Power: Mixed Effects Models (Random intercepts)

Crispin Jordan

21/07/2021

What does this document provide? What type of model does it address?

This document outlines simulations to conduct power analysis for one of the simplest model types that consider random effects: random effects specified as **random intercepts**. In `lmer()` (and some other packages, too), we specify such random effects as `(1|our.random.effect)`, where `our.random.effect` indicates the categorical variable modeled as a random effect. Models that include both random effects and fixed effects are termed **mixed effects models**; models with only random effects are **random effects models**. (General (or Generalized) linear models with only fixed effects are simply 'General (Generalized) linear models'.)

This document will:

- briefly introduce the variance components and their structure for an experiment with one factor, modeled as a fixed effect, and one random effect, modeled as random intercepts. We focus on General Linear Models (i.e., we assume normally distributed residuals).
- illustrate power analysis for this experimental design, employing random intercepts.

Note that we will consider power analyses for alternative random effects models in other documents (e.g., modeling random slopes).

Our example experiment

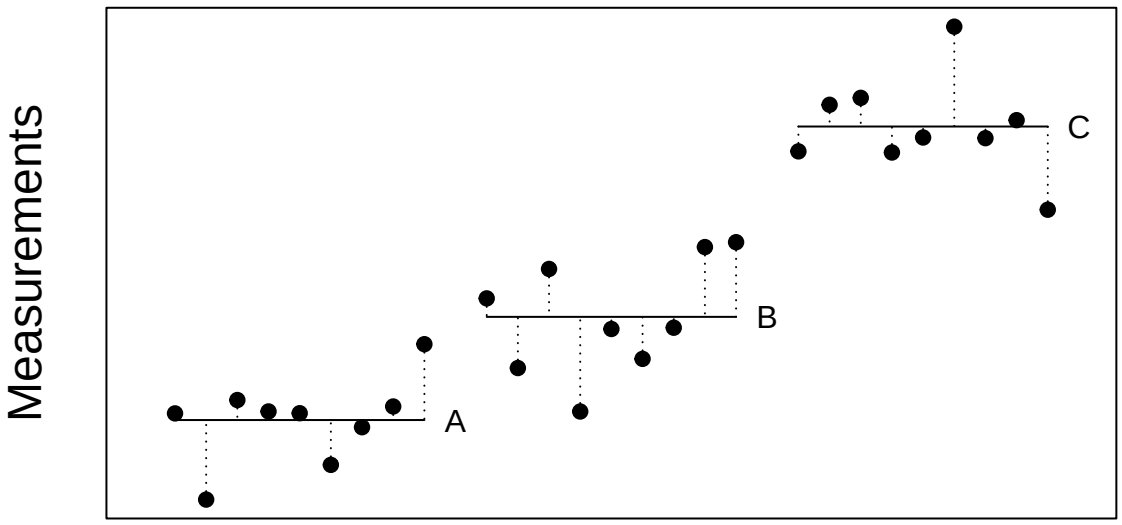
Our example experiment includes one factor (with three levels), modeled as a fixed effect, and one random effect. Imagine that we designed this experiment to test the effect of a drug on several genotypes of an organism (the organism could be a plant, mammal, fungus; it does not matter). We call our fixed effect, `Treatment`, and specify three levels: an unmanipulated treatment (`baseline`), a placebo treatment, (`placebo`), and a drug treatment (`drug`). We term the random effect, `Genotype`, which can have many levels; each level has a unique number, with the prefix `geno` appended (see below), which reminds us that `Genotype` is a `factor`.

Some additional details:

- In our experiment, each `Genotype` experiences all three levels of `Treatment`;
- Within each level of `Treatment`, we obtain multiple measurements for each `Genotype`. Here, we imagine that each measurement of a given `Genotype` within a given level of `Treatment` arose from a separate individual. *(Note that, if we obtained multiple measurements from individual subjects within a* ***`Genotype`****, our analysis should likely account for this additional level of non-independence, e.g., include a second random effect term,* ***`Subject`****. Our simpler design avoids this complication.)*
- We assume that our experiment has no additional sources of non-independence (e.g., subjects inhabiting shared cages, shared food sources, etc.).

