

Understanding mixed effects models through data simulation

Lisa M. DeBruine¹ & Dale J. Barr¹

¹ Institute of Neuroscience and Psychology, University of Glasgow

In press at *Advances in Methods and Practices in Psychological Science*

Abstract


Experimental designs that sample both subjects and stimuli from a larger population need to account for random effects of both subjects and stimuli using mixed effects models. However, much of this research is analyzed using ANOVA on aggregated responses because researchers are not confident specifying and interpreting mixed effects models. The tutorial will explain how to simulate data with random effects structure and analyse the data using linear mixed effects regression (with the lme4 R package), with a focus on interpreting the output in light of the simulated parameters. Data simulation can not only enhance understanding of how these models work, but also enables researchers to perform power calculations for complex designs. All materials associated with this article can be accessed at <https://osf.io/3cz2e/>.


Keywords: simulation, mixed effects models, power, lme4, R

Word count: 5572

Background

In this article, we walk through the simulation and analysis of multilevel data with crossed random effects of subjects and stimuli. The article's target audience is researchers who work with experimental designs that sample subjects and stimuli, such as is the case for a large amount of experimental research in face perception, psycholinguistics, and social cognition. Simulation is useful not only for helping understand how models work, but also for estimating power when planning a study or performing a sensitivity analysis. The tutorial assumes basic familiarity with R programming.

Lisa DeBruine  <https://orcid.org/0000-0002-7523-5539>

Dale J. Barr  <https://orcid.org/0000-0002-1121-4608>

Correspondence concerning this article should be addressed to Lisa M. DeBruine, 62 Hillhead Street, Glasgow, G12 8QB. E-mail: lisa.debruine@glasgow.ac.uk

Generalizing to a population of encounters

Many research questions in psychology and neuroscience are questions about certain types of *events*: What happens when people encounter particular types of stimuli? For example: Do people recognize abstract words faster than concrete words? What impressions do people form about a target person’s personality based on their vocal qualities? Can people categorize emotional expressions more quickly on the faces of social ingroup members than on the faces of outgroup members? How do brains respond to threatening versus non-threatening stimuli? In all of these situations, researchers would like to be able to make general statements about phenomena that go beyond the particular participants and particular stimuli that they happen to have chosen for the specific study. Traditionally, people speak of such designs as having *crossed random factors* of participants and stimuli, and think of the goal of inference as being simultaneous generalization to both populations. However, it may be more intuitive to construe the goal as generalizing to a single population of *events* called *encounters*: we want to say something general about what happens when our two types of sampling units meet—when a typical subject encounters (and responds to) a typical stimulus (Barr, 2018).

Most analyses using conventional statistical techniques, such as analysis of variance and t-test, commit the fallacy of treating stimuli as fixed rather than random. For example, imagine a sample of participants are rating the trustworthiness of a sample of faces, with the goal being to determine whether the faces of people born on even numbered days look more trustworthy than those born on odd-numbered days. Obviously, they don’t. At the extreme, imagine we only sample a single face from each category, but have 100 people rate each face. If our analysis treats the sample of faces as *fixed*, or a perfect representation of the larger population of faces on Earth, we will be almost guaranteed a significant difference in one direction or the other. We will have sufficient power to detect even tiny differences in apparent trustworthiness, so this result will be highly replicable with large samples of raters. As you increase the number of faces in the sample, the problem gets better (the sample means are more likely to approximate the population means), but if you increase the number of raters (and thus power to detect small differences in the sample means), it gets worse again.

The problem, and the solutions to the problem, have been known in psycholinguistics for over 50 years (Clark, 1973; Coleman, 1964), and most psycholinguistic journals require authors to demonstrate generalizability of findings over stimuli as well as over subjects. Even so, the quasi- F statistics for ANOVA (F' and min- F') that Clark proposed as a solution were widely recognized as unreasonably conservative (Forster & Dickinson, 1976), and until fairly recently, most psycholinguists performed separate by-subjects (F_1) and by-items analyses (F_2), declaring an effect “significant” only if it was significant for both analyses. This $F_1 \times F_2$ approach has been widely used, despite the fact that Clark had already shown it to be invalid, since both F statistics have higher than nominal false positives in the presence of a null effect— F_1 due to unmodeled stimulus variance, and F_2 due to unmodeled subject variance.

Recently, psycholinguists have adopted linear mixed-effects modeling as the standard

for statistical analysis, given numerous advantages over ANOVA, including the ability to simultaneously model subject and stimulus variation, to gracefully deal with missing data or unbalanced designs, and to accommodate arbitrary types of continuous and categorical predictors or response variables (Baayen, Davidson, & Bates, 2008; Locker, Hoffman, & Bovaird, 2007). This development has been facilitated by the `lme4` package for R (Bates, Mächler, Bolker, & Walker, 2015), which provides powerful functionality for model specification and estimation. With an appropriately specified model, mixed-effects models yield major improvements in power over quasi- F approaches and avoid the increased false positive rate associated with separate F_1 and F_2 (Barr et al., 2013).

Despite mixed-effects modeling becoming the *de facto* standard for analysis in psycholinguistics, the approach has yet to take hold in other areas where stimuli are routinely sampled, even in spite of repeated calls for improved analyses in social psychology (Judd, Westfall, & Kenny, 2012) and neuroimaging (Bedny, Aguirre, & Thompson-Schill, 2007; Westfall, Nichols, & Yarkoni, 2016). One of the likely reasons for the limited uptake outside of psycholinguistics is because mixed-effects models expose the analyst to a level of statistical and technical complexity far beyond most researchers' training. While some of this complexity is specific to mixed-effects modeling, some of it is simply hidden away from users of traditional techniques by GUIs and function defaults. The novice mixed modeler is suddenly confronted with the need to make decisions about how to specify categorical predictors, which random effects to include or exclude, which of the statistics in the voluminous output to attend to, and whether and how to re-configure the optimizer function when a convergence error or singularity warning appears.

We are optimistic that the increasing adoption of the mixed-effects approach will improve the generalizability and thus reproducibility of studies in psychology and related fields. Models that account for subjects and stimuli (or other factors) as non-essential, exchangeable features of an experiment will better characterize the uncertainty in the resulting estimates and thus, improve the generality of inferences we draw from them (Yarkoni, 2019). That said, we empathize with the frustration — and sometimes, exasperation — expressed by many novices when they attempt to grapple with these models in their research. A profitable way to build understanding and confidence is through data simulation. If you can create datasets by sampling from a population where you know the ground truth about the population parameters you are interested in (e.g., means and standard deviations of each group), you can check how often and under what circumstances a statistical model will give you the correct answer. Knowing the ground truth also allows you to experiment with various modeling choices and observe their impact on a model's performance.

