Understanding mixed effects models through simulating data

Lisa M. DeBruine[1] & Dale J. Barr[1]

[1] Institute of Neuroscience and Psychology, University of Glasgow

Author Note

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

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). The focus will be on interpreting the LMER output in light of the simulated parameters, using this method for power calculations, and comparing the results to by-items and by-subjects ANOVA.

Keywords: simulation, mixed effect models, power, lme4

Word count: X

Understanding mixed effects models through simulating data

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 in-group members than on the faces of out-group 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 have discussed the problem as one of simultaneous generalization to both populations. However, it may be more intuitive to think of the problem as wanting to generalize to a single population of events: in particular, to a population of *encounters* between the units from the sampled populations (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. The problem, and the solutions to the problem, have been known in psycholinguistics for over 50 years (Coleman, 1964,Clark (1973)), and most psycholinguistic journals require authors to demonstrate generality 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. The $F_1 \times F_2$

approach was 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 the 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, and Bovaird (2007)). This development has been facilitated by the `lme4` package for R (D. 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, Levy, Scheepers, & Tily, 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, and 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 optimisic that the increasing adoption of the mixed-effects approach will improve the generality and thus reproducibility of studies in psychology and related fields, but empathize with the frustration — and sometimes, exasperation — expressed by many novices when they attempt to grapple with these models in their research. Much of the uncertainty and unease around mixed-effects models comes from using them in situations where the ground truth is unknown. A profitable way to improve understanding and user confidence is through data simulation. Knowing the ground truth allows the user to experiment with various modeling choices and observe their impact on a model's performance.

Simulating data with crossed random factors

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 try to recover the parameter values we put in from the output. 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 intrinsic types: either from the subject's in-group or from an out-group. For simplicity, we further assume that each face appears only once in the stimulus set and expresses only one of two possible emotions (e.g., happiness or anger). 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. Any packages you are missing can be installed using R's `install.packages()` function, except for the development package `faux` (DeBruine, 2019) which, at the time of writing, must be installed from the development repository on github.

```r
library("lme4")        # model specification / estimation
library("afex")        # anova and deriving p-values from lmer
library("broom.mixed") # extracting data from model fits
library("faux")        # data simulation
# NOTE: to install the 'faux' package, use:
# devtools::install_github("debruine/faux")
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.

```r
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. In this hypothetical study, each of 100 subjects will respond to all 50 stimulus items (25 in-group and 25 out-group), for a total of 5000 observations.

Specify the data structure.

We want the resulting data to be in long format, with the structure shown below, where each row is a single observation for each trial. The variables `subj_id` run from `S001` to `S100` and index the subject number; `item_id` runs from `I01` to `I50` and indexes the item number; `condition` says whether the face is in-group or out-group, 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

Table 1

The target data structure.

row	subj_id	item_id	condition	RT
1	S001	I01	ingroup	750.2
2	S001	I02	ingroup	836.1
...	...	...	...	...
49	S001	I49	outgroup	811.9
50	S001	I50	outgroup	801.8
51	S002	I01	ingroup	806.7
52	S002	I02	ingroup	805.9
...	...	...	...	...
5000	S100	I50	outgroup	859.9

`item_id` labels.

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 factors is one where each subject receives only one of the levels. Likewise, a within-subjects factor is one where each stimulus appears across all of the levels of the independent factors. For our current example, this is clearly not the case, given that each stimulus item is either in-group or out-group.

Let's first define parameters related to the number of observations.

```
nsubj  <- 100 # number of subjects
nitem  <- c(ingroup = 25, outgroup = 25)  # number of items
```

Specify the fixed effects.

Getting an appropriately structured dataset is the easy part. The difficult part is randomly generating the RT values. For this, we need to establish an underlying statistical model. 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:

$$RT_{si} = \beta_0 + \beta_1 X_i + e_{si}$$

In other words, it is the 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 in-group and out-group stimuli, plus random noise e_{si}. To make β_0 equal the grand mean and β_1 the mean out-group minus the mean in-group RT, we will code the `condition` variable as -.5 for the in-group condition and +.5 for the out-group condition.

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 80 ms. You will need to use disciplinary expertise and/or pilot data to choose these parameters. For power calculations, consider setting effects to the smallest effect size of interest (Lakens, Scheel, & Isager, 2018).

```
b0 <- 800 # intercept; i.e., the grand mean
b1 <-  80 # slope; i.e, effect of condition
```

The parameters β_0 and β_1 are *fixed effects*: they characterize properties of the population of encounters between subjects and stimuli. Thus, we assign the mean RT for a "typical" subject encountering a "typical" stimulus to 800 ms, and that responses are typically 80 ms slower for outgroup than ingroup faces.

