14 Adventures in Covariance

Recall the coffee robot from the introduction to the previous chapter (page 413). This robot is programmed to move among cafés, order coffee, and record the waiting time. The previous chapter focused on the fact that the robot learns more efficiently when it pools information among the cafés. Varying intercepts are a mechanism for achieving that pooling.

Now suppose that the robot also records the time of day, morning or afternoon. The average wait time in the morning tends to be longer than the average wait time in the afternoon. This is because cafés are busier in the morning. But just like cafés vary in their *average* wait times, they also vary in their *differences* between morning and afternoon. In conventional regression, these differences in wait time between morning and afternoon are slopes, since they express the change in expectation when an indictor (or *dummy*, page 157) variable for time of day changes value. The linear model might look like this:

$$\mu_i = \alpha_{\text{CAFÉ}[i]} + \beta_{\text{CAFÉ}[i]} A_i$$

where A_i is a 0/1 indicator for *afternoon* and $\beta_{\text{CAFÉ}[i]}$ is a parameter for the expected difference between afternoon and morning for each café.

Since the robot more efficiently learns about the intercepts, $\alpha_{\text{CAFÉ}[i]}$ above, when it pools information about intercepts, it likewise learns more efficiently about the slopes when it also pools information about slopes. And the pooling is achieved in the same way, by estimating the population distribution of slopes at the same time the robot estimates each slope. The distributions assigned to both intercepts and slopes enable pooling for both, as the model (robot) learns the prior from the data.

This is the essence of the general **VARYING EFFECTS** strategy: Any batch of parameters with *exchangeable* index values can and probably should be pooled. Exchangeable just means the index values have no true ordering, because they are arbitrary labels. There's nothing special about intercepts; slopes can also vary by unit in the data, and pooling information among them makes better use of the data. So our coffee robot should be programmed to model both the population of intercepts and the population of slopes. Then it can use pooling for both and squeeze more information out of the data.

But here's a fact that will help us to squeeze even more information out of the data: Cafés covary in their intercepts and slopes. Why? At a popular café, wait times are on average long in the morning, because staff are very busy (FIGURE 14.1). But the same café will be much less busy in the afternoon, leading to a large difference between morning and afternoon wait times. At such a popular café, the intercept is high and the slope is far from zero, because the difference between morning and afternoon waits is large. But at a less popular café, the difference will be small. Such an unpopular café makes you wait less in the morning—because

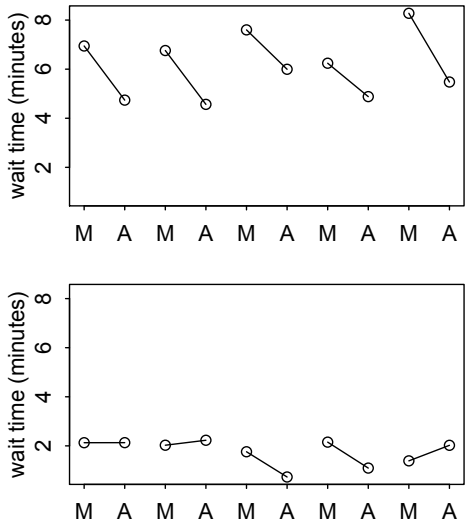

FIGURE 14.1. Waiting times at two cafés. Top: A busy café at which wait times nearly always improve in the afternoon. Bottom: An unpopular café where wait times are nearly always short. In a population of cafés like these, long morning waits (intercepts) covary with larger differences between morning and afternoon (slopes).

it's not busy—but there isn't much improvement in the afternoon. In the entire population of cafés, including both the popular and the unpopular, intercepts and slopes covary.

This covariation is information that the robot can use. If we can figure out a way to pool information *across* parameter types—intercepts and slopes—what the robot learns in the morning can improve learning about afternoons, and vice versa. Suppose for example that the robot arrives at a new café in the morning. It observes a long wait for its coffee. Even before it orders a coffee at the same café in the afternoon, it can update its expectation for how long it will wait. In the population of cafés, a long wait in the morning is associated with a shorter wait in the afternoon.

In this chapter, you'll see how to really do this, to specify **VARYING SLOPES** in combination with the varying intercepts of the previous chapter. This will enable pooling that will improve estimates of how different units respond to or are influenced by predictor variables. It will also improve estimates of intercepts, by borrowing information across parameter types. Essentially, varying slopes models are massive interaction machines. They allow every unit in the data to have its own unique response to any treatment or exposure or event, while also improving estimates via pooling. When the variation in slopes is large, the average slope is of less interest. Sometimes, the pattern of variation in slopes provides hints about omitted variables that explain why some units respond more or less. We'll see an example in this chapter.

The machinery that makes such complex varying effects possible will be used later in the chapter to extend the varying effects strategy to more subtle model types, including the use of continuous categories, using **GAUSSIAN PROCESSES**. Ordinary varying effects work only with discrete, unordered categories, such as individuals, countries, or ponds. In these cases, each category is equally different from all of the others. But it is possible to use pooling with categories such as age or location. In these cases, some ages and some locations are more similar than others. You'll see how to model covariation among continuous categories of this kind, as well as how to generalize the strategy to seemingly unrelated types of models

such as phylogenetic and network regressions. Finally, we'll circle back to causal inference and use our new powers over covariance to go beyond the tools of Chapter 6), introducing INSTRUMENTAL VARIABLES. Instruments are ways of inferring cause without closing back-door paths. However are very tricky both in design and estimation.

The material in this chapter is difficult. So if it suddenly seems both conceptually and computationally much more difficult, that only means you are paying attention. Material like this requires repetition, discussion, and learning from mistakes. The struggle is definitely worth it. You don't have to understand it all at once.

14.1. Varying slopes by construction

How should the robot pool information across intercepts and slopes? By modeling the joint population of intercepts and slopes, which means by modeling their covariance. In conventional multilevel models, the device that makes this possible is a joint multivariate Gaussian distribution for all of the varying effects, both intercepts and slopes. So instead of having two independent Gaussian distributions of intercepts and of slopes, the robot can do better by assigning a two-dimensional Gaussian distribution to both the intercepts (first dimension) and the slopes (second dimension).

You've been working with multivariate Gaussian distributions ever since Chapter 4, when you began using the quadratic approximation for the posterior distribution. The variance-covariance matrix, `vcov`, for a fit model describes how each parameter's posterior probability is associated with each other parameter's posterior probability. Now we'll use the same kind of distribution to describe the variation within and covariation among different kinds of varying effects. Varying intercepts have variation, and varying slopes have variation. Intercepts and slopes covary.

In order to see how this works and how varying slopes are specified and interpreted, let's simulate the coffee robot from the introduction. Like previous simulation exercises, this will simultaneously help you see how to conduct your own prospective power analyses, in addition to reemphasizing the generative nature of Bayesian statistical models.

Rethinking: Why Gaussian? There is no reason the multivariate distribution of intercepts and slopes must be Gaussian. But there are both practical and epistemological justifications. On the practical side, there aren't many multivariate distributions that are easy to work with. The only common ones are multivariate Gaussian and multivariate Student-t distributions. On the epistemological side, if all we want to say about these intercepts and slopes is their means, variances, and covariances, then the maximum entropy distribution is multivariate Gaussian. But thin Gaussian tails can still be risky.