Box 1. Glossary of terms

crossed random factors	Refers to a design with multiple random factors, such as subjects and items, the levels of which are crossed (e.g., each subject encounters each stimulus)
data-generating process (DGP)	The mathematical model capturing assumptions about the processes giving rise to the data
fixed effect	An effect whose value is constant across realizations of the experiment
random effect	An effect whose value varies across potential realizations of the experiment (e.g., due to sampling)
random intercept	A random effect capturing the deviation of a sampling unit (subject or item) from the model intercept
random slope	A random effect capturing the deviation of a sampling unit (subject or item) from the model slope
variance components	Parameters describing the distribution of random effects in the population

Data, materials, and online resources

The code to reproduce the analyses reported in this article and appendices with extended examples are publicly available via the Open Science Framework and can be accessed at <https://osf.io/3cz2e>. This code is supplemented by web apps that perform data simulation without requiring knowledge of R code. These apps allow you to change parameters and inspect the results of LMEM and ANOVA analyses, as well as calculate power and false positive rates for these analyses.

Simulating data with crossed random factors

Data simulation can play a powerful role in statistics education, enhancing understanding of the use and interpretation of statistical models and the assumptions behind them. The data simulation approach to learning about statistical models differs from the standard approach in most statistics textbooks, where the learner is presented with a step-by-step analysis of a sample of data from some population of interest. Such exercises usually culminate in inferences about characteristics of the population of interest from model estimates. Although this reflects the typical uncertain situation of the analyst, the learner cannot fully appreciate the performance of the model without knowing the ground truth. In a data simulation approach, the learner starts out knowing the ground truth about the population and writes code to simulate the process of taking and analyzing samples from that population. Giving learners knowledge of the underlying population parameters as well as the ability to explore how population parameters are reflected in model estimates can yield powerful insight into the appropriate specification of models and the interpretation of statistical output.

Data simulation also has a wide variety of scientific uses, one of which is to estimate properties of statistical models in situations where algorithms for computing those properties are unknown or can only be applied with difficulty. For instance, Forster and Dickinson (1976) used Monte Carlo simulation to explore the behavior of the quasi- F statistics for ANOVA (F' and min- F') under various conditions. In a Monte Carlo simulation, the long run properties of a process are estimated by generating and analyzing many simulated datasets—usually, thousands or tens of thousands of them.

One of the most important applications of Monte Carlo simulation is in the estimation of power for complex models. The notion of power appears most frequently in the context of study planning, where a power analysis is used to determine the target N for a study. Power is the probability that a specified statistical test will generate a significant result for a sample of data of a specified size taken from a population with a specified effect. If you can characterize the population parameters, you can repeatedly simulate and analyze data from this population. The proportion of times that this procedure produces a significant result provides an estimate of the power of your test given your assumed sample size and effect size. You can adjust any parameters in this simulation in order to estimate other parameters. Instead of estimating power, you can perform a *sensitivity analysis* by varying the effect size while holding the sample size and desired power constant—for instance, to determine the minimum effect size that your analysis can detect with 80% power and an N of 200 per group. You can also set the population effect size to zero and calculate the proportion of significant results to check if your analysis procedure inflates *false positives*.

For most traditional statistical procedures such as t -test or ANOVA, there are analytical procedures for estimating power. Westfall, Kenny, and Judd (2014) present analytic power curves for simple mixed effects designs such as the one described in this tutorial (with a corresponding app at <https://jakewestfall.shinyapps.io/crossedpower>). But even where analytical solutions exist, simulation can still be useful to estimate power or false positive rates, because real psychological data nearly always deviates from the statistical assumptions behind traditional procedures. For instance, most statistical procedures used in psychology assume a continuous and unbounded dependent variable, but it is often the case that researchers use discrete (e.g., Likert) response scales. When assumptions are not met, power simulations can provide a more reliable estimate than analytical procedures.

To give an overview of the simulation task, we will simulate data from a design with crossed random factors of subjects and stimuli, fit a model to the simulated data, and then see whether the resulting sample estimates are similar to the population values we specified when simulating the data. In this hypothetical study, subjects classify the emotional expressions of faces as quickly as possible, and we use their response time as the primary dependent variable. Let's imagine that the faces are of two types: either from the subject's ingroup or from an outgroup. For simplicity, we further assume that each face appears only once in the stimulus set. The key question is whether there is any difference in classification speed across the type of face.

Required software

The simulation will be presented in the R programming language (R Core Team, 2018). To run the code, you will need to have some add-on packages available.

```
# load required packages
library("lme4")      # model specification / estimation
library("lmerTest")  # provides p-values in the output
library("tidyverse") # data wrangling and visualisation
```

Because the code uses random number generation, if you want to reproduce the exact results below you will need to set the random number seed at the top of your script and ensure you are using R version 3.6.0 or higher. If you change the seed or are using a lower version of R, your exact numbers will differ, but the procedure will still produce a valid simulation.

```
# ensure this script returns the same results on each run
set.seed(8675309)
```

Establishing the data-generating parameters

The first thing to do is to set up the parameters that govern the process we assume to give rise to the data, the *data-generating process* or DGP. Let's start by defining the sample size: In this hypothetical study, each of 100 subjects will respond to all 50 stimulus items (25 ingroup and 25 outgroup), for a total of 5000 observations.

Specify the data structure. We want the resulting data to be in long format, with the structure shown in Table 1, where each row is a single observation for each trial. The variable `subj_id` runs from 1 to 100 and indexes the subject number; `item_id` runs from 1 to 50 and indexes the item number; `category` is whether the face is ingroup or outgroup, with items 1-25 always ingroup and items 26-50 always outgroup; and `RT` is the participant's response time for that trial. Note that a trial is uniquely identified by the combination of the `subj_id` and `item_id` labels.

Table 1
The target data structure.

row	subj_id	item_id	category	RT
1	1	1	ingroup	750.2
2	1	2	ingroup	836.1
...
49	1	49	outgroup	811.9
50	1	50	outgroup	801.8
51	2	1	ingroup	806.7
52	2	2	ingroup	805.9
...
5000	100	50	outgroup	859.9

Note that for independent variables in designs where subjects and stimuli are crossed, you can't think of factors as being solely "within" or "between" because we have two sampling units; you must ask not only whether independent variables are within- or between- subjects, but also whether they are within- or between- stimulus items. Recall that a within-subjects factor is one where each and every subject receives all of the levels, and a between-subjects factor is one where each subject receives only one of the levels. Likewise, a within-items factor is one for which each stimulus receives all of the levels. For our current example, the ingroup/outgroup factor (`category`) is within subjects but between items, given that each stimulus item is either ingroup or outgroup.

Table 2

Variables in the data-generating model and associated R code.

model	code	description
RT_{si}	<code>RT</code>	reaction time for subject s to item i
X_i	<code>X_i</code>	condition for item i (-.5 = ingroup, .5 = outgroup)
β_0	<code>beta_0</code>	intercept; grand mean RT
β_1	<code>beta_1</code>	slope; mean effect of ingroup/outgroup
τ_0	<code>tau_0</code>	standard deviation of by-subject random intercepts
τ_1	<code>tau_1</code>	standard deviation of by-subject random slopes
ρ	<code>rho</code>	correlation between by-subject random intercepts and slopes
ω_0	<code>omega_0</code>	standard deviation of by-item random intercepts
σ	<code>sigma</code>	standard deviation of residuals
T_{0s}	<code>T_0s</code>	random intercept for subject s
T_{1s}	<code>T_1s</code>	random slope for subject s
O_{0i}	<code>O_0i</code>	random intercept for item i
e_{si}	<code>e_si</code>	residual for the trial involving subject s and item i

Specify the fixed effect parameters. Now that we have an appropriate structure for our simulated dataset, we need to generate the RT values. For this, we need to establish an underlying statistical model. In this and the next section, we will build up a statistical model step by step, defining variables in the code as we go along that reflect our choices for parameters. For convenience, Table 2 lists all of the variables in the statistical model and their associated variable names in the code.

