Power Analysis: 1-Factor GLM

Crispin Jordan

13/06/2021

Opening comments

This file will demonstrate power analysis for 1-Factor GLM via simulation. Before beginning, we hightlight several key features:

- Performing power analysis for 1-Factor GLM via simulation can provide more relevant results than you would obtain with standard software, e.g., using GPower, or even the `pwr` library in **R**. This is because these latter approached determine power at the level of the whole experiment (i.e., at the level of the p-value for GLM (or equivalently, 1-way ANOVA)), and not at the level of post-hoc tests. As a result, an experimental design with high power at the level of the overall test may have low power at the level of post-hoc tests (e.g., pairwise comparisons). This is often not realized, and may be undesirable. This file demonstrates power analysis at both levels of analysis (overall result, and post-hoc tests) to provide you with the tools to design experiments with appropriate power for your purposes.
- We will demonstrate two approaches to post-hoc tests using the functions 1) `pairwise.t.test()` and 2) `emmeans` and `pairs` (the latter two in the `emmeans` library).
- We will demonstrate power analysis based on two criteria for 'successful' experiments: 1) obtaining a p-value with at least 'moderate' evidence for an effect (i.e., `p < 0.05`); 2) estimating an effect size with (at least) a specified level of precision (based on the standard error (SE) of the effect size).

Please note that this file builds upon a previous file in which we demonstrated power analysis via simulation for a t-test. Therefore, to minimize repetition from this previous file, this current document will focus upon new aspects of our approach.

Please note that Nick Colegrave and Graeme Ruxton's book (published 2021), *"Power Analysis: An introduction for the life sciences"* (Oxford Biology Primers) provides an excellent and accessible discussion of power analysis, and teaches power analysis via simulations in **R**. I have drawn heavily from this book to create the current document. If you wish to know more about power analysis, please consider this book.

Getting ready for a power analysis.

Power analysis requires preparation. First, we need a sense of a relevant effect size; this decision is based on biology, not statistics (discussed in videos), and we will not re-visit this topic here. Second, we need a sense of the expected with-group variation for our planned experiment; i.e., an estimate of the 'residual' (or 'error') variance. Or, equivalently, standard deviation. (Recall that standard deviation equals the square-root of the variance). Here, we demonstrate one method to obtain the residual standard deviation from a dataset.

Imagine that you have conducted a pilot study to obtain a reasonable sense of the residual variation: once you have these data, how can you obtain an appropriate value? Let's use **R**'s dataset, `PlantGrowth`, as an example to demonstrate. Our first step would be to load our data into **R**. In our case, we do:

```r
data("PlantGrowth")
```

Let's have a look at the top of the dataframe to learn the names of our variables:

```r
head(PlantGrowth)
```

```
##   weight group
## 1   4.17  ctrl
## 2   5.58  ctrl
## 3   5.18  ctrl
## 4   6.11  ctrl
## 5   4.50  ctrl
## 6   4.61  ctrl
```

We see two columns: `weight` and `group`. This experiment was presumably performed to test the hypothesis that qualities of `group` would affect `weight`. Based on this premise, we conclude that `weight` is the *dependent* variable and `group` is the *independent* variable.

We will model these data, using `lm()`, to estimate the residual variance. We model our data as shown, below, listing the *dependent* variable to the left of the tilda (`~`) and *independent* variable to the right:

```r
pilot.study.lm <- lm(weight ~ factor(group), data = PlantGrowth)
```

Note that we used the function, `factor()`, to ensure that `lm()` treats our independent variable as a factor (or categorical variable). Recent versions of R can require the step; it is a good habit to build.

Ordinarily, we would check the model assumptions to determine whether the data are appropriate for this analysis. However, we'll skip that step because it is not our focus and I do not want to make this file overly long.

We can obtain an estimate of residual variation in two ways from the model output. First, we'll use the model `summary()`:

```r
summary(pilot.study.lm)
```

```
##
## Call:
## lm(formula = weight ~ factor(group), data = PlantGrowth)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0710 -0.4180 -0.0060  0.2627  1.3690
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)          5.0320     0.1971  25.527   <2e-16 ***
## factor(group)trt1  -0.3710     0.2788  -1.331   0.1944
## factor(group)trt2   0.4940     0.2788   1.772   0.0877 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6234 on 27 degrees of freedom
## Multiple R-squared:  0.2641, Adjusted R-squared:  0.2096
## F-statistic: 4.846 on 2 and 27 DF,  p-value: 0.01591
```

Notice the term, `Residual standard error:`. This term is unfortunate because the value being displayed (0.6234) is not, in fact, a standard error. Instead, this value equals the *standard deviation* of the residual variation for our model. This is the value we would often use to simulate the residual variation in our data (as we will do, below).

