Chapter 12
Beyond ANOVA, regression, and chi-squared: Mixed-effects modeling

James Myers
2022/5/13 DRAFT

1. Introduction

As we've seen, regression modeling lies pretty close to the heart of statistics: *t* tests and ANOVA are really linear regression, and chi-squared tests are closely related to logistic regression. But most of the regression models we've worked with assume that each observation is independent of all of the others. How should we incorporate the grouping of data by units? The traditional method was to use **repeated-measures regression**, which, as we've seen, is closely related to repeated-measures ANOVA and paired *t* tests. In this method, we first create regression models for each grouping unit (e.g., experimental participant) and then run one-sample *t* tests on the coefficients across the models. In other words, when we partial out the **random effects** of the grouping units from the **fixed effects** represented the coefficients, we do it in separate steps.

This approach works fine for many purposes (including for categorical data, in the form of **repeated-measures logistic regression**), but it looks like its days are numbered. More and more experimental linguists are giving up ANOVA and traditional linear and logistic regression and adopting a method called **mixed-effects modeling** (sometimes also called **multi-level modeling**). Unlike mixed ANOVA, what is being mixed in mixed-effects modeling is not just within-group and between-group variables, but more crucially, the fixed and random effects. That is, rather than partialling the fixed and random effects out in separate steps, they are instead processed together in a single model. Not only is this mixed-effects approach more flexible (for more complex mixings of random and fixed variables), but it also makes it possible to handle more than one random variable at the same time, thus providing a solution to the language-as-fixed-effects fallacy of Clark (1973). It also deals with other annoyances of ANOVA: regression-based approaches aren't affected as much by violations of sphericity or unbalanced data (i.e., unequal cells sizes).

Open up an experimental linguistics paper today, and you are more and more likely to see the authors using **linear mixed-effects modeling** (**LME**) or **mixed-effects logistic regression** or other types of **generalized linear mixed models** (**GLMM**). It's even possible to combine the power of mixed models with the nonlinear generalized additive models (GAM) mentioned at the end of the last chapter, producing something called **generalized additive mixed-effects modeling** (**GAMM**). All of this is thanks to the increasing power of personal computers, required for running the estimation algorithms used by LME and GLMM and GAM and

GAMM. As with the older estimation algorithms for logistic regression, though, these algorithms are prone to crashing, and they can also take a very long time to finish running (hours or even days, for large data sets and complex models).

But Spiderman's words of wisdom still apply: with great power comes great responsibility. As Matuschek et al. (2017, p. 305) admit, "The advantages of LMMs [linear mixed-effects models] over ANOVAs come at a cost; setting up an LMM is not as straightforward as running an ANOVA." Barr et al. (2013, p. 277) put it more vividly: "At a recent workshop on mixed-effects models, a prominent psycholinguist [G. T. M. Altmann] memorably quipped that encouraging psycholinguists to use linear mixed-effects models was like giving shotguns to toddlers." In his own R-for-linguists book, Gries (2013) restricts his discussion of mixed modeling to a few skeptical comments (p. 335):

> [M]ixed-effects modeling is one of the most fascinating but also among the most complex statistical techniques I have seen.... [I]ndeed the potential of this approach is immense and far-reaching [but] I must admit that sometimes I think that some of the hype about this method is a bit premature simply because so many thing are still unclear. Ask any two or three experts on how to do X with multi-level models, and you often get very different responses. Pick any two to three references on mixed-effects modeling and you will see that not only is there very little agreement on some seemingly central questions, but also that some types of problems are not even mentioned very widely.

With these warnings in mind, let's see how we can make mixed-effects modeling work for us!

2. Mixing effects

Even though it took powerful computers to make it practical, mixing fixed and random effects is actually a simple idea; even our friend the genius statistician Fisher (inventor of ANOVA) was thinking along these lines. In this section we'll first see how mixed-effects modeling is related to, but differs from, repeated-measures regression, try it out on some linguistic data, deal with some complexities (including the surprising fact that even the experts can't agree on the best way to compute the *p* values), and survey some of the ways we can plot mixed-effects models.

2.1 Beyond repeated-measures regression

Before we start mixing the fixed and random effects, let's first review repeated-measures regression, where the fixed and random effects are processed in separate steps.

Consider the (fake) data in **sploink.txt**, which come from a longitudinal study on (fake) Martian children. The research question is: as children get older (from two to four years), does their (fake) "sploink" measure increase? This sounds like a job for linear regression, since both the independent and dependent variables are numerical (and if sploink is a normal sort of measure, it's probably normally distributed).

Indeed, if we plot the relation between Age and Sploink, it looks like there's a positive correlation (as in Figure 1):

```
sdat = read.delim("sploink.txt")
plot(sdat$Age, sdat$Sploink)
abline(lm(Sploink~Age, data=sdat))
```

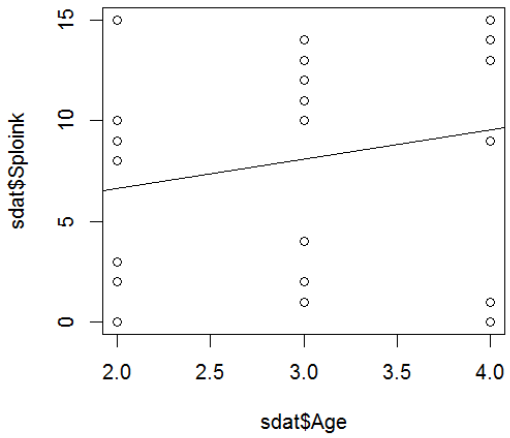

Figure 1. Sploink increases with age

But if we run an ordinary linear regression analysis on the data, nothing is significant, as shown in Table 1.

```
summary(lm(Sploink ~ Age, data = sdat)) # Just showing fixed coefficients below
```

Table 1. Regression table produced by R for Sploink ~ Age, ignoring grouping by Child

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	3.750	3.778	0.993	0.329
Age	1.450	1.215	1.193	0.243

If we make a **trellis** (網格) plot, with a separate plot for each child, we can see the problem: while most children clearly show sploink rising with age, some children don't. We can make this plot a number of different ways, as shown in Figures 2-4.

```
# Option 1: Do it with base R:
par(mfrow=c(2,5)) # Put 10 plots (one per child) into 2 rows and 5 columns
miny = min(sdat$Sploink); maxy = max(sdat$Sploink)
for (i in 1:10) {
  sdat.i = subset(sdat, sdat$Child==i)
  plot(sdat.i$Age, sdat.i$Sploink, ylim = c(miny, maxy), main = i)
}
```

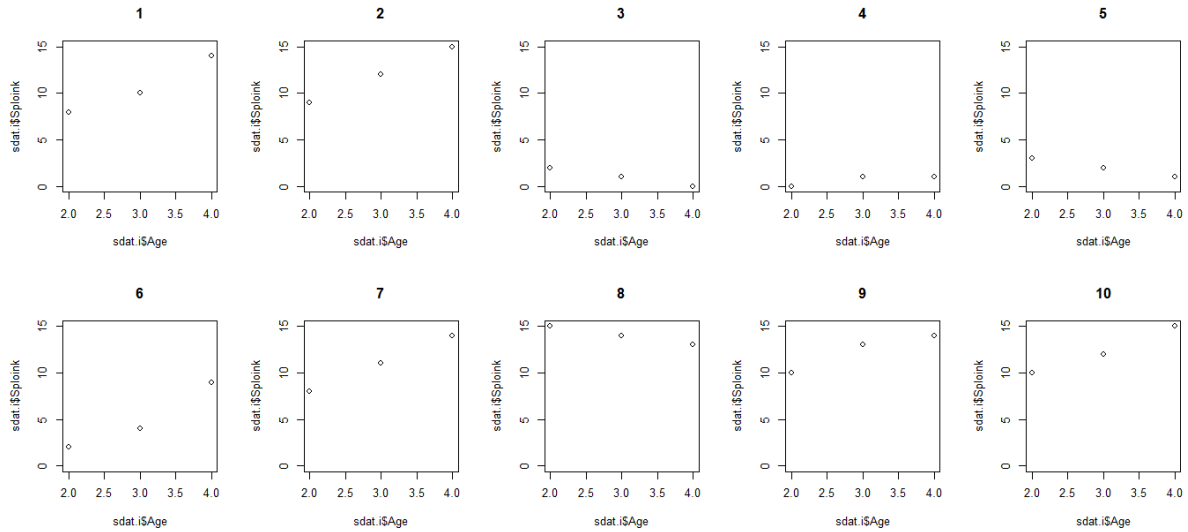

Figure 2. By-child plots using base R

```
# Option 2: Do it with the lattice package:
library(lattice) # You first have to make sure this package is installed, of course
xyplot(Sploink ~ Age | factor(Child), data = sdat)
```

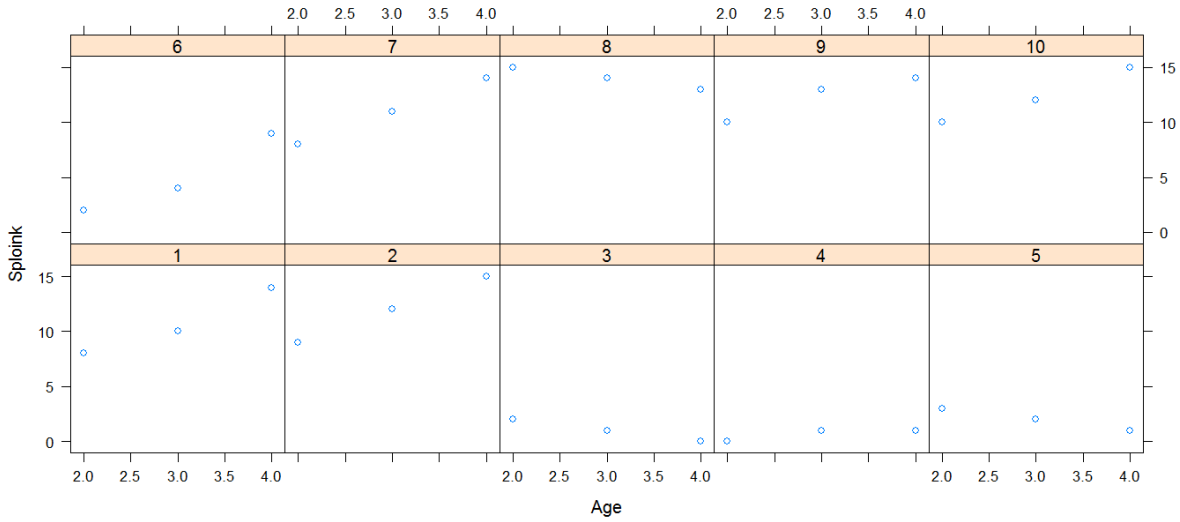

Figure 3. By-child plots using **lattice** package

```
# Option 3: Do it with the ggplot2 package:
library(ggplot2) # You first have to make sure this package is installed, of course
qplot(Age, Sploink, data = sdat, facets = ~Child)
```

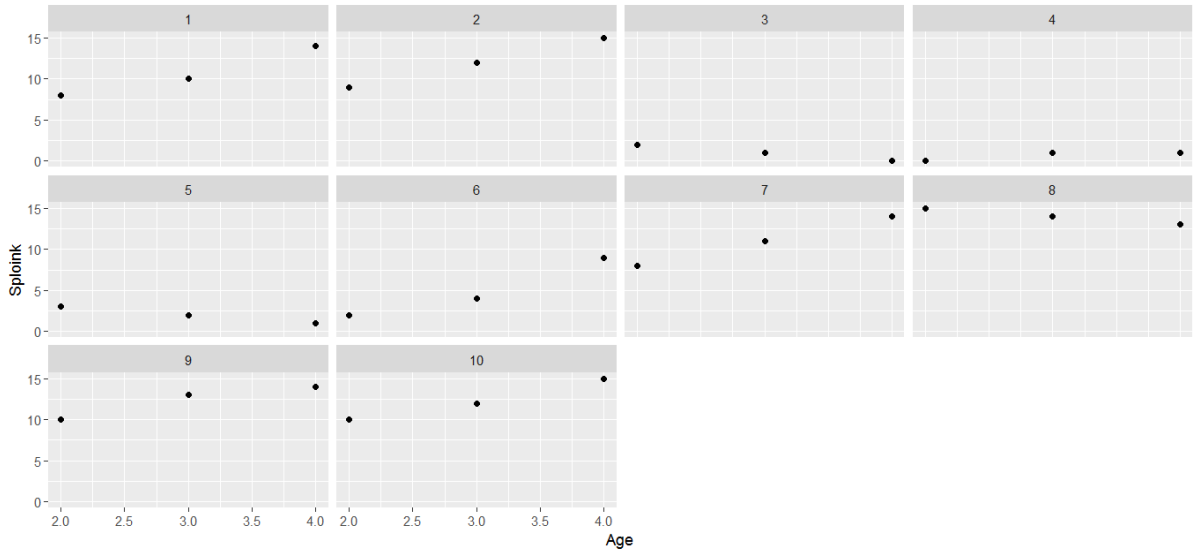

Figure 4. By-child plots using **ggplot2** package

This result isn't surprising, since the data here are repeated-measures: each child gives three sploink values (one per age in that child's data: 2, 3, 4). So let's do a repeated-measures regression. In the first step, we run a regression on each child separately and get the coefficient for the intercept (overall sploink level for that child) and the coefficient (slope) for the predictor age. In the second step, we run one-sample *t* tests on each of these sets of regression coefficients.

