4 Linear Models

History has been unkind to Ptolemy. Claudius Ptolemy (born 90 CE, died 168 CE) was an Egyptian mathematician and astronomer, most famous for his geocentric model of the solar system. These days, when scientists wish to mock someone, they might compare him to a supporter of the geocentric model. But Ptolemy was a genius. His mathematical model of the motions of the planets (FIGURE 4.1) was extremely accurate. To achieve its accuracy, it employed a device known as an *epicycle*, a circle on a circle. It is even possible to have epi-epicycles, circles on circles on circles. With enough epicycles in the right places, Ptolemy's model could predict with accuracy greater than anyone had achieved before him. And so the model was utilized for over a thousand years. And Ptolemy and people like him, toiling over centuries, worked it all out without the aid of a computer. Anyone should be flattered to be compared to Ptolemy.

The trouble of course is that the geocentric model is wrong, in many respects. If you used it to plot the path of your Mars probe, you'd miss the red planet by quite a distance. But for spotting Mars in the night sky, it remains an excellent model. It would have to be re-calibrated every century or so, depending upon which heavenly body you wish to locate. But the geocentric model continues to make useful predictions, provided those predictions remain within a narrow domain of questioning.

The strategy of using epicycles might seem crazy, once you know the correct structure of the solar system. But it turns out that the ancients had hit upon a generalized system of approximation. Given enough circles embedded in enough places, the Ptolemaic strategy is the same as a *Fourier series*, a way of decomposing a periodic function (like an orbit) into a series of sine and cosine functions. So no matter the actual arrangement of planets and moons, a geocentric model can be built to describe their paths against the night sky.

LINEAR REGRESSION is the geocentric model of applied statistics. By "linear regression," we will mean a family of simple statistical golems that attempt to learn about the mean and variance of some measurement, using an additive combination of other measurements. Like geocentrism, linear regression can usefully describe a very large variety of natural phenomena. Like geocentrism, linear regression is a descriptive model that corresponds to many different process models. If we read its structure too literally, we're likely to make mistakes. But used wisely, these little linear golems continue to be useful.

This chapter introduces linear regression as a Bayesian procedure. Under a probability interpretation, which is necessary for Bayesian work, linear regression uses a Gaussian (normal) distribution to describe our golem's uncertainty about some measurement of interest. This type of model is simple, flexible, and commonplace. Like all statistical models, it is not universally useful. But linear regression has a strong claim to being foundational, in the

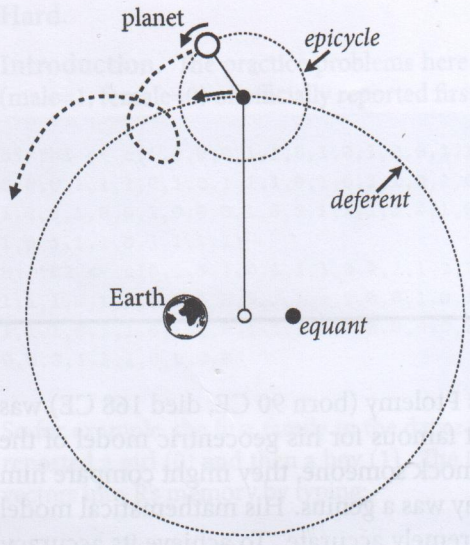


FIGURE 4.1. The Ptolemaic Universe, in which complex motion of the planets in the night sky was explained by orbits within orbits, called *epicycles*. The model is incredibly wrong, yet makes quite good predictions.

sense that once you learn to build and interpret linear regression models, you can quickly move on to other types of regression which are less normal.

4.1. Why normal distributions are normal

Suppose you and a thousand of your closest friends line up on the halfway line of a soccer field (football pitch). Each of you has a coin in your hand. At the sound of the whistle, you begin flipping the coins. Each time a coin comes up heads, that person moves one step towards the left-hand goal. Each time a coin comes up tails, that person moves one step towards the right-hand goal. Each person flips the coin 16 times, follows the implied moves, and then stands still. Now we measure the distance of each person from the halfway line. Can you predict what proportion of the thousand people who are standing on the halfway line? How about the proportion 5 yards left of the line?

It's hard to say where any individual person will end up, but you can say with great confidence what the collection of positions will be. The distances will be distributed in approximately normal, or Gaussian, fashion. This is true even though the underlying distribution is binomial. It does this because there are so many more possible ways to realize a sequence of left-right steps that sums to zero. There are slightly fewer ways to realize a sequence that ends up one step left or right of zero, and so on, with the number of possible sequences declining in the characteristic bell curve of the normal distribution.

4.1.1. Normal by addition. Let's see this result, by simulating this experiment in R. To show that there's nothing special about the underlying coin flip, assume instead that each step is different from all the others, a random distance between zero and one yard. Thus a coin is flipped, a distance between zero and one yard is taken in the indicated direction, and the process repeats. To simulate this, we generate for each person a list of 16 random numbers between -1 and 1. These are the individual steps. Then we add these steps together to get the position after 16 steps. Then we need to replicate this procedure 1000 times. This is the sort of task that would be harrowing in a point-and-click interface, but it is made trivial by the command line. Here's a single line to do the whole thing:

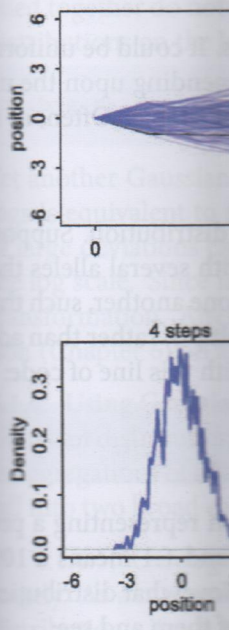


FIGURE 4.2. tribution. Th empirical dis imposed in t

```
pos <- replicate(
```

You can plot the dis
hist(pos) and plot
walks and how their
plots 100 different, in
dashes indicate the lo
after 4, 8, and 16 step
matic, after 16 steps,
the Gaussian distribu
even larger numbers
lining on the Gaussia
arbitrary ways, witho

Any process that
a normal. But it's no
Here's a conceptual v
distribution, each sam
When we begin to ad
A large positive fluctu
more chances for eac
in the opposite direct

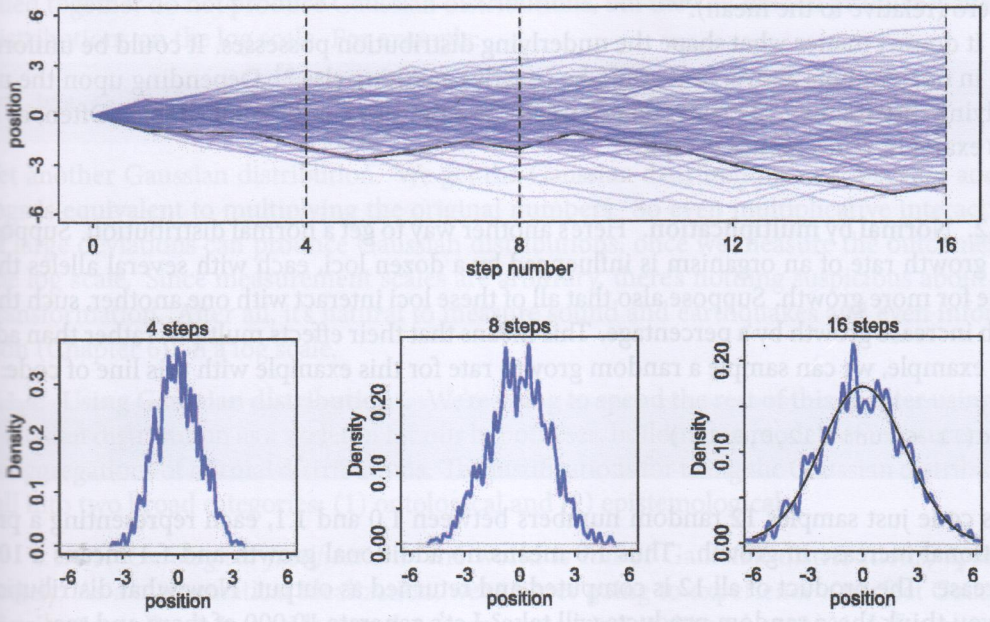


FIGURE 4.2. Random walks on the soccer field converge to a normal distribution. The more steps are taken, the closer the match between the real empirical distribution of positions and the ideal normal distribution, superimposed in the last plot in the bottom panel.

```
pos <- replicate( 1000 , sum( runif(16,-1,1) ) )
```

R code 4.1

You can plot the distribution of final positions in a number of different ways, including `hist(pos)` and `plot(density(pos))`. In FIGURE 4.2, I show the result of these random walks and how their distribution evolves as the number of steps increases. The top panel plots 100 different, independent random walks, with one highlighted in black. The vertical dashes indicate the locations corresponding to the distribution plots underneath, measured after 4, 8, and 16 steps. Although the distribution of positions starts off seemingly idiosyncratic, after 16 steps, it has already taken on a familiar outline. The familiar "bell" curve of the Gaussian distribution is emerging from the randomness. Go ahead and experiment with even larger numbers of steps to verify for yourself that the distribution of positions is stabilizing on the Gaussian. You can square the step sizes and transform them in a number of arbitrary ways, without changing the result: Normality emerges. Where does it come from?

Any process that adds together random values from the same distribution converges to a normal. But it's not easy to grasp why addition should result in a bell curve of sums.[61] Here's a conceptual way to think of the process. Whatever the average value of the source distribution, each sample from it can be thought of as a fluctuation from that average value. When we begin to add these fluctuations together, they also begin to cancel one another out. A large positive fluctuation will cancel a large negative one. The more terms in the sum, the more chances for each fluctuation to be canceled by another, or by a series of smaller ones in the opposite direction. So eventually the most likely sum, in the sense that there are the

most ways to realize it, will be a sum in which every fluctuation is canceled by another, a sum of zero (relative to the mean).[62]

It doesn't matter what shape the underlying distribution possesses. It could be uniform, like in our example above, or it could be (nearly) anything else.[63] Depending upon the underlying distribution, the convergence might be slow, but it will be inevitable. Often, as in this example, convergence is rapid.

4.1.2. Normal by multiplication. Here's another way to get a normal distribution. Suppose the growth rate of an organism is influenced by a dozen loci, each with several alleles that code for more growth. Suppose also that all of these loci interact with one another, such that each increase growth by a percentage. This means that their effects multiply, rather than add. For example, we can sample a random growth rate for this example with this line of code:

R code 4.2
```
prod( 1 + runif(12,0,0.1) )
```

This code just samples 12 random numbers between 1.0 and 1.1, each representing a proportional increase in growth. Thus 1.0 means no additional growth and 1.1 means a 10% increase. The product of all 12 is computed and returned as output. Now what distribution do you think these random products will take? Let's generate 10,000 of them and see:

R code 4.3
```
growth <- replicate( 10000 , prod( 1 + runif(12,0,0.1) ) )
dens( growth , norm.comp=TRUE )
```

The reader should execute this code in R and see that the distribution is approximately normal again. I said normal distributions arise from summing random fluctuations, which is true. But the effect at each locus was multiplied by the effects at all the others, not added. So what's going on here?

We again get convergence towards a normal distribution, because the effect at each locus is quite small. Multiplying small numbers is approximately the same as addition. For example, if there are two loci with alleles increasing growth by 10% each, the product is:

$$1.1 \times 1.1 = 1.21$$

We could also approximate this product by just adding the increases, and be off by only 0.01:

$$1.1 \times 1.1 = (1 + 0.1)(1 + 0.1) = 1 + 0.2 + 0.01 \approx 1.2$$

The smaller the effect of each locus, the better this additive approximation will be. In this way, small effects that multiply together are approximately additive, and so they also tend to stabilize on Gaussian distributions. Verify this for yourself by comparing:

R code 4.4
```
big <- replicate( 10000 , prod( 1 + runif(12,0,0.5) ) )
small <- replicate( 10000 , prod( 1 + runif(12,0,0.01) ) )
```

The interacting growth deviations, as long as they are sufficiently small, converge to a Gaussian distribution. In this way, the range of casual forces that tend towards Gaussian distributions extends well beyond purely additive interactions.

4.1.3. Normal by log-multiplication. But wait, there's more. Large deviates that are multiplied together do not produce Gaussian distributions, but they do tend to produce Gaussian distributions on the log scale. For example:

R code 4.5

```
log.big <- replicate( 10000 , log(prod(1 + runif(12,0,0.5))) )
```

Yet another Gaussian distribution. We get the Gaussian distribution back, because adding logs is equivalent to multiplying the original numbers. So even multiplicative interactions of large deviations can produce Gaussian distributions, once we measure the outcomes on the log scale. Since measurement scales are arbitrary, there's nothing suspicious about this transformation. After all, it's natural to measure sound and earthquakes and even information (Chapter 6) on a log scale.

4.1.4. Using Gaussian distributions. We're going to spend the rest of this chapter using the Gaussian distribution as a skeleton for our hypotheses, building up models of measurements as aggregations of normal distributions. The justifications for using the Gaussian distribution fall into two broad categories: (1) ontological and (2) epistemological.

4.1.4.1. *Ontological justification.* The world is full of Gaussian distributions, approximately. As a mathematical idealization, we're never going to experience a perfect Gaussian distribution. But it is a widespread pattern, appearing again and again at different scales and in different domains. Measurement errors, variations in growth, and the velocities of molecules all tend towards Gaussian distributions. These processes do this because at their heart, these processes add together fluctuations. And repeatedly adding finite fluctuations results in a distribution of sums that have shed all information about the underlying process, aside from mean and spread.

One consequence of this is that statistical models based on Gaussian distributions cannot reliably identify micro-process. This recalls the modeling philosophy from Chapter 1 (page 6). But it also means that these models can do useful work, even when they cannot identify process. If we had to know the development biology of height before we could build a statistical model of height, human biology would be sunk.

There are many other patterns in nature, so make no mistake in assuming that the Gaussian pattern is universal. In later chapters, we'll see how other useful and common patterns, like the exponential and gamma and Poisson, also arise from natural processes. The Gaussian is a member of a family of fundamental natural distributions known as the **EXPONENTIAL FAMILY**. All of the members of this family are important for working science, because they populate our world.

4.1.4.2. *Epistemological justification.* But the natural occurrence of the Gaussian distribution is only one reason to build models around it. Another route to justifying the Gaussian as our choice of skeleton, and a route that will help us appreciate later why it is often a poor choice, is that it represents a particular state of ignorance. When all we know or are willing to say about a distribution of measures (measures are continuous values on the real number line) is their mean and variance, then the Gaussian distribution arises as the most consistent with our assumptions.

That is to say that the Gaussian distribution is the most natural expression of our state of ignorance, because if all we are willing to assume is that a measure has finite variance, the Gaussian distribution is the shape that can be realized in the largest number of ways and does not introduce any new assumptions. It is the least surprising and least informative

assumption to make. In this way, the Gaussian is the distribution most consistent with our assumptions. Or rather, it is the most consistent with our golem's assumptions. If you don't think the distribution should be Gaussian, then that implies that you know something else that you should tell your golem about, something that would improve inference.