Alternatively, we can extract this value from our model output using the function, `sigma()`:

```r
sigma(pilot.study.lm)
```

```
## [1] 0.6233746
```

Notice the value is the same as in the output of `summary()`.

Overview

We will build our simulation in five steps:

- First, we'll simulate a dataset;
- Second, we'll analyze the simulated data using `lm()` and obtain and store the overall p-value;
- Third, we'll conduct post-hoc tests and store the p-values. We will use two approaches to demonstrate this step;
- Fourth, we will automate the code from steps 1-3 to run many times;
- Fifth, we will calculate power.

Afterwards, we will also demonstrate power analysis based on the precision to which we estimate an effect size, instead of based on p-values.

Building the simulation. Stage 1: simulate the data.

Imagine that our experiment had one Control treatment (`Control`) and two Treatment groups (`Treat1` and `Treat2`); together, these are 3 levels of a factor called, `Experimental.Groups`. Our goal is to compare the means from the two Treatments vs. the Control. We begin our simulation by establishing the relevant mean values for each treatment (these means would reflect our chosen effect sizes) and the appropriate measure of residual variation. For fun, we'll use the measure of residual standard deviation we obtained from our example, above, and assign it to the object, `sd.within`. We do this here:

```r
control.mean <- 10
treat1.mean <- 9.5
treat2.mean <- 10.5
sd.within <- 0.6233746
```

Further, let's imagine that we wish to determine the power of this experiment if we obtained 10 independent measurements per level of our factor, `Experimental.Groups`. We'll assign this sample size to the object, `n.per.group`:

```r
n.per.group <- 10
```

Now, we have the three pieces of information we need to simulate data:

- Mean values that correspond to desired effect sizes;
- An estimate of the residual variation;
- Our sample size (we need to know how many data points to simulate).

We will use the same approach to generate a dataset as we did to simulate data for a t-test, with one difference: we will simulate data for three groups instead of two. First, we'll create a vector, called, `Experimental.Groups`, which will denote to which level each data point belongs:

```r
Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
```

Next, we'll generate the data in a vector called, `y.var`

```r
y.var <- c(rnorm(n.per.group,control.mean,sd.within),rnorm(n.per.group,treat1.mean,sd.within),
           rnorm(n.per.group,treat2.mean,sd.within))
```

Notice that, above, we simulated data for treatments that match the treatment order in `Experimental.Groups` (i.e., we're being careful to match the information between `Experimental.Groups` and `y.var`.)

Now, we will combine our two vectors to create a dataframe; we'll call the dataframe, `sim.data`:

```
sim.data<-data.frame(Experimental.Groups,y.var)
```

Building the simulation. Stage 2: Model data to obtain primary p-value.

Now that we have our data, we will analyze it using a general linear model (`lm()`). We'll save our model output in the object, `sim.lm`:

```
sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
```

Now we need to extract the p-value from the model output and save it; we'll use this p-value, later, to determine the power of our experiment to obtain at least moderate evidence (`p < 0.05`) for an effect of `Experimental.Groups` on `y.var`. We can obtain the p-value from an ANOVA table of our model output:

```
anova(sim.lm)
```

```
## Analysis of Variance Table
##
## Response: y.var
##                             Df  Sum Sq Mean Sq F value    Pr(>F)
## factor(Experimental.Groups)  2  7.6309  3.8154  9.6526 0.0006877 ***
## Residuals                   27 10.6724  0.3953
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This table provide much more information than we need; we only want the p-value. We'll creep up on the p-value… First, let's obtain the first row of the output. We can do this by using the square-brackets ('[ ]'), in the same way that we do to obtain specific entries from a dataframe. Remember that, when obtaining values from a dataframe using square brackets, we list 'rows' before 'columns' and place a comma in between. We do the same here; to obtain the first row of the ANOVA table we enter:

```
anova(sim.lm)[1,]
```

```
## Analysis of Variance Table
##
## Response: y.var
##                             Df Sum Sq Mean Sq F value    Pr(>F)
## factor(Experimental.Groups)  2 7.6309  3.8154  9.6526 0.0006877 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Cool, eh?

