15 Missing Data and Other Opportunities

A big advantage of Bayesian inference is that it obviates the need to be clever. For example, there's a classic probability puzzle known as *Bertrand's box paradox*.[210] The version that I prefer involves pancakes. Suppose I cook three pancakes. The first pancake is burnt on both sides (BB). The second pancake is burnt on only one side (BU). The third pancake is not burnt at all (UU). Now I serve you—at random—one of these pancakes, and the side facing up on your plate is burnt. What is the probability that the other side is also burnt?

This is a hard problem, if we rely upon intuition. Most people say "one-half," but that is quite wrong. And with no false modesty, my intuition is no better. But I have learned to solve these problems by cold hard ruthless application of conditional probability. There's no need to be clever when you can be ruthless.

So let's get ruthless. Applying conditional probability means using what we do know to refine our knowledge about what we wish to know. In other words:

$$\Pr(\text{want to know}|\text{already know})$$

In this case, we know the up side is burnt. We want to know whether or not the down side is burnt. The definition of conditional probability tells us:

$$\Pr(\text{burnt down}|\text{burnt up}) = \frac{\Pr(\text{burnt up, burnt down})}{\Pr(\text{burnt up})}$$

This is just the definition of conditional probability, labeled with our pancake problem. We want to know if the down side is burnt, and the information we have is that the up side is burnt. We *condition* on the information, so we update our state of information in light of it. The definition tells us that the probability we want is just the probability of the burnt/burnt pancake divided by the probability of seeing a burnt side up. The probability of the burnt/burnt pancake is 1/3, because a pancake was selected at random. The probability the up side is burnt must average over each way we can get dealt a burnt top side of the pancake. This is:

$$\Pr(\text{burnt up}) = \Pr(\text{BB})(1) + \Pr(\text{BU})(0.5) + \Pr(\text{UU})(0) = (1/3) + (1/3)(1/2) = 0.5$$

So all together:

$$\Pr(\text{burnt down}|\text{burnt up}) = \frac{1/3}{1/2} = \frac{2}{3}$$

If you don't quite believe this answer, you can do a quick simulation to confirm it.

R code 15.1

```
# simulate a pancake and return randomly ordered sides
sim_pancake <- function() {
    pancake <- sample(1:3,1)
    sides <- matrix(c(1,1,1,0,0,0),2,3)[,pancake]
    sample(sides)
}

# sim 10,000 pancakes
pancakes <- replicate( 1e4 , sim_pancake() )
up <- pancakes[1,]
down <- pancakes[2,]

# compute proportion 1/1 (BB) out of all 1/1 and 1/0
num_11_10 <- sum( up==1 )
num_11 <- sum( up==1 & down==1 )
num_11/num_11_10
```

```
[1] 0.6777889
```

Two-thirds.

If you want to derive some intuition now at the end, having seen the right answer, the trick is to count *sides* of the pancakes, not the pancakes themselves. Yes, there are 2 pancakes that have at least one burnt side. And only one of those has 2 burnt sides. But it is the sides, not the pancakes, that matter. Conditional on the up side being burnt, there are three *sides* that could be down. Two of those sides are burnt. So the probability is 2 out of 3.

Probability theory is not difficult mathematically. It is just counting. But it is hard to interpret and apply. Doing so often seems to require some cleverness, and authors have an incentive to solve problems in clever ways, just to show off. But we don't need that cleverness, if we ruthlessly apply conditional probability. And that's the real trick of the Bayesian approach: to apply conditional probability in all places, for data and parameters. The benefit is that once we define our information state—our assumptions—we can let the rules of probability do the rest. The work that gets done is the revelation of the implications of our assumptions. Model fitting, as we've been practicing it, is the same un-clever approach. We define the model and introduce the data, and conditional probability does the rest, revealing the implications of our assumptions, in light of the evidence.

In this chapter, you'll meet two commonplace applications of this assume-and-deduce strategy. The first is the incorporation of **measurement error** into our models. The second is the estimation of **missing data** through **Bayesian imputation**. You'll see a fully worked, introductory example of each.

In neither application do you have to intuit the consequences of measurement errors nor the implications of missing values in order to design the models. All you have to do is state your information about the error or about the variables with missing values. Logic does the rest. Well, your computer does the rest. But it's just using fancy algorithms to perform Bayesian updating. It's not at all clever. But the implications it reveals are both counterintuitive and valuable.

15.1. Measurement error

Back in Chapter 5, you met the divorce and marriage data for the United States. Those data demonstrated a simple spurious association among the predictors, as well as how multiple regression can sort it out. What we ignored at the time is that both the divorce rate variable and the marriage rate variable are measured with substantial error, and that error is reported in the form of standard errors. Importantly, the amount of error varies a lot across States. Here, you'll see a simple and useful way to incorporate that information into the model. Then we'll let logic reveal the implications.

Let's begin by plotting the measurement error of the outcome as an error bar:

R code 15.2

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

# points
plot( d$Divorce ~ d$MedianAgeMarriage , ylim=c(4,15) ,
    xlab="Median age marriage" , ylab="Divorce rate" )

# standard errors
for ( i in 1:nrow(d) ) {
    ci <- d$Divorce[i] + c(-1,1)*d$Divorce.SE[i]
    x <- d$MedianAgeMarriage[i]
    lines( c(x,x) , ci )
}
```

The plot is shown on the left in FIGURE 15.1. Notice that there is a lot of variation in how uncertain the observed divorce rate is, as reflected in varying lengths of the vertical line segments. Why does the error vary so much? Large States provide better samples, so their measurement error is smaller. The data are displayed this way, to show the association between the population size of each State and its measurement error, in the right-hand plot in FIGURE 15.1.

Since the values in same States are more certain than in others, it makes sense for the more certain estimates to influence the regression more. There are all manner of *ad hoc* procedures for weighting some points more than others, and these can help. But they leave a lot of information on the table. And they prevent a helpful phenomenon that arises automatically in the fully Bayesian approach: Information flows among the measurements to provide improved estimates of the data itself. So let's see how to state the information as a model.

Rethinking: Generative thinking, Bayesian inference. Bayesian models are *generative*, meaning they can be used to simulate observations just as well as they can be used to estimate parameters. One benefit of this fact is that a statistical model can be developed by thinking hard about how the data might have arisen. This includes sampling and measurement, as well as the nature of the process we are studying. Then let Bayesian updating discover the implications. These implications may include the inability to infer the generative process from data. Bayes is an honest partner. It is not afraid to hurt your feelings.

15.1.1. Error on the outcome.

To incorporate measurement error, let's begin by thinking generatively. If we were to simulate measurement error, what would it look like? The first

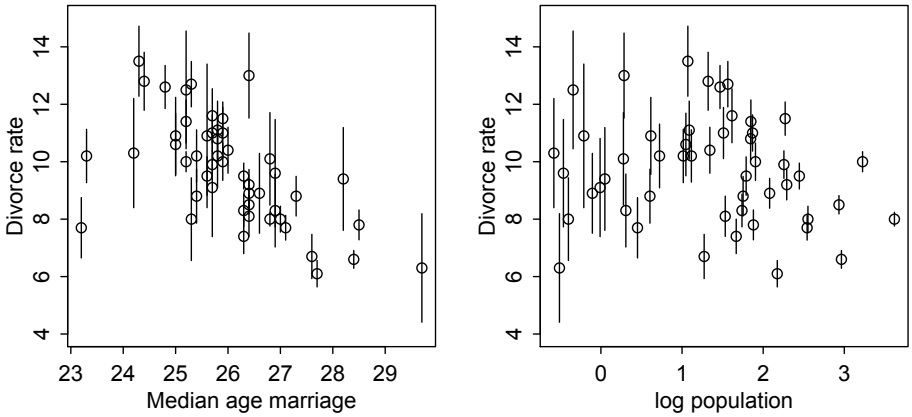
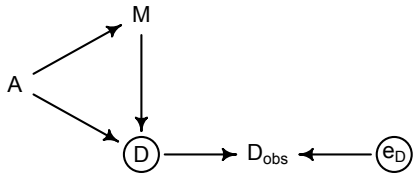


FIGURE 15.1. Left: Divorce rate by median age of marriage, States of the United States. Vertical bars show plus and minus one standard deviation of the Gaussian uncertainty in measured divorce rate. Right: Divorce rate, again with standard deviations, against log population of each State. Smaller States produce more uncertain estimates.

step would be to generate the true values of the variables. Then we simulate the observation process itself, where the measurement error arises. It is just part of the statistical model and likewise part of the causal model.

Recall the causal model of the divorce example from Chapter 5. Let's take that same model and now add observation error on the outcome:



There's a lot going on here. But we can proceed one step at a time. The left triangle of this DAG is the same system that we worked with back in Chapter 5. Age at marriage (A) influences divorce (D) both directly and indirectly, passing through marriage rate (M). Then we have the observation model. The true divorce rate D cannot be observed, so it is circled as an unobserved node. However we do get to observe D_{OBS}, which is a function of both the true rate D and some unobserved error e_{D}.

What are we supposed to do now? Note that D_{OBS} is a descendent of D. Using it in place of D doesn't necessarily introduce confounding. Probably the majority of regressions are really using proxies like D_{OBS}, because most variables are measurements with some error. But even though it doesn't necessarily open a non-causal path, using a proxy can introduce systematic bias, distorting the estimates. Since the extent of measurement error varies across States in a way that is associated with variables of interest, that is likely in this example.

We could do better by using D instead of D_{obs}. But we don't have D. However we can try to reconstruct it, respecting the uncertainty to avoid false confidence. In these data, the

reported standard errors `Divorce.SE` were calculated with knowledge of the process that produces the errors e_{D}. How can use this informaton in a statistical model? It's just like a simulation, but in reverse. If you wanted to simulate measurement error, you would assign a distribution to each observation and sample from it. For example, suppose the true value of a measurement is 10 meters. If it is measured with Gaussian error with standard deviation of 2 meters, this implies a probability distribution for any realized measurement y:

$$y \sim \text{Normal}(10, 2)$$

As the measurement error here shrinks, all the probability piles up on 10. But when there is error, many measurements are more and less plausible. This is what I mean by saying that ordinary data are a special case of a distribution. And here is the key insight: If we don't know the true value (10 in this example), then we can just put a parameter there and let Bayes do the rest.

Here's how to define the error distribution for each divorce rate. For each observed value $D_{\text{OBS},i}$, there will be one parameter, $D_{\text{TRUE},i}$, defined by:

$$D_{\text{OBS},i} \sim \text{Normal}(D_{\text{TRUE},i}, D_{\text{SE},i})$$

All this does is define the measurement $D_{\text{OBS},i}$ as having the specified Gaussian distribution centered on the unknown parameter $D_{\text{TRUE},i}$. So the above defines a probability for each State i's observed divorce rate, given a known measurement error. If you simulated observed divorce rates from known true rates, it would look like:

```
D_obs <- rnorm( N_states , D_true , D_se )
```

A simulation like this goes from assumptions about the distribution to data. When we instead estimate `D_true`, we run it in reverse, using Bayesian updating to go from data to distribution. This is what we've been doing since the beginning.

This is a lot to take in. But we'll go one step at a time. Recall that the goal is to model divorce rate D as a linear function of age at marriage A and marriage rate M. Here's what the model looks like, with the measurement errors highlighted in blue:

$$\begin{aligned}
D_{\text{OBS},i} &\sim \text{Normal}(D_{\text{TRUE},i}, D_{\text{SE},i}) && \text{[distribution for observed values]} \\
D_{\text{TRUE},i} &\sim \text{Normal}(\mu_i, \sigma) && \text{[distribution for true values]} \\
\mu_i &= \alpha + \beta_{\text{A}} A_i + \beta_{\text{M}} M_i && \text{[linear model to assess } A \rightarrow D\text{]} \\
\alpha &\sim \text{Normal}(0, 0.2) \\
\beta_{\text{A}} &\sim \text{Normal}(0, 0.5) \\
\beta_{\text{M}} &\sim \text{Normal}(0, 0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}$$

This is like a linear regression, but with the addition of the top line that connects the observation to the true value. Each D_{TRUE} parameter also gets a second role as the mean of another distribution, one that predicts the observed measurement. A cool implication that will arise here is that information flows in both directions—the uncertainty in measurement influences the regression parameters in the linear model, and the regression parameters in the linear model also influence the uncertainty in the measurements. There will be shrinkage.

Here is the `ulam` version of the model, with all the variables standardized:

R code 15.3

```
dlist <- list(
    D_obs = standardize( d$Divorce ),
```

```
    D_sd = d$Divorce.SE / sd( d$Divorce ),
    M = standardize( d$Marriage ),
    A = standardize( d$MedianAgeMarriage ),
    N = nrow(d)
)

m15.1 <- ulam(
    alist(
        D_obs ~ dnorm( D_true , D_sd ),
        vector[N]:D_true ~ dnorm( mu , sigma ),
        mu <- a + bA*A + bM*M,
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        bM ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    ) , data=dlist , chains=4 , cores=4 )
```

Consider the posterior means (abbreviating the `precis` output below):

R code 15.4

```
precis( m15.1 , depth=2 )
```

```
            mean   sd  5.5% 94.5% n_eff Rhat
D_true[1]   1.18 0.37  0.60  1.78  1696 1.00
D_true[2]   0.68 0.58 -0.20  1.63  2137 1.00
D_true[3]   0.43 0.34 -0.09  0.96  1953 1.00
...
D_true[48]  0.55 0.46 -0.15  1.30  2564 1.00
D_true[49] -0.64 0.27 -1.09 -0.20  3153 1.00
D_true[50]  0.84 0.59 -0.13  1.77  1815 1.00
a          -0.06 0.10 -0.21  0.11  1314 1.00
bA         -0.61 0.16 -0.86 -0.37  1021 1.01
bM          0.05 0.17 -0.21  0.31   936 1.01
sigma       0.60 0.11  0.44  0.78   628 1.00
```

If you look back at Chapter 5, you'll see that the former estimate for `bA` was about -1. Now it's almost half that, but still reliably negative. So compared to the original regression that ignores measurement error, the association between divorce and age at marriage has been reduced. The effect that measurement error has depends upon the context. Sometimes it exaggerates effects, as in this example. Other times it hides them. But you can't safely assume that measurement error makes estimates conservative.[211]

If you look again at FIGURE 15.1, you can see a hint of why this has happened. States with extremely low and high ages at marriage tend to also have more uncertain divorce rates. As a result those rates have been shrunk towards the expected mean defined by the regression line. FIGURE 15.2 displays this shrinkage phenomenon. On the left of the figure, the difference between the observed and estimated divorce rates is shown on the vertical axis, while the standard error of the observed is shown on the horizontal. The dashed line at zero indicates no change from observed to estimated. Notice that States with more uncertain divorce rates—farther right on the plot—have estimates more different from observed. This is your

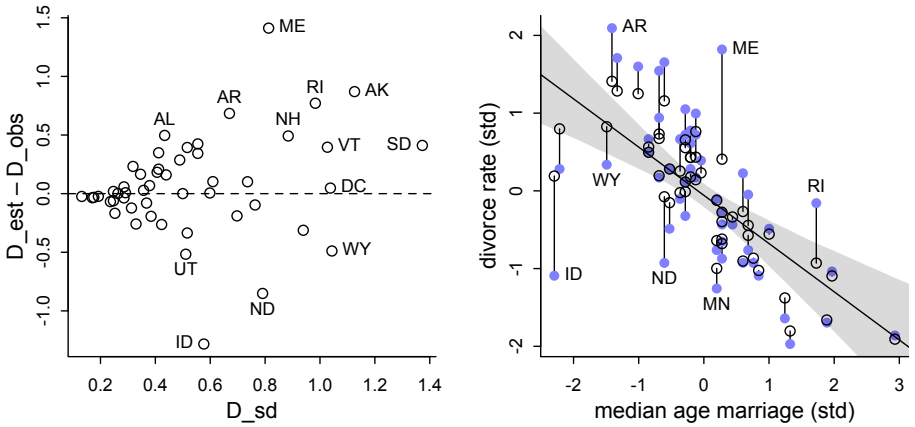


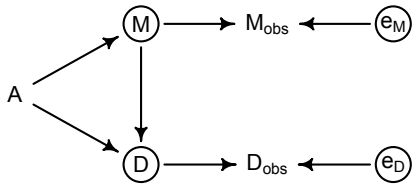
FIGURE 15.2. Left: Shrinkage resulting from modeling the measurement error. The less error in the original measurement, the less shrinkage in the posterior estimate. Right: Comparison of regression that ignores measurement error (dashed line and gray shading) with regression that incorporates measurement error (blue line and shading). The points and line segments show the posterior means and standard deviations for each posterior divorce rate, $D_{\text{EST},i}$.

friend **SHRINKAGE** from the previous two chapters. Less certain estimates are improved by pooling information from more certain estimates.

This shrinkage results in pulling divorce rates towards the regression line, as seen in the right-hand plot in the same figure. This plot shows the posterior mean divorce rate for each State against its observed median age at marriage. The vertical line segments show the posterior standard deviations of each divorce rate—the estimates have moved, but they are still uncertain.

As a result of their movement, however, the regression trend has moved. The old no-error regression is shown in gray. The fancy new with-error regression is shown in blue. Well, really both the estimates and the trend have moved one another at the same time. For a State with an uncertain divorce rate, the trend has strongly influenced the new estimate of divorce rate. For a State with a fairly certain divorce rate—a small standard error—the State has instead strongly influenced the trend. The balance of all of this information is the shift in both the estimated divorce rates and the regression relationship.

15.1.2. Error on both outcome and predictor. What happens when there is measurement error on predictor variables as well? The basic approach is the same. Again, consider the problem generatively: Each observed predictor value is a draw from a distribution with an unknown mean, the true value, but known standard deviation. So we define a vector of parameters, one for each unknown true value, and then make those parameters the means of a family of Gaussian distributions with known standard deviations. Here's the updated DAG:



Now there is a M_{obs} to mirror D_{obs}. Likewise there is an error e_M to match. This DAG assumes that the errors e_D and e_M are independent of one another. This is not necessarily the case.

Here's the updated model, with the new bits in blue:

$$
\begin{aligned}
D_{\text{OBS},i} &\sim \text{Normal}(D_{\text{TRUE},i}, D_{\text{SE},i}) && \text{[distribution for observed } D \text{ values]} \\
D_{\text{TRUE},i} &\sim \text{Normal}(\mu_i, \sigma) && \text{[distribution for true } D \text{ values]} \\
\mu_i &= \alpha + \beta_{\text{A}} A_i + \beta_{\text{M}} M_{\text{TRUE},i} && \text{[linear model]} \\
M_{\text{OBS},i} &\sim \text{Normal}(M_{\text{TRUE},i}, M_{\text{SE},i}) && \text{[distribution for observed } M \text{ values]} \\
M_{\text{TRUE},i} &\sim \text{Normal}(0, 1) && \text{[distribution for true } M \text{ values]} \\
\alpha &\sim \text{Normal}(0, 0.2) \\
\beta_{\text{A}} &\sim \text{Normal}(0, 0.5) \\
\beta_{\text{M}} &\sim \text{Normal}(0, 0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

The M_{TRUE} parameters will hold the posterior distributions of the true marriage rates. And fitting the model is much like before:

R code 15.5

```
dlist <- list(
    D_obs = standardize( d$Divorce ),
    D_sd = d$Divorce.SE / sd( d$Divorce ),
    M_obs = standardize( d$Marriage ),
    M_sd = d$Marriage.SE / sd( d$Marriage ),
    A = standardize( d$MedianAgeMarriage ),
    N = nrow(d)
)

m15.2 <- ulam(
    alist(
        D_obs ~ dnorm( D_true , D_sd ),
        vector[N]:D_true ~ dnorm( mu , sigma ),
        mu <- a + bA*A + bM*M_true[i],
        M_obs ~ dnorm( M_true , M_sd ),
        vector[N]:M_true ~ dnorm( 0 , 1 ),
        a ~ dnorm(0,0.2),
        bA ~ dnorm(0,0.5),
        bM ~ dnorm(0,0.5),
        sigma ~ dexp( 1 )
    ) , data=dlist , chains=4 , cores=4 )
```

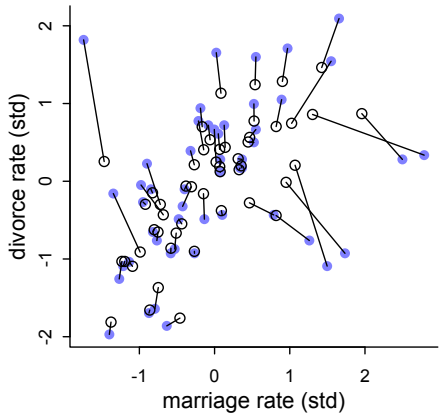


FIGURE 15.3. Shrinkage of both divorce rate and marriage rate. Solid points are the observed values. Open points are posterior means. Lines connect pairs of points for the same State. Both variables are shrunk towards the inferred regression relationship.

If you inspect the `precis` output, you'll see that the coefficients for age at marriage and marriage rate are essentially unchanged from the previous model. So adding error on the predictor didn't change the major inference. But it did provide updated estimates of marriage rate itself. We can visualize this by the shrinkage of both marriage and divorce rates:

R code 15.6

