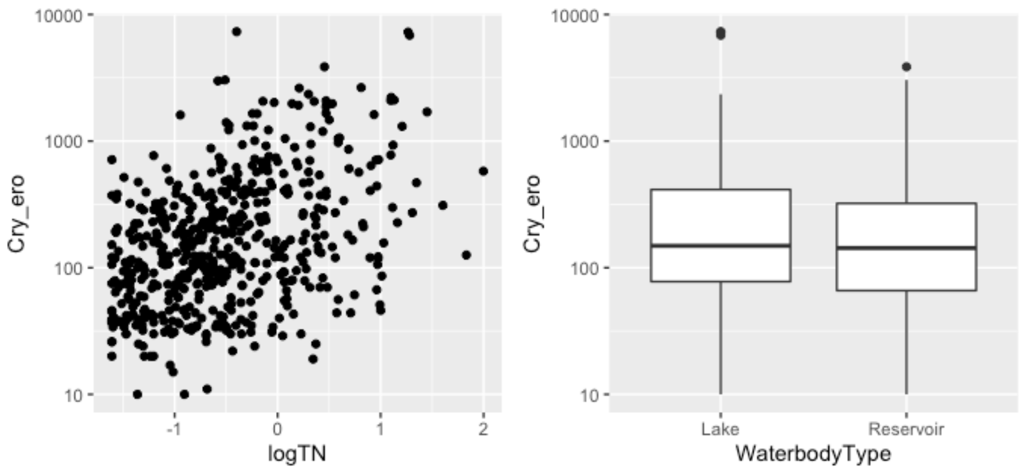
Lecture 20. Mixed models II.

Putting predictors into a random effects model

In this last lecture we learned about what random effects are, how they are specified in a model, how to interpret estimates of the variance and individual random effects, etc. We used the example of *Cryptomonas* density varying among lakes, and we made the simplest possible random effects model for this data (varying intercepts model). Now we're going to look at how to put predictors into a model with random effects.

Earlier we looked at the raw data for two predictors in the data:



I want to quantify the relationship between *Cryptomonas* density and total nitrogen, because nitrogen is one of the main limiting nutrients in aquatic ecosystems. I also want to see whether density differs between natural lakes and reservoirs, because these might differ ecologically in ways that are not well quantified by the commonly measured variables. Now that we are using random effects to model variability at multiple levels, it becomes important to think about *at which level the predictor varies*. In this case, total nitrogen is different for every sample, so it varies within waterbodies (over time) as well as between waterbodies. In contrast, WaterbodyType is a waterbody-level variable by definition. It turns out that we are going to specify both of these predictors the same way in lmer(), but for understanding the results and making inferences it is important to think about the hierarchical structure of the data and where the predictors lie in this hierarchy.

Let's add predictors into the syntax I used previously. At the level of single observation, we have an intercept $\alpha_{j[i]}$ that varies between waterbodies, and we also have a measurement of total nitrogen, which I will call X_i:

$$\mu_i = \alpha_{j[i]} + \beta_1 * X_i$$

$$Y_i \sim \text{Normal}(\alpha_{j[i]}, \sigma_Y)$$

So this is our normal linear model syntax, except for the random intercept. To add WaterbodyType to the mix (lake vs. reservoir), I will put that into the equation for the intercept:

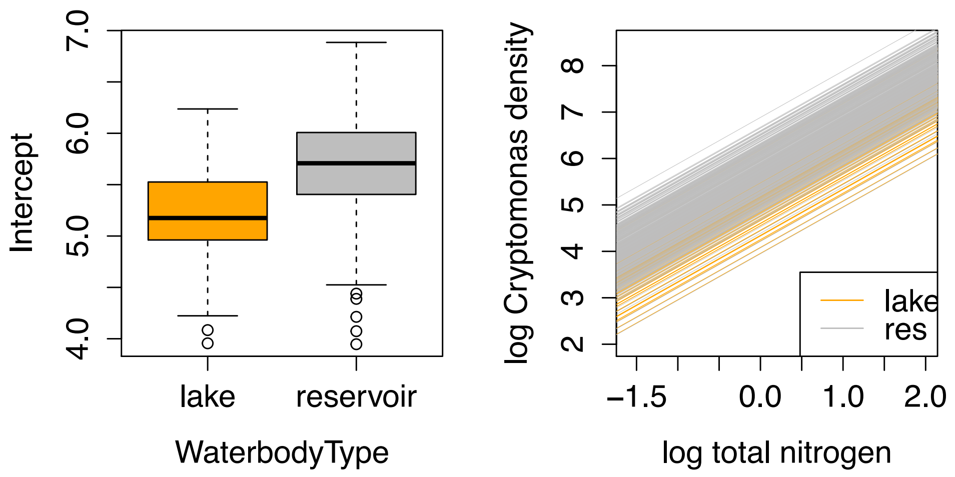
$$\mu_{\alpha j} = \gamma_0 + \gamma_1 * Z_j$$
$$\alpha_j \sim \text{Normal}(\mu_{\alpha j}, \sigma_\alpha)$$

This is a little more complex than what I used in the last lecture, because now we need some new symbols for the waterbody-level formula. Each waterbody intercept α_j is normally distributed with standard deviation σ_α and expected value (mean) $\mu_{\alpha j}$. And *now the equation for the intercept has its own linear model.* I've used γ_0 and γ_1 as the parameters, and Z_j as an indicator variable for the predictor WaterbodyType. So *Z* = 0 if it's a lake, and Z = 1 it's a reservoir. γ_0 is the mean for all lakes, and γ_1 is the difference between the mean for all reservoirs and the mean for all lakes.