Let's look at the structure of these data

If we were to plot the structure of data from our experiment, what would it look like? To answer this question, let's begin with a simpler experiment: a one-factor (3 level) experiment with no random effects. We illustrate the data structure and variance components in the plot, below.



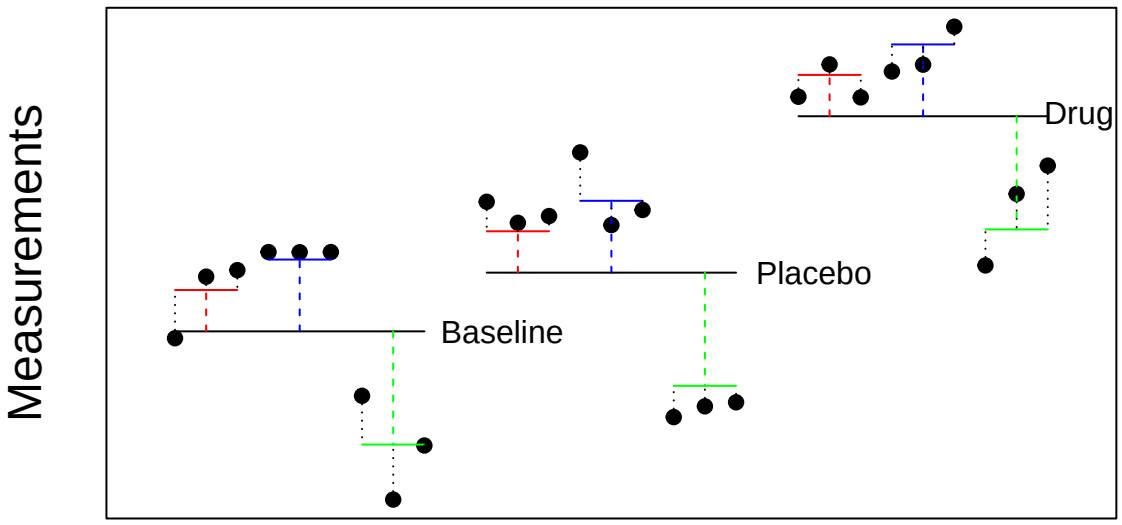
3 Treatments (black, horizontal lines)

This plot illustrates an experiment with one factor with three levels (treatments); the levels are labelled, `A`, `B`, and `C`. The experiment involves 9 measurements (the points) within each of the three treatments. The three, black horizontal lines represent the average values of data within each treatment. *The differences among these mean values* represent the first source of variance in our data: variance due to treatment effects. The dotted, vertical lines highlight the differences between each measurement and the mean of its associated treatment: these differences are the model 'residuals', and represent the second source of variance in our data (residual or unexplained variance).

How does this picture change if we add a random effect (modeled as random intercepts)? The figure, below, illustrates this additional source of variance. Note the similarity to our previous figure, immediately above: both involve experiments with one factor with three levels (i.e., three treatments), and both involve 9 measurements per treatment level. We also see variation among the `Treatment` levels, signifying variance due to treatment effects. Two primary differences arise between the plots:

- Below, we now see coloured, horizonal lines: these represent the random effects. Specifically, the three colours (red, blue and green) represent three levels of our random effect; i.e., three genotypes. Recall from our experimental design that each level of `Genotype` (red, blue, green) appears in each `Treatment` level (`baseline`, `placebo`, `drug`): this explains why each `Genotype` level appears in each `Treatment` level. Now, the important part: the differences between each `Genotype` level and the `Treatment` levels, indicated by the vertical, dashed, coloured lines, highlights a new source of variance in our data: variance associated with the random effect, `Genotype`. *(Please note that an experiment like this should ideally have more* ***`Genotype`*** *levels than illustrated here; we show only three* ***`Genotype`*** *levels to simplify the figure.)*
- Like the plot, above, the plot, below, also illustrates residual variance (i.e., the vertical, dotted, black lines that connect to the points). However, the residual variance, below, occurs as variation in the difference between a measurement and value of its associated `Genotype` level, rather than the difference between a measurement and its associated `Treatment` level, as we saw, above.

