Power Simulations for Multilevel Modeling

A Simple Example

Sean Devine

02 July 2021

Why do we need simulation?

For some analysis techniques, power can be computed analytically using software like G-Power (e.g., ANOVA, t-tests, etc.). In the case of multilevel modeling (MLM) however, this is not possible, as there is no closed-form solution for estimating power in these models. Fortunately, it is still possible to estimate power for MLM using a **simulation-based approach**.

What is simulation?

By simulation, I mean to produce data from *known* population parameters, *assuming a model is true*. To illustrate this, we will use the example of a basic Stroop task. In a Stroop task, we ask people to respond according to the font colour of a presented colour-word. Words are presented in two conditions: **congruent**, where the word matches the font colour (green), or **incongruent**, where the word and font colour do not match (green). The typical predictor is that incongruent word/colour pairs will result in **slower** and **less accurate** responses than congruent pairs. In this tutorial, we will focus on reaction time (RT).

Simulation is accomplished by:

1. Assuming this congruency effect to be true;
2. Specifying the expected effect and sample size;
3. Running the desired model and seeing whether a given effect is significant;
4. Repeating this process a large number of times (i.e., producing a large number of samples) and computing power at various sample sizes, α levels, effect sizes, and/or other parameterizations.

What is power?

Power is defined as the probability of rejecting the null hypothesis when a true effect is present. In the simulations discussed above, we *specify a true effect* ahead of time by setting a given population effect size from which the samples are simulated. As such, power in this case will correspond to the proportion of significant results obtained relative the number of simulations performed. Rules-of-thumb for desired power levels suggest at least 80% power is preferable for statistical inference, though lower power may be reasonable if your particular sample merits it. In any case, a simulation-based approach is a very flexible way to assess a future sample's power under different conditions.

Example

Step 1. Load packages

Before simulating, we need to load the relevant packages. In our case, we will be using MLM for each simulated sample and extracting p-values, so we need to load `lme4` and `lmerTest`. Additionally, we will set a seed at the beginning of the script for reproducibility[1].

```
library(lme4)
library(lmerTest)
set.seed(10408)
```

Step 2. Set constants

Next, we need to set our constants, a.k.a. parameters. These values will be the basis of our simulated sample and selecting them carefully is the core of simulation. Notably, the constants we select will depend on 1) the experiment we aim to run 2) the model we are assuming the data are produced by.

First, we will set the parameters specific to our experiment. This will include:

- The sample sizes we want to test. This can be any range of values you desire, but it can save time to choose values that are practical to your constrains. For instance, do not test the power of a sample of 500 subjects if you are only capable of acquiring 100; it will only waste computational resources and time;
- The number of trials in your experiment. Here, we will set this to a constant value, but if you are unsure of how many trials to include in your design, you can also vary this value and see how it affects your power. This can be a useful, data-driven, way to design your experiment.
- The number of simulations. For each sample size (and/or trial number size), we will simulate a series of samples. The more simulations you set, the more precise your power estimates will be, but the longer your simulation will take. As a minimum, it is advisable to set the number of simulations to at least 1000, if not more. For the purposes of illustration and speed, I will set this value to 100.
- The α level. This will dictate what counts as a significant effect. Largely values make rejecting the null easier and reduce required sample size, but can drive up Type I error (false positives). Conversely, smaller values make it harder to reject the null, but protect against Type I error. By convention, this value is set here to .05, but this should really be set according to your own field, hypotheses, and desired level of Type I vs. Type II error risk[2].

In R, we can set this at the outset.

```
# Experimental/simulation parameters
N <- seq(10, 100, by=10)   # sample sizes to test
nTrial <- 100              # number of trials per subject
nSim <- 100                # number of simulations per sample size per subject
alpha <- .05               # alpha thereshold
```

Second, we need to set population parameters for our model. Here, we will be using a linear MLM to predict simulated RT in a Stroop task. As such, we begin by setting population parameters for an MLM to simulate from. To know what parameters to set, let us write the MLM formula for the desired model:

[1] Computers cannot produce truly random numbers. Instead, they perform numerical transformations on a starting "seed" value. By setting this seed manually, we can ensure that any "randomness" in our simulations is reproducible should we run the same simulations again or wish to share our code with someone else.