Specify the random effects.

This model is completely unrealistic, however, because it doesn't allow for any individual differences among subjects or stimuli. Not all subjects are typical: 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* S_{0s}, where the first subscript, 0, indicates that the deflection goes with the intercept term, β_0. In other words, we assume each subject to have a unique *random intercept*. Likewise, it is unrealistic to assume that it is equally easy to categorize emotional expressions across all faces in the dataset; some will be easier than others. We incorporate this assumption by including by-item random intercepts I_{0i}, with the subscript 0 reminding us that it is a deflection from the β_0 term, and the i indicating a unique deflection for each of the 50 faces. Adding these terms to our model yields:

$$RT_{si} = \beta_0 + S_{0s} + I_{0i} + \beta_1 X_i + e_{si}.$$

Now, the actual values for S_{0s} and I_{0i} in our sampled dataset will depend on the luck of the draw, i.e., on which subject and which stimuli we happened to have sampled from their respective populations. So to capture that these are *random* rather than *fixed* factors, we will set parameters that capture the standard deviation among the random effects and then use these to "sample" from the populations. Below we assign the by-subject offsets a standard deviation of 100 ms (`sri_sd`), and the by-item offsets have a standard deviation of 80 ms (`iri_sd`). We will discuss below how you can estimate these parameters for your own designs.

```
sri_sd <- 100 # by-subject random intercept sd
iri_sd <-  80 # by-item random intercept sd
```

There is still a deficiency in our data-generating model related to β_1, the fixed effect of condition. 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* S_{1s}.

$$RT_{si} = \beta_0 + S_{0s} + I_{0i} + (\beta_1 + S_{1s})\, X_i + e_{si}.$$

A participant who is, on average, 90 ms faster for ingroup faces would have a random slope $S_{1s} = 10$ ($90 = 80 + 10$); a participant who goes against the grain and is, for whatever reason, on average 15 ms faster for *outgroup* faces would have a random slope of $S_{1s} = -95$ (-15 = 80 - 95). As we did for the random intercepts, we characterize the random slopes in terms of their standard deviation `srs_sd`, which we assign to be 40 ms.

But note that we are sampling *two* random effects for each subject s, a random intercept S_{0s} and a random slope S_{1s}. It is possible for these values to be 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 of the ingroup/outgroup manipulation (negative random slope) due to allocating less attention to the task. We can capture this by allowing for a small correlation between the two factors, `scor`, which we assign to be .2.

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

```
srs_sd <-  40 # by-subject random slope sd
scor   <-  .2 # correlation between intercept and slope
err_sd <- 200 # residual (error) sd
```

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

$$RT_{si} = \beta_0 + S_{0s} + I_{0i} + (\beta_1 + S_{1s})\, X_i + e_{si}$$

where 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 S_{0s}, a by-item random intercept I_{0i}, a fixed slope β_1, a by-subject random slope S_{1s}, and a trial-level residual e_{si}. Our data-generating process is fully determined by seven parameters: two fixed effects (intercept and slope), four variance parameters governing the random effects (defined in the code as `sri_sd`, `srs_sd`, `scor`, and `iri_sd`), and one parameter governing the trial level variance (`err_sd`).

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.

Simulate the sampling of stimulus items.

We need to create a table listing each item, which condition it is in, and simulated values for its random effects. We can do this with the code below, setting item ID to the numbers 1 through the total number of items, condition for the first 25 items to “ingroup” and the next 25 faces to “outgroup”, and sampling 50 numbers from a normal distribution with a mean of 0 and a standard deviation of `iri_sd`.

```r
items <- data.frame(
  item_id = 1:sum(nitem),
  condition = rep(c("ingroup", "outgroup"), nitem),
  I0i = rnorm(sum(nitem), 0, iri_sd)
)
```

The function `faux::sim_design()` is a more flexible way to generate data with specified parameters. This function will create a dataset with any number of between and/or within factors, `n` items per between-cell, and the specified means (`mu`), standard deviations (`sd`) and correlations (r). By default, it plots a schematic of the design you specified. See the vignette (DeBruine, 2019) for more details.