We incorporate these differences associated with the presence of a random effect in our simulation, below. First, we simulate variance due to random effects around `Treatment` levels, then we simulate additional residual variance around `Genotype` levels.



3 Treatments (black, horizontal lines)

Please note:

- by modelling the random effects *via* random intercepts, we impose constraints on the data: (*i*) the variance associated with `Genotype` is identical among `Treatment` levels, and (*ii*) and responses by `Genotype` levels are identical among levels of `Treatment`. These constraints are reflected in the fact that the difference between each `Genotype` level and each `Treatment` level is identical for all three `Treatment` levels.
- this structure / model of random effects, despite the fact that it is commonly used to model data (it is the default for some software), may not be biologically realistic. Therefore, it is important to realize that alternative structures / models for the random effects are available. For example, it may be more realistic for variance among levels of `Genotype` to differ among `Treatment` levels. We will deal with some of these alternative random effects structures elsewhere.

Simulating a single dataset

Overall, our approach to conduct power analysis for mixed effects models is identical to what we've seen previously with other analyses: we simulate data, extract and store our criterion for an experiment's success (e.g., p-value or SE for effect size), and repeat. The only prominent differences involve adding a random effect to our data and the function to conduct the analysis (e.g., using `lmer()` or another similar function rather than `lm()`). We will follow these steps to generate code for our power analysis:

- simulate a single dataset;
- analyze this dataset to test whether we simulated the data correctly. If the results of the analysis match our expectations based on how (we believed) we simulated the data, then we gain confidence that our code simulates data correctly.
- apply this code to generate many datasets, analyse each one, and extract and save the the criterion by which we judge the simulated experiment's success (e.g., p-value); i.e., conduct a power analysis.

Before we generate our first dataset, please note that:

- **R** purists will be rather horrified by our code, as it is not as elegant as it could be. We use relatively simple code to communicate the stages of the simulations as clearly as possible, rather than to teach efficient and concise coding.
- we assume that you have seen our previous files that illustrated power analyses via simulations. As a result, we assume you will be familiar with most of the code and we will provide little explanation of how the code works. Instead, we focus on what we need code to accomplish.

Let's begin by creating parameters that describe our experimental design. Recall that `Treatment` has three levels: `baseline`, `placebo` and `drug`. We obtain measurements from each `Genotype` in all three `Treatment` levels. How many levels of (i.e., types of) `Genotype` will our experiment involve? Let's specify the number of `Genotype`s as:

```
no.Genotype <- 200
```

Here, we're imagining that we have 200 `Genotypes`. Is this realistic for a real experiment? That depends on the experiment. A recent GWAS study of cattle used about 58,000 bulls (genotypes). Biomedical science often uses a handful of genotypes. Our goal here is to use a sufficiently large dataset to provide a strong test of our code (via analysing the data, below); 200 should be sufficient.

We said earlier that we will obtain multiple measurements per `Genotype` per `Treatment` level. How many? Let's specify this number via:

```
reps.per.Genotype.per.level <- 10
```

We now can determine the total number of measurements in our experiment: we have 200 levels of `Genotype`, each of which experience each of 3 levels of `Treatment` 10 times. Therefore, our total number of measurements equals `200*3*10 = 6000`.