This epistemological justification is premised on **INFORMATION THEORY** and **MAXIMUM ENTROPY**. We'll dwell on information theory in Chapter 6 and maximum entropy in Chapter 9. Then in later chapters, other common and useful distributions will be used to build *generalized linear models* (GLMs). When these other distributions are introduced, you'll learn the constraints that make them the uniquely most appropriate (consistent with our assumptions) distributions.

For now, let's take the ontological and epistemological justifications of just the Gaussian distribution as reasons to start building models of measures around it. Throughout all of this modeling, keep in mind that using a model is not equivalent to swearing an oath to it. The golem is your servant, not the other way around. And so the golem's beliefs are not your own. There is no contract that forces us to believe the assumptions of our models. They just need to be useful robots.

Overthinking: Gaussian distribution. You don't have to memorize the Gaussian probability distribution formula to make good use of it. You're computer already knows it. But a little knowledge of its form can help demystify it. The probability *density* (see below) of some value y, given a Gaussian (normal) distribution with mean μ and standard deviation σ, is:

$$p(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

This looks monstrous. But the important bit is just the $(y-\mu)^2$ bit. This is the part that gives the normal distribution its fundamental shape, a quadratic shape. Once you exponentiate the quadratic shape, you get the classic bell curve. The rest of it just scales and standardizes the distribution so that it sums to one, as all probability distributions must. But an expression as simple as $\exp(-y^2)$ yields the Gaussian prototype.

The Gaussian is also a continuous distribution, unlike the binomial probabilities of earlier chapters. This means that the value y in the Gaussian distribution can be any continuous value. The binomial, in contrast, requires integers. Probability distributions with only discrete outcomes, like the binomial, are usually called *probability mass* functions and denoted Pr. Continuous ones like the Gaussian are called *probability density* functions, denoted with p or just plain old f, depending upon author and tradition. For mathematical reasons, probability densities, but not masses, can be greater than 1. Try `dnorm(0,0,0.1)`, for example, which is the way to make R calculate $p(0|0,0.1)$. The answer, about 4, is no mistake. Probability *density* is the rate of change in cumulative probability. So where cumulative probability is increasing rapidly, density can easily exceed 1. But if we calculate the area under the density function, it will never exceed 1. Such areas are also called *probability mass*.

You can usually ignore all these density/mass details while doing computational work. In the conceptual parts of this book, I'll treat them identically. But it's good to be aware of the distinction. Sometimes the difference matters.

The Gaussian distribution is routinely seen without σ but with another parameter, τ. The parameter τ in this context is usually called *precision* and defined as $\tau = 1/\sigma^2$. This change of parameters gives us the equivalent formula (just substitute $\sigma = 1/\sqrt{\tau}$):

$$p(y|\mu,\tau) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\tfrac{1}{2}\tau(y-\mu)^2\right)$$

This form is common in Bayesian data analysis, and Bayesian model fitting software, such as BUGS or JAGS, sometimes requires using τ rather than σ.

4.2. A language for describing models

This book adopts a standard language for describing and coding statistical models. You find this language in many statistical texts and in nearly all statistical journals, as it is general to both Bayesian and non-Bayesian modeling. Scientists increasingly use this same language to describe their statistical methods, as well. So learning this language is an investment, no matter where you are headed next.

Here's the approach, in abstract. There will be many examples later. These numbered items describe the choices that will be encoded in your model description.

(1) First, we recognize a set of measurements that we hope to predict or understand, the *outcome* variable or variables.
(2) For each of these outcome variables, we define a likelihood distribution that defines the plausibility of individual observations. In linear regression, this distribution is always Gaussian.
(3) Then we recognize a set of other measurements that we hope to use to predict or understand the outcome. Call these *predictor* variables.
(4) We relate the exact shape of the likelihood distribution—its precise location and variance and other aspects of its shape, if it has them—to the predictor variables. In choosing a way to relate the predictors to the outcomes, we are forced to name and define all of the parameters of the model.
(5) Finally, we choose priors for all of the parameters in the model. These priors define the initial information state of the model, before seeing the data.

After all these decisions are made—and most of them will come to seem automatic to you before long—we summarize the model with something mathy like:

$$\begin{aligned} \text{outcome}_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \beta \times \text{predictor}_i \\ \beta &\sim \text{Normal}(0, 10) \\ \sigma &\sim \text{HalfCauchy}(0, 1) \end{aligned}$$

If that doesn't make much sense, good. That indicates that you are holding the right textbook, since this book teaches you how to read and write these mathematical model descriptions. We won't do any mathematical manipulation of them. Instead, they provide an unambiguous way to define and communicate our models. Once you get comfortable with their grammar, when you start reading these mathematical descriptions in other books or in scientific journals, you'll find them less obtuse.

The approach above surely isn't the only way to describe statistical modeling, but it is a widespread and productive language. Once a scientist learns this language, it becomes easier to communicate the assumptions of our models. We no longer have to remember seemingly arbitrary lists of bizarre conditions like *homoscedasticity* (constant variance), because we can just read these conditions from the model definitions. We will also be able to see natural ways to change these assumptions, instead of feeling trapped within some procrustean model type, like regression or multiple regression or ANOVA or ANCOVA or such. These are all the same kind of model, and that fact becomes obvious once we know how to talk about models as mappings of one set of variables through a probability distribution onto another set of variables. Fundamentally, these models define the ways values of some variables can arise, given values of other variables (Chapter 2).

4.2.1. Re-describing the globe tossing model. It's good to work with examples. Recall the proportion of water problem from previous chapters. The model in that case was always:

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

where w was the observed count of water samples, n was the total number of samples, and p was the proportion of water on the actual globe. Read the above statement as:

☆ The count w is distributed binomially with sample size n and probability p. The prior for p is assumed to be uniform between zero and one.

even tacit or unknown ones

Once we know the model in this way, we automatically know all of its assumptions. We know the binomial distribution assumes that each sample (globe toss) is independent of the others, and so we also know that the model assumes that sample points are independent of one another.

For now, we'll focus on simple models like the above. In these models, the first line defines the likelihood function used in Bayes' theorem. The other lines define priors. Both of the lines in this model are STOCHASTIC, as indicated by the $\sim$ symbol. A stochastic relationship is just a mapping of a variable or parameter onto a distribution. It is *stochastic* because no single instance of the variable on the left is known with certainty. Instead, the mapping is probabilistic: Some values are more plausible than others, but very many different values are plausible under any model. Later, we'll have models with deterministic definitions in them as well.

Overthinking: From model definition to Bayes' theorem. To relate the mathematical format above to Bayes' theorem, you could use the model definition to define the posterior distribution:

$$\Pr(p|w, n) = \frac{\text{Binomial}(w|n, p)\text{Uniform}(p|0, 1)}{\int \text{Binomial}(w|n, p)\text{Uniform}(p|0, 1)dp}$$

That monstrous denominator is just the average likelihood again. It standardizes the posterior to sum to 1. The action is in the numerator, where the posterior probability of any particular value of p is seen again to be proportional to the product of the likelihood and prior. In R code form, this is the same grid approximation calculation you've been using all along. In a form recognizable as the above expression:

R code 4.6

```
w <- 6; n <- 9;
p_grid <- seq(from=0,to=1,length.out=100)
posterior <- dbinom(w,n,p_grid)*dunif(p_grid,0,1)
posterior <- posterior/sum(posterior)
```

Compare to the calculations in earlier chapters.

4.3. A Gaussian model of height

Let's build a linear regression model now. Well, it'll be a "regression" once we have a predictor variable in it. For now, we'll get the scaffold in place and construct the predictor variable in the next section.

We'll work through this material by using real sets of data. In this case, we want a single measurement variable to model as a Gaussian distribution. There will be two parameters

describing the distribution's shape, the mean μ and the standard deviation σ. Bayesian updating will allow us to consider every possible combination of values for μ and σ and to score each combination by its relative plausibility, in light of the data. These relative plausibilities are the posterior probabilities of each combination of values μ, σ.

Another way to say the above is this. There are an infinite number of possible Gaussian distributions. Some have small means. Others have large means. Some are wide, with a large σ. Others are narrow. We want our Bayesian machine to consider every possible distribution, each defined by a combination of μ and σ, and rank them by posterior plausibility. Posterior plausibility provides a measure of the logical compatibility of each possible distribution with the data and model.

In practice we'll use approximations to the formal analysis. So we won't really consider every possible value of μ and σ. But that won't cost us anything in most cases. Instead the thing to worry about is keeping in mind that the "estimate" here will be the entire posterior distribution, not any point within it. And as a result, the posterior distribution will be a distribution of Gaussian distributions. Yes, a distribution of distributions. If that doesn't make sense yet, then that just means you are being honest with yourself. Hold on, work hard, and it will make plenty of sense before long.

4.3.1. The data. The data contained in `data(Howell1)` are partial census data for the Dobe area !Kung San, compiled from interviews conducted by Nancy Howell in the late 1960s.[64] For the non-anthropologists reading along, the !Kung San are the most famous foraging population of the 20th century, largely because of detailed quantitative studies by people like Howell.

Load the data and place them into a convenient object with:

R code 4.7

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

What you have now is a *data frame* named simply `d`. I use the name `d` over and over again in this book to refer to the data frame we are working with at the moment. I keep its name short to save you typing. A *data frame* is a special kind of object in R. It is a table with named columns, corresponding to variables, and numbered rows, corresponding to individual cases. In this example, the cases are individuals. Inspect the structure of the data frame, the same way you'd inspect the structure of any symbol in R:

R code 4.8

```
str( d )
```

```
'data.frame': 544 obs. of  4 variables:
 $ height: num  152 140 137 157 145 ...
 $ weight: num  47.8 36.5 31.9 53 41.3 ...
 $ age   : num  63 63 65 41 51 35 32 27 19 54 ...
 $ male  : int  1 0 0 1 0 1 0 1 0 1 ...
```

This data frame contains four columns. Each column has 544 entries, so there are 544 individuals in these data. Each individual has a recorded height (centimeters), weight (kilograms), age (years), and "maleness" (0 indicating female and 1 indicating male).

We're going to work with just the `height` column, for the moment. The column containing the heights is really just a regular old R *vector*, the kind of list we have been working with in many of the code examples. You can access this vector by using its name:

R code 4.9
```
d$height
```

Read the symbol `$` as *extract*, as in *extract the column named* `height` *from the data frame* `d`. All it does is give you the column that follows it.

Overthinking: Data frames. It might seem like this whole data frame thing is kind of annoying, right now. If we're working with only one column here, why bother with this `d` thing at all? You don't have to use a data frame, as you can just pass raw vectors to every command we'll use in this book. But keeping related variables in the same data frame is a huge convenience. Once we have more than one variable, and we wish to model one as a function of the others, you'll better see the value of the data frame. You won't have to wait long.

More technically, a data frame is a special kind of `list` in R. So you access the individual variables with the usual list "double bracket" notation, like `d[[1]]` for the first variable or `d[['x']]` for the variable named `x`. Unlike regular lists, however, data frames force all variables to have the same length. That isn't always a good thing. And that's why some statistical packages, like the powerful Stan Markov chain sampler (mc-stan.org), accept plain lists of data, rather than proper data frames.

All we want for now are heights of adults in the sample. The reason to filter out non-adults for now is that height is strongly correlated with age, before adulthood. Later in the chapter, I'll ask you to tackle the age problem. But for now, better to postpone it. You can filter the data frame down to individuals of age 18 or greater with:

R code 4.10
```
d2 <- d[ d$age >= 18 , ]
```

We'll be working with the data frame `d2` now. It should have 352 rows (individuals) in it.

Overthinking: Index magic. The square bracket notation used in the code above is *index* notation. It is very powerful, but also quite compact and confusing. The data frame `d` is a matrix, a rectangular grid of values. You can access any value in the matrix with `d[row,col]`, replacing `row` and `col` with row and column numbers. If `row` or `col` are lists of numbers, then you get more than one row or column. If you leave the spot for `row` or `col` blank, then you get all of whatever you leave blank. For example, `d[ 3 , ]` gives all columns at row 3. Typing `d[,]` just gives you the entire matrix, because it returns all rows and all columns.

So what `d[ d$age >= 18 , ]` does is give you all of the rows in which `d$age` is greater-than-or-equal-to 18. It also gives you all of the columns, because the spot after the comma is blank. The result is stored in `d2`, the new data frame containing only adults. With a little practice, you can use this square bracket index notion to perform custom searches of your data, much like performing a database query.

4.3.2. The model. Our goal is to model these values using a Gaussian distribution. First, go ahead and plot the distribution of heights, with `dens(d2$height)`. These data look rather Gaussian in shape, as is typical of height data. This may be because height is a sum of many small growth factors. As you saw at the start of the chapter, a distribution of sums tends to

converge to a Gaussian distribution. Whatever the reason, adult heights are nearly always approximately normal.

So it's reasonable for the moment to adopt the stance that the model's likelihood should be Gaussian. But be careful about choosing the Gaussian distribution only when the plotted outcome variable looks Gaussian to you. Gawking at the raw data, to try to decide how to model them, is usually not a good idea. The data could be a mixture of different Gaussian distributions, for example, and in that case you won't be able to detect the underlying normality just by eyeballing the outcome distribution. Furthermore, as mentioned earlier in this chapter, the empirical distribution needn't be actually Gaussian in order to justify using a Gaussian likelihood.

So which Gaussian distribution? There are an infinite number of them, with an infinite number of different means and standard deviations. We're ready to write down the general model and compute the plausibility of each combination of μ and σ. To define the heights as normally distributed with a mean μ and standard deviation σ, we write:

$$h_i \sim \text{Normal}(\mu, \sigma)$$

In many books you'll see the same model written as $h_i \sim \mathcal{N}(\mu, \sigma)$, which means the same thing. The symbol h refers to the list of heights, and the subscript i means *each individual element of this list*. It is conventional to use i because it stands for *index*. The index i takes on row numbers, and so in this example can take any value from 1 to 352 (the number of heights in `d2$height`). As such, the model above is saying that all the golem knows about each height measurement is defined by the same normal distribution, with mean μ and standard deviation σ. Before long, those little i's are going to show up on the right-hand side of the model definition, and you'll be able to see why we must bother with them. So don't ignore the i, even if it seems like useless ornamentation right now.

Rethinking: Independent and identically distributed. The short model above is sometimes described as assuming that the values h_i are *independent and identically distributed*, which may be abbreviated i.i.d., iid, or IID. You might even see the same model written:

$$h_i \overset{\text{iid}}{\sim} \text{Normal}(\mu, \sigma).$$

"iid" indicates that each value h_i has the same probability function, independent of the other h values and using the same parameters. A moment's reflection tells us that this is hardly ever true, in a physical sense. Whether measuring the same distance repeatedly or studying a population of heights, it is hard to argue that every measurement is independent of the others. For example, heights within families are correlated because of alleles shared through recent shared ancestry.