Condition is a between-items factor, so we need to include it in the `between` argument. Set `n = nitem` to specify the number of items per condition. Set `sd = iri_sd` to set the standard deviation for the by-item random effects. Set `dv = "I0i"` to give the random effect column that name. Set `id = "item_id"`; we'll use this later to join this information to the table of trials.

```r
items <- faux::sim_design(
  between = list(condition = c("ingroup", "outgroup")),
  n = nitem,
  sd = iri_sd,
  dv = "I0i",
  id = "item_id"
)
```

We will also introduce a numerical predictor to represent what condition 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

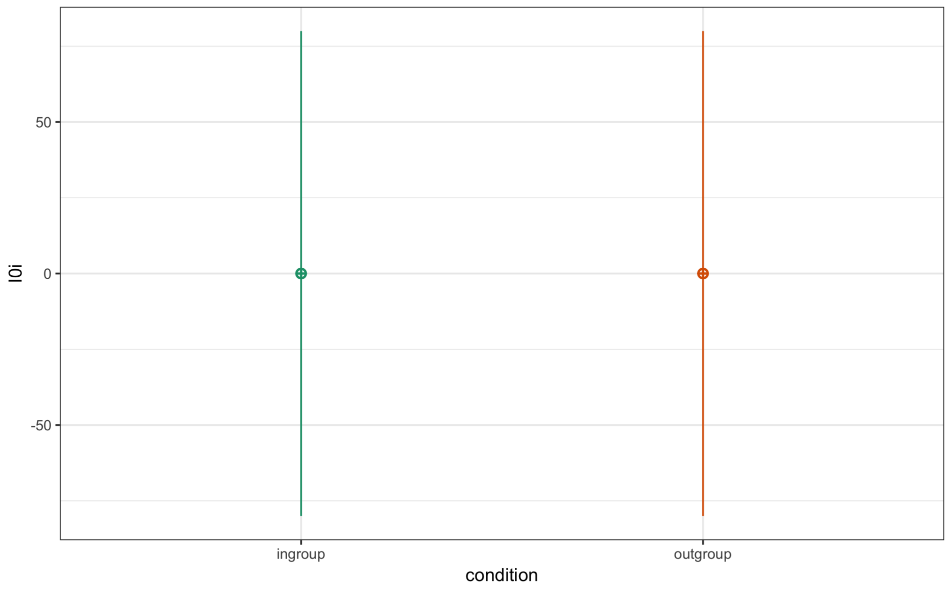

Figure 1. The specified distribution of random effects for ingroup and outgroup faces.

will later multiply this *effect coded* factor by the fixed effect of condition (`b1` = 80) to simulate data where the ingroup faces are on average -40 ms different from the grand mean, while the outgroup faces are 40 ms different from the grand mean.

```
# effect code condition
items$cond <- recode(items$condition, "ingroup" = -0.5, "outgroup" = +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. We will again use `faux::sim_design()` for this task.

Set the `within` argument in `sim_design()` to a list with one factor (`effect`) that has two levels: `S0s` and `S1s`. If you set a factor's levels as a named vector, the names (`S0s` and `S1s`) become the column names in the data table and the values are used in plots created by faux.

Table 2

The resulting table of item parameters.

item_id	condition	I0i	cond
S01	ingroup	59.56	-0.50
S02	ingroup	-107.73	-0.50
S03	ingroup	26.41	-0.50
S04	ingroup	-1.02	-0.50
S05	ingroup	-37.09	-0.50
S06	ingroup	16.40	-0.50

Set `n = nsubj` to specify the number of subjects. There are two random effects to specify standard deviation for, so set `sd` using a named vector and set their correlation with r = scor. Set `dv = "value"`; this will only be used in faux plots. Set `id = "subj_id"`; we'll use this later to join this information to the table of trials.

```r
subjects <- faux::sim_design(
  within = list(effect = c(S0s = "By-subject random intercepts",
                           S1s = "By-subject random slopes")),
  n = nsubj,
  sd = c(sri_sd = sri_sd, srs_sd = srs_sd),
  r = scor,
  dv = "value",
  id = "subj_id"
)
```

Let's have a look at the resulting table.

Simulate trials (encounters).

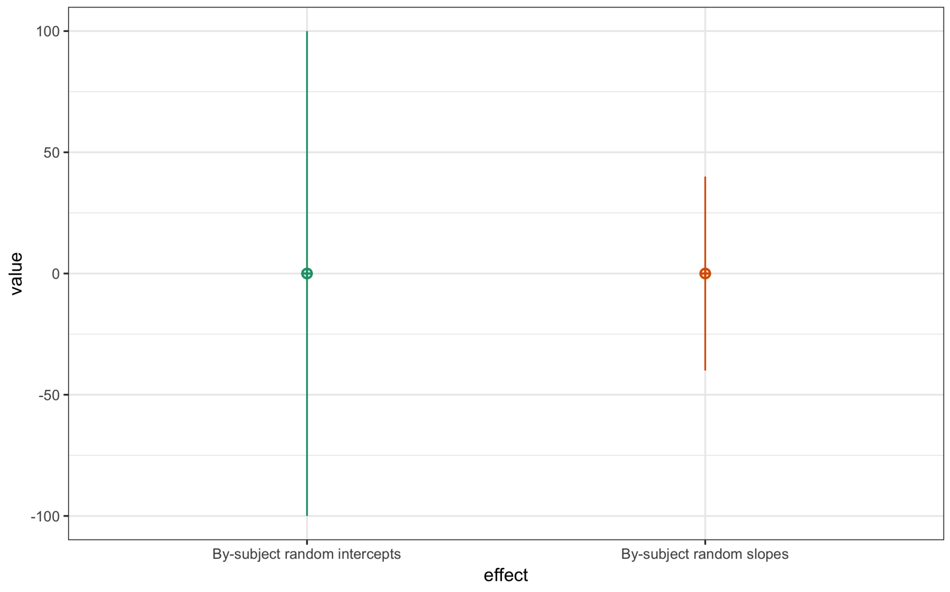

Figure 2. The specified distribution of random effects for subjects

Since all subjects respond to all items, we can set up a table of trials by crossing the subject IDs with the item IDs. Each trial has random error associated; we simulate this from a normal distribution with a mean of 0 and SD of `err_sd`.