Here's a new version of the R code we used in the multiple regression chapter. Note that running it generates the warning "essentially perfect fit: summary may be unreliable" because each of the individual child analyses contains so few data points (just three) that the regression analysis gets suspicious (maybe you remember that the algorithm totally crashes if you try to simulate a paired *t* test using repeated-measures regression). They're just warnings, though, so let's just ignore them so we can finish this example:

```
n = length(unique(sdat$Child)) # Number of children
b0 = numeric(n) # For the by-unit intercepts (empty for now)
b1 = numeric(n) # For the by-unit coefficients for Age (also empty at the start)
for (i in 1:n) {
  lm.i = lm(Sploink~Age,data=subset(sdat,sdat$Child==i))
  b0[i] = summary(lm.i)$coefficients[1,1] # Put in child i's intercept coefficient
  b1[i] = summary(lm.i)$coefficients[2,1] # Put in child i's coefficient for Age
}
# Mean = coefficients; also gives df, t, and p
t.test(b0) # Mean = coefficients; also gives df, t, and p
```

```
        One Sample t-test

data:  b0
t = 2.0554, df = 9, p-value = 0.07
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.3771502  7.8771502
sample estimates:
mean of x
     3.75
```

as.numeric(mean(b0)/t.test(b0)$statistic) # Gives SE

```
[1] 1.824431
```

 # Slope results:
t.test(b1) # Mean = coefficients; also gives df, t, and p

```
        One Sample t-test

data:  b1
t = 2.4422, df = 9, p-value = 0.03723
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.1069186 2.7930814
sample estimates:
mean of x
     1.45
```

as.numeric(mean(b1)/t.test(b0)$statistic) # Gives SE

```
[1] 0.7054467
```

Copy/pasting the text reports into the R-style regression report in Table 2, we see that repeated measures regression gives us exactly the same coefficients as ordinary regression (compare with Table 1 above), but the *SE* values are lower, since we've factored out the noise of cross-Child variation. This makes the *t* values higher and the *p* values lower. In fact, Age is now found to have a significant effect.

Table 2. A hand-made R-style regression table for Sploink ~ Age, grouped by Child

	Coefficient	SE	t	df	p	
(Intercept)	3.75	1.824431	2.0554	9	0.07	.
Age	1.45	0.7054467	**2.4422**	9	**0.03723**	*

Now see what we get when we analyze the same data as a repeated-measures ANOVA, treating Child as a grouping variable. This actually an ANCOVA (analysis of covariance),

since the predictor is a numerical variable, not a factor. The results are in Table 3. Note the highlighted values and compare them with the highlighted values in Table 2.

summary(aov(Sploink ~ Age + Error(factor(Child)/Age), data = sdat))

Table 3. Repeated-measures AN(C)OVA for Sploink ~ Age, grouped by Child

Error: factor(Child)

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
Residuals	9	760	84.45			

Error: factor(Child):Age

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
Age	1	42.05	42.05	**5.965**	**0.0372**	*
Residuals	9	63.45	7.05			

Error: Within

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Residuals	10	3.167	0.3167		

I hope you are not surprised to see (since this is just a review of stuff we've discussed before) that $F = 5.965$ for Age in the repeated-measures ANOVA is the square of $t^2 = 5.964341$ for Age in the repeated-measures regression, and the p values are identical.

We can extract the significant effect of Age because we've also pulled out the systematic effect of the cross-Child differences. The repeated-measures regression expresses this as the intercept (reflecting the "default" response of each child), and the repeated-measures ANOVA expresses this with the first ANOVA report table in Table 3 labeled "Error: factor(Child)". A second reason for the increased power compared to ordinary regression is that the Age effect is modeled as an interaction between Child and Age. The second ANOVA report table in Table 3 expresses this explicitly with the label "Error: factor(Child):Age" (":" = "×" here), and in the repeated-measures regression reported in Table 2, the t value for Age is actually an analysis across Child for each child's Age effect.

Now let's compare this repeated-measures analysis with a mixed-effects analysis, which processes the random effect of Child and the fixed effect of Age in the same model, instead of processing them in separate models and steps.

The oldest R package for mixed-effects modeling is **nlme** (Pinheiro & Bates, 2000), which stands for "non-linear mixed-effects" (since it also handles loglinear models like mixed-effects logistic regression). Few researchers use this package anymore (but see Johnson, 2008, for a simple tutorial), since it has been superseded by the much more powerful **lme4** package (Bates et al., 2015), which gets its weird name from the fact that it was originally inspired by version 4 of S (remember that this is R's original, non-free, version). We'll be using **lme4** in most of this chapter, but I have a pedagogical reason to start our discussion with **nlme**.

So let's install and load the **nlme** package:

```
library(nlme)
```

The key function is **lme()** (for "linear mixed effects"). The syntax of this function expresses the fixed part of the analysis using the familiar formula syntax, and the random part using the **random** argument, which has the syntax **~WithinVariable | GroupingUnit** (reverse of **aov()**'s **Error(GroupingUnit/WithinVariable)** syntax). In our case, the grouping unit is Child (which **lme()** is smart enough to convert to a factor for us, unlike **aov()**) and the within-group variable is Age. Thus we can build and summarize the model like this:

```
sdat.lme = lme(Sploink~Age, random = ~Age|Child, data=sdat)
summary(sdat.lme)
```

The summary includes a lot of stuff, but let's just look at the part we most want to see (since we're going to redo all this with the **lme4** package anyway). That of course is the regression table for the fixed effects (i.e., the overall intercept and Age effect), as shown in Table 4. Note that the coefficients and *t* values are exactly the same as repeated-measures regression (compare with Table 2), but the *p* values are lower due to the lower *SE* and higher *df* values:

Table 4. Mixed-effects model of Sploink ~ Age, grouped by Child, using **lme()** in **nlme**

Fixed effects: Sploink ~ Age

	Value	Std.Error	DF	t-value	p-value
(Intercept)	3.75	1.8244313	19	2.055435	0.0538
Age	1.45	0.5937171	19	2.442241	0.0245

The *df* values for repeated-measures regression in Table 3 were based on the number of grouping units, using the usual one-sample *t* test *df* formula, where *g* = number of grouping units (in our case, 10 kids, so *df* = 9 in Table 3):

One-sample *t* test *df* formula: $df = g - 1$

By contrast, the *df* values used by the **lme()** function are calculated using the formula below, from Pinheiro & Bates (2000), where *N* = total observations (30 for the sploink data), *g* = number of grouping units (e.g., 10 children in the sploink data), and *k* = number of parameters (1 in the sploink data: Age). So the value that has the greatest effect on *df* is *N*. This *df* formula results from the one-step approach taken by LME: Pinheiro & Bates (2000) argue

that this means that there should be a single *df* for all fixed effects (see discussion about this in Bates, 2006).

lme() *df* formula: $df = N - g - k$

The summary for **lme()** also includes the standard deviations of the random intercept (5.6351) and random slope (1.8348), which are suspiciously close to the standard deviations for the intercept and Age effect in the repeated-measures regression (**sd(b0) ==** 5.769, **sd(b1) ==** 1.877). The **lme()** summary reports the standard deviation for the residuals as well (0.5627). Since repeated-measures regression is the same as repeated-measures ANOVA, and ANOVA is based on variance (calling it *MS* for mean sum of squares), and variance is just the square of standard deviation (s^2), we can compare these three values across the ANOVA and LME reports, as shown in Table 5. These "noise" values in the LME for the Child intercepts and Child × Age interactions (random slopes) are lower than those in both repeated-measures regression and ANOVA.

Table 5. Partialling variance in ANOVA and LME

	ANOVA		LME		
Component	Label	**MS (s^2)**	Label	*s*	**s^2**
Cross-Child means for Sploink	Error: factor(Child) Residuals	**84.45**	Random effects: (Intercept)	5.635	**31.755**
Cross-Child slopes for Age	Error: factor(Child):Age Residuals	**7.05**	Random effects: Age	1.835	**3.367**
Overall residuals	Error: Within Residuals	**0.3167**	Random effects: Residual	0.563	**0.3167**

Less noise means more power: LME extracts more information from the sploink data than repeated-measures ANOVA.

2.2 More about the math of mixed-effects modeling

Maybe the easiest way to think about how mixed-effects modeling works is to think about how you could fake some realistic data. Suppose you're teaching a statistics class and want to make a data set that's supposed to come from a study on 10 Martian children, each producing sploink values at three different ages. How could we make this data set seem realistic? If you think about it carefully, you can see that each sploink value actually depends on *six* different types of things: totally random noise, the default sploink value (when age = 0), the overall

effect of age, each child's default sploink value (i.e., whether that child is a low or high sploinker in general), each child's sensitivity to the fixed effect (i.e., whether that child sploinks equally at all ages, or more when young, or more when old), and random noise in the behavior of that individual child.

Each child's default value and sensitivity to age are called the **random intercept** and the **random slope**, respectively. This is because, just as in repeated-measures regression (or ANOVA or ANCOVA), conceptually each child has his or her own regression equation, like this:

Equation for each child *i*: $\text{Sploink} = \beta_{i0} + \beta_{i1}\text{Age} + \text{Error}_i$

In a repeated-measures analysis, as we've seen, we fit separate models like this to every grouping unit (here, each child), and then run one-sample tests across all of the random intercepts (β_{i0}) and random slopes (β_{i1}) to see if the overall values are significantly different from the null hypothesis of 0. Thus the repeated-measures regression/AN(C)OVA approach actually involves two different levels of regression equations, one dealing with the fixed effects (like Age) and one dealing with the random effect (like Child).

By contrast, in a mixed-effects analysis, we go back to the original concept and just add up all six things together in one equation (hence the names "mixed-effects" modeling or "multi-level" modeling). In the equation below, the *B*s represent the fixed intercept and slope, the betas represent the random intercepts and random slopes, and the *i*s represent the kids:

Mixed equation for all data: $\text{Sploink} = B_0 + B_1\text{Age} + \text{Error} + \beta_{i0} + \beta_{i1}\text{Age} + \text{Error}_i$

Because of the two different kinds of coefficients, there's no simple formula for computing them. Thus just as with logistic regression, we have to loop through many iterations, incrementally adjusting the coefficient values up and down until the algorithm decides that our fit isn't getting any better, or that we've run out of our preset loops.

The challenge posed by mixed-effects modeling is even harder than for logistic regression, though, because of the multiple sources of random error. This means we can't use the kind of maximum likelihood estimation (MLE) used for logistic regression, but instead must use an approximate version called **restricted**, **residual**, or **reduced maximum likelihood** (all abbreviated as **REML**). The trick used by REML is to model only the crucial contrasts instead of all of the raw data all at once; that is, it cheats, kind of how *minF'* cheats by estimating *F'* under artificially simple assumptions (cheating is common in statistics!).

Here's another thing you should know about those random slopes and intercepts. It doesn't make sense to talk about the fixed intercept and slope being correlated, since there's only one overall best-fit line, but since there are many of grouping units, their random slopes and

intercepts may indeed be correlated. For example, suppose that the mean sploink value for every child is the same, but the children differ in their random intercepts and random slopes. If you think about this geometrically, this implies that even if some best-fit lines are flat, others are rising, while others are falling, they will still all cross each other somewhere in the middle (the mean on the *y*-axis). This in turn implies that the random intercepts and random slopes are correlated; for example, for observed *x* and *y* values all above zero, a rising best-fit line means that the intercept is low, and a falling line means that the intercept is high (can you picture it?). This adds another bit of complexity to mixed-effects modeling: the potential collinearity among the random coefficients makes it even more difficult to figure out their "true" values.

Such complexities are why Fisher's old multilevel dreams had to wait until computers got fast enough to run the estimation algorithms. If you want to learn more about the math, take a look at Pinheiro and Bates (2000) or Vasishth and Broe (2011); the latter is a bit less technical.