14.1.1. Simulate the population.

Begin by defining the population of cafés that the robot might visit. This means we'll define the average wait time in the morning and the afternoon, as well as the correlation between them. These numbers are sufficient to define the *average* properties of the cafés. Let's define these properties, then we'll sample cafés from them.

R code 14.1

```
a <- 3.5            # average morning wait time
b <- (-1)           # average difference afternoon wait time
sigma_a <- 1        # std dev in intercepts
sigma_b <- 0.5      # std dev in slopes
rho <- (-0.7)       # correlation between intercepts and slopes
```

These values define the entire population of cafés. To use these values to simulate a sample of cafés for the robot, we'll need to build them into a 2-dimensional multivariate Gaussian distribution. This means we need a vector of two means and 2-by-2 matrix of variances and covariances. The means are easiest. The vector we need is just:

R code 14.2
```
Mu <- c( a , b )
```

That's it. The value in `a` is the mean intercept, the wait in the morning. And the value in `b` is the mean slope, the difference in wait between afternoon and morning.

The matrix of variances and covariances is arranged like this:

$$\begin{pmatrix} \text{variance of intercepts} & \text{covariance of intercepts \& slopes} \\ \text{covariance of intercepts \& slopes} & \text{variance of slopes} \end{pmatrix}$$

And now in mathematical form:

$$\begin{pmatrix} \sigma_\alpha^2 & \sigma_\alpha\sigma_\beta\rho \\ \sigma_\alpha\sigma_\beta\rho & \sigma_\beta^2 \end{pmatrix}$$

The variance in intercepts is σ_α^2, and the variance in slopes is σ_β^2. These are found along the *diagonal* of the matrix. The other two elements of the matrix are the same, $\sigma_\alpha\sigma_\beta\rho$. This is the covariance between intercepts and slopes. It's just the product of the two standard deviations and the correlation. It might help to imagine an ordinary variance as the covariance of a variable with itself. If you are rusty on the definition of a covariance—it's okay, most people are—then see the Overthinking box further down.

To build this matrix with R code, there are several options. I'll show you two, both very common. The first is to just use `matrix` to build the entire covariance matrix directly:

R code 14.3
```
cov_ab <- sigma_a*sigma_b*rho
Sigma <- matrix( c(sigma_a^2,cov_ab,cov_ab,sigma_b^2) , ncol=2 )
```

The awkward thing is that R matrices defined this way fill down each column before moving to the next row over. So the order inside the code above looks odd, but works. To see what I mean by "fill down each column," try this:

R code 14.4
```
matrix( c(1,2,3,4) , nrow=2 , ncol=2 )
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

The first column filled, and then R started over at the top of the second column.

The other common way to build the covariance matrix is conceptually very useful, because it treats the standard deviations and correlations separately. Then it matrix multiplies them to produce the covariance matrix. We're going to use this approach later on, to define priors, so it's worth seeing it now. Here's how it's done:

R code 14.5
```
sigmas <- c(sigma_a,sigma_b) # standard deviations
Rho <- matrix( c(1,rho,rho,1) , nrow=2 ) # correlation matrix
```

```
# now matrix multiply to get covariance matrix
Sigma <- diag(sigmas) %*% Rho %*% diag(sigmas)
```

If you are not sure what `diag(sigmas)` accomplishes, then try typing just `diag(sigmas)` at the R prompt.

Now we're ready to simulate some cafés, each with its own intercept and slope. Let's define the number of cafés:

```
N_cafes <- 20
```

R code 14.6

And to simulate their properties, we just sample randomly from the multivariate Gaussian distribution defined by `Mu` and `Sigma`:

```
library(MASS)
set.seed(5) # used to replicate example
vary_effects <- mvrnorm( N_cafes , Mu , Sigma )
```

R code 14.7

Note the `set.seed(5)` line above. That's there so you can replicate the precise results in the example figures. The particular number, 5, produces a particular sequence of random numbers. Each unique number generates a unique sequence. Including a `set.seed` line like this in your code allows others to exactly replicate your analyses. Later you'll want to repeat the example without repeating the `set.seed` call, or with a different number, so you can appreciate the variation across simulations.

Look at the contents of `vary_effects` now. It should be a matrix with 20 rows and 2 columns. Each row is a café. The first column contains intercepts. The second column contains slopes. For transparency, let's split these columns apart into nicely named vectors:

```
a_cafe <- vary_effects[,1]
b_cafe <- vary_effects[,2]
```

R code 14.8

To visualize these intercepts and slopes, go ahead and `plot` them against one another. This code will also show the distribution's contours:

```
plot( a_cafe , b_cafe , col=rangi2 ,
    xlab="intercepts (a_cafe)" , ylab="slopes (b_cafe)" )

# overlay population distribution
library(ellipse)
for ( l in c(0.1,0.3,0.5,0.8,0.99) )
    lines(ellipse(Sigma,centre=Mu,level=l),col=col.alpha("black",0.2))
```

R code 14.9

FIGURE 14.2 displays a typical result. In any particular simulation, the correlation may not be as obvious. But on average, the intercepts in `a_cafe` and the slopes in `b_cafe` will have a correlation of -0.7, and you'll be able to see this in the scatterplot. The contour lines in the plot, produced by the `ellipse` package (make sure you install it), display the multivariate Gaussian population of intercepts and slopes that the 20 cafés were sampled from.

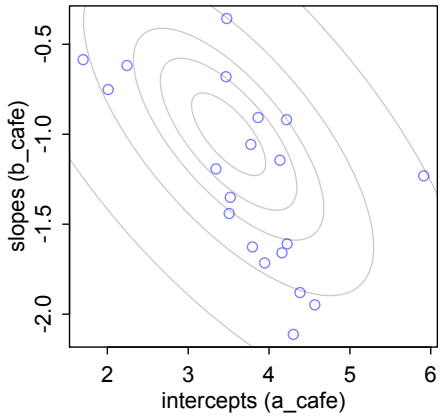

FIGURE 14.2. 20 cafés sampled from a statistical population. The horizontal axis is the intercept (average morning wait) for each cafe. The vertical axis is the slope (average difference between afternoon and morning wait) for each café. The gray ellipses illustrate the multivariate Gaussian population of intercepts and slopes.

Overthinking: Variance, covariance, correlation. In typical statistical usage, we define covariance using three parameters: (1) the standard deviation of the first variable (σ_α for example), (2) the standard deviation of the second variable (σ_β for example), and (3) the correlation between the two variables ($\rho_{\alpha\beta}$ for example). Why is the covariance equal to $\sigma_\alpha\sigma_\beta\rho_{\alpha\beta}$?

The usual definition of the covariance between two variables x and y is $\text{cov}(x,y) = \text{E}(xy) - \text{E}(x)\,\text{E}(y)$. You can say this as "the covariance is the difference between the average product and the product of the averages." The variance is just a special case of this, the covariance of a variable with itself: $\text{var}(x) = \text{cov}(x,x) = \text{E}(x^2) - \text{E}(x)^2$. If we consider only random variables with expectation zero—no harm done, since we can recenter at will—then these are just $\text{cov}(x,y) = \text{E}(xy)$ and $\text{var}(x) = \text{E}(x^2)$.

A correlation is just a rescaled covariance, so that the minimum is -1 and the maximum is 1. We can standardize a covariance this way by dividing it by the maximum possible covariance, which turns out to be $\sqrt{\text{var}(x)\,\text{var}(y)}$, the product of the standard deviations. Now to show you that this is the largest that $\text{cov}(x,y) = \text{E}(xy)$ can ever be. A covariance will be largest when the second variable y is just a rescaled copy of x. For example, let $y_i = px_i$, where p is some proportion like 0.5 or 1.5. So $y = px$ is just a stretched x. The covariance is now $\text{cov}(x,y) = \text{E}(px^2) = p\,\text{E}(x^2)$. The variances are $\text{var}(x) = \text{E}(x^2)$ and $\text{var}(y) = \text{E}(y^2) = \text{E}(p^2x^2) = p^2\,\text{E}(x^2)$. Having fun yet? Here comes the end. $\text{var}(x)\,\text{var}(y) = p^2\,\text{E}(x^2)^2$ and so $\sqrt{\text{var}(x)\,\text{var}(y)} = p\,\text{E}(x^2) = \text{cov}(x,y)$. That's the largest the covariance can get. So if we want a standardized measure of associate, the correlation, we divide the covariance by this maximum value, which gives us the usual definition of a correlation coefficient, $\rho_{xy} = \text{cov}(x,y)/\sqrt{\text{var}(x)\,\text{var}(y)}$. Solve this equation for $\text{cov}(x,y)$ and you get $\text{cov}(x,y) = \sqrt{\text{var}(x)\,\text{var}(y)}\rho_{xy}$. Whew. All of this is just to show that the applied statistics usage of covariance as $\text{cov}(x,y) = \sigma_x\sigma_y\rho_{xy}$ is as justified as it is convenient.

14.1.2. Simulate observations. We're almost done simulating. What we did above was simulate individual cafés and their average properties. Now all that remains is to simulate our robot visiting these cafés and collecting data. The code below simulates 10 visits to each café, 5 in the morning and 5 in the afternoon. The robot records the wait time during each visit. Then it combines all of the visits into a common data frame.

R code 14.10