[2] see Kline, R. B. (2013). *Beyond significance testing: Statistics reform in the behavioral sciences*. APA. p. 71

$$
\begin{aligned}
RT_{t,id} &= \beta_{0,id} + \beta_{1,id}\text{congruency}_{t,id} + R_{t,id} \\
\beta_{0,id} &= \gamma_{00} + U_{0,id} \\
\beta_{1,id} &= \gamma_{10} + U_{1,id}
\end{aligned} \tag{1}
$$

where t refers to a given trial, id refers to a simulated participant, congruency refers to an effects-coded Stroop congruency condition equal to either congruent (=1) or incongruent (=-1), $U_{p,id}$ refers to the random variation around the fixed effect, γ_{p0}, for each simulated participant, which is normally distributed: $\mathcal{N}(0, \tau_p^2)$, and $R_{t,id}$ refers to the within-participant residual variation, which is normally distribed: $\mathcal{N}(0, \sigma^2)$

With this model in mind, we must now choose values for each of these parameters (γ_{00}, τ_1^2, etc.). This step is particularly important, because large effect sizes (parameter values) are easier to detect in small samples. As such, setting these values higher than they are in the actual population can result in an underestimate of true power. Similarly, setting these values too low can result in overestimated power, which can lead to overpowered samples and the significance of otherwise meaningless effects (e.g., ms difference between congruency conditions = 1 ms).

So, how can we decide on these parameter values[3]? This is a tricky question. Partly, this will depend on a researcher's experience with their research topic and knowledge of past work to justify these values. A pilot sample of the experiment may also be conducted and the same model fit to the pilot data as a means of estimating the true effect in the population, or at least values somewhat in that range. Alternatively, the researcher can specify an effect of "minimal interest" that reflects the smallest possible effect that is theoretically interesting. This way, they are guaranteed to find a sample size that has very low Type I error for theoretically interesting effects. If the researcher is truly at a loss, they can also choose a range of possible parameter values and simulate data at each value to gain a lower and upper bound of reasonable sample sizes. All of this approaches are reasonable and should be evaluated on a case-by-case basis. For the sake of demonstration, I chose arbitrary values in this case, which I specify below. Even though this are semi-random values, my experience working with cognitive tasks like the Stroop tells me that these are at least somewhat reasonable numbers. All values are in ms.

```
# Model parameters
G00   <- 350.05            # gamma00 -- grand mean RT
G10   <- 35.55             # gamma11 -- congruency effect for Stroop (assuming effects coding)
tau20 <- 35000             # tau20 -- cluster-level variance about G00
tau21 <- 2500              # tau21 -- cluster-level variance about G10
sigma2 <- 50000            # sigma2 -- within-subject variance
```

Step 3. Create design matrix

Next, we will create a design matrix. This will be a matrix of our predictors that we will then simulated samples of responses for using the model parameters specified above. In the Stroop example, we only need one column to our design matrix: congruency condition. However, it can often be useful for researchers to create a "fuller" design matrix with irrelevant variables that will nevertheless show up in their final dataset, as a way to get a "feel" for the data, prior to actual collection. I do this below by including the words and font colours that will determine whether a pair is congruent or incongruent. I also ensure that that there is an even split of congruent and incongruent pairs in my design matrix.

```
options <- c('green', 'blue', 'red', 'purple')
words <- sample(rep(options, each=nTrial/4))
cols  <- sapply(1:length(words), function(x) ifelse(x<=50, words[x], sample(words[words!=words[x]], siz
congruency <- ifelse(words==cols, -1, 1) # effects coding
```

[3]For those of you familiar with Bayesian analysis, this is analogous to the problem of choosing good priors

```
stroop = data.frame(words, cols, congruency)
head(stroop)
```

```
##    words   cols congruency
## 1    red    red         -1
## 2   blue   blue         -1
## 3    red    red         -1
## 4   blue   blue         -1
## 5 purple purple         -1
## 6 purple purple         -1
```

In reality, the order of participants' data would be randomized for each new participant, but this doesn't matter for simulation.

Step 4. Simulate data

Simulating data is a matter of iterating through the parameters of interest `nSim` many times, simulating data according to the model parameters, computing a MLM on these simulated data, and storing the results. In practice, this is done using a `for` loop, which iterates through each value in a provided vector until it reaches the end. The explanation of `for` loops would be too lengthy for this tutorial document, so I point programming novices to this introduction to the concept: https://www.datamentor.io/r-programming/for-loop/.