Let us start with a basic model and build up from there. We want a model of RT for subject s and item i that looks something like:

$$RT_{si} = \beta_0 + \beta_1 X_i + e_{si}. \quad (1)$$

According to the formula, response RT_{si} for subject s and item i is defined as sum of an intercept term β_0 , which in this example is the grand mean reaction time for the population of stimuli, plus β_1 , the mean RT difference between ingroup and outgroup stimuli, plus random noise e_{si} . To make β_0 equal the grand mean and β_1 equal the mean outgroup minus

the mean ingroup RT, we will code the item category variable X_i as -.5 for the ingroup category and +.5 for the outgroup category.

In the model formula, we use Greek letters (β_0, β_1) to represent population parameters that are being directly estimated by the model. In contrast, Roman letters represent the remaining variables: observed variables whose values are determined by sampling (e.g., RT_{si} , T_{0s} , e_{si}) or fixed by the experiment design (X_i).

Although this model is incomplete, we can go ahead and choose parameters for β_0 and β_1 . For this example, we set a grand mean of 800 ms and a mean difference of 50 ms. You will need to use disciplinary expertise and/or pilot data to choose these parameters; by the end of this tutorial you will understand how to extract those parameters from an analysis.

```
# set fixed effect parameters
beta_0 <- 800 # intercept; i.e., the grand mean
beta_1 <- 50  # slope; i.e, effect of category
```

The parameters β_0 and β_1 are *fixed effects*: they characterize the population of events in which a typical subject encounters a typical stimulus. Thus, we set the mean RT for a “typical” subject encountering a “typical” stimulus to 800 ms, and assume that responses are typically 50 ms slower for outgroup than ingroup faces.

Specify the random effect parameters. This model is completely unrealistic, however, because it doesn’t allow for any individual differences among subjects or stimuli. Subjects are not identical in their response characteristics: some will be faster than average, and some slower. We can characterize the difference from the grand mean for each subject s in terms of a *random effect* T_{0s} , where the first subscript, 0, indicates that the deflection goes with the intercept term, β_0 . This *random intercept* term captures the value that must be added or subtracted to the intercept for subject s , which in this case corresponds to how much slower or faster this subject is relative to the average reaction time of 800 ms. Just as it is unrealistic to expect the same intercept for every subject, it is also unrealistic to assume this for stimuli; it will be easier to categorize emotional expressions for some faces than others, and we can incorporate this assumption by including by-item random intercepts O_{0i} , with the subscript 0 reminding us that it is a deflection from the β_0 term, and the i indexing each of the 50 stimulus items (faces). Each face is assigned a unique *random intercept* that characterizes how much slower or faster responses to this particular face tend to be relative to the average reaction time of 800 ms. Adding these terms to our model yields:

$$RT_{si} = \beta_0 + T_{0s} + O_{0i} + \beta_1 X_i + e_{si}. \quad (2)$$

Now, whatever values of T_{0s} and O_{0i} we end up with in our sampled dataset will depend on the luck of the draw, i.e., on which subjects and stimuli we happened to have sampled from their respective populations. Unlike fixed effects, we assume these values will differ across different realizations of the experiment where subjects and/or stimuli are different. In practice, we often re-use the same stimuli across many studies, but we still need to treat the stimuli as sampled if we want to be able to generalise our findings to the whole

population of stimuli.¹

It is an important conceptual feature of mixed-effects models that they do not directly estimate the individual random effects (T_{0s} and O_{0i} values), but rather, they estimate the random effects parameters that characterize the distributions from which these effects are drawn.² It is this feature that enables generalization beyond the particular subjects and stimuli in the experiment. We assume that each T_{0s} comes from a normal distribution with a mean of zero and unknown standard deviation, τ_0 (`tau_0` in the code). The mean of zero reflects the assumption that each random effect is a deflection from the grand mean. Similarly, for the by-item random intercepts, we assume the O_{0i} values to be sampled from a normal distribution also with a mean of zero and with an unknown standard deviation, ω_0 (`omega_0`). In our simulation, we will set the by-subject random intercept SD to 100, and the by-item random intercept SD to 80.

```
# set random effect parameters
tau_0    <- 100 # by-subject random intercept sd
omega_0  <- 80  # by-item random intercept sd
```

There is still a deficiency in our data-generating model related to β_1 , the fixed effect of category. Currently our model assumes that each and every subject is exactly 80 ms faster to categorize emotions on ingroup faces than on outgroup faces. Clearly, this assumption is totally unrealistic; some participants will be more sensitive to ingroup/outgroup differences than others. We can capture this in an analogous way to which we captured variation in the intercept, namely by including by-subject *random slopes* T_{1s} .

$$RT_{si} = \beta_0 + T_{0s} + O_{0i} + (\beta_1 + T_{1s}) X_i + e_{si} \quad (3)$$

The random slope T_{1s} is an estimate of how much faster or slower subject s is in categorizing ingroup/outgroup faces than the population mean effect β_1 , which we already set to 50 ms. Given how we coded the X_i variable, the mean effect for subject s is given by the $\beta_1 + T_{1s}$ term. So, a participant who is 90 ms faster on average to categorize ingroup than outgroup faces would have a random slope T_{1s} of 40 ($\beta_1 + T_{1s} = 50 + 40 = 90$). As we did for the random intercepts, we assume that the T_{1s} effects are drawn from a normal distribution, with a mean of zero and standard deviation of τ_1 (`tau_1` in the code). For this example, we assume the standard deviation is 40 ms.

But note that we are sampling *two* random effects for each subject s , a random intercept T_{0s} and a random slope T_{1s} . It is possible for these values to be positively or negatively correlated, in which case we should not sample them independently. For instance, perhaps people who are faster than average overall (negative random intercept) also show a smaller than average effect of the ingroup/outgroup manipulation (negative random slope)

¹To ensure a finding is generalizable, when attempting a replication it is useful to sample new items, just as one would typically sample new participants.

²You will sometimes see the assumption of random effects drawn from a normal distribution notated mathematically as, e.g.: $O_{0i} \sim N(0, \omega_0^2)$, which you can read as “the random intercept O_{0i} for each subject i is drawn from a normal distribution with mean of zero and variance of ω_0^2 .”

due to allocating less attention to the task. We can capture this by allowing for a small positive correlation between the two factors, `rho`, which we assign to be 0.2.