```
set.seed(22)
N_visits <- 10
afternoon <- rep(0:1,N_visits*N_cafes/2)
cafe_id <- rep( 1:N_cafes , each=N_visits )
mu <- a_cafe[cafe_id] + b_cafe[cafe_id]*afternoon
sigma <- 0.5  # std dev within cafes
wait <- rnorm( N_visits*N_cafes , mu , sigma )
d <- data.frame( cafe=cafe_id , afternoon=afternoon , wait=wait )
```

Go ahead and look inside the data frame `d` now. You'll find exactly the sort of data that is well-suited to a varying slopes model. There are multiple *clusters* in the data. These are the cafés. And each cluster is observed under different conditions. So it's possible to estimate both an individual intercept for each cluster, as well as an individual slope.

In this example, everything is *balanced*: Each café has been observed exactly 10 times, and the time of day is always balanced as well, with 5 morning and 5 afternoon observations for each café. But in general the data do not need to be balanced. Just like the tadpoles example from the previous chapter, lack of balance can really favor the varying effects analysis, because partial pooling uses information about the population where it is needed most.

Rethinking: Simulation and misspecification. In this exercise, we are simulating data from a generative process and then analyzing that data with a model that reflects exactly the correct structure of that process. But in the real world, we're never so lucky. Instead we are always forced to analyze data with a model that is MISSPECIFIED: The true data-generating process is different than the model. Simulation can be used however to explore misspecification. Just simulate data from a process and then see how a number of models, none of which match exactly the data-generating process, perform. And always remember that Bayesian inference does not depend upon data-generating assumptions, such as the likelihood, being true. Non-Bayesian approaches may depend upon sampling distributions for their inferences, but this is not the case for a Bayesian model. In a Bayesian model, a likelihood is a prior for the data, and inference about parameters can be surprisingly insensitive to its details.

14.1.3. The varying slopes model. Now we're ready to play the process in reverse. We just generated data from a set of 20 cafés, and those cafés were themselves generated from a statistical population of cafés. Now we'll use that data to learn about the data-generating process, through a model.

The model is much like the varying intercepts models from the previous chapter. But now the joint population of intercepts and slopes appears, instead of just a distribution of varying intercepts. This is the varying slopes model, with explanation to follow. First we have the probability of the data and the linear model:

$$W_i \sim \text{Normal}(\mu_i, \sigma) \quad \text{[likelihood]}$$
$$\mu_i = \alpha_{\text{CAFÉ}[i]} + \beta_{\text{CAFÉ}[i]} A_i \quad \text{[linear model]}$$

Then comes the matrix of varying intercepts and slopes, with it's covariance matrix:

$$\begin{bmatrix} \alpha_{\text{CAFÉ}} \\ \beta_{\text{CAFÉ}} \end{bmatrix} \sim \text{MVNormal}\left(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S} \right) \quad \text{[population of varying effects]}$$
$$\mathbf{S} = \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \quad \text{[construct covariance matrix]}$$

These lines state that each café has an intercept $\alpha_{\text{CAFÉ}}$ and slope $\beta_{\text{CAFÉ}}$ with a prior distribution defined by the two-dimensional Gaussian distribution with means α and β and covariance matrix $\mathbf{S}$. This statement of prior will adaptively regularize the individual intercepts, slopes, and the correlation among them. The second line above defines how we're constructing the covariance matrix $\mathbf{S}$, by factoring it into separate standard deviations, σ_α and σ_β, and a correlation matrix $\mathbf{R}$. There are other ways to go about this, but by splitting the covariance up into standard deviations and correlations, it'll be easier to later understand the inferred structure of the varying effects.

And then come the hyper-priors, the priors that define the adaptive varying effects prior:

$$\begin{aligned}
\alpha &\sim \text{Normal}(5, 2) && \text{[prior for average intercept]}\\
\beta &\sim \text{Normal}(-1, 0.5) && \text{[prior for average slope]}\\
\sigma &\sim \text{Exponential}(1) && \text{[prior stddev within cafés]}\\
\sigma_\alpha &\sim \text{Exponential}(1) && \text{[prior stddev among intercepts]}\\
\sigma_\beta &\sim \text{Exponential}(1) && \text{[prior stddev among slopes]}\\
\mathbf{R} &\sim \text{LKJcorr}(2) && \text{[prior for correlation matrix]}
\end{aligned}$$

The final line probably looks unfamiliar. The correlation matrix $\mathbf{R}$ needs a prior. It isn't easy to conceptualize what a distribution of matrices means. But in this introductory case, it isn't so hard. This particular correlation matrix is only 2-by-2 in size. So it looks like this:

$$\mathbf{R} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

where ρ is the correlation between intercepts and slopes. So there's just one parameter to define a prior for. In larger matrices, with additional varying slopes, it gets more complicated.

So whatever is the LKJcorr distribution? What LKJcorr(2) does is define a weakly informative prior on ρ that is skeptical of extreme correlations near -1 or 1.[194] You can think of it as a regularizing prior for correlation matrices. This distribution has a single parameter, η, that controls how skeptical the prior is of large correlations in the matrix. When we use LKJcorr(1), the prior is flat over all valid correlation matrices. When the value is greater than 1, such as the 2 we used above, then extreme correlations are less likely. To visualize this family of priors, it will help to sample random matrices from it:

R code 14.11

