Power Analysis by Simulation using R, simglm, and Shiny

Brandon LeBeau[1]

[1] University of Iowa

Author Note

Department of Psychological and Quantitative Foundations

Correspondence concerning this article should be addressed to Brandon LeBeau, Postal address. E-mail: brandon-lebeau@uiowa.edu

Abstract

Keywords: power; simulation; R; simglm

Word count: X

Power Analysis by Simulation using R, simglm, and Shiny

Statistical power is the probability that a statistical analysis is able to detect a non-zero population effect for a binary hypothesis test. In probability terms, statistical power reflects the probability of correctly rejecting the null hypothesis when it is false in the population or mathematically as: $power = P(reject\ H_0 \mid H_1\ is\ true)$. The inverse of power is the probability of making a type II error or the false negative rate.

Power analyses can take on two forms, a priori power analyses occur prior to collecting data and post hoc power analyses occur after data analysis. Post hoc power analyses are controversial, can be misleading given the analysis has already been done, and can give inaccurate results when calculating power using sample data. Therefore, this paper will focus on a priori power analyses, however the process described in this paper could be applied to post hoc power analyses.

Factors affecting power

There are numerous known factors that can impact power in a given analysis. The most well known include sample size, the alpha rate or false positive rate, and the magnitude of the effect size. Other factors such as statistical design, statistical analysis, missing data, or whether statistical assumptions have been met.

Traditional procedures for power

Power is often evaluated using closed form solutions that assume statistical assumptions hold. For example, residuals for many statistical analyses are assumed to follow a normal distribution.

Statistical software has been developed to estimate power when these statistical assumptions have been made such as G*Power, PowerUp!, or Optimal Design. There are also packages in statistical software programs such as pwr, WebPower, or stats in R or statsmodels in Python that estimate power for relatively simple statistical analyses such as t-tests, analysis of variance (ANOVA), linear regression, correlation, or general linear models. G*Power implements power for similar analyses as those found in traditional statistcal software implementations, but offers a graphical user interface (GUI) that may aid users in the power estimation. Finally, the specialized software, PowerUp! and Optimal Design are commonly used to estimate power for randomized control trials and when there are nesting effects that are common in educational or pscyhological research.

Power can be estimated to explore what the probability is given a specific effect size. The minimum detectable effect size given specific sample sizes can also be estimated for power analyses. The latter is often estimated when writing grant applications where the minimum detectable effect size is estimated for specified power levels. Regardless of the framework, it needs to be articulated whether the effect size is of substantive interest and large enough to have a meaningful effect in the population.

Power by Simulation

Power by simulation differs from the traditional approaches in that it is able to be flexible to evaluate the impact on power when statistical assumptions are not met or under different missing data mechanisms which may better reflect realistic data collection conditions.

The following are genearl steps that are taken in a power by simulation example.

1. Assume population parameters, including
    - population effect size(s) of interest.
    - distribution of variable(s) and residuals.
    - variance of variable(s) and residuals.
2. Simulate data based on a statistical model.
3. Fit a statistical model to the simulated data.

4. Replicate steps 1 - 3 many times.
5. Calculate the proportion of statistical tests that appropriate reject the null hypothesis.

In the power by simulation framework, data are simulated based on assumed values for the population, for example the population mean difference between two groups. Given that these values are true, data are simulated many times, replicated, and a statistical model is fitted to the simulated data. To estimate power in this framework, the number of statistical tests that properly reject the null hypothesis compared to the number of replications

Benefits of Power by Simulation

Power by simulation can be used for any statistical design or statistical analysis. The limiting factor is the ability of the researcher to use statistical software to follow the steps outlined above. Power by simulation can allow researchers to relax statistical assumptions that may have an impact on statistical power and more readily mirror real data conditions. If statistical assumptions do not hold in the population, the power analysis that makes these assumptions will commonly overestimate power. This can have important considerations and implications for researchers, funders, or other relavent stakeholders that are invested in the research idea.

Furthermore, as power by simulation is not limited by the software and the steps for employing a power by simulation analysis are the same regardless of the statistical design or analysis, once the process of simluation by power are well understood, the only major change across different statistical designs and analysis are the generating and fitted models. In some cases, getting estimates of population effects may be more challenging as the model complexity increases, but this can be a limiting factor of traditional power analyses as well.