We need a vector to store the identity `Genotype` levels for which we will simulate data. Each `Genotype` level will have `3*10 = 30` measurements. Recall that we specify each `Genotype` level with a number (to which we will add the prefix, `geno`). As we have 200 `Genotype` levels, and each level is repeated 30 times, we can use the following code to generate a sequence of numbers from 1 to 200, where each number is repeated 30 times:

```
Genotype <- rep(1:no.Genotype, each=3*reps.per.Genotype.per.level)
```

We can append `geno` to these numbers using the `paste()` function:

```
Genotype <- paste("geno",Genotype, sep = "")
```

Let's check our work. How many entries are in Genotype?

```
length(Genotype)
```

```
## [1] 6000
```

Just as we expected! Let's look at the first 100 entries, to check that our code works properly:

```
Genotype[1:100]
```

```
##   [1] "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1"
##  [10] "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1"
##  [19] "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1" "geno1"
##  [28] "geno1" "geno1" "geno1" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2"
##  [37] "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2"
##  [46] "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno2"
##  [55] "geno2" "geno2" "geno2" "geno2" "geno2" "geno2" "geno3" "geno3" "geno3"
##  [64] "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3"
##  [73] "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3"
##  [82] "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3" "geno3"
##  [91] "geno4" "geno4" "geno4" "geno4" "geno4" "geno4" "geno4" "geno4" "geno4"
## [100] "geno4"
```

That looks right!

We will now generate data for each level of `Genotype`, one at a time. (We recognize that this is not the most efficient approach, but we hope it is clear.)

We will generate our measurements (our 'data') in several steps, and will store these results in a vector

named, `Measurements`. (We keep the nature of our measurements general to allow the reader to imagine many different experiments.) Recall, from our description of random effects above, that the value of each level of `Genotype` lies some distance from each `Treatment` level, and these distances are normally distributed. Therefore, as our first step to simulate `Measurements`, we will simulate the distances that `Genotype` levels lie from `Treatment` means. To do so, let's first specify the mean values for each `Treatment` level:

```
baseline.mean = 10
placebo.mean = 12
drug.mean = 20
```

These mean values suggest that the `placebo` will differ slightly from `baseline`, but the `drug` will differ more substantially.

The differences between `Genotype` levels and the mean values of `Treatment` levels will be normally distributed with a mean of zero and a standard deviation of:

```
sd.genotype <- 5
```

With this in mind, we can simulate the difference between a single `Genotype` level and the mean of `Treatment` levels using the following code. Notice that we also create a new vector that stores the `Treatment` level:

```
#We create a vector that stores Treatment level
Treatment <- c()
Treatment <- append(Treatment, c(rep("baseline",reps.per.Genotype.per.level),
                                 rep("placebo",reps.per.Genotype.per.level),
                                 rep("drug",reps.per.Genotype.per.level)))
#We create a vector that stores Measurements
Measurements <- c()
means.vector <- c(rep(baseline.mean,reps.per.Genotype.per.level),
                  rep(placebo.mean,reps.per.Genotype.per.level),
                  rep(drug.mean,reps.per.Genotype.per.level))
random.effects <- c(rep(rnorm(1,0,sd.genotype),3*reps.per.Genotype.per.level))
Measurements <- append(Measurements, means.vector+random.effects)
```

Now, let's see what we have generated for a single `Genotype` level:

```
Measurements
```

```
##  [1] 12.11826 12.11826 12.11826 12.11826 12.11826 12.11826 12.11826 12.11826
##  [9] 12.11826 12.11826 14.11826 14.11826 14.11826 14.11826 14.11826 14.11826
## [17] 14.11826 14.11826 14.11826 14.11826 22.11826 22.11826 22.11826 22.11826
## [25] 22.11826 22.11826 22.11826 22.11826 22.11826 22.11826
```

Excellent! Notice that the difference between each value and the mean for the respective `Treatment` level is consistent for all `Treatment` levels. This matches our description of a *random intercepts* model, above.