The code to run this simulation is included below. First, a data frame, `PowerSim` is created which will store the output from each simulation. We then begin iterating through each value of `1:nSim` (which is just each value from 1 to `nSim`). Within each simulation iteration, `sim`, we iterate again through each sample size, `N`. If we were also testing other parameterizations, like different values for `nTrial`, we would have a third layer of iteration, and so on.

Once in this sample size iteration for this simulation, `thisN`, we can simulate data according to our parameters. `thisb1` corresponds to $\beta_{1,id}$, which is the sum of γ_{00} (`G00`) and a normal random deviation with mean 0 and variance τ_0^2 (`tau20`). We do the same for `thisb1`, which represents $\beta_{1,id}$. Note, these variations will be random, but totally reproducible, because we set the seed the beginning.

Next, we simulate RT values for each participant in `thisN` according to Eq. 1 and the `b0` and `b1` values computed above. These simulated RT values are then stored in `RTsim`.

Finally, the design matrix is repeated for each participant in the sample and the simulated RT are added to the design matrix, along with an `id` column to deliniate RT belonging to each simulated participant (each of which have their own random deviation from the fixed effects, $U_{p,id}$). A model, `thisMLM`, is then fit to the simulated data according to Eq. 1[4]. The results in this model are then stored in a data.frame, `thisSim` and appended to `PowerSim`.

The whole process then repeats for each sample size, `nSim` times.

[4]For some samples, these models are likely to fail to converge, especially in small sample sizes. If one wants, they can explicitly code to exclude these non-converged samples or repeat the sample if it fails to convege. I do no such thing here, but it is an option.

```r
PowerSim <- data.frame(stringsAsFactors = F)

for(sim in 1:nSim) {

  for(thisN in N) {

    # Keep track of progress
    cat('simulation', sim, '/', nSim, 'for sample size:', thisN, '\n')

    # Simulate!
    thisb0 <- G00 + rnorm(thisN, 0, sqrt(tau20))
    thisb1 <- G10 + rnorm(thisN, 0, sqrt(tau21))

    RTsim = c()
    for(sub in 1:thisN) {
      thisRT <- thisb0[sub] + thisb1[sub]*stroop$congruency + rnorm(nTrial, 0, sqrt(sigma2))
      RTsim <- c(RTsim, thisRT)
    }

    # Model!
    thisStroop <- do.call("rbind", replicate(thisN, stroop, simplify = FALSE))
    thisStroop$RT <- RTsim
    thisStroop$id <- rep(1:thisN, each=nTrial)

    thisMLM <- lmer(RTsim ~ congruency + (congruency|id), data=thisStroop)

    # store estimates
    b0 <- fixef(thisMLM)[['(Intercept)']]
    b1 <- fixef(thisMLM)[['congruency']]
    tau20_sim <- as.data.frame(VarCorr(thisMLM))[1,'vcov']
    tau21_sim <- as.data.frame(VarCorr(thisMLM))[2,'vcov']
    sigma2_sim <- sigma(thisMLM)^2
    p <- summary(thisMLM)$coefficients[,'Pr(>|t|)'][['congruency']]

    thisSim <- data.frame(thisN, sim, b0, b1, tau20_sim, tau21_sim, sigma2_sim, p)
    PowerSim <- rbind(PowerSim, thisSim)

  }
}
```

