# Lecture 20: Hierachical redeux—gettin' cosy with covariance

** This lecture is based on chapter 13, of Statistical Rethinking by Richard McElreath.*

```
library(rstan)
library(shinystan)
library(car)
library(mvtnorm)
library(rethinking)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
source("../utilityFunctions.R")
```

A long time ago, in those halcyon days of yore, we were all idealistic about Bayes, having listened to sweet nothings of statistical magic courtesy of the Bayesian fairy.

Back then, before we got jaded with the fact that Bayes isn't magic, and isn't that much more complicated than regular likelihood-based statistics, we learned the basics of hierarchical modeling.

This lecture will revisit multilevel modeling in a regression framework.

- Recall back in lecture 6 we learned how to create hierarchical models with varying intercepts. In these models, our parameters of interest shared *hyperparameters*—priors that themselves had prior distributions.
  - i.e., hyperpriors

These varying intercepts experienced shrinkage because of the hyperpriors. As a recap:

- In partial pooling models, the hyperprior (overarching prior describing the lower level parameters) is adaptive.
  - The hyperprior learns from the lower level parameters and describes the overall "average effect."
- Just as with non-hierarchical models, if there isn't a lot of information, the prior will overwhelm the likelihood. If there isn't a lot of info in a cluster or group—perhaps because of few observations—that parameter will be pulled (shrunk) considerably towards the hyperprior (the grand mean).
- Conversely, if there is a lot of info, the likelihood will run the show and the parameter will not be shrunk very much.

## Bees!

Suppose we are interested in the amount of time that bees spend nectaring on flowers.

We set up an experiment where we pick 20 different plants in a field and record the amount of time $X$ number of bees spend on each plant.

- Each bee visit on a particular plant isn't independent, so we would probably want to model this as a partial-pooling model with varying intercepts.

Now suppose we record time of day. The average nectaring time might be longer in the morning than the afternoon because the bees are cold and take a while to get going, or because the nectar content in flowers goes down in the afternoon due to depletion or plant dehydration.

- Just as bees on flowers vary in average nectaring times, they also vary in the *difference* between morning and afternoon (i.e., slopes).

This linear model might look like:

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

where $A_i$ is a binary indicator for `afternoon` and $\beta_{\text{plant}[i]}$ is a parameter for the expected difference between morning and afternoon for each plant.

As we learn about the intercepts when we pool information, we also learn more efficiently by pooling information about the slopes.

- This pooling happens the same way, by estimating the **population distribution** of slopes simultaneously to estimating each slope.

This is the general *varying effects* strategy: any collection of *exchangable* groups or categories can and probably should be pooled.

- Remember exchangability means groups don't have true ordering because they have arbitrary index labels.

However, when we generalize varying effects beyond intercepts, we have the ability to gain extra information out of our data. Flower nectaring times will *covary* in their intercepts and slopes. Why?

- Suppose the mechanism between the difference in morning and afternoon nectaring times is depletion.
- If a plant has a lot of nectar in the morning, bees will tell their sisters to visit the plant more often and for longer, depleting the reserves. Thus, by afternoon, the difference in nectar amounts as compared to morning will be large.
  - The intercept will be high (and far from zero) and the slope will be strongly negative.
- If there is only a small amount of nectar to begin with, the bees may not bother and the difference between morning and afternoon will be less.
- Across the entire population of plants—including high nectar and low-nectar ones, intercepts and slopes will covary.

This covariation is information that our model can use. If both slopes and intercepts covary, pooling across parameter types—intercepts and slopes—will improve estimation of how different groups are influenced by predictor variables.

- By borrowing information from the slopes, our intercept estimates are also improved.
  - Varying intercepts **AND** slopes together implicitly create interaction machines.

## Lets simulate bees!

We will start by defining a population of plants that our bees visit. This involves defining the average nectaring time in the morning and afternoon, as well as the correlation between.

- These values will define our entire population of plants.

```r
a <- 3.5          # average morning nectaring time
b <- -1         # average diff in afternoon time
sigma.a <- 1      # std dev. in intercept
sigma.b <- 0.5    # std dev. in slope
rho <- -0.7       # Correlation between intercept and slope
```

Now we will build a 2-D multivariate normal distribution with a vector of two means ($\boldsymbol{\mu}$), and $\Sigma$, a $2 \times 2$ matrix of variances and covariances.

```r
Mu <- c(a, b)
```

The value `a` is the mean intercept—the average morning nectar time.