2.3 The lme4 package and the great *p* value debate

Now let's do this exact same LME analysis using the more powerful and popular **lme4** package. Once we've installed it on our computer, we have to load it:

```
library(lme4) # That's right, you gotta install it first
```

Now the key function is **lmer()** (where the "r" stands for "regression" or "the R language"; its inventors say it's pronounced like the English male name "Elmer"). The syntax is a bit different from that of **lme()**, incorporating the random parts into the formula notation itself, to reflect the fact that fixed and random effects are modeled together. Now the random part is expressed in the formula inside parentheses, as **(WithinVariable|GroupingUnit)**. So in the case of the sploink data set, this part of the formula would say **(Age|Child)**. Again, there's no need to tell **lmer()** that Child is actually a categorical factor.

Let's try it out:

```
sdat.lmer = lmer(Sploink ~ Age + (Age|Child), data = sdat) # No need for as.factor!
summary(sdat.lmer)
```

The results are shown in Table 6, again focusing for now just on the coefficients table for the fixed effects.

Table 6. Mixed-effects model of Sploink ~ Age, grouped by Child, using **lmer()** in **lme4**

Fixed effects:

	Estimate	Std.Error	t value
(Intercept)	3.7500	1.8245	2.055
Age	1.4500	0.5937	2.442

Note that the fixed effects table gives the same coefficients as we got with **nlme**'s **lme()** function (see Table 4) above, as well as the same *SE* and *t* values.

But wait a minute! Where are the *p* values?

Welcome to the world of cutting-edge statistics! It turns out that the way *p* values are calculated in the **nlme** package is controversial, because the degrees of freedom (*df*) are controversial (they're also missing in Table 6). Why? Well, both of R's LME packages were (co)written by one guy, Douglas Bates, and his approach to *df*, expressed in Pinheiro & Bates (2000), conflicts with the default originally assumed by SAS, a highly influential "not-free" statistics program. The current version of SAS actually allows the user to choose among five competing *df* values (Bell et al., 2013)! This is because it's not clear whether we should use the same *df* for everything (as argued by Pinheiro & Bates, 2000, and assumed by **nlme**), or vary *df* depending on the particular fixed and/or random variables that we're testing.

So how can we get "uncontroversial" *p* values using the **lmer()** function? Here are a few different ways.

The simplest but least reliable method is to pretend that the *t* values are actually *z* values, so we don't need to worry about *df*. After all, logistic regression uses *z* values regardless of sample size, so why not here? We can calculate these *p* values using **pnorm()**:

```
intercept.p = 2*pnorm(-2.055) # 0.039879
Age.p = 2*pnorm(-2.442) # 0.01460615
```

The sploink sample is very small, so the *p* values here are definitely too low (a Type I error); they're even lower than those given by **nlme**'s **lme()** function. But since $t \rightarrow z$ as $n \rightarrow \infty$, this method should be OK for large samples, especially if Pinheiro & Bates (2000) are basically right to claim that the major influence on *df* is the total number *N* of data points, not the number of grouping units or the number of model parameters, which will always be much smaller than *N*.

A second method is slightly harder: we use the **anova()** function to run a likelihood ratio test on our full model vs. models missing each parameter. This is pretty easy to do with a simple model like the one for sploink, producing the results in Table 7:

```
sdat.lmer.noAge = lmer(Sploink ~ 1 + (Age|Child), data = sdat)
anova(sdat.lmer.noAge, sdat.lmer) # Tests significance of Age
```

Table 7. Using a likelihood ratio test to test the significance of Age

	Df	AIC	BIC	logLik	deviance	Chisq	Chi Df	Pr(>Chisq)	
sdat.lmer.noAge	5	148.30	155.31	-69.150	138.30				
sdat.lmer	6	145.22	153.62	-66.608	133.22	5.0846	1	0.02414	*

But the third and best method is to install the package **afex** (pun for "effects"; Singmann, 2014). This package automatically loads **lme4** for you; in fact, the creators recommend loading **afex** first, especially if your version of **lme4** is out of date. If you've been following along with the R code so far, you're already running **lme4**. So let's first unload it:

```
detach("package:lme4", unload=TRUE)
```

Now we load **afex**:

```
library(afex) # Install it first, and let it load lme4 for you
```

Now when we use the **lmer()** function, the **afex** package will compute *p* values for us (technically this is done via the **lmerTest** package [Kuznetsova *et al.*, 2017], run by the **afex** package), by default using something called the **Satterthwaite approximation** (Satterthwaite, 1946), which, as you can guess from the publication date, was invented long before mixed-effects modeling. In fact, it's related to the Welch test that we used for unpaired *t* tests without assuming equal variance (hence the follow-up on related methods in Welch, 1947). Basically, it takes the *df* used by Pinheiro & Bates (2000) and lowers it, in an attempt to avoid Type I errors. Here we go:

```
sdat.lmer.S = lmer(Sploink ~ Age + (Age|Child), data = sdat)
```

Using the **summary()** function on this gives us the usual **lmer()** report, but now with *df* and *p* values, as shown in Table 8 (note also the text at the start of the R output saying that Satterthwaite's method was used):

```
summary(sdat.lmer.S)
```

Table 8. Using the Satterthwaite approximation to test the significance of Age

```
Fixed effects:
             Estimate  Std.Error      df  t value  Pr(>|t|)
(Intercept)   3.7500     1.8245  8.9996    2.055    0.0700  .
Age           1.4500     0.5937  8.9996    2.442    0.0372  *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The *df* values of 8.9996 are adjusted quite a bit downward from what the simple Pinheiro & Bates (2000) formula would give us ($df = N - g - k = 30 - 10 - 1 = 19$). The *p* value is also slightly higher than what we got with the likelihood ratio test, but it's still significant anyway.

We could report the Age effect as $B = 1.45$, $t(9) = 2.4$, $p < .05$ (clarifying in the text of our report that the *p* value is based on the Satterthwaite approximation).

Besides the Satterthwaite method, the **afex** package also gives you other ways to estimate *p* values for linear mixed-effects models, including an automatic application of the likelihood ratio test, the **Kenward-Roger approximation** (Kenward & Roger, 1997; Judd et al., 2012), which uses the Satterthwaite method but besides adjusting the *df* also adjusts the *F* ratio (which is related to the *t* value, as you remember), and a **parametric bootstrap** method that estimates *p* values by resampling many times from a normally distributed null hypothesis population specified by your data and model formula. If you ever want to try these non-defaults, use the special **afex** function **mixed()**, which works sort of like **lmer()** except that it lets you include the argument **method** (**method="S"**, **method="LRT"**, **method="KR"**, **method="PB"**). Other differences are that it gives you *p* values for the effects (though not the intercept) directly, without using **summary()**, and that they're computed (again via the **lmerTest** package) an ANOVA (or even analysis of deviance) table rather than a regression table (recall from the multiple regression chapter how these approaches are related).

But let's not bother with these other options, for three reasons. First, the Satterthwaite approximation is the most sensitive (i.e., it gives the lowest *p* values, though this potentially brings a risk of Type I errors), and it is also the most flexible, applicable not just to REML but also to ML (i.e., the ordinary maximum likelihood logic used in ordinary regression and stuff like *t* tests - hence the Welch link). The Kenward-Roger approximation and bootstrap method may be slightly more accurate, but the former is somewhat harder to report as a regression (being based on *F* instead of *t*) and the latter can run very slow (resampling requires a lot of looping and recalculating). Second, getting the output in an ANOVA table is less useful than getting it in a regression table, since the ANOVA table doesn't report results for the intercept or show the coefficients, which, as we've seen, tell us something about effect direction and size. Third, and most important, if your sample is large and normal enough, which it should be anyway when doing parametric statistics, any reasonable way to compute the *p* values should give you pretty much the same result, unless your *p* values are just at the edge of significance (e.g. $p = .049$), but in that case you should be cautious in interpreting your results anyway: the alpha level is just an arbitrary convention, not something out there in the real world.

2.4 Plotting mixed-effects models

One natural way to plot an LME is to use the trellis plots we saw at the beginning of this chapter. But this can take a lot of space (a separate plot for each participant) and requires the viewers to find the overall pattern themselves.

If you want a single plot for the entire data set, it's often safe to use the best-fit line for an ordinary linear regression, since this tends to be similar to that for an LME analysis of the same

data, as we saw with the sploink example. However, this isn't guaranteed. You can get a sense of how these best-fit lines can differ by simulating a bunch of random data sets for y~x and comparing the two estimates of the x slope coefficient, as in Figure 5.

```
set.seed(1) # So you get the same results as I do
g=rep(1:10,10) # 10 grouping units for 100 data points in each data set
lm.coefs = numeric(20) # space for 20 ordinary linear model coefficients
lme.coefs = numeric(20) # space for 20 linear mixed-effects model coefficients
for (i in 1:20) { # Create 20 random data sets
  dat = data.frame(y=rnorm(100), g, x=runif(100))
  lm.i = lm(y~x, data = dat) # Ordinary linear model
  lm.coefs[i] = summary(lm.i)$coefficients["x",1] # lm coefficient for x
  lme.i = lmer(y~x + (x|g), data = dat) # Mixed-effects model
  lme.coefs[i] = summary(lme.i)$coefficients["x",1] # lmer coefficient for x
}
plot(lm.coefs,lme.coefs) # Not a perfectly straight line (i.e., not fully identical)
```

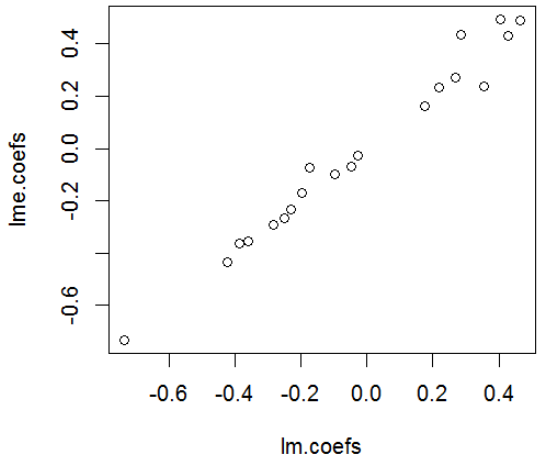

Figure 5. The not-totally-perfect fit between ordinary and mixed-effects regression

Like the **lm()** function, **lmer** objects can be put inside the **predict()** function to generate the predicted values (y-hat: $\hat{y}$). But unlike **lm()**, it doesn't draw a single line (or plane), since it's unclear how to incorporate the fixed and random effects into one plot. Instead, it gives you the observation-by-observation predictions, as shown in Figure 6 for the sploink data. This is useful for visually checking the fit of your model, but doesn't show your audience the overall linear trend.

```
plot(sdat$Age, sdat$Sploink, ylim=c(0,20)) # observed values (large white dots)
points(sdat$Age, predict(sdat.lmer), pch=20) # predicted values (small black dots)
legend("topright", pch=c(1,20), legend=c("Observed","LME predictions"))
```

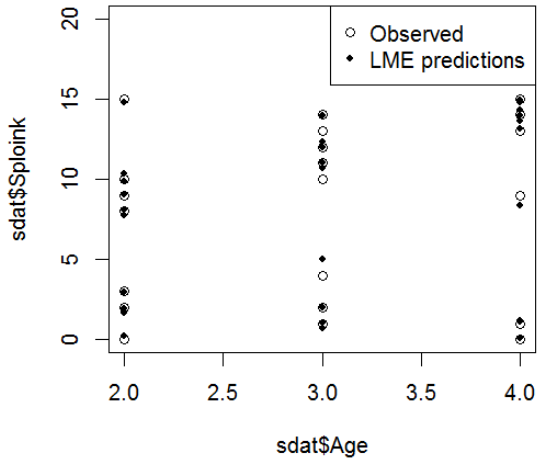

Figure 6. The observed and predicted sploink values across different children

One way to draw a single best-fit line for LME is to use **predict()** to plot the by-unit predictions. For the sploink data this happens to be the same as for the ordinary linear model, as can be seen by the perfectly overlapping best-fit lines in Figure 7.