Finally, we need to characterize the trial-level noise in the study (e_{si}) in terms of its standard deviation. Here we simply assign this parameter value `sigma` to be twice the size of the by-subject random intercept SD.

```
# set more random effect and error parameters
tau_1 <- 40 # by-subject random slope sd
rho <- .2 # correlation between intercept and slope
sigma <- 200 # residual (error) sd
```

To summarize, we established a reasonable statistical model underlying the data having the form:

$$RT_{si} = \beta_0 + T_{0s} + O_{0i} + (\beta_1 + O_{1s}) X_i + e_{si} \quad (4)$$

The response time for subject s on item i , RT_{si} , is decomposed into a population grand mean β_0 , a by-subject random intercept T_{0s} , a by-item random intercept O_{0i} , a fixed slope β_1 , a by-subject random slope T_{1s} , and a trial-level residual e_{si} . Our data-generating process is fully determined by seven population parameters, all denoted by Greek letters: β_0 , β_1 , τ_0 , τ_1 , ρ , ω_0 , and σ (see Table 2). In the next section we will apply this data-generating process to simulate the sampling of subjects, items, and trials (encounters).

Simulating the sampling process

Let's first define parameters related to the number of observations. In this example, we will simulate data from 100 subjects responding to 25 ingroup faces and 25 outgroup faces. There are no between-subject factors, so we can set `n_subj` to 100. We set `n_ingroup` and `n_outgroup` to the number of stimulus items in each condition.

```
# set number of subjects and items
n_subj <- 100 # number of subjects
n_ingroup <- 25 # number of ingroup stimuli
n_outgroup <- 25 # number of outgroup stimuli
```

Simulate the sampling of stimulus items. We need to create a table listing each item i , which category it is in, and its random effect O_{0i} .

```
# simulate a sample of items
# total number of items = n_ingroup + n_outgroup
items <- data.frame(
  item_id = seq_len(n_ingroup + n_outgroup),
  category = rep(c("ingroup", "outgroup"), c(n_ingroup, n_outgroup)),
  O_0i = rnorm(n = n_ingroup + n_outgroup, mean = 0, sd = omega_0)
)
```

For the first variable in the dataset, `item_id`, we have used `seq_len()` to assign a unique integer to each of the 50 stimulus faces; these IDs function like names. The `category` variable designates whether the face is ingroup or outgroup, with the first 25 items being ingroup and the last 25 being outgroup. Finally, we sample the values of O_{0i} from a normal distribution using the `rnorm()` function, with a mean of 0 and SD of ω_0 .

Let us introduce a numeric predictor to represent what category each stimulus item i appears in (i.e., for the X_i in our model). Since we predict that responses to ingroup faces will be faster than outgroup faces, we set ingroup to -0.5 and outgroup to +0.5. We will later multiply this *effect coded* factor by the fixed effect of category (`beta_1 = 50`) to simulate data where the ingroup faces are on average -25 ms different from the grand mean, while the outgroup faces are on average 25 ms different from the grand mean. After adding this variable, the resulting table `items` should look like Table 3, although the specific values you obtain for O_{0i} may differ, depending on whether you set the random seed.

```
# effect-code category
items$X_i <- recode(items$category, "ingroup" = -0.5, "outgroup" = +0.5)
```

In R, most regression procedures can handle two-level factors like `category` as predictor variables. By default, the procedure will create a new numeric predictor that codes one level of the factor as zero and the other as one. Why not just use the defaults? The short explanation is that the default of 0, 1 coding is not well-suited to the kinds of factorial experimental designs often found in psychology and related fields. For the current example, using the default coding for the X predictor would change the interpretation of β_0 : instead of the grand mean, it would reflect the mean for the group coded as zero. One could change the default, but we feel it is better to be explicit in the code about what values are being used. See <https://talklab.psy.gla.ac.uk/tvw/catpred> for further discussion; see also the R mixed modeling package `afex` (Singmann, Bolker, Westfall, & Aust, 2019), which provides better defaults for specifying categorical predictors in ANOVA-style designs.

Table 3
The resulting sample of items.

item_id	category	O_Oi	X_i
1	ingroup	-79.7	-0.5
2	ingroup	57.7	-0.5
3	ingroup	-49.4	-0.5
4	ingroup	162.4	-0.5
5	ingroup	85.2	-0.5
6	ingroup	79	-0.5
...
44	outgroup	54.7	0.5
45	outgroup	-20.2	0.5
46	outgroup	-12.1	0.5
47	outgroup	-70	0.5
48	outgroup	-158.2	0.5
49	outgroup	19	0.5
50	outgroup	2.9	0.5

Simulate the sampling of subjects. Now we will simulate the sampling of individual subjects, resulting in a table listing each subject and their two correlated random effects. This will be slightly more complicated than what we just did, because we cannot simply sample the T_{0s} values from a univariate distribution using `rnorm()` independently from the T_{1s} values. Instead, we must sample $\langle T_{0s}, T_{1s} \rangle$ pairs—one pair for each subject—from a bivariate normal distribution. To do this, we will use the `mvrnorm()` function, a multivariate version of `rnorm()` from the MASS package that comes pre-installed with R. We specify the three parameters describing this distribution—two variances and a correlation—by entering them into a 2x2 *variance-covariance* matrix using the `matrix()` function, and then passing this matrix to `mvrnorm()` using the `Sigma` argument. This requires converting the standard deviations into variances (by squaring them) and calculating the covariance, which is the product of the correlation and two standard deviations, i.e., $\rho \times \tau_0 \times \tau_1$.

We only need this one function from MASS, so we can call it directly using the `package::function()` syntax instead of loading the library (specifically, `MASS::mvrnorm()` instead of `library(MASS)`).³ The resulting table `subjects` should have the structure shown in Table 4.

```
# simulate a sample of subjects

# calculate random intercept / random slope covariance
covar <- rho * tau_0 * tau_1

# put values into variance-covariance matrix
```

³Note that MASS package has a function we don't need named `select()`, but loading it using `library()` would overwrite the function of the same name from the `dplyr` package that we often do need, so in general it is a good idea to avoid loading MASS.

```

cov_mx <- matrix(
  c(tau_0^2, covar,
    covar, tau_1^2),
  nrow = 2, byrow = TRUE)

# generate the by-subject random effects
subject_rfx <- MASS::mvrnorm(n = n_subj,
                             mu = c(T_0s = 0, T_1s = 0),
                             Sigma = cov_mx)

# combine with subject IDs
subjects <- data.frame(subj_id = seq_len(n_subj),
                       subject_rfx)

```

Table 4

The resulting sample of subjects.

subj_id	T_0s	T_1s
1	-14.7	11.1
2	-8.4	-36.7
3	87.7	-47.5
4	209.3	62.9
5	-23.6	21.5
6	90.1	56.7
...
94	99.5	-31
95	44.3	69.3
96	12.2	37.1
97	-121.9	42.3
98	-49.9	-41.1
99	-134.5	16.6
100	-30.2	37.5

322 An alternative way to sample from a bivariate distribution would be to use the
 323 function `rnorm_multi()` from the `faux` package (DeBruine, 2020), which generates a table
 324 of `n` simulated values from a multivariate normal distribution by specifying the means (`mu`)
 325 and standard deviations (`sd`) of each variable, plus the correlations (`r`), which can be either
 326 a single value (applied to all pairs), a correlation matrix, or a vector of the values in the
 327 upper right triangle of the correlation matrix.

```

# simulate a sample of subjects

# sample from a multivariate random distribution
subjects <- faux::rnorm_multi(