To summarize, we now have a multi-level linear model:

- the 'lower' level is a linear model for log (density) vs. TN
- the intercept of the lower-level model varies randomly by waterbody
- the random intercepts have their own upper-level linear model: they vary by waterbody type

To visualize what our model is predicting, I simulated some values. On the left are 533 random effects intercepts (one per waterbody), and these vary by WaterbodyType; on the right are the fitted relationships between density and TN for each lake. Note that the intercepts vary but each line has the same slope. That's because we are not letting the slope vary by group (we'll try that with a subsequent example).



Multi-level predictors in lmer

As with many methods we've covered, mixed models can be challenging to understand but are relatively easy to specify in R. Above I wrote out a kind of 2-level regression equation. To fit this model with lmer, we just write:

```
epamod = lmer(log(Cry_ero) ~ logTN + WaterbodyType + (1|WaterbodyID),
data = crysub)
```

The random effects syntax is the same as before, but now we add in logTN and WaterbodyType as predictors. Conveniently, we don't have to tell lmer that WaterbodyType is a higher-level predictor. That's because the expected value of the observation is the same regardless. For the lower-level part, $\mu_i = \alpha_{j[i]} + \beta_1 * X_i$; but the expected value of $\alpha_{j[i]}$ is $\mu_{\alpha j}$, and $\mu_{\alpha j} = \gamma_0 + \gamma_1 * Z_j$, so we can just plug in $\gamma_0 + \gamma_1 * Z_j$ for $\alpha_{j[i]}$. That gives us

$$\mu_i = \gamma_0 + \gamma_1 * Z_i + \beta_1 * X_i$$

So this is just the syntax for a multiple regression, and that's how predictors are added into the lmer formula. We just need to make sure to include the appropriate random effects so that the variability around the expected value μ_i is correctly modeled.

Let's look at the output:

```
summary(epamod)

## Linear mixed model fit by REML ['lmerMod']
## Formula: log(Cry_ero) ~ logTN + WaterbodyType + (1 | WaterbodyID)
##    Data: crysub
```

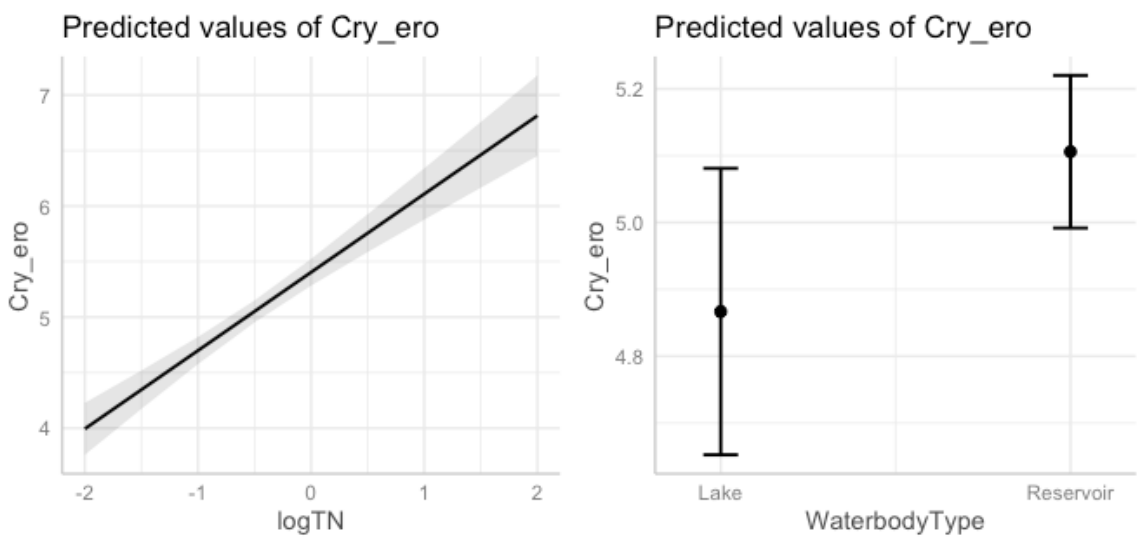
```
##
## REML criterion at convergence: 1582
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.2579 -0.6383 -0.0091  0.5893  3.1371
##
## Random effects:
##  Groups      Name        Variance Std.Dev.
##  WaterbodyID (Intercept) 0.220    0.469
##  Residual                0.924    0.961
## Number of obs: 533, groups:  WaterbodyID, 320
##
## Fixed effects:
##                          Estimate Std. Error t value
## (Intercept)                5.2186     0.1033    50.5
## logTN                      0.7052     0.0713     9.9
## WaterbodyTypeReservoir     0.2391     0.1262     1.9
##
## Correlation of Fixed Effects:
##             (Intr) logTN
## logTN        0.005
## WtrbdyTypRs -0.817  0.351
```

The output is the same as the previous example (random effect for WaterbodyID as the only term), except now we have new fixed effects for logTN and WaterbodyType. The interpretation of the coefficient estimates, standard errors, etc. are the same as for the other kinds of models we've used. It looks like logTN is probably more significant than WaterbodyType, but it will be easier to assess this by making some plots.