```
plot(sdat$Age, sdat$Sploink, ylim=c(0,20))
xvals = sort(unique(sdat$Age)) # The three ages, in order
pred.mat = matrix(predict(sdat.lmer),nrow=3) # Predicted values as a 3 x 10 matrix
yvals = apply(pred.mat,1,mean) # Means of rows (i.e., means across 10 children)
lines(xvals,yvals) # Fit line for LME
abline(lm(Sploink~Age, data=sdat), lty=2, lwd=3) # Fit for ordinary linear model
legend("topright", lty=c(1,2), lwd=c(1,3), legend=c("LME","ordinary lm"))
```

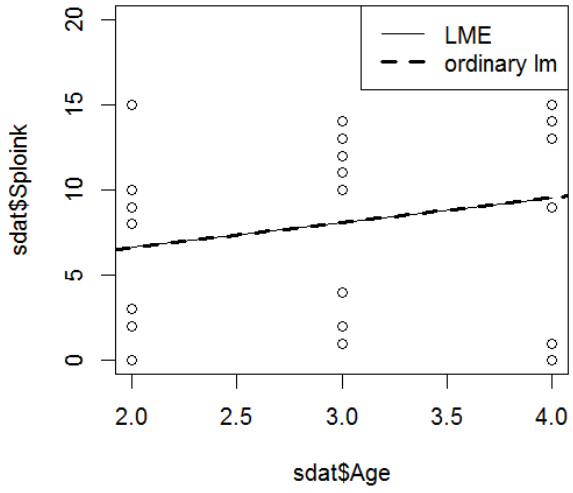

Figure 7. The ordinary and mixed-effects regression best fit lines for the sploink data

But why struggle? There's a much easier way to plot a mixed-effects model: simply use the **effects** package again. Just as with **lm** and **glm** and **aov** objects, this package lets us plot an **lmer** object using **plot(allEffects(...))**, as in Figure 8. By the way, note the huge confidence band, reflecting how unreliable our data are, given that each child only provided three data points.

```
library(effects) # You should have installed it by now
sdat.lmer = lmer(Sploink ~ Age + (Age|Child), data = sdat) # In case you forgot
plot(allEffects(sdat.lmer))
```

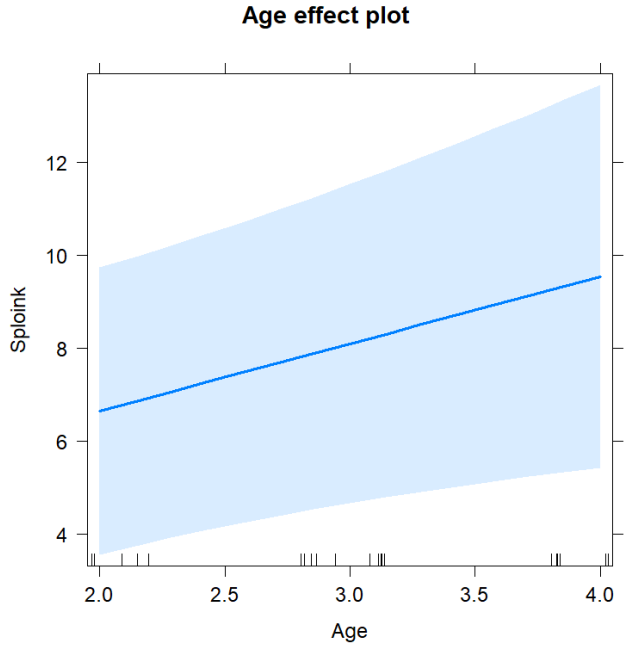

Figure 8. An effects plot for the sploink data based on LME

3. Revisiting the language-as-mixed-effects fallacy

As I've promised since the chapter on repeated-measures ANOVA, one big advantage of LME is that deals with the language-as-fixed-effect fallacy of Clark (1973) better than Clark's own proposal of *minF'*. Let's see how it works in easy situations, and then explore some of the dark underbelly of LME (convergence problems, crashes, and more controversies)

3.1 Two random variables

Let's go back to the data in **dormiR.txt**, reporting the results of an experiment run by Dorami (the sister of Doraemon) that looked at how Martian reaction times are affected by education level (between participants but within items), syntactic category (within participants but between items), and lexical frequency (also within participants but between items), all

treated as categorical factors. Note that some data are missing, so even though this data set comes from a factorial experiment, it's not balanced (a common real-life situation).

```
ddat = read.delim("doramiR.txt")
ddat = na.omit(ddat) # Some data is missing, so let's clear out the NA's
```

We have two random variables: participants and items. If we try to use **aov()** on the whole data set, using two **Error()** terms, it just gives us an error message:

```
ddat.aov = aov(RT~Education*SynCat*Freq # Fixed part
  + Error(as.factor(Participant)/(SynCat*Freq)) # By participant
  + Error(as.factor(Item)/Education), # By item
  data = ddat)
```

```
Error in aov(RT ~ Education * SynCat * Freq + Error(Participant/(SynCat *    :
   there are 2 Error terms: only 1 is allowed
```

One big advantage of the **lme4** package **lmer()** function is that it can handle more than one random variable (unlike the **lme()** function in the **nlme** package). Thus we can test by-participants and by-items analyses in a single model, just as Clark dreamed of.

For comparison, here's how we did the *minF'* analysis for Freq, which was the only variable that was significant both by participant and by item. As before, we'll skip the usual lognorming step for RT, and ignore the Pair variable.

```
attach(ddat) # To save space below
ddat.part = aggregate(RT, list(Participant, Education, SynCat, Freq), mean)
colnames(ddat.part) = c("Participant","Education","SynCat","Freq","RT")
ddat.item = aggregate(RT, list(Item, Education, SynCat, Freq), mean)
colnames(ddat.item) = c("Item","Education","SynCat","Freq","RT")
detach(ddat) # Always remember...
ddat.part$Participant = as.factor(ddat.part$Participant) # Don't forget!!!
bypart.aov = summary(aov(RT~Education*SynCat*Freq
  +Error(Participant/(SynCat*Freq)), data = ddat.part))
ddat.item$Item = as.factor(ddat.item$Item) # Don't forget!!!
byitem.aov = summary(aov(RT~Education*SynCat*Freq
  +Error(Item/Education), data = ddat.item))
F1.Freq = bypart.aov[[3]][[1]][1,4] # 3rd table, 1st part of it, 1st row, 4th column
dfnum1 = bypart.aov[[3]][[1]][1,1] # etc
dfdenom1 = bypart.aov[[3]][[1]][3,1]
F2.Freq = byitem.aov[[1]][[1]][2,4]
dfdenom2 = byitem.aov[[1]][[1]][4,1]
# Finally, the minF' results:
minF = (F1.Freq*F2.Freq)/(F1.Freq+F2.Freq)
dfnum.minF = dfnum1
dfdenom.minF = (F1.Freq+F2.Freq)^2/(F1.Freq^2/dfdenom2 + F2.Freq^2/dfdenom1)
pf(minF,dfnum.minF,dfdenom.minF,lower.tail=F)
```

[1] 0.03604612

After all that work, it just gives us one little number: the *minF'*-based *p* value. We could report this as *minF'*(1, 30.05) = 4.82, *p* = .036, but nobody uses *minF'* anyway.

LME is much easier to run on Dorami's data. To imitate what we tried to do above, we would run a so-called **maximal model**, which is just like a typical ANOVA analysis in including *all* of the fixed-effect interactions and random intercepts and random slopes, making sure that we group the fixed effects properly within their units.

In the R code below, I first recode the factors using effect coding (using the **contr.sum()** function), to make the interactions act as they do in an ANOVA (remember that by default, R uses dummy coding for regression factors).

Moreover, now that you've learned a bit about the math of mixed-effects modeling, I can tell you a bit more about the syntax behind the random parts. In particular, the notation **(Education|Item)** is actually short for **(1+ Education|Item)**, which represents a regression equation showing the Item random intercept (1) and random slope (Education). Since the thing to the left of | is a formula (unlike in **Error(...)** for **aov()**), we don't need parentheses around **SynCat*Freq** as we do for the by-participants ANOVA. If we wanted to test only the random intercepts, we would write the by-items random part as **(1|Item)**. If we wanted to test only the random slopes, we would write it as **(0+Education|Item)**, since as we saw in the regression chapter, in R's formula syntax "adding" a zero is how to represent a model without an intercept. Finally, since random intercepts and random slopes may be correlated, we can tell the model to ignore this correlation by adding up these two random components separately: **(1|Item)** + **(0+Education|Item)**. This is a so-called **zero-random-correlation model** (we'll come back to this concept later).

But in this case, we just want to mimic the ANOVA analysis, which does include something like random slopes (as expressed in the **Error(...)** part), so we'll stick with the maximal model. For this demo, let's make sure the **afex** package isn't interfering (so we get the "pure" *p*-less **lmer()** report):

```
detach("package:afex", unload=TRUE) # In case you were still running it
library(lme4) # Just in case it's not running

ddat = read.delim("doramiR.txt", stringsAsFactors=T) # To make sure they're factors
ddat = na.omit(ddat) # Since some data are missing
contrasts(ddat$Education) = contr.sum(levels(ddat$Education)) # 1 = College
contrasts(ddat$SynCat) = contr.sum(levels(ddat$SynCat)) # 1 = Noun
contrasts(ddat$Freq) = contr.sum(levels(ddat$Freq)) # 1 = High
ddat.lmer = lmer(RT~Education*SynCat*Freq # Fixed part
  +(SynCat*Freq|Participant)+(Education|Item), data = ddat) # Random parts
```

Running the last command gives us a warning that we'll ignore for now, and then we generate the summary (this time, the whole thing):

summary(ddat.lmer)

Table 9. A maximal LME model of the Dorami data

Linear mixed model fit by REML ['lmerMod']
Formula: RT ~ Education * SynCat * Freq + (SynCat * Freq | Participant) + (Education | Item)
Data: ddat

REML criterion at convergence: 5016.7

Scaled residuals:

Min	1Q	Median	3Q	Max
-1.8515	-0.6317	-0.2075	0.4085	3.9700

Random effects:

Groups	Name	Variance	Std.Dev.	Corr		
Participant	(Intercept)	635.1	25.20			
	SynCat1	239.3	15.47	1.00		
	Freq1	185.0	13.60	-1.00	-0.99	
	SynCat1:Freq1	1095.9	33.10	0.05	0.13	0.03
Item	(Intercept)	1188.2	34.47			
	Education1	844.7	29.06	-0.33		
Residual		27146.4	164.76			

Number of obs: 386, groups: Participant, 20; Item, 20

Fixed effects:

	Estimate	Std.Error	t value
(Intercept)	742.9592	12.7177	58.419
Education1	-0.4537	12.0235	-0.038
SynCat1	18.4146	11.917	1.545
Freq1	-30.67	11.803	-2.598
Education1:SynCat1	1.1824	11.1731	0.106
Education1:Freq1	17.7279	11.0514	1.604
SynCat1:Freq1	5.7752	13.5957	0.425
Education1:SynCat1:Freq1	-3.0658	12.9486	-0.237

Correlation of Fixed Effects:

	(Intr)	Edctn1	SynCt1	Freq1	Ed1:SC1	Ed1:F1	SC1:F1
Education1	-0.114						
SynCat1	0.131	0.006					
Freq1	-0.112	0	-0.071				
Edctn1:SyC1	0.006	0.148	-0.131	-0.006			
Edctn1:Frq1	0	-0.126	-0.006	-0.134	-0.081		
SynCt1:Frq1	0.016	-0.005	0.023	0.007	0	0.006	
Edc1:SC1:F1	-0.005	0.017	0	0.005	0.026	0.007	-0.099

optimizer (nloptwrap) convergence code: 0 (OK)
Model failed to converge with max|grad| = 0.0147632 (tol = 0.002, component 1)

We'll look first at the results for the fixed effects, concentrating on the line starting Freq1 (that is, High Freq vs. Low Freq, since H comes before L in the alphabet, which is how R does effect coding). It was the only parameter significant in both the by-subject and by-item ANOVA models, and it's also the parameter that we derived the *minF' p* value for. Is it significant by this LME model?

You can already get a sense of this from the magnitude of the |*t*| value, above the critical value of 1.96 for the *t* = *z* method; since the number of observations is 386, it seems reasonably safe to assume that no matter what formal method we use to compute the actual *p* value, it will be below .05. Let's see:

```
2*pnorm(-2.598)

[1] 0.009376849
```

To be more confident about this conclusion, let's compute the *p* values using the Satterthwaite method (ignoring the warnings again):

```
detach("package:lme4", unload=TRUE) # To allow afex to load it for us
library(afex)
ddat.lmer.p = lmer(RT ~ Education * SynCat * Freq + (SynCat*Freq|Participant)
  + (Education|Item), data = ddat)
summary(ddat.lmer.p)
```

Here's the result it gives for Freq:

```
Fixed effects:
             Estimate     Std.Error         df    t value    Pr(>|t|)
...               ...           ...        ...        ...         ...
Freq1       -30.6700       11.8030    16.5385     -2.598       0.019   *
...               ...           ...        ...        ...         ...
```

As expected, since our sample is kind of large, this more accurate *p* value is still quite close to what we got with the *t* = *z* method. The other fancier methods (the Kenward-Roger approximation or parametric bootstrapping) are thus not worth trying, since the results will be pretty much the same as well.

Before I finish this section, we should take a look at the rest of the summary report we get for **lmer()**, besides the fixed effects coefficients. If you look back at Table 9, you'll see that the first thing the report says (after repeating the model formula) is the final REML value of 5016.7, which is related to the model likelihood that the LME algorithm is trying to maximize as it iterates through various coefficient values, but it has no useful meaning otherwise. Then it shows the scaled residuals, which should be normally distributed; maybe there's some skew

or outliers or hidden variables here, since the minimum (-1.8515) is not the exact inverse of the maximum (3.9700); Dorami might want to look into this (but not us).

Next we see the results for the random variables. Instead of showing us their coefficients, it shows their variances, or more accurately, what the model thinks are their variances. As we saw in the math discussion above, the mixed-effects model represents the random part in terms of sets of coefficients (the β_i values), or more accurately, in terms of the parameters of a distribution of imaginary coefficients of this sort. So this stuff is useful in the computation of LME but I don't think we humans need to care about it much.

The random part of the report also shows the correlations expected among these random variables, that is, if we collected more samples from the same population and ran the same model on them, we'd expect the new coefficients to correlate with these. These values are potentially useful, since they help indicate if the model is too complicated. For example, for our analysis, the random correlations are quite high, even reaching perfect correlations of the random intercept with the SynCat and Freq random slopes. This implies that there's no point for the model to include both random intercepts and random slopes for these predictors, since one is totally predictable from the other. Thus a simpler model that uses just one or the other may fit the data well enough, or perhaps a zero-random-correlation model, that keeps both but doesn't bother modeling the correlation (again, we'll come back to this soon).

Next, we get a helpful reminder of our data size; I used the number of observations to make my guess that Freq is probably significant, since this seems like a relatively large *N*, from the perspective the Central Limit Theorem:

```
Number of obs: 386, groups:  Participant, 20; Item, 20
```

Then there's the fixed-effects coefficients, which we've already discussed, and finally the expected correlations of the fixed variables, which has the same interpretation as for the random-effect correlations, namely, they don't describe collinearity in our data set, but rather predictions about how models derived from the same population will vary in their coefficients (which, again, Dorami might want to worry about).

The final lines explain why we got a warning when we built the model, namely that the model failed to converge (though we're also told that this is "OK"). We'll come back to this commonly encountered issue later.

```
optimizer (nloptwrap) convergence code: 0 (OK)
Model failed to converge with max|grad| = 0.0147632 (tol = 0.002, component 1)
```

What about effect size for LME models? Is there any way to compute something like R^2 to tell us how much of the variance in the dependent variable is explained by the model? That's another one of those things beset by controversy, but one simple idea is to compute the simple

Pearson's correlation between your raw data and the values predicted by your model, and squaring that. The better your predictions match the data, the closer this value will be to 1. This is not mathematically identical to what R^2 means for ordinary multiple regression (i.e., we can't assume that this value is the best way to represents the proportion of variance explained by your model), but at least it's very simple to compute and uses familiar old concepts. Here's what we get for the Dorami LME model:

```
cor(ddat$RT, predict(ddat.lmer))^2
```

```
[1] 0.2661778
```

This fit isn't great: the model only explains about 27% of the variance in the reaction times. This isn't surprising, given that most of the variables aren't significant: most of Dorami's data live in the land of residuals. She needs to get back to the lab!

More accurate values are available if we use the **r.squaredGLMM()** function in the **MuMIn** package, which stands for "Multi-Model Inference" (Bartoń, 2022):

```
library(MuMIn) # Get that capitalization straight!
r.squaredGLMM(ddat.lmer)
```

```
              R2m       R2c
 [1,] 0.04976091 0.176602
Warning message:
'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

See the help page, eh? All right....

```
?r.squaredGLMM
```

Ah, it says they just changed the way these values are calculated from an older version of the package, but since we never used the older version, we don't need to know the history. If you're curious, the current version uses methods developed by Nakagawa & Schielzeth (2013), Johnson (2014), and Nakagawa et al. (2017). The reason it gives two R^2 values instead of just one is because "R2m" stands for the "marginal" R^2 that represents the variance explained just by the fixed factors, whereas "R2c" stands for the "conditional" R^2 representing the variance explained by the whole model, including the random factors. And indeed, the value we get for R2m is pretty close to what we get for a linear model without any random factors at all:

```
ddat.lm = lm(RT~Education*SynCat*Freq, data = ddat) # Only fixed factors
summary(ddat.lm)$adj.r.squared
# [1] 0.03422124
```

Even though R2c isn't huge either (.177, or about 18% of the variance is explained), it's still larger than the R2m value that ignores the random factors.

You can also compute **Cohen's *d*** for mixed-effects models, though it takes some work. I'm not sure, but maybe no built-in R package handles this yet...? If not, you can try this guy's homemade function: https://gist.github.com/jrosen48/00031865396cc855c9702e2b49a35fee.

How would such a function work? Well, do you remember, in the *t* test chapter, where we computed the difference in the two sample means and then divided by (something like) the standard deviation? Cohen's *d* is a number that reflects the size of a contrast relative to the noisiness of the data, where by convention, $d < .2$ is taken as indicating a small effect and $d > .8$ indicates a big effect. According to Westfall et al. (2014) the same idea works here.

As with *t* tests, Cohen's *d* is relatively easy to compute for LME: we divide the contrast you care about (e.g., the difference in the mean RTs for high vs. low frequency words) by the square root of the sum of all of the variance in the model, including random intercepts, random slopes, and residuals (not associated with any grouping variable), which acts something like a standard deviation for the data noisiness. So for an experiment with both participants and items as random variables, the formula would look like this (following the example in Brysbaert & Stevens, 2018, p. 6):

$$d = \frac{mean_1 - mean_2}{\sqrt{var\_int_{part} + var\_int_{items} + var\_slope_{part} + var\_slope_{items} + var_{residual}}}$$

In the case of Freq, the difference in mean RTs is this:

```
Freq.RT.diff =
  mean(ddat$RT[ddat$Freq=="Low"]) - mean(ddat$RT[ddat$Freq=="High"])
Freq.RT.diff
```

```
[1] 62.01181
```

You can get the variance values using the **lme4** package's **VarCorr()** function applied to your model, which also gives the correlations among the random variables (an issue we noted earlier):

```
VarCorr(ddat.lmer)
```

Groups	Name	Std.Dev.	Corr		
Participant	(Intercept)	25.211			
	SynCat1	15.470	0.997		
	Freq1	13.605	-0.996	-0.988	
	SynCat1:Freq1	33.107	0.056	0.127	0.030
Item	(Intercept)	34.470			
	Education1	29.052	-0.332		
	Residual	164.761			

It's actually easier to work with the information arranged as a data frame:

```
ddat.lmer.var = as.data.frame(VarCorr(ddat.lmer))
ddat.lmer.var
```

	grp	var1	var2	vcov	sdcor
1	Participant	(Intercept)	<NA>	635.58314	25.21077435
2	Participant	SynCat1	<NA>	239.32923	15.47026929
3	Participant	Freq1	<NA>	185.10932	13.60548850
4	Participant	SynCat1:Freq1	<NA>	1096.0587	33.10677728
5	Participant	(Intercept)	SynCat1	388.92634	0.99720236
6	Participant	(Intercept)	Freq1	-341.69392	-0.99617797
7	Participant	(Intercept)	SynCat1:Freq1	46.66497	0.05590979
8	Participant	SynCat1	Freq1	-207.88093	-0.98764904
9	Participant	SynCat1	SynCat1:Freq1	65.15714	0.12721761
10	Participant	Freq1	SynCat1:Freq1	13.38163	0.02970831
11	Item	(Intercept)	<NA>	1188.18928	34.47012160
12	Item	Education1	<NA>	844.03162	29.05222238
13	Item	(Intercept)	Education1	-332.92799	-0.33245137
14	Residual	<NA>	<NA>	27146.25911	164.7612185

Then you can extract the specific variances (vcov) that you need like so:

```
var.int.part = ddat.lmer.var[1,"vcov"] # Variance in participant intercepts
var.int.item = ddat.lmer.var[11,"vcov"] # Variance in item intercepts
var.slope.part = ddat.lmer.var[3,"vcov"] # Variance in Freq/participant slopes
var.slope.item = 0 # no variance in Freq/item slopes, since each item has a fixed Freq
var.resid = ddat.lmer.var[14,"vcov"]
```

Here's the square root of the sum of all this:

```
sd.total = sqrt(var.int.part + var.slope.part + var.slope.item + var.resid)
sd.total
```

```
[1] 167.2332
```

And now we just divide one by the other:

```
cohens.d.val = Freq.RT.diff/sd.total
cohens.d.val
```

```
[1] 0.3708104
```

By convention, then, Freq only shows a moderate effect (between .2 and .8), consistent with the relatively low R^2 of the model as a whole.

3.2 More complications and controversies

As we've been discussing, the LME analyses above are all **maximal models**, that is, they include all fixed-effect interactions and all slopes for within-unit factors and factor interactions in the random-effect part of the model. This follows the influential advice of Barr et al. (2013), who argue that maximal models are the best way to build on the traditions long used for ANOVA. We've already seen the connections between ANOVA and LME. For example, both models include so-called **treatment-by-unit interactions** (e.g., the Age × Child random slopes for the sploink data).

We can see the importance of including such random interactions by using a likelihood ratio test to compare the maximum model we got for the sploink data with a model that only includes the random intercept, symbolized by 1. That is, the simpler model assumes that each child has his/her own default sploink value, but it doesn't assume that each child has his/her own personal age-related slope. Put another way, the simpler model assumes that every child shows the same effect of Age on Sploink. We already know this isn't true from the discussion at the start of this chapter, so we expect that a likelihood ratio test will favor the more complex model. Indeed, this is true:

```
sdat.lmer = lmer(Sploink ~ Age + (Age|Child), data = sdat) # Maximal model again
sdat.lmer.int = lmer(Sploink ~ Age + (1|Child), data = sdat) # Only random intercepts
anova(sdat.lmer.int, sdat.lmer)
```

	Df	AIC	BIC	logLik	deviance	Chisq	Chi Df	Pr(>Chisq)	
sdat.lmer.int	4	160.51	166.11	-76.254	152.51				
sdat.lmer	6	145.22	153.62	-66.608	133.22	19.292	2	6.468e-05	***

This reveals another useful property of LME: we can used likelihood ratio tests to check which random effects we need to include in the model. Remember that in the chapter on repeated-measures ANOVA, I noted that Raaijmakers et al. (1999) argue against running by-item analyses (and thus against using *minF'*) if the items in your experiment are matched in terms of all variables except for the one you're testing. With LME, we can actually test if we need to include a by-items component in the analysis.

For example, since the items in the fake Dorami experiment were supposedly matched on all properties other than syntactic category and frequency, maybe we can get rid of the **(...|Item)** part of the model. Let's find out (again, the maximal model fails to converge):

```
ddat.lmer.max = lmer(RT~Education*SynCat*Freq
  +(SynCat*Freq|Participant)+(Education|Item), data = ddat) # Maximal model again
ddat.lmer.part = lmer(RT~Education*SynCat*Freq
  +(SynCat*Freq|Participant), data = ddat) # By-participants-only model
```

As we saw earlier, for the full model R warns us about a failure to converge, but the simpler model converges just fine. As with logistic regression algorithms, this means that for the maximal model, the LME algorithm had to stop before it reached its criterion for success. Nevertheless, the results for the simpler look quite similar to the maximal model, as you can see by comparing Table 9 with Table 10:

```
summary(ddat.lmer.part) # Showing just the fixed effects
```

Table 10. Fixed effects for by-participant-only analysis

Fixed effects:

	Estimate	Std. Error	t value
(Intercept)	743.0171	10.1024	73.549
Education1	-0.2342	10.1024	-0.023
SynCat1	18.1499	9.2679	1.958
Freq1	-30.847	9.1605	-3.367
Education1:SynCat1	1.5222	9.2679	0.164
Education1:Freq1	17.6282	9.1605	1.924
SynCat1:Freq1	5.0982	11.2385	0.454
Education1:SynCat1:Freq1	-2.8677	11.2385	-0.255

Moreover, this simpler model fits the data no worse than the maximal model, as shown by a likelihood ratio test:

```
anova(ddat.lmer.part, ddat.lmer.max) # df = 3, chi-squared = 2.5727, p = 0.4623
```

In this case, Raaijmakers et al. (1999) seem to be right: we may not always need a by-items analysis in experiments with well-matched stimuli.