```
R <- rlkjcorr( 1e4 , K=2 , eta=2 )
dens( R[,1,2] , xlab="correlation" )
```

This is shown in FIGURE 14.3, along with two other η values. When the matrix is larger, there are more correlations inside it, but the nature of the distribution remains the same. There is an example density for a 3-by-3 matrix in the help page examples, `?rlkjcorr`.

To fit the model, we use a list of formulas that closely mirrors the model definition above. Note the use of `c()` to combine parameters into a vector.

R code 14.12

```
set.seed(867530)
m14.1 <- ulam(
    alist(
        wait ~ normal( mu , sigma ),
        mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
```

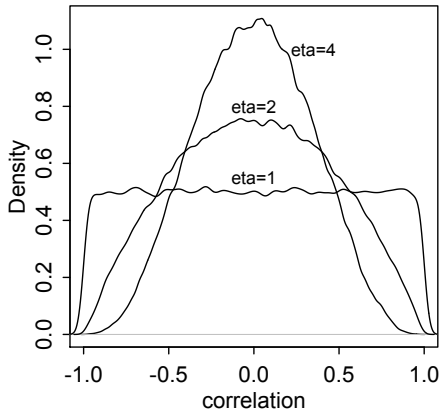

FIGURE 14.3. LKJcorr(η) probability density. The plot shows the distribution of correlation coefficients extracted from random 2-by-2 correlation matrices, for three values of η. When $\eta = 1$, all correlations are equally plausible. As η increases, extreme correlations become less plausible.

```
    c(a_cafe,b_cafe)[cafe] ~ multi_normal( c(a,b) , Rho , sigma_cafe ),
    a ~ normal(5,2),
    b ~ normal(-1,0.5),
    sigma_cafe ~ exponential(1),
    sigma ~ exponential(1),
    Rho ~ lkj_corr(2)
) , data=d , chains=4 , cores=4 )
```

The distribution `multi_normal` is a multivariate Gaussian notation that takes a vector of means, `c(a,b)`, a correlation matrix, `Rho`, and a vector of standard deviations, `sigma_cafe`. It constructs the covariance matrix internally. If you are interested in the details, you can peek at the raw Stan code with `stancode(m14.1)`. The name `multi_normal` is what Stan uses in its raw code. The similar R functions are `dmvnorm` and `dmvnorm2`.

Now instead of looking at the marginal posterior distributions in the `precis` output, let's go straight to inspecting the posterior distribution of varying effects. First, let's examine the posterior correlation between intercepts and slopes.

R code 14.13

```
post <- extract.samples(m14.1)
dens( post$Rho[,1,2] , xlim=c(-1,1) ) # posterior
R <- rlkjcorr( 1e4 , K=2 , eta=2 )    # prior
dens( R[,1,2] , add=TRUE , lty=2 )
```

The result is shown in FIGURE 14.4, with some additional decoration and the addition of the prior for comparison. The blue density is the posterior distribution of the correlation between intercepts and slopes. The posterior is concentrated on negative values, because the model has learned the negative correlation you can see in FIGURE 14.2. Keep in mind that the model did not get to see the true intercepts and slopes. All it had to work from was the observed wait times in morning and afternoon.

If you are curious about the impact of the prior, then you should change the prior and repeat the analysis. I suggest trying a flat prior, LKJcorr(1), and then a more strongly regularizing prior like LKJcorr(4) or LKJcorr(5).

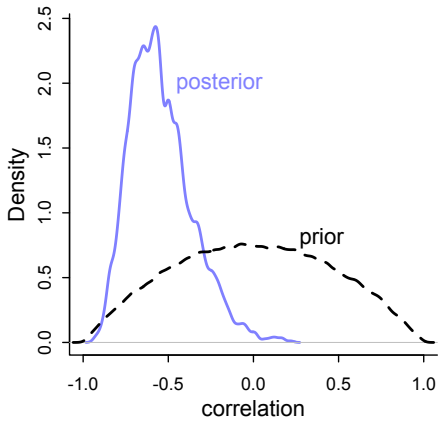

FIGURE 14.4. Posterior distribution of the correlation between intercepts and slopes. Blue: Posterior distribution of the correlation, reliably below zero. Dashed: Prior distribution, the LKJcorr(2) density.

Next, consider the shrinkage. The multilevel model estimates posterior distributions for intercepts and slopes of each café. The inferred correlation between these varying effects was used to pool information across them. This is just as the inferred variation among intercepts pools information among them, as well as how the inferred variation among slopes pools information among them. All together, the variances and correlation define an inferred multivariate Gaussian prior for the varying effects. And this prior, learned from the data, adaptively regularizes both the intercepts and slopes.

To see the consequence of this adaptive regularization, shrinkage, let's plot the posterior mean varying effects. Then we can compare them to raw, unpooled estimates. We'll also show the contours of the inferred prior—the population of intercepts and slopes—and this will help us visualize the shrinkage. Here's code to plot the unpooled estimates and posterior means.

R code 14.14

```
# compute unpooled estimates directly from data
a1 <- sapply( 1:N_cafes ,
        function(i) mean(wait[cafe_id==i & afternoon==0]) )
b1 <- sapply( 1:N_cafes ,
        function(i) mean(wait[cafe_id==i & afternoon==1]) ) - a1

# extract posterior means of partially pooled estimates
post <- extract.samples(m14.1)
a2 <- apply( post$a_cafe , 2 , mean )
b2 <- apply( post$b_cafe , 2 , mean )

# plot both and connect with lines
plot( a1 , b1 , xlab="intercept" , ylab="slope" ,
    pch=16 , col=rangi2 , ylim=c( min(b1)-0.1 , max(b1)+0.1 ) ,
    xlim=c( min(a1)-0.1 , max(a1)+0.1 ) )
points( a2 , b2 , pch=1 )
for ( i in 1:N_cafes ) lines( c(a1[i],a2[i]) , c(b1[i],b2[i]) )
```

And to superimpose the contours of the population:

R code 14.15

```
# compute posterior mean bivariate Gaussian
Mu_est <- c( mean(post$a) , mean(post$b) )
rho_est <- mean( post$Rho[,1,2] )
sa_est <- mean( post$sigma_cafe[,1] )
sb_est <- mean( post$sigma_cafe[,2] )
cov_ab <- sa_est*sb_est*rho_est
Sigma_est <- matrix( c(sa_est^2,cov_ab,cov_ab,sb_est^2) , ncol=2 )