Visualizing fixed and random effects

For the fixed effects we can use the ggeffects package as we have in the past to visualize the results:

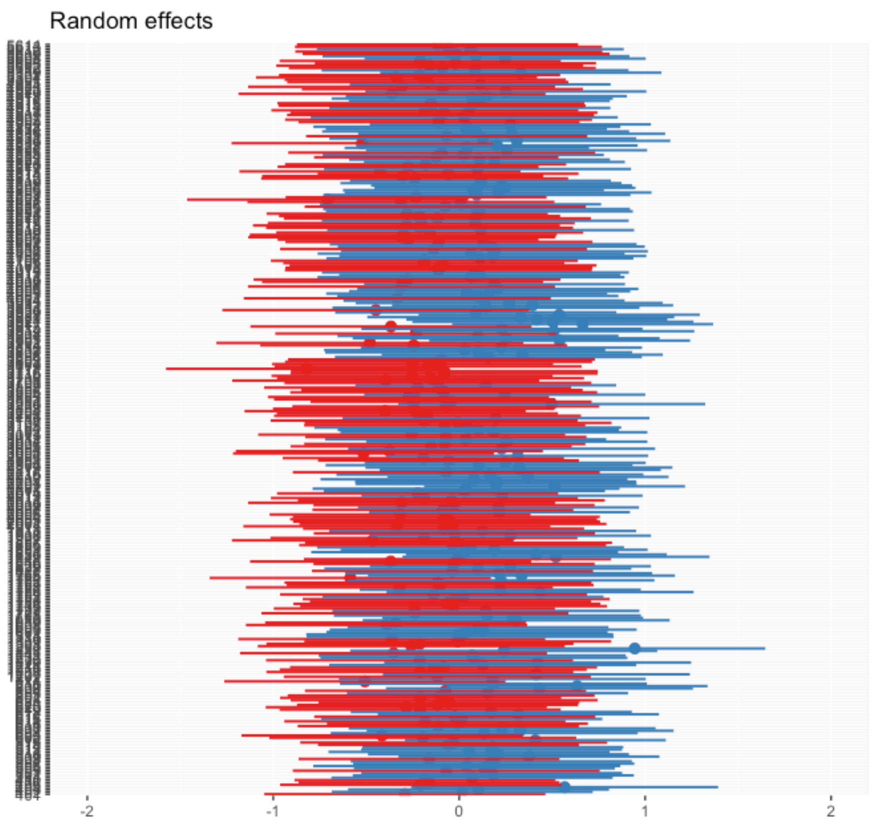
```
grid.arrange(grobs = plot(ggeffect(epamod)), nrow = 1)
```



The effect of logTN is fairly strong; log density increases from 4 to nearly 7, or exp(4) = 55 to exp(7) = 1097 cells mL^{-1}. Clearly there is a lot of residual variation not explained by the model (remember that residuals here include the random effect estimates as part of the fitted model). In contrast, the effect of WaterbodyType is fairly weak; reservoirs tend to have more *Crypomonas*, but the difference is about 0.25 log units, which is a factor of 1.28 on the raw scale.

The effects package helps us visualize the fixed effects, but it does not have functionality for random effects. We can use the caterpillar plot to see what the sorted random effects look like:

```
plot_model(epamod, type = 're')
```



I’m using the entire dataset for this model, so there are 533 observations on 320 waterbodies; with 320 random effects the plotted estimates are hard to distinguish, but we can see that most the the waterbody variation is between -0.5 and 0.5, consistent with the standard deviation for WaterbodyID of 0.47 from the model summary. And as we saw earlier, the individual estimates have large confidence intervals due to small within-waterbody sample size.

As I’ve said several times already, we can think of this mixed model as a random-intercepts model. One way to visualize the fitted results is to look at 1) how well the random intercepts are predicted by the group-level predictor (WaterbodyType), and 2) plot the fitted model as a set of regression lines, one for each group (waterbody). Extracting this info is a bit of work, but it is useful for visualizing exactly what the model means. First, note that we can get the fixed effects estimates using fixef():

```
#fixed effects estimates
fixef(epamod)

##            (Intercept)                  logTN WaterbodyTypeReservoir
##                 5.2186                 0.7052                 0.2391
```

This does the same thing as coef() for LMs and GLMs. What I want to compute is the predicted intercept for each waterbody. That will have two parts: 1) the random effects estimate for that waterbody (recall that these are centered around zero); 2) the fixed effects coefficients that correspond to the group-level predictor for that waterbody (lake vs. reservoir). For lakes, the overall mean is the intercept, and the different lakes vary randomly around that mean. For reservoirs, the overall mean is Intercept + WaterbodyTypeReservoir, and the different reservoirs vary randomly around that mean. So we can combine the the fixed and random components with some code:

```
#extract the random effects for waterbody
random.intercepts = ranef(epamod)$WaterbodyID[[1]]

#make a new factor that codes WaterbodyType for each waterbody
waterbody.type = factor(tapply(crysub$WaterbodyType, crysub$WaterbodyID,
function(x) as.character(x[1])))

#what is the predicted intercept for each waterbody?
fixed.effects.waterbody = fixef(epamod)["(Intercept)"] +
fixef(epamod)["WaterbodyTypeReservoir"]*(waterbody.type == "Reservoir")

#the estimate for each waterbody is the random effect plus the effect of the
group-level predictor
waterbody.effects = random.intercepts + fixed.effects.waterbody
```

