Nagle, C. (2019). An introduction to fitting and evaluating mixed-effects models in R. In J. Levis, C. Nagle, & E. Todey (Eds.), *Proceedings of the 10th Pronunciation in Second Language Learning and Teaching Conference*, ISSN 2380-9566, Ames, IA, September 2018 (pp. 82-105). Ames, IA: Iowa State University.

RESEARCH WORKSHOP

AN INTRODUCTION TO FITTING AND EVALUATING MIXED-EFFECTS MODELS IN R

Charles Nagle, Iowa State University

Mixed-effects modeling is a multidimensional statistical analysis capable of modeling complex relationships between predictor and outcome variables while accounting for random variance in various dimensions of the data. Although this technique is gaining popularity in applied linguistics research, learning how to model, and how to do so in R, can be intimidating. This guide provides an introduction to fitting mixed-effects models in R (Version 3.5.3) using RStudio. It includes a written introduction describing the modeling process, a video tutorial that focuses on getting started in RStudio, a sample data set, and an R script containing code to analyze the data. By the end of this introduction, researchers should have developed a basic understanding of the modeling process and should be able to (1) read data into R and inspect its structure, (2) create a series of plots to visualize trends and/or primary variables, and (3) fit and evaluate models.

INTRODUCTION

Mixed-effects models (also known as hierarchical or multilevel models) involve two fundamental components: fixed effects and random effects. Fixed effects are variables whose levels are defined, or do not change from one study to another. For example, lexical stress is a fixed effect because the levels are always reproducible across studies: stress vs. unstressed syllable, primary vs. secondary stress, etc. Gender is another classic example of a fixed effect since it is traditionally treated as a binary predictor: male vs. female. In contrast, random effects change across studies because they represent a random sample of a larger set. For instance, participants are treated as a random effect because the levels change across studies; participant 1 in study *x* is not the same person as participant 1 in study *y*. Raters and items can also be treated as random effects.

Mixed-effects models are superior to traditional analyses, such as ANOVA. Imagine that we collect data from 30 participants over four sessions, but at the last session, only 15 of our participants return. ANOVA employs listwise deletion (cases with missing data are excluded), leaving us with an analyzable sample of 15 participants; in other words, we lose all of the data for the 15 participants who completed the first three sessions. In contrast, mixed-effects models are robust in the face of missing data, which means that models are estimated using all available data points, even if some cells are missing (e.g., session 4 for 15 of 30 participants). This makes mixed-effects modeling an ideal approach for complex studies where attrition can be an issue, such as studies involving multiple tasks, sessions, or both. Mixed-effects models are also more flexible than ANOVA in terms of the assumptions that must be met. For example, the assumption of independence of observations is not required for mixed-effects models. Mixed-effects models are specifically designed to deal with dependent observations since we can treat various facets of the model as nested within one another. Finally, mixed-effects models allow for far more complex analyses, such as modeling curvilinear development and the effect of time-varying predictors, predictors whose values change over time (e.g., language use, motivation, etc.).

HOW TO USE THIS GUIDE

I began learning how to model five years ago. At the time, I did not know anything about R, so I actually learned how to use R as I was learning to model. From my perspective, learning how to model in R is remarkably similar to learning another language: as you learn how to model, you gradually restructure your knowledge, leading to a deeper and more intuitive understanding of the process, and as your familiarity increases, you become ready to learn about more complex topics. In fact, I still learn something new about modeling every time I fit models to a new data set. I share this information because I think it is important that you look at learning how to model in R as a longer-term endeavor whose payout will increase over time. This guide can serve as a starting point, but you will need to consult other resources and begin modeling your own data as soon as possible; as far as modeling is concerned, experience really is the best teacher. At the same time, I have tried to make the process as straightforward and anxiety-free as possible. In this guide, you will find step-by-step instructions on how to fit models to a sample data set using an annotated R script that I have provided. In other words, you will not need to write your own code at this stage. I have also recorded a video tutorial that will help you with preliminary steps, including setting up R and RStudio. I recommend watching the tutorial and reading this guide before modeling the accompanying data set.

Before you begin, you will need to install the latest version of R (https://www.r-project.org/) and RStudio (https://www.rstudio.com/). When you launch RStudio, it will automatically load R and ask you to create a new project in a new working directory. If you have already downloaded RStudio and created one or more projects, it may load an existing project. I prefer to create a new working directory for each project, saving all associated files (e.g., the master project files, R scripts, datasets, plots, etc.) into the folder. The written guide starts from loading the dataset and therefore assumes you have already loaded RStudio and created a new project. The video tutorial starts from opening RStudio, creating a project, and opening the script with the R code for data analysis. I will include a few illustrative screenshots of the RStudio interface in this written guide, but for information on where to click, see the video tutorial. Materials for this workshop can be accessed at https://iastate.box.com/s/bf0kerv0g17jnmqsdxgofgzldyo0ubgf.

TRANSLATING A STUDY INTO A MIXED-EFFECTS MODEL

We are going to use a data set similar to the one described in Nagle (2017). In that study, I was interested in how learners' production of L2 stop consonants changed over time. I created a set of fictitious characters to control for the phonetic context in which the stop appeared and participants' familiarity with the target items. The four fictitious characters relevant to the present analysis are *Pafo*, *Bafo*, *Pamuso*, and *Bamuso*. In the first two characters, *Pafo* and *Bafo*, the stop occurs in a stressed syllable, whereas in *Pamuso* and *Bamuso*, the stop occurs in an unstressed syllable. The outcome variable was voice onset time (VOT), an acoustic measure that represents the time that elapses between voicing onset and the release of the stop closure.

For the purpose of modeling, we will work with a data set consisting of 24 L1-English university students that I recruited from various sections of a second-semester Spanish course. Some participants had taken Spanish classes in elementary school and high school and were placed into the second-semester course, whereas others had begun learning Spanish at university. Learners

participated in five sessions over their second, third, and fourth semesters of Spanish, at approximately half-semester intervals. At each session, they completed a sentence formation task and a reading task. On the sentence formation task, they saw pictures representing one of the characters, a verb, and an object or location. Using these pictures, they created simple sentences in Spanish, such as *Pafo corre en el parque* ('Pafo runs/is running in the park'). On the reading task, they saw a similar sentence printed on the computer screen and read it aloud. Ten sentences were elicited for each target character.

From this point forward, variables will appear in italics. We have the following variables in the "VOT data final.csv" data set (levels are labeled as they appear in the data set):

- *id*: categorical, 24 levels (one per participant)
- *session*: continuous, 0 to 4 (could also be treated as a factor if sessions were not evenly spaced)
- *task*: categorical, two levels (formation, reading)
- *stress*: categorical, two levels (stressed, unstressed)
- *phone*: categorical, two levels (b, p)
- *item*: categorical, four levels (this is a dummy variable that shows the character names)
- *age of learning (aol)*: continuous, the age at which the participant began learning Spanish
- *previous experience (pe)*: continuous, the number of years of Spanish participants had taken before the study
- *class*: categorical, two levels (a, b), this is a variable I have added to the data set to illustrate the principle of nesting (i.e., this variable was not part of the original study)
- *vot*: continuous, dependent variable

Before we translate this design into mixed-effects models, we should review some facts about stop consonant VOT. In English, word-initial voiceless stops such as /p/ are aspirated, or produced with a strong burst of air that delays the onset of voicing in the following segment for about 30 to 60 milliseconds depending on point of articulation (closer to 30 for bilabial /p/ and 60 for velar /k/). In contrast, voiceless stops in Spanish are unaspirated, which means that the delay between the release of the stop and the onset of voicing in the following segment is very short, ranging from 10 to 30 milliseconds. Consequently, English speakers need to minimize the burst of air that occurs on stop release to produce more Spanish-like voiceless stops. Voiced stops in English, such as /b/, are variably realized as either voiced or voiceless unaspirated. In Spanish, voiced stops are always voiced, so English speakers who produce voiced stops need to learn that only voiced variants are used in Spanish, and speakers who do not produce voiced stops need to acquire them. English speakers need to produce shorter VOT values for Spanish /p/ and negative VOT values for Spanish /b/, since negative VOT is a coding convention that indicates that voicing begins before the stop is released (i.e., that the stop is produced with voicing during closure).

Our modeling will focus on VOT in Spanish /p/, a continuous outcome whose lower and upper limits are approximately 10 and 100 milliseconds. We will focus on modeling development over time, or how *vot* changes over *session*, while examining how *task* and *stress* affect *vot*. We will also incorporate *aol* and *pe* as covariates to control for their potential relationship with *vot* (e.g., perhaps learners with an earlier *aol* produce more accurate *vot*). We could also investigate whether learners improve their VOT production more rapidly on one task, which would involve a *session* × *task* interaction.