Two-Sample t-test Example

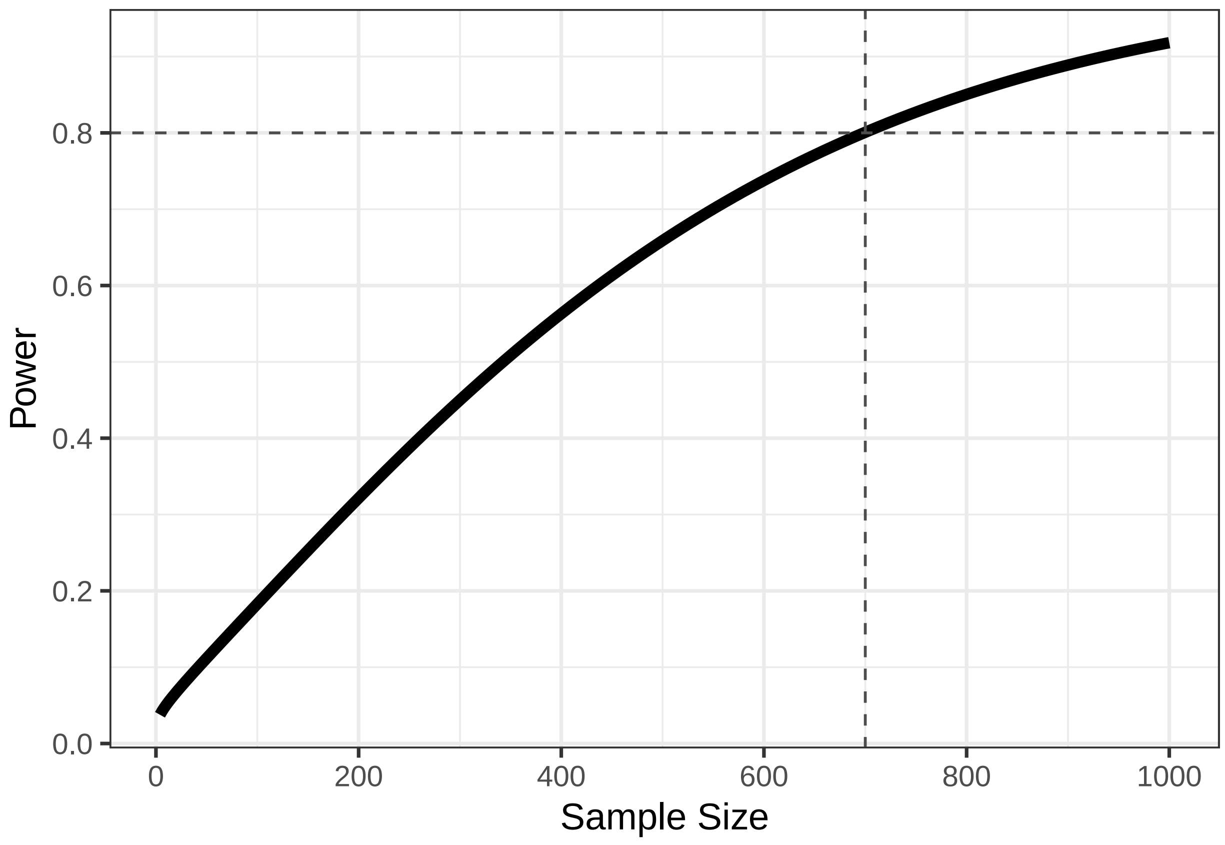
Power for a two-sample design using the t-test can be done with the following code for a single effect size, standardized mean difference of 0.15. This example generates power for sample sizes ranging from 4 up to 1000 increasing by intervals of 2. This will generate power values for 499 sample sizes.

```r
library(tidyverse)
library(simglm)
library(future.apply)
library(lme4)
```

```r
n <- seq(4, 1000, 2)
power <- sapply(seq_along(n), function(i)
  power.t.test(n = n[i], delta = .15, sd = 1, type = 'two.sample')$power)
```

The power for the first iteration can be extracted directly using `power[1]` 0.04, but generally showing a figure would be more interesting.

```r
power_df <- data.frame(
  n = n,
  power = power
)

ggplot(power_df, aes(x = n, y = power)) +
  geom_line(size = 2) +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  geom_vline(xintercept = 700, linetype = 2, color = 'gray30') +
  scale_x_continuous("Sample Size", breaks = seq(0, 1000, 200)) +
  scale_y_continuous("Power", breaks = seq(0, 1, .2)) +
  theme_bw(base_size = 14)
```



Power Curves

It is common for the effect size of interest to not be completely certain a priori when many power analyses are conducted. In these cases, a form of sensitivity analysis or descriptive power analyses are often conducted that vary the effect size as well.

A similar structure can be done to add the power curves for different effect sizes; the primary differences being the addition of different effect sizes in addition to the different sample size conditions.