Now, notice that the p-value lies in the fifth column. Therefore, to obtain only the p-value, specify the fifth column of the first row of the ANOVA table:

```
anova(sim.lm)[1,5]
```

```
## [1] 0.0006877138
```

Now that we know how to extract the p-value from our GLM, we wish to save it in a place where we can collect p-values from `lm()` output arising from many simulations.

To do so, we'll first save the p-value in an object called `p`:

```
p <- anova(sim.lm)[1,5]
```

Now, we'll create an empty vector (`anova.vector.p`), where we will store all of the p-values from the ANOVA tables that result from our various simulations.

We create the vector, and then add the p-value to the vector, like this:

```r
anova.vector.p <- c()
anova.vector.p <- append(anova.vector.p,p)
```

Building the simulation. Stage 3: post-hoc tests.

Now that we have obtained a p-value for our overall model, we will obtain p-values for post-hoc tests. As mentioned, above, we will explore two methods to obtain these p-values.

First, we'll use a method implementing the function, `pairwise.t.test()`. As its name implies, this function compares all combinations of mean values among groups via a series of t-tests. To run `pairwise.t.test()` we supply the original data (not the output of `lm()`; see below) by naming the columns that hold the dependent variable (which we named, `y.var`) and the independent variable (which we named, `Experimental.Groups`); we list the dependent variable first. We also specify whether we wish to adjust the p-values to account for multiple comparisons, and if so, which method to use.

We'll make a brief tangent here to consider the topic of p-value adjustment for multiple comparisons. Most students will expect us to adjust p-values using the Tukey method (i.e., perform a Tukey test). However, opinion varies widely both in terms of which method of adjustment to use and whether p-values should be adjusted in the first place! For example, here is a paper that argues against the adjustment of p-values for multiple comparisons: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-842X.2012.00652.x. To make our lives simple, in this example we'll not adjust p-values here (but we revisit this topic, below).

We conduct the pairwise t-tests and save the output (in an object we called, `t.out`) using this code:

```r
t.out <- pairwise.t.test(y.var,Experimental.Groups,p.adj = "none")
```

Now, let's look at the output:

```r
t.out
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  y.var and Experimental.Groups
##
##        Control Treat1
## Treat1 0.02980 -
## Treat2 0.04537 0.00016
##
## P value adjustment method: none
```

The output provides three p-values: one for each comparison among our three groups. We explained at the top of this file that we were most interested in comparisons between the two treatment groups (`Treat1`, `Treat2`) vs. the `Control`. Therefore, we wish to obtain p-values for these comparisons, specifically. We begin by obtaining the p-values by adding `$p.value` to our object:

```r
t.out$p.value
```

```
##           Control       Treat1
## Treat1 0.02979808           NA
## Treat2 0.04536842 0.0001557106
```

Now, we can use the square brackets (`[]`'s) in the same way that we did, earlier, to obtain our focal p-values. The output, immediately above, holds the p-value for the comparison between `Treat1` vs. `Control` in row 1, column 1. Therefore, we obtain this p-value with:

```r
t.out$p.value[1,1]
```

```
## [1] 0.02979808
```

Similarly, we can obtain the p-value for the comparison of `Treat2` vs. `Control` from row 2, column 1 like this:

```
t.out$p.value[2,1]
```

```
## [1] 0.04536842
```

Finally, as we did above, we will store each p-value in its own object (which we name `p.1vc` (i.e., `Treat1` vs `Control`) and `p.2vc`) and add these p-values to their respective vectors (which store p-values from pairwise comparisons):

```
p.1vc.vector <- c() #Creating one vector
p.2vc.vector <- c() #Creating another vector

p.1vc <- t.out$p.value[1]
p.1vc.vector <- append(p.1vc.vector,p.1vc)
p.2vc <- t.out$p.value[2]
p.2vc.vector <- append(p.2vc.vector,p.2vc)
```

As a check, let's examine the contents of each vector:

```
p.1vc.vector
```

```
## [1] 0.02979808
```

```
p.2vc.vector
```

```
## [1] 0.04536842
```

Nice! That's what we expect, so we're doing well.

We now have all of the basic code required for our power analyses. All we need to do now is automate the code to run many times and then calculate power; we do that in the next section.

Before we automate the code, however, we'll take another brief tangent. Although we used pairwise comparisons for post-hoc analyses, many alternatives are possible. For example, in one of my publications (https://academic.oup.com/aob/article/118/3/523/1741768?login=true) I compared the average of one level of a factor vs. the *average* of two other levels of that factor. The point here is that, via simulations, we can perform power analyses for any type of comparisons that we wish. We will address methods to implement non-pairwise post-hoc comparisons elsewhere.