Now I want to make two plots. The first is to just plot the waterbody-level prediction against the waterbody-level predictor:

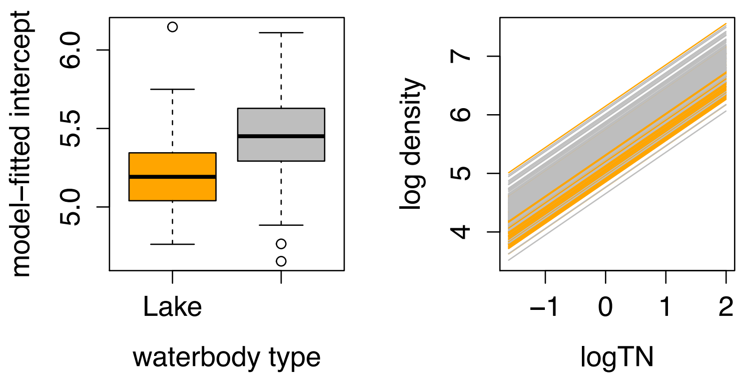
```
#plot the waterbody estimate by waterbody type
plot(waterbody.effects ~ waterbody.type, col = c('orange', 'grey'), xlab =
'waterbody type', ylab = 'model-fitted intercept')
```

For the second plot, I want to show the predicted relationship between log(density) and total nitrogen within each waterbody. We can extract this info easily, because each waterbody gets its own intercept (we just calculated that), and all waterbodies have the same slope for logTN:

```
#plot the model-predicted regression within each waterbody. just drawing a lin
e with curve(), where the intercept varies by waterbody but each waterbody has
 the same slope.
curve(waterbody.effects[1] + fixef(epamod)["logTN"]*x, from = min(crysub$logT
N), to = max(crysub$logTN), xlab = 'logTN', ylab = 'log density', ylim = c(3.
5, 7.5))

for (i in 2:length(waterbody.effects)) {
  curve(waterbody.effects[i] + fixef(epamod)["logTN"]*x, from = min(crysub$log
TN), to = max(crysub$logTN), col = c('orange', 'grey')[waterbody.type[i]], add
 = T)
}
```

Here I’ve used curve() to plot a line for each waterbody. The intercept varies by waterbody, but the slope does not. Here are the two plots:

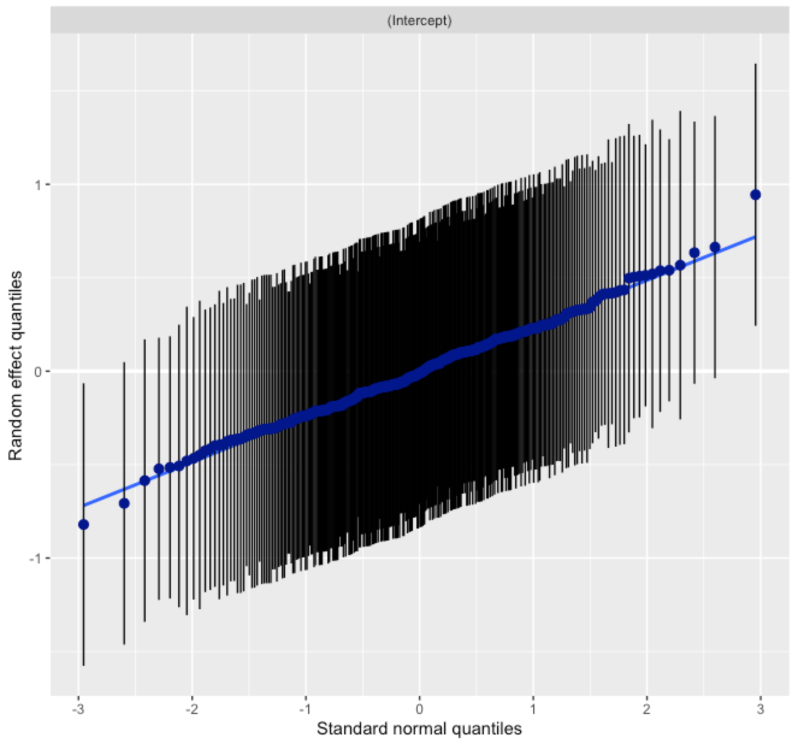


The plot on the left is similar to what we saw from the effects plot; now I'm showing the random effects variation as well. The plot on the right shows how density is predicted to vary with logTN within each waterbody. Because there are 320 lakes the lines bleed together, but we can get a sense for the modeled variation in this relationship.