The i.i.d. assumption doesn't have to seem awkward, however, as long as you remember that probability is inside the golem, not outside in the world. The i.i.d. assumption is about how the golem represents its uncertainty. It is an *epistemological* assumption. It is not a physical assumption about the world, an *ontological* one, unless you insist that it is. E. T. Jaynes (1922–1998) called this the *mind projection fallacy*, the mistake of confusing epistemological claims with ontological claims.[65]

The point isn't to say epistemology trumps reality, but rather that in ignorance of such correlations the most conservative distribution to use is i.i.d.[66] This issue will return in Chapter 9. Furthermore, there is a mathematical result known as *de Finetti's theorem* that tells us that values which are EXCHANGEABLE can be approximated by mixtures of i.i.d. distributions. Colloquially, exchangeable values can be reordered. The practical impact of this is that "i.i.d." as an assumption cannot be read too literally, as different process models again correspond to the same statistical model (as argued in

Chapter 1). Even furthermore, there are many types of correlation that do little or nothing to the overall shape of a distribution, but only affect the precise sequence in which values appear. For example, pairs of sisters have highly correlated heights. But the overall distribution of female height remains almost perfectly normal. In such cases, i.i.d. remains perfectly useful, despite ignoring the correlations. Consider for example that Markov chain Monte Carlo (Chapter 8) can use highly correlated sequential samples to estimate most any iid distribution we like.

To complete the model, we're going to need some priors. The parameters to be estimated are both μ and σ, so we need a prior $\Pr(\mu, \sigma)$, the joint prior probability for all parameters. In most cases, priors are specified independently for each parameter, which amounts to assuming $\Pr(\mu, \sigma) = \Pr(\mu)\Pr(\sigma)$. Then we can write:

$$\begin{aligned} h_i &\sim \text{Normal}(\mu, \sigma) && \text{[likelihood]} \\ \mu &\sim \text{Normal}(178, 20) && [\mu \text{ prior}] \\ \sigma &\sim \text{Uniform}(0, 50) && [\sigma \text{ prior}] \end{aligned}$$

The labels on the right are not part of the model, but instead just notes to help you keep track of the purpose of each line. The prior for μ is a broad Gaussian prior, centered on 178cm, with 95% of probability between 178 ± 40.

Why 178 cm? Your author is 178 cm tall. And the range from 138 cm to 218 cm encompasses a huge range of plausible mean heights for human populations. So domain-specific information has gone into this prior. Everyone knows something about human height and can set a reasonable and vague prior of this kind. But in many regression problems, as you'll see later, using prior information is more subtle, because parameters don't always have such clear physical meaning.

Whatever the prior, it's a very good idea to plot your priors, so you have a sense of the assumption they build into the model. In this case:

R code 4.11
```
curve( dnorm( x , 178 , 20 ) , from=100 , to=250 )
```

Execute that code yourself, to see that the golem is assuming that the average height (not each individual height) is almost certainly between 140 cm and 220 cm. So this prior carries a little information, but not a lot. The σ prior is a truly flat prior, a uniform one, that functions just to constrain σ to have positive probability between zero and 50cm. View it with:

R code 4.12
```
curve( dunif( x , 0 , 50 ) , from=-10 , to=60 )
```

A standard deviation like σ must be positive, so bounding it at zero makes sense. How should we pick the upper bound? In this case, a standard deviation of 50cm would imply that 95% of individual heights lie within 100cm of the average height. That's a very large range.

All this talk is nice, but it'll help to really see what these priors imply about the distribution of individual heights. You didn't specify a prior probability distribution of heights directly, but once you've chosen priors for μ and σ, these imply a prior distribution of individual heights. You can quickly simulate heights by sampling from the prior, like you sampled from the posterior back in Chapter 3. Remember, every posterior is also potentially a prior for a subsequent analysis, so you can process priors just like posteriors.

R code 4.13

```
sample_mu <- rnorm( 1e4 , 178 , 20 )
sample_sigma <- runif( 1e4 , 0 , 50 )
prior_h <- rnorm( 1e4 , sample_mu , sample_sigma )
dens( prior_h )
```

The density plot you get shows a vaguely bell-shaped density with thick tails. It is the expected distribution of heights, averaged over the prior. Notice that the prior probability distribution of height is not itself Gaussian. This is okay. The distribution you see is not an empirical expectation, but rather the distribution of relative plausibilities of different heights, before seeing the data.

Play around with the numbers in the priors above, to explore their effects on the prior probability density of heights.

Rethinking: A farewell to epsilon. Some readers will have already met an alternative notation for a Gaussian linear model:

$$h_i = \mu + \epsilon_i$$
$$\epsilon_i \sim \text{Normal}(0, \sigma)$$

This is equivalent to the $h_i \sim \text{Normal}(\mu, \sigma)$ form, with the ϵ standing in for the Gaussian density. But this ϵ form is poor form. The reason is that it does not usually generalize to other types of models. This means it won't be possible to express non-Gaussian models using tricks like ϵ. Better to learn one system that does generalize.

Overthinking: Model definition to Bayes' theorem again. It can help to see how the model definition on the previous page allows us to build up the posterior distribution. The height model, with its priors for μ and σ, defines this posterior distribution:

$$\Pr(\mu, \sigma | h) = \frac{\prod_i \text{Normal}(h_i|\mu, \sigma)\text{Normal}(\mu|178, 20)\text{Uniform}(\sigma|0, 50)}{\int\int \prod_i \text{Normal}(h_i|\mu, \sigma)\text{Normal}(\mu|178, 20)\text{Uniform}(\sigma|0, 50)d\mu d\sigma}$$

This looks monstrous, but it's the same creature as before. There are two new things that make it seem complicated. The first is that there is more than one observation in h, so to get the joint likelihood across all the data, we have to compute the probability for each h_i and then multiply all these likelihoods together. The product on the right-hand side takes care of that. The second complication is the two priors, one for μ and one for σ. But these just stack up. In the grid approximation code in the section to follow, you'll see the implications of this definition in the R code. Everything will be calculated on the log scale, so multiplication will become addition. But otherwise it's just a matter of executing Bayes' theorem.

4.3.3. Grid approximation of the posterior distribution.

Since this is the first Gaussian model in the book, and indeed the first model with more than one parameter, it's worth quickly mapping out the posterior distribution through brute force calculations. This isn't the approach I encourage in any other place, because it is laborious and computationally expensive. Indeed, it is usually so impractical as to be essentially impossible. But as always, it is worth knowing what the target actually looks like, before you start accepting approximations of it. A little later in this chapter, you'll use quadratic approximation to estimate the posterior distribution, and that's the approach you'll use for several chapters more. Once you

have the samples you'll produce in this subsection, you can compare them to the quadratic approximation in the next.

Unfortunately, doing the calculations here requires some technical tricks that add little, if any, conceptual insight. So I'm going to present the code here without explanation. You can execute it and keep going for now, but later return and follow the endnote for an explanation of the algorithm.[67] For now, here are the guts of the golem:

R code 4.14
```
mu.list <- seq( from=140, to=160 , length.out=200 )
sigma.list <- seq( from=4 , to=9 , length.out=200 )
post <- expand.grid( mu=mu.list , sigma=sigma.list )
post$LL <- sapply( 1:nrow(post) , function(i) sum( dnorm(
                d2$height ,
                mean=post$mu[i] ,
                sd=post$sigma[i] ,
                log=TRUE ) ) )
post$prod <- post$LL + dnorm( post$mu , 178 , 20 , TRUE ) +
    dunif( post$sigma , 0 , 50 , TRUE )
post$prob <- exp( post$prod - max(post$prod) )
```

You can inspect this posterior distribution, now residing in `post$prob`, using a variety of plotting commands. You can get a simple contour plot with:

R code 4.15
```
contour_xyz( post$mu , post$sigma , post$prob )
```

Or you can plot a simple heat map with:

R code 4.16
```
image_xyz( post$mu , post$sigma , post$prob )
```

The functions `contour_xyz` and `image_xyz` are both in the `rethinking` package.

4.3.4. Sampling from the posterior. To study this posterior distribution in more detail, again I'll push the flexible approach of sampling parameter values from it. This works just like it did in Chapter 3, when you sampled values of p from the posterior distribution for the globe tossing example. The only new trick is that since there are two parameters, and we want to sample combinations of them, we first randomly sample row numbers in `post` in proportion to the values in `post$prob`. Then we pull out the parameter values on those randomly sampled rows. This code will do it:

R code 4.17
```
sample.rows <- sample( 1:nrow(post) , size=1e4 , replace=TRUE ,
    prob=post$prob )
sample.mu <- post$mu[ sample.rows ]
sample.sigma <- post$sigma[ sample.rows ]
```

You end up with 10,000 samples, with replacement, from the posterior for the height data. Take a look at these samples:

R code 4.18
```
plot( sample.mu , sample.sigma , cex=0.5 , pch=16 , col=col.alpha(rangi2,0.1) )
```

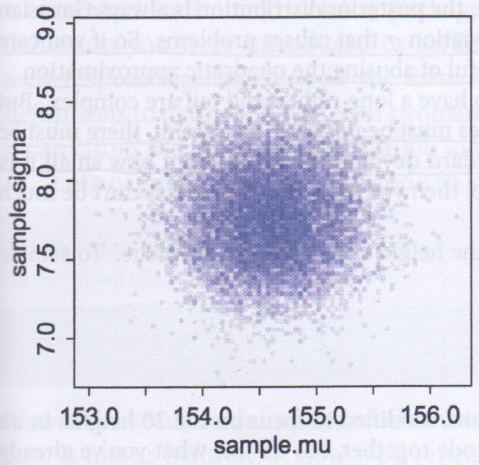


FIGURE 4.3. Samples from the posterior distribution for the heights data. The density of points is highest in the center, reflecting the most plausible combinations of μ and σ. There are many more ways for these parameter values to produce the data, conditional on the model.

I reproduce this plot in FIGURE 4.3. Note that the function `col.alpha` is part of the `rethinking` R package. All it does is make colors transparent, which helps the plot in FIGURE 4.3 more easily show density, where samples overlap. Adjust the plot to your tastes by playing around with `cex` (character expansion, the size of the points), `pch` (plot character), and the 0.1 transparency value.

Now that you have these samples, you can describe the distribution of confidence in each combination of μ and σ by summarizing the samples. Think of them like data and describe them, just like in Chapter 3. For example, to characterize the shapes of the *marginal* posterior densities of μ and σ, all we need to do is:

```
dens( sample.mu )
dens( sample.sigma )
```

R code 4.19

The jargon "marginal" here means "averaging over the other parameters." Execute the above code and inspect the plots. These densities are very close to being normal distributions. And this is quite typical. As sample size increases, posterior densities approach the normal distribution. If you look closely, though, you'll notice that the density for σ has a longer right-hand tail. I'll exaggerate this tendency a bit later, to show you that this condition is very common for standard deviation parameters.

To summarize the widths of these densities with highest posterior density intervals, just like in Chapter 3:

```
HPDI( sample.mu )
HPDI( sample.sigma )
```

R code 4.20

Since these samples are just vectors of numbers, you can compute any statistic from them that you could from ordinary data. If you want the mean or median, just use the corresponding R functions.

Overthinking: Sample size and the normality of σ's posterior. Before moving on to using quadratic approximation (`map`) as shortcut to all of this inference, it is worth repeating the analysis of the height data above, but now with only a fraction of the original data. The reason to do this is to demonstrate

that, in principle, the posterior is not always so Gaussian in shape. There's no trouble with the mean, μ. For a Gaussian likelihood and a Gaussian prior on μ, the posterior distribution is always Gaussian as well, regardless of sample size. It is the standard deviation σ that causes problems. So if you care about σ—often people do not—you do need to be careful of abusing the quadratic approximation.

The deep reasons for the posterior of σ tending to have a long right-hand tail are complex. But a useful way to conceive of the problem is that variances must be positive. As a result, there must be more uncertainty about how big the variance (or standard deviation) is than about how small it is. For example, if the variance is estimated to be near zero, then you know for sure that it can't be much smaller. But it could be a lot bigger.

Let's quickly analyze only 20 of the heights from the height data to reveal this issue. To sample 20 random heights from the original list:

R code 4.21
```
d3 <- sample( d2$height , size=20 )
```

Now I'll repeat all the code from the previous subsection, modified to focus on the 20 heights in `d3` rather than the original data. I'll compress all of the code together, but it's just what you've already seen above.

R code 4.22
```
mu.list <- seq( from=150, to=170 , length.out=200 )
sigma.list <- seq( from=4 , to=20 , length.out=200 )
post2 <- expand.grid( mu=mu.list , sigma=sigma.list )
post2$LL <- sapply( 1:nrow(post2) , function(i)
    sum( dnorm( d3 , mean=post2$mu[i] , sd=post2$sigma[i] ,
    log=TRUE ) ) )
post2$prod <- post2$LL + dnorm( post2$mu , 178 , 20 , TRUE ) +
    dunif( post2$sigma , 0 , 50 , TRUE )
post2$prob <- exp( post2$prod - max(post2$prod) )
sample2.rows <- sample( 1:nrow(post2) , size=1e4 , replace=TRUE ,
    prob=post2$prob )
sample2.mu <- post2$mu[ sample2.rows ]
sample2.sigma <- post2$sigma[ sample2.rows ]
plot( sample2.mu , sample2.sigma , cex=0.5 ,
    col=col.alpha(rangi2,0.1) ,
    xlab="mu" , ylab="sigma" , pch=16 )
```

After executing the code above, you'll see another scatter plot of the samples from the posterior density, but this time you'll notice a distinctly longer tail at the top of the cloud of points. You should also inspect the marginal posterior density for σ, averaging over μ, produced with:

R code 4.23
```
dens( sample2.sigma , norm.comp=TRUE )
```

This code will also show a normal approximation with the same mean and variance. Now you can see that the posterior for σ is not Gaussian, but rather has a long tail of uncertainty towards higher values.

4.3.5. Fitting the model with `map`. Now we leave grid approximation behind and move on to one of the great engines of applied statistics, the quadratic approximation. Our interest in quadratic approximation, recall, is as a handy way to quickly make inferences about the shape of the posterior. The posterior's peak will lie at the *maximum a posteriori* estimate (MAP), and we can get a useful image of the posterior's shape by using the quadratic approximation of the posterior distribution at this peak.

To find the values of μ and σ that maximize the posterior probability, we'll use `map`, a command in the `rethinking` package. The way that `map` works is by using the model definition you were introduced to earlier in this chapter. Each line in the definition has a corresponding definition in the form of R code. The engine inside `map` then uses these definitions to define the posterior probability at each combination of parameter values. Then it can climb the posterior distribution and find the peak, its MAP.

Let's begin by repeating the code to load the data and select out the adults:

R code 4.24

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

Now we're ready to define the model, using R's formula syntax. The model definition in this case is just as before, but now we'll repeat it with each corresponding line of R code shown on the right-hand margin:

$$h_i \sim \text{Normal}(\mu, \sigma) \qquad \texttt{height ~ dnorm(mu,sigma)}$$
$$\mu \sim \text{Normal}(178, 20) \qquad \texttt{mu ~ dnorm(156,10)}$$
$$\sigma \sim \text{Uniform}(0, 50) \qquad \texttt{sigma ~ dunif(0,50)}$$

Now place the R code equivalents into an `alist`. Here's an `alist` of the formulas above:

R code 4.25

```
flist <- alist(
    height ~ dnorm( mu , sigma ) ,
    mu ~ dnorm( 178 , 20 ) ,
    sigma ~ dunif( 0 , 50 )
)
```

Note the commas at the end of each line, except the last. These commas separate each line of the model definition.