However, this data set also illustrates a common problem: when trying to follow Barr et al. (2013), the maximal model can get too complex for it to converge. How should we simplify the model to avoid this problem?

First note that "maximal" doesn't mean "put every fixed variable into every unit variable". As Barr et al. (2013, p. 275) explain, "if any one factor involved in the interaction is between-

unit, then the random slope associated with that interaction cannot be estimated, and is not needed."

Researchers sometimes simplify their LME models by testing only for random intercepts, e.g. **(1|Child)** instead of **(Age|Child)**. But Barr et al. (2013) advise against this common practice, due to its greatly increased risk for Type I errors. After all, the random slope relates to the fixed effect(s) that we designed the experiment to test, and it is quite common for separate grouping units (people or linguistic forms) not only to show overall different default values (intercepts), but also different effects (slopes).

If the problem is that our model is too complex, then maybe we should simplify it a bit. While some experts advise against this (e.g., see Winter, 2020, p. 266), since it changes the conditional probabilities in the same "naughty" way that stepwise regression does, others think that it's worth trying; even Winter allows it as a "last resort". More precisely, Winter advises that you *should* simplify your model if the real-world situation is not as complicated as your model implies; his only philosophical worry is with simplifying a model just to make it fit.

But if you want to simplify just to get a fit (as published papers generally still do), Barr et al. (2013) suggest that we should do this as minimally as possible, by first removing only the intercept and leaving the slope. Using R's formula notation, this is the same as "adding" a zero, as we saw in the multiple regression chapter. So for the sploink data, this kind of model would have **(0+Age|Child)**.

They suggest that it's also OK to run a zero-random-correlation model, that is, a model that ignores the **random correlations** between random intercepts and slopes. To do this, you include an intercept (1) without the parameter and then "add" it to an intercept-less model, as in **(1|Child) + (0+Age|Child)** (since the notation **(Age|Child)** implies the intercept × slope interaction), or more elegantly as **(Age||Child)** (though unfortunately the || notation only works for numerical variables, which we don't have in the Dorami data set). Barr et al. base their conclusions on numerous simulated data set where they have created the "reality" themselves, so they know how reliable the LME models are.

In part because of the delicacy of LME (including its tendency to crash for overly complex models), the pro-maximal-model approach of Barr et al. (2013) has met with resistance from other LME experts. In particular, Matuschek, et al. (2017) argue that for most normal-sized data sets (i.e., not gigantic ones), maximal models commit the sin of **overfitting**: medium-complexity models are actually better at generalizing to new data. Overfitting often has the mathematical effect of **singularity**, where certain crucial model estimates are (close to) zero, similar to the problem we saw in the previous chapter when logistic regression tries to model a (nearly) "perfect" fit. They demonstrate the advantage of medium-complexity models with their own simulated data, showing how a maximal model can be simplified step by step, in order to increase statistical power without increasing the risk of Type I errors. Their arguments are set in a larger debate over "hypothesis testing" vs. "data exploration". Whereas Barr et al.

want everybody to do the same thing so that hypotheses are always tested in a consistent way, Matuschek et al. argue that in real life, even a fixed factorial design requires some data exploration as you compare models in search of the best fit. After all, the *p* values in an ordinary regression (or ANOVA) are related to the ones you get from likelihood ratio tests comparing two different models.

Here is what they suggest (see also Bates et al., 2015, for more explicit R-oriented instructions): you start with a maximal model, then create a zero-random-correlation model, then continue to simplify its random-effects structure in a "systematic" way, testing each model against the previous one using likelihood ratio tests, until no further improvement occurs, and then add the random correlations back for the random effects that survive this procedure. The "systematic" part of the procedure involves using principal components analysis (that data exploration method that I mentioned in the regression chapter, but which we don't have space to discuss in detail in this draft of the book) to estimate how many random variables are necessary.

This seems like a lot of work to do by hand, so until an R package automates the process, it's not clear how many people are going to try it, rather than using the conceptually simpler (if more crash-prone and overfitted) approach of Barr et al. (2013). In any case, I wouldn't worry about your LME model unless the coefficients or *p* values dramatically change when the random effects structure changes (i.e., adding or removing random intercepts and/or slopes). If this happens, common sense would suggest that your theoretical conclusions may be too delicate to take seriously anyway. Statistics isn't magic!

So what else can we do when our model doesn't converge? Here are a few other suggestions, none of which is guaranteed to work in all cases.

First, note that the probability that a LME algorithm will crash (or give unreliable results) is increased if the predictors differ too much in their scales (i.e., their standard deviations). The estimation algorithm for logistic regression is delicate in the same way, and even ordinary linear regression works best if the predictors have similar scales. Fortunately, this situation is easy to fix: just rescale the independent variables to *z* scores before running the regression.

For example, Myers (2015) describes a mixed-effects regression analysis where some predictors were raw frequencies, ranging up into the 100s, while other predictors were a measure of syllable element collocation, ranging only from 0 to 1. The model using the raw variables crashed, but after converting both to *z* scores, it ran just fine.

Let me create a fake example, so we can discuss the problem and possible solutions:

```
set.seed(3) # Makes your fake data the same as mine
y = rnorm(100) # Dependent variable; a linear model is appropriate
x1 = runif(100) # ranges from 0 to 1
x2 = runif(100)*10000 # ranges from 0 to 10000
g = rep(1:10,10) # 10 grouping units with 10 observations each
```

Now watch it crash (try it!):

yxgmodel.lmer1 = lmer(y~x1*x2+(x1*x2|g))

The output of **lmer()** gives us lots of warnings and complaints, but also a helpful suggestion ("Some predictor variables are on very different scales: consider rescaling"), so let's try it again, using *z* scores created using the **scale()** function. As noted in the earlier regression chapters, scaling also has the nice effect of giving us (something similar to) standardized regression coefficients in the final model (though they're not exactly like standardized regression coefficients unless we also standardize the dependent variable, which we're not doing in this particular case, though we could, since the dependent variable is numeric):

x1.z = scale(x1); x2.z = scale(x2)

Now the warnings are much less nasty (try it!):

yxgmodel.lmer2 = lmer(y~x1.z*x2.z+(x1.z*x2.z|g))

The only warning we still get is this:

boundary (singular) fit: see help('isSingular')

I mentioned singularity earlier in this chapter (search for it!): it's not a fatal problem, but it does suggest that our model is too complex for our data set. If we follow their advice, we'll see several suggestions, some of which we've already considered:

?isSingular

A second method for dealing with converge and singularity problems is to use a different fitting algorithm until we get a result with no errors. Technically, such an algorithm performs **optimization**, since it tries to find a fit that's the "best possible" in this never-perfect world (which is where Optimality Theory gets its name). Optimization has to do with the searching algorithm, not the form of the final statistical model itself, but the algorithm still matters, since certain solutions are easier (or faster) to find using certain methods versus others (though faster does not necessarily mean better).

By default, **lmer()** uses an optimizer called **nloptwrap**, but you can try a different optimizer with the following argument inside **lmer()** (putting the name of your favorite optimizer in the "**xxx**" part):

control = lmerControl(optimizer = "xxx")

But which optimizer should we try? Well, why not all of them? The key function is **allFit()**, which runs in the **afex** package, but in order to work, you first have to install two other packages for optimization, namely **optimx** (Nash & Varadhan, 2011; Nash, 2014) and **dfoptim** (Varadhan et al., 2020). Of course, trying out several different fitting algorithms can be quite slow, since your data has to get modeled several times in a row, but our fake example here is pretty tiny so let's try it:

```
install.packages(c("optimx","dfoptim")) # To avoid scrolling in the long list of packages
library(afex) # If you haven't already started it
allFit(yxgmodel.lmer2)
```

Ah, buried in that long list of results and warnings, it seems that there are two algorithms that didn't generate singularity warnings (**Nelder_Mead** and **nmkbw**):

```
bobyqa : boundary (singular) fit: see help('isSingular')
[OK]
Nelder_Mead : [OK]
nlminbwrap : boundary (singular) fit: see help('isSingular')
[OK]
nmkbw : [OK]
optimx.L-BFGS-B : boundary (singular) fit: see help('isSingular')
[OK]
nloptwrap.NLOPT_LN_NELDERMEAD : boundary (singular) fit: see help('isSingular')
[OK]
nloptwrap.NLOPT_LN_BOBYQA : boundary (singular) fit: see help('isSingular')
[OK]
```

Sadly, this turns out to mean merely that those algorithms didn't converge at all. For example, this is what happens when try **Nelder_Mead**:

```
yxgmodel.lmer3 = lmer(y~x1.z*x2.z+(x1.z*x2.z|g),
  control = lmerControl(optimizer = "Nelder_Mead"))
```

```
Warning message:
In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,    :
   Model failed to converge with max|grad| = 0.447507 (tol = 0.002, component 1)
```

The "tol" stands for **tolerance**, which is a tiny preset value (1e-5 = .00001) that determines when to stop the looping algorithm: if the current and previous estimate differ by less than the tolerance, then our fit is considered to be good enough and the looping stops. Sometimes it helps to increase the number of loops, which you can do with this argument inside **lmer()**, where "**xxx**" is some gigantic number (Miller, 2018, suggests 2e5 = 200000):

optCtrl=list(maxfun=xxx)

I don't think that's going to help convergence with **Nelder_Mead**, based on how far away from the tolerance it is with the default number of loops: 0.447507 vs. 0.002!

Let's try one final option suggested on the **?isSingular** page, namely to use "a partially Bayesian method". We'll talk about Bayesian statistics in detail in the next chapter, but the relevant idea here is that Bayes gives us mathematical tools for modeling our **prior** assumptions, before we even look at our data. This is relevant to model fit because an optimization algorithm has to start somewhere, and the "smarter" the starting point is, the more readily the algorithm can actually reach its goals.

We can try out this idea using the **blme** package (Chung et al., 2013; "b" for "Bayes" of course), which gives us the **blmer()** function:

```
library(blme) # Gotta install it first
yxgmodel.lmer4 = blmer(y~x1.z*x2.z+(x1.z*x2.z|g))

Warning messages:
1: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,    :
   unable to evaluate scaled gradient
2: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,    :
   Model failed to converge: degenerate    Hessian with 3 negative eigenvalues
```

Oh well. I think in this case we just have to admit that our model is too complex for our data set, so singularity (overfitting) is pretty much unavoidable.

4. Generalized mixed-effects modeling

There isn't much to say about **generalized linear mixed-effects modeling** (**GLMM**) for its own sake, since as the name suggests, it's just an application of the linear mixed-effects approach (like LME) to generalized linear modeling (like logistic regression). In this section we'll go through an example using another function in the indispensible **lme4** package, discuss some complexities (involving crashing again), and then look briefly at yet another generalization of regression: **generalized additive mixed-effects modeling** (**GAMM**).

4.1 Reanalyzing an acceptability judgment experiment

GLMM is for when your dependent variable is categorical (**mixed-effects logistic regression** for binary data, **mixed-effects Poisson regression** for small count data), and the observations are grouped by units (people or linguistic items) rather than all being independent of each other.

We'll focus on mixed-effects logistic regression. Tutorials on mixed-effects logistic regression include Baayen (2008) and Jaeger (2008). Linguistic applications include Xu et al. (2006) (categorical perception of tones), Barr (2008) (eyetracking for picture-finding experiments), and Moreton (2008) (artificial grammar learning).

As a concrete example, let's take another look at the data in **demo.txt**, which, as you may remember, are the results of a real syntax experiment, where seven Mandarin speakers were each given 20 sentences to judge as good (Judgment = 1) or bad (Judgment = 0). The sentences came in sets of four that were matched as closely as possible, varying only two factors: ComplexNP (1 = complex noun phrase, -1 = simple noun phrase) and Topic (1 = element extracted from the noun phrase to topic position, -1 = no extraction). Remember that our main interest is testing whether there is an interaction between ComplexNP and Topic, which could mean that it's ungrammatical to extract topics from complex noun phrases.