The value `b` is the mean slope—the difference between morning and afternoon.

The covariance matrix looks like this:

$$\Sigma = \begin{bmatrix} \sigma_\alpha^2 & \sigma_\alpha\sigma_\beta\rho \\ \sigma_\alpha\sigma_\beta\rho & \sigma_\beta^2 \end{bmatrix}$$

where the variances of the intercepts and slopes are $\sigma_\alpha^2$ and $\sigma_\beta^2$, respectively on the diagonal. The covariance between intercept and slope ($\sigma_\alpha\sigma_\beta\rho$) is on the off-diagonals.

- It is the product of the two standard deviations and the correlation ($\rho$).
- Useful to remember the variance is the covariance of a variable with itself.

There are a few options to build the VCV:

1. Directly

```r
cov.ab <- sigma.a * sigma.b * rho
Sigma <- matrix(c(sigma.a^2, cov.ab,
                  cov.ab, sigma.b^2), ncol=2)
```

2. Decompose the VCV into diagonal matrices of standard deviations and a correlation matrix ($\Omega$) separately and then use matrix multiplication to produce the covariance matrix (much easier conceptually and practically).
    - We will use this later on to build priors so lets learn now!

Basically, we can create the VCV by premultiplying a diagonal matrix of standard deviations by the correlation matrix and then postmultiplying by the diagonal standard deviation matrix again.

$$\Sigma = \begin{bmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{bmatrix} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \begin{bmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{bmatrix}$$

```
sigmas <- c(sigma.a, sigma.b)
Omega <- matrix(c(1, rho, rho, 1), nrow=2)
Sigma <- diag(sigmas) %*% Omega %*% diag(sigmas)
```

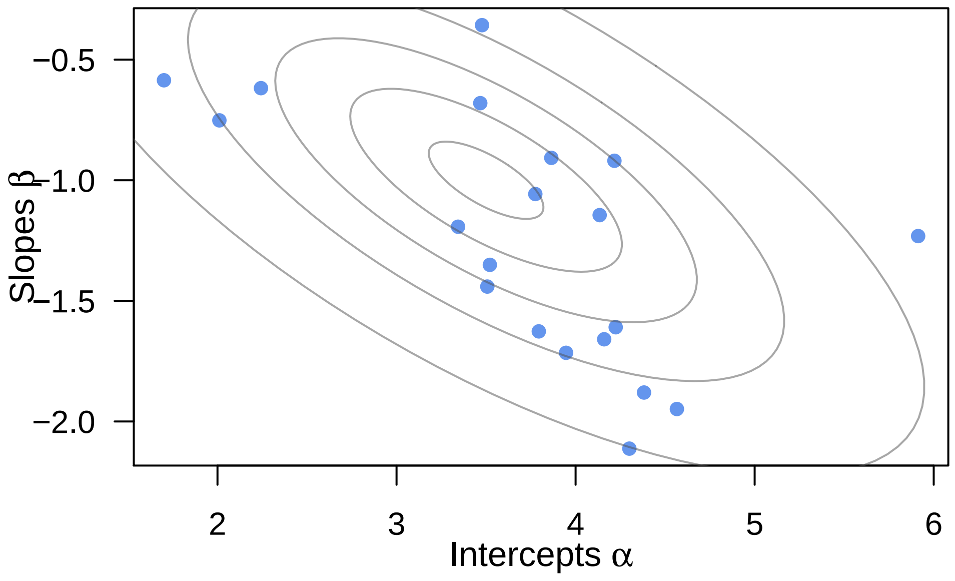
Now that we have our population-level parameters, lets simulate data.

```
n.plants <- 20 # no. of plants
library(MASS)
set.seed(5)
plantEffects <- mvrnorm(n.plants, Mu, Sigma)
aPlant <- plantEffects[,1]
bPlant <- plantEffects[,2]
```

The content of the `plantEffects` object should now be a matrix of 20 rows and 2 columns, where each row is a plant, the first column is an intercept, and the second column contains the slopes.