Fit the model to the data in the data frame `d2` with:

R code 4.26

```
m4.1 <- map( flist , data=d2 )
```

After executing this code, you'll have a fit model stored in the symbol `m4.1`. Now take a look at the fit *maximum a posteriori* model:

R code 4.27

```
precis( m4.1 )
```

```
        Mean StdDev   5.5%  94.5%
mu    154.61   0.41 153.95 155.27
sigma   7.73   0.29   7.27   8.20
```

These numbers provide Gaussian approximations for each parameter's *marginal* distribution. This means the plausibility of each value of μ, after averaging over the plausibilities of each value of σ, is given by a Gaussian distribution with mean 154.6 and standard deviation 0.4.

The 5.5% and 94.5% quantiles are percentile interval boundaries, corresponding to an 89% interval. Why 89%? It's just the default. It displays a quite wide interval, so it shows a

high-probability range of parameter values. If you want another interval, such as the conventional and mindless 95%, you can use `precis(m4.1,prob=0.95)`. But I don't recommend 95% intervals, because readers will have a hard time not viewing them as significance tests. 89 is also a prime number, so if someone asks you to justify it, you can stare at them meaningfully and incant, "Because it is prime." That's no worse justification than the conventional justification for 95%.

I encourage you to compare these 89% boundaries to the HPDIs from the grid approximation earlier. You'll find that they are almost identical. When the posterior is approximately Gaussian, then this is what you should expect.

Overthinking: Start values for `map`. `map` estimates the posterior by climbing it like a hill. To do this, it has to start climbing someplace, at some combination of parameter values. Unless you tell it otherwise, `map` starts at random values sampled from the prior. But it's also possible to specify a starting value for any parameter in the model. In the example in the previous section, that means the parameters μ and σ. Here's a good list of starting values in this case:

R code 4.28

```
start <- list(
    mu=mean(d2$height),
    sigma=sd(d2$height)
)
```

These start values are good guesses of the rough location of the MAP values.

Note that the list of start values is a regular `list`, not an `alist` like the formula list is. The two functions `alist` and `list` do the same basic thing: allow you to make a collection of arbitrary R objects. They differ in one important respect: `list` evaluates the code you embed inside it, while `alist` does not. So when you define a list of formulas, you should use `alist`, so the code isn't executed. But when you define a list of start values for parameters, you should use `list`, so that code like `mean(d2$height)` will be evaluated to a numeric value.

The priors we used before are very weak, both because they are nearly flat and because there is so much data. So I'll splice in a more informative prior for μ, so you can see the effect. All I'm going to do is change the standard deviation of the prior to 0.1, so it's a very narrow prior. I'll also build the formula right into the call to `map`, so you can see how to build it all at once.

R code 4.29

```
m4.2 <- map(
        alist(
            height ~ dnorm( mu , sigma ) ,
            mu ~ dnorm( 178 , 0.1 ) ,
            sigma ~ dunif( 0 , 50 )
        ) ,
        data=d2 )
precis( m4.2 )
```

```
       Mean StdDev   5.5%  94.5%
mu   177.86   0.10 177.70 178.02
sigma  24.52   0.93  23.03  26.00
```

Notice that the estimate for μ has hardly moved off the prior. The prior was very concentrated around 178. So this is not surprising. But also notice that the estimate for σ has changed quite

a lot, even though we didn't change its prior at all. Once the golem is certain that the mean is near 178—as the prior insists—then the golem has to estimate σ conditional on that fact. This results in a different posterior for σ, even though all we changed is prior information about the other parameter.

At this point, it pays to play around with different priors, to get a sense of their effect on inference. There's so much data here that you'll have to use pretty extreme priors to have any effect on inference. Also keep in mind that most non-Bayesian estimates implicitly use flat priors. Before long, you're going to see that perfectly flat priors are hardly ever the best priors. But with as much data as we have in this example, and with such a simple model, it hardly matters which priors you use.

Overthinking: How strong is a prior? A prior can usually be interpreted as a former posterior inference, as previous data. So it's sometimes useful to talk about the strength of a prior in terms of which data would lead to the same posterior distribution, beginning with a flat prior. In the very narrow prior used just above, the $\mu \sim \text{Normal}(178, 0.1)$ prior, we can compute the implied amount of data easily, because there is a simple formula for the standard deviation of a Gaussian posterior for μ:

$$\sigma_{\text{post}} = 1/\sqrt{n}$$

(This is by no coincidence the same as the formula for the standard error of the sampling distribution of the mean, in non-Bayesian inference.) This formula implies that the implied amount of data (with mean 178) is $n = 1/\sigma_{\text{post}}^2$. So in this case the implied amount of data corresponding to the prior with standard deviation 0.1 is $n = 1/0.01 = 100$. So the $\mu \sim \text{Normal}(178, 0.1)$ is equivalent to having previously observed 100 heights with mean value 178. That's a pretty strong prior. In contrast, the former $\text{Normal}(178, 20)$ prior implies $n = 1/20^2 = 0.0025$ of an observation. This is an extremely weak prior. But of course exactly how strong or weak either prior is will depend upon how much data is used to update it.

4.3.6. Sampling from a `map` fit. The above explains how to get a MAP quadratic approximation of the posterior, using `map`. But how do you then get samples from the quadratic approximate posterior distribution? The answer is rather simple, but non-obvious, and it requires recognizing that a quadratic approximation to a posterior distribution with more than one parameter dimension—μ and σ each contribute one dimension—is just a multi-dimensional Gaussian distribution.

As a consequence, when R constructs a quadratic approximation, it calculates not only standard deviations for all parameters, but also the covariances among all pairs of parameters. Just like a mean and standard deviation (or its square, a variance) are sufficient to describe a one-dimensional Gaussian distribution, a list of means and a matrix of variances and covariances are sufficient to describe a multi-dimensional Gaussian distribution. To see this matrix of variances and covariances, for model `m4.1`, use:

R code 4.30

```
vcov( m4.1 )
```

```
              mu         sigma
mu    0.1697395865 0.0002180593
sigma 0.0002180593 0.0849057933
```

The above is a **VARIANCE-COVARIANCE** matrix. It is the multi-dimensional glue of a quadratic approximation, because it tells us how each parameter relates to every other parameter in the posterior distribution. A variance-covariance matrix can be factored into two

elements: (1) a vector of variances for the parameters and (2) a correlation matrix that tells us how changes in any parameter lead to correlated changes in the others. This decomposition is usually easier to understand. So let's do that now:

R code 4.31

```
diag( vcov( m4.1 ) )
cov2cor( vcov( m4.1 ) )
```

```
        mu      sigma
0.16973959 0.08490579

                mu        sigma
mu    1.000000000 0.001816412
sigma 0.001816412 1.000000000
```

The two-element vector in the output is the list of variances. If you take the square root of this vector, you get the standard deviations that are shown in `precis` output. The two-by-two matrix in the output is the correlation matrix. Each entry shows the correlation, bounded between −1 and +1, for each pair of parameters. The 1's indicate a parameter's correlation with itself. If these values were anything except 1, we would be worried. The other entries are typically closer to zero, and they are very close to zero in this example. This indicates that learning μ tells us nothing about σ and likewise that learning σ tells us nothing about μ. This is typical of simple Gaussian models of this kind. But it is quite rare more generally, as you'll see in later chapters.

Okay, so how do we get samples from this multi-dimensional posterior? Now instead of sampling single values from a simple Gaussian distribution, we sample vectors of values from a multi-dimensional Gaussian distribution. The `rethinking` package provides a convenience function to do exactly that:

R code 4.32

```
library(rethinking)
post <- extract.samples( m4.1 , n=1e4 )
head(post)
```

```
       mu    sigma
1 155.0031 7.443893
2 154.0347 7.771255
3 154.9157 7.822178
4 154.4252 7.530331
5 154.5307 7.655490
6 155.1772 7.974603
```

You end up with a data frame, `post`, with 10,000 (`1e4`) rows and two columns, one column for μ and one for σ. Each value is a sample from the posterior, so the mean and standard deviation of each column will be very close to the MAP values from before. You can confirm this by summarizing the samples:

R code 4.33

```
precis(post)
```

```
       Mean StdDev  |0.89  0.89|
mu   154.61   0.42 153.95 155.27
sigma  7.73   0.29   7.26   8.19
```

The 89% HPDI is displayed now, because we used samples. `|0.89` means the lower boundary, and `0.89|` means the upper boundary. Compare these values to the output from `precis(m4.1)`. And you can use `plot(post)` to see how much they resemble the samples from the grid approximation in FIGURE 4.3 (page 85).

These samples also preserve the covariance between μ and σ. This hardly matters right now, because μ and σ don't covary at all in this model. But once you add a predictor variable to your model, covariance will matter a lot.

Overthinking: Under the hood with multivariate sampling. The function `extract.samples` is for convenience. It is just running a simple simulation of the sort you conducted near the end of Chapter 3. Here's a peak at the motor. The work is done by a multi-dimensional version of `rnorm`, `mvrnorm`. The function `rnorm` simulates random Gaussian values, while `mvrnorm` simulates random vectors of multivariate Gaussian values. Here's how to use it directly to do what `extract.samples` does:

R code 4.34

```
library(MASS)
post <- mvrnorm( n=1e4 , mu=coef(m4.1) , Sigma=vcov(m4.1) )
```

You don't usually need to use `mvrnorm` directly like this, but sometimes you want to simulate multivariate Gaussian outcomes. In that case, you'll need to access `mvrnorm` directly. And of course it's always good to know a little about how the machine operates.

Overthinking: Getting σ right. The quadratic assumption for σ can be problematic, as seen on page 85. A conventional way to improve the situation is the estimate $\log(\sigma)$ instead. Why does this help? While the posterior distribution of σ will often not be Gaussian, the distribution of its logarithm can be much closer to Gaussian. So if we impose the quadratic approximation on the logarithm, rather than the standard deviation itself, we can often get a better approximation of the uncertainty. Here's how you can do this, using `map`.

R code 4.35

```
m4.1_logsigma <- map(
        alist(
            height ~ dnorm( mu , exp(log_sigma) ) ,
            mu ~ dnorm( 178 , 20 ) ,
            log_sigma ~ dnorm( 2 , 10 )
        ) , data=d2 )
```

Notice the `exp` inside the likelihood. That converts a continuous parameter, `log_sigma`, to be strictly positive, because $\exp(x) > 0$ for any real value x. Notice also the prior for `log_sigma`. Since `log_sigma` is continuous now, it can have a Gaussian prior.

When you extract samples, it is `log_sigma` that has a Gaussian distribution. To get the distribution of `sigma`, you just need to use the same `exp` as in the model definition, to get back on the natural scale:

R code 4.36

```
post <- extract.samples( m4.1_logsigma )
sigma <- exp( post$log_sigma )
```

When you have a lot of data, this won't make any noticeable difference. But the use of `exp` to effectively constrain a parameter to be positive is a robust and useful one. And it relates to *link functions*, which will be very important when we arrive at generalized linear models in Chapter 9.

4.4. Adding a predictor

What we've done above is a Gaussian model of height in a population of adults. But it doesn't really have the usual feel of "regression" to it. Typically, we are interested in modeling how an outcome is related to some predictor variable. And by including a predictor variable in a particular way, we'll have linear regression.

So now let's look at how height in these Kalahari foragers covaries with weight. This isn't the most thrilling scientific question, I know. But it is an easy relationship to start with, and it only seems dull because you don't have a theory about growth and life history in mind. If you did, it would be thrilling. Go ahead and plot height and weight against one another to get an idea of how strongly they covary:

R code 4.37

```
plot( d2$height ~ d2$weight )
```

The resulting plot is not shown here. You really should do it yourself. Once you can see the plot, you'll see that there's obviously a relationship: Knowing a person's weight helps you predict height.

To make this vague observation into a more precise quantitative model that relates values of `weight` to plausible values of `height`, we need some more technology. How do we take our Gaussian model from the previous section and incorporate predictor variables?

Rethinking: What is "regression"? Many diverse types of models are called "regression." The term has come to mean using one or more predictor variables to model the distribution of one or more outcome variables. The original use of term, however, arose from anthropologist Francis Galton's (1822–1911) observation that the sons of tall and short men tended to be more similar to the population mean, hence *regression to the mean*.[68]

This phenomenon arises statistically whenever individual measurements are assigned a common distribution, leading to *shrinkage* as each measurement informs the others. In the context of Galton's height data, attempting to predict each son's height on the basis of only his father's height is folly. Better to use the population of fathers. This leads to a prediction for each son which is similar to each father but "shrunk" towards the overall mean. Such predictions are routinely better. This same regression/shrinkage phenomenon applies at higher levels of abstraction and forms one basis of multilevel modeling (Chapter 12).

4.4.1. The linear model strategy.

The strategy is to make the parameter for the mean of a Gaussian distribution, μ, into a linear function of the predictor variable and other, new parameters that we invent. This strategy is often simply called the LINEAR MODEL. The linear model strategy instructs the golem to assume that the predictor variable has a perfectly constant and additive relationship to the mean of the outcome. The golem then computes the posterior distribution of this constant relationship.

What this means, recall, is that the machine considers every possible combination of the parameter values. With a linear model, some of the parameters now stand for the strength of association between the mean of the outcome and the value of the predictor. For each combination of values, the machine computes the posterior probability, which is a measure of relative plausibility, given the model and data. So the posterior distribution ranks the infinite possible combinations of parameter values by their logical plausibility. As a result, the posterior distribution provides relative plausibilities of the different possible strengths of association, given the assumptions you programmed into the model.

Here's how it works, in the simplest case of only one predictor variable. We'll wait until the next chapter to confront more than one predictor. Recall the basic Gaussian model:

$$h_i \sim \text{Normal}(\mu, \sigma) \quad \text{[likelihood]}$$
$$\mu \sim \text{Normal}(178, 20) \quad [\mu \text{ prior}]$$
$$\sigma \sim \text{Uniform}(0, 50) \quad [\sigma \text{ prior}]$$

Now how do we get weight into a Gaussian model of height? Let x be the mathematical name for the column of weight measurements, `d2$weight`. Now we have a predictor variable x, which is a list of measures of the same length as h. We'd like to say how knowing the values in x can help us describe or predict the values in h. To get `weight` into the model in this way, we define the mean μ as a function of the values in x. This is what it looks like, with explanation to follow:

$$h_i \sim \text{Normal}(\mu_i, \sigma) \quad \text{[likelihood]}$$
$$\mu_i = \alpha + \beta x_i \quad \text{[linear model]}$$
$$\alpha \sim \text{Normal}(178, 100) \quad [\alpha \text{ prior}]$$
$$\beta \sim \text{Normal}(0, 10) \quad [\beta \text{ prior}]$$
$$\sigma \sim \text{Uniform}(0, 50) \quad [\sigma \text{ prior}]$$

Again, I've labeled each line on the right-hand side, by the type of definition it encodes. We'll discuss each in turn.