Building the simulation. Stage 4: Automating the simulations

Above, we learned to simulate a single dataset and obtain appropriate p-values, both for the overal model (`lm()` output) and for pairwise comparisons among levels of our factor. Here, we will collect all of our work, above, and use a `for()` loop to run our simulations many times.

Here is the collected code. Note that:

- We have created an additional variable, `n.sims`, which designates the total number of simulated datasets;
- We have grouped all the variables and vectors we created at the top of the code;
- We placed the code that creates the dataframe and subsequent analyses in a `for()` loop, which runs for `n.sims` times (we run 10000 simulations in this example):

```
#Our parameters:
n.sims <- 10000
control.mean <- 10
treat1.mean <- 9.5
treat2.mean <- 10.5
sd.within <- 0.6233746
n.per.group <- 10
```

```r
#Our vectors:
anova.vector.p <- c()
p.1vc.vector <- c()
p.2vc.vector <- c()

for(i in 1:n.sims){
        Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
        y.var <- c(rnorm(n.per.group,control.mean,sd.within),
                   rnorm(n.per.group,treat1.mean,sd.within),
                   rnorm(n.per.group,treat2.mean,sd.within))

        sim.data<-data.frame(Experimental.Groups,y.var)
        #Perform GLM and obtain p-value:
        sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
        p <- anova(sim.lm)[1,5]
        anova.vector.p <- append(anova.vector.p,p)
        #Perform pairwise comparisons and obtain p-values:
        t.out <- pairwise.t.test(y.var,Experimental.Groups,p.adj = "none")
        p.1vc <- t.out$p.value[1]
        p.1vc.vector <- append(p.1vc.vector,p.1vc)
        p.2vc <- t.out$p.value[2]
        p.2vc.vector <- append(p.2vc.vector,p.2vc)
}
```

Building the simulation. Stage 5: Calculating power.

Now, our code will have stores `n.sims` number of p-values in each of the vectors, `anova.vector.p`, `p.1vc.vector` and `p.2vc.vector`. We will use these vectors to calculate power for each of the three aspects of our experiment. Let's begin with the results from the overall glm, in vector `anova.vector.p`.

Following standard practice for power analysis, we'll determine how often our glm provided a p-value less than 0.05. We can count the number of p-values in our vector with code like this:

```r
overall.small.p <- length(which(anova.vector.p < 0.05))
overall.small.p
```

```
## [1] 8745
```

How does this code work? The `which()` function evaluates all of the values in the vector, `anova.vector.p` and, for each value, determines whether it is less than 0.05. It then returns the number of the position in the vector at which a `p-value < 0.05`. For example, if the vector, `anova.vector.p` contained only 7 p-values that were less than 0.05, the `which()` function would return the 7 locations within the vector that held these p-values. The `length()` function then counts the number of locations that were returned by `which()`. Together, these functions determine the number is p-values less than 0.05. We stored this number in the object, `overall.small.p`.

Now, to determine the power of our experiment (given the specified sample sizes, effect sizes, residual variation, etc.) to provide at least moderate evidence (i.e., `p < 0.05`) for an overall effect of `Experimental.Groups` on `y.var`, we divide the number of simulations that yielded $p < 0.05$ (stored in `anova.vector.p`) by the number of simulations we ran (`n.sims`):

```r
overall.small.p/n.sims
```

```
## [1] 0.8745
```

We see that, for our overall result from the general linear model, our proposed experiment has over 80% power! Nice!

We will use the same approach to determine the power for each of our pairwise comparisons. The power for a comparison between `Treat1` and `Control` equals:

```
overall.1vc <- length(which(p.1vc.vector<0.05))
overall.1vc/n.sims
```

```
## [1] 0.4003
```

And similarly, the power for a comparison between `Treat2` and `Control` equals:

```
overall.2vc <- length(which(p.2vc.vector<0.05))
overall.2vc/n.sims
```

```
## [1] 0.4179
```

Notice that the power for our pairwise comparisons is much lower (on the order of 40%) than the power for the overall model (over 80%). Which result is most relevant to your study? That depends on the type of conclusion you primarily wish to make. If you're most interested in designing a study that has high power for the overall result (from the p-value of the general linear model, `lm()`), then the current experimental design and sample size might meet your needs. *(Note that some programs, such as GPower, perform power analysis at this level of analysis.)* If, however, your focus lies in specific comparisons among groups, then this power analysis suggest that the current design / sample size is unlikely to suit your needs because power is only around 40%.