```

```

n = n_subj,
mu = 0, # means for random effects are always 0
sd = c(tau_0, tau_1), # set SDs
r = rho, # set correlation, see ?faux::rnorm_multi
varnames = c("T_0s", "T_1s")
)

# add subject IDs
subjects$subj_id <- seq_len(n_subj)

```

328 **Simulate trials (encounters).** Since all subjects respond to all items, we can set
 329 up a table of trials by making a table with every possible combination of the rows in the
 330 subject and item tables using the tidyverse function `crossing()`. Each trial has random
 331 error associated with it, reflecting fluctuations in trial-by-trial performance due to unknown
 332 factors; we simulate this by sampling values from a normal distribution with a mean of 0
 333 and SD of `sigma`. The resulting table should correspond to Table 5.

```

# cross subject and item IDs; add an error term
# nrow(.) is the number of rows in the table
trials <- crossing(subjects, items) %>%
  mutate(e_si = rnorm(nrow(.), mean = 0, sd = sigma)) %>%
  select(subj_id, item_id, category, X_i, everything())

```

Table 5
The resulting table of trials (encounters).

subj_id	item_id	category	X_i	T_0s	T_1s	O_0i	e_si
1	1	ingroup	-0.50	-14.65	11.13	-79.73	-66.54
1	2	ingroup	-0.50	-14.65	11.13	57.75	-34.74
1	3	ingroup	-0.50	-14.65	11.13	-49.38	-37.49
1	4	ingroup	-0.50	-14.65	11.13	162.35	231.26
1	5	ingroup	-0.50	-14.65	11.13	85.23	-187.64
1	6	ingroup	-0.50	-14.65	11.13	78.98	104.81
...
100	44	outgroup	0.50	-30.15	37.52	54.73	-3.38
100	45	outgroup	0.50	-30.15	37.52	-20.16	18.47
100	46	outgroup	0.50	-30.15	37.52	-12.08	87.92
100	47	outgroup	0.50	-30.15	37.52	-69.99	25.47
100	48	outgroup	0.50	-30.15	37.52	-158.15	91.23
100	49	outgroup	0.50	-30.15	37.52	19.01	78.14
100	50	outgroup	0.50	-30.15	37.52	2.89	-34.31

334 **Calculate the response values.** With this resulting table, in combination with
 335 the constants `beta_0` and `beta_1`, we have the full set of values that we need to compute
 336 the response variable `RT` according to the linear model we defined above:

$$RT_{si} = \beta_0 + T_{0s} + O_{0i} + (\beta_1 + O_{1s}) X_i + e_{si} \quad (5)$$

337 Thus, we calculate the response variable RT by adding together:

- 338 • the grand intercept (`beta_0`),
- 339 • each subject-specific random intercept (`T_0s`),
- 340 • each item-specific random intercept (`O_0i`),
- 341 • each sum of the category effect (`beta_1`) and the random slope (`T_1s`), multiplied by
- 342 the numeric predictor (`X_i`), and
- 343 • each residual error (`e_si`).

344 After this we will use `dplyr::select()` to keep the columns we need. Note that the resulting
 345 table (Table 6) has the structure that we set as our goal at the start of this exercise, with
 346 the additional column `X_i`, which we will keep to use in the estimation process, described in
 347 the next section.

```
# calculate the response variable
dat_sim <- trials %>%
  mutate(RT = beta_0 + T_0s + O_0i + (beta_1 + T_1s) * X_i + e_si) %>%
  select(subj_id, item_id, category, X_i, RT)
```

Table 6
The final simulated dataset.

subj_id	item_id	category	X_i	RT
1	1	ingroup	-0.5	609
1	2	ingroup	-0.5	778
1	3	ingroup	-0.5	668
1	4	ingroup	-0.5	1148
1	5	ingroup	-0.5	652
1	6	ingroup	-0.5	939
...
100	44	outgroup	0.5	865
100	45	outgroup	0.5	812
100	46	outgroup	0.5	889
100	47	outgroup	0.5	769
100	48	outgroup	0.5	747
100	49	outgroup	0.5	911
100	50	outgroup	0.5	782

348 **Data simulation function.** To make it easier to try out different parameters or
 349 to generate many datasets for the purpose of power analysis, you can put all of the code
 350 above into a custom function. Set up the function to take all of the parameters we set
 351 above as arguments. We'll set the defaults to the values we used, but you can choose your
 352 own defaults. The code below is just all of the code above, condensed a bit. It returns one
 353 dataset with the parameters you specified.

```

# set up the custom data simulation function
my_sim_data <- function(
  n_subj      = 100,  # number of subjects
  n_ingroup   = 25,   # number of ingroup stimuli
  n_outgroup  = 25,   # number of outgroup stimuli
  beta_0      = 800,  # grand mean
  beta_1      = 50,   # effect of category
  omega_0     = 80,   # by-item random intercept sd
  tau_0       = 100,  # by-subject random intercept sd
  tau_1       = 40,   # by-subject random slope sd
  rho         = 0.2,  # correlation between intercept and slope
  sigma       = 200) { # residual (standard deviation)

  items <- data.frame(
    item_id = seq_len(n_ingroup + n_outgroup),
    category = rep(c("ingroup", "outgroup"), c(n_ingroup, n_outgroup)),
    X_i = rep(c(-0.5, 0.5), c(n_ingroup, n_outgroup)),
    O_0i = rnorm(n = n_ingroup + n_outgroup, mean = 0, sd = omega_0))

  # variance-covariance matrix
  cov_mx <- matrix(
    c(tau_0^2,          rho * tau_0 * tau_1,
      rho * tau_0 * tau_1, tau_1^2),
    nrow = 2, byrow = TRUE)

  subjects <- data.frame(subj_id = seq_len(n_subj),
    MASS::mvrnorm(n = n_subj,
      mu = c(T_0s = 0, T_1s = 0),
      Sigma = cov_mx))

  crossing(subjects, items) %>%
    mutate(e_si = rnorm(nrow(.), mean = 0, sd = sigma),
      RT = beta_0 + T_0s + O_0i + (beta_1 + T_1s) * X_i + e_si) %>%
    select(subj_id, item_id, category, X_i, RT)
}

```

354 Now you can generate a dataset with the default parameters using `my_sim_data()` or,
 355 for example, a dataset with 500 subjects and no effect of category using `my_sim_data(n_subj`
 356 `= 500, beta_1 = 0)`.

Analyzing the simulated data

Setting up the formula

Now we're ready to analyse our simulated data. The first argument to `lmer()` is a model formula that defines the structure of the linear model. The formula for our design maps onto how we calculated the response above.

```
RT ~ 1 + X_i + (1 | item_id) + (1 + X_i | subj_id)
```

- `RT` is the response;
- `1` corresponds to the grand intercept (`beta_0`);
- `X_i` is the predictor for the ingroup/outgroup manipulation for item `i`;
- `(1 | item_id)` specifies a by-subject random intercept (`0_0i`);
- `(1 + X_i | subj_id)` specifies a subject-specific random intercept (`T_0s`) plus the subject-specific random slope of category (`T_1s`).