As a reminder, here's repeated-measures logistic regression. First load the data:

```
demdat = read.delim("demo.txt") # We're already using the name "ddat" for Dorami!
```

And here's the repeated-measures logistic regression:

```
int.coef = numeric(7) # Will hold intercept coefficients across participants
CNP.coef = numeric(7) # Will hold ComplexNP coefficients across participants
Top.coef = numeric(7) # Will hold Topic coefficients across participants
CxT.coef = numeric(7) # Will hold interaction coefficients across participants
for (i in 1:7) { # Run logistic regressions for each participant
  demdat.i = subset(demdat, demdat$Speaker==i) # Participant i's data
  glm.i = glm(Judgment~ComplexNP*Topic, family=binomial, data=demdat.i)
  int.coef[i] = summary(glm.i)$coefficients["(Intercept)","Estimate"]
  CNP.coef[i] = summary(glm.i)$coefficients["ComplexNP","Estimate"]
  Top.coef [i] = summary(glm.i)$coefficients["Topic","Estimate"]
  CxT.coef [i] = summary(glm.i)$coefficients["ComplexNP:Topic","Estimate"]
}
# Coefficients, t, p:
t.test(int.coef); t.test(CNP.coef); t.test(Top.coef); t.test(CxT.coef)
# SE:
as.numeric(mean(int.coef)/t.test(int.coef)$statistic)
as.numeric(mean(CNP.coef)/t.test(CNP.coef)$statistic)
as.numeric(mean(Top.coef)/t.test(Top.coef)$statistic)
as.numeric(mean(CxT.coef)/t.test(CxT.coef)$statistic)
```

Table 11 shows the results from the above work, manually arranged in an R-like regression table:

Table 11. Repeated-measures logistic regression analysis of demo data

	Estimate	SE	df	t	p
(Intercept)	6.07439	2.89626	6	2.0973	0.08078
ComplexNP	-6.07576	1.95096	6	-3.1142	0.02074
Topic	-13.3231	0.94895	6	-14.0399	8.145e-06
ComplexNP:Topic	-6.07576	1.95096	6	-3.1142	0.02074

Should we follow Raaijmakers et al. (1999) and assume that we don't need to include a by-items random analysis, since the sentences were well matched? Or should we follow Barr et al. (2013) and run a maximal model? Or should we follow Matuschek et al. (2017) and start with the maximal model and then simplify it until it converges? Or should we follow Winter (2020) and reserve model simplification as a last resort, and instead first try rescaling and alternative optimization algorithms? Or...?

Welcome to real life in the era of mixed-effects modeling!

Well, we have to do something, so let's just follow Matuschek et al. (2017), not that I'm saying this is necessarily the best thing to do. So we start with the maximal model, which requires random intercepts and random slopes for both fixed variables and their interaction for the participants (since ComplexNP, Topic, and their interaction are within Speaker), but just random intercepts for the items, since all of our fixed variables are between items (i.e., they describe properties of individual test sentences).

We'll do this analysis using the **glmer()** function in the **lme4** package. The name should look familiar: just as **lmer()** is the mixed-effects version of **lm()**, so **glmer()** is the mixed-effects version of **glm()**, for generalized linear models. Its syntax should also be familiar: the random part is like **lmer()** and the rest is like **glm()**, including here a mention of the binomial family, since we're doing a kind of logistic regression. By the way, be patient: this takes a little bit of time to finish iterating through the algorithm.

```
demdat.glmer = glmer(Judgment~ComplexNP*Topic + (ComplexNP*Topic|Speaker)
  + (1|Item), family = binomial, data = demdat)
```

In the end, the model crashes:

```
Warning message:
In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,    :
   Model failed to converge with max|grad| = 0.0294537 (tol = 0.002, component 1)
```

This isn't surprising. Not only are maximal models crash-prone, but our sample size is quite small: only 120 observations. It's hard for the GLMM algorithm to find good evidence for all of the random intercepts, slopes, and possible correlations in such a small sample. In

other words, the advice of Barr et al. (2013) is not practical: a maximal model is too complex for our data set, just as Matuschek et al. (2017) would warn.

We can simplify the random effects structure in a minimal way, namely by ignoring the random interactions. Since our independent variables ComplexNP and Topic are numerical, the syntax is easy: we just replace the | operator with the || operator. This creates a zero-random-intercepts model (we could also make such a model if our independent variables were coded as factors, but then we'd have to spell out the intercept and slope separately: **(1|Speaker) + (0+ ComplexNP*Topic | Speaker)**).

Unfortunately, this slightly-less-complex model still crashes (try it yourself):

```
demdat.glmer.nocorr = glmer(Judgment~ComplexNP*Topic
  + (ComplexNP*Topic || Speaker) + (1|Item),
  family = binomial, data = demdat)
```

If we follow the Matuschek et al. (2017) approach, our next step is to stick with the zero-random-correlation approach, but now try to simplify the ComplexNP*Topic part of the random structure. This formula can be unpacked as ComplexNP + Topic + ComplexNP:Topic. Even though our theoretical interest is mainly in ComplexNP:Topic, our hypothesis only cares about the fixed part of the model, so it's OK to drop this interaction in the random part. That gives us this model:

```
demdat.glmer.nocorr.noCxT = glmer(Judgment~ComplexNP*Topic
  + (ComplexNP+Topic || Speaker) + (1|Item),
  family = binomial, data = demdat)
```

It still fails to converge:

```
Warning message:
In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,    :
   Model failed to converge with max|grad| = 0.0285757 (tol = 0.002, component 1)
```

So to keep this chapter from going on forever, let's just jump down to the simplest model that might be valid, namely one where we keep the random intercepts for the items, but for the speakers, we only care about their random intercepts, that is, each person's overall tendency to say "yes" or "no" in this binary acceptability judgment task, and ignore any cross-speaker differences in their response to the fixed variables. Since we're not testing any random slopes, the model naturally doesn't include random correlations either.

That makes our model look like the following, and finally we get one that doesn't crash:

```
demdat.glmer.onlyrandint = glmer(Judgment~ComplexNP*Topic
  + (1|Speaker) + (1|Item),
```

family = binomial, data = demdat)

So now we can finally look at the results:

summary(demdat.glmer.onlyrandint)

Let's put its entire output into Table 12:

Table 12: By-participants-and-items random-intercepts-only model for demo data

Generalized linear mixed model fit by maximum likelihood (Laplace Approximation) ['glmerMod']
Family: binomial (logit)
Formula: Judgment ~ ComplexNP * Topic + (1 | Speaker) + (1 | Item)
Data: demdat

AIC	BIC	logLik	deviance	df.resid
72.9	90.5	-30.4	60.9	134

Scaled residuals:

Min	1Q	Median	3Q	Max
-2.69730	-0.05818	0.01109	0.05770	1.60172

Random effects:

Groups	Name	Variance	Std.Dev.
Item	(Intercept)	3.028	1.740
Speaker	(Intercept)	7.336	2.708

Number of obs: 140, groups: Item, 20; Speaker, 7

Fixed effects:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	2.8146	1.6843	1.671	0.0947	.
ComplexNP	-1.7625	0.9042	-1.949	0.0513	.
Topic	-4.6613	1.8238	-2.556	0.0106	*
ComplexNP:Topic	-1.7624	0.9042	-1.949	0.0513	.

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Correlation of Fixed Effects:

	(Intr)	CmplNP	Topic
ComplexNP	-0.409		
Topic	-0.685	0.621	
CmplxNP:Tpc	-0.409	0.250	0.621

The report first shows that **glmer()** uses something called the "Laplace Approximation" to maximize likelihood; more precisely, it uses something called adaptive Gauss-Hermite

quadrature, though other options are available, like penalized quasi-likelihood. I have no idea what these things are either, except that they're "cheats" to give us something like maximum likelihood when actual maximum likelihood is technically impossible to calculate in a mixed model.

Then it gives us some estimates of model fit, including the AIC, which we introduced in the multiple regression chapter, as well as log likelihood, deviance, and the deviance *df*, which we introduced in the logistic regression chapter (remember that deviance is -2 times the log likelihood). And there's that information about the residual distribution (looks reasonably symmetrical to me). The random effects part is relatively simple, since we only modeled random intercepts, not random slopes. The fixed effects correlations at the bottom are there too, and not really important as usual.

And look: **glmer()** gives us *p* values! Like **glm()**, the Wald test allows **glmer()** function to avoid the mixed-effects *df* controversy by interpreting *B*/*SE* in terms of *z* values instead of *t* values. As with ordinary logistic regression, this means larger samples are safer, and as often noted above, our sample size is probably too small.

Sadly, the *p* values bring bad news: they suggest that everything is significant, except for the theoretically crucial thing: the interaction is only marginal. Too bad!

But do we really need to include the by-items random intercept at all? Raaijmakers et al. (1999) advise that by-items analyses are unnecessary if the experimental materials are well-matched, and indeed they were in this experiment. Instead of just assuming that their logic is valid, however, we can actually test if it is, by creating a model that drops the by-items random component and comparing it with the above model.

Here we go, creating the participants-only model:

```
demdat.glmer.onlyrandint.part = glmer(Judgment~ComplexNP*Topic + (1|Speaker),
  family = binomial, data = demdat)
```

Let's use a likelihood ratio test to see if this simpler model does as well with the data as the more complex one. As with ordinary (logistic or linear) regression or LME, we can use the **anova()** function to do this (putting the simpler model first), and as with ordinary logistic regression, I will include **test = "Chisq"** to get a *p* value (even though **glmer** objects are "smarter" than **glm** objects in that they know what kind of test you need here even without you telling it):

```
# test = "Chisq" is optional here, unlike the case for glm objecs
anova(demdat.glmer.onlyrandint.part, demdat.glmer.onlyrandint, test = "Chisq")
```

Good news: The model comparison gives us a chi-squared report showing $\chi^2(1) = 3.5214$, $p = .06058$ (check yourself!), so this simpler model doesn't really do worse than the more

complex one. Just as Raaijmakers et al. (1999) argued, you might not need to include a by-items analysis if your experimental materials are properly matched across conditions.

Now let's see what our simpler model says:

summary(demdat.glmer.onlyrandint.part)

The complete summary report is shown in Table 13:

Table 13. By-participants-only random-intercepts-only model for demo data

Generalized linear mixed model fit by maximum likelihood (Laplace Approximation) ['glmerMod']
Family: binomial (logit)
Formula: Judgment ~ ComplexNP * Topic + (1 | Speaker)
Data: demdat

AIC	BIC	logLik	deviance	df.resid
74.4	89.1	-32.2	64.4	135

Scaled residuals:

Min	1Q	Median	3Q	Max
-4.6175	-0.1573	0.0460	0.1327	2.5770

Random effects:

Groups	Name	Variance	Std.Dev.
Speaker	(Intercept)	3.354	1.831

Number of obs: 140, groups: Speaker, 7

Fixed effects:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	1.9124	0.9153	2.089	0.0367	*
ComplexNP	-1.2382	0.4814	-2.572	0.0101	*
Topic	-3.2668	0.7257	-4.502	6.74e-06	***
ComplexNP:Topic	-1.2382	0.4814	-2.572	0.0101	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

	(Intr)	CmplNP	Topic
ComplexNP	-0.051		
Topic	-0.489	0.282	
CmplxNP:Tpc	-0.051	-0.297	0.282

Now we can safely and honestly conclude that our crucial interaction is truly significant. As we've seen from our *minF'* discussions, by-participant *p* values tend to be lower than by-participant-and-items *p* values, and the same holds for LME models.

On the other hand, our data set is quite tiny: only 140 observations. So we might just have a Type I error here. The small sample size is likely also an important reason why the more complex models crashed: there weren't enough data points to handle all of those random slopes and their correlations with the random intercepts.

4.2 More about mixed-effects logistic regression

If you don't trust the Wald test to compute *p* values from those sample-size-ignoring *z* values, you can use the **mixed()** function in the **afex** package to try out other types of *p* values, as for LME. For example, let's try the likelihood ratio test (we could also try the parametric bootstrap, but that's much slower).

First we start up **afex** (letting it load **lme4** for us, as recommended):

```
detach("package:lme4", unload=TRUE) # If we started running lme4 before afex
library(afex)
```

Now here's our simple model again, to double-check its *p* values (in an analysis of deviance table):

```
demdat.mixed.onlyrandint.part.LRT = mixed(Judgment~ComplexNP*Topic
  + (1|Speaker), method="LRT", family = binomial, data = demdat)
demdat.mixed.onlyrandint.part.LRT
```

	Effect	df	Chisq	p.value
1	ComplexNP	1	5.79 *	0.016
2	Topic	1	76.64 ***	<.001
3	ComplexNP:Topic	1	5.79 *	0.016

So even using a method that takes sample size into account, our crucial interaction is still significant.

What about overall model fit? As we saw in the **glmer()** reports in Tables 12 and 13, GLMM is just like any other kind of regression in being associated with AIC (Akaike information criterion), the related BIC (**Bayesian information criterion**), log likelihood, and deviance measures: in each case, values closer to zero imply a better fit. In fact, all are calculated in related ways. We've discussed all of them in the logistic regression chapter, except for BIC, and since this thing has "Bayesian" in the name, I'll save it for the Bayesian chapter.