We can visualize by plotting them against each other and then overlaying the population distribution:

```
par(mar=c(3,3.3,0.1,0.5))

plot(aPlant, bPlant, col="cornflowerblue", las=1, ann=FALSE,
  pch=16)
mtext(expression(paste("Intercepts ",alpha)), side=1, line=2,
  cex=1.1)
mtext(expression(paste("Slopes ",beta)), side=2, line=2.2, cex=1.1)

library(ellipse)
c.int <- c(0.05, 0.25, 0.5, 0.75, 0.95)
for(i in c.int) {
  lines(ellipse(Sigma, centre=as.vector(Mu), level=i),
    col="#50505080")
}
```



The contour lines display the multivariate normal population of intercepts and slopes that the plants were sampled from.

So far we have simulated plants and their average properties. Now let's simulate bees nectaring. We are going to simulate 10 visits per plant, 5 in the morning, and 5 in the afternoon.

```r
n.visits <- 10
afternoon <- rep(0:1, n.visits*n.plants/2)
plantID <- rep(1:n.plants, each=n.visits)

mu <- aPlant[plantID] + bPlant[plantID]*afternoon
tau <- 0.5  # std dev. w/in plant

set.seed(2)
nectar <- rnorm(n.visits * n.plants, mu, tau)
dat <- data.frame(plant=plantID, afternoon=afternoon,
  nectar=nectar)
head(dat)
```

```
  plant afternoon   nectar
1     1         0 3.775505
2     1         1 2.707031
3     1         0 5.017885
4     1         1 2.049418
5     1         0 4.183837
6     1         1 2.680816
```

In our data, we have multiple clusters. These are the plants. Each cluster is observed under different conditions, so we can estimate both an individual intercept and individual slope for each cluster.

Also note that our data are balanced, but this need not be the case. In fact, lack of balance is where multilevel models really shine, because partial pooling uses information about the population where it's needed the most.

## Hierarchical modeling

Now that we have our data, let's make a model. This is going to look a lot like all the other models we have done, but now we have a joint population of intercepts and slopes that we want to estimate, and hierarchical structure.

Here is the model:

$$
\begin{aligned}
y_i &\sim Normal(\mu_i, \tau) \\
\mu_i &= \alpha_{\text{plant}[i]} + \beta_{\text{plant}[i]} \\
\begin{bmatrix} \alpha_{\text{plant}} \\ \beta_{\text{plant}} \end{bmatrix} &\sim \text{MVNormal}(\begin{bmatrix} \gamma_\alpha \\ \gamma_\beta \end{bmatrix}, \Sigma) \\
\Sigma &= \begin{bmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{bmatrix} \Omega \begin{bmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{bmatrix} \\
\gamma &\sim \text{Normal}(0, 1) \\
\tau &\sim \text{Cauchy}^+(0, 1) \\
\sigma_\alpha &\sim \text{Cauchy}^+(0, 1) \\
\sigma_\beta &\sim \text{Cauchy}^+(0, 1) \\
\Omega &\sim \text{LKJcorr}(2)
\end{aligned}
$$

A few things to point out:

$$\begin{bmatrix} \alpha_{\text{plant}} \\ \beta_{\text{plant}} \end{bmatrix} \sim \text{MVNormal}(\begin{bmatrix} \gamma_\alpha \\ \gamma_\beta \end{bmatrix}, \Sigma)$$

- This line says that each plant has an intercept $\alpha_{\text{plant}}$ and slope $\beta_{\text{plant}}$ with a prior distribution defined by a 2-D Normal distribution with means $\gamma_\alpha$ and $\gamma_\beta$ and covariance matrix $\Sigma$.

$$\Omega \sim \text{LKJcorr}(2)$$

Our correlation matrix needs a prior. In our case,

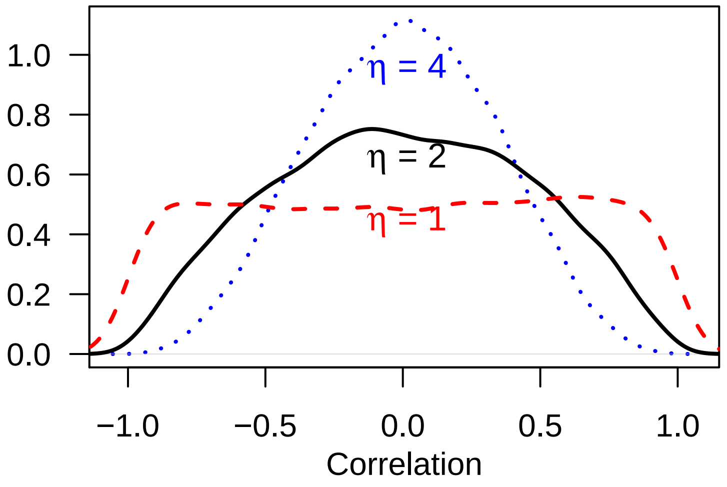
$$\Omega = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

where $\rho$ is the correlation between slopes and intercepts, so there is only one parameter we need to define a prior for.