4.4.1.1. *Likelihood.* To decode all of this, let's begin with just the likelihood, the first line of the model. This is nearly identical to before, except now there is a little index i on the μ, as well as the h. This is necessary now, because the mean μ now depends upon unique predictor values on each row i. So the little i on μ_i indicates that *the mean depends upon the row.*

4.4.1.2. *Linear model.* The mean μ is no longer a parameter to be estimated. Rather, as seen in the second line of the model, μ_i is constructed from other parameters, α and β, and the predictor variable x. This line is not a stochastic relationship—there is no $\sim$ in it, but rather an $=$ in it—because the definition of μ_i is deterministic, not probabilistic. That is to say that, once we know α and β and x_i, we also know μ_i.

The value x_i is just the weight value on row i. It refers to the same individual as the height value, h_i, on the same row. The parameters α and β are more mysterious. Where did they come from? We made them up. The parameters μ and σ are necessary and sufficient to describe a Gaussian distribution. But α and β are instead devices we invent for manipulating μ, allowing it to vary systematically across cases in the data.

You'll be making up all manner of parameters as your skills improve. One way to understand these made-up parameters is to think of them as targets of learning. Each parameter is something that must be described in the posterior density. So when you want to know something about the data, you ask your golem by inventing a parameter for it. This will make more and more sense as you progress. Here's how it works in this context. The second line of the model definition is just:

$$\mu_i = \alpha + \beta x_i$$

What this tells the regression golem is that you are asking two questions about the mean of the outcome.

(1) What is the expected height, when $x_i = 0$? The parameter α answers this question. For this reason, α is often called the *intercept*.
(2) What is the change in expected height, when x_i changes by 1 unit? The parameter β answers this question.

Jointly these two parameters, together with x, ask the golem to find a line that relates x to h, a line that passes through α when $x_i = 0$ and has slope β. That is a task that golems are very good at. It's up to you, though, to be sure it's a good question.

Rethinking: Nothing special or natural about linear models. Note that there's nothing special about the linear model, really. You can choose a different relationship between α and β and μ. For example, the following is a perfectly legitimate definition for μ_i:

$$\mu_i = \alpha \exp(-\beta x_i)$$

This does not define a linear regression, but it does define a regression model. The linear relationship we are using instead is conventional, but nothing requires that you use it. It is very common in some fields, like ecology and demography, to use functional forms for μ that come from theory, rather than the geocentrism of linear models. Models built out of substantive theory can dramatically outperform linear models of the same phenomena.[69]

Overthinking: Units and regression models. Readers who had a traditional training in physical sciences will know how to carry units through equations of this kind. For their benefit, here's the model again (omitting priors for brevity), now with units of each symbol added.

$$h_i \text{cm} \sim \text{Normal}(\mu_i \text{cm}, \sigma \text{cm})$$
$$\mu_i \text{cm} = \alpha \text{cm} + \beta \frac{\text{cm}}{\text{kg}} x_i \text{kg}$$

So you can see that β must have units of cm/kg in order for the mean μ_i to have units of cm. One of the facts that labeling with units clears up is that a parameter like β is a kind of rate. There's also a tradition called *dimensional analysis* that advocates constructing variables so that they are unit-less ratios. In this context, for example, we might divide height by a reference height, removing its units. Measurement scales are arbitrary human constructions, and sometimes the unit-less analysis is more natural.

4.4.1.3. *Priors.* The remaining lines in the model define priors for the parameters to be estimated: α, β, and σ. All of these are weak priors, leading to inferences that will echo non-Bayesian methods of model fitting, such as maximum likelihood. But as always, you should play around with the priors—plotting them, changing them, and refitting the model—to get a sense of their influence.

You've seen priors for α and σ before, although α was called μ back then. I've widened the prior for α, since as you'll see it is common for the intercept in a linear model to swing a long way from the mean of the outcome variable. The flat prior here with a huge standard deviation will allow it to move wherever it needs to. This won't make sense right now, but once you see the posterior distribution, it will.

The prior for β deserves explanation. Why have a Gaussian prior with mean zero? This prior places just as much probability below zero as it does above zero, and when $\beta = 0$, weight has no relationship to height. So many people see it as conservative assumption. And such a prior will pull probability mass towards zero, leading to more conservative estimates

than a perfectly flat prior will. But note that a Gaussian prior with standard deviation of 10 is still very weak, so the amount of conservatism it induces will be very small. As you make the standard deviation in this prior smaller, the amount of shrinkage towards zero increases and your model produces more and more conservative estimates about the relationship between height and weight. In Chapter 6, you'll see why such conservative priors are useful for improving inference.

Before fitting this model, though, consider whether this zero-centered prior really makes sense. Do you think there's just as much chance that the relationship between height and weight is negative as that it is positive? Of course you don't. In this context, such a silly prior is harmless, because there is a lot of data. But in other contexts, your golem may need a little nudge in the right direction.

Rethinking: What's the correct prior? People commonly ask what the correct prior is for a given analysis. The question sometimes implies that for any given set of data, there is a uniquely correct prior that must be used, or else the analysis will be invalid. This is a mistake. There is no more a uniquely correct prior than there is a uniquely correct likelihood. Statistical models are machines for inference. Many machines will work, but some work better than others. Priors can be wrong, but only in the same sense that a kind of hammer can be wrong for building a table.

In choosing priors, there are simple guidelines to get you started. Priors encode states of information before seeing data. So priors allow us to explore the consequences of beginning with different information. In cases in which we have good prior information that discounts the plausibility of some parameter values, like negative associations between height and weight, we can encode that information directly into priors. When we don't have such information, we still usually know enough about the plausible range of values. And you can vary the priors and repeat the analysis in order to study how different states of initial information influence inference. Frequently, there are many reasonable choices for a prior, and all of them produce the same inference. And conventional Bayesian priors are *conservative*, relative to conventional non-Bayesian approaches.

Making choices tends to make novices nervous. There's an illusion sometimes that default procedures are more objective than procedures that require user choice, such as choosing priors. If that's true, then all "objective" means is that everyone does the same thing. It carries no guarantees of realism or accuracy.

4.4.2. Fitting the model. The code needed to fit this model via quadratic approximation is a straightforward modification of the kind of code you've already seen. All we have to do is incorporate our new model for the mean into the model specification inside `map` and be sure to add our new parameters to the `start` list. Let's repeat the model definition, now with the corresponding R code on the right-hand side:

$$
\begin{aligned}
h_i &\sim \text{Normal}(\mu_i, \sigma) && \texttt{height ~ dnorm(mu,sigma)} \\
\mu_i &= \alpha + \beta x_i && \texttt{mu <- a + b*weight} \\
\alpha &\sim \text{Normal}(178, 100) && \texttt{a ~ dnorm(156,100)} \\
\beta &\sim \text{Normal}(0, 10) && \texttt{b ~ dnorm(0,10)} \\
\sigma &\sim \text{Uniform}(0, 50) && \texttt{sigma ~ dunif(0,50)}
\end{aligned}
$$

Notice that the linear model, in the R code on the right-hand side, uses the R assignment operator, `<-`, even though the mathematical definition uses the symbol =. This is a code convention shared by several Bayesian model fitting engines, so it's worth getting used to

the switch. You just have to remember to use <- instead of = when defining a linear model. That's it.

And the above allows us to build the MAP model fit:

R code 4.38

```
# load data again, since it's a long way back
library(rethinking)
data(Howell1)
d <- Howell1
d2 <- d[ d$age >= 18 , ]

# fit model
m4.3 <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight ,
        a ~ dnorm( 156 , 100 ) ,
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d2 )
```

The parameter `mu` is no longer really a parameter here, because it has been replaced by the linear model, `a+b*weight`, where `a` is α and `b` is β and `weight` is of course our x in this instance. So there is a prior for the parameter `a` now, but not one for `mu`, since `mu` is defined by the linear model instead.

In the `start` list, `mu` is replaced by `a`, which starts at the overall mean, just like `mu` used to. We also have to add the new parameter `b` to this list, and as is usually a conservative first guess, we start the slope out at zero (0), which is equivalent to no relationship between the outcome and predictor.

Note that starting `b` at zero is not the same as having β's prior with mean zero. The values in the `start` list don't alter the posterior probabilities, while priors definitely do. But you do have to think a little about your choice of `start` values, because if you start the MAP search far outside the high density region of the posterior, R may give up looking before it finds the MAP values.

Rethinking: Everything that depends upon parameters has a posterior distribution. In the model introduced above, the parameter μ is no longer a parameter, since it has become a function of the parameters α and β. But since the parameters α and β have a joint posterior, so too does μ. Later in the chapter, you'll work directly the posterior distribution of μ, even though it's not a parameter anymore. Since parameters are uncertain, everything that depends upon them is also uncertain. This includes statistics like μ, as well as model-based predictions, measures of fit, and everything else that uses parameters. By working with samples from the posterior, all you have to do to account for posterior uncertainty in any quantity is to compute that quantity for each sample from the posterior. The resulting quantities, one for each posterior sample, will approximate the quantity's posterior distribution.

Overthinking: Embedding linear models. It may help to understand what the linear model is doing, by seeing another way to fit the same model, but without a separate line for the linear model. You can just merge the linear model into the likelihood definition, like this:

R code 4.39

```
m4.3 <- map(
    alist(
        height ~ dnorm( a + b*weight , sigma ) ,
        a ~ dnorm( 178 , 100 ) ,
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d2 )
```

This is exactly the same model, but with the linear definition for μ embedded in the first line of code. The form with the explicit linear model, `mu <- a + b*weight`, is easier to read. But the form just above better reflects the actual computation. Some of the helper functions you'll use later depend upon the explicit linear model however, so I recommend using it. It'll make life a little easier.

4.4.3. Interpreting the model fit.

One trouble with statistical models is that they are hard to understand. Once you've fit the model, it can only report posterior probabilities. These are the right answer to the question that is this combination of model and data. But it's your responsibility to process the answer and make sense of it.

There are two broad categories of processing: (1) reading tables and (2) plotting. For some simple questions, it's possible to learn a lot just from tables of MAP values, their standard deviations, and intervals. But most models are very hard to understand from tables of estimates alone. A major difficulty with tables alone is their apparent simplicity compared to the complexity of the model and data that generated them. Once you have more than a couple of parameters in a model, it is very hard to figure out from numbers alone how all of them act to influence prediction. Once you begin adding interaction terms (Chapter 7) or polynomials (later in this chapter), it may not even be possible to guess the direction of influence a predictor variable has on an outcome.

So throughout this book, I emphasize plotting posterior distributions and posterior predictions, instead of attempting to understand a table. Once you become experienced with a particular type of model and kind of data, you'll be able to confidently read tables of estimates. But that will be because you will have won valuable contextual knowledge about the problem at hand, knowledge that will transfer, but incompletely, to other problems.

To win such knowledge, most of us must do a lot of plotting. Plotting the implications of your estimates will allow you to inquire about several things that are sometimes hard to read from tables:

(1) Whether or not the model fitting procedure worked correctly
(2) The *absolute* magnitude, rather than merely *relative* magnitude, of a relationship between outcome and predictor
(3) The uncertainty surrounding an average relationship
(4) The uncertainty surrounding the implied predictions of the model, as these are distinct from mere parameter uncertainty

In addition, once you get the hang of processing estimates into plots, you can ask any question you can think of, for any model type. And readers of your results will appreciate a figure much more than they will a table of estimates.

So in the remainder of this section, I first spend a little time talking about tables of estimates. I use this opportunity to briefly introduce a data transformation called **CENTERING** that aids in interpreting estimates. Then I move on to show how to plot estimates that always incorporate information from the full posterior distribution, including correlations among parameters.

Rethinking: What do parameters mean? A basic issue with interpreting model-based estimates is in knowing the meaning of parameters. There is no consensus about what a parameter means, however, because different people take different philosophical stances towards models, probability, and prediction. The perspective in this book is a common Bayesian perspective: *Posterior probabilities of parameter values describe the relative compatibility of different states of the world with the data, according to the model.* These are small world (Chapter 2) numbers. So reasonable people may disagree about the large world meaning, and the details of those disagreements depend strongly upon context. Such disagreements are productive, because they lead to model criticism and revision, something that golems cannot do for themselves.

4.4.3.1. *Tables of estimates.* Before looking closely at the new table of estimates, it's important to realize that models cannot in general be understood by tables of estimates. In this simple model, a lot can be learned from the summary output. But this is not a general property of models, Bayesian or not, because of the covariation among parameters.

With the new linear regression fit to the Kalahari data, we inspect the estimates:

R code 4.40

```
precis( m4.3 )
```

```
       Mean StdDev   5.5%  94.5%
a    113.90   1.91 110.85 116.94
b      0.90   0.04   0.84   0.97
sigma  5.07   0.19   4.77   5.38
```

The first row gives the quadratic approximation for α, the second the approximation for β, and the third approximation for σ. Let's try to make some sense of them in this very simple model.

Best to begin with `b` (β), because it's the new parameter. Since β is a slope, the value 0.90 can be read as *a person 1 kg heavier is expected to be 0.90 cm taller.* 89% of the posterior probability lies between 0.84 and 0.97. That suggests that β values close to zero or greatly above one are highly incompatible with these data and this model. If you were thinking that perhaps there was no relationship at all between height and weight, then this estimate indicates strong evidence of a positive relationship instead. But maybe you just wanted as precise a measurement as possible of the relationship between height and weight. This estimate embodies that measurement, conditional on the model. For a different model, the measure of the relationship might be different.

The estimate of α, `a` in the `precis` table, indicates that a person of weight 0 should be 114cm tall. This is nonsense, since real people always have positive weight, yet it is also true. Parameters like α are "intercepts" that tell us the value of μ when all of the predictor variables have value zero. As a consequence, the value of the intercept is frequently uninterpretable

if you recenter the predictors life becomes easier

without also studying any β parameters. This is why we need very weak priors for intercepts, in many cases.

Finally, the estimate for σ, `sigma`, informs us of the width of the distribution of heights around the mean. A quick way to interpret it is to recall that about 95% of the probability in a Gaussian distribution lies between two standard deviations. So in this case, the estimate tells us that 95% of plausible heights lie within 10cm (2σ) of the mean height. But there is also uncertainty about this, as indicated by the 89% percentile interval.