We can model these relationships as fixed and random effects. Fixed effects essentially represent the group trend, and random effects can be conceptualized as individual variation around that trend. For instance, if we include *session* as a fixed effect, we are modeling rate of change in *vot* for a prototypical participant, pooling all of the individual data. If we include *session* as a by-subject random effect, we are modeling individual variation in rate of change. In other words, we are instructing our model to estimate a unique rate of change for each individual in the data set, or each of our 24 participants. In principle, we could include a random effect for each of our fixed effects, creating a maximal random effects structure (Barr, Levy, Scheepers, & Tily, 2013). Likewise, we could include another group of random effects, such as by-item random effects. The design of the current study includes only two target items for /p/, *Pafo* representing the stressed condition and *Pamuso* representing the unstressed condition. If we had ten target items per condition, then we could introduce by-item random effects to account for random variance associated with the particular target items we had selected.

PREPARING DATA FOR MODELING

With the conceptual basis of our model in place, we can now inspect, analyze, and plot the data in R. I recommend that you open the R script "Script for PSLLT Proceedings Article" so that you can follow each of the steps outlined below. All R code provided in this written guide will appear in Calibri. In addition to the baseline R packages, we will need the following packages: "lme4" (Version 1.1-21; Bates, Maechler, Bolker, & Walker, 2014) to fit the models, "lmerTest" (Version 3.1-0; Kuznetsova, Brockhoff, & Christensen, 2017) to produce *p* values for fixed effects, and "ggplot2" (Version 3.1.0; Wickham, 2016) to plot the data.

To install these packages, we use the install.packages() function. We could install each package separately using three commands, or we can install all of them simultaneously by telling R that we have multiple items, which is generally what c() does in R code.

```
install.packages(c("lme4", "lmerTest", "ggplot2"))
```

Having installed the packages, we now need to load or activate them using the library() function. In this case, we must do so individually; we cannot load all three packages simultaneously using c().

```
library(lme4)
library(lmerTest)
library(ggplot2)
```

As we build our models and plots, we will create objects in R. The text that appears to the left of the arrow (which is actually the less than sign and a dash) is the name we are giving the object, and the text that appears to the right of the arrow refers to the function(s) that we are executing to create that object. First, we need to read our data into R using the read.csv() function.

```
data <- read.csv("VOT data.csv", row.names = NULL)
```

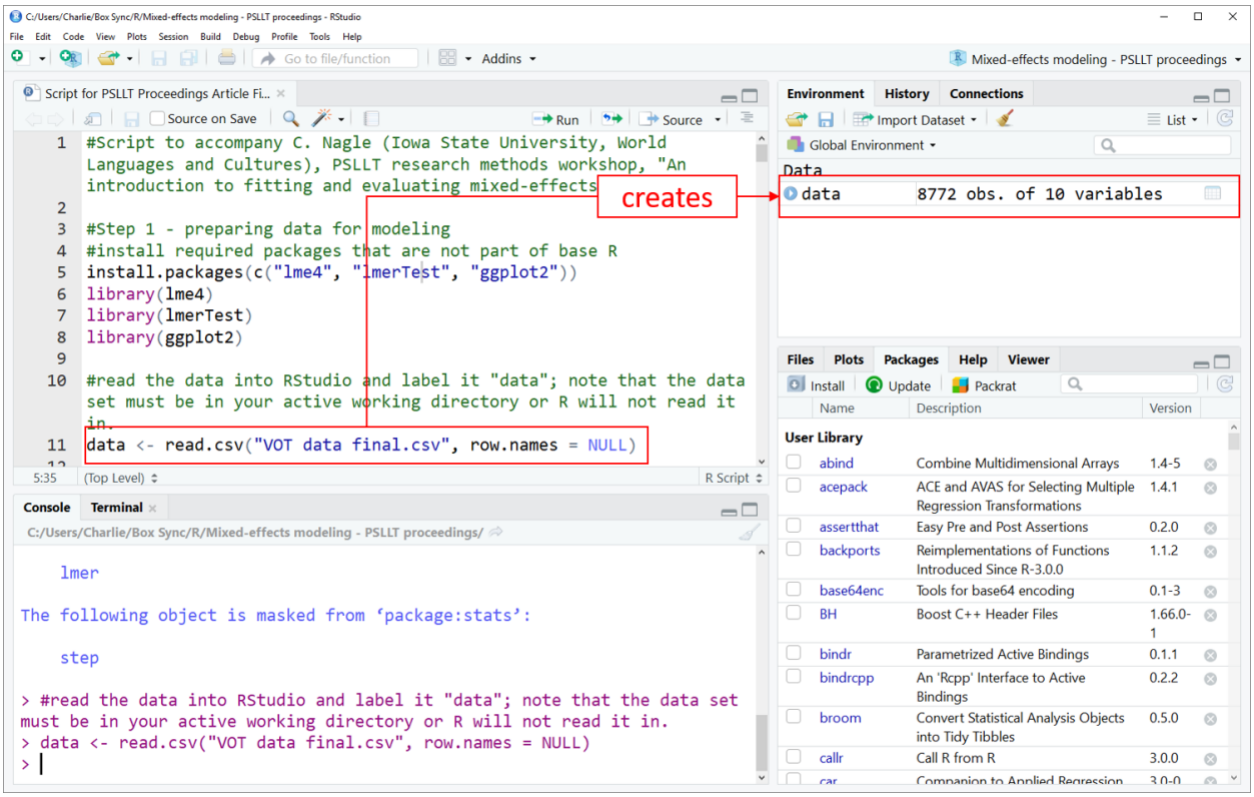

Figure 1. Screenshot of RStudio interface showing "data" dataframe on the Environment tab. The font, size, and background of the interface will depend on your settings in tools > global options > appearance.

We now have a dataframe named "data" (Figure 1) that we can inspect using the str() function. We should inspect every dataframe to make sure that R has interpreted our data structure properly. In my research, I typically use numbers to refer to participants (e.g., 1, 2, 3, etc.). R interprets numbers as integers or continuous variables. To avoid this, you could label participants with letters or a combination of a letter and number (i1, i2, i3, etc.). However, if you like to use numbers like I do, then we can use the as.factor() function to tell R that *id* is a categorical variable. This function is slightly more complicated since we need to use $ to tell R to look inside the dataframe for the *id* variable and interpret it as a factor.

```
str(data)
data$id <- as.factor(data$id)
```

Now if we reinspect the data using the str() function, we see that R is interpreting *id* as a factor. All of our other variables have been interpreted correctly. We are going to focus on the /p/ data for this analysis, so we need to create a new data set consisting of only the /p/ data using the subset() function.

```
data.p <- subset(data, phone == "p")
```

From this point forward, we will work with the "data.p" dataframe. The last step before we begin analyzing is to check for normality using the qqnorm() function. If our continuous variable is normally distributed, the points will fall more or less on a straight diagonal line. While slight deviations are acceptable, major departures indicate that the distribution is not normal.

```
qqnorm(data.p$vot)
```

Immediately from the plot we can see that some participants produced negative VOT (or voiced variants) for /p/. For these target items, participants probably made a mistake, reading /p/ as /b/, so it makes sense to eliminate these few outliers from the data set and retest for normality. We can use subset() to remake the data set including only items for which VOT > 0. In this case, we are modifying (overwriting) our dataframe instead of creating a new one, so we include "data.p" to the left of the arrow.

```
data.p <- subset(data.p, vot > 0)
```

If we look at the environment tab, typically displayed in the upper right corner of RStudio, we can see that the number of observations decreased from 4392 to 4375 when we executed the subset function, which tells us that we have eliminated 17 observations whose VOT ≤ 0. Now we check for normality again, and see that the data is beginning to look more normal since we have eliminated outliers on the lower end. However, we can still see that there is a relatively substantial curve in the line, so we will transform the data to enhance normality. Before we do, we should look at histogram of *vot*, which can help us determine what type of transformation to apply. We can generate a plot object (Figure 2) using the following code:

```
histogram.p <- ggplot(data.p, aes(x = vot)) +
    geom_histogram()
```