The LKJcorr(2) does is define a weakly informative prior for the correlation between slope and intercept that is slightly skeptical of correlations near -1 or 1.

- The LKJ distribution has a single parameter, $\eta$, that controls how skeptical the prior is of large correlations in the matrix. When LKJcorr is 1, the prior is flat.
- Increasing $\eta$ increases our skepticism of correlations far from 0.

```r
par(mar=c(3,3.1,0.1,0.5))

plot(density(rlkjcorr(1e4, K=2, eta=4)[,1,2]), lty=3, lwd=2, adj=1,
  ann=FALSE, las=1, col="blue")
mtext("Correlation", side=1, line=2)

lines(density(rlkjcorr(1e4, K=2, eta=2)[,1,2]), lty=1, lwd=2)
lines(density(rlkjcorr(1e4, K=2, eta=1)[,1,2]), lty=2, lwd=2,
  col="red")

text(-0.13, 0.39, expression(paste(eta, " = 1")), adj=c(0,0),
  cex=1.1, col="red")
text(-0.13, 0.6, expression(paste(eta, " = 2")), adj=c(0,0), cex=1.1)

text(-0.13, 0.9, expression(paste(eta, " = 4")), adj=c(0,0),
  cex=1.1, col="blue")
```



Here is the model

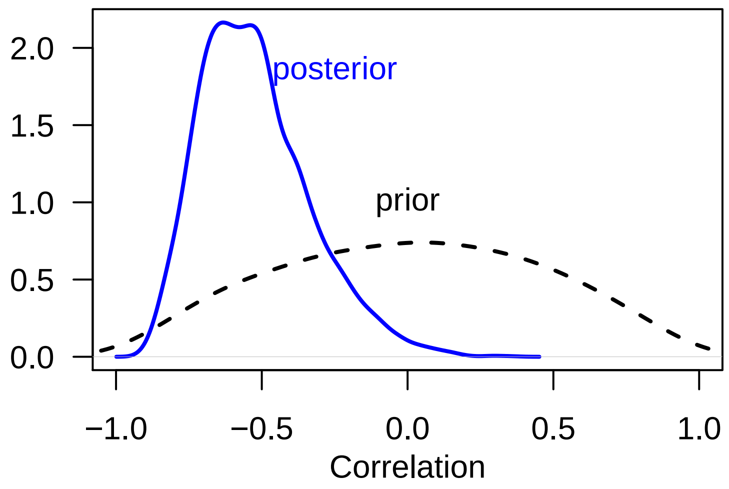
```
data {
  int<lower=1> nObs;
  int<lower=1> nPlants;
  vector[nObs] obs;
  int<lower=1> plant[nObs];
  int afternoon[nObs];
}

parameters {
  vector[nPlants] bPlant;
  vector[nPlants] aPlant;
  vector[2] gamma;
  vector<lower=0>[2] sigma;
  real<lower=0> tau;
  corr_matrix[2] Omega;
}

transformed parameters {
  vector[2] beta[nPlants];
  cov_matrix[2] Sigma;
  for(p in 1:nPlants) {
    beta[p,1] = aPlant[p];
    beta[p,2] = bPlant[p];
  }

  Sigma = quad_form_diag(Omega,sigma);
}

model {
  vector[nObs] mu;
  Omega ~ lkj_corr(2);
  tau ~ cauchy(0 , 1);
  sigma ~ cauchy(0 , 1);
  gamma ~ normal(0 , 1);
  beta ~ multi_normal( gamma , Sigma);

  for ( n in 1:nObs )
    mu[n] = aPlant[plant[n]] + bPlant[plant[n]] * afternoon[n];

  obs ~ normal( mu , tau );
}
```

```
obs <- nectar
nObs <- nrow(dat)
d <- list(nObs=nObs, obs=obs, nPlants=max(dat$plant), plant=dat$plant, afternoon=dat$aftern

m1 <- stan(file = "varSlopesMod.stan", data = d, iter = 2000,
  chains = 4, seed = 4)

post <- extract(m1, c("aPlant", "bPlant", "gamma", "Sigma"))
```

First, ignore the warnings we get from the model.