```
post <- extract.samples( m15.2 )
D_true <- apply( post$D_true , 2 , mean )
M_true <- apply( post$M_true , 2 , mean )
plot( dlist$M_obs , dlist$D_obs , pch=16 , col=rangi2 ,
    xlab="marriage rate (std)" , ylab="divorce rate (std)" )
points( M_true , D_true )
for ( i in 1:nrow(d) )
    lines( c( dlist$M_obs[i] , M_true[i] ) , c( dlist$D_obs[i] , D_true[i] ) )
```

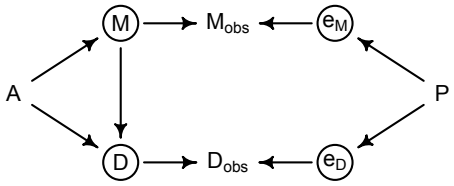
The result is FIGURE 15.3. What has happened is that since the States with highly uncertain marriage rates tend to be small States with high marriage rates, pooling has resulted in smaller estimates for those States.

The big take home point for this section is that when you have a distribution of values, don't reduce it down to a single value to use in a regression. Instead, use the entire distribution. Anytime we use an average value, discarding the uncertainty around that average, we risk overconfidence and spurious inference. This doesn't only apply to measurement error, but also to cases in which data are averaged before analysis.

In the previous model, with error on both the outcome and one of the predictors, we used a standardized Normal(0,1) prior for the M values. This is okay, but it ignores some information. Consider again the DAG for this system: $A \rightarrow M \rightarrow D, A \rightarrow D$. This implies that a better prior for the M values would include A as a predictor. In other words, the entire generative model belongs. We'll attempt this in a practice problem at the end of the chapter.

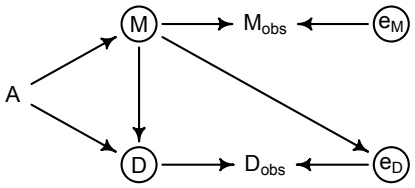
15.1.3. Measurement terrors. In the models above, measurement error is rather benign. The errors are uncorrelated with one another and with the other variables in the model. This means there are no new confounds (non-causal paths) introduced by the errors. But sometimes errors are more difficult to manage.

Consider for example a DAG in which the errors on D and M are correlated with one another, because they are both influenced by a variable P:



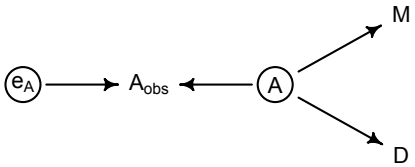
In this case, if we naively regress $\mathrm{D}_{\mathrm{obs}}$ on $\mathrm{M}_{\mathrm{obs}}$, then there is an open non-causal path through P. If we have information about the measurement process, such that we can model the true variables D and M, there is still hope. But we'll need to consider the covariance between the errors. This is computationally similar to how we did instrumental variable regression in the previous chapter. There's a problem at the end of this chapter where I ask you to attempt this.

Another unfortunate situation can arise when another variable influences the error and creates another non-causal path. For example, suppose that the true marriage rate M influences the error on divorce rate D:



Why might this happen? If marriages are rare, then there aren't as many couples that could possibly get divorced. This means a smaller sample size to measure the divorce rate. So smaller M induces a larger error e_{D}. This produces a non-causal path from M_{obs} to D_{obs} that passes through e_{D}. And again, if we can average other the uncertainty in the true M and D, using information about the measurement process, then we might do alright. But ignoring the measurement error isn't alright. And that's what almost everyone does almost every time.[212]

Another pattern of measurement error to worry about is when a causal variable is measured less precisely than a non-causal variable. Suppose for example that we know D and M very precisely but that now A is measured with error. Also assume that M has zero causal effect on D, like this:



In this circumstance, it can happen that a naive regression of D on A_{obs} and M will strongly suggest that M influences D. The reason is that M contains information about the true A. And M is measured more precisely than A is. It's like a proxy A. Here's a small simulation you can toy with that will produce such a frustration:

R code 15.7

```
N <- 500
A <- rnorm(N)
M <- rnorm(N,-A)
D <- rnorm(N,A)
A_obs <- rnorm(N,A)
```

In a problem at the end of the Chapter, I'll ask you to investigate this further.

When you have your own data and your own particular measurement concerns, all of this can be overwhelming. But the way to proceed is same as always: Use your background knowledge to write down a generative model or models, simulate data from these models in order to understand the inferential risks, and design a statistical approach that can work at least in theory.

15.2. Missing data

With measurement error, the insight is to realize that any uncertain piece of data can be replaced by a distribution that reflects uncertainty. But sometimes data are simply missing—no measurement is available at all. At first, this seems like a lost cause. What can be done when there is no measurement at all, not even one with error?

The most common treatment of missing values is just to drop all cases with any missing values. This is known as **COMPLETE CASE ANALYSIS**. It is the default and silent behavior of most statistical software. Another common response is to replace missing values with some assumed value, like the mean of the variable or a reference value like zero. Neither of these treatments is safe. Complete case analysis is at best inefficient. It throws away data. But it can also produce bias, depending upon the causal details. Replacing missing values with static values is never warranted—we do not know those values, and if you fix them, the model will think it knows them with certainty.

So what can we do instead? We can think causally about missingness, and we can use the model to **IMPUTE** missing values. A generative model tells you whether the process that produced the missing values will also prevent the identification of causal effects. Sometimes it does. Other times it does not. Luckily, we can add missingness to a DAG and use the same criteria you already learned to figure out whether it produces confounding. A generative model also provides information about values you have not yet seen.[213] And this information can be used to average over our uncertainty and make full use of the non-missing values, dropping nothing.

All this will become clearer, if we draw some diagrams. We'll start with some simple, fictional examples. Then we'll turn to some real examples.

Rethinking: Missing data are meaningful data. The fact that a variable has an unobserved value is still an observation. It is data, just with a very special value. The meaning of this value depends upon the context. Consider for example a questionnaire on personal income. If some people refuse to fill in their income, this may be associated with low (or high) income. Therefore a model that tries to predict the missing values can be enlightening. In ecology, the absence of an observation of a species is a subtle kind of observation. It could mean the species isn't there. Or it could mean it is there but you didn't see it. An entire category of models, **OCCUPANCY MODELS**,[214] exists to take this duality into account. Missing values are always produced by some process, and thinking about that process can sometimes solve big problems.

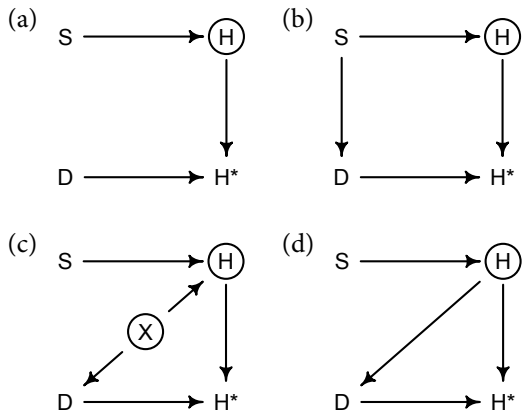


FIGURE 15.4. Four causal scenarios for the missing homework. See text for a complete explanation. (a) Dogs (D) eat homework (H) completely at random. (b) Dogs eat homework of students who study (S) too much. (c) Dogs eat more homework in noisy (X) homes, where the homework is also worse. (d) Dogs prefer to eat bad homework.

15.2.1. DAG ate my homework. Consider a sample of students, all of whom own dogs. The students produce homework (H). This homework varies in quality, influenced by how much each student studies (S). We could simulate 100 students, their attributes, and their homework like this:

R code 15.8

```
N <- 100
S <- rnorm( N )
H <- rbinom( N , size=10 , inv_logit(S) )
```

I've assumed here that homework H will be graded on a 10-point scale. More studying produces more points, on average.

And then some dogs eat some homework. One way to get a grasp on the problem of missing data is to think of missingness as its own variable, a 0/1 indicator for missingness. So let D be a 0/1 indicator variable for whether each dog ate homework. Once homework has been eaten, we cannot observe the true distribution of homework. But we do get to observe H^*, a copy of H with missing values where $D = 1$. In DAG form, this implies $H \rightarrow H^* \leftarrow D$.

We'd like to learn the causal influence of studying (S) on homework (H), $S \rightarrow H$. But since we don't observe H, we have to use H^* instead. So we are relying on $S \rightarrow H^*$ being a good approximation of $S \rightarrow H$. When will this be true? The impact of any missing values in H^* depends upon how the missing values are generated. It depends upon their cause. Let's consider four scenarios, depicted as DAGs in FIGURE 15.4.

The simplest scenario, (a) in the upper left, is when dogs are completely random. A dog's decision to eat a piece of homework or not is not influenced by any relevant variable. Therefore there is no arrow entering D in the DAG. Let's simulate some random eating:

R code 15.9

```
D <- rbern( N ) # dogs completely random
Hm <- H
Hm[D==1] <- NA
```

That `Hm` variable is H^*. We can't use $\star$ in a variable name. Look inside `Hm` and you'll see random `NA`s scattered about. Is this a problem? We can decide by considering whether the outcome H is independent of D. More generally, a minimal condition for missing values to

be benign is that the outcome is independent of (d-separated from) them. In this case, H is independent of D ($H \perp\!\!\!\perp D$), because H^* is a collider.

A more intuitive way to think about this scenario is the following. Since the missing values are completely random, missingness doesn't necessarily change the overall distribution of homework scores. It removes data, and that makes estimation less efficient. But missing homework doesn't necessarily bias our estimate of the causal effect of studying. You should try to build a binomial model to estimate the causal effect of S on H, using both the completely observed data and the data with missing values. There's a practice problem at the end of this chapter that asks you to do this.

Now consider DAG (b) in the upper right of FIGURE 15.4. Here studying influences whether a dog eats homework, S $\rightarrow$ D. Suppose for example that students who study a lot do not play with their dogs. Then the dogs take revenge by eating homework. Again let's simulate:

R code 15.10

```
D <- ifelse( S > 0 , 1 , 0 )
Hm <- H
Hm[D==1] <- NA
```

Now every student who studies more than average (0) is missing homework. This scenario isn't as benign as the previous. But it isn't doom either. Notice that there is now a non-causal path (a backdoor path) from $H \rightarrow$ H* $\leftarrow$ D $\leftarrow$ S. If we don't close this path, it will confound inference along S $\rightarrow$ H. Luckily, we can close the non-causal path by conditioning on S, and we want to condition on S anyway. So this scenario isn't necessarily bad, as long as we can condition on the variable that influences missingness (the dogs D). Again there is a problem at the end that asks you to compare inference with all the homework and without missing homework.

This doesn't mean there is no danger here. If we get the functions or distributions wrong, then we may get the wrong answer and the missing data may prevent us from seeing the absurdity of it in posterior predictive checks. Suppose for example that studying doesn't help at all until a student does more than the average amount (0). In that case, we never get to see homework from those students, so we can't possibly figure out the function that relates study effort to homework score.

The next scenario, FIGURE 15.4 (c), is more difficult. The basic situation is the same: There is a variable that influences both H and D. Previously this was S. Now it is a new variable X, the noisy level of the student's house. In a noisy house, students produce worse homework, $X \rightarrow H$. Simultaneously, dogs in noisy houses tend to misbehave, $X \rightarrow D$. I've put a circle around X to signal that it is unobserved. Now when we regress H* on S, a new non-causal path is in play: $H^* \leftarrow D \leftarrow X \rightarrow H$.

The tricky question, however, is what effect this path has on our estimate of $S \rightarrow H$. Let's actually code this one out, using the simulated data. Here's a complete data simulation for the DAG in FIGURE 15.4 (c):

R code 15.11

```
set.seed(501)
N <- 1000
X <- rnorm(N)
S <- rnorm(N)
H <- rbinom( N , size=10 , inv_logit( 2 + S - 2*X ) )
```

```
D <- ifelse( X > 1 , 1 , 0 )
Hm <- H
Hm[D==1] <- NA
```

Assuming a simple binomial model, first let's see what we get when we fully observe H. Remember, we haven't observed X, so we can't put it in the model.

R code 15.12