```
# crossing() is from the tidyr package;
# see ?tidyr::crossing for details
trials <- crossing(subj_id = subjects$subj_id,
                   item_id = items$item_id) %>%
  mutate(err = rnorm(nrow(.), mean = 0, sd = err_sd))
```

Calculate the response values. Now that we have a table of all trials, we can join the information in this table to the information in our `subjects` and `items` tables. We join them together using `dplyr::inner_join()`.

```
joined <- trials %>%
  inner_join(subjects, "subj_id") %>%
  inner_join(items, "item_id")
```

Table 3

The resulting table of subject parameters.

subj_id	S0s	S1s
S001	-98.77	-49.49
S002	37.33	23.77
S003	-127.42	39.56
S004	-56.37	10.13
S005	-39.73	-8.05
S006	43.70	37.66

Table 4

The resulting table of trials.

subj_id	item_id	err
S001	S01	307.99
S001	S02	85.33
S001	S03	-205.25
S001	S04	-138.06
S001	S05	-190.78
S001	S06	-351.12

Table 5

The resulting table of trials joined to subject and item parameters.

subj_id	item_id	err	S0s	S1s	condition	I0i	cond
S001	S01	307.99	-98.77	-49.49	ingroup	59.56	-0.50
S001	S02	85.33	-98.77	-49.49	ingroup	-107.73	-0.50
S001	S03	-205.25	-98.77	-49.49	ingroup	26.41	-0.50
S001	S04	-138.06	-98.77	-49.49	ingroup	-1.02	-0.50
S001	S05	-190.78	-98.77	-49.49	ingroup	-37.09	-0.50
S001	S06	-351.12	-98.77	-49.49	ingroup	16.40	-0.50

Note how this resulting table contains the full decomposition of effects that we need to compute the response according to the linear model we defined above:

$$RT_{si} = \beta_0 + S_{0s} + I_{0i} + (\beta_1 + S_{1s})\,X_i + e_{si}.$$

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

- the grand intercept (`b0`),
- each subject-specific random intercept (`S0s`),
- each item-specific random intercept (`I0i`),
- each sum of the condition effect (`b1`) and the random slope (`S1s`), multiplied by the numerical predictor (`cond`), and
- each residual error (`err`).

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

Table 6

The final simulated dataset.

subj_id	item_id	condition	cond	RT
S001	S01	ingroup	-0.50	1,053.53
S001	S02	ingroup	-0.50	663.58
S001	S03	ingroup	-0.50	507.14
S001	S04	ingroup	-0.50	546.90
S001	S05	ingroup	-0.50	458.10
S001	S06	ingroup	-0.50	351.25

```r
dat_sim <- joined %>%
  mutate(RT = b0 + S0s + I0i + (b1 + S1s) * cond + err) %>%
  select(subj_id, item_id, condition, cond, RT)
```

Analyse Data

Now we're ready to analyse our simulated data. The formula for `lmer()` maps onto how we calculated the response above.

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

- `RT` is the response
- `1` is the grand intercept (`b0`),
- `cond` is the effect of condition (`b1 * cond`),
- `1` in `(1 | item_id)` is the item-specific random intercept (`iri`),
- `1` in `(1 + cond | subj_id)` is the subject-specific random intercept (`sri`),
- `cond` in `(1 + cond | subj_id)` is the subject-specific random slope of condition (`S1s`

```
    * cond)
```

The `lmer()` function takes this formula as its first argument, then the data table. Set `REML = FALSE` to choose the method for estimating variance components (`REML = TRUE` is better when you have fairly unequal cell sizes).