Collecting all code, together

Here, we provide all of our code thus far, in one place:

```
#Our parameters:
n.sims <- 10000
control.mean <- 10
treat1.mean <- 9.5
treat2.mean <- 10.5
sd.within <- 0.6233746
n.per.group <- 10

#Our vectors:
anova.vector.p <- c()
p.1vc.vector <- c()
p.2vc.vector <- c()

for(i in 1:n.sims){
        Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
        y.var <- c(rnorm(n.per.group,control.mean,sd.within),
                   rnorm(n.per.group,treat1.mean,sd.within),
                   rnorm(n.per.group,treat2.mean,sd.within))

        sim.data<-data.frame(Experimental.Groups,y.var)
        #Perform GLM and obtain p-value:
        sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
        p <- anova(sim.lm)[1,5]
        anova.vector.p <- append(anova.vector.p,p)
        #Perform pairwise comparisons and obtain p-values:
        t.out <- pairwise.t.test(y.var,Experimental.Groups,p.adj = "none")
        p.1vc <- t.out$p.value[1]
        p.1vc.vector <- append(p.1vc.vector,p.1vc)
        p.2vc <- t.out$p.value[2]
        p.2vc.vector <- append(p.2vc.vector,p.2vc)
}
#Power for overall result from general linear model:
overall.small.p <- length(which(anova.vector.p < 0.05))
overall.small.p/n.sims
```

```
## [1] 0.8739
```

```
#Power for conparison between Treat1 vs. Control:
overall.1vc <- length(which(p.1vc.vector<0.05))
overall.1vc/n.sims
```

```
## [1] 0.416
```

```
#Power for comparison between Treat2 vs. Control:
overall.2vc <- length(which(p.2vc.vector<0.05))
overall.2vc/n.sims
```

```
## [1] 0.4132
```

An alternative apporach to pairwise comparisons

Above, we performed post-hoc comparisons using `pairwise.t.test()`. In this section, we will demonstrate an alternative approach using the `emmeans` library.

To demonstrate this approach, we'll first demonstrate a post-hoc comparison using two functions in the `emmeans` library: a function of the same name (`emmeans()`), and the function `pairs()`. We will then incorporate this approach into our general code, developed above.

Let's begin by opening the `emmeans` library:

```
library(emmeans)
```

As discussed in materials that teach 1-Factor GLM, we begin post-hoc analysis with the `emmeans` library by first calculating the means (and SE's) for each level of our factor (i.e., `Experimental.Groups`). Here is the code:

```
#Using the function, `emmeans`, to calculate means:
sim.emmeans <- emmeans(sim.lm, "Experimental.Groups")
#Let's look at the output:
sim.emmeans
```

```
##  Experimental.Groups emmean    SE df lower.CL upper.CL
##  Control               10.0 0.198 27     9.62    10.43
##  Treat1                 9.4 0.198 27     8.99     9.81
##  Treat2                10.4 0.198 27    10.00    10.81
##
## Confidence level used: 0.95
```

The output provides the estimated means for each group, their SE's, and 95% Confidence Intervals. Notice that `emmeans()` used output from the *model* (not the original dataframe) to provide this output.

Next, we will perform pairwise comparisons using `pairs()`, also from the `emmeans` library. To do so, we simply provide the output from `emmeans()` to `pairs()`:

```
pairs(sim.emmeans)
```

```
##  contrast         estimate   SE df t.ratio p.value
##  Control - Treat1    0.625 0.28 27   2.231  0.0839
##  Control - Treat2   -0.383 0.28 27  -1.367  0.3717
##  Treat1 - Treat2    -1.008 0.28 27  -3.599  0.0035
##
## P value adjustment: tukey method for comparing a family of 3 estimates
```

We can also ask for a summary of this output (we demonstrate this here, because we need `summary()` to extract p-values, below):

```
summary(pairs(sim.emmeans))
```

```
##  contrast         estimate   SE df t.ratio p.value
##  Control - Treat1    0.625 0.28 27   2.231  0.0839
##  Control - Treat2   -0.383 0.28 27  -1.367  0.3717
##  Treat1 - Treat2    -1.008 0.28 27  -3.599  0.0035
##
## P value adjustment: tukey method for comparing a family of 3 estimates
```

This output provides the p-values for each comparison. Remember that we want p-values for comparisons between the `Control` group vs. the two treatment groups (`Treat1`, `Treat2`).

