log(lambda) <- a + bA*AGE_est[i],
        AGE ~ dnorm( AGE_est , 5 ),
        a ~ dnorm(0,10),
        bA ~ dnorm(0,1)
    ),
    start=list(AGE_est=d$AGE),
    data=d , chains=4 , WAIC=FALSE )
precis(m14H1_2)
```

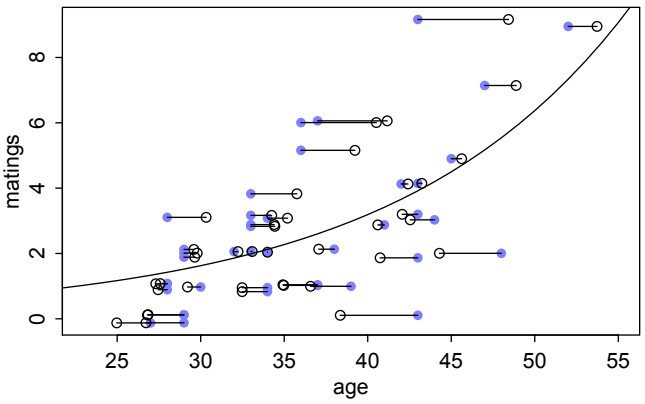
```
41 vector or matrix parameters omitted in display. Use depth=2 to show them.
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  -1.63   0.64      -2.63      -0.60  1118    1
bA  0.07   0.02       0.04       0.09  1036    1
```

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 12.15

```
post <- extract.samples(m14H1_2)
AGE_est <- apply(post$AGE_est,2,mean)

# 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( AGE_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.

14H2. 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. Note that increasing the standard error makes the posterior harder to sample from, so I've added a control list and upped the iterations.

R code 12.16

```
m14H2 <- map2stan(
    alist(
        MATINGS ~ dpois(lambda),
        log(lambda) <- a + bA*AGE_est[i],
        AGE ~ dnorm( AGE_est , 10 ),
        a ~ dnorm(0,10),
        bA ~ dnorm(0,1)
    ),
    start=list(AGE_est=d$AGE),
    data=d , chains=4 , cores=4 , warmup=2000 , iter=1e4 ,
    control=list(adapt_delta=0.99) , WAIC=FALSE )
precis(m14H2)
```

```
41 vector or matrix parameters omitted in display. Use depth=2 to show them.
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  -0.96   0.49      -1.73      -0.19 12958    1
bA  0.05   0.01       0.03       0.07 12136    1
```

The posterior mean for `bA` has gotten closer to zero, but it's still reliably above zero. Let's try doubling it again. The posterior will get increasingly difficult to sample from. But the priors are informative enough to pull it off.

R code 12.17

```
m14H2 <- map2stan(
    alist(
        MATINGS ~ dpois(lambda),
        log(lambda) <- a + bA*AGE_est[i],
        AGE ~ dnorm( AGE_est , 20 ),
        a ~ dnorm(0,10),
        bA ~ dnorm(0,1)
    ),
    start=list(AGE_est=d$AGE),
    data=d , chains=4 , cores=4 , warmup=2000 , iter=1e4 ,
    control=list(adapt_delta=0.99) , WAIC=FALSE )
precis(m14H2)
```

```
41 vector or matrix parameters omitted in display. Use depth=2 to show them.
    Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  -0.18   0.33      -0.70       0.34 11760    1
bA  0.03   0.01       0.02       0.04  9976    1
```

Getting very close to zero, but still almost all the posterior probability mass is above zero. It seems what the increasing error is doing is making the inferred relationship smaller in magnitude, but it isn't creating any doubt that it is positive.

Can we pull off a standard error of 100? Let's try:

R code 12.18

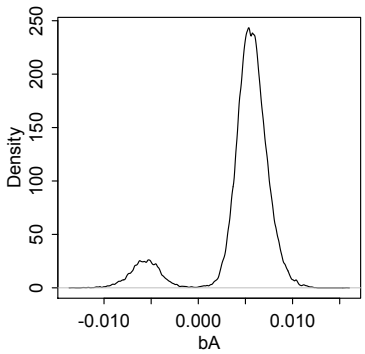
```
m14H2 <- map2stan(
    alist(
        MATINGS ~ dpois(lambda),
        log(lambda) <- a + bA*AGE_est[i],
        AGE ~ dnorm( AGE_est , 100 ),
        a ~ dnorm(0,10),
        bA ~ dnorm(0,1)
    ),
    start=list(AGE_est=d$AGE),
    data=d , chains=4 , cores=4 , warmup=2000 , iter=1e4 ,
    control=list(adapt_delta=0.99) , WAIC=FALSE )
precis(m14H2)
```

```
41 vector or matrix parameters omitted in display. Use depth=2 to show them.
   Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a  0.66   0.22       0.31       1.03   245 1.01
bA 0.00   0.00       0.00       0.01   100 1.02
```

Now the inferred relationship is essentially zero. But let's plot the posterior for `bA`:

R code 12.19

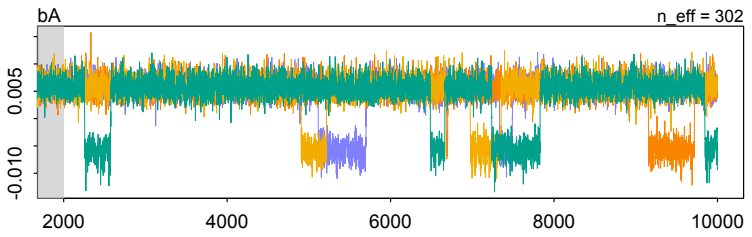
```
bA <- extract.samples(m14H2)$bA
dens( bA , xlab="bA" )
```



There are now two modes in the posterior: A major mode above zero and a minor mode an equal distance below zero. The measurement error has gotten so large now that it becomes possible that the relationship between age and matings is negative. But note that the vast majority of the probability mass remains above zero, although very close to it. If you inspect the trace plot, you can see the chains switching between these two modes:

R code 12.20

```
plot(m14H2,pars="bA",window=c(2000,10000),n_col=1)
```



This kind of posterior distribution is challenging to sample from. You'll see another example in the next problem.

14H3. Run the provided code to generate the data. It should look like this:

R code 12.21

```
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 12.22

```
precis( lm(y~x,d) )
```

```
            Mean StdDev  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 12.23

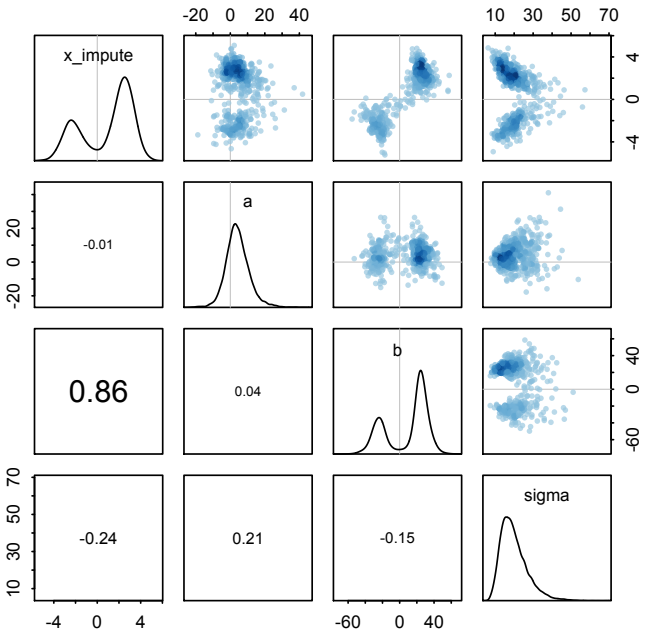
```
m14H3 <- map2stan(
    alist(
        y ~ dnorm(mu,sigma),
        mu <- a + b*x,
        x ~ dnorm(0,1),
        c(a,b) ~ dnorm(0,100),
        sigma ~ dcauchy(0,1)
    ),
    data=d , chains=4 , warmup=1000 , iter=4000 ,
    control=list(adapt_delta=0.99) , WAIC=FALSE )
precis(m14H3)
```

```
          Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
x_impute  0.90   2.38      -2.97       3.80   215 1.03
a         4.21   6.92      -6.43      14.36  2415 1.00
b         8.94  24.88     -32.43      36.06   225 1.03
sigma    19.93   7.10       9.59      29.77  1534 1.01
```

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`:

```
pairs(m14H3)
```

R code 12.24



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.

```
post <- extract.samples(m14H3)
post_pos <- post
post_neg <- post
```

R code 12.25

```
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 12.26

```
x_seq <- seq(from=-2.6,to=2.6,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 12.27

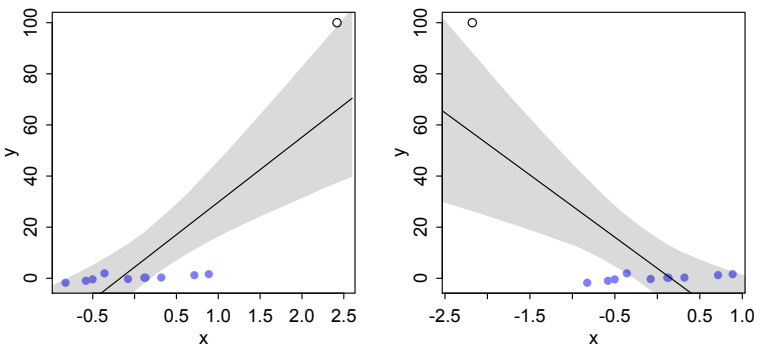
```
plot( y ~ x , d , pch=16 , col=rangi2 , xlim=c(-0.85,2.5) )
x_impute <- mean(post_pos$x_impute)
points( x_impute , 100 )
lines( x_seq , mu_mu )
shade( mu_PI , x_seq )
```

And before showing the result, let's also compute the negative relationship, so we can easily compare them:

R code 12.28

```
mu <- sapply( x_seq , mu_link , post=post_neg )
mu_mu <- apply(mu,2,mean)
mu_PI <- apply(mu,2,PI)
plot( y ~ x , d , pch=16 , col=rangi2 , xlim=c(-2.4,0.9) )
x_impute <- mean(post_neg$x_impute)
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.