```
dat_list <- list(
    H = H,
    S = S )

m15.3 <- ulam(
    alist(
        H ~ binomial( 10 , p ),
        logit(p) <- a + bS*S,
        a ~ normal( 0 , 1 ),
        bS ~ normal( 0 , 0.5 )
    ), data=dat_list , chains=4 )
precis( m15.3 )
```

```
   mean   sd 5.5% 94.5% n_eff Rhat
a  1.11 0.03 1.07  1.15  1265    1
bS 0.69 0.03 0.65  0.73  1366    1
```

The true coefficient on S should be 1.00. We don't expect to get that exactly, but the estimate above is way off. This model used the complete data, before dogs ate any homework, so it can't be missingness that is the problem. This is just a case of **OMITTED VARIABLE BIAS** (Chapter 10). Recall that in a generalized linear model, even if an unobserved variable like X doesn't structurally confound or interact with the predictor of interest like S, that doesn't mean that it won't cause bias in estimation of the effect of S. The reason is that there are ceiling and floor effects on the outcome variable that induce interactions among all predictors.

Now what impact does missing data have? Surely it will make things even worse. Let's see. We'll run the same model now, but with H^* instead of H, dropping all cases where $D = 1$.

R code 15.13

```
dat_list0 <- list(
    H = H[D==0],
    S = S[D==0] )

m15.4 <- ulam(
    alist(
        H ~ binomial( 10 , p ),
        logit(p) <- a + bS*S,
        a ~ normal( 0 , 1 ),
        bS ~ normal( 0 , 0.5 )
    ), data=dat_list0 , chains=4 )
precis( m15.4 )
```

```
   mean   sd 5.5% 94.5% n_eff Rhat
```

```
a  1.80 0.04 1.74  1.85  1051    1
bS 0.83 0.03 0.78  0.88  1060    1
```

The estimate for bS is still biased, but not as badly. This is only one example, but you can run thousands of simulations like this one (I show you how in the Overthinking box at the end of the section), and you'll get this pattern on average. How has dropping students helped our estimate? The homework that is missing is from noisy houses. And it is noisy houses that mess up our estimate of bS, through omitted variable bias. So when we delete those houses from the data, the estimate actually gets better.

Note that this improvement is not a general property of missing data in such a DAG. For example, if you change the missingness rule instead to:

R code 15.14

```
D <- ifelse( abs(X) < 1 , 1 , 0 )
```

now missingness actually makes the estimate worse. Give it a try. What happens under missingness depends upon the details of the functions in the full structural causal model. The DAG isn't enough to say what will happen. But the DAG is enough to say that we should be wary.

Just one more set of dogs remain. In FIGURE 15.4 (d), there is no X, but there is a path from $H \rightarrow D$. Now dogs prefer to eat bad homework. This is possibly because their owners feed it to them, but maybe it somehow tastes better too. To simulate from this DAG:

R code 15.15

```
N <- 100
S <- rnorm(N)
H <- rbinom( N , size=10 , inv_logit(S) )
D <- ifelse( H < 5 , 1 , 0 )
Hm <- H
Hm[D==1] <- NA
```

Go ahead and try to estimate the causal effect $S \rightarrow H$. You won't be able to do a good job. And there is nothing to do here, because there is nothing we can condition on to block the non-causal path $S \rightarrow H \rightarrow D \rightarrow H^*$. This type of missingness, in which the variable causes its own missing values, is the worst. Unless you know the mechanism that produces the missingness (D in this case), there is little hope. And if you do you know the mechanism, even then it might be hopeless. Sometimes measurement is all we have. We have to do it better.

The point of these examples is not to give you nightmares. The point is to illustrate that the consequences of missing data are diverse. But the diversity is explicable causally, in terms of which variables cause missing values in which other variables. And the point of emphasizing simulation in these examples is to empower you to explore your own scenarios, the ones relevant to your own research. Even when we cannot completely eliminate the impact of missing data, we might be able to show, through simulation, that the expected impact is rather small.

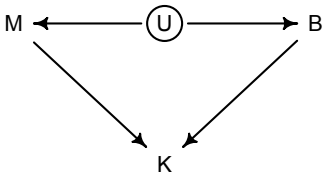
Rethinking: Naming completely at random. Statistical terminology can be very confusing. The field uses ordinary words in highly technical ways. The everyday meanings of words like *likelihood*, *significant*, and *confidence* barely resemble their statistical definitions. The topic of missing data is no better. The dog-homework scenarios (FIGURE 15.4) sometimes go by the unhelpful names (a)

MISSING COMPLETELY AT RANDOM (MCAR), (b) and (c) MISSING AT RANDOM (MAR), and (d) the impressively absurd MISSING NOT AT RANDOM (MNAR).[215] The semantic difference between random and completely random is insignificant for nearly all people. No one likes these terms, but you'll still see them in use. Even if these terms were easy to remember, they are not sufficient to decide how to handle missing data, as the difference between scenarios (b) and (c) demonstrates. Don't worry about categorization. Sketch the causal model, and then figure out your next move.

15.2.2. Imputing primates. Addressing missing data often involves the IMPUTATION of missing values. We impute both to avoid biased estimation and so that we can use all of the observed (not missing) data. The key idea with imputation is that any generative model necessarily contains information about variables that have not been observed. Some data go missing, but the model stays the same. In theory then imputing missing data is easy. In practice there can be challenges, as always.

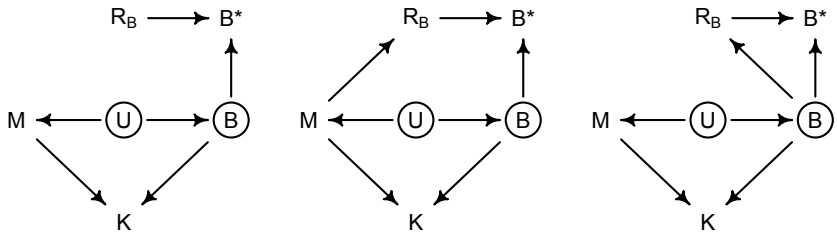
To see how this works, let's return to the primate milk example, from Chapter 5. We used `data(milk)` to illustrate masking, using both neocortex percent and body mass to predict milk energy. One aspect of those data are 12 missing values in the `neocortex.perc` column. We used a COMPLETE-CASE analysis back then, which means we dropped those 12 cases from the analysis. That means we also dropped 12 perfectly good body mass and milk energy values. That left us with only 17 cases to work with. Was that a bad idea?

To answer that question, we need to think more clearly about why those values are missing. The basic DAG from this example is:



where M is body mass, B is neocortex percent, K is milk energy, and U is some unobserved variable that renders M and B positively correlated. We want to add missingness to this graph, just like we added missingness to the dog-homework graphs in the previous section. We haven't observed B (neocortex percent). We've instead observed B^*, a partially observed set of values generated by B and some process. Which process? We don't know yet. All we know is that the observed values B^* are a function of B and the "missingness" process. Whatever the process, it generates a variable R_B that indicates which species have missing values. The variable R_B is like the vector of dogs D in the dog-homework section.

The crucial question is which variables influence R_B. Let's consider three possibilities.



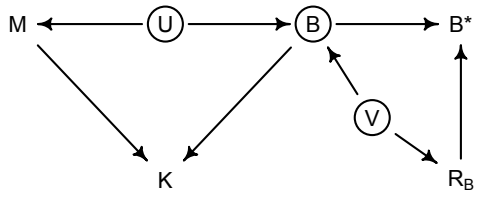
In all three DAGs above, the variable B is circled now to indicate that it in unobserved. Each DAG is a different hypothesis about what causes the missing brain values R_B. Let's consider each, going from left to right.

On the left, nothing influences R_B. It is completely random. In this case, there is no new non-causal path introduced. Dropping the species with missing brain values wastes information—it means dropping all the observed mass values too—but it doesn't necessarily bias inference.

In the middle, now body mass M influences which species have missing values. This could happen, for example, if smaller primates like lemurs are less often studied than larger primates like gorillas. If M influences R_B, it also creates a new non-causal path $B^* \leftarrow R_B \leftarrow M \rightarrow K$. But luckily conditioning on M blocks this path, and we want to condition on M anyway. We still want to impute missing values, so that we don't throw away information.

How do we know if M influences R_B? You could test this idea by trying to measure the causal influence of M on R_B. But keep in mind that all that backdoor path stuff still applies. In a practice problem at the end of this chapter, I'll ask you to estimate the causal influence of M on R_B.

The third example DAG, on the right, shows brain size B itself influencing R_B. This could happen because anthropologists are more interested in large brained species. There is a lot more research on chimpanzees, for example, than on lemurs. This scenario is awful. If true, it means that estimation of $B \rightarrow K$ will be biased by a non-causal path through R_B. It will also not be possible to test, with these data, whether B influences R_B. Lots of different graphs can lead to this scenario. Here's another possibility:



Now it isn't the B values themselves that produce missingness. Rather there is an unobserved variable V that influences both B and R_B. V could be for example phylogenetic similarity to humans. Humans have an unreasonable amount of neocortex—that is reason we pay attention to it—and other primates closely related to us also tend to have more neocortex. If those primates are studied more intensely, B values will be missing more as distance from humans increases. Just about the only hope in this scenario is the have detailed knowledge of the process that produces R_B, allowing imputation of B. And that will nearly always require strong modeling assumptions, assumptions which usually cannot be tested with the data at hand.

In every DAG described above, we want to impute missing values of B. In the first and second, we do so in order to not throw away corresponding values of M. In the third, we have to impute to hope for any sensible estimate of $B \rightarrow K$. So let's see how to actually do the imputation.

The statistical trick with Bayesian imputation is to model the variable that has missing values. Each missing value is assigned a unique parameter. The observed values give us information about the distribution of the values. This distribution becomes a prior for the missing values. This prior will then be updated by full model. So there will be a posterior

distribution for each missing value. Conceptually this is like the measurement error case—if we don't know something, we condition it on what we know and let Bayes figure it out.

In our case, the variable with missing values is neocortex percent. Again, we'll call it B, for "brain":

$$B = [0.55, B_2, B_3, B_4, 0.65, 0.65, ..., 0.76, 0.75]$$

For every index i at which there is a missing value, there is also a parameter B_i that will form a posterior distribution for it. The simplest model will simply impute B from its own normal distribution. Here it is, with the neocortex pieces in blue:

$$\begin{aligned}
K_i &\sim \text{Normal}(\mu_i, \sigma) && \text{[distribution for outcome } k\text{]} \\
\mu_i &= \alpha + \beta_B B_i + \beta_M \log M_i && \text{[linear model]} \\
B_i &\sim \text{Normal}(\nu, \sigma_B) && \text{[distribution for obs/missing } B\text{]} \\
\alpha &\sim \text{Normal}(0, 0.5) \\
\beta_B &\sim \text{Normal}(0, 0.5) \\
\beta_M &\sim \text{Normal}(0, 0.5) \\
\sigma &\sim \text{Exponential}(1) \\
\nu &\sim \text{Normal}(0.5, 1) \\
\sigma_B &\sim \text{Exponential}(1)
\end{aligned}$$

This model ignores that B and M are associated through U. But let's start with this model, just to keep things simple. The interpretation of $B_i \sim \text{Normal}(\nu, \sigma_B)$ is awkward at first. Note that when B_i is observed, then this line is a likelihood, just like any old linear regression. The model learns the distributions of ν and σ_B that are consistent with the data. But when B_i is missing and therefore a parameter, that same line is interpreted as a prior. Since the parameters ν and σ_B are also estimated, the prior is learned from the data, just like the varying effects in previous chapters.

One issue with this model is that it assumes each B value has a standardized Gaussian uncertainty. But we know that these values are bounded between zero and one, because they are proportions. So it is possible to do a little better. In the practice problems at the end of the chapter, you'll see how. But keep in mind that assigning a Gaussian distribution doesn't really mean that the frequency distribution of the variable is a bell curve. It just means we will use only the mean and variance to describe it. The Gaussian is a very conservative choice, because it is the flattest unbounded distribution with a given variance (Chapter 10). But as described way back in Chapter 7, if you have reason to suspect the tails of the distribution are thick, then definitely do not use a Gaussian distribution.

Implementing an imputation model can be done several different ways. All of the ways are a little awkward, because the locations of missing values have to respected, and that means plenty of index management. The approach I'll use here hews closely to the discussion just above: We'll merge the observed values and parameters into a vector that we'll use as "data" in the regression. For convenience, `ulam` can automate this merging. The Overthinking box at the end of this section presents a full implementation in raw Stan code.

To fit the model with `ulam`, first get the data loaded and transform the predictors:

R code 15.16