# draw contours
library(ellipse)
for ( l in c(0.1,0.3,0.5,0.8,0.99) )
    lines(ellipse(Sigma_est,centre=Mu_est,level=l),
        col=col.alpha("black",0.2))
```

The result appears on the left in FIGURE 14.5. The blue points are the unpooled estimates for each café. The open points are the posterior means from the varying effects model. A line connects the points that belong to the same café. Each open point is displaced from the blue towards the center of the contours, as a result of shrinkage in both dimensions. Blue points farther from the center experience more shrinkage, because they are less plausible, given the inferred population.

But notice too that shrinkage is not in direct lines towards the center. This is most obvious for the café that appears in the top-middle of the plot. That particular café had an average intercept, so it lies in the middle of the horizontal axis. But it also had an unusually high slope, so it lies at the top of the vertical axis. Pooled information from the other cafés results in skepticism about the slope. But since intercepts and slopes are correlated in the population as a whole, shrinking the slope down also shrinks the intercept. So all those angled shrinkage lines reflect the negative correlation between intercepts and slopes.

The right-hand plot in FIGURE 14.5 displays the same information, but now on the outcome scale. You can compute these average outcomes from knowledge of the linear model:

R code 14.16

```
# convert varying effects to waiting times
wait_morning_1 <- (a1)
wait_afternoon_1 <- (a1 + b1)
wait_morning_2 <- (a2)
wait_afternoon_2 <- (a2 + b2)

# plot both and connect with lines
plot( wait_morning_1 , wait_afternoon_1 , xlab="morning wait" ,
    ylab="afternoon wait" , pch=16 , col=rangi2 ,
    ylim=c( min(wait_afternoon_1)-0.1 , max(wait_afternoon_1)+0.1 ) ,
    xlim=c( min(wait_morning_1)-0.1 , max(wait_morning_1)+0.1 ) )
points( wait_morning_2 , wait_afternoon_2 , pch=1 )
for ( i in 1:N_cafes )
    lines( c(wait_morning_1[i],wait_morning_2[i]) ,
    c(wait_afternoon_1[i],wait_afternoon_2[i]) )
abline( a=0 , b=1 , lty=2 )
```

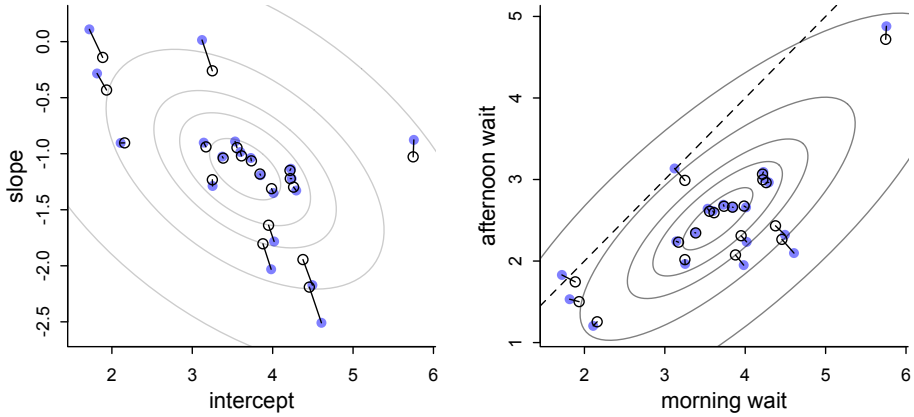

FIGURE 14.5. Shrinkage in two dimensions. Left: Raw unpooled intercepts and slopes (filled blue) compared to partially pooled posterior means (open circles). The gray contours show the inferred population of varying effects. Right: The same estimates on the outcome scale.

To add the contour, we need the variances and covariance. We could use a formula—there are some simple relations among Gaussian random variables. But to make this lesson more general, let's simulate instead, so you can see how to compute anything of interest.

R code 14.17

```
# now shrinkage distribution by simulation
v <- mvrnorm( 1e4 , Mu_est , Sigma_est )
v[,2] <- v[,1] + v[,2] # calculate afternoon wait
Sigma_est2 <- cov(v)
Mu_est2 <- Mu_est
Mu_est2[2] <- Mu_est[1]+Mu_est[2]

# draw contours
library(ellipse)
for ( l in c(0.1,0.3,0.5,0.8,0.99) )
    lines(ellipse(Sigma_est2,centre=Mu_est2,level=l),
        col=col.alpha("black",0.5))
```

The horizontal axis in the plot shows the expected morning wait, in minutes, for each café. The vertical axis shows the expected afternoon wait. Again the blue points are unpooled empirical estimates from the data. The open points are posterior predictions, using the pooled estimates. The diagonal dashed line shows where morning wait is equal to afternoon wait. What I want you to appreciate in this plot is that shrinkage on the parameter scale naturally produces shrinkage where we actually care about it: on the outcome scale. And it also implies a population of wait times, shown by the gray contours. That population is now positively correlated—cafés with longer morning waits also tend to have longer afternoon waits. They are popular, after all. But the population lies mostly below the dashed line where the waits are equal. You'll wait less in the afternoon, on average.

14.2. Advanced varying slopes

To see how to construct a model with more than two varying effects—varying intercepts plus more than one varying slope—as well as with more than one type of cluster, we'll return to the chimpanzee experiment data that was introduced in Chapter 11. In these data, there are two types of clusters: actors and blocks. We explored **CROSS-CLASSIFICATION** with two kinds of varying intercepts back on page 429. We also modeled the experiment with two different slopes: one for the effect of the prosocial option (the side of the table with two pieces of food) and one for the interaction between the prosocial option and the presence of another chimpanzee. Now we'll model both types of clusters and place varying effects on the intercepts and both slopes. All of this machinery is not always necessary. But sometimes it is, and this is a relatively simple example to lay it all out.

I'll also use this example to emphasize the importance of **NON-CENTERED PARAMETERIZATION** for some multilevel models. For any given multilevel model, there are several different ways to write it down. These ways are called "parameterizations." Mathematically, these alternative parameterizations are equivalent, but inside the MCMC engine they are not. Remember, how you fit the model is part of the model. Choosing a better parameterization is an awesome way to improve sampling for your MCMC model fit, and the non-centered parameterization tends to help a lot with complex varying effect models like the one you'll work with in this section. I'll hide the details of the technique in the main text. But as usual, there is an Overthinking box at the end that provides some detail.

Okay, let's construct a cross-classified varying slopes model. To maintain some sanity with this complicated model, we'll use more than one linear model in the formulas. This will allow us to compartmentalize sub-models for the intercepts and each slope. Here's what the likelihood and its linear model looks like:

$$\begin{aligned} L_i &\sim \text{Binomial}(1, p_i) \\ \text{logit}(p_i) &= \gamma_{\text{TID}[i]} + \alpha_{\text{ACTOR}[i],\text{TID}[i]} + \beta_{\text{BLOCK}[i],\text{TID}[i]} \end{aligned}$$

The linear model for $\text{logit}(p_i)$ contains an average log-odds for each treatment, $\gamma_{\text{TID}[i]}$, an effect for each actor in each treatment, $\alpha_{\text{ACTOR}[i],\text{TID}[i]}$, and finally an effect for each block in each treatment, $\beta_{\text{BLOCK}[i],\text{TID}[i]}$. This is essentially an interaction model that allows the effect of each treatment to vary by each actor and each block. This is to say that the average treatment effect can vary by block, and the each individual chimpanzee can also respond (across blocks) to each treatment differently. This yields a total of $4+7\times4+6\times4 = 56$ parameters. Pooling is really needed here.

So let's do some pooling. The next part of the model are the adaptive priors. Since there are two cluster types, actors and blocks, there are two multivariate Gaussian priors. The multivariate Gaussian priors are both 4-dimensional, in this example, because there are 4 treatments. But in general, you can choose to have different varying effects in different

cluster types. Here are the two priors in this case:

$$\begin{bmatrix}\alpha_{j,1}\\ \alpha_{j,2}\\ \alpha_{j,3}\\ \alpha_{j,4}\end{bmatrix} \sim \text{MVNormal}\left(\begin{bmatrix}0\\0\\0\\0\end{bmatrix}, \mathbf{S}_{\text{ACTOR}}\right)$$
$$\begin{bmatrix}\beta_{j,1}\\ \beta_{j,2}\\ \beta_{j,3}\\ \beta_{j,4}\end{bmatrix} \sim \text{MVNormal}\left(\begin{bmatrix}0\\0\\0\\0\end{bmatrix}, \mathbf{S}_{\text{BLOCK}}\right)$$

What these priors state is that actors and blocks come from two different statistical populations. Within each, the 4 features of each actor or block are related through a covariance matrix $\mathbf{S}$ specific to that population. There are no means in these priors, just because we already placed the average treatment effects—γ—in the linear model.

And the `ulam` code for this model looks as you'd expect, given previous examples. To define the multiple linear models, just write each into the formula list in order. I'll add some white space and comments to this formula list, to make it easier to read.

R code 14.18

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
d$block_id <- d$block
d$treatment <- 1L + d$prosoc_left + 2L*d$condition

dat <- list(
    L = d$pulled_left,
    tid = d$treatment,
    actor = d$actor,
    block_id = as.integer(d$block_id) )

set.seed(4387510)
m14.2 <- ulam(
    alist(
        L ~ dbinom(1,p),
        logit(p) <- g[tid] + alpha[actor,tid] + beta[block_id,tid],

        # adaptive priors
        vector[4]:alpha[actor] ~ multi_normal(0,Rho_actor,sigma_actor),
        vector[4]:beta[block_id] ~ multi_normal(0,Rho_block,sigma_block),

        # fixed priors
        g[tid] ~ dnorm(0,1),
        sigma_actor ~ dexp(1),
        Rho_actor ~ dlkjcorr(4),
        sigma_block ~ dexp(1),
        Rho_block ~ dlkjcorr(4)
    ) , data=dat , chains=4 , cores=4 )
```