Now we must repeat this process for all `Genotype` levels. We will do so using `for()` loops:

```
#We create a vector that stores Treatment level
Treatment <- c()
#We populate that vector for the entire length of the dataset:
for(i in 1:no.Genotype){
  Treatment <- append(Treatment, c(rep("baseline",reps.per.Genotype.per.level),
                                   rep("placebo",reps.per.Genotype.per.level),
                                   rep("drug",reps.per.Genotype.per.level)))

}
#We create a vector that stores Measurements
Measurements <- c()
```

```
#We populate this vector with means + random deviations for the whole dataset:
for(i in 1:no.Genotype){
  means.vector <- c(rep(baseline.mean,reps.per.Genotype.per.level),
                    rep(placebo.mean,reps.per.Genotype.per.level),
                    rep(drug.mean,reps.per.Genotype.per.level))
  random.effects <- c(rep(rnorm(1,0,sd.genotype),3*reps.per.Genotype.per.level))
  Measurements <- append(Measurements, means.vector+random.effects)
}
```

Let's check our work by examining the first 100 entries of `Treatment` and `Measurements`:

```
Treatment[1:100]
```

```
##   [1] "baseline" "baseline" "baseline" "baseline" "baseline" "baseline"
##   [7] "baseline" "baseline" "baseline" "baseline" "placebo"  "placebo"
##  [13] "placebo"  "placebo"  "placebo"  "placebo"  "placebo"  "placebo"
##  [19] "placebo"  "placebo"  "drug"     "drug"     "drug"     "drug"
##  [25] "drug"     "drug"     "drug"     "drug"     "drug"     "drug"
##  [31] "baseline" "baseline" "baseline" "baseline" "baseline" "baseline"
##  [37] "baseline" "baseline" "baseline" "baseline" "placebo"  "placebo"
##  [43] "placebo"  "placebo"  "placebo"  "placebo"  "placebo"  "placebo"
##  [49] "placebo"  "placebo"  "drug"     "drug"     "drug"     "drug"
##  [55] "drug"     "drug"     "drug"     "drug"     "drug"     "drug"
##  [61] "baseline" "baseline" "baseline" "baseline" "baseline" "baseline"
##  [67] "baseline" "baseline" "baseline" "baseline" "placebo"  "placebo"
##  [73] "placebo"  "placebo"  "placebo"  "placebo"  "placebo"  "placebo"
##  [79] "placebo"  "placebo"  "drug"     "drug"     "drug"     "drug"
##  [85] "drug"     "drug"     "drug"     "drug"     "drug"     "drug"
##  [91] "baseline" "baseline" "baseline" "baseline" "baseline" "baseline"
##  [97] "baseline" "baseline" "baseline" "baseline"
```

```
Measurements[1:100]
```

```
##   [1] 10.565830 10.565830 10.565830 10.565830 10.565830 10.565830 10.565830
##   [8] 10.565830 10.565830 10.565830 12.565830 12.565830 12.565830 12.565830
##  [15] 12.565830 12.565830 12.565830 12.565830 12.565830 12.565830 20.565830
##  [22] 20.565830 20.565830 20.565830 20.565830 20.565830 20.565830 20.565830
##  [29] 20.565830 20.565830  7.476926  7.476926  7.476926  7.476926  7.476926
##  [36]  7.476926  7.476926  7.476926  7.476926  7.476926  9.476926  9.476926
##  [43]  9.476926  9.476926  9.476926  9.476926  9.476926  9.476926  9.476926
##  [50]  9.476926 17.476926 17.476926 17.476926 17.476926 17.476926 17.476926
##  [57] 17.476926 17.476926 17.476926 17.476926  5.341830  5.341830  5.341830
##  [64]  5.341830  5.341830  5.341830  5.341830  5.341830  5.341830  5.341830
##  [71]  7.341830  7.341830  7.341830  7.341830  7.341830  7.341830  7.341830
##  [78]  7.341830  7.341830  7.341830 15.341830 15.341830 15.341830 15.341830
##  [85] 15.341830 15.341830 15.341830 15.341830 15.341830 15.341830 11.832092
##  [92] 11.832092 11.832092 11.832092 11.832092 11.832092 11.832092 11.832092
##  [99] 11.832092 11.832092
```

Convince yourself that this output is correct. (Because it is!)