Once again, we'll use the square brackets to obtain the entries we want. For example, we can obtain output for the first row using:

```
summary(pairs(sim.emmeans))[1,]
```

```
##           contrast  estimate        SE df  t.ratio   p.value
## 1 Control - Treat1 0.6247118 0.2799729 27 2.231329 0.0838617
```

. . . And, we can obtain the p-value for the comparison between `Control` vs. `Treat1` using:

```
summary(pairs(sim.emmeans))[1,6]
```

```
## [1] 0.0838617
```

Similarly, we obtain the p-value for the comparison between `Control` and `Treat2` with:

```
summary(pairs(sim.emmeans))[2,6]
```

```
## [1] 0.3716685
```

A brief tangent to discuss multiple comparisons

Here, we briefly discuss a few options with respect to correcting p-values for multiple comparisons.

Notice in the output of `pairs()`, above, that p-values were adjusted for multiple comparisons using a Tukey approach (see comment, `P value adjustment: tukey method for comparing a family of 3 estimates`). We will make two brief comments on this topic.

First, if we wish to do so, we can turn off the p-value adjustment, like this:

```
summary(pairs(sim.emmeans), adjust = "none")
```

```
##  contrast         estimate   SE df t.ratio p.value
##  Control - Treat1    0.625 0.28 27   2.231  0.0342
##  Control - Treat2   -0.383 0.28 27  -1.367  0.1828
##  Treat1 - Treat2    -1.008 0.28 27  -3.599  0.0013
```

Notice:

- The output no longer states that p-values were adjusted via the Tukey method;

- The p-values are smaller (because they have not been adjusted upwards).

Second, notice that `pairs()`' p-value adjustment assumed that we were making three comparisons, but we only wish to make two comparisons. This means that, for our purposes (and if you're someone who agrees with the practice of adjusting p-values for multiple comparisons), the Tukey method will have adjusted our p-values upwards more than necessary. This is because greater adjustments occur for greater numbers of comparisons. What can we do?

A simple solution is to make the adjustment, ourselves. Specifically, we can 1) obtain p-values that were **not** adjusted for multiple comparisons, as demonstrated immediately, above, and 2) compare the un-adjusted p-values against a different threshold p-value than 0.05. For example, we can use the Dunn-Sidak approach (one of many) to calculate a different threshold p-value. If we aim to maintain a 5% Type 1 error rate, we calculate this threshold via, `alpha_new = 1 - (1 - alpha)^(1/m)`, where `alpha = 0.05` and `m` equals the number of comparisons (2, in our case). Therefore, the new threshold p-value would equal `0.02532057`. To summarize, if we wished to adjust p-values, ourselves, we could obtain un-adjusted p-values from `pairs()` and compare these values to a critical p-value of 0.02532057, rather than 0.05, to determine whether a simulated experiment is 'successful', or not.

I remind the reader that making conclusions by comparing a p-value versus a threshold p-value is not recommended (e.g., see https://www.nature.com/articles/d41586-019-00857-9). In the context of a power-analysis, I view 'threshold' p-values as indicating at least 'moderate' evidence for an effect, rather than 'statistical significance'.

Full code using `emmeans` to implement comparisons:

Here is the full code to implement a power analysis for our experiment, using `emmeans`. Note that the code implements the suggestions, above, as an example of controlling p-values when we make only two (not three) comparisons. Once again, we might do this if we're of a mindset to control for multiple comparisons, generally.

```
#Our parameters:
n.sims <- 10000
control.mean <- 10
treat1.mean <- 9.5
treat2.mean <- 10.5
sd.within <- 0.6233746
n.per.group <- 10

#Our vectors:
anova.vector.p <- c()
p.1vc.vector <- c()
p.2vc.vector <- c()

#Open emmeans library:
library(emmeans)

for(i in 1:n.sims){
        Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
        y.var <- c(rnorm(n.per.group,control.mean,sd.within),
                   rnorm(n.per.group,treat1.mean,sd.within),
                   rnorm(n.per.group,treat2.mean,sd.within))

        sim.data<-data.frame(Experimental.Groups,y.var)
        #Perform GLM and obtain p-value:
        sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
        p <- anova(sim.lm)[1,5]
        anova.vector.p <- append(anova.vector.p,p)
        #Perform pairwise comparisons and obtain p-values:
        sim.emmeans <- emmeans(sim.lm, "Experimental.Groups")
        p.1vc <- summary(pairs(sim.emmeans), adjust = "none")[1,6]
        p.1vc.vector <- append(p.1vc.vector,p.1vc)
        p.2vc <- summary(pairs(sim.emmeans), adjust = "none")[2,6]
        p.2vc.vector <- append(p.2vc.vector,p.2vc)
}
#Power for overall result from general linear model:
overall.small.p <- length(which(anova.vector.p < 0.05))
overall.small.p/n.sims
```

```
## [1] 0.8684
```

```
#Power for comparison between Treat1 vs. Control; NOTICE Dunn-Sidak p-value:
overall.1vc <- length(which(p.1vc.vector<0.02532057))
overall.1vc/n.sims
```

```
## [1] 0.2967
```

```
#Power for comparison between Treat2 vs. Control; NOTICE Dunn-Sidak p-value:
overall.2vc <- length(which(p.2vc.vector<0.02532057))
overall.2vc/n.sims
```

```
## [1] 0.3004
```