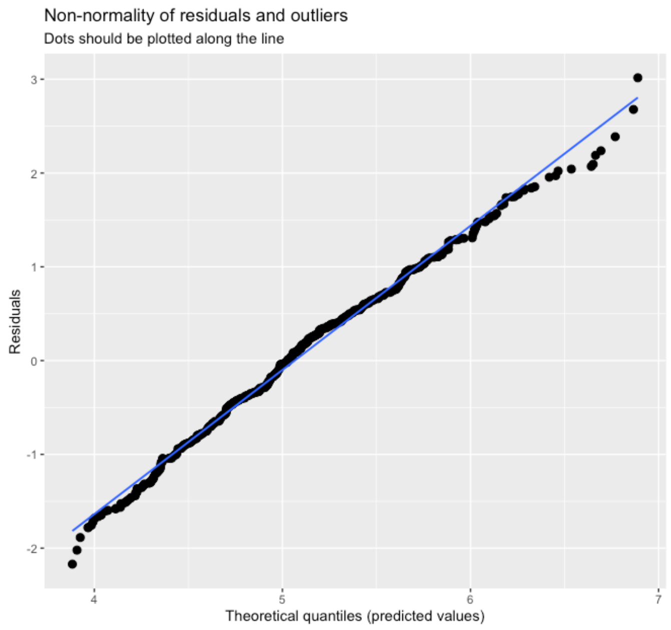
Assessing model fit

Assessing model fit for mixed models can be challenging, due to the multiple levels of random variation. To start with, we can look at the random effects diagnostics and residual diagnostics we used in the last lecture:

```
plot_model(epamod, type = 'diag')[[2]]
```



```r
plot_model(epamod, type = 'diag')[[1]]
```



The random effects look normally distributed, as do the residuals, so that's good.

Next we should consider how much variation there is from the different random components. We can get this info from summary(), or we can get it using the function VarCorr():

```
VarCorr(epamod)

##  Groups      Name        Std.Dev.
##  WaterbodyID (Intercept) 0.469
##  Residual                0.961
```

The variation among waterbodies has a st dev of 0.47, while the variation within waterbodies has a st dev of 0.96. This is similar to what we saw last lecture with a subset of this data: both sources of variation are important, but the variability within waterbodies over time is about twice as large as the variation across waterbodies. A residual standard deviation of 0.96 means that two observations from the same lake typically differ by a factor of exp(0.96) = 2.6.

R^2 for mixed models

Now we need to consider the fixed effects, and how much variation they explain. We'll get to the 'significance' of these effects shortly, for now let's think about how to assess the explanatory power of the different effects. Plotting the fitted effects is a good way to get a sense for this; the plots we've already made show that logTN has a fairly large effect but with much unexplained noise, while WaterbodyType has a weak effect. Ideally we would also have some kind of R^2 to tell us how much variation is explained. The difficulty is that we now have multiple sources of random variation in the model; so when we ask "how much variation is explained", do we mean variation of a particular kind, or 'all the variation'? And when we get to GLMMs this will be compounded with the fact that R^2 is hard to define for non-normal responses.

First let's think about explained variance for each random effect separately. In our example we have to variance components, the variance among waterbodies (WaterbodyID) and the variance within waterbodies (Residual). The standard deviation estimates for these are 0.46 and 0.96, respectively. One way to think about explained variance is two compare the full model to a model where one or more predictors is removed. For example, we can fit a model without WaterbodyType, and see what these variance estimates are:

```
epamod.noType = lmer(log(Cry_ero) ~ logTN + (1|WaterbodyID), data = crysub)
VarCorr(epamod.noType)

##  Groups      Name        Std.Dev.
##  WaterbodyID (Intercept) 0.473
##  Residual                0.963
```

It looks like the variance estimates are nearly unchanged. We can do it for TN as well:

```
epamod.noTN = lmer(log(Cry_ero) ~ WaterbodyType + (1|WaterbodyID), data = crys
ub)
VarCorr(epamod.noTN)

##  Groups      Name        Std.Dev.
##  WaterbodyID (Intercept) 0.694
##  Residual                0.954
```

It looks like TN does reduce the random effects variance, but the residual is not changed.

Eyeballing individual predictors is useful, but to get more quantitative, we can calculate an R^2 for the total effect of all the predictors on a variance term like this:

$$1 - \frac{\sigma_F^2}{\sigma_0^2}$$

Where σ_F^2 is the variance estimate in the full model, and σ_0^2 is the variance estimate in the model with no predictors. For the random effect WaterbodyID, this is

```
epamod.nopred = lmer(log(Cry_ero) ~ 1 + (1|WaterbodyID), data = crysub)


1 - VarCorr(epamod)$WaterbodyID[1]/VarCorr(epamod.nopred)$WaterbodyID
[1]

## [1] 0.5501
```

Or 55% of the variance explained. Note that when we look at the R output for VarCorr it returns the standard deviation, but when we extract it with `VarCorr(epamod)$WaterbodyID[1]` it returns the variance (SD^2), which is what we want. We can do the same thing for the residual variance:

```
1 - (sigma(epamod)^2)/(sigma(epamod.nopred)^2)

## [1] -0.01763
```

For the Residual variance I've used the function sigma(), which extracts the residual standard deviation from the model. So it looks like the predictors explain half of the variance for at the waterbody level, but none at the residual level.

This is an interesting result, because total nitrogen varies both within waterbodies and among waterbodies, but it looks like it only explains variation in density at the waterbody level. Let's go back and see how much total nitrogen varies at the two scales; we can quantify this using logTN as the response in a random effects model:

```
mod.TN = lmer(logTN ~ 1 + (1|WaterbodyID), data = crysub)
VarCorr(mod.TN)

##  Groups      Name        Std.Dev.
##  WaterbodyID (Intercept) 0.712
##  Residual                0.273
```

The variation among waterbodies is actually a fair bit larger than the variation within waterbodies. So even though there is seasonal variation in total nitrogen, it varies more across lakes, and so it makes sense that it does a better job of predicting variation in *Cryptomonas* across waterbodies.