```
mod_sim <- lmer(RT ~ 1 + cond + (1 | item_id) + (1 + cond | subj_id),
                data = dat_sim, REML = TRUE)
```

Use the `summary()` function to view the results. Notice where the parameters you set at the beginning show up in the results. If you analyze existing data with a mixed effect model, you can use these estimates to help you set reasonable values for random effects in your own simulations.

```
summary(mod_sim, corr = FALSE)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: RT ~ 1 + cond + (1 | item_id) + (1 + cond | subj_id)
##    Data: dat_sim
##
## REML criterion at convergence: 67691.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.7638 -0.6737 -0.0046  0.6776  3.6428
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subj_id  (Intercept) 10396    101.96
```

Table 7

The simulation parameters versus the model estimations.

variable	explanation	simulated value	estimated by model
b0	intercept (grand mean)	800.00	816.05
b1	fixed effect of condition	80.00	82.42
sri_sd	by-subject random intercept SD	100.00	101.96
srs_sd	by-subject random slope SD	40.00	49.21
scor	cor between intercept and slope	0.20	0.16
iri_sd	by-item random intercept SD	80.00	68.72
err_sd	residual (error) SD	200.00	201.91

```
##            cond         2421     49.21   0.16
##  item_id  (Intercept)  4723     68.72
##  Residual             40769    201.91
## Number of obs: 5000, groups:  subj_id, 100; item_id, 50
##
## Fixed effects:
##              Estimate Std. Error     df t value Pr(>|t|)
## (Intercept)   816.05      14.37 123.21  56.778  < 2e-16 ***
## cond           82.42      20.85  53.33   3.954 0.000228 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can also use `broom.mixed::tidy()` to output fixed and/or random effects in a tidy table. This is especially useful when you need to combine the output from hundreds of simulations to calculate power. The code below adds a column with the simulated parameters we set above so you can compare them to the estimated parameters from this

Table 8

The output of the tidy function from broom.mixed.

effect	group	term	sim.params	estimate	std.error	statistic	df	p.
fixed	NA	(Intercept)	800.000	816.045	14.373	56.778	123.214	0.
fixed	NA	cond	80.000	82.425	20.848	3.954	53.332	0.
ran_pars	subj_id	sd___(Intercept)	100.000	101.963	NA	NA	NA	N
ran_pars	subj_id	sd___cond	40.000	49.205	NA	NA	NA	N
ran_pars	subj_id	cor___(Intercept).cond	0.200	0.159	NA	NA	NA	N
ran_pars	item_id	sd___(Intercept)	80.000	68.723	NA	NA	NA	N
ran_pars	Residual	sd___Observation	200.000	201.913	NA	NA	NA	N

simulated dataset.

```
broom.mixed::tidy(mod_sim) %>%
  mutate(sim.params = c(b0, b1, sri_sd, srs_sd, scor, iri_sd, err_sd)) %>%
  select(1:3, 9, 4:8) %>%
  apa_table(digits = 3, caption="The output of the tidy function from broom.mixed.")
```

Calculate Power

You can set up a function that takes all of the parameters we set above as arguments. We'll set the to default to the values we used, but you can choose your own defaults. The code below is just all of the data simulation code above, condensed a bit. It returns one dataset with the parameters you specified.

```
my_sim_data <- function(nsubj  = 100, # number of subjects
                        nitem  = c(ingroup = 25, outgroup = 25),  # number of items
                        b0     = 800, # grand mean
                        b1     =  80, # effect of condition
                        iri_sd =  80, # by-item random intercept sd
                        sri_sd = 100, # by-subject random intercept sd
                        srs_sd =  40, # by-subject random slope sd
                        scor   = 0.2, # correlation between intercept and slope
                        err_sd = 200  # residual (standard deviation)
                        ) {
# simulate items
items <- faux::sim_design(
  between = list(condition = c("ingroup", "outgroup")),
  n = nitem,
  sd = iri_sd,
  dv = "I0i",
  id = "item_id",
  plot = FALSE
)

# effect code condition
items$cond <- recode(items$condition, "ingroup" = -0.5, "outgroup" = 0.5)

# simulate subjects
subjects <- faux::sim_design(
  within = list(effect = c(S0s = "By-subject random intercepts",
                           S1s = "By-subject random slopes")),
  n = nsubj,
  sd = c(sri = sri_sd, srs = srs_sd),
    r = scor,
    dv = "value",
    id = "subj_id",
    plot = FALSE
  )

  # simulate trials
  dat_sim <- crossing(subj_id = subjects$subj_id,
                      item_id = items$item_id) %>%
    mutate(err = rnorm(nrow(.), mean = 0, sd = err_sd)) %>%
    inner_join(subjects, "subj_id") %>%
    inner_join(items, "item_id") %>%
    mutate(RT = b0 + S0s + I0i + (b1 + S1s) * cond + err)

  dat_sim
}
```