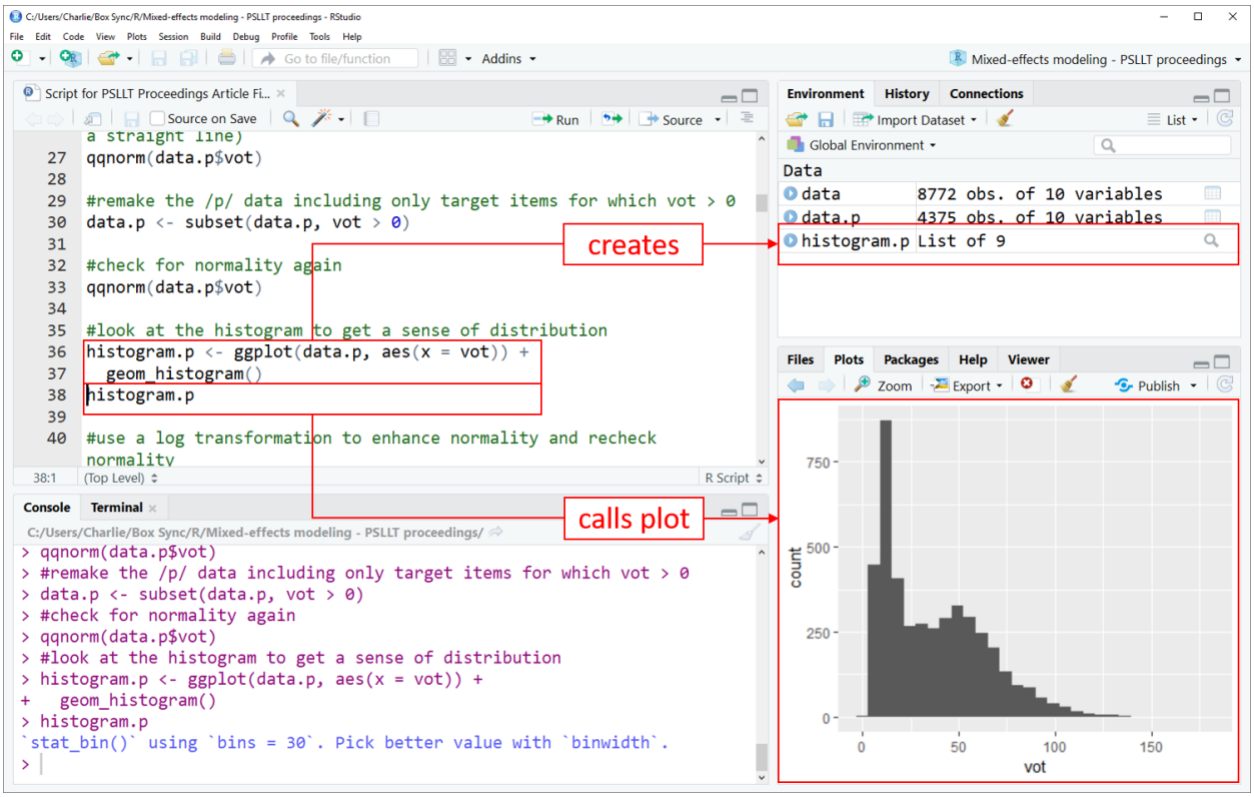

Figure 2. Screenshot of code to generate "histogram.p" plot and code to call the plot in the "Plots" tab.

The histogram seems to suggest that the data is somewhat log-normal, making a log transformation appropriate. Consequently, we will add a new variable, *log_vot*, to our dataframe using the log() function.

```
data.p$log_vot <- log(data.p$vot)
```

Now when we check normality of *log_vot*, qqnorm(data.p$log_vot), we see that we have a reasonably straight line. We will build models using *log_vot* as our dependent variable.

VISUALIZING THE DATA

When we model, we are trying to represent the data as accurately as possible, assessing relationships between our predictors and outcome. Plotting provides insight into the data and helps us fit the appropriate model, particularly when we are dealing with complex data sets. For longitudinal data in particular, we are trying to visualize the shape of the developmental curve so that we can specify the time predictor appropriately. For instance, if we see that development slows down over time, then we could include linear and quadratic time variables to estimate linear and quadratic rates of change (i.e., rate of change and rate of deceleration).

I always generate at least two primary plots, one that summarizes the group trajectory against individual trajectories (plots that include individual trajectories are sometimes referred to as

spaghetti plots) and another that plots trajectories for individual participants over time. We will use the "ggplot2" package to make our plots. Code for this package can be complicated depending on what we want to graph and how we want it displayed. I recommend starting each piece of code on a new line for the sake of readability. Make sure to include a plus sign (+) at the end of each line, except the last, so that R continues reading the next piece of code. If you take this approach, R will automatically indent lines below the first to indicate that indented lines pertain to a larger block of code, as in the following example.

```
plot.group.p  <- ggplot(data.p, aes(session, vot)) +
    stat_smooth(method = "loess", se = F, size = 2) +
    stat_summary(aes(group = id), fun.y = mean, geom = "line", alpha = 0.3) +
    xlab("Session") +
    ylab("Voice Onset Time")
```

This creates the basic plot. However, I like to remove the gridlines and use a black and white theme, which we can accomplish by adding the following pieces of code. The two sets of code will generate the plot displayed in Figure 3.

```
    theme_bw() +
    theme(strip.background = element_blank()) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```

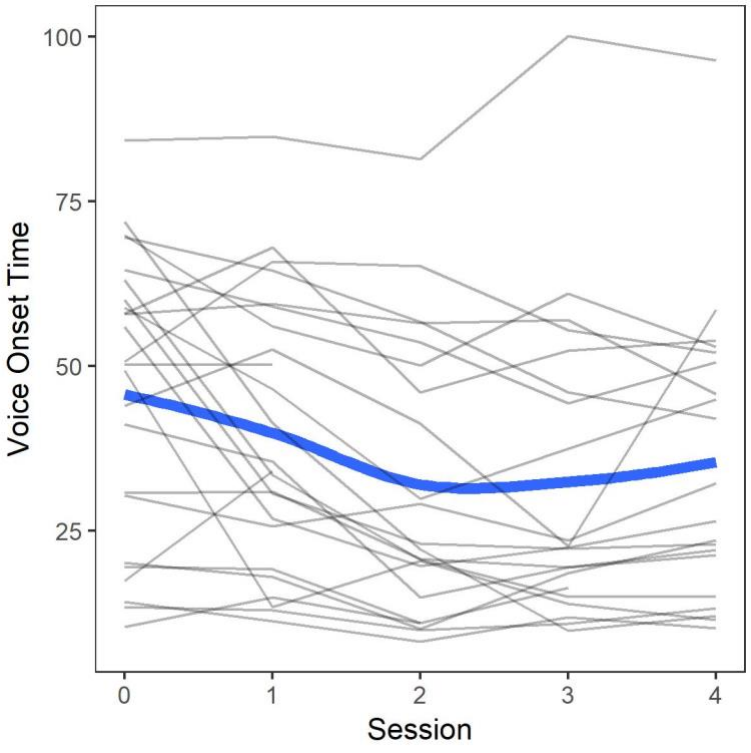

Figure 3. Group trend in VOT for Spanish /p/ versus individual trajectories.

If we want to save our plot, we use the ggsave() function, specifying the output file, the plot object in R, and the dimensions of the plot in inches and its dpi (300 is typically required by most journals).

```
ggsave("group plot p data.jpeg", plot.group.p, width = 4, height = 4, dpi = 300)
```

We can rework the code for the group plot to make individual boxes for each participant by telling R to facet (or array the data) by *id* using the facet_wrap() function. We can also optionally include the number of columns and rows if we want a particular configuration. In this case, we will specify six columns (generating four rows) since we have 24 participants.