Notice that the power declines when we use a more stringent criterion for a 'successful' experiment (i.e., threshold `p = 0.02532057` rather than 0.05).

Power analysis based on precision of effect size estimates, rather than p-values.

In a previous file and video related to power analysis for t-tests, we discussed that we needn't define a 'successful' experiment in terms of `p < 0.05`. As one alternative, we suggested that an experiment might, instead, aim to estimate an effect size with a specified level of precision. In this case, we defined a 'successful' experiment as one that obtained a standard error for an effect size estimate that was equal to, or or less than, a threshold value. In other words, we set an upper-limit for the SE of an effect size estimate (i.e., this upper limit for SE represents a minimum level of precision), and define an experiment as 'successful' if the SE for the effect size estimate is less than the upper limit (i.e., more precise). We will implement this approach, here.

For example, we established two effect sizes in our experiment: `Treat1` vs. `Control` and `Treat2` vs. `Control`. In both cases we set the effect size to equal 1.0. Imagine that we wished to estimate these effect sizes so that the 95% CI's spanned, **at most**, `(mean effect size - 0.6)` to `(mean effect size + 0.6)`. In other words, we aimed to have a maximum 95% CI for our effect size of +/- 0.6. *(We will not discuss how one might settle upon this particular magnitude of 95% CI's; this decision should reflect the researcher's particular goals.)* Recall that we can approximate a 95% CI as `1.96*SE`. (This approximation works best for large sample sizes; for small sample sizes, as we have in this file, we should replace the value '1.96' with an appropriate value from the t-distribution; see videos on 95% Confidence Intervals. For simplicity, we skip this detail in this example, noting that, because we skipped this detail, our results in this file will not be as accurate as we'd like.). **Therefore, in this example, we can define a 'successful' experiment as one in which the SE's for the effect sizes are equal to or less than 0.3 (roughly half of 0.6).**

To implement this approach, we simply obtain SE's for relevant effect sizes, rather than p-values. The output from `pairs()` provides these SE's:

```
sim.emmeans <- emmeans(sim.lm, "Experimental.Groups")
summary(pairs(sim.emmeans))
```

```
##  contrast          estimate    SE df t.ratio p.value
##  Control - Treat1    0.266 0.285 27  0.933  0.6247
##  Control - Treat2   -0.364 0.285 27 -1.276  0.4204
##  Treat1 - Treat2    -0.630 0.285 27 -2.209  0.0877
##
## P value adjustment: tukey method for comparing a family of 3 estimates
```

We can obtain the SE's in a similar manner as we obtained p-values, above:

```
#SE for Control vs. Treat1:
summary(pairs(sim.emmeans))[1,3]
```

```
## [1] 0.2852211
```

```
#SE for Control vs. Treat2:
summary(pairs(sim.emmeans))[2,3]
```

```
## [1] 0.2852211
```

We would compare these SE values vs. our desired, maximum SE value (0.3, in this example) to determine whether a simulation yielded a 'successful' experiment.