So far, our data in `Measurments` only includes information about `Treatment` means and variation due to random effect: we need to add residual variation to complete our simulation of these data. We'll do this by drawing as many random numbers from a normal distribution as we have data points in our experiment (`3*no.Genotype*reps.per.Genotype.per.level = 6000`). The random numbers will have a mean of zero and a standard deviation equal to:

```
sd.residual <- 10
```

We will draw these random numbers and add them to the values already stored in `Measurements` to complete our simulation of these data:

```
Measurements <- Measurements + rnorm(3*no.Genotype*reps.per.Genotype.per.level, 0, sd.residual)
```

Let's now look at the first 100 values in `Measurments` to see what we did:

```
Measurements[1:100]
```

```
##   [1]  -3.76505903  23.68884113  13.32309805  -0.03334066   6.54333621
##   [6]  19.16131558  12.22296277   7.35262469   9.69582077  20.25492192
##  [11]   5.48828278   6.68890190  11.28950844   6.69762482   9.11171673
##  [16]  15.76604179  10.00128917   3.44973908  25.31150921  -3.05517210
##  [21]  38.47962441  20.06225942  13.90155685  12.11300832  13.04022566
##  [26]  11.36097272  36.66968407  25.70556619  11.87166611  23.18170081
##  [31]   7.47587396   5.21261073  25.82798783  10.62452912  -4.70754728
##  [36]  -0.47028246 -13.26887836  14.98627064   3.21160771 -11.30775877
##  [41]  21.88311079  28.35612908  11.26439191  25.70285869  21.78524652
##  [46]   8.58786638  16.66986788   4.49153909   7.85092582  -2.23790870
##  [51]  23.53308106  11.68613990   2.66270545   9.68727775   5.02881016
##  [56]  29.68338948  29.36039847  19.77734401  27.92331062  27.28133552
##  [61] -11.27121757   7.91932386  23.98255840   3.44547407  -9.49847163
##  [66]   1.22394438   4.75957303  10.72402465   0.79850397  -4.21877339
##  [71]  -2.05161914  -8.64328009   6.81538404  31.31342507   9.44092082
##  [76]  -1.37557813  -8.49862489  13.09210883  -1.87980883  -6.77273250
##  [81]  15.76752799  21.50901088  10.49531415  26.31603104  -9.39837893
##  [86]  10.64445277  10.40634878  30.72880002   5.44570753   7.61617094
##  [91]  -1.50908625  -5.47137536   2.03992353   4.33352841   6.78053055
##  [96]  11.78634299   9.59399458   6.50836935   0.30829546  21.43795733
```

Notice how the contents of `Measurements` has changed: individual values are no longer identical due to the addition of residual variation.

Analyzing our simulated dataset

Now, we have simulated a single dataset and we are almost ready to analyze these data to ensure we simulated the data correctly. First, let's collect all of our relevant code and put it in one place to help us see what we have done:

```
#Define parameters:
no.Genotype <- 200
reps.per.Genotype.per.level <- 10

#Define mean values and variation for random effect and residuals
baseline.mean = 10
placebo.mean = 12
drug.mean = 20
sd.genotype <- 5
sd.residual <- 10

#Create vector with Genotype:
Genotype <- rep(1:no.Genotype, each=3*reps.per.Genotype.per.level)
Genotype <- paste("geno",Genotype, sep = "")

#We create a vector that stores Treatment level
```

```
Treatment <- c()
#We populate that vector for the entire length of the dataset:
for(i in 1:no.Genotype){
  Treatment <- append(Treatment, c(rep("baseline",reps.per.Genotype.per.level),
                                   rep("placebo",reps.per.Genotype.per.level),
                                   rep("drug",reps.per.Genotype.per.level)))

}

#We create a vector that stores Measurements
Measurements <- c()
#We populate this vector with means + random deviations for the whole dataset:
for(i in 1:no.Genotype){
  means.vector <- c(rep(baseline.mean,reps.per.Genotype.per.level),
                    rep(placebo.mean,reps.per.Genotype.per.level),
                    rep(drug.mean,reps.per.Genotype.per.level))
  random.effects <- c(rep(rnorm(1,0,sd.genotype),3*reps.per.Genotype.per.level))
  Measurements <- append(Measurements, means.vector+random.effects)
}

Measurements <- Measurements + rnorm(3*no.Genotype*reps.per.Genotype.per.level, 0, sd.residual)
```