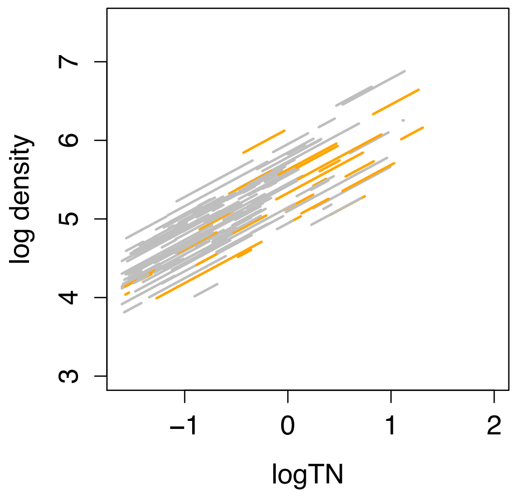
I was curious to better visualize how logTN varies among waterbody, and how that relates to the model predictions. I amended the code that plots a regression line for each waterbody, so that the predicted line for each waterbody only extends across the range of logTN values observed in that waterbody:

```r
#can we visualize how/why logTN reduces the variation among waterbodies?

datause = subset(crysub, WaterbodyID == levels(WaterbodyID)[i])

curve(waterbody.effects[1] + fixef(epamod)["logTN"]*x, from =
min(datause$logTN), to = max(datause$logTN), xlab = 'logTN', ylab = 'log
density', xlim = range(crysub$logTN), ylim = c(3,7.5), col = c('orange',
'grey')[waterbody.type[i]], lwd = 2)

for (i in 2:length(waterbody.effects)) {
  datause = subset(crysub, WaterbodyID == levels(WaterbodyID)[i])
  curve(waterbody.effects[i] + fixef(epamod)["logTN"]*x, from =
min(datause$logTN), to = max(datause$logTN), col = c('orange',
'grey')[waterbody.type[i]], add = T, lwd = 2)
}
```



There are still too many lines on this plot, but now we can see that logTN helps explain among-lake variation in density, because lakes with a higher mean density also have a higher mean TN.

I've shown how to assess explained variance for particular predictors and particular variance components. It is also possible to say 'how much of the total variation is explained by the fixed effects?', and 'how much of the total variation is explained by the fixed + random effects?'. These formulas were recently developed by Nakagawa and Scheilzeth:

$$R^2_{LMM(m)} = \frac{\sigma_f^2}{\sigma_f^2 + \sigma_\alpha^2 + \sigma_Y^2}$$

$$R^2_{LMM(c)} = \frac{\sigma_f^2 + \sigma_\alpha^2}{\sigma_f^2 + \sigma_\alpha^2 + \sigma_Y^2}$$

The formula on the top is 'marginal' R^2 for a linear mixed model. σ_f^2 is the variance predicted by the fixed effects; σ_α^2 is the variance for the random effects term; and σ_Y^2 is the residual variance. So this compares the variance predicted by the fixed effects to the variance from all sources of variation. If there were additional random effects terms, those would get added to the denominator. The formula on the bottom is 'conditional' R^2; this asks how much variation is explained by both the fixed and random effects, compared to the lowest-level residual variation. These formulas are not flawless, but they at least give a sense for whole-model performance. They can be calculated with a function in the MuMIn package:

```
r.squaredGLMM(epamod)

##    R2m    R2c
## 0.1726 0.3315
```

So for our model, the fixed effects (logTN and WaterbodyType) explain 17% of the total variation. If the random intercepts for WaterbodyID are also included, then 33% of the variation is explained. This means that in either case most of the unexplained variation is within waterbodies, which we already knew by looking at the variance estimates.

We can also compute this for restricted models, to get sense for the partial explanatory power of a predictor:

```
r.squaredGLMM(epamod.noTN)

##      R2m      R2c
## 0.004345 0.348819
```

When total nitrogen is removed, the remaining fixed effect (WaterbodyType) explains basically zero variation (as we saw before), but with the random effects the model still explains 35% of the total variation. This confirms that logTN can explain variation among waterbodies, but not lower level variation.

Null hypothesis tests and AIC with mixed models

Doing inference with mixed models is tricky, definitely trickier than the kinds of models we've used so far in this course. As a result, we're going to have to get into some esoteric details in order to use the best options available. I'll start by reviewing why inference with mixed models is challenging.

Effective sample size can be small / LRT is asymptotic. To do null hypothesis tests with GLMs, we mostly relied on likelihood ratio tests (LRTs). The derivation of the Chi-square approximation for the LRT assumes a large sample size (actually infinite, because the derivation is asymptotic as *N* -> Infinity). For most situations with GLMs this assumption is not very problematic, because it's only when sample size is pretty small (<10? < 15? hard to say) that the approximation can really break down. In this case, because we are assuming a large sample size (but we actually have a small one), the Type I error rate can be too high, leading to p-values and confidence intervals that are too small, and spurious results.

The issue with mixed models is that the *effective* sample size can often be quite small. For example, maybe we have survey data from 100 sites, but these sites are selected from 5 islands. If we have a predictor that varies at the island scale (such as island area), then we really only have 5 samples to test the effect of island area, and the individual sites are subsamples. If we used a Chi-square LRT to test the effect of island area, the results would be *anti-conservative*, which just means that the p-values would be too small.