The error term (`e_si`) is automatically included in all models, so is left implicit. The “fixed” part of the formula, `RT ~ 1 + X_i`, establishes the $RT_{si} + \beta_0 + \beta_1 X_i + e_{si}$ part of our linear model. Every model has an intercept (β_0) term and residual term (e_{si}) by default, so you could alternatively leave the `1` out and just write `RT ~ X_i`.

The terms in parentheses with the “pipe” separator (`|`) define the random effects structure. For each of these bracketed terms, the left-hand side of the pipe names the effects you wish to allow to vary and the right hand side names the variable identifying the levels of the random factor over which the terms vary (e.g., subjects or items). The first term, `(1 | item_id)` allows the intercept (`1`) to vary over the random factor of items (`item_id`). This is an instruction to estimate the parameter underlying the `0_0i` values, namely `omega_0`. The second term, `(1 + X_i | subj_id)`, allows both the intercept and the effect of category (coded by `X_i`) to vary over the random factor of subjects (`subj_id`). It is an instruction to estimate the three parameters that underlie the `T_0s` and `T_1s` values, namely `tau_0`, `tau_1`, and `rho`.

Interpreting the lmer summary

The other arguments to the `lme4` function are the name of the data frame where the values are found (`dat_sim`). Because we loaded in `lmerTest` after `lme4`, the p -values are derived using the Satterthwaite approximation, for which the default estimation technique in `lmer()`—restricted likelihood estimation (`REML = TRUE`)—is the most appropriate (Luke, 2017). Use the `summary()` function to view the results.

```
# fit a linear mixed-effects model to data
mod_sim <- lmer(RT ~ 1 + X_i + (1 | item_id) + (1 + X_i | subj_id),
               data = dat_sim)

summary(mod_sim, corr = FALSE)
```

```

389 ## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
390 ## lmerModLmerTest]
391 ## Formula: RT ~ 1 + X_i + (1 | item_id) + (1 + X_i | subj_id)
392 ## Data: dat_sim
393 ##
394 ## REML criterion at convergence: 67740.7
395 ##
396 ## Scaled residuals:
397 ##      Min       1Q   Median       3Q      Max
398 ## -3.7370 -0.6732  0.0075  0.6708  3.5524
399 ##
400 ## Random effects:
401 ##   Groups      Name      Variance Std.Dev. Corr
402 ##   subj_id (Intercept)  8416      91.74
403 ##           X_i         3298      57.43  0.12
404 ##   item_id (Intercept)  4072      63.81
405 ##   Residual         41283     203.18
406 ## Number of obs: 5000, groups:  subj_id, 100; item_id, 50
407 ##
408 ## Fixed effects:
409 ##              Estimate Std. Error    df t value Pr(>|t|)
410 ## (Intercept)   807.72      13.19 119.05  61.258  <2e-16 ***
411 ## X_i           39.47      19.79  56.30   1.994   0.051 .
412 ## ---
413 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Let's break down the output step-by-step and try to find estimates of the seven parameters we used to generate the data: `beta_0`, `beta_1`, `tau_0`, `tau_1`, `rho`, `omega_0` and `sigma`. If you analyze existing data with a mixed effects model, you can use these estimates to help you set reasonable values for random effects in your own simulations.

After providing general information about the model fit, the output is divided into a **Random effects** and a **Fixed effects** section. The fixed effects section should be familiar from other types of linear models.

```

421 ## Fixed effects:
422 ##              Estimate Std. Error    df t value Pr(>|t|)
423 ## (Intercept)   807.72      13.19 119.05  61.258  <2e-16 ***
424 ## X_i           39.47      19.79  56.30   1.994   0.051 .

```

The **Estimate** column gives us parameter estimates for the fixed effects in the model, i.e., β_0 and β_1 , which are estimated at about 807.72 and 39.47. The next columns give us the standard errors, estimated degrees of freedom (using the Satterthwaite approach), t value, and finally, p value.

The **Random effects** section is specific to mixed-effects models, and will be less familiar to the reader.

```

431 ## Random effects:
432 ##   Groups   Name          Variance Std.Dev. Corr
433 ##   subj_id (Intercept)  8416      91.74
434 ##           X_i          3298      57.43  0.12
435 ##   item_id (Intercept)  4072      63.81
436 ##   Residual              41283     203.18

```

437 These are the estimates for the *variance components* in the model. Note that there are no
 438 p-values associated with these effects. If you wish to determine whether a random effect
 439 is significant, you need to run the model with and without the random effect term and
 440 compare the log-likelihoods of the models. But usually the random effects parameters are
 441 not the target of statistical tests because they reflect the existence of individual variation,
 442 which can be trivially assumed to exist for any manipulation that has a non-zero effect.

443 To avoid confusion, it is best to think of the information in the **Random effects**
 444 section as coming from three separate tables divided up by the values in the **Groups** column.
 445 The first subtable, where the value of **Groups** is **subj_id**, gives the estimates for the random
 446 effects parameters defining the by-subject random effects.

```

447 ##   Groups   Name          Variance Std.Dev. Corr
448 ##   subj_id (Intercept)  8416      91.74
449 ##           X_i          3298      57.43  0.12

```

450 We have estimates for the variance of the intercept and slope (**X_i**) in the **Variance** column,
 451 which is just the square of the standard deviation in the **Std.Dev.** column. We obtain
 452 estimates for **tau_0** and **tau_1** of 91.74 and 57.43 respectively. The **Corr.** column gives us
 453 the estimated correlation between the by-subject random intercepts and slopes, estimated
 454 here as 0.12.

455 The second subtable gives us the by-item random effect parameter estimates of which
 456 there is only one, 63.81, corresponding to **omega_0**. Again, the **Variance** column is just this
 457 value squared.

```

458 ##   Groups   Name          Variance Std.Dev. Corr
459 ##   item_id (Intercept)  4072      63.81

```

460 The last subtable gives us the estimate of the residual term, 203.18.

```

461 ##   Groups   Name          Variance Std.Dev. Corr
462 ##   Residual              41283     203.18

```

463 We have found all seven parameter estimates in the output. Let's compare them to
 464 the actual parameter values that we specified (Table 7).

Table 7

The simulation parameters compared to the model estimations.

variable	explanation	simulated value	estimated by model
beta_0	intercept (grand mean)	800.0	807.72
beta_1	fixed effect of category	50.0	39.47
tau_0	by-subject random intercept SD	100.0	91.74
rho	correlation between intercept and slope	0.2	0.12
tau_1	by-subject random slope SD	40.0	57.43
omega_0	by-item random intercept SD	80.0	63.81
sigma	residual (error) SD	200.0	203.18

465 You can also use `broom.mixed::tidy()` to output fixed and/or random effects in a
 466 tidy table (Table 8). This is especially useful when you need to combine the output from
 467 hundreds of simulations to calculate power. The column “effect” specifies whether a row
 468 is a fixed effect (“fixed”) or a random effect parameter (“ran_pars”). The column “group”
 469 specifies the group by the random factor column name (or “Residual”) for random effect
 470 parameters. The column “term” refers to the predictor term for fixed effects, and also the
 471 parameter for random effects; for example, “sd__X_i” refers to the standard deviation of the
 472 random slope for X_i, while “cor__(Intercept).X_i” refers to rho, the correlation between
 473 the random intercept and slope for X_i. We added the column “sim” to the standard output
 474 of `broom.mixed::tidy()` so you can compare the simulated parameters we set above to
 475 the estimated parameters from this simulated dataset, which are in the column “estimate”.
 476 The last four columns give the standard error, *t*-statistic, estimated degrees of freedom, and
 477 *p*-value for the fixed effects.