Next, let's look at the posterior distribution of correlations between slope and intercept.

```
par(mar=c(3,3.1,0.1,0.5))
Omega <- as.matrix(m1, pars="Omega")
Rho <- Omega[,2]

plot(density(Rho), lty=1, lwd=2, ann=FALSE, las=1, col="blue", xlim=c(-1,1))
mtext("Correlation", side=1, line=2)

dens(rlkjcorr(1e4, K=2, eta=2)[,1,2], add=TRUE, lwd=2,lty=2, adj=2)
text(0,1, "prior")
text(-0.25,1.85, "posterior", col="blue")
```

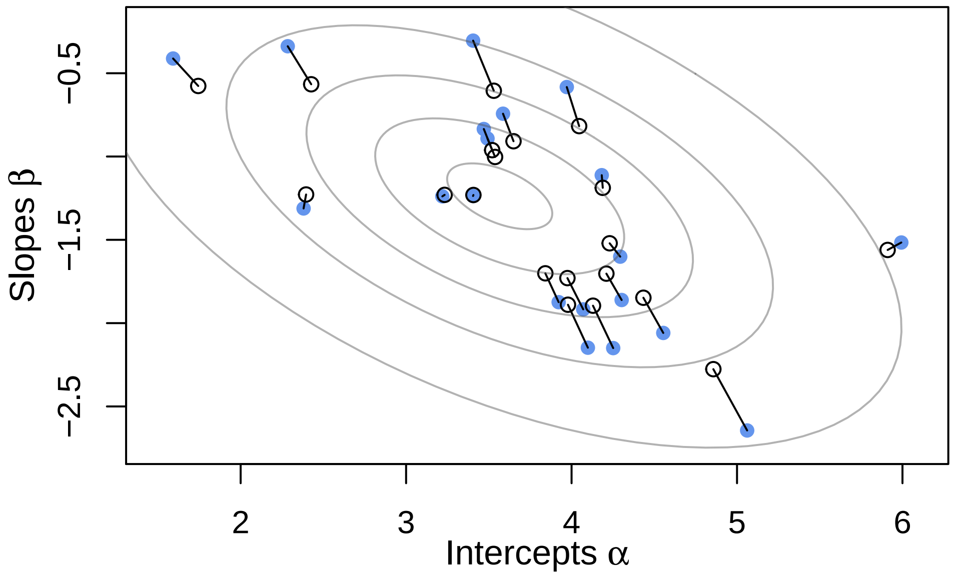


The blue density is the posterior distribution of the correlation between intercepts and slopes. It is concentrated on negative values because our model has adaptively learned.

- Note that all the model had to work with were the observed nectaring times between morning and afternoon.

Next, let's look at some shrinkage!

```
par(mar=c(3,3.1,0.1,0.5))
# compute unpooled estimates from data
a1 <- sapply(1:n.plants, function(i) mean(nectar[plantID==i &
    afternoon == 0]))
b1 <- sapply(1:n.plants, function(i) mean(nectar[plantID==i &
    afternoon == 1]))-a1

# Extract posterior means of partially-pooled estimates
aPP <- colMeans(post$aPlant)
bPP <- colMeans(post$bPlant)

# plot both and connect with lines
plot(a1, b1, ann=FALSE, pch=16, col="cornflowerblue",
  xlim=c(min(a1)-0.1, max(a1)+0.1), ylim=c(min(b1)-0.1,
    max(b1)+0.1))
points(aPP,bPP, pch=1)

mtext(expression(paste("Intercepts ",alpha)), side=1, line=2,
  cex=1.1)
mtext(expression(paste("Slopes ",beta)), side=2, line=2, cex=1.1)

for(n in 1:n.plants) lines(c(a1[n], aPP[n]), c(b1[n], bPP[n]))

### now superimpose contours of population
Mu.est <- colMeans(post$gamma)
Sigma.est <- colMeans(post$Sigma)

for(l in c.int) {
  lines(ellipse(Sigma.est, centre=Mu.est, level=l), col="#50505070")
}
```



Our model estimates posterior distributions for the intercepts and slopes for each plant.