298 We will also make a separate function that analyses the simulated data. This makes it
299 easier to try out differnt analyses using the same generation function, which we will do in the
300 next section comparing the results of mixed models to ANOVA.

```
# ... is a shortcut that sends any arguments to my_sim_data()
my_lmer_power <- function(...) {
  dat_sim <- my_sim_data(...)
  mod_sim <- lmer(RT ~ cond + (1 | item_id) + (1 + cond | subj_id),
                  dat_sim, REML = FALSE)

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

Table 9

The output of lmer_power().

effect	group	term	estimate	std.error	statistic	df	p.value
fixed	NA	(Intercept)	791.962	12.770	62.020	128.375	0.000
fixed	NA	cond	96.975	17.798	5.448	53.872	0.000
ran_pars	subj_id	sd___(Intercept)	93.522	NA	NA	NA	NA
ran_pars	subj_id	sd___cond	37.946	NA	NA	NA	NA
ran_pars	subj_id	cor___(Intercept).cond	0.104	NA	NA	NA	NA
ran_pars	item_id	sd___(Intercept)	58.130	NA	NA	NA	NA
ran_pars	Residual	sd___Observation	200.176	NA	NA	NA	NA

```
}
```

Run the function once with default parameters.

```
my_lmer_power()
```

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 condition to 20 ms?

```
my_lmer_power(nitem = c(ingroup = 50, outgroup = 50), b1 = 20)
```

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 using just a few repetitions (`reps`) first, then make sure you save the full results to a CSV file so you can set this code chunk to not run (`eval = FALSE` in the chunk header) and load from the saved data for the rest of your script in the future.

Table 10

The output of lmer_power(nitem = c(ingroup = 50, outgroup = 50), b1 = 20).

effect	group	term	estimate	std.error	statistic	df	p.value
fixed	NA	(Intercept)	830.708	11.720	70.879	183.924	0.000
fixed	NA	cond	19.581	16.618	1.178	125.776	0.241
ran_pars	item_id	sd___(Intercept)	74.900	NA	NA	NA	NA
ran_pars	subj_id	sd___(Intercept)	87.924	NA	NA	NA	NA
ran_pars	subj_id	sd___cond	59.954	NA	NA	NA	NA
ran_pars	subj_id	cor___(Intercept).cond	0.484	NA	NA	NA	NA
ran_pars	Residual	sd___Observation	198.863	NA	NA	NA	NA

```
reps <- 100
sims <- purrr::map_df(1:reps, ~my_lmer_power())
write_csv(sims, "sims.csv")
```