```
plot.individual.p  <- ggplot(data.p, aes(session, vot)) +
    stat_summary(fun.y = mean, geom = "line") +
    xlab("Session") +
    ylab("Voice Onset Time") +
    theme_bw() +
    theme(strip.background = element_blank()) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
    facet_wrap(~id, ncol = 6)
```

We then save this plot using the ggsave() function, adjusting the dimensions to fit the plot. In general, getting the dimensions of the plots right requires some trial and error, so I think the simplest approach is to save the file, copy it into a Word document, and adjust the dimensions as needed until it displays correctly. For example, I specified dimensions of 6.5” width and 4” height to create Figure 4.

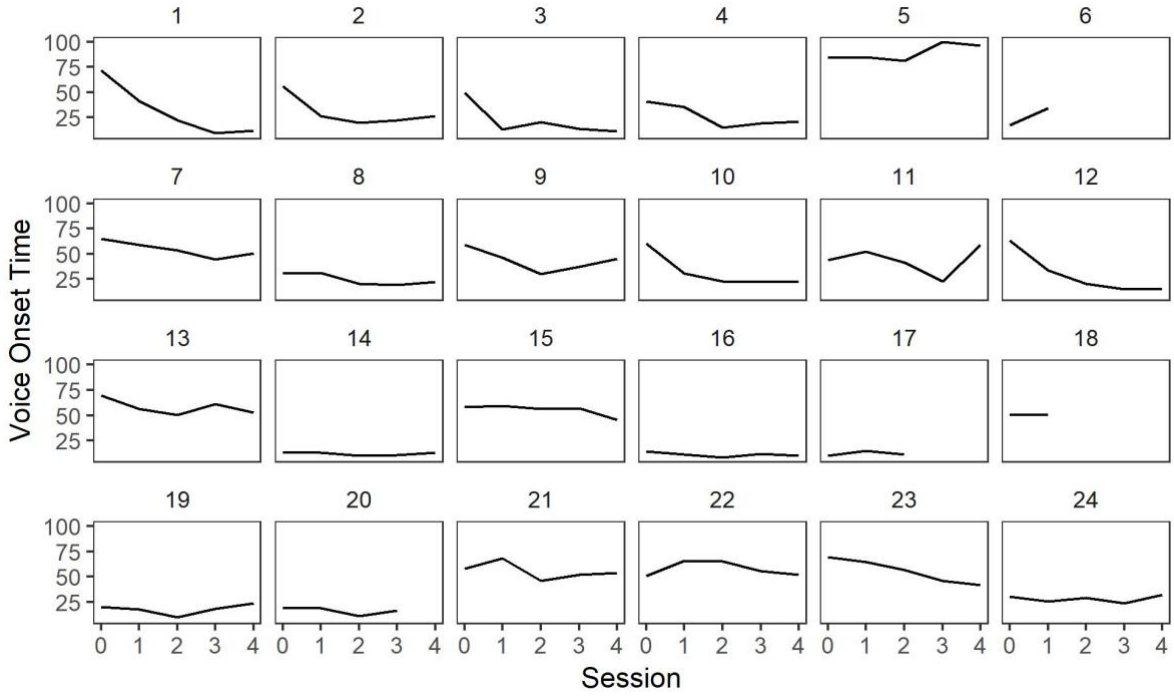

Figure 4. Individual trajectories in VOT for Spanish /p/.

These two plots (Figures 3 and 4) show us that most participants improved their VOT production since we see a downward trend toward shorter, more Spanish-like VOT in /p/. We also see that for many individuals, VOT improved more over the first half of the study, suggesting that we should try modeling linear and quadratic rates of change. In both plots, we can see that some learners did not participate in all five sessions (e.g., 6, 18). As I mentioned in the introduction, mixed-effects models can handle missing data, so these cases are not problematic.

We could also generate a plot to illustrate the effect of stress on VOT production. In the code below, we map line type to stress (we could also map color, but since many journals print in greyscale, I try to avoid using color to differentiate conditions or groups) within the aes() function and include an additional line of code to move the legend from its default location (vertical display to the right of the plot) to the bottom of the plot to avoid compressing the x-axis.

```
plot.individual.p  <- ggplot(data.p, aes(session, vot, linetype = stress)) +
    stat_summary(fun.y = mean, geom = "line") +
    xlab("Session") +
    ylab("Voice Onset Time") +
    theme_bw() +
    theme(strip.background = element_blank()) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
    facet_wrap(~id, ncol = 6) +
    theme(legend.position = "bottom")
```

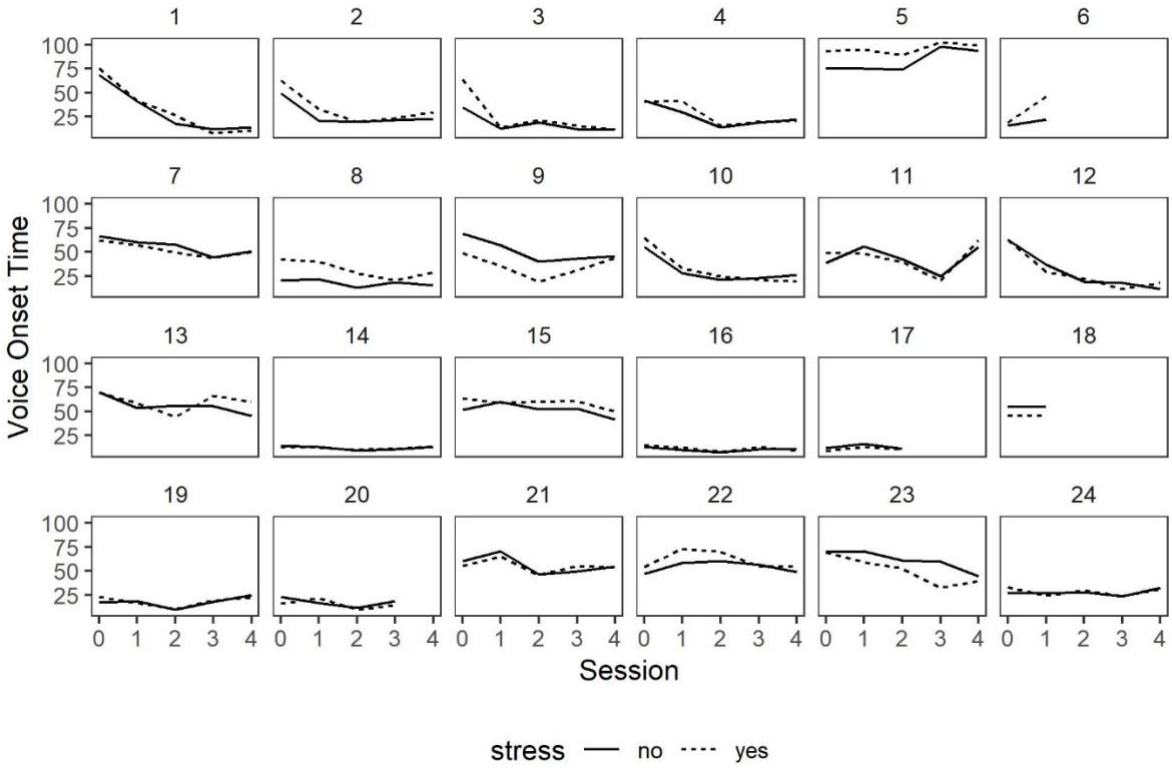

Figure 5. Individual trajectories in VOT for Spanish /p/ in stressed and unstressed environments.

FITTING MIXED-EFFECTS MODELS TO THE DATA

Before we can begin modeling, we should create a variable that represents the quadratic trend for time by squaring *session*. We want to create this variable because in Figures 3 and 4 we observed curvature in the group and individual trajectories. In other words, VOT production over time did not follow a straight line (cf. participants 1, 2, 10, 12, etc. in Figure 4 above). We can approximate this curvature using a quadratic function for *session*, or *session_sq*.

```
data.p$session_sq  <- data.p$session^2
```

We should also center *aol* and *pe*, our continuous predictors. Centering is essentially a form of standardizing variables, making them easier to interpret without altering the model (coefficients will change but significant effects will not). There are various ways to center, but grand-mean centering makes the most sense for our predictors. In this form of centering, we compute the mean and subtract it from each participant's score on the relevant variable. In our centered variables, a negative score indicates that the participant was below the mean, a positive score that the participant was above the mean, and a score of zero refers to the mean. For example, *aol* refers to the age at which participants began learning Spanish. When we fit a model, the intercept is computed by setting all predictors to zero, but a score of zero is not possible in our data (i.e., a score of zero would in theory represent a native speaker of Spanish). Centering resolves this issue by setting zero to refer to the sample mean. A score of zero is possible for *pe* because some participants had not taken Spanish before enrolling in university language coursework. However, it is still advantageous to center *pe* to represent the average amount of previous experience that participants had, since ultimately we are trying to model a prototypical participant's trajectory. We can create the centered predictors and add them to our data set in R.