```
library(rethinking)
data(milk)
```

```
d <- milk
d$neocortex.prop <- d$neocortex.perc / 100
d$logmass <- log(d$mass)
dat_list <- list(
    K = standardize( d$kcal.per.g ),
    B = standardize( d$neocortex.prop ),
    M = standardize( d$logmass ) )
```

The model code looks absolutely ordinary, except for defining a distribution for `B`.

R code 15.17

```
m15.3 <- ulam(
    alist(
        K ~ dnorm( mu , sigma ),
        mu <- a + bB*B + bM*M,
        B ~ dnorm( nu , sigma_B ),
        c(a,nu) ~ dnorm( 0 , 0.5 ),
        c(bB,bM) ~ dnorm( 0, 0.5 ),
        sigma_B ~ dexp( 1 ),
        sigma ~ dexp( 1 )
    ) , data=dat_list , chains=4 , cores=4 )
```

When you start the model, it will notify you that it found 12 `NA` values and is trying to impute them. Once it finishes, take a look at the posterior summary:

R code 15.18

```
precis( m15.3 , depth=2 )
```

```
               mean   sd  5.5% 94.5% n_eff Rhat
nu            -0.04 0.20 -0.35  0.28  2013    1
a              0.03 0.16 -0.22  0.28  2319    1
bM            -0.55 0.21 -0.88 -0.21  1238    1
bB             0.50 0.25  0.09  0.88   909    1
sigma_B        1.00 0.17  0.77  1.31  1593    1
sigma          0.84 0.15  0.63  1.11  1266    1
B_impute[1]   -0.56 0.91 -1.95  0.95  2602    1
B_impute[2]   -0.69 0.91 -2.10  0.79  2025    1
B_impute[3]   -0.68 0.94 -2.10  0.84  2086    1
B_impute[4]   -0.25 0.87 -1.61  1.15  3091    1
B_impute[5]    0.48 0.85 -0.93  1.82  2532    1
B_impute[6]   -0.16 0.85 -1.50  1.16  2626    1
B_impute[7]    0.19 0.85 -1.08  1.58  2640    1
B_impute[8]    0.28 0.86 -1.06  1.62  3697    1
B_impute[9]    0.52 0.87 -0.93  1.84  2574    1
B_impute[10] -0.46 0.89 -1.87  0.93  2092    1
B_impute[11] -0.27 0.86 -1.61  1.09  2650    1
B_impute[12]  0.17 0.85 -1.21  1.49  2749    1
```

Each of the 12 imputed distributions for missing values is shown here, along with the ordinary regression parameters above them. To see how including all cases has impacted inference, let's do a quick comparison to the estimates that drop missing cases. I'll drop the cases with missing values, but the model will be identical.

R code 15.19

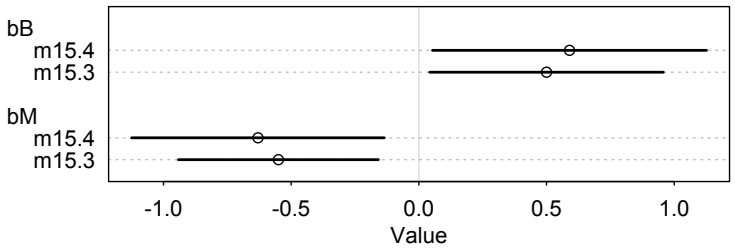
```
obs_idx <- which( !is.na(d$neocortex.prop) )
dat_list_obs <- list(
    K = dat_list$K[obs_idx],
    B = dat_list$B[obs_idx],
    M = dat_list$M[obs_idx] )
m15.4 <- ulam(
    alist(
        K ~ dnorm( mu , sigma ),
        mu <- a + bB*B + bM*M,
        B ~ dnorm( nu , sigma_B ),
        c(a,nu) ~ dnorm( 0 , 0.5 ),
        c(bB,bM) ~ dnorm( 0, 0.5 ),
        sigma_B ~ dexp( 1 ),
        sigma ~ dexp( 1 )
    ) , data=dat_list_obs , chains=4 , cores=4 )
precis( m15.4 )
```

```
         mean   sd  5.5% 94.5% n_eff Rhat
nu       0.00 0.22 -0.34  0.37  1821    1
a        0.10 0.20 -0.21  0.42  1923    1
bM      -0.63 0.25 -1.01 -0.21  1276    1
bB       0.59 0.27  0.14  1.01  1244    1
sigma_B  1.04 0.18  0.79  1.36  1458    1
sigma    0.88 0.19  0.64  1.20  1145    1
```

Comparing this posterior to the previous will be easier with a plot:

R code 15.20

```
plot( coeftab(m15.3,m15.4) , pars=c("bB","bM") )
```



The model that imputes the missing values, `m15.3`, has narrower marginal distributions for both effects. How could this happen? We used more information, the values of body mass that are not missing but are discarded by `m15.4`. These values suggest a slightly smaller influence of body mass, `bM`, and this also cascades into `bB`.

Let's do some plotting to visualize what's happened here.

R code 15.21

```
post <- extract.samples( m15.3 )
B_impute_mu <- apply( post$B_impute , 2 , mean )
B_impute_ci <- apply( post$B_impute , 2 , PI )

# B vs K
```

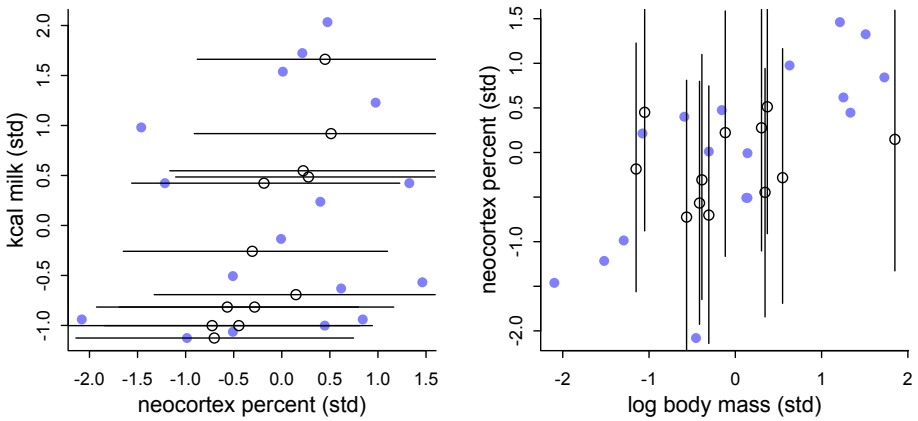


FIGURE 15.5. Left: Inferred distribution of milk energy (vertical) and neocortex proportion (horizontal), with imputed values shown by open points. The line segments are 89% posterior compatibility intervals. Right: Inferred distribution between the two predictors, neocortex proportion and log mass. Imputed values again shown by open points.

```
plot( dat_list$B , dat_list$K , pch=16 , col=rangi2 ,
    xlab="neocortex percent (std)" , ylab="kcal milk (std)" )
miss_idx <- which( is.na(dat_list$B) )
Ki <- dat_list$K[miss_idx]
points( B_impute_mu , Ki )
for ( i in 1:12 ) lines( B_impute_ci[,i] , rep(Ki[i],2) )

# M vs B
plot( dat_list$M , dat_list$B , pch=16 , col=rangi2 ,
    ylab="neocortex percent (std)" , xlab="log body mass (std)" )
Mi <- dat_list$M[miss_idx]
points( Mi , B_impute_mu )
for ( i in 1:12 ) lines( rep(Mi[i],2) , B_impute_ci[,i] )
```

FIGURE 15.5 displays both the inferred relationship between milk energy and neocortex (left) and the relationship between the two predictors (right). Both plots show imputed neocortex values in blue, with 89% compatibility intervals shown by the line segments. Although there's a lot of uncertainty in the imputed values—hey, Bayesian inference isn't magic, just logic—they do show a gentle tilt towards the regression relationship. This has happened because the observed values provide information that guides the estimation of the missing values.

The right-hand plot shows the inferred relationship between the two predictors. We already know that these two predictors are positively associated—that's what creates the masking problem. But notice here that the imputed values do not show an upward slope. They do not, because the imputation model—the first regression with neocortex (observed and missing) as the outcome—assumed no relationship.

We can improve this model by changing the imputation model to estimate the relationship between the two predictors. This really just means that we use the entire generative model. In the DAG, B and M are associated as a result of U. If we can include that fact in the model, we might make better imputations and therefore better inferences. The technique is only to change the imputation line of the model from the simple:

$$B_i \sim \text{Normal}(\nu, \sigma_B)$$

to a bivariate normal that includes both M and B:

$$(M_i, B_i) \sim \text{MVNormal}((\mu_M, \mu_B), \mathbf{S})$$

The $\mathbf{S}$ matrix is another covariance matrix, and it will measure the correlation between M and B, using the observed cases, and then use that correlation to infer the missing B values. Note that this is the simplest model we could have of the association between M and B. It assumes that the covariance is sufficient to describe their relationship. That will not always be the case, as many different bivariate relationships can produce the same covariance. If you have a better idea, then you should use that instead.

Here's the `ulam` implementation. This is complex code, because we have to construct a variable that includes both the observed M values and the merged list of observed and imputed B values. I'll also do the merging more explicitly. In the Overthinking box at the end, I walk through the Stan code, explaining some of the coding details.

R code 15.22

```
m15.5 <- ulam(
    alist(
       # K as function of B and M
        K ~ dnorm( mu , sigma ),
        mu <- a + bB*B_merge + bM*M,

       # M and B correlation
        MB ~ multi_normal( c(muM,muB) , Rho_BM , Sigma_BM ),
        matrix[29,2]:MB <<- append_col( M , B_merge ),