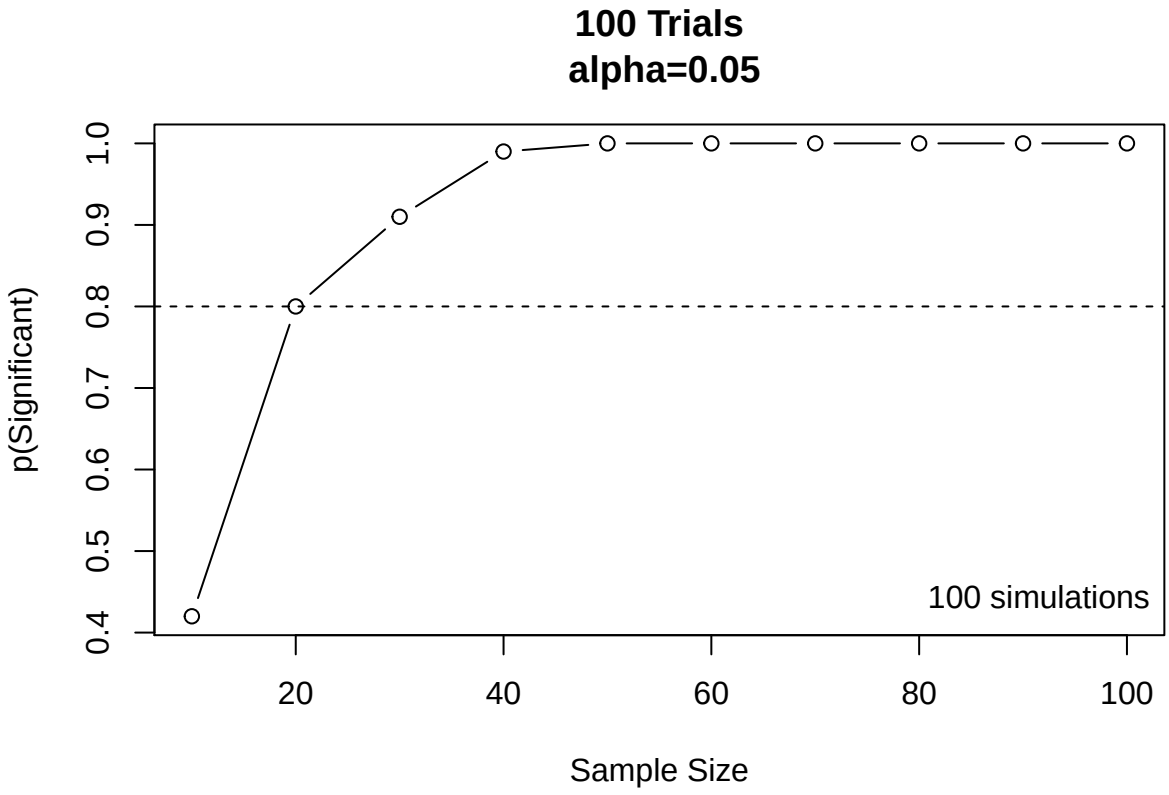
Step 5. Visualise results

`PowerSim` now contains the resluts of our simulation. We can compute power by computing the proportion of significant samples for each sample size; i.e., the proportion of samples where $p < \alpha$. We store the average proportion of significance across sample sizes in `pwr`.

```r
PowerSim$sig <- ifelse(PowerSim$p < alpha, 1, 0)
pwr <- tapply(PowerSim$sig, PowerSim$thisN, mean)
```

We can then plot these values to form a **power curve** and draw a dashed line at the power = 80% mark. Doing so, we can see that 20 participants would yield power of 80% and 30 participants yields power of 91%.

```r
plot(names(pwr), pwr, type='b',
     xlab='Sample Size', ylab='p(Significant)',
     main=paste0(nTrial, ' Trials\n alpha=', alpha))
legend('bottomright', legend=paste0(nSim, ' simulations'), bty='n')
abline(h=0.80, lty='dashed')
```



Conclusion

Simulation-based power analysis is a powerful and flexible tool for estimating power for MLM. It is powerful because it is very simple to increase precision by increasing the number of simulations. It is flexible because the researcher has complete control over the parameters of interest and can manipulate them to determine how it affects power.

Some words of caution before concluding. In the current example, we simulated RT according to linear MLM. That is, we assumed that the residuals of RT were normally distributed with a mean of 0 and variance σ^2, as were deviations around the fixed effects (γ_{00} and γ_{10}). This is certainly untrue, because our simulations allow RT to be negative, which is not possible. This speaks more broadly to the importance of specifying proper models to explain phenomena of interest[5]. Of course, it is common to analyze RT with linear MLM in cognitive tasks and that is why I use it as a prototypical example, but it always important to be aware of the assumptions one is making during simulation as it will ultimately bear on the validity of the estimates in question: here, power.

[5]For instance, when measuring RT, a drift-diffusion model may be more appropriate (Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation, 20*(4), 873-922).