```
data.p$aol_c  <- data.p$aol – mean(data.p$aol)
data.p$pe_c  <- data.p$pe – mean(data.p$pe)
```

We are now in a position to begin modeling. Scholars have advocated for a variety of approaches to modeling, but the most common is forward-testing random effects and backward-testing fixed effects. This means that we will add random effects one by one and compare models to one another, and we will add fixed effects as a group and compare models by progressively dropping the least significant effects. Even though backward-testing fixed effects is generally advisable, this does not mean that we should include every possible fixed effect. Rather, our fixed effects should be guided by our theoretical framework and the design of our study. In certain scenarios, we may decide to retain a fixed effect even if is not significant. For instance, if we are interested in higher order interactions among predictors, we would not eliminate their baseline components.

In practice, backward-testing fixed effects can be challenging when you first transition to mixed-effects models. In my view, a data-driven, bottom-up approach (Cunnings & Finlayson, 2015) is perfectly acceptable so long as you report your modeling process as transparently as possible, including the model containing your "final" set of predictors. The data-driven approach can be particularly advantageous when dealing with longitudinal data since decisions need to be made about the shape of development over time. In some of my previous work (Nagle, 2017a, 2017b), I opted to forward-test models, reporting all of the models I built in a table or appendix to illustrate

the process (for examples of how to format such a table, see Murakami, 2016; Singer & Willett, 2003).

For the purpose of this illustration, we will take a hybrid approach. First, we will build the unconditional growth model, which is a model that describes how VOT, our dependent variable, changes over time. To this model, we will add task and stress, our primary predictor variables, using backward-testing to evaluate the fixed terms and forward-testing to evaluate the random terms. To build our models we will use the lmer() function, which takes the following general form. Note that I place fixed and random effects on separate lines so that the code is easier to read.

```
name of model  <- lmer(dependent variable ~ fixed1 + fixed2 +
    (random1 + random2 | random grouping term 1) +
    (random1 + random2 | random grouping term 2), data = name of dataframe)
```

First, we build the null or random intercepts model:

```
null.p  <- lmer(log_vot ~ 1 + (1 | id), data = data.p, REML = F)
```

In the code above, we are creating a model, "null.p," in which *log_vot* is the dependent variable. We have only one fixed effect, the intercept, represented by the *1* after the tilde (~), and we have included by-subject random intercepts using the code (1 | id). In this piece of code, the random effects appear to the left of the vertical bar, and the grouping term over which they are computed to the right. We have also specified that our data set is "data.p," and we have told the model to use maximum likelihood estimation rather than restricted maximum likelihood estimation (REML) so that we can compare models with different fixed-effects structures to one another. If we were interested in comparing models with the same fixed effects but different random effects, then we could fit and compare REML models. In lmer(), REML is the default, so we turn it off using the code REML = F.

Next, we build unconditional linear and quadratic growth models using *session* and *session_sq*. Unconditional growth simply means that we have not yet included any predictors that would affect the intercept or rate of change over time (i.e., we have not yet placed any conditions on the intercept or rates of change). R will always include intercepts unless we suppress them, so we do not need to carry *1* forward in our model specification.

```
linear.p  <- lmer(log_vot ~ session +
    (session | id), data = data.p, REML = F)

quadratic.p  <- lmer(log_vot ~ session +
    session_sq +
    (session + session_sq | id), data = data.p, REML = F)
```

The quadratic model with by-subject random slopes for session_sq fails to converge, which is not uncommon for models involving large data sets and/or complex random effects structures. Failure to converge does not always indicate a problem with model specificiation. There are a number of ways to facilitate convergence, such as using a different optimization function, removing

covariances among random effects, and simplifying the random effects. For the purpose of this guide, we will instruct R to use the BOBYQA optimizer instead of the default nloptwrap implemented in lme4 version 1.1-20 (for more information on convergence, see the FAQ section at the end of this guide). We can do this using the following code. For the sake of readability, I will move the data and fit specifications to a separate line.

```
quadratic.p  <- lmer(log_vot ~ session +
    session_sq +
    (session + session_sq | id),
    data = data.p, REML = F, lmerControl(optimizer = "bobyqa "))
```

Now we compare the three models (null, linear growth, and quadratic growth, each with the accompanying by-subject random effects) using the anova() function. This function performs a chi-square test on the change in the deviance statistic for nested models, or models that can be derived from one another by setting one or more parameters to zero.

```
anova(null.p, linear.p, quadratic.p)
```

The output in R (Figure 6) shows that each model is an improvement over its predecessor, which is not surprising since our plotting already revealed a quadratic trend in both the group and individual data. We can also see that the change in degrees of freedom (the Df column) is 3 for the comparison between the null.p and linear.p models, and 4 for the comparison between the linear.p and quadratic.p models. This may seem odd since we only added two terms, one representing the fixed effect and one representing the random effect, to each model. When we added those terms, R also included covariances among the random effects: between *session* and the intercept for the linear.p model; between *session_sq* and the intercept, and *session_sq* and *session* for the quadratic.p model.

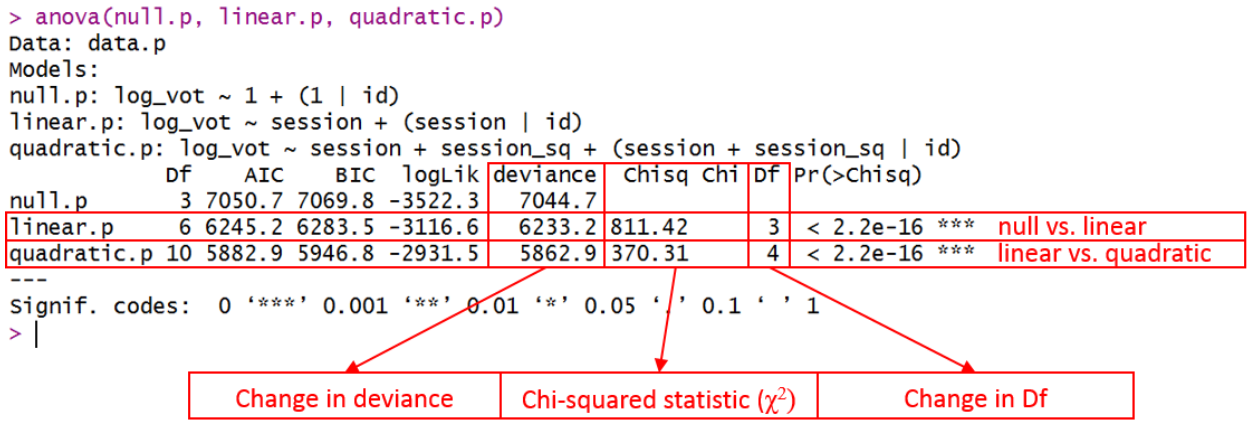

Figure 6. Screenshot of output for model comparisons using the ANOVA() function. Rows represent model comparisons (i.e., linear.p reports the null.p vs. linear.p comparison, and quadratic.p reports the linear.p vs. quadratic.p comparison).

We can request a summary (Figure 7) of our unconditional quadratic growth model using the summary() function.

```
summary(quadratic.p)
```

The first part of the summary shows the formula for the model we fit, followed by fit statistics (e.g., AIC, BIC, and deviance), and residuals. The random effects summarize the variance in intercepts, linear slopes (*session*), and quadratic slopes (*session_sq*), as well as the residual within-subjects variance in the model. The fixed effects, listed below the random effects portion of the model, demonstrate that there is a negative trajectory over the course of the study (i.e., the coefficient for *session* is negative), which in this case indicates improvement. The positive coefficient for the *session_sq*, when interpreted with respect to the negative coefficient for *session*, indicates that development decelerated over time. These findings align with the initial plots we generated. We can compare the magnitude of the coefficients and their directionality, but we must remember that we are fitting models to *log_vot*, so the coefficients do not refer to the intercept or rate of change on the original VOT scale.

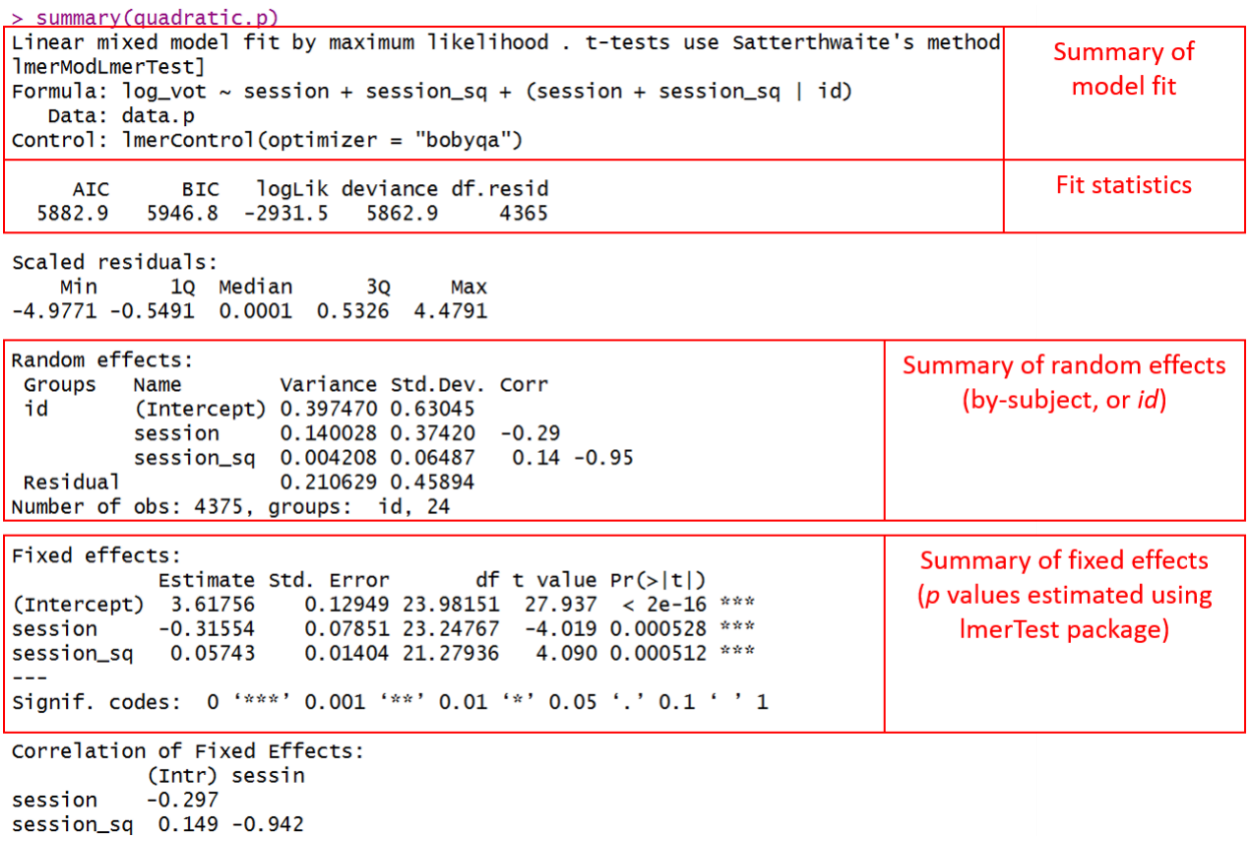

Figure 7. Screenshot of summary of quadratic.p.

We are now ready to integrate our remaining fixed-effect predictors (*stress*, *task*, *aol_c*, and *pe_c*) as a block and backwards test them. In the following code, I have included our time predictors, *session* and *session_sq*, on the same line so that you can more easily see the four new fixed effects we have added.