       # define B_merge as mix of observed and imputed values
        vector[29]:B_merge <- merge_missing( B , B_impute ),

       # priors
        c(a,muB,muM) ~ dnorm( 0 , 0.5 ),
        c(bB,bM) ~ dnorm( 0, 0.5 ),
        sigma ~ dexp( 1 ),
        Rho_BM ~ lkj_corr(2),
        Sigma_BM ~ dexp(1)
    ) , data=dat_list , chains=4 , cores=4 )
precis( m15.5 , depth=3 , pars=c("bM","bB","Rho_BM" ) )
```

```
              mean   sd  5.5% 94.5% n_eff Rhat
bM           -0.65 0.22 -1.00 -0.30  1262    1
bB            0.58 0.26  0.16  0.99  1048    1
Rho_BM[1,1]   1.00 0.00  1.00  1.00   NaN  NaN
Rho_BM[1,2]   0.60 0.13  0.37  0.78  1592    1
Rho_BM[2,1]   0.60 0.13  0.37  0.78  1592    1
Rho_BM[2,2]   1.00 0.00  1.00  1.00  1981    1
```

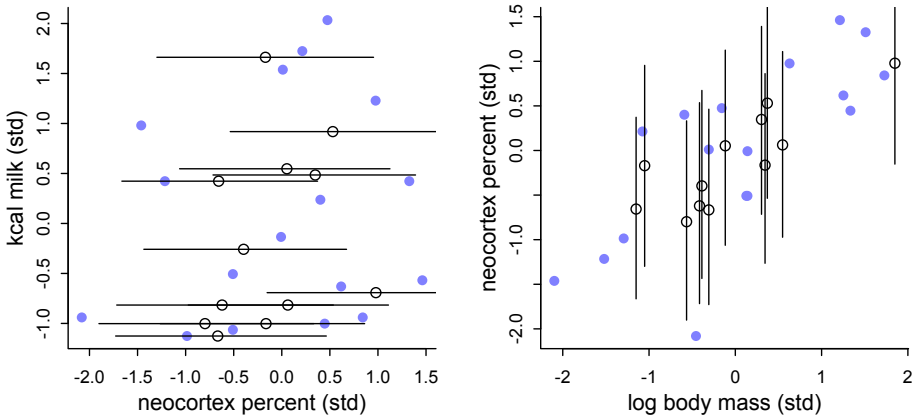


FIGURE 15.6. Same relationships as shown in FIGURE 15.5, but now for the imputation model that estimates the association between the predictors. The information in the association between predictors has been used to infer a stronger relationship between milk energy and the imputed values.

The slopes `bM` and `bB` haven't changed much, although `bM` is perhaps a little more precise now. We're interested in that correlation and how it has influenced the imputed values. The posterior correlation is quite strong, 0.6 on average. This shows the strong positive relationship between M and B that we already knew existed.

What does this correlation do to the imputed values? You can use the same plotting code as before. FIGURE 15.6 displays the same kind of plots as before, but now for the new imputation model. On the right, you can see now that the model has imputed in a way to preserve the positive association between neocortex and log mass. Although in this example this doesn't make a big difference in the inferred relationships with the outcome, it is clearly better. Doing better is good.

Rethinking: Multiple imputations. Missing data imputation has a messy history. There are many forms of imputation, and most of them are *ad hoc* devices without a strong basis in probability theory: Hot-deck imputation, cold-deck imputation, mean substitution, stochastic imputation, among others. None of these procedures is considered respectable today. A common non-Bayesian procedure is **MULTIPLE IMPUTATION**.[216] Multiple imputation was developed in the context of survey non-response, and it actually has a Bayesian justification. But it was invented when Bayesian imputation on the desktop was impractical, so it tries to approximate the full Bayesian solution to a "missing at random" missingness model. If you aren't comfortable dropping incomplete cases, then you should be comfortable using multiple imputation either. The procedure performs multiple draws from an approximate posterior distribution of the missing values, performs separate analyses with these draws, and then combines the analyses in a way that approximates full Bayesian imputation. Multiple imputation is more limited than full Bayesian imputation, so now we just use the real thing. But lots of non-Bayesian analyses still use multiple imputation. Remember that frequentist statistics isn't a theory of how to produce estimates but rather just a theory of how to evaluate them.

Overthinking: Stan imputation algorithm. In principle, imputation is just using the same model but replacing data with parameters. Data are observed variables. Parameters are unobserved variables. The same generative model allows us to learn about both. But in practice, additional programming is necessary. It's necessary, because we have to construct a new variable that is a mix of observed and unobserved values. The `ulam` code for `m15.3` automates this. But it is worth seeing the guts of the machine, because it will increase understanding and teach you how to do this manually, in raw Stan code.

If you inspect the Stan code `stancode(m15.3)`, you'll a `functions` block at the top. This is where you can put special code that you don't want cluttering up the `model` block. In this case:

```
functions{
    vector merge_missing( int[] miss_indexes , vector x_obs , vector x_miss ) {
        int N = dims(x_obs)[1];
        int N_miss = dims(x_miss)[1];
        vector[N] merged;
        merged = x_obs;
        for ( i in 1:N_miss )
            merged[ miss_indexes[i] ] = x_miss[i];
        return merged;
    }
}
```

This code exists only to merge a vector of observed values with a vector of parameters to stand in place of missing values. It is called in the `model` block. Here are the important lines:

```
    B_merge = merge_missing(B_missidx, to_vector(B), B_impute);
    B_merge ~ normal( nu , sigma_B );
    for ( i in 1:29 ) {
        mu[i] = a + bB * B_merge[i] + bM * M[i];
    }
    K ~ normal( mu , sigma );
```

The first line above merges the observed data `B` with the imputation parameters in `B_impute`. The vector `B_missidx` is just a list of the index positions of the missing values. If you use `ulam`, it builds `B_missidx` for you. But if you use Stan directly, you'll need to build it yourself. One line is enough:

R code 15.23

```
B_missidx <- which( is.na( dat_list$B ) )
```

You pass `B_missidx` to the Stan model as data. The function `merge_missing` replaces each missing value with the value of each corresponding parameter in `B_impute`. This is a bit awkward—it is joyless index shuffling. But it gets the job done, and in the end we have a vector `B_merge` that contains both observed values and imputation parameters in all the right places. The next lines of code then use `B_merge`. The second line above is just the probability of the brain (neocortex percent) values, as stated by the model. Then the loop constructs the linear predictor `mu` for each species, which `B_merge` appearing, so that both observed values and imputation parameters are used as appropriate.

You can use `merge_missing` directly in `ulam` models as well. It will declare the merged vector and the vector of imputation parameters. The model `m15.5` contains an example. Even `m15.3` inserts `merge_missing` behind the scenes. See: `m15.3@formula_parsed$formula`. If you use Stan directly, you'll need to declare all of this yourself. You can see the necessary declarations in the `parameters` and `model` blocks of `stancode(m15.3)`.

15.2.3. Where is your god now?

Sometimes there are no statistical solutions to scientific problems. But even then, careful statistical thinking can be useful because it will tell us that there is no statistical solution. Here's an example involving missing data.

Religion is a human universal, as common among human societies as walking on two legs and naming stars. Anthropologists, archaeologists, and scholars of religion are sometimes curious about the impact of religious beliefs on the welfare of human societies. Some of the most successful religious traditions involve gods (and other supernatural entities) that enforce moral norms. For example, in the Abrahamic traditions, God punishes the wicked and rewards the just. Such gods might be called "moralizing gods." In other traditions, gods behave in their own self-interest, with no interest in encouraging humans to cooperate with one another. Does such a difference in belief have any consequences for the society? For example, if people who believe in a moralizing god are better at cooperating with one another, then maybe societies that believe in moralizing gods grow faster and tend to replace societies with less moralizing gods.

Let's look at a set of historical data that was used to evaluate this idea.[217]

R code 15.24

```
data(Moralizing_gods)
str(Moralizing_gods)
```

```
'data.frame': 864 obs. of  5 variables:
 $ polity         : Factor w/ 30 levels "Big Island Hawaii",..: 1 1 1 1 1 1 ...
 $ year           : int  1000 1100 1200 1300 1400 1500 1600 1700 1800 -600 ...
 $ population     : num  3.73 3.73 3.6 4.03 4.31 ...
 $ moralizing_gods: int  NA NA NA NA NA NA NA NA 1 NA ...
 $ writing        : int  0 0 0 0 0 0 0 0 0 0 ...
```

These data are `population` sizes (on the log scale) of different regions (`polity`) in different centuries (`year`). The key explanatory variable is `moralizing_gods`, which indicates whether members of a society believed in supernatural enforcement of morality (1), did not believe (0), or there is insufficient evidence for assigning a value (NA). This last value (NA) is usually associated with lack of any written evidence about religious belief. There is also an indicator variable for literacy (`writing`).

Does belief in moralizing gods increase the rate of population growth? This is a difficult causal query. There are plausibly many unobserved confounds that could produce a non-causal association between population growth rate and the content of religious traditions. And belief in moralizing gods may not produce an immediately detectable increase in population. Instead the causal effect could work over long time periods or only during periods of conflict or ecological stress. Minimally, what we need is some comparison of population growth rates before and after each society adopts moralizing gods. This is not a causal identification strategy that does anything about confounds—the appearance of moralizing gods and larger populations could still be driven by other (unmeasured) variables. There is no sense in which we can think of the year that moralizing gods appear as being a random treatment, in the sense of a **REGRESSION DISCONTINUITY** (Chapter 14, page **??**). But if we playfully assume that there are no confounds, how should we go about this analysis?

The first obstacle is that there are a lot of missing values in the `moralizing_gods` variable. This prevents us from knowing exactly when (if ever) each society adopts belief in moralizing gods. How many values are missing? Let's count:

R code 15.25

```
table( Moralizing_gods$moralizing_gods , useNA="always" )
```

```
   0    1 <NA>
  17  319  528
```

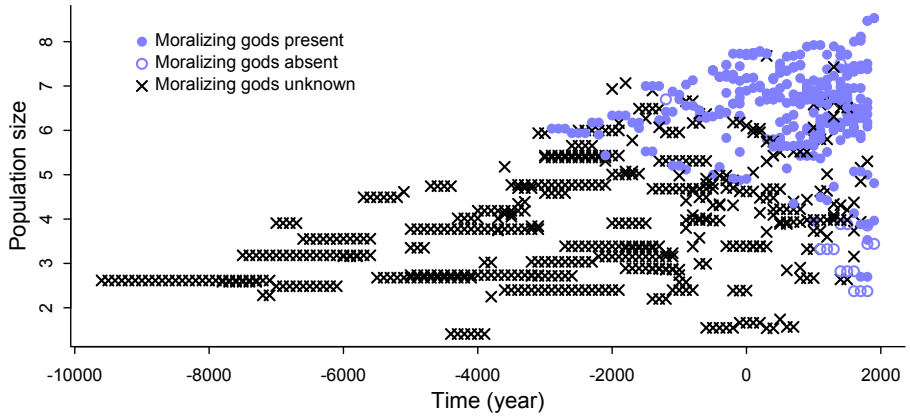


FIGURE 15.7. Missing values in the `Moralizing_gods` data. The blue points, both open and filled, are observed values for the presence of beliefs about moralizing gods. The x symbols are unknowns, the missing values.

Of 864 cases, 528 of them (60%) are missing. Only 17 of the observed cases are zeros, which means "no moralizing gods." This is a lot of missing data, to be sure. But the raw amount of missing data is not necessarily a reason to worry. Remember the homework-eating dogs from earlier—the impact of missing data depends upon the process that produces missing data. If the missing gods are scattered at random, then we're in luck. It'll be useful to visualize the missingness pattern.

R code 15.26