Now, let's analyze the data. We'll use `lmer()` in the `lme4` library (and the `lmerTest` library, which will generate p-values for us). We open the libraries, create our model, and examine the output like this:

```
library(lme4)
```

```
## Loading required package: Matrix
```

```
library(lmerTest)
```

```
##
## Attaching package: 'lmerTest'
```

```
## The following object is masked from 'package:lme4':
##
##     lmer
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```
check.sims <- lmer(Measurements ~ Treatment + (1|Genotype))

summary(check.sims)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: Measurements ~ Treatment + (1 | Genotype)
##
## REML criterion at convergence: 45309.9
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.8366 -0.6553 -0.0108  0.6727  3.8501
##
## Random effects:
```

```
##  Groups   Name        Variance Std.Dev.
##  Genotype (Intercept)  30.38    5.512
##  Residual             103.33   10.165
## Number of obs: 6000, groups:  Genotype, 200
##
## Fixed effects:
##                   Estimate Std. Error        df t value Pr(>|t|)
## (Intercept)         9.4769     0.4512  287.9029  21.005  < 2e-16 ***
## Treatmentdrug       9.9888     0.3215 5798.0000  31.074  < 2e-16 ***
## Treatmentplacebo    2.4631     0.3215 5798.0000   7.662 2.13e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Trtmntd
## Treatmntdrg -0.356
## Tretmntplcb -0.356  0.500
```

Our output indicates that our simulation worked correctly, based on (*i*) the estimates of standard deviation (for random effects and residuals) and (*ii*) the estimated mean values for the `Treatment` levels. First, examine the estimates of standard deviation, provided in the output under `Random effects:`. You will find two rows, labelled `Genotype (Intercept)` and `Residual`. The values on the far-right of these rows (under `Std.Dev.`) provide the model estimates of the standard deviation for our random effect (`Genotype`) and the residuals: notice that these estimates closely match the values we specified (5 and 10, respectively). Next, examine the `Estimate` values under `Fixed effects` (these are the model coefficients). The first row (labeled `(Intercept)`) provides the estimate of the mean for the `baseline` level: notice that its value under `Estimate` is very close to the value of 10 that we specified. The `Estimate` values in the second and third rows indicate the differences between the mean `drug` and `placebo`, respectively, vs. the mean of `baseline`. Recall that we specified the means of `drug` and `placebo` as 20 and 12, respectively, so the differences between these means and the mean of `baseline` will equal 10 and 2, respectively. Again, notice how closely these values match the `Estimate`s. This close match between our model estimates and our expectations suggest that we have simulated the data effectively.

Conduct power analysis

We are now in a position to conduct our power analysis! The overall approach is identical to what we've seen before: simulate many datasets and, for each dataset, perform an analysis and record our criterion of 'success'. Following convention, our example power analysis will obtain the p-value for the overall effect of `Treatment` and test whether it is less than 0.05. Below, we remind you of other approaches.