When sampling from this model, you will notice many "divergent transitions":

```
Warning messages:
1: There were 154 divergent transitions after warmup.
```

We first discussed these back in Chapter 9. If you look at the diagnostics and the `trankplot`, you see that the chains are not mixing quite right. In the previous chapter, we saw how re-parameterizing the model can help. We'll do that again here. Our goal is to factor all the parameters out of the adaptive priors and place them instead in the linear model. But now that we have covariance matrixes in the priors, how are we going to do that?

The basic strategy is the same, just extrapolated to matrixes. What we'll do is again make some z-scores for each random effect. But now we need matrixes of z-scores, just let we had matrixes of random effects in the previous model. Then we'll want to multiply those z-scores into a covariance matrix so that we get back the random effects on the right scale for the linear model. There is a special matrix algebra trick for this, and `ulam` has a function `compose_noncentered` for performing this trick. The Overthinking box at the end of the section explains in more detail. This is how the non-centered version of the model looks:

R code 14.19

```
set.seed(4387510)
m14.3 <- ulam(
    alist(
        L ~ binomial(1,p),
        logit(p) <- g[tid] + alpha[actor,tid] + beta[block_id,tid],

        # adaptive priors - non-centered
        transpars> matrix[actor,4]:alpha <-
                compose_noncentered( sigma_actor , L_Rho_actor , z_actor ),
        transpars> matrix[block_id,4]:beta <-
                compose_noncentered( sigma_block , L_Rho_block , z_block ),
        matrix[4,actor]:z_actor ~ normal( 0 , 1 ),
        matrix[4,block_id]:z_block ~ normal( 0 , 1 ),

        # fixed priors
        g[tid] ~ normal(0,1),
        vector[4]:sigma_actor ~ dexp(1),
        cholesky_factor_corr[4]:L_Rho_actor ~ lkj_corr_cholesky( 2 ),
        vector[4]:sigma_block ~ dexp(1),
        cholesky_factor_corr[4]:L_Rho_block ~ lkj_corr_cholesky( 2 ),

        # compute ordinary correlation matrixes from Cholesky factors
        gq> matrix[4,4]:Rho_actor <<- Chol_to_Corr(L_Rho_actor),
        gq> matrix[4,4]:Rho_block <<- Chol_to_Corr(L_Rho_block)
    ) , data=dat , chains=4 , cores=4 , log_lik=TRUE )
```

No more divergent transitions! There are several advanced features of `ulam` on display above. One important bit to note is the last two lines. These compute the ordinary correlation matrixes from those Cholesky factors. This will help you interpret the correlations, if you want. That `gq>` tag in front of each line tells Stan to do this calculation only at the end of each transition. This is more efficient. If you are still curious about the details, see the Overthinking box further down for the raw Stan version of this model.

How has the non-centered parameterization helped here? If you compare the `precis` output of the two models, you'll see that they arrive at roughly the same inferences. But the

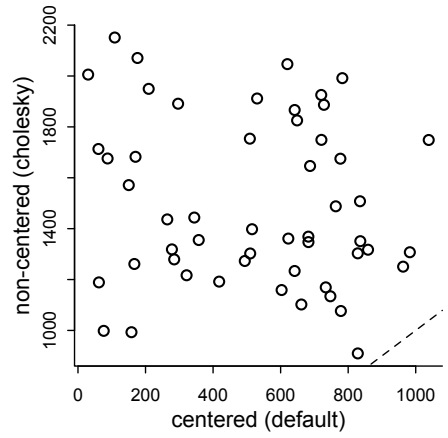

FIGURE 14.6. Distributions of effective samples, `n_eff`, for the centered and non-centered parameterizations of the cross-classified varying slopes model, `m14.2` and `m14.3`, respectively. Both models arrive at equivalent inferences, but the non-centered version samples much more efficiently.

`n_eff` values for `m14.2` are much larger, and it sampled more quickly in real time. Let's show the difference in effective samples visually, using a simple scatterplot:

R code 14.20

```
# extract n_eff values for each model
neff_nc <- precis(m14.3,3,pars=c("alpha","beta"))$n_eff
neff_c <- precis(m14.2,3,pars=c("alpha","beta"))$n_eff
plot( neff_c , neff_nc , xlab="centered (default)" ,
    ylab="non-centered (cholesky)" , lwd=1.5 )
abline(a=0,b=1,lty=2)
```

FIGURE 14.6 displays the result. The non-centered version of the model samples much more efficiently, producing more effective samples per parameter. In practice, this means you don't need as many actual iterations, `iter`, to arrive at an equally good portrait of the posterior distribution. For larger data sets, the savings can mean hours of time. And in some problems, the centered version of the model just won't give you a useful posterior.