Defining an F-statistic is tricky. Dealing with small sample sizes is not an uncommon issue in statistics. For linear models, the t-test and F-test are a kind of likelihood ratio test that appropriately accounts for the amount of data in the model. For mixed models, it is not obvious how to define an appropriate F-statistic, which in linear models has the form $F = \frac{\text{explained variance}}{\text{unexplained variance}}$. As we've already seen, defining the explained and unexplained variance in mixed models is tricky.

Counting parameters and degrees of freedom is not obvious. This is an issue that applies to both null hypothesis tests and AIC. In order to calculate an approximate F-test, we need to be able to count the number of parameters in the model, and the number of 'degrees of freedom' left over after those parameters are estimated. Likewise, for AIC we need to know the number of parameters in the model. How many parameters are estimated when we fit a mixed model? If we fit a random effect that has 20 groups, are we fitting one parameter (the variance among groups) or 20 parameters (the random effect for each group)? In addition, it seems like the answer to this question may depend on which predictor we're trying to test, and at which level(s) it varies.

The punchline of all these difficulties is that people use several methods to get p-values and information criteria for mixed models, but you should be cautious about taking the results too seriously, particularly if the effect only has moderate support. Here are some recommendations:

Null hypothesis tests for fixed effects: approximate F-test. For testing fixed effects predictors, such as logTN and WaterbodyType in the example, it is easy to get a Chi-square LRT for each component using the Anova() function in the car package, or using drop1(model, test = "Chisq"). However, these results will be anti-conservative (p-value is too small) when the predictor of interest varies across a small number of groups. Instead, we can use an approximate F-test that attempts to account for sample size in the hypothesis test. This can be done with the package 'lmerTest':

```
library(lmerTest)

anova(epamod, ddf = "Kenward-Roger")

## Analysis of Variance Table of type 3  with  Kenward-Roger
## approximation for degrees of freedom
##               Sum Sq Mean Sq NumDF DenDF F.value Pr(>F)
## logTN           89.6    89.6     1   394    97.3 <2e-16 ***
## WaterbodyType    3.3     3.3     1   342     3.6  0.059 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The package lmerTest provides an method for the anova() function, which calculates and approximate F-statistic, as well as approximate denominator degrees of freedom for this statistic. The degrees of freedom can be fractional (just like for GAMs), because it depends on how the parameters are counted, and this depends on the predictor and the structure of the data. I've told it to use the Kenward-Roger approximation, as this is thought to have the best Type 1 error properties. The F-test here is 'type 3', which means it is a marginal test that asks how much variation a predictor explains after the other predictors are accounted for. This also means that you should drop non-significant interactions from the model, if you want to test the main effects.

The results show that logTN is highly significant, which is not surprising, and that WaterbodyType is only marginally significant, which is also not surprising based on the plots we've looked at.

Null hypothesis tests for fixed effects: parametric bootstrap. An alternative approach to the approximate F-test is to use simulation to create an appropriate null distribution for the statistic you're using in your test. Since we've already covered simulation and how to use it to understand power and Type I error, this should be a natural extension. The process works like this:

- We fit our model, and calculate the likelihood for the fitted model.
- We fit a restricted model that removes a predictor of interest. We calculate the likelihood for that model.

- We want to see if the likelihood ratio of the two models is 'significantly' large. But we don't want to use the Chi-square approximation, because it assumes a very large sample size.
- Instead, we simulate from the restricted model many times, which is equivalent to simulating data under the null hypothesis. In each iteration we fit the full model and the restricted model to the simulated data, and calculate the likelihood ratio. This lets us ask 'If the null hypothesis were true, what would the distribution of the likelihood ratio look like? And is the real likelihood ratio significantly large, compared to the null distribution?'

This process is called a *parametric bootstrap*. It is a bootstrap because it relies on simulating new datasets to do inference, and it is parametric because it assumes the model assumptions are appropriate (when we simulate from the null model). The advantage is that we don't have to make additional assumptions to get an approximate F-test, or a chi-square test. If we wanted make our own parametric boostrap, we could use the simulate() function:

```
epamod.noType = lmer(log(Cry_ero) ~ logTN + (1|WaterbodyID), data = crysub, RE
ML = FALSE)

sim.null = simulate(epamod.noType)


null.full = lmer(sim.null[[1]] ~ logTN + WaterbodyType + (1|WaterbodyID), data
 = crysub, REML = FALSE)
null.restricted = lmer(sim.null[[1]] ~ logTN + (1|WaterbodyID), data = crysub,
 REML = FALSE)

logLik(null.full)

## 'log Lik.' -773 (df=5)

logLik(null.restricted)

## 'log Lik.' -773 (df=4)

-2*(logLik(null.restricted)[1] - logLik(null.full)[1])

## [1] 0.08931
```

Here I've made a null model (by removing WaterbodyType from the model). Then I simulated 1 set of values using the function simulate(). Then I fit the full model and the restricted (null) model to the simulated data. Then I calculate the likelihood ratio test statistic, which is $-2 * \log\left(\frac{L(restricted\ model)}{L(full\ model)}\right)$. For this iteration the statistic is 0.089. To finish the process, we would do this many times (e.g. 1000), to get a null distribution of the test statistic.

Luckily kind souls have made a package to do this with lmer models:

```
epamod = lmer(log(Cry_ero) ~ logTN + WaterbodyType + (1|WaterbodyID), data = c
rysub, REML = FALSE)
epamod.noType = lmer(log(Cry_ero) ~ logTN + (1|WaterbodyID), data = crysub, RE
ML = FALSE)


library(pbkrtest)
PBmodcomp(epamod, epamod.noType)
```

```
## Parametric bootstrap test; time: 28.81 sec; samples: 1000 extremes: 55;
## large : log(Cry_ero) ~ logTN + WaterbodyType + (1 | WaterbodyID)
## small : log(Cry_ero) ~ logTN + (1 | WaterbodyID)
##          stat df p.value
## LRT       3.6  1   0.058 .
## PBtest    3.6      0.056 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The PBmodcomp function performed 1000 simulations (you can specify how many), which took 29 seconds on my laptop. The simulation-based p-value is 0.056, which is a little smaller than what the Kenward-Roger F-test, or the Chi-square LRT, calculates. For this model the kind of test we use doesn't matter very much, because there are a large number of groups (320) and a large number of observations (533). In the next example we do, we will see a case where small sample sizes become important.

The upshot of the parameteric bootstrap is that it requires less assumptions. The downside is that it can be slow, especially when we get to GLMMs, and the results will not be identical if you perform the test multiple times, because it is based on simulation. In general the Kenward-Roger F-test is usually a fine approach, but when we get to GLMMs we won't have this option.

REML vs. ML. When we fit a model with lmer, it says 'linear mixed model fit with REML'. REML stands for restricted maximum likelihood. It is a correction to the normal maximum likelihood estimation, because it is known that maximum likelihood tends to underestimate the magnitude of the random effects variances. Hence REML is the default. *However, if you want to compare models that have different fixed effects, for example with a likelihood ratio test or AIC, you need to specify REML = FALSE when you fit the model with lmer*. Note that for the PBmodcomp code above, I set REML = FALSE. Different models fit with REML cannot be compared in terms of their likelihoods.

AIC with mixed models. Using AIC with mixed models is a matter of contention. It is certainly done, and I've done it myself, but there are definitely issues. The main issue is that it's not clear how to count parameters. If you use AIC() or AICc() on an lmer object, it will define K = # fixed effect parameters + # variances for random effect terms. As I said when explaining random effects models, these are technically the only parameters that the model is fitting. Nonetheless it seems like the # of groups for the random effects should matter for model comparison, especially if we are interested in these particular groups, as opposed to treating the grouping factor as a nuisance variable. There are various alternative information criteria that try to take account of these issues (e.g. DIC), but none of them has emerged as a consensus alternative and so I won't cover them in this course. So I recommend using AICc if you want to use the information theory approach for model selection, but be aware that it tends to be anti-conservative in the same way that the likelihood ratio test is anti-conservative.

We can compare AICc for the full model I've been fitting, plus models that remove the fixed effects predictors:

```
AICc(epamod)
```

```
## [1] 1582
```

```
AICc(epamod.noTN)
```

```
## [1] 1664
```

```
AICc(epamod.noType)
```

```
## [1] 1583
```

This is similar to the other tests we've used. Total nitrogen clearly has a huge effect on AIC, but removing WaterbodyType only increases AIC by 1. So there is not strong evidence that WaterbodyType is an important predictor, but there is also not strong evidence against.

<u>Testing random effects</u>. So far I've talked about testing fixed effects predictors. In some cases you might want to test the importance of a random effect term. Often this isn't necessary, or a good idea, because random effects terms are often used in models to account for non-independence in the data, as opposed to being an interesting predictor. This is the case for our example; accounting for among-lake variation is necessary, in order to correctly model the role of other predictors. However, there are cases when the random effect represents a hypothesis and you want to see what evidence there is for that hypothesis.

Testing random effects adds more complexity to the mix, because the rules of thumb for fixed effects don't apply. Likelihood ratio tests tend to be anti-conservative for fixed effects, so I recommended using a parametric bootstrap or F-test. In constrast, likelihood ratio tests tend to be *conservative* when testing random effects. This means that p-values tend to be too high, leading to reduced power. The cause has to do with the fact that the null hypothesis is that the random effects variance is 0, but it's not possible for a variance to go below zero. The upshot is that you can use a Chi-square LRT, and you will know that your answer is conservative. This is often the recommended approach. The alternative is to use a parametric bootstrap; the advantage is again that fewer assumptions are necessary. Finally, you can use AIC, but like the LRT it tends to be conservative.

For our example model, there is only one random effect, which adds an extra complication. We can't compare the likelihood of a lmer model and an lm model that removes the random effect; the reason is that the likelihood is not calculated the same way for the two functions, so you can only compare models fit using the same function. But we can still do a parametric bootstrap, using the package RLRsim:

```
library(RLRsim)

epamod = lmer(log(Cry_ero) ~ logTN + WaterbodyType + (1|WaterbodyID), d
ata = crysub, REML = TRUE)

exactRLRT(epamod)

## 
##  simulated finite sample distribution of RLRT.
## 
##  (p-value based on 10000 simulated values)
## 
## data:
## RLRT = 8.937, p-value = 0.0013
```

The function exactRLRT uses the REML fit to do 10000 parametric bootstrap simulations from a null model where the variance for WaterbodyID equals zero. It uses these to generate a null distribution for the likelihood ratio test statistic. In this case it finds that the probability of finding the observed likelihood ratio between the full model and the null model is 0.0013, so the random effect term is clearly significant. As I said earlier, in this case there is no reason to test the significance of this random effect, because we want it to be present in the model regardless, to account for the hierarchical structure of the data. But I'm showing how to test it here as an example.