As I mentioned at the start of this section, the numbers in the default `precis` output aren't sufficient to describe the quadratic posterior completely. For that, we also require the variance-covariance matrix. We're interested in correlations among parameters—we already have their variance in the table above—so let's go straight to the correlation matrix:

R code 4.41

```
precis( m4.3 , corr=TRUE )
```

```
       Mean StdDev   5.5%  94.5%     a     b sigma
a    113.90   1.91 110.85 116.94  1.00 -0.99     0
b      0.90   0.04   0.84   0.97 -0.99  1.00     0
sigma  5.07   0.19   4.77   5.38  0.00  0.00     1
```

The new columns on the far right show the correlations among the parameters. This is the same information you'd get by using `cov2cor(vcov(m4.3))`. Notice that α and β are almost perfectly negatively correlated. Right now, this is harmless. It just means that these two parameters carry the same information—as you change the slope of the line, the best intercept changes to match it. But in more complex models, strong correlations like this can make it difficult to fit the model to the data. So we'll want to use some golem engineering tricks to avoid it, when possible.

the issue is fitting not something else

The first trick is CENTERING. Centering is the procedure of subtracting the mean of a variable from each value. To create a centered version of the weight variable:

R code 4.42

```
d2$weight.c <- d2$weight - mean(d2$weight)
```

You can confirm that the average value of `weight.c` is zero: `mean(d2$weight.c)`. Now let's refit the model and see what this gains us:

R code 4.43

```
m4.4 <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight.c ,
        a ~ dnorm( 178 , 100 ) ,
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d2 )
```

The above code just replaces `weight` with `weight.c`, the new variable. Now for the new estimates:

R code 4.44

```
precis( m4.4 , corr=TRUE )
```

```
       Mean StdDev   5.5%  94.5% a b sigma
a    154.60   0.27 154.17 155.03 1 0     0
b      0.91   0.04   0.84   0.97 0 1     0
sigma  5.07   0.19   4.77   5.38 0 0     1
```

The estimates for β and σ are unchanged (within rounding error), but the estimate for α (a) is now the same as the average height value in the raw data. Try it yourself: `mean(d2$height)`. And the correlations among parameters are now all zero. What has happened here?

The estimate for the intercept, α, still means the same thing it did before: the expected value of the outcome variable, when the predictor variable is equal to zero. But now the mean value of the predictor is also zero. So the intercept also means: the expected value of the outcome, when the predictor is at its average value. This makes interpreting the intercept a lot easier.

4.4.3.2. *Plotting posterior inference against the data.* In truth, tables of estimates are usually insufficient for understanding the information contained in the posterior distribution. It's almost always much more useful to plot the posterior inference against the data. Not only does plotting help in interpreting the posterior, but it also provides an informal check on model assumptions. When the model's predictions don't come close to key observations or patterns in the plotted data, then you might suspect the model either did not fit correctly or is rather badly specified.

But even if you only treat plots as a way to help in interpreting the posterior, they are invaluable. For simple models like this one, it is easy to just read the table of numbers and understand what the model says. But for even slightly more complex models, especially those that include interaction effects (Chapter 7), interpreting posterior distributions is hard. Combine with this the problem of incorporating the information in `vcov` into your interpretations, and the plots are irreplaceable.

We're going to start with a simple version of that task, superimposing just the MAP values over the height and weight data. Then we'll slowly add more and more information to the prediction plots, until we've used the entire posterior distribution.

To superimpose the MAP values for mean height over the actual data:

R code 4.45

```
plot( height ~ weight , data=d2 )
abline( a=coef(m4.3)["a"] , b=coef(m4.3)["b"] )
```

You can see the resulting plot in FIGURE 4.4. Each point in this plot is a single individual. The black line is defined by the MAP slope β and MAP intercept α. Notice that in the code above, I pulled the numbers straight from the fit model. The function `coef` returns a vector of MAP values, and the names of the parameters extract the intercept and slope.

4.4.3.3. *Adding uncertainty around the mean.* The MAP line is just the posterior mean, the most plausible line in the infinite universe of lines the posterior distribution has considered. Plots of the MAP line, like FIGURE 4.4, are useful for getting an impression of the magnitude of the estimated influence of a variable, like `weight`, on an outcome, like `height`.

But they do a poor job of communicating uncertainty. Remember, the posterior distribution considers every possible regression line connecting height to weight. It assigns a relative plausibility to each. This means that each combination of α and β has a posterior probability. It could be that there are many lines with nearly the same posterior probability

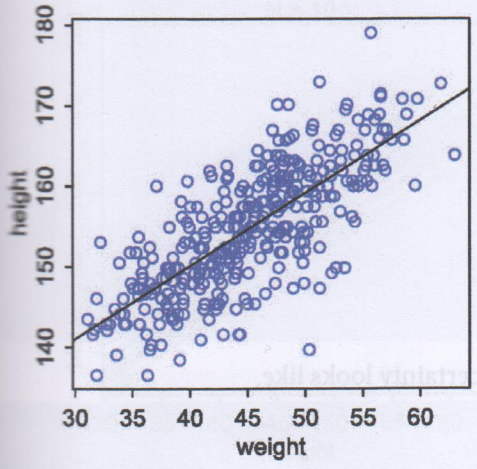


FIGURE 4.4. Height in centimeters (vertical) plotted against weight in kilograms (horizontal), with the *maximum a posteriori* line for the mean height at each weight plotted in black.

as the MAP line. Or it could be instead that the posterior distribution is rather narrow near the MAP line.

So how can we get that uncertainty onto the plot? Together, a combination of α and β define a line. And so we could sample a bunch of lines from the posterior distribution. Then we could display those lines on the plot, to visualize the uncertainty in the regression relationship.

To better appreciate how the posterior distribution contains lines, extract some samples from the model:

```
post <- extract.samples( m4.3 )
```

R code 4.46

Then inspect the first 5 rows of the samples:

```
post[1:5,]
```

R code 4.47

```
      a         b     sigma
1 114.7880 0.8822921 5.121102
2 112.7115 0.9230855 4.907987
3 114.4557 0.9018482 5.276036
4 114.7696 0.8831561 5.021958
5 112.6333 0.9383632 4.898554
```

Each row is a correlated random sample from the joint posterior of all three parameters, using the covariances provided by `vcov(m4.3)`. The paired values of `a` and `b` on each row define a line. The average of very many of these lines is the MAP line. But the scatter around that average is meaningful, because it alters our confidence in the relationship between the predictor and the outcome.

So now let's display a bunch of these lines, so you can see the scatter. This lesson will be easier to appreciate, if we use only some of the data to begin. Then you can see how adding in more data changes the scatter of the lines. So we'll begin with just the first 10 cases in `d2`. The following code extracts the first 10 cases and re-estimates the model:

R code 4.48

```
N <- 10
dN <- d2[ 1:N , ]
mN <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b*weight ,
        a ~ dnorm( 178 , 100 ) ,
        b ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) , data=dN )
```

Now let's plot 20 of these lines, to see what the uncertainty looks like.

R code 4.49

```
# extract 20 samples from the posterior
post <- extract.samples( mN , n=20 )

# display raw data and sample size
plot( dN$weight , dN$height ,
    xlim=range(d2$weight) , ylim=range(d2$height) ,
    col=rangi2 , xlab="weight" , ylab="height" )
mtext(concat("N = ",N))

# plot the lines, with transparency
for ( i in 1:20 )
    abline( a=post$a[i] , b=post$b[i] , col=col.alpha("black",0.3) )
```

The last line loops over all 20 lines, using `abline` to display each.

The result is shown in the upper-left plot in FIGURE 4.5. By plotting multiple regression lines, sampled from the posterior, it is easy to see both the highly confident aspects of the relationship and the less confident aspects. The cloud of regression lines displays greater uncertainty at extreme values for weight. This is very common.

The other plots in FIGURE 4.5 show the same relationships, but for increasing amounts of data. Just re-use the code from before, but change `N <- 10` to some other value. Notice that the cloud of regression lines grows more compact as the sample size increases. This is a result of the model growing more confident about the location of the mean.

4.4.3.4. *Plotting regression intervals and contours.* The cloud of regression lines in FIGURE 4.5 is an appealing display, because it communicates uncertainty about the relationship in a way that many people find intuitive. But it's much more common to see the uncertainty displayed by plotting an interval or contour around the MAP regression line. In this section, I'll walk you through how to compute any arbitrary interval you like, using the underlying cloud of regression lines embodied in the posterior distribution. Then we'll plot a shaded region around the MAP line, to display the interval.

Here's how to plot an interval around the regression line. This interval incorporates uncertainty in both the slope β and intercept α at the same time. To understand how it works, focus for the moment on a single `weight` value, say 50 kilograms. You can quickly make a list of 10,000 values of μ for an individual who weighs 50 kilograms, by using your samples from the posterior:

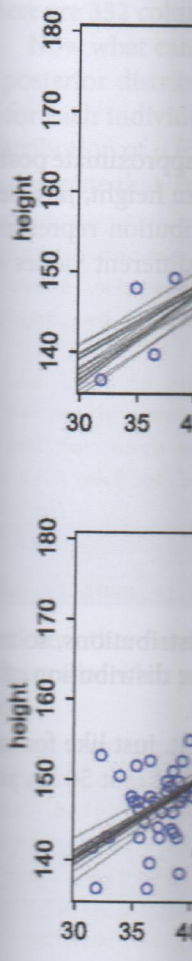


FIGURE 4.5
for the heig
plot, 20 lin
tainty in th

mu_at_50 <- post

The code to the rig

The value of x_i in th
vector of predicted
went into comput
and correlation b
the density for this

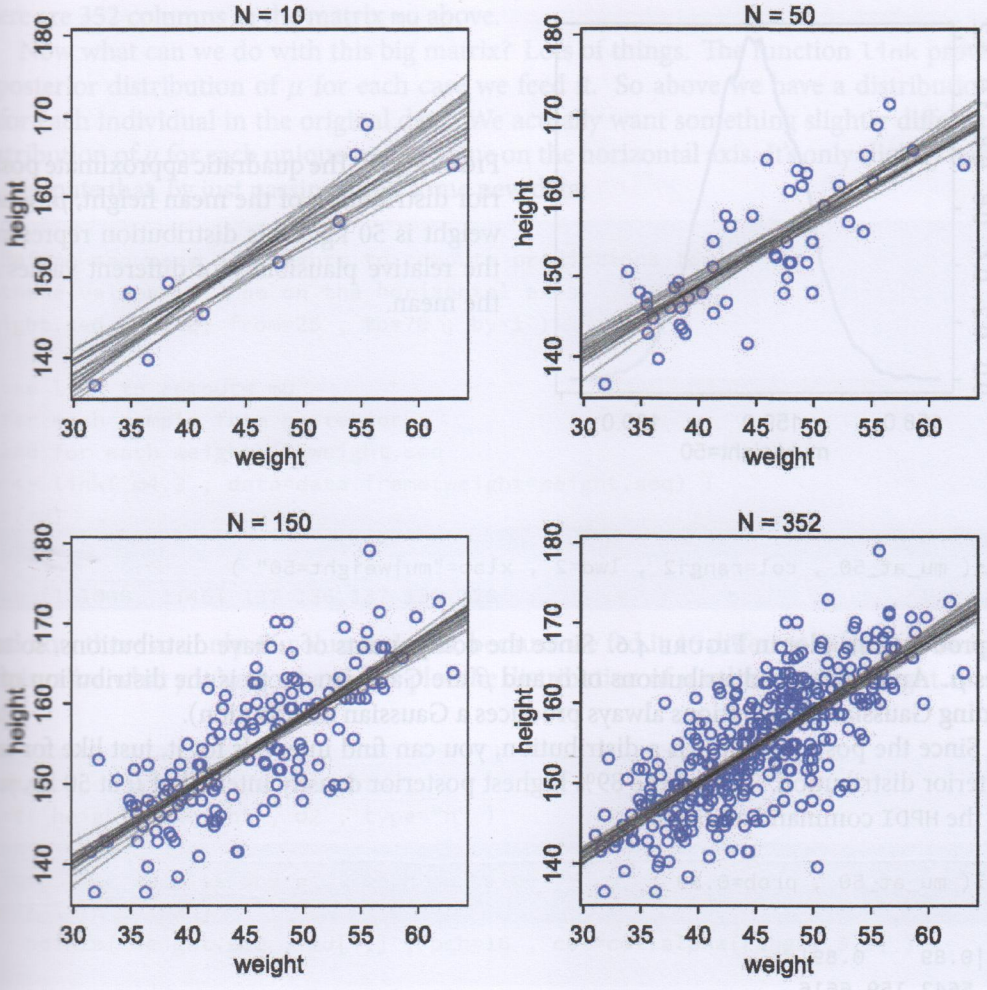


FIGURE 4.5. Samples from the quadratic approximate posterior distribution for the height/weight model, `m4.3`, with increasing amounts of data. In each plot, 20 lines sampled from the posterior distribution, showing the uncertainty in the regression relationship.

```
mu_at_50 <- post$a + post$b * 50
```

R code 4.50

The code to the right of the `<-` above takes its form from the equation for μ_i:

$$\mu_i = \alpha + \beta x_i$$

The value of x_i in this case is 50. Go ahead and take a look inside the result, `mu_at_50`. It's a vector of predicted means, one for each random sample from the posterior. Since joint `a` and `b` went into computing each, the variation across those means incorporates the uncertainty in and correlation between both parameters. It might be helpful at this point to actually plot the density for this vector of means:

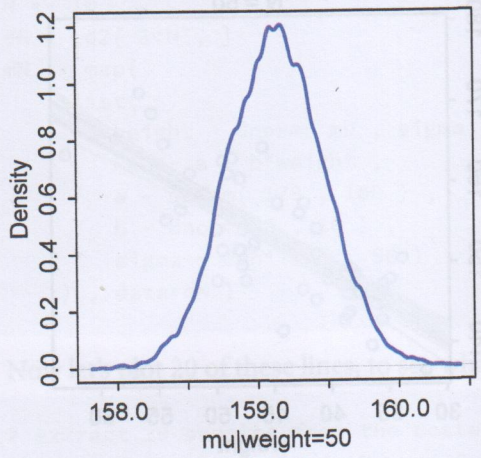


FIGURE 4.6. The quadratic approximate posterior distribution of the mean height, μ, when weight is 50 kg. This distribution represents the relative plausibility of different values of the mean.

R code 4.51

```
dens( mu_at_50 , col=rangi2 , lwd=2 , xlab="mu|weight=50" )
```

I reproduce this plot in FIGURE 4.6. Since the components of μ have distributions, so too does μ. And since the distributions of α and β are Gaussian, so to is the distribution of μ (adding Gaussian distributions always produces a Gaussian distribution).

Since the posterior for μ is a distribution, you can find intervals for it, just like for any posterior distribution. To find the 89% highest posterior density interval of μ at 50 kg, just use the `HPDI` command as usual:

R code 4.52

```
HPDI( mu_at_50 , prob=0.89 )
```

```
   |0.89    0.89|
158.5642 159.6616
```

What these numbers mean is that the central 89% of the ways for the model to produce the data place the average height between about 159 cm and 160 cm (conditional on the model and data), assuming the weight is 50 kg.

That's good so far, but we need to repeat the above calculation for every `weight` value on the horizontal axis, not just when it is 50 kg. We want to draw 89% HPDIs around the MAP slope in Figure 4.4.