```
fm1.p  <- lmer(log_vot ~ session + session_sq +
    task + stress + aol_c + pe_c +
    (session + session_sq | id),
    data = data.p, REML = F, lmerControl(optimizer = "bobyqa "))
```

Again, we ask for a summary of the model using the summary() function. In the list of fixed effects, *task* is labeled "taskreading" and *stress* is labeled "stressyes." R interprets categorical predictors alphabetically. Thus, the sentence formation task, "formation," and the unstressed context, "no," have been set as the baseline conditions against which the reading task and stressed context are compared (for information on contrast coding categorical predictors, see Linck & Cunnings, 2015). In other words, the intercept refers to VOT production on the formation task when the stop occurs in an unstressed syllable. If we have installed the "lmerTest" package, then summary() will return *p* value estimates for each predictor.

We can see from our summary that *task* is significant; the negative coefficient for "taskreading" indicates that on average, participants produced lower VOT values on the reading task than on sentence formation, and the large *t* value suggests that this effect was relatively robust. In contrast, *stress* was not significantly related to VOT production, and neither were our grand-mean centered covariates, *aol_c* and *pe_c*. We now have two options. Following the principle of backward-testing fixed effects, we could rank these predictors in terms of their *t* value and begin dropping them from the model in the following order: *pe_c*, *stress*, and *aol_c*. In this case, we would need to report the order in which we dropped the nonsignificant fixed effects and the corresponding anova() model comparisons at each stage. The second option would be to retain all effects, reporting our final model so that readers can more easily compare and contrast estimates, standard errors, and *t* values for all of the fixed effects included in the study. In my view, it is important to create a parismonious model that does not include a large number of nonsignificant fixed effects, but it is not always advisable to strive for the minimally adequate model, or a model that contains only those fixed effects that enhance fit (i.e., only significant fixed effects).

For our data set, we will refit the model without stress and compare the simpler model to its more complex predecessor. However, we will keep *aol_c* and *pe_c* since we have included them as control covariates.

```
fm2.p  <- lmer(log_vot ~ session + session_sq +
    task + aol_c + pe_c +
    (session + session_sq | id),
    data = data.p, REML = F, lmerControl(optimizer = "bobyqa "))
anova(fm1.p, fm2.p)
```

The chi-square statistic ($\chi^2(1) = .35$, $p = .56$) indicates that including *stress* does not significantly change model fit, so we can drop it from the model. However, in discussing the models, we would still report and interpret stress since lack of significance is an important finding that should not be ignored.

Now we can focus on our random effects. Some scholars have advocated for a maximal approach, which means including a random effect for every fixed effect (Barr et al., 2013; Cunnings &

Finlayson, 2015). In my view, fitting an appropriate set of random effects can be challenging, so I would not recommend taking a maximal approach. For the sake of our model, we will add a random effect for *task* to see if it improves the model, keeping in mind that as we add more random effects, the model may take longer to converge (i.e., random effects and their covariances can be computationally intensive). When we model *task* as a by-subject random effect, we are allowing R to estimate unique coefficients for *task* for each participant. Put another way, for some participants, *task* may have had a strong impact on VOT production (i.e., large differences in VOT by task), whereas for others its effect may have been comparatively weak (i.e., small differences in VOT by task). We test this possibility by including *task* as a random effect.

```
fm3.p  <- lmer(log_vot ~ session + session_sq +
    task + aol_c + pe_c +
    (session + session_sq + task | id),
    data = data.p, REML = F, lmerControl(optimizer = "bobyqa "))
anova(fm2.p, fm3.p)
```

Including *task* as a by-subject random effect has significantly improved fit ($\chi^2(4) = 93.05$, $p < .001$), so we will keep it in the model. We can now consider this our "final" model, or the model that is the best representation of our data given our research aims and predictors. Now that we have a final model, we should calculate 95% confidence intervals for fixed effects using the confint() function. Profiling confidence intervals can take a very long time depending on the complexity of the model and the computer's processor. Using an older desktop with eight gigabytes of RAM, I waited nearly 15 minutes for R to produce confidence intervals before stopping confint().

```
confint(fm3.p)
```

We can speed up the process by approximating the confidence intervals using the Wald method, which is far quicker (less than one second on the same machine). If we approximate the intervals, then we should report this in the manuscript.

```
confint(fm3.p, method = "Wald")
```

Table 1 reports the final model following Linck and Cunning's (2015) format, including a note to indicate that the confidence intervals are approximate. I have assigned the fixed effects (i.e., parameters) a more informative label instead of using the variable names as they appear in our data set.

Table 1

Summary of components in mixed-effects model of VOT development in L2 /p/

	Fixed effects					*Random effects* By Subject
Parameter	Estimate	*SE*	95% CI	*t*	*p*	*SD*
Intercept	3.70	.13	[3.45, 3.95]	28.91	< .001	.62
Linear slope	–.31	.08	[–.47, –.16]	–4.02	.001	.37
Quadratic slope	.06	.01	[.03, .08]	4.11	.001	.06
Task: Reading	–.17	.04	[–.24, –.10]	–4.64	< .001	.17
Age of learning	.05	.08	[–.10, .20]	.63	.53	
Previous experience	.05	.10	[–.15, .24]	.46	.65	

Note. 95% CI were approximated using the Wald method.

Finally, there are a number of ways to evaluate how well our model fits the data, such as plotting fitted against residual values. In my experience, the latter typically works well for linear relationships but can be misleading when modeling polynomial change, such as the quadratic term (*session_sq*) we included in our models. One simple alternative is to generate a set of predicted values based on our model and then compare predicted growth to observed growth over time. Including this type of plot as an appendix or supplementary file can be helpful. We can generate predicted values and add them to our data set using the predict() function.

```
data.p$predicted <- predit(fm3.p)
```

We currently have two dependent variables, *log_vot* and *predicted*, that are stored in two separate columns. If we want to plot the predicted and observed data on the same plot, we need to transform the data into a new longitudinal data set by merging the two dependent variables into a single column and creating a new identifier variable. There are many packages and approaches we could take, but I prefer the "tidyverse" package (Version 1.2.1). In the R code below, I use the gather() function. We include the data set (data.p), the name of the new identifier column (*type*, for type of data: observed vs. predicted), the name of the new outcome variable (*log_vot2*), and the variables to be merged (*log_vot* and *predicted*). We also use c() to tell R that we are dealing with multiple variables.