```
# get a tidy table of results
broom.mixed::tidy(mod_sim) %>%
  mutate(sim = c(beta_0, beta_1, tau_0, rho, tau_1, omega_0, sigma)) %>%
  select(1:3, 9, 4:8)
```

Table 8

The output of the tidy function from broom.mixed.

effect	group	term	sim	estimate	std.error	statistic	df	p.value
fixed	NA	(Intercept)	800.0	807.72	13.2	61.3	119.1	0.000
fixed	NA	X_i	50.0	39.47	19.8	2.0	56.3	0.051
ran_pars	subj_id	sd__(Intercept)	100.0	91.74	NA	NA	NA	NA
ran_pars	subj_id	cor__(Intercept).X_i	0.2	0.12	NA	NA	NA	NA
ran_pars	subj_id	sd__X_i	40.0	57.43	NA	NA	NA	NA
ran_pars	item_id	sd__(Intercept)	80.0	63.81	NA	NA	NA	NA
ran_pars	Residual	sd__Observation	200.0	203.18	NA	NA	NA	NA

Setting Parameters

Now that you see where each parameter we used to generate the data appears in the analysis output, you can use the analysis of pilot data to get estimates for these parameters for further simulation. For example, if you have pilot data from 10 participants on this task, you can analyse their data using the same code as above, and estimate values for `beta_0`, `beta_1`, `tau_0`, `tau_1`, `rho`, `omega_0`, and `sigma` for use in a power calculation or a sensitivity analysis (see appendix 1C). If you lack any pilot data to work with, you can start with the general rule of thumb setting the residual variance to about twice the size of the by-subject or by-item variance components (see supplementary materials from Barr et al., 2013 at <https://talklab.psy.gla.ac.uk/singen/realdata.html> for results from an informal convenience sample).

Calculate Power

Data simulation is a particularly flexible approach for estimating power when planning a study. The basic idea of a power simulation is to choose parameter values with which to generate a large number of datasets, fit models to each dataset, and then calculate the proportion of models that reject the null hypothesis. This proportion is an estimate of your power for those particular parameter values. To estimate this value accurately using Monte Carlo simulation, you need to generate and analyze a large number (typically, hundreds or thousands) of datasets.

First we create a function `single_run()` that generates data using `my_sim_data()`, analyzes it, and returns a table with the parameter estimates. This would correspond to a single “run” for our power simulation.

```
# simulate, analyze, and return a table of parameter estimates
single_run <- function(...) {
  # ... is a shortcut that forwards any arguments to
  # my_sim_data(), the function created above
  dat_sim <- my_sim_data(...)
  mod_sim <- lmer(RT ~ X_i + (1 | item_id) + (1 + X_i | subj_id),
                 dat_sim)

  broom.mixed::tidy(mod_sim)
}
```

```
# run one model with default parameters
single_run()
```

You can also change parameters. For example, what would happen if you increase the number of items to 50 in each group and decrease the effect of category to 20 ms? Example results of a single run with these parameters are shown in Table 9.

```
# run one model with new parameters
single_run(ningroup = 50, n_outgroup = 50, beta_1 = 20)
```

Table 9

The output of single_run() with 50 items per group and a category effect of 20 ms.

effect	group	term	estimate	std.error	statistic	df	p.value
fixed	NA	(Intercept)	832.38	12.6	66.1	174.0	0.000
fixed	NA	X_i	24.95	16.0	1.6	114.9	0.121
ran_pars	item_id	sd__(Intercept)	73.66	NA	NA	NA	NA
ran_pars	subj_id	sd__(Intercept)	100.27	NA	NA	NA	NA
ran_pars	subj_id	cor__(Intercept).X_i	0.00	NA	NA	NA	NA
ran_pars	subj_id	sd__X_i	47.57	NA	NA	NA	NA
ran_pars	Residual	sd__Observation	199.15	NA	NA	NA	NA

You can use the `purrr::map_df` function to run the simulation repeatedly and save the results to a data table. This will take a while, so test it first using just a few runs (`n_runs`) to debug and check the output. Once you are satisfied it is working properly, we suggest that you use at least a thousand runs to obtain stable estimates. It will save you a lot of time if you save the full results to disk so that you don't have to re-run it each time you execute your script; you can just comment out this code and load from the saved data for the rest of your script in the future.

```
# run simulations and save to a file
n_runs <- 100 # use at least 1000 to get stable estimates
sims <- purrr::map_df(1:n_runs, ~ single_run())
write_csv(sims, "sims.csv")
```

Note that some runs may throw warnings about non-convergence or messages about **boundary (singular) fit**. The message about the singular fit can usually be ignored—see `?isSingular` for information about what this means. Non-convergence will be relatively rare with simulated data provided the sample is not unreasonably small relative to the number of estimated parameters; as long as there are not too many of these non-convergence warnings relative to the number of runs, you can probably ignore them because they won't affect the overall estimates. Alternatively, you can re-write your function to trap the warning (see appendix 1C); for more information on trapping errors and warnings, see the chapter on “Exceptions and Debugging” in the Advanced R textbook (Wickham, 2019).

Once our simulations are complete let's read the data back in and have a look at the estimates for our fixed effects.

```
# read saved simulation data
sims <- read_csv("sims.csv", col_types = cols(
  # makes sure plots display in this order
  group = col_factor(ordered = TRUE),
  term = col_factor(ordered = TRUE)
))

sims %>%
  filter(effect == "fixed") %>%
  select(term, estimate, p.value)
```

```

521 ## # A tibble: 200 x 3
522 ##   term      estimate p.value
523 ##   <ord>      <dbl>   <dbl>
524 ## 1 (Intercept)  813.    2.93e-86
525 ## 2 X_i          83.4    3.53e- 4
526 ## 3 (Intercept)  799.    1.25e-82
527 ## 4 X_i          57.9    1.58e- 2
528 ## 5 (Intercept)  782.    6.17e-88
529 ## 6 X_i          63.9    4.54e- 3
530 ## 7 (Intercept)  812.    1.97e-83
531 ## 8 X_i          45.7    4.78e- 2
532 ## 9 (Intercept)  824.    8.69e-77
533 ## 10 X_i         1.78    9.45e- 1
534 ## # ... with 190 more rows

```

Each row in the table is an estimate of a fixed-effects parameter and associated p -value from a single run of the Monte Carlo simulation ((Intercept) = β_0 and X_i = β_1). We need to calculate the proportion of runs that are significant. To start, we compute `p.value < alpha`, where `alpha` is your false positive rate (e.g., .05). This will yield a logical vector of TRUE wherever the effect was significant and FALSE where it was non-significant. Because TRUE is represented internally as a 1 and FALSE as a 0, you can take the mean of this logical vector and it will yield the proportion of significant runs.