This is made simple by strategic use of the `link` function, a part of the `rethinking` package. What `link` will do is take your `map` model fit, sample from the posterior distribution, and then compute μ for each case in the data and sample from the posterior distribution. Here's what it looks like for the data you used to fit the model:

R code 4.53

```
mu <- link( m4.3 )
str(mu)
```

```
 num [1:1000, 1:352] 157 157 157 157 157 ...
```

You end up with a big matrix of values of μ. Each row is a sample from the posterior distribution. The default is 1000 samples, but you can use as many or as few as you like. Each column

is a case (row) in the data. There are 352 rows in d2, corresponding to 352 individuals. So there are 352 columns in the matrix mu above.

Now what can we do with this big matrix? Lots of things. The function link provides a posterior distribution of μ for each case we feed it. So above we have a distribution of μ for each individual in the original data. We actually want something slightly different: a distribution of μ for each unique weight value on the horizontal axis. It's only slightly harder to compute that, by just passing link some new data:

R code 4.54

```
# define sequence of weights to compute predictions for
# these values will be on the horizontal axis
weight.seq <- seq( from=25 , to=70 , by=1 )

# use link to compute mu
# for each sample from posterior
# and for each weight in weight.seq
mu <- link( m4.3 , data=data.frame(weight=weight.seq) )
str(mu)
```

```
 num [1:1000, 1:46] 137 136 137 137 136 ...
```

And now there are only 46 columns in mu, because we fed it 46 different values for weight. To visualize what you've got here, let's plot the distribution of μ values at each height, on the plot.

R code 4.55

```
# use type="n" to hide raw data
plot( height ~ weight , d2 , type="n" )

# loop over samples and plot each mu value
for ( i in 1:100 )
    points( weight.seq , mu[i,] , pch=16 , col=col.alpha(rangi2,0.1) )
```

The result is shown on the left-hand side of FIGURE 4.7. At each weight value in weight.seq, a pile of computed μ values are shown. Each of these piles is a Gaussian distribution, like that in FIGURE 4.6. You can see now that the amount of uncertainty in μ depends upon the value of weight. And this is the same fact you saw in the right-hand plot in FIGURE 4.5.

The final step is to summarize the distribution for each weight value. We'll use apply, which applies a function of your choice to a matrix.

R code 4.56

```
# summarize the distribution of mu
mu.mean <- apply( mu , 2 , mean )
mu.HPDI <- apply( mu , 2 , HPDI , prob=0.89 )
```

Read apply(mu,2,mean) as *compute the mean of each column (dimension "2") of the matrix* mu. Now mu.mean contains the average μ at each weight value, and mu.HPDI contains 89% lower and upper bounds for each weight value. Be sure to take a look inside mu.mean and mu.HPDI, to demystify them. They are just different kinds of summaries of the distributions in mu, with each column being for a different weight value.

You can plot these summaries on top of the data with a few lines of R code:

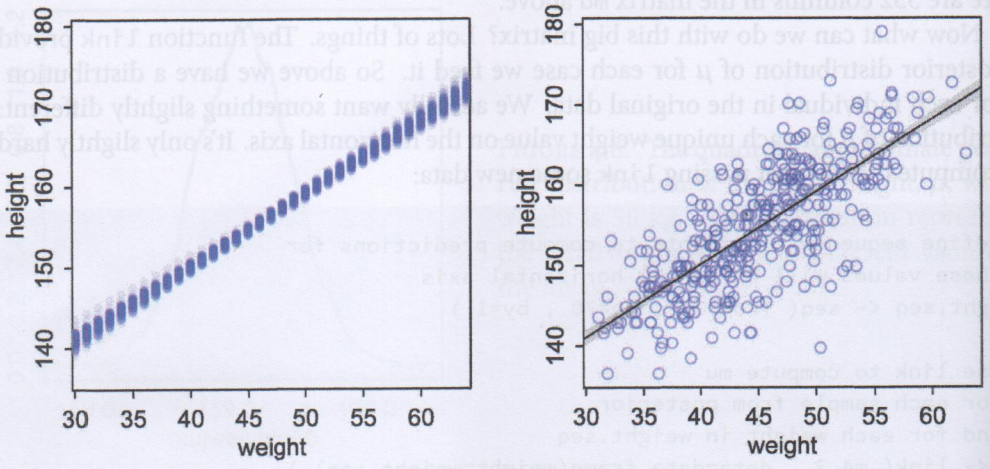


FIGURE 4.7. Left: The first 100 values in the distribution of μ at each weight value. Right: The !Kung height data again, now with 89% HPDI of the mean indicated by the shaded region. Compare this region to the distributions of blue points on the left.

R code 4.57

```
# plot raw data
# fading out points to make line and interval more visible
plot( height ~ weight , data=d2 , col=col.alpha(rangi2,0.5) )

# plot the MAP line, aka the mean mu for each weight
lines( weight.seq , mu.mean )

# plot a shaded region for 89% HPDI
shade( mu.HPDI , weight.seq )
```

You can see the results in the right-hand plot in FIGURE 4.7.

Using this approach, you can derive and plot posterior prediction means and intervals for quite complicated models, for any data you choose. It's true that it is possible to use analytical formulas to compute intervals like this. I have tried teaching such an analytical approach before, and it has always been disaster. Part of the reason is probably my own failure as a teacher, but another part is that most social and natural scientists have never had much training in probability theory and tend to get very nervous around $\int$'s. I'm sure with enough effort, every one of them could learn to do the mathematics. But all of them can quickly learn to generate and summarize samples derived from the posterior distribution. So while the mathematics would be a more elegant approach, and there is some additional insight that comes from knowing the mathematics, the pseudo-empirical approach presented here is very flexible and allows a much broader audience of scientists to pull insight from their statistical modeling. And again, when you start estimating models with MCMC (Chapter 8), this is really the only approach available. So it's worth learning now.

To summarize, here's the recipe for generating predictions and intervals from the posterior of a fit model.

(1) Use `link` to generate distributions of posterior values for μ. The default behavior of `link` is to use the original data, so you have to pass it a list of new horizontal axis values you want to plot posterior predictions across.
(2) Use summary functions like `mean` or `HPDI` or `PI` to find averages and lower and upper bounds of μ for each value of the predictor variable.
(3) Finally, use plotting functions like `lines` and `shade` to draw the lines and intervals. Or you might plot the distributions of the predictions, or do further numerical calculations with them. It's really up to you.

This recipe works for every model we fit in the book. As long as you know how parameters relate to the data, you can use samples from the posterior to describe any aspect of the model's behavior.

Rethinking: Overconfident confidence intervals. The confidence interval for the regression line in FIGURE 4.7 clings tightly to the MAP line. Thus there is very little uncertainty about the average height as a function of average weight. But you have to keep in mind that these inferences are always conditional on the model. Even a very bad model can have very tight confidence intervals. It may help if you think of the regression line in FIGURE 4.7 as saying: *Conditional on the assumption that height and weight are related by a straight line, then this is the most plausible line, and these are its plausible bounds.*

Overthinking: How `link` works. The function `link` is not really very sophisticated. All it is doing is using the formula you provided when you fit the model to compute the value of the linear model. It does this for each sample from the posterior distribution, for each case in the data. You could accomplish the same thing for any model, fit by any means, by performing these steps yourself. This is how it'd look for `m4.3`.

R code 4.58

```
post <- extract.samples(m4.3)
mu.link <- function(weight) post$a + post$b*weight
weight.seq <- seq( from=25 , to=70 , by=1 )
mu <- sapply( weight.seq , mu.link )
mu.mean <- apply( mu , 2 , mean )
mu.HPDI <- apply( mu , 2 , HPDI , prob=0.89 )
```

And the values in `mu.mean` and `mu.HPDI` should be very similar (allowing for simulation variance) to what you got the automated way, using `link`.

Knowing this manual method is useful both for (1) understanding and (2) sheer power. Whatever the model you find yourself with, this approach can be used to generate posterior predictions for any component of it. Automated tools like `link` save effort, but they are never as flexible as the code you can write yourself.

4.4.3.5. *Prediction intervals.* Now let's walk through generating an 89% prediction interval for actual heights, not just the average height, μ. This means we'll incorporate the standard deviation σ and its uncertainty as well. Remember, the statistical model here is (omitting priors for brevity):

$$h_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta x_i$$

What you've done so far is just use samples from the posterior to visualize the uncertainty in μ_i, the linear model of the mean. But actual predictions of heights depend also upon the stochastic definition in the first line. The Gaussian distribution on the first line tells us that the model expects observed heights to be distributed around μ, not right on top of it. And the spread around μ is governed by σ. All of this suggests we need to incorporate σ in the predictions somehow.

Here's how you do it. Imagine simulating heights. For any unique weight value, you sample from a Gaussian distribution with the correct mean μ for that weight, using the correct value of σ sampled from the same posterior distribution. If you do this for every sample from the posterior, for every weight value of interest, you end up with a collection of simulated heights that embody the uncertainty in the posterior *as well as* the uncertainty in the Gaussian likelihood.

R code 4.59

```
sim.height <- sim( m4.3 , data=list(weight=weight.seq) )
str(sim.height)
```

```
 num [1:1000, 1:46] 139 144 141 140 130 ...
```

This matrix is much like the earlier one, `mu`, but it contains simulated heights, not distributions of plausible average height, μ.

We can summarize these simulated heights in the same way we summarized the distributions of μ, by using `apply`:

R code 4.60

```
height.PI <- apply( sim.height , 2 , PI , prob=0.89 )
```

Now `height.PI` contains the 89% posterior prediction interval of observable (according to the model) heights, across the values of weight in `weight.seq`.

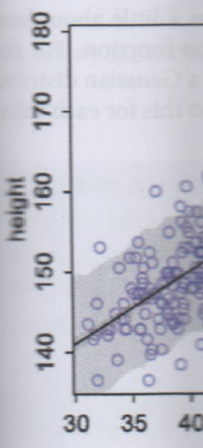
Let's plot everything we've built up: (1) the MAP line, (2) the shaded region of 89% plausible μ, and (3) the boundaries of the simulated heights the model expects.

R code 4.61

```
# plot raw data
plot( height ~ weight , d2 , col=col.alpha(rangi2,0.5) )

# draw MAP line
lines( weight.seq , mu.mean )

# draw HPDI region for line
shade( mu.HPDI , weight.seq )

# draw PI region for simulated heights
shade( height.PI , weight.seq )
```

The code above uses some objects computed in previous sections, so go back and execute that code, if you need to.

In FIGURE 4.8, I plot the result. The wide shaded region in the figure represents the area within which the model expects to find 89% of actual heights in the population, at each weight. There is nothing special about the value 89% here. You could plot the boundary for other percents, such as 67% and 97% (also both primes), and add those to the plot. Doing so would help you see more of the shape of the predicted distribution of heights. I leave that as



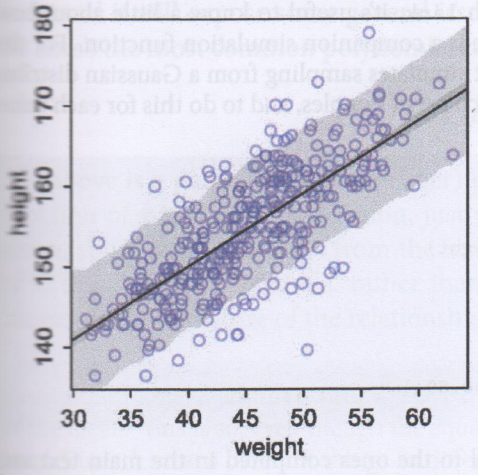


FIGURE 4.8. 89% prediction interval for height, as a function of weight. The solid line is the MAP estimate of the mean height at each weight. The two shaded regions show different 89% plausible regions. The narrow shaded interval around the line is the distribution of μ. The wider shaded region represents the region within which the model expects to find 89% of actual heights in the population, at each weight.

an exercise for the reader. Just go back to the code above and add `prob=0.67`, for example, to the call to `PI`. That will give you 67% intervals, instead of 89% ones.

Notice that the outline for the wide shaded interval is a little jagged. This is the simulation variance in the tails of the sampled Gaussian values. If it really bothers you, increase the number of samples you take from the posterior distribution. The optional `n` parameter for `sim.height` controls how many samples are used. Try for example:

R code 4.62

```
sim.height <- sim( m4.3 , data=list(weight=weight.seq) , n=1e4 )
height.PI <- apply( sim.height , 2 , PI , prob=0.89 )
```

Run the plotting code again, and you'll see the shaded boundary smooth out some. With extreme percentiles, it can be very hard to get out all of the jaggedness. Luckily, it hardly matters, except for aesthetics. Moreover, it serves to remind us that all statistical inference is approximate. The fact that we can compute an expected value to the 10th decimal place does not imply that our inferences are precise to the 10th decimal place.

Rethinking: Two kinds of uncertainty. In the procedure above, we encountered both uncertainty in parameter values and uncertainty in a sampling process. These are distinct concepts, even though they are processed much the same way and end up blended together in the posterior predictive simulation. The posterior distribution is a ranking of the relative plausibilities of every possible combination of parameter values. The distribution of simulated outcomes, like height, is instead a distribution that includes sampling variation from some process that generates Gaussian random variables. This sampling variation is still a model assumption. It's no more or less objective than the posterior distribution. Both kinds of uncertainty matter, at least sometimes. But it's important to keep them straight, because they depend upon different model assumptions. Furthermore, it's possible to view the Gaussian likelihood as a purely epistemological assumption (a device for estimating the mean and variance of a variable), rather than an ontological assumption about what future data will look like. In that case, it may not make complete sense to simulate outcomes.

Overthinking: Rolling your own `sim`. Just like with `link`, it's useful to know a little about how `sim` operates. For every distribution like `dnorm`, there is a companion simulation function. For the Gaussian distribution, the companion is `rnorm`, and it simulates sampling from a Gaussian distribution. What we want R to do is simulate a height for each set of samples, and to do this for each value of weight. The following will do it:

R code 4.63

```
post <- extract.samples(m4.3)
weight.seq <- 25:70
sim.height <- sapply( weight.seq , function(weight)
    rnorm(
        n=nrow(post) ,
        mean=post$a + post$b*weight ,
        sd=post$sigma ) )
height.PI <- apply( sim.height , 2 , PI , prob=0.89 )
```

The values in `height.PI` will be practically identical to the ones computed in the main text and displayed in FIGURE 4.8.

4.5. Polynomial regression

In the next chapter, you'll see how to use linear models to build regressions with more than one predictor variable. But before then, it helps to see how to model the outcome as a curved function of a single predictor. The models so far all assume that a straight line describes the relationship. But there's nothing special about straight lines, aside from their simplicity.

Let's work through an example, using the full !Kung data:

R code 4.64

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

```
'data.frame': 544 obs. of  4 variables:
 $ height: num  152 140 137 157 145 ...
 $ weight: num  47.8 36.5 31.9 53 41.3 ...
 $ age   : num  63 63 65 41 51 35 32 27 19 54 ...
 $ male  : int  1 0 0 1 0 1 0 1 0 1 ...
```

Go ahead and plot `height` against `weight`. The relationship is visibly curved, now that we've included the non-adult individuals.