```
install.packages("tidyverse")
library(tidyverse)
data.predicted <- gather(data.p, type, log_vot2, c(log_vot, predicted), factor_key = TRUE)
```

We can now plot the data as before, using our code to generate individual plots while mapping *type* to line type:

```
plot.predicted.p <- ggplot(data.predicted, aes(session, log_vot2, linetype = type)) +
      stat_summary(fun.y = mean, geom = "line") +
      theme_bw() +
      theme(strip.background = element_blank()) +
      theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
      xlab("Session") +
      ylab("Log VOT") +
      facet_wrap(~id, ncol = 6) +
      theme(legend.position="bottom")
```

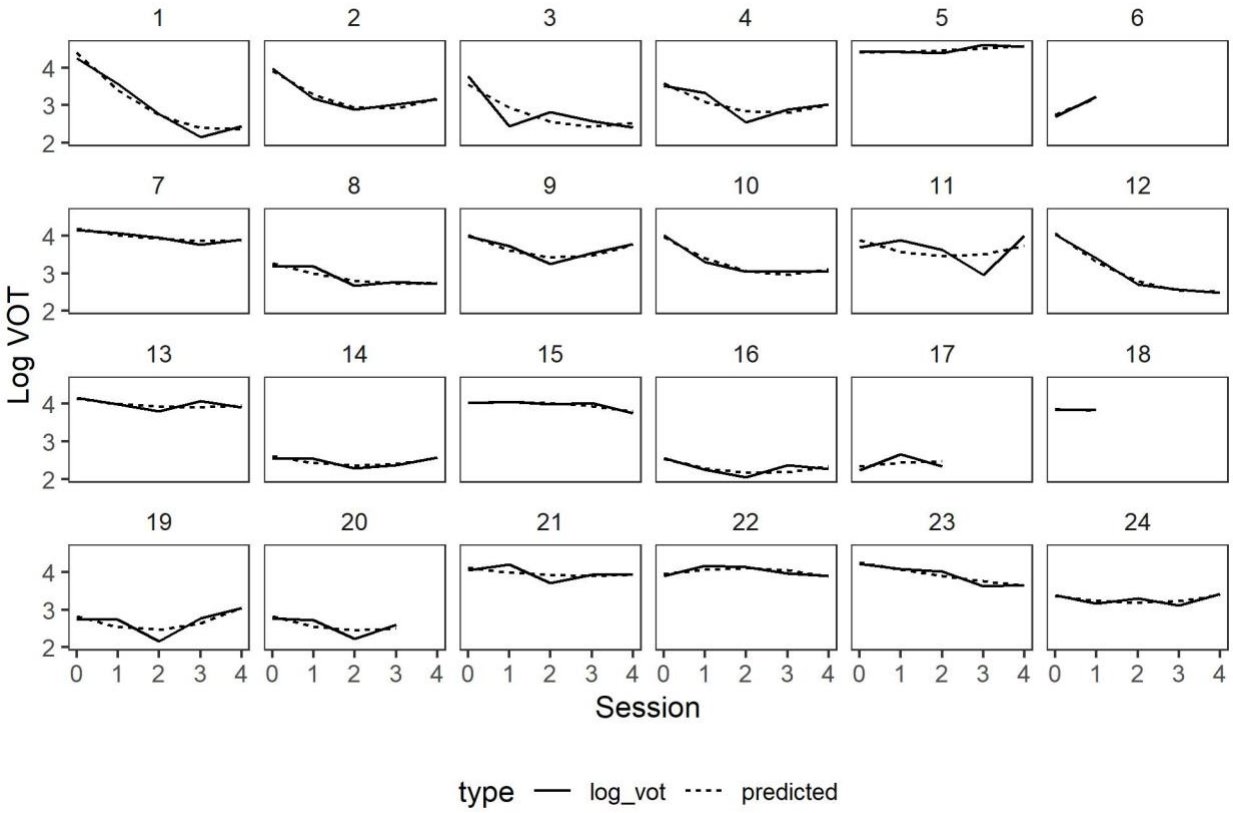

Figure 8. Observed (log_vot) vs. model-predicted individual trajectories.

Comparing the dotted lines in Figure 8, which represent the model-estimated values, we can see that each individual plot displays different rates of linear and quadratic change. This serves as a visual reminder that we instructed R to estimate unique rates of change for each participant in the data set by including *session* and *session_sq* in the random-effects structure of our model. We can also see that the model represents the data reasonably well.

One last plot that we might be interested in generating is the model-estimated group trajectory. This plot is similar to Figure 3, but we will graph the model trajectory as a dashed line. In general, I use solid lines for observed data and dashed lines for model-estimated data as appropriate.

```
plot.model.p <- ggplot(data.p, aes(session, log_vot)) +
    stat_summary(aes(y = fitted(fm3.p)), fun.y = mean, geom = "line", linetype = "dashed", size
    = 2) +
    stat_summary(aes(session, log_vot, group = id), fun.y = mean, geom = "line", alpha = 0.2) +
    theme_bw() +
    theme(strip.background = element_blank()) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
    xlab("Session") +
    ylab("Log VOT")
```

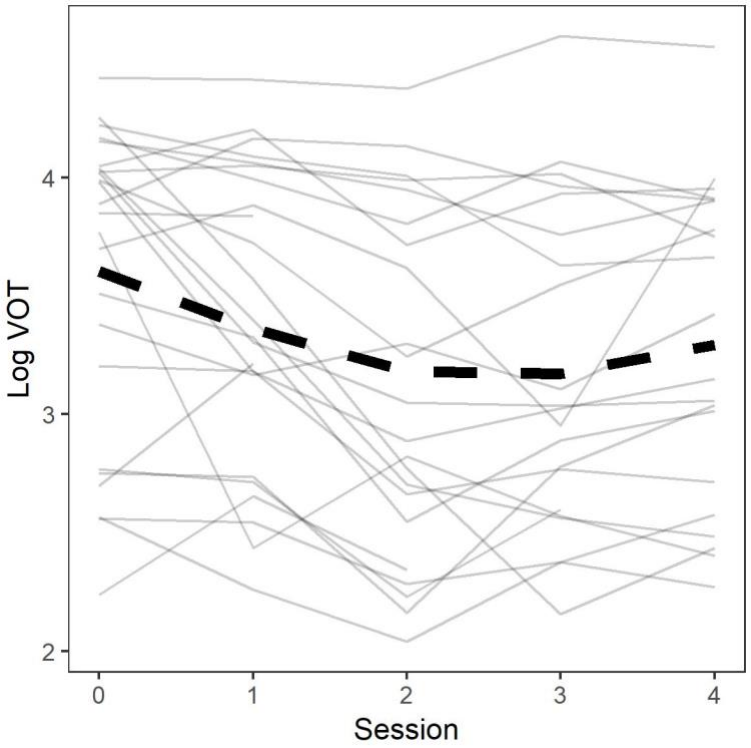

Figure 8. Model-estimated group trajectory vs. observed individual trajectories.

MODELING NESTED DATA

Up until now, we have assumed that we have drawn a random sample of students from a variety of sections of the same course, and that these students over time have had different instructors. Now we will consider another case. Let's assume that we recruited students from two different sections of Spanish, labeled a and b in the data set, and followed them over a single semester of coursework. In this scenario, we can say that the students are nested in classes. If we had a multisite design, then students would be nested in classes and classes nested in schools (the latter would be a three-level model). Modeling nesting is important because each class (or school) may display a unique growth rate, and we would expect growth rates for students in the same class to be more similar to one another than to growth rates for students in different classes (e.g., higher correlation within classes). The R code for creating nesting is a forward slash with the higher-order group first

and the lower-order group second, such as class/id or school/class/id. This syntax gets included as a grouping factor for the random effects (i.e., on the right side of the vertical bar).