```
symbol <- ifelse( Moralizing_gods$moralizing_gods==1 , 16 , 1 )
symbol <- ifelse( is.na(Moralizing_gods$moralizing_gods) , 4 , symbol )
color <- ifelse( is.na(Moralizing_gods$moralizing_gods) , "black" , rangi2 )
plot( Moralizing_gods$year , Moralizing_gods$population , pch=symbol ,
    col=color , xlab="Time (year)" , ylab="Population size" , lwd=1.5 )
```

The result is shown in FIGURE 15.7. I've just plotted log population against year. The symbols show the value of `moralizing_gods`. Filled blue points have value 1 (belief in moralizing gods known to be present). The open blue points have value 0 (belief in moralizing gods known to be absent). The × symbols are points where the value is `NA`. This is a highly non-random missingness pattern. The reason is that written records are usually needed to determine historical religious beliefs. Let's look at the cross-tabulation of gods and literacy:

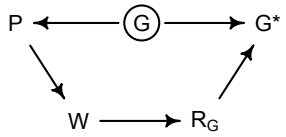
R code 15.27

```
with( Moralizing_gods ,
    table( gods=moralizing_gods , literacy=writing , useNA="always" ) )
```

```
      literacy
gods     0   1 <NA>
  0     16   1    0
  1      9 310    0
  <NA> 442  86    0
```

442 (84%) of 528 missing values are for non-literate polities. No writing means no evidence of any kind, in most cases. And as you can see in FIGURE 15.7, missing values are associated with smaller polities. This is possibly because smaller polities were (in the past) less likely to be literate. These data are structured by the strong association between literacy, moralizing gods, and missing values. Beneath that mass of × symbols in FIGURE 15.7, belief in moralizing gods could be common or rare, depending on your theoretical preference.

This situation cannot be saved by statistics, but it is useful to carefully reason out why. After all, in many cases missing data don't block inference. The details matter. First we must consider whether we can just ignore the missing values, using a **COMPLETE CASE ANALYSIS**. But doing that in this context will almost certainly bias our inference, because the missingness is strongly associated with other variables, like `writing`, which are in turn strongly associated with the outcome. It'll help to consider the causal structure of missingness. Here's my most optimistic guess:



Here P is rate of population growth (not the same as the `population` size variable in the data), G is the presence of belief in moralizing gods (which is unobserved), G^* is the observed variable with missing values, W is writing, and R_G is the missing values indicator. This is an optimistic scenario, because it assumes there are no unobserved confounds among P, G, and W. These are purely observational data, recall. But the goal is to use this example to think through the impact of missing data. If we can't recover from missing data with the DAG above, adding confounds isn't going to help.

Remember from the previous sections that the goal is to determine whether the outcome (here P) is independent of missingness (here R_G). This is clearly not a dog-eats-homework-at-random situation, because R_G is not completely random. It assumes missingness R_G is explained entirely by an observed variable (W). Unfortunately, since P influences W, if we condition on W to try to separate P and R_G, it just adds another confound. In the practice problems at the end, I'll ask you to simulate from the DAG above to verify this claim.

There is one last hope. If we could somehow condition on G instead of G^*, we'd be safe and clear. This is where imputation can help, by reconstructing G with appropriate uncertainty. This is not trivial, however, because successful imputation requires a good approximation of the generative model of the variable. How is G generated? There is no obvious answer. Consider for example the data for Hawaii, which by the time Europeans (Captain James Cook and his crew) finally made contact (in 1778) was a very large and complex polity with moralizing gods. Here is Hawaii:

R code 15.28

```
haw <- which( Moralizing_gods$polity=="Big Island Hawaii" )
columns <- c("year","writing","moralizing_gods")
t( Moralizing_gods[ haw , columns ] )
```

```
                  1    2    3    4    5    6    7    8    9
year           1000 1100 1200 1300 1400 1500 1600 1700 1800
writing           0    0    0    0    0    0    0    0    0
moralizing_gods  NA   NA   NA   NA   NA   NA   NA   NA    1
```

After Captain Cook, Hawaii is correctly coded with 1 for belief in moralizing gods. It is also a fact that Hawaii never developed its own writing system. So there is no direct evidence of when moralizing gods appeared in Hawaii. Any imputation model needs to decide how to fill in those NA values. With so much missing data, any imputation model would necessarily make very strong assumptions.

The strongest assumption would be to just replace all the of the NA values with some constant, like zero. This implies a generative model in which any polity that believes in moralizing gods will never produce a missing value. In the case of Hawaii, it assumes that moralizing gods appear only after Captain Cook arrives. This procedure results in biased estimates of time of adoption of moralizing gods, because presumably more than just Hawaii believed in moralizing gods before they started writing about them. You might think no analyst would impute missing values this way. But this sort of arbitrary imputation is not rare.[218]

What else could we do? In principle we could perform a model-based imputation of the missing values in `moralizing_gods`. But we don't have any obviously correct way to do this. We can't just associate presence/absence of moralizing gods with population size, because that's the very question under investigation. Assuming the answer seems like a bad idea. Sometimes all that statistics can do for us is confirm that we'll just have to gather more evidence. Here that means doing research to replace NA values with observations.

But if we were going to try to impute the missing values, there is another obstacle. The `moralizing_gods` variable is discrete. It can take the values of zero or one only. Whether imputing or dealing with measurement error, discrete variables are computationally trickier than continuous variables. The next section shows you how to handle them.

Rethinking: Present details about missing data. The moralizing gods example contains a lot of missing data—60% of the primary exposure variable is NA. Obviously in cases like this one, it is very important to inform readers about missing data and carefully justify how they were handled. But even in more routine contexts, with more modest amounts of missing data, clear documentation of missing data and its treatment is necessary. This is best done with a causal model that makes transparent what is being assumed about the source of missing values and simultaneously justifies how they are handled. But the minimum is to report the counts of missing values in each variable and what was done with them.

15.3. Categorical errors and discrete absences

The examples above focused on nice continuous variables. In the section on measurement error, the variables were continuous. In the section on missing data, neocortex percent is continuous. When a variable is continuous, you can just assign a parameter to each unknown value—whether it is measured with error or rather completely missing—and let the Markov chain do the hard part.

But when a variable is instead discrete—0/1 or 1,2,3,4 for example—then the Markov chain needs some extra tutoring. Discrete unobserved variables require discrete parameters. There are two issues with discrete parameters. First, a discrete variable will not produce a smooth surface for Hamiltonian Monte Carlo to glide around on. HMC just doesn't do discrete variables. Second, other estimation approaches also have problems with discrete parameter spaces, because discrete jumps are difficult to calibrate. Chains tend to get stuck for long periods.

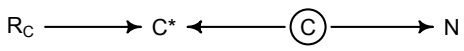
But that doesn't mean we are stuck. In almost every case, we don't need to sample discrete parameters at all. Instead we can use a special technique, known to experts as a

"weighted average," to remove discrete parameters from the model. After sampling the other parameters, we can then use their samples to compute the posterior distribution of any discrete parameter that we removed. So no information is given up. And removing the discrete parameters actually makes the Markov chain more efficient, whatever engine you are using, so it is usually worth doing, even if you aren't using HMC. The technique can even be useful when the parameters aren't discrete, because removing continuous parameters also speeds up the chain.

This all sounds too good to be true. It is all true. But implementing it is not at all obvious. In this section, I'll teach you how to do it, using the simplest example possible. The key idea, whatever the context, is that whether a variable is observed (data) or not (parameter), the generative model defines its information. There is a little bit of mathematics in this section, but no more than you learned in secondary school. Once you grasp the general approach, you can apply it to discrete variables that are not binary, including count and categorical variables.

15.3.1. Discrete cats. Imagine a neighborhood in which every house contains a songbird. Suppose we survey the neighborhood and sample one minute of song from each house, recording the number of notes. You notice that some houses also have house cats, and wonder if the presence of a cat changes the amount that each bird sings. So you try to also figure out which houses have cats. You can do this easily in some cases, either by seeing the cat or by asking a human resident. But in about 20% of houses, you can't determine whether or not a cat lives there.

This very silly example sets us a very practical working example of how to cope with discrete missing data. We will translate this story into a generative model, simulate data from it, and then build a statistical model that copes with the missing values. Let's consider the story above first as a DAG:



The presence/absence of a cat C influences the number of sung notes N. Because of missing values R_C however, we only observe C^*. To make this into a fully generative model, we must now pick functions for each arrow above. Here are my choices, in statistical notation:

$$N_i \sim \text{Poisson}(\lambda_i) \quad \text{[Probability of notes sung]}$$
$$\log \lambda_i = \alpha + \beta C_i \quad \text{[Rate of notes as function of cat]}$$
$$C_i \sim \text{Bernoulli}(k) \quad \text{[Probability cat is present]}$$
$$R_{C,i} \sim \text{Bernoulli}(r) \quad \text{[Probability of not knowing } C_i\text{]}$$

And then to actually simulate some demonstration data, we'll have to pick values for α, β, k, and r. Here's a working simulation.

R code 15.29

```
set.seed(9)
N_houses <- 100L
alpha <- 5
beta <- (-3)
k <- 0.5
r <- 0.2
```

```
cat <- rbern( N_houses , k )
notes <- rpois( N_houses , alpha + beta*cat )
R_C <- rbern( N_houses , r )
cat_obs <- cat
cat_obs[R_C==1] <- (-9L)
dat <- list(
    notes = notes,
    cat = cat_obs,
    RC = R_C,
    N = as.integer(N_houses) )
```

At the end, I've replaced each unknown value of `cat_obs` with -9. There is nothing special about this value. The model will skip them. But it is usually good to use some invalid value, so that if you make a mistake in coding, an error will result. In this case, since `cat` has a Bernoulli distribution, if the model ever asks for the probability of observing -9, there should be an error, because -9 is impossible.

To program this model, we cannot declare a parameter for each unobserved cat. So instead we'll just average over our uncertainty in whether the cat was there or not. What this means, precisely, is that the likelihood of observing N_i notes, unconditional on C_i, is:

$$\begin{aligned}\Pr(N_i) &= (\text{probability of a cat})(\text{probability of } N_i \text{ when there is a cat}) \\ &\quad + (\text{probability of no cat})(\text{probability of } N_i \text{ when there is no cat}) \\ \Pr(N_i) &= \Pr(C_i = 1)\Pr(N_i|C_i = 1) + \Pr(C_i = 0)\Pr(N_i|C_i = 0)\end{aligned}$$

When we don't know C_i, we compute the likelihood of N_i for each possible value of C_i—here one or zero—and then average these likelihoods using the probabilities that C_i takes on each value. The above expression is what we need to code into the model. We can do this either by using Stan directly or by using `custom` distribution in `ulam()`. Let me show you the `ulam()` code. Then I'll explain it.

R code 15.30