There are many ways to model a curved relationship between two variables. Here, I'll show you a very common one, **POLYNOMIAL REGRESSION**. In this context, "polynomial" means equations for μ_i that add additional terms with squares, cubes, and even higher powers of the predictor variable. There's still only one predictor variable in the model, so this is still a bivariate regression. But the definition of μ_i has more parameters now.

While this section teaches polynomial regression, in general it's a bad thing to do. Why? Because polynomials are very hard to interpret. Better would be to have a more mechanistic model of the data, one that builds the non-linear relationship up from a principled beginning. In the practice problems for this chapter, you'll see how to do this with the height data.

But it's still worth working through a polynomial example, both because it is very common and it will expose some general issues with modeling that we'll take up in Chapter 6. So here's the most common polynomial regression, a parabolic model of the mean:

$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2$$

The above is a parabolic (second order) polynomial. The $\alpha + \beta_1 x_i$ part is the same linear function of x in a linear regression, just with a little "1" subscript added to the parameter name, so we can tell it apart from the new parameter. The additional term uses the square of x_i to construct a parabola, rather than a perfectly straight line. The new parameter β_2 measures the curvature of the relationship.

Rethinking: Linear, additive, funky. The parabolic model of μ_i above is still called a "linear model" of the mean. This is so, even though the equation is clearly not of a straight line. Unfortunately, the word "linear" means different things in different contexts, and different people use it differently in the same context. What "linear" usually means in this context is that μ_i is a *linear function* of any single parameter. Such models have the advantage of being easier to fit to data. They are also often easier to interpret, because they assume that parameters act independently on the mean. They have the disadvantage of being strongly conventional. They are often used thoughtlessly. When you have real knowledge of your study system, it is often easy to do better than a linear model. These models are geocentric engines, devices for describing partial correlations among variables. We should feel embarrassed to use them, just so we don't become satisfied with the phenomenological explanations they provide.

Fitting these models to data is easy. Interpreting them can be hard. We'll begin with the easy part, fitting a parabolic model of height on weight. The first thing to do is to **STANDARDIZE** the predictor variable. This means to first center the variable and then divide it by its standard deviation. Why do this? You already have some sense of the value of centering. Going further to standardize leaves the mean at zero but also rescales the range of the data. This is helpful for two reasons:

(1) Interpretation might be easier. For a standardized variable, a change of one unit is equivalent to a change of one standard deviation. In many contexts, this is more interesting and more revealing than a one unit change on the natural scale. And once you start making regressions with more than one kind of predictor variable, standardizing all of them makes it easier to compare their relative influence on the outcome, using only estimates. On the other hand, you might want to interpret the data on the natural scale. So standardization can make interpretation harder, not easier. With a little practice, though, you can learn to quickly convert back and forth between natural and standard scales.

(2) More important though are the advantages for fitting the model to the data. When predictor variables have very large values in them, there are sometimes numerical glitches. Even well-known statistical software can suffer from these glitches, leading to mistaken estimates. These problems are very common for polynomial regression, because the square or cube of a large number can be truly massive. Standardizing largely resolves this issue.

To standardize `weight`, all you do is subtract the mean and then divide by the standard deviation. This will do it:

R code 4.65

```
d$weight.s <- ( d$weight - mean(d$weight) )/sd(d$weight)
```

This new variable `weight.s` has mean zero and standard deviation 1. No information has been lost in this procedure. Go ahead and plot `height` on `weight.s` to verify that. You'll see the same curved relationship as before, but now with a different range on the horizontal axis.

To fit the parabolic model, just modify the definition of μ_i. Here's the model (with very weak priors):

$$h_i \sim \text{Normal}(\mu_i, \sigma) \qquad \texttt{height ~ dnorm(mu,sigma)}$$
$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 \qquad \texttt{mu <- a + b1*weight.s + b2*weight.s\^2}$$
$$\alpha \sim \text{Normal}(178, 100) \qquad \texttt{a ~ dnorm(140,100)}$$
$$\beta_1 \sim \text{Normal}(0, 10) \qquad \texttt{b1 ~ dnorm(0,10)}$$
$$\beta_2 \sim \text{Normal}(0, 10) \qquad \texttt{b2 ~ dnorm(0,10)}$$
$$\sigma \sim \text{Uniform}(0, 50) \qquad \texttt{sigma ~ dunif(0,50)}$$

And fitting is straightforward, as well. Just modify the definition of `mu` so that it contains both the ordinary and quadratic terms. But in general it is better to pre-process any variable transformations. So I'll also build the square of `weight.s` as a separate variable:

R code 4.66

```
d$weight.s2 <- d$weight.s^2
m4.5 <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b1*weight.s + b2*weight.s2 ,
        a ~ dnorm( 178 , 100 ) ,
        b1 ~ dnorm( 0 , 10 ) ,
        b2 ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d )
```

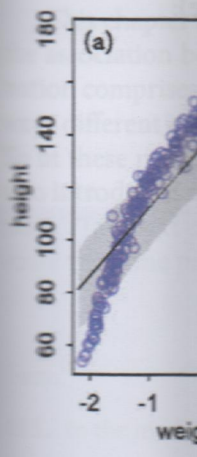
Interpreting the estimates from the summary table can be hard, though. Let's take a look:

R code 4.67

```
precis( m4.5 )
```

```
        Mean StdDev   5.5%  94.5%
a     146.66   0.37 146.07 147.26
b1     21.40   0.29  20.94  21.86
b2     -8.42   0.28  -8.87  -7.97
sigma   5.75   0.17   5.47   6.03
```

The parameter α (`a`) is still the intercept, so it tells us the expected value of `height` when `weight.s` is zero. But it is no longer equal to the mean height in the sample, since there is no guarantee it should in a polynomial regression.[70] And those β_1 and β_2 parameters are the linear and square components of the curve, respectively. But that doesn't make them transparent.



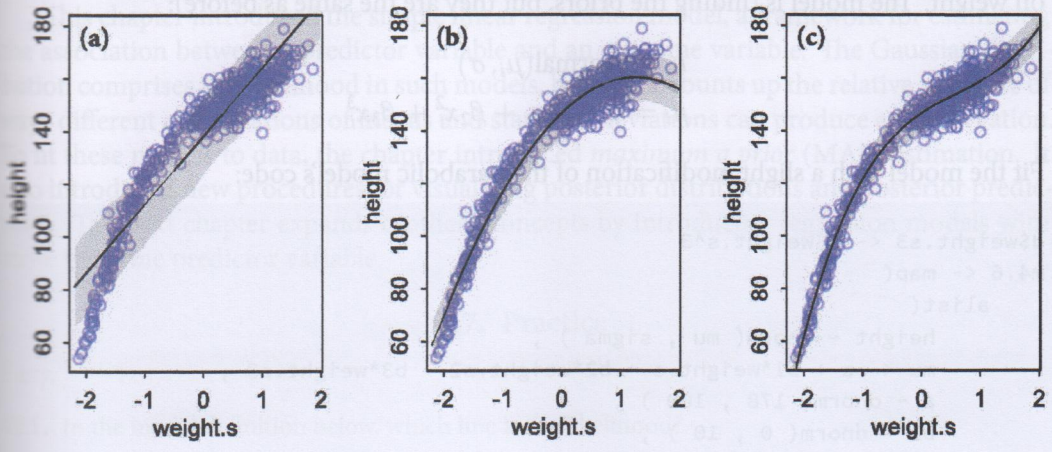


FIGURE 4.9. Polynomial regressions of height on weight (standardized), for the full !Kung data. In each plot, the raw data are shown by the circles. The solid curves show the path of μ in each model, and the shaded regions show the 89% interval of the mean (close to the solid curve) and the 89% interval of predictions (wider). (a) Linear regression. (b) A second order polynomial, a parabolic regression. (c) A third order polynomial, a cubic regression.

You have to plot these model fits to understand what they are saying. So let's do that. We'll calculate the mean relationship and the 89% intervals of the mean and the predictions, like in the previous section. Here's the working code:

R code 4.68

```
weight.seq <- seq( from=-2.2 , to=2 , length.out=30 )
pred_dat <- list( weight.s=weight.seq , weight.s2=weight.seq^2 )
mu <- link( m4.5 , data=pred_dat )
mu.mean <- apply( mu , 2 , mean )
mu.PI <- apply( mu , 2 , PI , prob=0.89 )
sim.height <- sim( m4.5 , data=pred_dat )
height.PI <- apply( sim.height , 2 , PI , prob=0.89 )
```

Plotting all of this is straightforward:

R code 4.69

```
plot( height ~ weight.s , d , col=col.alpha(rangi2,0.5) )
lines( weight.seq , mu.mean )
shade( mu.PI , weight.seq )
shade( height.PI , weight.seq )
```

The results are shown in FIGURE 4.9. Panel (a) of the figure shows the familiar linear regression from earlier in the chapter, but now with the standardized predictor and full data with both adults and non-adults. The linear model makes some spectularly poor predictions, at both very low and middle weights. Compare this to panel (b), our new parabolic regression. The curve does a much better job of finding a central path through the data.

Panel (c) in FIGURE 4.9 shows a higher-order polynomial regression, a cubic regression on weight. The model is (hiding the priors, but they are the same as before):

$$h_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3$$

Fit the model with a slight modification of the parabolic model's code:

R code 4.70

```
d$weight.s3 <- d$weight.s^3
m4.6 <- map(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b1*weight.s + b2*weight.s2 + b3*weight.s3 ,
        a ~ dnorm( 178 , 100 ) ,
        b1 ~ dnorm( 0 , 10 ) ,
        b2 ~ dnorm( 0 , 10 ) ,
        b3 ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
    ) ,
    data=d )
```

Computing the curve and intervals is similarly a small modification of the previous code. This cubic curve is even more flexible than the parabola, so it fits the data even better.

But it's not clear that any of these models make a lot of sense. They are good geocentric descriptions of the sample, yes. But later in the book (Chapter 6), you'll see how a better fit might not actually be a better model. For the moment, consider that a pregnant promise. The same issue arises in regression models with more than one type of predictor. That's the subject of the next chapter.

Overthinking: Converting back to natural scale. The plots in FIGURE 4.9 have standard units on the horizontal axis. These units are sometimes called *z-scores*. But suppose you fit the model using standardized variables, but want to plot the estimates on the original scale. All that's really needed is first to turn off the horizontal axis when you plot the raw data:

R code 4.71

```
plot( height ~ weight.s , d , col=col.alpha(rangi2,0.5) , xaxt="n" )
```

The `xaxt` at the end there turns off the horizontal axis. Then you explicitly construct the axis, using the `axis` function.

R code 4.72

```
at <- c(-2,-1,0,1,2)
labels <- at*sd(d$weight) + mean(d$weight)
axis( side=1 , at=at , labels=round(labels,1) )
```

The first line above defines the location of the labels, in standardized units. The second line then takes those units and converts them back to the original scale. The third line draws the axis. Take a look at the help `?axis` for more details.

4.6. Summary

This chapter introduced the simple linear regression model, a framework for estimating the association between a predictor variable and an outcome variable. The Gaussian distribution comprises the likelihood in such models, because it counts up the relative numbers of ways different combinations of means and standard deviations can produce an observation. To fit these models to data, the chapter introduced *maximum a prior* (MAP) estimation. It also introduced new procedures for visualizing posterior distributions and posterior predictions. The next chapter expands on these concepts by introducing regression models with more than one predictor variable.

4.7. Practice

Easy.

4E1. In the model definition below, which line is the likelihood?

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

4E2. In the model definition just above, how many parameters are in the posterior distribution?

4E3. Using the model definition above, write down the appropriate form of Bayes' theorem that includes the proper likelihood and priors.

4E4. In the model definition below, which line is the linear model?

$$y_i \sim \text{Normal}(\mu, \sigma)$$
$$\mu_i = \alpha + \beta x_i$$
$$\alpha \sim \text{Normal}(0, 10)$$
$$\beta \sim \text{Normal}(0, 1)$$
$$\sigma \sim \text{Uniform}(0, 10)$$

4E5. In the model definition just above, how many parameters are in the posterior distribution?

Medium.

4M1. For the model definition below, simulate observed heights from the prior (not the posterior).

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

4M2. Translate the model just above into a `map` formula.

4M3. Translate the `map` model formula below into a mathematical model definition.

```
flist <- alist(
    y ~ dnorm( mu , sigma ),
    mu <- a + b*x,
    a ~ dnorm( 0 , 50 ),
    b ~ dunif( 0 , 10 ),
    sigma ~ dunif( 0 , 50 )
```

4M4. A sample of students is measured for height each year for 3 years. After the third year, you want to fit a linear regression predicting height using year as a predictor. Write down the mathematical model definition for this regression, using any variable names and priors you choose. Be prepared to defend your choice of priors.

4M5. Now suppose I tell you that the average height in the first year was 120 cm and that every student got taller each year. Does this information lead you to change your choice of priors? How?

4M6. Now suppose I tell you that the variance among heights for students of the same age is never more than 64cm. How does this lead you to revise your priors?

Hard.

4H1. The weights listed below were recorded in the !Kung census, but heights were not recorded for these individuals. Provide predicted heights and 89% intervals (either HPDI or PI) for each of these individuals. That is, fill in the table below, using model-based predictions.

Individual	weight	expected height	89% interval
1	46.95		
2	43.72		
3	64.78		
4	32.59		
5	54.63		

4H2. Select out all the rows in the `Howell1` data with ages below 18 years of age. If you do it right, you should end up with a new data frame with 192 rows in it.

(a) Fit a linear regression to these data, using `map`. Present and interpret the estimates. For every 10 units of increase in weight, how much taller does the model predict a child gets?

(b) Plot the raw data, with height on the vertical axis and weight on the horizontal axis. Superimpose the MAP regression line and 89% HPDI for the mean. Also superimpose the 89% HPDI for predicted heights.

(c) What aspects of the model fit concern you? Describe the kinds of assumptions you would change, if any, to improve the model. You don't have to write any new code. Just explain what the model appears to be doing a bad job of, and what you hypothesize would be a better model.

4H3. Suppose a colleague of yours, who works on allometry, glances at the practice problems just above. Your colleague exclaims, "That's silly. Everyone knows that it's only the *logarithm* of body weight that scales with height!" Let's take your colleague's advice and see what happens.

(a) Model the relationship between height (cm) and the natural logarithm of weight (log-kg). Use the entire `Howell1` data frame, all 544 rows, adults and non-adults. Fit this model, using quadratic approximation:

$$h_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta \log(w_i)$$
$$\alpha \sim \text{Normal}(178, 100)$$
$$\beta \sim \text{Normal}(0, 100)$$
$$\sigma \sim \text{Uniform}(0, 50)$$

where h_i is the height of individual i and w_i is the weight (in kg) of individual i. The function for computing a natural log in R is just `log`. Can you interpret the resulting estimates?

(b) Begin with this plot:

```
plot( height ~ weight , data=Howell1 ,
    col=col.alpha(rangi2,0.4) )
```

R code 4.73

Then use samples from the quadratic approximate posterior of the model in (a) to superimpose on the plot: (1) the predicted mean height as a function of weight, (2) the 97% HPDI for the mean, and (3) the 97% HPDI for predicted heights.