Here, we provide the full code to implement this approach. Note that this code changes the vector names to reflect the fact that they store SE values, rather than p-values.

```
#Our parameters:
n.sims <- 10000
control.mean <- 10
treat1.mean <- 9.5
treat2.mean <- 10.5
sd.within <- 0.6233746
n.per.group <- 10

#Our vectors:
anova.vector.p <- c()
se.1vc.vector <- c()
se.2vc.vector <- c()

#Open emmeans library:
library(emmeans)

for(i in 1:n.sims){
        Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
        y.var <- c(rnorm(n.per.group,control.mean,sd.within),
                   rnorm(n.per.group,treat1.mean,sd.within),
                   rnorm(n.per.group,treat2.mean,sd.within))

        sim.data<-data.frame(Experimental.Groups,y.var)
        #Perform GLM and obtain p-value:
        sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
        p <- anova(sim.lm)[1,5]
        anova.vector.p <- append(anova.vector.p,p)
        #Perform pairwise comparisons and obtain p-values:
        sim.emmeans <- emmeans(sim.lm, "Experimental.Groups")
        se.1vc <- summary(pairs(sim.emmeans))[1,3]
        se.1vc.vector <- append(se.1vc.vector,se.1vc)
        se.2vc <- summary(pairs(sim.emmeans))[2,3]
        se.2vc.vector <- append(se.2vc.vector,se.2vc)
}
#Power for overall result from general linear model:
overall.small.p <- length(which(anova.vector.p < 0.05))
overall.small.p/n.sims
```

```
## [1] 0.8677
```

```
#Power for comparison between Treat1 vs. Control; NOTICE use of threshold SE:
overall.1vc <- length(which(se.1vc.vector<0.3))
overall.1vc/n.sims
```

```
## [1] 0.7455
```

```
#Power for comparison between Treat2 vs. Control; NOTICE use of threshold SE:
overall.2vc <- length(which(se.2vc.vector<0.3))
overall.2vc/n.sims
```

```
## [1] 0.7455
```

Please note that this code uses two separate criteria to calculate power:

- `p < 0.05` to determine power for the overall result from the general linear model (`lm()`)
- `SE < 0.3` for estimates of effect size.

Notice that our power, based upon SE for effect size, is very different from what we found when using `p < 0.05` for post-hoc comparisons. Our power would have been smaller if we'd required a more precise experiment.

The (non-) influence of effect size on power, when using SE of effect size to determine 'success'.

We previously demonstrated that, in the context of t-tests, effect size did not affect power when we designed experiments to measure an effect size with a minimum level of precision. Is this also true for 1-Factor GLM? Let's find out.
e Here, we use identical code to that used directly, above, except I changed the differences between `Control` vs. `Treat1` and `Treat2` from 1 unit to 1000 units. What happens?

```
#Our parameters:
n.sims <- 10000
control.mean <- 10
treat1.mean <- -990
treat2.mean <- 1010
sd.within <- 0.6233746
n.per.group <- 10

#Our vectors:
anova.vector.p <- c()
se.1vc.vector <- c()
se.2vc.vector <- c()

#Open emmeans library:
library(emmeans)

for(i in 1:n.sims){
        Experimental.Groups <- c(rep("Control",n.per.group), rep("Treat1",n.per.group),
                         rep("Treat2",n.per.group))
        y.var <- c(rnorm(n.per.group,control.mean,sd.within),
                   rnorm(n.per.group,treat1.mean,sd.within),
                   rnorm(n.per.group,treat2.mean,sd.within))

        sim.data<-data.frame(Experimental.Groups,y.var)
        #Perform GLM and obtain p-value:
        sim.lm <- lm(y.var ~ factor(Experimental.Groups), data=sim.data)
        p <- anova(sim.lm)[1,5]
        anova.vector.p <- append(anova.vector.p,p)
        #Perform pairwise comparisons and obtain p-values:
        sim.emmeans <- emmeans(sim.lm, "Experimental.Groups")
        se.1vc <- summary(pairs(sim.emmeans))[1,3]
        se.1vc.vector <- append(se.1vc.vector,se.1vc)
        se.2vc <- summary(pairs(sim.emmeans))[2,3]
        se.2vc.vector <- append(se.2vc.vector,se.2vc)
}
#Power for overall result from general linear model:
overall.small.p <- length(which(anova.vector.p < 0.05))
overall.small.p/n.sims
```

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

```
#Power for comparison between Treat1 vs. Control; NOTICE use of threshold SE:
overall.1vc <- length(which(se.1vc.vector<0.3))
overall.1vc/n.sims
```

```
## [1] 0.743
```

```
#Power for comparison between Treat2 vs. Control; NOTICE use of threshold SE:
overall.2vc <- length(which(se.2vc.vector<0.3))
overall.2vc/n.sims
```

```
## [1] 0.743
```

Notice what happened when we drastically changed the effect size. On one hand, the larger effect size greatly increased power for the overall result from `lm()`, where we assessed power via `p < 0.05`; now, power is essentially 100%. On the other hand, changing the effect size had no noticeable influence on power when we were interested in the precision to which we can estimate an effect size. This observation is very useful. **This implies that, if we design an experiment with the goal to estimate an effect size with a minimum level of precision, and we analyze the data with a general linear model (e.g., `lm()`), we do not need to know a reasonable effect size a priori to conduct the power analysis.**