```
m15.6 <- ulam(
    alist(
        # singing bird model
        ## cat known present/absent:
        notes|RC==0 ~ poisson( lambda ),
        log(lambda) <- a + b*cat,
        ## cat NA:
        notes|RC==1 ~ custom( log_sum_exp(
                log(k) + poisson_lpmf( notes | exp(a + b) ),
                log(1-k) + poisson_lpmf( notes | exp(a) )
            ) ),

        # priors
        a ~ normal(0,1),
        b ~ normal(0,0.5),

        # sneaking cat model
        cat|RC==0 ~ bernoulli(k),
```

```
        k ~ beta(2,2)
    ), data=dat , chains=4 , cores=4 )
```

The likelihood of `notes` at the top is split into two cases. You can read `notes|RC==0` as "the probability of N when $R_C = 0$." So the first line in the model code above is just the ordinary Poisson probability when the cat is known present or absent ($R_C = 0$). The next lines are the average likelihood, when we haven't observed the presence or absence of the cat, when $R_C = 1$. It looks complicated, but it is just the previous expression on the log scale. The term `log(k) + poisson_lpmf( notes | exp(a + b) )` is $\log(\Pr(C_i = 1)\Pr(N_i|C_i = 1))$, and `log(1-k) + poisson_lpmf( notes | exp(a) )` is $\log(\Pr(C_i = 0)\Pr(N_i|C_i = 0))$. These two terms are then combined to make the weighted sum, on the log scale, using the helper function `log_sum_exp`. This function just takes a vector of log-probabilities, exponentiates them, sums them, and then returns the log of the sum. But it does all of this in a numerically stable way.

The rest of the model above is more familiar. Be sure to note however the cat presence/absence model at the bottom. When the cat is known present or absent, $R_C = 0$, we want to use that observation to update the parameter k, the probability a cat is present. This is the same k in the likelihood. This means that the non-missing observations inform the prior k for the missing observations. Take a look at the posterior of `m15.6` and verify that it mixes well and produces results that are consistent with the data generating process.

Now suppose we want to infer the unknown C values. To compute the probability that any particular cat was present or absent, we can refer back to the generative model. The thing we want to know is $\Pr(C_i = 1)$. Prior to seeing the data, this is just the prior $\Pr(C_i = 1) = k$. Once we observe N_i, the number of notes sung, we can update this prior with Bayes' rule. In this case:

$$\Pr(C_i = 1|N_i) = \frac{\Pr(N_i|C_i = 1)\Pr(C_i = 1)}{\Pr(N_i|C_i = 1)\Pr(C_i = 1) + \Pr(N_i|C_i = 0)\Pr(C_i = 0)}$$

This looks like a mess. But really it is just a definition. The top is the probability of N_i notes when $C_i = 1$. The bottom is just the average probability of N_i notes. There are just two terms to calculate, and we actually already used them in our model. The denominator in the expression above is the same average probability of N_i that we wrote into the model code.

To compute $\Pr(C_i = 1|N_i)$ for each i, we just need a few extra lines in the model code. We'll perform these calculations in Stan's **GENERATED QUANTITIES** block, which means the calculations are performed only once per HMC transition and are saved in the returned samples. When using `ulam`, we can tag a line with `gq>` to indicate this is what we want. Here is the updated model, with the new lines at the bottom:

R code 15.31

```
m15.7 <- ulam(
    alist(
        # singing bird model
        notes|RC==0 ~ poisson( lambda ),
        notes|RC==1 ~ custom( log_sum_exp(
                log(k) + poisson_lpmf( notes | exp(a + b) ),
                log(1-k) + poisson_lpmf( notes | exp(a) )
            ) ),
        log(lambda) <- a + b*cat,
```

```
        a ~ normal(0,1),
        b ~ normal(0,0.5),

        # sneaking cat model
        cat|RC==0 ~ bernoulli(k),
        k ~ beta(2,2),

        # imputed values
        gq> vector[N]:PrC1 <- exp(lpC1)/(exp(lpC1)+exp(lpC0)),
        gq> vector[N]:lpC1 <- log(k) + poisson_lpmf( notes[i] | exp(a+b) ),
        gq> vector[N]:lpC0 <- log(1-k) + poisson_lpmf( notes[i] | exp(a) )
    ), data=dat , chains=4 , cores=4 )
```

Those three lines that begin with `gq>` perform the calculations for $\Pr(C_i = 1|N_i)$. The first one defines a vector to hold the probabilities, and the formula is just the mathematical expression from before, Bayes rule. The `exp` stuff is necessary because we do the other calculations on the log scale, as always. The next two lines are just the same likelihood calculations as before, the likelihoods of N_i conditional on the cat being present (`lpC1`) or absent (`lpC0`).

In the practice problems at the end, I'll ask you to compare the posterior probabilities in `PrC1` to the true values from the simulation. You can process these samples just like any other parameter, even though we computed them in an unusual way.

The strategy presented here extrapolates to discrete variables with more than two possible values. In that case, you just need more than two terms in your average likelihood. For example, if houses can have up to two cats, then cats might be instead binomially distributed across houses. Then the code for the likelihood might be instead:

```
notes|RC==1 ~ custom( log_sum_exp(
        binomial_lpmf(2|2,k) + poisson_lpmf( notes | exp(a + b*2) ),
        binomial_lpmf(1|2,k) + poisson_lpmf( notes | exp(a + b*1) ),
        binomial_lpmf(0|2,k) + poisson_lpmf( notes | exp(a + b*0) )
    ) )
```

Read each line above as the log probability of a specific number of cats, assuming cats are binomially distributed with maximum 2 and probability k, plus the log probability of a certain number of notes, assuming that specific number of cats. Unordered categories work the same way, but the leading terms would be from some simplex of probabilities.

The same approach also works when you have more than one discrete variable with missing values. In that case, you need a different average likelihood (`custom()` distribution) for each combination of missing values. For example, suppose we also classify each house i by whether or not a dog (D_i) lives there. So a house can have one of four possible observed combinations: (1) a cat and a dog, (2) a cat, (3) a dog, (4) neither a cat nor a dog (sad). Again for some fraction of houses, we were unable to learn whether or not they have a dog. Now in the data, a house can have either or both the cat variable and the dog variable `NA`. If both are `NA`, then we must average over all four possibilities listed above, with terms for both the

prior probability of a cat and a dog, like this:

$$\begin{aligned}\Pr(N_i) = &\Pr(C_i = 1)\Pr(D_i = 1)\Pr(N_i|C_i = 1, D_i = 1) \\ &+ \Pr(C_i = 1)\Pr(D_i = 0)\Pr(N_i|C_i = 1, D_i = 0) \\ &+ \Pr(C_i = 0)\Pr(D_i = 1)\Pr(N_i|C_i = 0, D_i = 1) \\ &+ \Pr(C_i = 0)\Pr(D_i = 0)\Pr(N_i|C_i = 0, D_i = 0)\end{aligned}$$

If only the cat is `NA` and the dog is known present ($D_i = 1$), then we only have to average over possibilities (1) and (3), like this:

$$\Pr(N_i) = \Pr(C_i = 1)\Pr(N_i|C_i = 1, D_i = 1) + \Pr(C_i = 0)\Pr(N_i|C_i = 0, D_i = 1)$$

If only the dog is `NA` and the cat is known absent ($C_i = 0$), we average over possibilities (3) and (4), like this:

$$\Pr(N_i) = \Pr(D_i = 1)\Pr(N_i|C_i = 0, D_i = 1) + \Pr(D_i = 0)\Pr(N_i|C_i = 0, D_i = 0)$$

In principle, this is algorithmic and easy. In practice, it makes for complicated code, because you have to account all combinations of missingness and assign each a different average likelihood.

We'll see this general technique again in the next chapter, where we'll encounter a **STATE SPACE MODEL**. State space models can have a large number of discrete (or continuous) unobserved variables. Typical we don't write out each possibility in the code, but instead use an algorithm to work over all of the possibilities and compute the necessary average likelihood. For example, in a **HIDDEN MARKOV MODEL**, an algorithm known as the **FORWARD ALGORITHM** is used to do the averaging. The Stan user manual provides an example.

15.3.2. Discrete error. The example above concerned missing data. But when the data are measured instead with error, the procedure is very similar. Suppose for example that in the example above each house is assigned a probability of a cat being present. Call this probability k_i. When we are sure there is a cat there, $k_i = 1$. When we are sure there is no cat, $k_i = 0$. When we think it is a coin flip, $k_i = 0.5$. These k_i values replace the parameter k in the previous model, becoming the weights for averaging over our uncertainty.

15.4. Summary

This chapter has been a quick introduction to the design and implementation of measurement error and missing data models. Measurement error and missing data have causes. Incorporating those causes into the generative model helps us decide how error and missingness impact inference as well as how to design a statistical procedure. This chapter highlights the general principles of the book, that effective statistical modeling requires both careful thought about how the data were generated and delicate attention to numerical algorithms. Neither can lift inference alone.

15.5. Practice

Easy.

15E1. Rewrite the Oceanic tools model (from Chapter 11) below so that it assumes measured error on the log population sizes of each society. You don't need to fit the model to data. Just modify the mathematical formula below.

$$\begin{aligned} T_i &\sim \text{Poisson}(\mu_i) \\ \log \mu_i &= \alpha + \beta \log P_i \\ \alpha &\sim \text{Normal}(0, 1.5) \\ \beta &\sim \text{Normal}(0, 1) \end{aligned}$$

15E2. Rewrite the same model so that it allows imputation of missing values for log population. There aren't any missing values in the variable, but you can still write down a model formula that would imply imputation, if any values were missing.

Medium.

15M1. Using the mathematical form of the imputation model in the chapter, explain what is being assumed about how the missing values were generated.

15M2. In earlier chapters, we threw away cases from the primate milk data, so we could use the neocortex variable. Now repeat the WAIC model comparison example from Chapter 6, but use imputation on the neocortex variable so that you can include all of the cases in the original data. The simplest form of imputation is acceptable. How are the model comparison results affected by being able to include all of the cases?

15M3. Repeat the divorce data measurement error models, but this time double the standard errors. Can you explain how doubling the standard errors impacts inference?

Hard.

15H1. The data in `data(elephants)` are counts of matings observed for bull elephants of differing ages. There is a strong positive relationship between age and matings. However, age is not always assessed accurately. First, fit a Poisson model predicting `MATINGS` with `AGE` as a predictor. Second, assume that the observed `AGE` values are uncertain and have a standard error of ± 5 years. Re-estimate the relationship between `MATINGS` and `AGE`, incorporating this measurement error. Compare the inferences of the two models.

15H2. Repeat the model fitting problem above, now increasing the assumed standard error on `AGE`. How large does the standard error have to get before the posterior mean for the coefficient on `AGE` reaches zero?

15H3. The fact that information flows in all directions among parameters sometimes leads to rather unintuitive conclusions. Here's an example from missing data imputation, in which imputation of a single datum reverses the direction of an inferred relationship. Use these data:

R code 15.32

```
set.seed(100)
x <- c( rnorm(10) , NA )
y <- c( rnorm(10,x) , 100 )
d <- list(x=x,y=y)
```