Our power analysis will use larger variances for random effects and residuals, as well as fewer `Genotype` levels and fewer `reps.per.Genotype.per.level` than we used, above, to obtain statistical power consistent with what's commonly desired in the literature (our simulated data, above, where `no.Genotype` equaled 200, would have very high power).

```
#Open libraries:
library(lme4)
library(lmerTest)

#Define parameters:
no.Genotype <- 25
reps.per.Genotype.per.level <- 4

#Define mean values and variation for random effect and residuals
baseline.mean = 10
placebo.mean = 12
```

```r
drug.mean = 20
sd.genotype <- 15
sd.residual <- 25

#Define the number of data simulations:
n.sims <- 1000
p.less005.counter <- 0

#Create vector with Genotype:
Genotype <- rep(1:no.Genotype, each=3*reps.per.Genotype.per.level)
Genotype <- paste("geno",Genotype, sep = "")

#We create a vector that stores Treatment level
Treatment <- c()
#We populate that vector for the entire length of the dataset:
for(i in 1:no.Genotype){
  Treatment <- append(Treatment, c(rep("baseline",reps.per.Genotype.per.level),
                                   rep("placebo",reps.per.Genotype.per.level),
                                   rep("drug",reps.per.Genotype.per.level)))
  }

#It is only the part of the data-simulation process, below, that must be repeated many times.

for(i in 1:n.sims){
  #We create a vector that stores Measurements
  Measurements <- c()
  #We populate this vector with means + random deviations for the whole dataset:
  for(i in 1:no.Genotype){
    means.vector <- c(rep(baseline.mean,reps.per.Genotype.per.level),
                  rep(placebo.mean,reps.per.Genotype.per.level),
                  rep(drug.mean,reps.per.Genotype.per.level))
    random.effects <- c(rep(rnorm(1,0,sd.genotype),3*reps.per.Genotype.per.level))
    Measurements <- append(Measurements, means.vector+random.effects)
    }
  Measurements <- Measurements + rnorm(3*no.Genotype*reps.per.Genotype.per.level, 0, sd.residual)
  #Analyze the data:
  sim.lmer <- lmer(Measurements ~ Treatment + (1|Genotype))
  #Obtain p-value, and increment counter if p < 0.05:
  if(anova(sim.lmer)[1,6] < 0.05) {p.less005.counter <- p.less005.counter + 1}
}

print(paste("power = ",(p.less005.counter/n.sims)))
```

```
## [1] "power =  0.794"
```

A few last words on ways to expand on this approach

We end with a few suggestions to expand on the methods demonstrated, above.

- We have demonstrated power analysis for one focal experimental design. We can conduct power analyses for similar, but non-identical experimental designs in a similar manner we used here, by altering the structure of the data to reflect the alternative experimental design. For example, if each `Genotype` level occurred within one `Treatment` level *only* (as opposed occurring in all three `Treatment` levels), we would alter the simulations to specify a mean for each `Genotype` level within a single `Treatment` level,

only. Indeed, we can use the methods illustrated, above, for any experimental design (e.g., multiple factors, covariates and random effects), so long as we simulate the data accordingly.

- Note that a variety of methods exist to obtain p-values for mixed effects models. The approach used, above, has been shown to be anti-conservative, and other methods are preferable. For example, the function, `PBmodcomp` (in the `pbkrtest` library) uses parametric bootstrapping to generate p-values, which are more accurate (see Halekoh and Højsgaard (2014), *A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models – The R Package pbkrtest*). Note, however, that parametric bootstrapping is much slower and so was avoided, here.
- Our power analysis, above, focused on a p-value for an overall effect of `Treatment`. We might, instead, wish to base the power analysis on p-values from post-hoc comparisons (e.g., Tukey test). We can use the same approach to do so with mixed effects models as we illustrated previously with 1-factor general linear models.
- We used p-values to judge the 'success' of simulated experiments. Alternatively, we can base 'success' upon the precision with which we estimate an effect size. We used this approach in our discussion of power analysis for 1-factor GLM (where we used the Standard Error of an effect size to assess 'success'), and we can use an identical approach with mixed effects models.

Enjoy!