- The inferred correlation between the two was used to pool information across them, just as the inferred variation among intercepts and inferred variation among slopes pools information among them.

Together, the variances and covariances define the multivariate normal prior that then adaptively shrinks both intercepts and slopes!

This can be seen in the plot. The blue points are our unpooled estimates from the data itself. The open points are posterior means from our model.

- Each open point is shrunk towards the center of the contours in both dimensions, which represent the multivariate posterior distribution of the population of slopes and intercepts.

Note that the shrinkage is pulled along the contours. Shrinking the slope down also shrinks the intercept along the diagonal as a result of the correlation.

## Non-centered parameterization

Sometimes, when hierarchical models are hard to fit, they will develop weird pathologies where some parts of the posterior will be very narrow and dense, while other parts will be very wide and diffuse, like a funnel.

These cause problems that are manifested as *divergent transitions.* If you get more than one or two of these, they indicate problems.

The modeling strategy I showed above is called the centered parameterization because it is centered around the population mean.

- Often has the nasty funnel shape in hierarchical models.

When data are sparse, the centered parameterization of the posterior resembles this funnel.

- With low data, the constraint of the likelihood means that the model will be completely dominated by the hierarchical posterior and the funnel.
- With lots of data, the likelihood "cuts out" large regions of the funnel and we can zoom in on regions that are easier to sample from.

If we get divergent transitions, we can use a different strategy.

- Essentially, for any Gaussian distribution, we can describe a Gaussian with another Gaussian by subtracting out the mean and factoring out the standard deviation.

So this:

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

is equivalent to this:

$$\begin{aligned} y &= \mu + z\sigma \\ z &\sim \text{Normal}(0, 1). \end{aligned}$$

This means we can take out the $\mu's$ and $\sigma's$ from a Normal distribution and put them into a linear model. This leaves only a standardized multivariate normal Normal prior, with all means at 0 and all standard deviations at 1.

We can further improve efficiency by using a **Cholesky Decomposition** of our correlation matrix. This is essentially a way to take the square root of a matrix.

Here is how we do it in Stan:

```
data {
  int<lower=1> nObs;
  int<lower=1> nPar;
  int<lower=1> nPlants;
  vector[nObs] obs;
  int<lower=1> plant[nObs];
  matrix[nPlants, 1] hVar;
  matrix[nObs, nPar] afternoon;
}

parameters {
  matrix[nPar, nPlants] z;
  matrix[1, nPar] gamma;
  vector<lower=0>[2] sigma;
  real<lower=0> tau;
  cholesky_factor_corr[2] Omega;
}

transformed parameters {
  matrix[nPlants, nPar] beta;

  beta = hVar*gamma + (diag_pre_multiply(sigma,Omega) * z)';
}
```

```
model {
  vector[nObs] mu;

  Omega ~ lkj_corr_cholesky(2);
  tau ~ cauchy(0,1);
  sigma ~ cauchy(0,1);
  to_vector(gamma) ~ normal(0,1);
  to_vector(z) ~ normal(0,1);

  for(n in 1:nObs)
  mu[n] <- afternoon[n] * beta[plant[n]]';

  obs ~ normal(mu, tau);
}
```

```r
afternoon <- model.matrix(~dat$afternoon)
nPar <- ncol(afternoon)
hVar <- as.matrix(rep(1, max(dat$plant)))
d <- list(nObs=nObs, nPar=nPar, nPlants=max(dat$plant), obs=obs,
  plant=dat$plant, hVar=hVar, afternoon=afternoon)

m2 <- stan(file = "noncentered.stan", data = d, iter = 2000,
  chains = 4, seed = 4, pars="z", include=FALSE)
```

Generally, when there is a lot of data, the centered parameterization behaves nicely and the posterior geometry is nice.

When the data are sparse, the non-centered parameterization does better. However, as the amount of data increases, the constraints imposed by the data are more complex and give pathological posteriors.

As a general rule, most models will perform better with either centering or non-centering. If one isn't working well, try the other.

## References

McElreath (2016). Statistical Rethinking: A Bayesian Course with Examples in R and Stan. CRC Press.