```
sims <- read_csv("sims.csv")
```

You can use these data to calculate power for each fixed effect or plot the distribution of your fixed or random effects.

Comparison to ANOVA

One way many researchers would normally analyse data like this is by averaging each subject's reaction times across the ingroup and outgroup stimuli and compare them using a paired-samples t-test or ANOVA (which is formally equivalent). Here, we use `afex::aov_ez` to analyse a version of our dataset that is aggregated by subject.

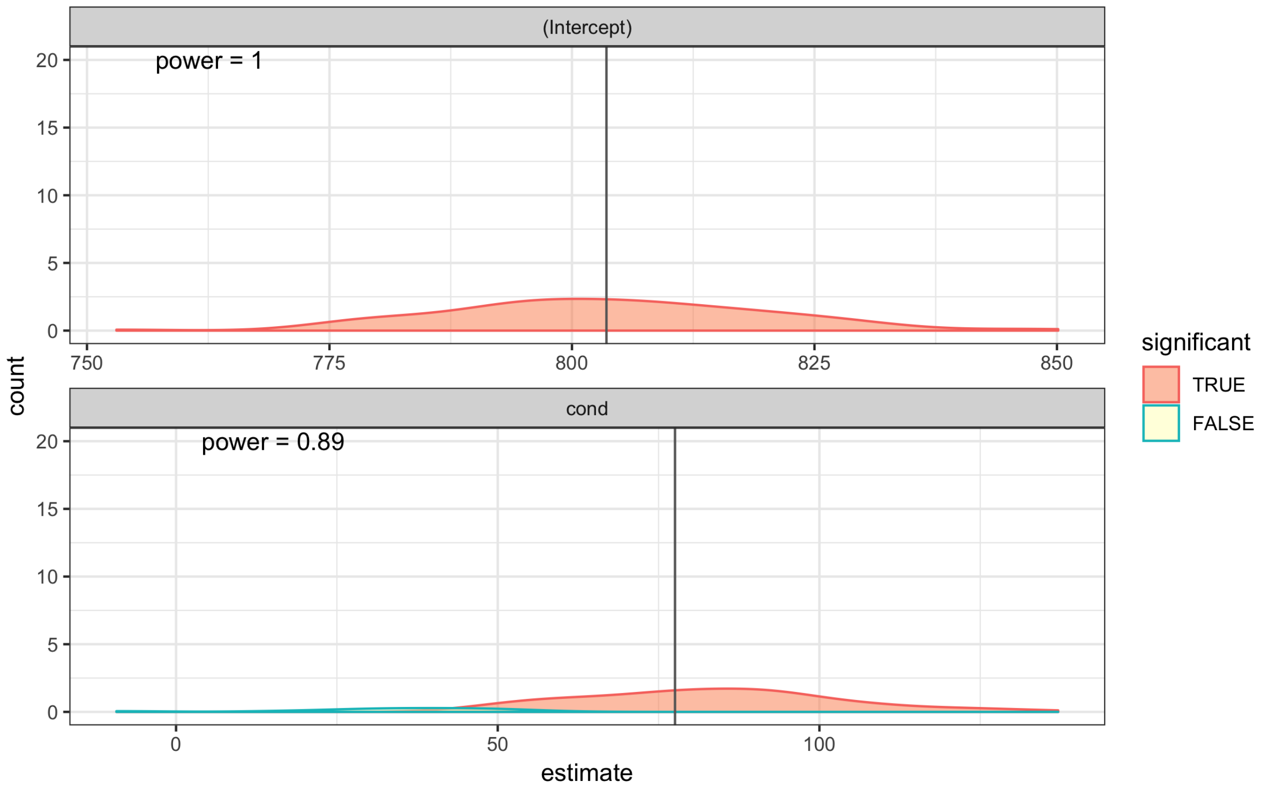

Figure 3. Distribution of fixed effects across 1000 simulations

316 Alternatively, you could aggregate by item, averaging all subjects' scores for each item.

```
dat_item <-  dat_sim %>%
  group_by(item_id, condition, cond) %>%
  summarise(RT = mean(RT))


a_item <- afex::aov_ez(
  id = "item_id",
  dv = "RT",
  between = "condition",
  data = dat_item
)
```

317 We can create a new power analysis function that simulates data using our data

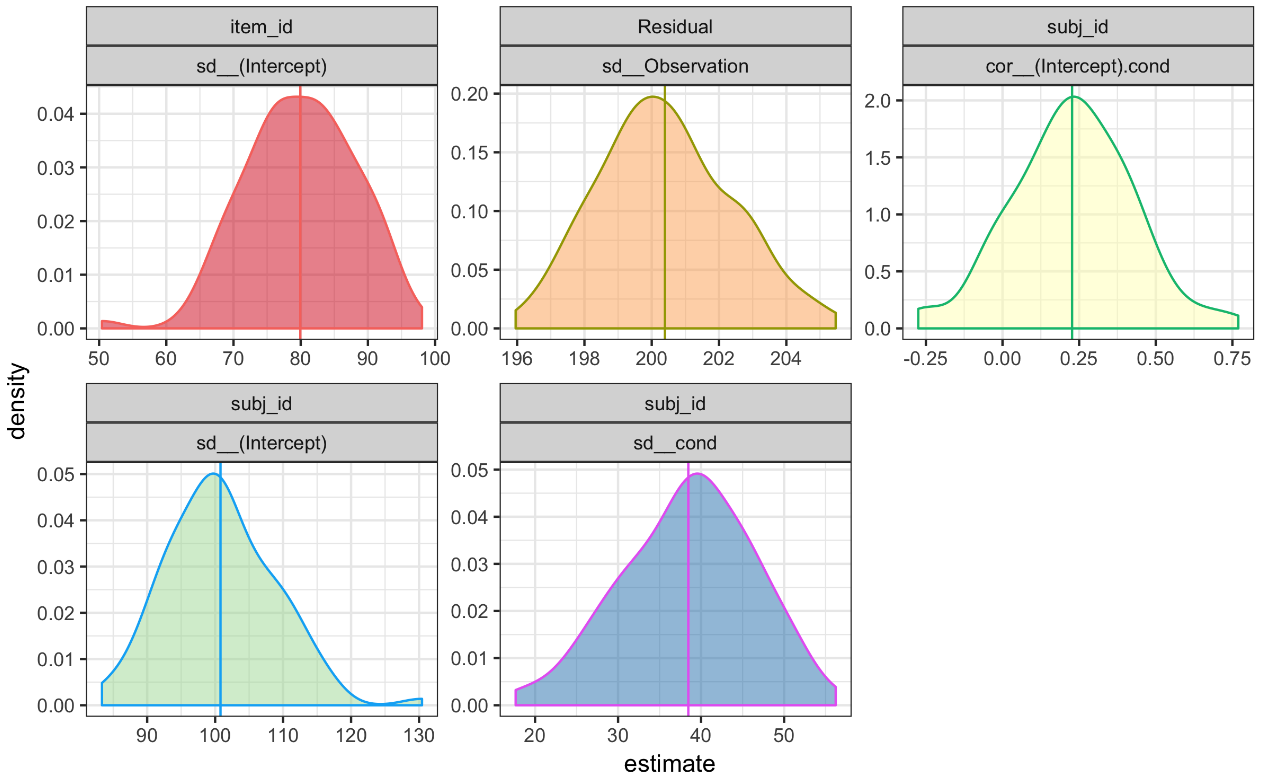

Figure 4. Distribution of random effects across 1000 simulations

318 generating model from `my_sim_data()`, creates these two aggregated datasets, and analyses
319 them with ANOVAs. Here, we'll just return the p-values for the effect of interest.