```
fm.p.nested  <- lmer(log_vot ~ session + session_sq +
    task + aol_c + pe_c +
    (session + session_sq + task| class/id),
    data = data.p, REML = F, lmerControl(optimizer = "bobyqa "))
```

If we generate a summary of the nested model using the summary() function, we can see that the random effects structure now includes both "id:class" and "class." We now have estimates for the variance among classes and the variance among students, taking into consideration the fact that students were nested in classes. However, when we fit the model, R returned a singular fit warning, which means that the model was over-specified. Singular fit is not surprising since in our data set we have very few observations for class (n = 2). Thus, estimating unique linear and quadratic slopes for each class would not be advisable. If we were interested in variance among classes, we would probably want to achieve a sample size of at least 10–20 classes, with 10–20 students in each class.

FREQUENTLY ASKED QUESTIONS

1. What if one of the variance components in the random effects structure of my model is very small?

   Interpret the variance components with respect to their corresponding fixed effect. A variance component that is very small relative to its fixed effect indicates that there is virtually no between-subjects variance in that parameter. In that case, even if the inclusion of the random effect significantly improves model fit, it may be advantageous to select a simpler model without that term. In our final model reported in Table 1, the between-subjects variance in quadratic slopes (*session_sq*) is .004. This is small relative to some of the other variance components, but proportionate relative to the fixed effect estimate (.06). For more information on model selection, see Murakami (2016).

2. What if my model fails to converge?

   More complex models may fail to converge. There are a number of solutions you can attempt. First, you can change the default optimizer, which is what we did using lmerControl(optimizer = "bobyqa"). You could also try fitting the model using all optimizers by fitting the model using the default optimizer and then running the allFit() function on your model object.

   Simplifying the random effects can also facilitate convergence. You can eliminate covariances among the random effects by including a double vertical bar: (1 + session + session_sq || id). If the model stills fails to converge, then simplifying the random effects could help (i.e., eliminating higher-order random effects such as interaction terms). For more information, see Barr et al. (2013) and Linck and Cunnings (2015).

3. What if my outcome variable is categorical?

   Linear models are appropriate for continuous outcome variables. Generalized linear models (glimmer) are appropriate for categorical outcomes. Thus, glimmer would be appropriate for perception data that are coded as correct/incorrect, for production data that are coded as intelligible/unintelligible, etc. You can fit a generalized linear model using the glmer() function. The specification for glmer() is essentially the same as lmer(), but the interpretation of glimmer is not as straightforward. For more information on glmer(), see Baayen (2008) and Linck and Cunnings (2015).

4. What about ratings data?

   When comprehensibility, accentedness, and other ratings are carried out on a 1000-point sliding scale, they can be considered continuous. In that case, the same procedure outlined above can be followed, and rater would be included as a grouping for random effects. In other words, two sets of random effects would be expected: one grouping for speakers and another for raters. For example, had this study been a ratings study, we might expect the following random effects structure.

   ```
   (1 + session + session_sq | speaker) +
   (1 + session + session_sq | rater)
   ```

   When ratings are carried out on a shorter scale, such as a 9-point scale, then the data is ordinal. The best approach for modeling this type of data is to pool data over raters, in which case rater would not be included as a random effect. This will linearize the scale and make it suitable for modeling with lmer().

5. What if my data is not longitudinal?

   Most of my research is longitudinal, which is why I have concentrated on modeling longitudinal data. You can model cross-sectional data following the exact same procedure, but you will not need to introduce time predictors, such as *session* and *session_sq*, into the model.

6. What if I do not know or remember the R code for a particular function or analysis?

   First, if you are confused about a particular function, you can type a ? before the function and R will give you a description of what it does, such as ?lmer. The R community is also very large. In general, you can find what you are looking for by consulting the R cookbook (http://www.cookbook-r.com/) or searching R forums. For online searching, start by including the package and/or function you are using and a short description (e.g., fitting piece-wise growth models using lmer, modifying the x-axis in ggplot2). Do not be afraid to experiment with R code you find online, modifying it to meet your needs (this is precisely why I have included code in this document and in the accompanying R script). I look up information nearly every time I use R, so I now have a list of helpful bookmarks. You will acquire a similar set of bookmarks as you become more familiar with R and/or modeling.

7. How do I know if I have fit my model correctly and/or that the model is correct?

   Mixed-effects modeling is complicated because it involves some trial and error, and modeling experts from different disciplinary traditions recommend different approaches. Moreover, as the field evolves, recommendations will change. All of these factors can make modeling intimidating, but do not feel intimidated! The best way to learn modeling, and to learn R, is to start modeling your own data with this guide and the other excellent introductions that have been published (Cunnings & Finlayson, 2015; Linck & Cunnings, 2015). As you model and write up results for publication, report your process and results as clearly as possible so that reviewers can offer assistance. Over time, you will become more confident and develop a more intuitive sense of how to fit and evaluate models. In short, learning how to model takes time.

8. What other resources can you recommend?

   First, know that you can always write me with questions related to modeling and I will do my best to answer them. In general, to help someone fit and evaluate models, it is helpful to have a description of the study and data set as well as the R code that is being used. You should also consult the references contained in this introduction. If you are interested in step-by-step guides, see Linck and Cunnings (2015) for a general overview, Cunnings and Finlayson (2015) for modeling longitudinal data, and Baayen (2008). The latter is very comprehensive and especially helpful for working with reaction time data but could be overwhelming for beginners. If you are interested in the theory behind mixed-effects models, especially as applied to longitudinal data, see Singer and Willett (2003). If you are interested in a general R statistics book that includes information on mixed-effects models, see Field, Miles, and Field (2013). Finally, there are many recent publications featuring mixed-effects models that can serve as excellent resources such as Barrios, Namyst, Lau, Feldman, and Idsardi (2016) and Offerman and Olson (2016).

ACKNOWLEDGMENTS

I would like to thank Nick Pandža (Faculty Research Specialist, University of Maryland Center for Advanced Study of Language) for first introducing me to mixed-effects modeling in R. Over the past five years, his expertise, encouragement, and friendship have proven invaluable. I would also like to thank Dr. Jared Linck (Research Scientist, University of Maryland Center for Advanced Study of Language) for his help during the early stages of modeling and Dr. Philip Dixon (University Professor of Statistics, Iowa State University) for helping me review and interpret models when I arrived at Iowa State. Finally, I am grateful to Dr. John Levis (Professor of TESL and Applied Linguistics, Iowa State University) and Dr. Pavel Trofimovich (Professor of Applied Linguistics, Concordia University) for encouraging me to present and publish on modeling and for supporting my broader research program. I am lucky to have had the help I have had over the years, and I hope I can provide similar assistance and encouragement to other researchers in the future.

ABOUT THE AUTHOR

Charles Nagle is an Assistant Professor of Spanish and the Director of the Spanish Language Program at Iowa State University. His main area of research is in second language pronunciation and individual differences. He is particularly interested in how learners' pronunciation develops over time, including the extent to which changing patterns of motivation and language use affect development, and the relationship between the perception and production of second language sounds. He is also passionate about statistics, particularly mixed-effects modeling. At Iowa State, he teaches all levels of Spanish language and upper-level linguistics courses such as "Introduction to Spanish Phonology" and "Bilingualism in the Spanish-Speaking World."

REFERENCES

Baayen, H. R. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*. New York: Cambridge University Press.

Bates, D., Maechler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software, 67*(1), 1–48. doi: 10.18637/jss.v067.i01

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. doi:10.1016/j.jml.2012.11.001

Barrios, S. L., Namyst, A. M., Lau, E. F., Feldman, N. H., & Idsardi, W. J. (2016). Establishing new mappings between familiar phones: Neural and behavioral evidence for early automatic processing of nonnative contrasts. *Frontiers in Psychology*, *7*, 1-16. doi:10.3389/fpsyg.2016.00995

Cunnings, I., & Finlayson, I. (2015). Mixed effects modeling and longitudinal data analysis. In L. Plonsky (Ed.), *Advancing Quantitative Methods in Second Language Research* (pp. 159-181). New York: Routledge.

Field, A., Miles, J., & Field, Z. (2013). *Discovering statistics using R*. Los Angeles: Sage.

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. doi: 10.18637/jss.v082.i13

Linck, J. A., & Cunnings, I. (2015). The utility and application of mixed-effects models in second language research. *Language Learning, 65*(S1), 185-207. doi:10.1111/lang.12117

Murakami, A. (2016). Modeling systematicity and individuality in nonlinear second language development: The case of English grammatical morphemes. *Language Learning, 66*(4), 834-871. doi:10.1111/lang.12166

Nagle, C. (2017a). Individual developmental trajectories in the L2 acquisition of Spanish spirantization. *Journal of Second Language Pronunciation, 3*(2), 219-242.

Nagle, C. (2017b). A longitudinal study of voice onset time development in L2 Spanish stops. *Applied Linguistics*. doi:10.1093/applin/amx011

Offerman, H. M., & Olson, D. J. (2016). Visual feedback and second language segmental production: The generalizability of pronunciation gains. *System, 59*, 45-60. doi:10.1016/j.system.2016.03.003

R Core Team. (2019). R: A language and environment for statistical computing (Version 3.5.3). Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org/

Singer, J. D., & Willett, J. B. (2003). *Applied longitudinal data analysis*. New York: Oxford University Press.

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. New York: Springer.