This model has 76 parameters: 4 average treatment effects, 4×7 varying effects on actor, 4×6 varying effects on block, 8 standard deviations, and 12 free correlation parameters. You can check them all for yourself with `precis(m14.3,depth=3)`. But effectively the model has only about 27 parameters—check `WAIC(m14.3)`. The two varying effects populations, one for actors and one for blocks, regularize the varying effects themselves. So as usual, each varying intercept or slope counts less than one effective parameter.

We can inspect the standard deviation parameters to get a sense of how aggressively the varying effects are being regularized:

R code 14.21

```
precis( m14.3 , depth=2 , pars=c("sigma_actor","sigma_block") )
```

```
                mean   sd 5.5% 94.5% n_eff Rhat
sigma_actor[1] 1.37 0.47 0.77  2.20   832    1
sigma_actor[2] 0.91 0.40 0.42  1.62  1108    1
sigma_actor[3] 1.85 0.55 1.12  2.82   961    1
sigma_actor[4] 1.58 0.58 0.87  2.58  1109    1
sigma_block[1] 0.40 0.32 0.04  0.98  1112    1
```

```
sigma_block[2] 0.42 0.33 0.03  1.03   903    1
sigma_block[3] 0.31 0.28 0.02  0.80  1740    1
sigma_block[4] 0.48 0.37 0.04  1.16   942    1
```

While these are just posterior means, and the amount of shrinkage averages over the entire posterior, you can get a sense from the small values that shrinkage is pretty aggressive here, especially in the case of the blocks. This is what takes the model from 76 actual parameters to 27 effective parameters, as measured by WAIC (or PSIS—it agrees in this case).

This is a good example of how varying effects adapt to the data. The overfitting risk is much milder here than it would be with ordinary fixed effects. It can of course be challenging to define and fit these models. But if you don't check for variation in slopes, you may never notice it. And even if the average slope is almost zero, there might still be substantial variation in slopes across clusters.

Before leaving this example behind, let's look at the posterior predictions against the average for each actor and each treatment, as we did back in Chapter 11. This is going to be a big chunk of code, just like it was back in the earlier chapter. But there is nothing new here really. I'll use block number 5 in these predictions, because it had almost zero effect, and we want to average over blocks in this visualization.

R code 14.22

```
# compute mean for each actor in each treatment
pl <- by( d$pulled_left , list( d$actor , d$treatment ) , mean )

# generate posterior predictions using link
datp <- list(
    actor=rep(1:7,each=4) ,
    tid=rep(1:4,times=7) ,
    block_id=rep(5,times=4*7) )
p_post <- link( m14.3 , data=datp )
p_mu <- apply( p_post , 2 , mean )
p_ci <- apply( p_post , 2 , PI )

# set up plot
plot( NULL , xlim=c(1,28) , ylim=c(0,1) , xlab="" ,
    ylab="proportion left lever" , xaxt="n" , yaxt="n" )
axis( 2 , at=c(0,0.5,1) , labels=c(0,0.5,1) )
abline( h=0.5 , lty=2 )
for ( j in 1:7 ) abline( v=(j-1)*4+4.5 , lwd=0.5 )
for ( j in 1:7 ) text( (j-1)*4+2.5 , 1.1 , concat("actor ",j) , xpd=TRUE )

xo <- 0.1 # offset distance to stagger raw data and predictions
# raw data
for ( j in (1:7)[-2] ) {
    lines( (j-1)*4+c(1,3)-xo , pl[j,c(1,3)] , lwd=2 , col=rangi2 )
    lines( (j-1)*4+c(2,4)-xo , pl[j,c(2,4)] , lwd=2 , col=rangi2 )
}
points( 1:28-xo , t(pl) , pch=16 , col="white" , cex=1.7 )
points( 1:28-xo , t(pl) , pch=c(1,1,16,16) , col=rangi2 , lwd=2 )

yoff <- 0.175
text( 1-xo , pl[1,1]-yoff , "R/N" , pos=1 , cex=0.8 )
text( 2-xo , pl[1,2]+yoff , "L/N" , pos=3 , cex=0.8 )
text( 3-xo , pl[1,3]-yoff , "R/P" , pos=1 , cex=0.8 )
text( 4-xo , pl[1,4]+yoff , "L/P" , pos=3 , cex=0.8 )
```

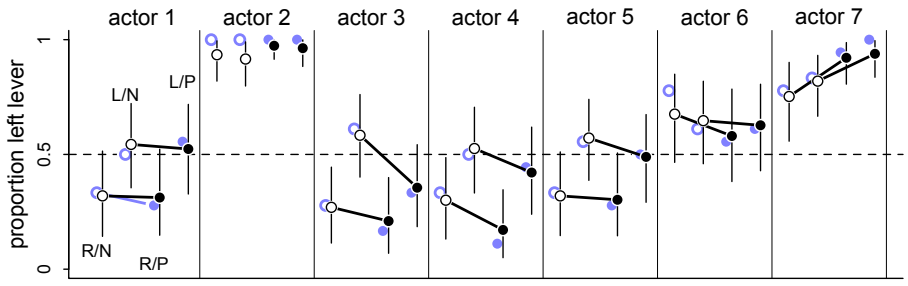

FIGURE 14.7. Posterior predictions, in black, against the raw data, in blue, for model `m14.3`, the cross-classified varying effects model. The line segments are 89% compatibility intervals. Open circles are treatments without a partner. Filled circles are treatments with a partner. The prosocial location alternates right-left-right-left, as labeled in actor 1.

```
# posterior predictions
for ( j in (1:7)[-2] ) {
    lines( (j-1)*4+c(1,3)+xo , p_mu[(j-1)*4+c(1,3)] , lwd=2 )
    lines( (j-1)*4+c(2,4)+xo , p_mu[(j-1)*4+c(2,4)] , lwd=2 )
}
for ( i in 1:28 ) lines( c(i,i)+xo , p_ci[,i] , lwd=1 )
points( 1:28+xo , p_mu , pch=16 , col="white" , cex=1.3 )
points( 1:28+xo , p_mu , pch=c(1,1,16,16) )
```

The result appears as FIGURE 14.7. The raw data are shown in blue. The posterior means and 89% compatibility intervals are shown in black. As in the earlier chapter, open circles are treatments without a partner. Filled circles are those with a partner. The prosocial treatments alternate right-left-right-left, as labeled in actor 1. The most obvious difference from earlier is that the model accommodates a lot more variation among individuals. Letting each actor have his or her own parameters allows achieves this, at least when there is sufficient data for each actor. Notice however that the posterior does not just repeat the data—there is shrinkage in several places. Actor 2 is the most obvious. Recall that actor 2 always, in every treatment and block, pulled the left lever. The blue points cling to the top. But the posterior predictions shrink inward. Why do they shrink inward more for some treatments, like 1 and 2, than others? Because those treatments had less variation among actors. Look back at the `precis` output on the previous page. The less variation among actors in a treatment, the more shrinkage among actors in that same treatment.