```

# calculate mean estimates and power for specified alpha
alpha <- 0.05

sims %>%
  filter(effect == "fixed") %>%
  group_by(term) %>%
  summarize(
    mean_estimate = mean(estimate),
    mean_se = mean(std.error),
    power = mean(p.value < alpha),
    .groups = "drop"
  )

```

Table 10
Power calculation for fixed effects.

term	Mean Estimate	Mean Std. Error	Power
(Intercept)	797.5	15.4	1.00
X_i	48.4	23.4	0.54

The results of our power analysis appear in Table 10. The attained power of 0.54 in the second row is the estimated probability of finding a significant effect of category (as

represented by X_i) given our starting parameters. In other words, it is the probability of rejecting the null hypothesis for β_1 , which is the coefficient associated with X_i in the model ($H_0 : \beta_1 = 0$). If we wanted to see how power changes with different parameter settings, we would need to re-run the simulations with different values passed to `single_run()`.

Conclusion

Mixed-effects modeling is a powerful technique for analyzing data from complex designs. The technique is close to ideal for analyzing data with crossed random factors of subjects and stimuli: it gracefully and simultaneously accounts for subject and item variance within a single analysis, and outperforms traditional techniques in terms of type I error and power (Barr et al., 2013). However, this additional power comes at the price of technical complexity. Through this article, we have attempted to make mixed-effects models more approachable using data simulation.

We considered only a simple, one-factor design. However, the general principles are the same for higher-order designs. For instance, consider a 2x2 design, with factors A and B both within subjects, but A within items and B between items. For such a design, you would have four instead of two by-subject random effects: the intercept, main effect of A , main effect of B , and the AB interaction. You would also need to specify correlations between each of these effects. You would also have two by-item random effects: one for the intercept and one for A . Our supplemental online materials (<https://osf.io/3cz2e/>) include such an extension of the example in this paper with `category` as a within-subject and between-item factor and adding `expression` as a within-subject and within-item factor. For further guidance and discussion on how to specify the random effects structure in complex designs, see Barr (2013).

Here we only considered a design with a normally distributed response variable. However, generalised linear mixed effect models allow for response variables with different distributions, such as binomial. Our supplemental online materials (<https://osf.io/3cz2e/>) illustrate the differences in simulation required for the study design in this paper with a binomial accuracy score (correct/incorrect) as the response variable.

We also have not said much in this tutorial about estimation issues, such as what to do when the fitting procedure fails to converge. Further guidance on this point can be found in Barr et al. (2013) as well as in the help materials in the `lme4` package (`?lme4::convergence`). We have also assumed that the random effects specification for the `lmer()` function should be based on the study design. However, we note that others have argued in favor of data-driven approaches for random effects specification (Matuschek, Kliegl, Vasishth, Baayen, & Bates, 2017).

In this tutorial, we have introduced the main concepts needed to get started with mixed effects models. Through data simulation of your own study designs, you can develop your understanding and perform power calculations to guide your sample size plans.

Acknowledgements

Author Contributions. DJB drafted the substantive explanation and LDB drafted the tutorial. Both authors revised the draft and approved the final submitted version of the manuscript. LDB created the app http://shiny.psy.gla.ac.uk/lmem_sim/.

Declaration of Conflicting Interests. The author(s) declared that there were no conflicts of interest with respect to the authorship or the publication of this article.

Funding. LMD is supported by European Research Council grant #647910.

Research Software. This tutorial uses the following open-source research software: R Core Team (2018); Bates et al. (2015); Kuznetsova, Brockhoff, and Christensen (2017); Bolker and Robinson (2019); Singmann et al. (2019); Wickham (2017); DeBruine (2020); Aust and Barth (2018).

References

- Aust, F., & Barth, M. (2018). *papaja: Create APA manuscripts with R Markdown*. Retrieved from <https://github.com/crsh/papaja>
- 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. (2013). Random effects structure for testing interactions in linear mixed-effects models. *Frontiers in Psychology*, 4, 328.
- Barr, D. J. (2018). Generalizing over encounters: Statistical and theoretical considerations. In S.-A. Rueschemeyer & M. G. Gaskell (Eds.), *Oxford handbook of psycholinguistics*. Oxford University Press.
- 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.
- Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1), 1–48. <https://doi.org/10.18637/jss.v067.i01>
- Bedny, M., Aguirre, G. K., & Thompson-Schill, S. L. (2007). Item analysis in functional magnetic resonance imaging. *Neuroimage*, 35(3), 1093–1102.
- Bolker, B., & Robinson, D. (2019). *Broom.mixed: Tidying methods for mixed models*. Retrieved from <https://CRAN.R-project.org/package=broom.mixed>
- Clark, H. 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.
- Coleman, E. B. (1964). Generalizing to a language population. *Psychological Reports*, 14, 219–226.
- DeBruine, L. (2020). *Faux: Simulation for factorial designs*. Zenodo. <https://doi.org/10.5281/zenodo.2669586>
- Forster, K., & Dickinson, R. (1976). More on the language-as-fixed-effect fallacy: Monte carlo estimates of error rates for F_1 , F_2 , F' , and $\min F'$. *Journal of Verbal Learning and Verbal Behavior*, 15, 135–142.
- 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, 54.
- 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. <https://doi.org/10.18637/jss.v082.i13>

- 629 Locker, L., Hoffman, L., & Bovaird, J. (2007). On the use of multilevel modeling as an
630 alternative to items analysis in psycholinguistic research. *Behavior Research Methods*,
631 *39*, 723–730.
- 632 Luke, S. G. (2017). Evaluating significance in linear mixed-effects models in r. *Behavior*
633 *Research Methods*, *49*(4), 1494–1502.
- 634 Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing type
635 I error and power in linear mixed models. *Journal of Memory and Language*, *94*,
636 305–315.
- 637 R Core Team. (2018). *R: A language and environment for statistical computing*. Vienna,
638 Austria: R Foundation for Statistical Computing. Retrieved from [https://www.R-](https://www.R-project.org/)
639 [project.org/](https://www.R-project.org/)
- 640 Singmann, H., Bolker, B., Westfall, J., & Aust, F. (2019). *Afex: Analysis of factorial*
641 *experiments*. Retrieved from <https://CRAN.R-project.org/package=afex>
- 642 Westfall, J., Kenny, D. A., & Judd, C. M. (2014). Statistical power and optimal design in
643 experiments in which samples of participants respond to samples of stimuli. *Journal*
644 *of Experimental Psychology: General*, *143*(5), 2020–2045.
- 645 Westfall, J., Nichols, T. E., & Yarkoni, T. (2016). Fixing the stimulus-as-fixed-effect fallacy
646 in task fMRI. *Wellcome Open Research*, *1*.
- 647 Wickham, H. (2017). *Tidyverse: Easily install and load the 'tidyverse'*. Retrieved from
648 <https://CRAN.R-project.org/package=tidyverse>
- 649 Wickham, H. (2019). *Advanced R*. CRC press. Retrieved from <http://adv-r.had.co.nz/>
- 650 Yarkoni, T. (2019). The generalizability crisis. Retrieved from <https://psyarxiv.com/jqw35>