```
my_anova_power <- function(...) {
  dat_sim <- my_sim_data(...)

  dat_subj <-  dat_sim %>%
    group_by(subj_id, condition, cond) %>%
    summarise(RT = mean(RT))

  dat_item <-  dat_sim %>%
    group_by(item_id, condition, cond) %>%
    summarise(RT = mean(RT))
  a_subj <- afex::aov_ez(id = "subj_id",
                         dv = "RT",
                         within = "condition",
                         data = dat_subj)
  suppressMessages(
    # check contrasts message is annoying
    a_item <- afex::aov_ez(
      id = "item_id",
      dv = "RT",
      between = "condition",
      data = dat_item
    )
  )

  list(
    "subj" = a_subj$anova_table$`Pr(>F)`,
    "item" = a_item$anova_table$`Pr(>F)`
  )
}
```

Run this function with the default parameters to determine the power each analysis has to detect an effect of condition of 80 ms.

```
alpha <- 0.05

reps <- 100
anova_sims <- purrr::map_df(1:reps, ~my_anova_power())
```

```r
power_subj <- mean(anova_sims$subj < alpha)
power_item <- mean(anova_sims$item < alpha)
```

The by-subjects ANOVA has power of 1, while the by-items ANOVA has power of 0.89. This isn't simply a consequence of within versus between design or the number of subjects versus items, but rather a consequence of the inflected false positive rate of some aggregated analyses.

Set the effect of condition to 0 to calculate the false positive rate. This is the probability of concluding there is an effect when there is no actual effect in your population.

```r
reps <- 100
anova_fp <- purrr::map_df(1:reps, ~my_anova_power(b1 = 0))

false_pos_subj <- mean(anova_fp$subj < alpha)
false_pos_item <- mean(anova_fp$item < alpha)
```

Ideally, your false positive rate will be equal to alpha, which we set here at 0.05. You can see that the by-subject aggregated analysis has a massively inflated false positive rate of 0.51. This is not a mistake, but a consequence of averaging items and analysing a between-item factor.

Conclusion

In this tutorial, we have introduced the main concepts needed to get started with mixed effect models. Through data simulation, you can develop your understanding and perform power calculations to guide your sample size plans. We have also demonstrated through simulation the dangers of aggregating data over a unit of analysis. The R code in this tutorial is supplemented by a Shiny app that will allow you to change parameters and inspect the results of LMEM and ANOVA analyses, as well as calculate power and false positives for these analyses.

Term	Definition
by-subjects analysis (F_1)	def
by-items analysis (F_2)	def
crossed random factors	def
data-generating process	def
deflection	def
fixed effect	def
intercept/grand mean	def
random effect	def
random intercept	def
random slope	def
slope	def

Glossary.

References

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. (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. doi: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.

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. (2019). *Faux (beta) (version v0.0.0.9011-beta).* Zenodo. doi:10.5281/zenodo.2669587

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.

Lakens, D., Scheel, A. M., & Isager, P. M. (2018). Equivalence testing for psychological research: A tutorial. *Advances in Methods and Practices in Psychological Science*, *1*(2), 259–269. doi:10.1177/2515245918770963

Locker, L., Hoffman, L., & Bovaird, J. (2007). On the use of multilevel modeling as an alternative to items analysis in psycholinguistic research. *Behavior Research Methods*, *39*, 723–730.

R Core Team. (2018). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Westfall, J., Nichols, T. E., & Yarkoni, T. (2016). Fixing the stimulus-as-fixed-effect fallacy in task fMRI. *Wellcome Open Research*, *1*.