Our interpretation of this experiment has not changed. These chimpanzees simply did not behave in any consistently different way in the partner treatments. The model we've used here does have some advantages, though. Since it allows for some individuals to differ in how they respond to the treatments, it could reveal a situation in which a treatment has no effect on average, even though some of the individuals respond strongly. That wasn't the case here. But often we are more interested in the distribution of responses that the average response, so a model that estimates the distribution of treatment effects is very useful.

Suppose for example that we are testing a pain reliever, like aspirin. For many medications, only some people benefit. The average treatment effect is not really as interesting as the distribution of treatment effects, in such cases.

Overthinking: Non-centered parameterization of the multilevel model. When there are inefficient chains, often running the chains long enough will produce reliable samples from the posterior. This was the case with `m14.2` in the main text. But this is both inefficient and unreliable. The chains could still be biased in subtle ways that are hard to detect. Better to re-parameterize, as explained in the preceding section.[195] How does this work in the case of covariance matrixes?

Model `m14.3` uses a trick known as the Cholesky decomposition to smuggle the covariance matrix out of the prior. The top part of the model is the same as the centered version, `m14.2`. The changes are the extra lines that construct the adaptive priors:

```
# adaptive priors - non-centered
transpars> matrix[actor,4]:alpha <-
        compose_noncentered( sigma_actor , L_Rho_actor , z_actor ),
transpars> matrix[block_id,4]:beta <-
        compose_noncentered( sigma_block , L_Rho_block , z_block ),
matrix[4,actor]:z_actor ~ normal( 0 , 1 ),
matrix[4,block_id]:z_block ~ normal( 0 , 1 ),
```

These two lines that begin with `transpars>` define the matrixes of varying effects `alpha` and `beta`. Each is a matrix with a row for each actor/block and a column for each effect. As a convenience, `compose_noncentered` mixes the vector of standard deviations, the correlation matrix, and the z-scores together to make a matrix of parameters on the correct scale for the linear model. This means that the matrixes of z-scores—the third and fourth lines above—can just be `normal(0,1)`. The other change to the model, to make it non-centered, is that the correlation matrixes have been replaced with something called a Cholesky factor, `cholesky_factor_corr` to be precise.

So what is `compose_concentered` doing? And what are these mysterious Cholesky factors? A **Cholesky decomposition** $\mathbf{L}$ is a way to represent a square, symmetric matrix like a correlation matrix $\mathbf{R}$ such that $\mathbf{R} = \mathbf{L}\mathbf{L}^{\intercal}$. It is a marvelous fact that you can multiply $\mathbf{L}$ by a matrix of uncorrelated samples (z-scores) and end up with a matrix of correlated samples (the varying effects). This is the trick that lets us take the covariance matrix out of the prior. We just sample a matrix of uncorrelated z-scores and then multiply those by the Cholesky factor and the standard deviations to get the varying effects with the correct scale and correlation. It would be magic, except that it is just algebra.

Let's look at the raw Stan code, to demystify all of this and help you transition to building models directly in Stan, where you will have more control. Those `transpars>` flags in the `ulam` code define the matrixes `alpha` and `beta` as **transformed parameters**, which means that Stan will include them in the posterior, even though they are just functions of parameters. So if you look at `stancode(m14.3)`, you'll see a new block above the `model` block:

```
transformed parameters{
    matrix[7,4] alpha;
    matrix[6,4] beta;
    beta = (diag_pre_multiply(sigma_block, L_Rho_block) * z_block)';
    alpha = (diag_pre_multiply(sigma_actor, L_Rho_actor) * z_actor)';
}
```

These are the calculations that merge vectors of standard deviations, `sigma_actor` and `sigma_block`, with Cholesky correlation factors, `L_Rho_actor` and `L_Rho_block`. The function `diag_pre_multiply` does this—all it does is make a diagonal matrix from the `sigma` vector and then multiply, producing a Cholesky factor for the right covariance matrix. Finally, that Cholesky covariance factor is matrix multiplied by the matrix of z-scores. For convenience, the thing is transposed—that `'` on the end of each line—so we can index it as `alpha[actor,effect]` instead of `alpha[effect,actor]`. But really that step isn't necessary.

Then down in the model block, the matrixes `alpha` and `beta` are just available as parameters, so the linear model part looks the same:

```
model{
    vector[504] p;
    L_Rho_block ~ lkj_corr_cholesky( 2 );
    sigma_block ~ exponential( 1 );
    L_Rho_actor ~ lkj_corr_cholesky( 2 );
    sigma_actor ~ exponential( 1 );
    g ~ normal( 0 , 1 );
    to_vector( z_block ) ~ normal( 0 , 1 );
    to_vector( z_actor ) ~ normal( 0 , 1 );
    for ( i in 1:504 ) {
        p[i] = g[tid[i]] + alpha[actor[i], tid[i]] + beta[block_id[i], tid[i]];
        p[i] = inv_logit(p[i]);
    }
    L ~ binomial( 1 , p );
}
```

From top to bottom: The vector `p` is declared to hold our linear model calculations for each case, then the priors are defined in terms of Cholesky correlation factors and vectors of standard deviations. The z-score matrixes are assigned their prior using `to_vector`, because `normal(0,1)` applies to vectors, not matrixes. The z-scores are still stored in matrix format—this `to_vector` stuff is just needed to force the same `normal(0,1)` prior on each cell in the matrix. Finally the linear model is computed, using the `alpha` and `beta` matrixes form the `transformed parameters` block, and then the probability of the data is defined as usual.

The last bit is generated quantities, where variables that are functions of each sample can be calculated. This block is used here to transform the Cholesky factors into ordinary correlation matrixes, so they can be interpreted as such, as well as to compute the log-probabilities needed to calculate WAIC or PSIS.

```
generated quantities{
    vector[504] log_lik;
    vector[504] p;
    matrix[4,4] Rho_actor;
    matrix[4,4] Rho_block;
    Rho_block = multiply_lower_tri_self_transpose(L_Rho_block);
    Rho_actor = multiply_lower_tri_self_transpose(L_Rho_actor);
    for ( i in 1:504 ) {
        p[i] = g[tid[i]] + alpha[actor[i], tid[i]] + beta[block_id[i], tid[i]];
        p[i] = inv_logit(p[i]);
    }
    for ( i in 1:504 ) log_lik[i] = binomial_lpmf( L[i] | 1 , p[i] );
}
```

The function `multiply_lower_tri_self_transpose` is just a compact and efficient way to perform the matrix algebra needed to turn the Cholesky factor $\mathbf{L}$ into the corresponding matrix $\mathbf{R} = \mathbf{L}\mathbf{L}^{\intercal}$.

There is an obvious cost to these non-centered forms: They look a lot more confusing. Hard-to-read models and model code limit our ability to share implementations with our colleagues, and sharing is a principle goal of scientific computation.

Finally, not all combinations of model structure and data benefit from the non-centered parameterization. Sometimes the centered version—putting the means and standard deviations in the prior—is better. So you might try the form that is most natural for you personally. If it gives you trouble, try an alternative form. With some experience, different forms of the same model become familiar. There is a practice problem at the end of this chapter that may help.

14.3. Instruments and causal designs

Back in Chapter 6, you met a framework for deciding which variables to use in a regression. The key idea is that, in a graphic model like a DAG, many paths may connect a variable