Another way to test model fit is to use the **predict()** function to compare predictions with the observed data. For LME, we used this function to make plots and compute something like the R^2 value for model fit. We can do the same thing for GLMM, but just for variety, let's use

it to do something else (and this also works for ordinary logistic regression, LME, and regression more generally): **cross-validation** (see, e.g., Baayen, 2008, and Johnson, 2008).

In this approach, we rebuild the model on most but not all of the data, then see how well it predicts the left-over data that we didn't build the model on. Recall that for logistic regression, the **predict()** function outputs logits by default, with positive values implying a response towards 1 and negative values implying a response towards 0. We'll run through 100 **glmer()** models based on the participants-only model we ended up with above (**demdat.glmer.onlyrandint.part**), built on only 85% of the data each time. That's an arbitrary size, but since our total number of observations is only 120, I wanted to give each model enough information to make a reasonable guess, while still leaving enough left over data to test the predictions on. Each time through the loop, we'll calculate the proportion of the remaining 15% of the data points that it guesses correctly (e.g.., negative logit matches real response of 0), and then compute the mean of all of these 100 proportions. A perfect match would be 100%, so let's see how close we can get.

Since, as you've already experienced, **glmer()** itself runs through loops, each of these 100 models itself takes some time to run, so be patient when you run the following code. It will also throw off a lot of warnings, since several of the models don't converge or have other problems.

```
set.seed(1) # So your results match mine
N = nrow(demdat); prop.correct = numeric(100) # Proportion correct for each try
for (i in 1:100) { # Try 100 random samples (or fewer, if you're impatient)
  S.i = sample(1:N, size = round(0.85*N,0)) # Randomly sample 85% rows
  demdat.train.i = demdat[S.i,] # Just rows in the vector S.i (85%)
  demdat.test.i = demdat[setdiff(1:N,S.i),] # Rows NOT in the vector S.i (15% left over)
  demdat.i.glmer = glmer(Judgment~ComplexNP*Topic + (1|Speaker),
    family = binomial, data = demdat.train.i) # Derive model from 85%
  predict.i = predict(demdat.i.glmer, demdat.test.i, type="response") # Predict for 15%
  hits.i = sum((predict.i>0) == demdat.test.i$Judgment) # Count hits (>0 implies 1)
  prop.correct[i] = hits.i/nrow(demdat.test.i) # Proportion of hits for test set i
}
mean(prop.correct); sd(prop.correct) # Overall accuracy (and its variability)
```

```
[1] 0.6814286
[1] 0.103909
```

This gives us 68% accuracy (SD 10%), which is not absolutely awful, but it's not very impressive either. I guess this is a small-sample problem again: either we don't have enough information to make good predictions, or else the large number of poorly-fit 85% samples generated such weird coefficients that their predictions were worthless.

To polish off our discussion of mixed-effects logistic regression, let's take a quick final look at VARBRUL again. Since logistic regression is at the heart of VARBRUL, and

sociolinguists often deal with grouped data (e.g., *these* corpus items were produced by one speaker, while *those* corpus items were produced by another speaker), it seems reasonable to adopt mixed-effects logistic regression for sociolinguistic data too. Indeed, the **Rbrul** package, introduced in the logistic regression chapter, also includes mixed-effects modeling among its bag of tricks (Johnson, 2009). Try it if you want!

However, the sociolinguist Paolillo (2013) gives a thought-provoking argument in favor of sticking to the traditional VARBRUL approach of treating the speakers as a *fixed* variable, rather than as a random variable as in GLMM. Namely, unlike a psycholinguistic experiment, in sociolinguistics the speakers are often not intended to represent a random sampling of a larger population. Indeed, sociolinguists are often reluctant to assume *a priori* that a given set of speakers must necessarily all represent a single population (e.g., that older, younger, male, female speakers are all the same). Paolillo thus gives instructions on how to analyze speakers as fixed, while still including the associated slopes and interactions properly in the model.

One more thing: in case your curiosity remains unsatisfied and you simply must know more technical details about GLMM, check out the frequently-asked-questions page set up by Ben Bolker (one of the programmers behind **lme4**) at http://bbolker.github.io/mixedmodels-misc/glmmFAQ.html.

4.3 Generalized additive mixed-effects modeling

If we can generalize ordinary regression into generalized additive modeling (GAM), where the independent variables may have any crazy shape necessary to fit the data (instead of sticking to lines or simple curves), then what's stopping us from generalizing this again up into mixed-effects modeling, giving us **generalized additive mixed-effects modeling** (GAMM)? Now that we have fancy high-speed computers, what's stopping is: nothing! We can do whatever we want!

To give this approach its due, we'd need some rich numerical data that has a grouping variable, plus a lot more time to play around with it. So instead of going through an example, I'll just suggest that to learn about GAMM you can look at some of the GAM references that I mentioned before (Baayen et al., 2017; Tremblay & Newman, 2014). To run relatively "simple" GAMMs, you can use the **mgcv** package (Wood, 2006) that we tried out last time, since it has a function called **gamm()** that does for generalized additive mixed-effects modeling what its **gam()** function does for ordinary generalized additive modeling. However, the **gamm()** function relies on the **lme()** function in the **nlme** package, so if you want to have more than one random variable, you instead need to use the **gamm4** package (Wood & Scheipl, 2015), which contains the function **gamm4()** which works its mixed-effects magic using the **lmer()** and **glmer()** functions in the **lme4** package.

6. Conclusions

Are mixed-effects models the best thing that ever happened to statistics? It depends on who you ask. On the one hand, they seem like a dream come true: they harness the power of estimation-based algorithms to finally make it possible to deal with the language-as-fixed-effects problem in the most complete way possible, something like how Fisher imagined way back when. We can use the mixed-effects approach whether our dependent variable is continuous or categorical, and we can test for random intercepts (differences in the default responses of each person or test item), random slopes (differences in their own sensitivity to the fixed variables we're testing), and even model any interaction between these two things, while still being able to compare models using likelihood ratio tests, plotting the results, standardizing coefficients so we can get effect sizes, and all the other things we learned to do with regular regression. On the other hand, the mixed-effects algorithm sure crashes a lot, and it's kind of annoying having to simplify the model by hand until it stops crashing, and even the experts don't agree on the best way to do this: they can't even agree on how to compute the *p* values, which are supposed to be a major goal of doing inferential statistics (at least the non-Bayesian hypothesis-testing kind). Yet it seems that mixed-effects models are here to stay: they have already taken over psycholinguistics (particularly in lexical and syntactic processing in adults) and to a lesser extent phonetics and language acquisition, and they are also becoming more and more popular in sociolinguistics and language teaching (with some caveats). This makes sense, since linguists deal with a lot of grouped data, but it seems reasonable to be cautious too. A lot of studies can still be analyzed just as well, if not better, with simple *t* tests, ANOVAs, or more traditional sorts of regressions.

References

Baayen, H., Vasishth, S., Kliegl, R., & Bates, D. (2017). The cave of shadows: Addressing the human factor with generalized additive mixed models. *Journal of Memory and Language*, *94*, 206-234.

Baayen, R. H. (2008). *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge University Press.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language, 59*, 390-412.

Barr, D. J. (2008). Analyzing "visual world" eyetracking data using multilevel logistic regression. *Journal of Memory and Language, 59*(4), 457-474.

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language, 68* (3), 255-278.

Bartoń, K. (2022). MuMIn: Multi-Model Inference. R package.

Bates, D. (2006). lmer, p-values and all that. https://stat.ethz.ch/pipermail/r-help/2006-May/094765.html

Bates, D., Kliegl, R., Vasishth, S., & Baayen, R. H. Parsimonious mixed models. http://arxiv.org/abs/1506.04967.

Bates, D., Maechler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software, 67*(1), 1-48.

Bell, B. A., Smiley, W., Ene, M., Sherlock, Jr., P. R., & Blue, G. L. (2013). An intermediate primer to estimating linear multilevel models using SAS. Proceedings of SESUG 2013. Available at http://analytics.ncsu.edu/sesug/2013/SD-14.pdf

Brysbaert, M. & Stevens, M. (2018). Power analysis and effect size in mixed effects models: A tutorial. *Journal of Cognition, 1*(1), 1-20.

Chung, Y., Rabe-Hesketh, S., Dorie, V., Gelman, A., & Liu, J. (2013). A nondegenerate penalized likelihood estimator for variance parameters in multilevel models. *Psychometrika, 78*(4), 685-709.

Clark, H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of Verbal Learning and Verbal Behavior, 12*, 335-359.

Gries, S. T. (2013). *Statistics for linguistics with R: A practical introduction* (2nd edition). Berlin: De Gruyter.

Jaeger, T. F. (2008). Categorical data analysis: Away from ANOVAs (transformation or not) and towards logit mixed models. *Journal of Memory and Language, 59* (4), 434-446.

Johnson, D. E. (2009). Getting off the GoldVarb standard: Introducing Rbrul for mixed-effects variable rule analysis. *Language and Linguistics Compass, 3*(1), 359-383. Software available at http://www.danielezrajohnson.com/rbrul.html.

Johnson, K. (2008). *Quantitative methods in linguistics*. Wiley.

Johnson, P. C. (2014). Extension of Nakagawa & Schielzeth's R^2_{GLMM} to random slopes models. *Methods in Ecology and Evolution, 5*(9), 944-946.

Judd, C. M., Westfall, J., & Kenny, D. A. (2012). Treating stimuli as a random factor in social psychology: A new and comprehensive solution to a pervasive but largely ignored problem. *Journal of Personality and Social Psychology, 103* (1), 54-69.

Kenward, M. G., & Roger, J. H. (1997). Small sample inference for fixed effects from restricted maximum likelihood. *Biometrics, 53*, 983-997.

Kuznetsova, A., Brockhoff, P. B., Christensen, R. H. B. (2017). lmerTest package: Tests in linear mixed effects models. *Journal of Statistical Software, 82*(13), 1-26.

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language, 94*, 305-315.

Miller, S. V. (2018) Mixed effects modeling tips: Use a fast optimizer, but perform optimizer checks. https://svmiller.com/blog/2018/06/mixed-effects-models-optimizer-checks/

Moreton, E. (2008). Analytic bias and phonological typology. *Phonology, 25*(1), 83-127.

Myers, J. (2015). Stuck in the middle: Mandarin medials in articulation, parsing, and association. In Y. E. Hsiao & L.-H. Wee (Eds.) *Capturing phonological shades within and across languages* (pp. 101-119). Cambridge, UK: Cambridge Scholars Publishing.

Nakagawa, S., & Schielzeth, H. (2013). A general and simple method for obtaining R^2 from Generalized Linear Mixed-effects Models. *Methods in Ecology and Evolution, 4* (2), 133-142.

Nakagawa, S., Johnson, P. C. D., & Schielzeth, H. (2017). The coefficient of determination R^2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. *Journal of the Royal Society Interface, 14* (134), 20170213.

Nash, J. C. (2014). On best practice optimization methods in R. *Journal of Statistical Software, 60*(2), 1-14.

Nash, J. C., & Varadhan, R. (2011). Unifying optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software, 43*(9), 1-14.

Paolillo, J. C. (2013). Individual effects in variation analysis: Model, software, and research design. *Language Variation and Change, 25*(1), 89-118.

Pinheiro, J. C., & Bates, D. M. (2000). *Mixed-effects models in S and S-Plus*. Springer.

Raaijmakers, J. G. W., Schrijnemakers, J. M. C., & Gremmen, F. (1999). How to deal with "the language-as-fixed-effect fallacy": Common misconceptions and alternative solutions. *Journal of Memory and Language, 41*, 416-426.

Satterthwaite, F. E. (1946). An approximate distribution of estimates of variance components. *Biometrics Bulletin, 2* (6), 110-114.

Singmann, H. (2014). afex. R package.

Tremblay, A., & Newman, A. J. (2015). Modeling nonlinear relationships in ERP data using mixed-effects regression with R examples. *Psychophysiology, 52*(1), 124-139.

Varadhan, R., Borchers, H. W., Bechard, V. (2020). dfoptim: Derivative-Free Optimization. R package.

Vasishth, S., & Broe, M. (2011). *The foundations of statistics: A simulation-based approach*. Springer.

Welch, B. L. (1947). The generalization of "Student's" problem when several different population variances are involved. *Biometrika, 34*, 28-35.

Westfall, J., Kenny, D. A., & Judd, C. M. (2014). Statistical power and optimal design in experiments in which samples of participants respond to samples of stimuli. *Journal of Experimental Psychology: General, 143*(5), 2020-2045.

Wood, S. (2006). *Generalized additive models: An introduction with R*. CRC Press.

Wood, S., & Scheipl, F. (2015). gamm4: Generalized additive mixed models using mgcv and lme4. R package.