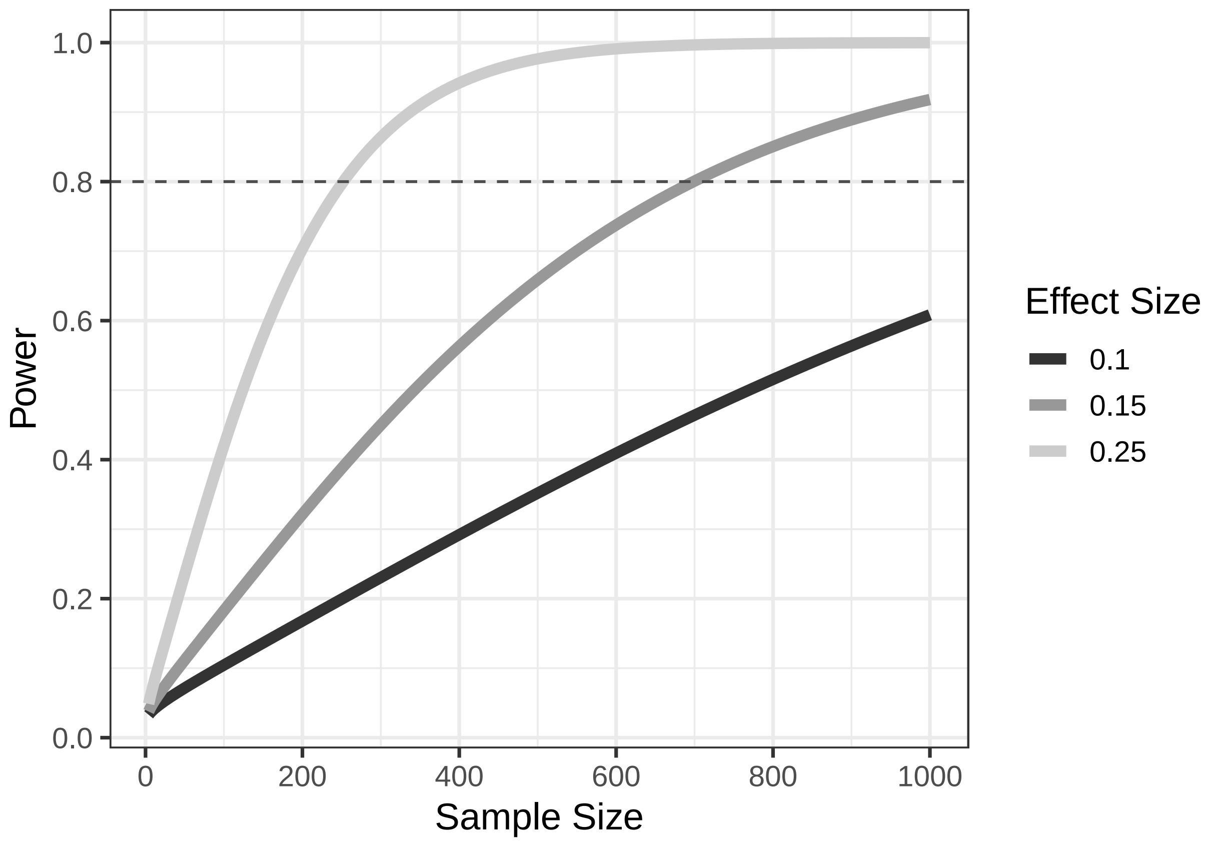
```r
effect_sizes <- c(.10, .15, .25)
conditions <- expand.grid(n = n, effect_sizes = effect_sizes)
head(conditions)
```

```
##    n effect_sizes
## 1  4          0.1
## 2  6          0.1
## 3  8          0.1
## 4 10          0.1
## 5 12          0.1
## 6 14          0.1
```

```
power_curve <- sapply(seq_len(nrow(conditions)), function(i)
  power.t.test(n = conditions[i, 'n'],
               delta = conditions[i, 'effect_sizes'],
               sd = 1, type = 'two.sample')$power)
```

These can then be visualized after converting to a data frame and combining with the original conditions object.

```
power_curve_df <- bind_cols(
  conditions,
  power = power_curve
)

ggplot(power_curve_df, aes(x = n, y = power)) +
  geom_line(aes(color = factor(effect_sizes)), size = 2) +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  #geom_vline(xintercept = 700, linetype = 2, color = 'gray30') +
  scale_x_continuous("Sample Size", breaks = seq(0, 1000, 200)) +
  scale_y_continuous("Power", breaks = seq(0, 1, .2)) +
  scale_color_grey("Effect Size") +
  theme_bw(base_size = 14)
```



Two sample Data simulation with `simglm`

The same two sample power analysis can be conducted by simulation with the `simglm` R package (LeBeau, 2019). This package simulates data based on general(-ized) linear (mixed) models. The two sample t-test shown above can be thought of as a linear model as well with a single indicator variable that specifies which group each data point belongs to.

```r
simulation_arguments <- list(
  formula = y ~ 1 + sex,
  fixed = list(sex = list(var_type = 'factor',
                          levels = c('male', 'female'))),
  sample_size = 20,
  error = list(variance = 1),
  reg_weights = c(0, .15)
)

simulate_fixed(data = NULL, simulation_arguments) %>%
  simulate_error(simulation_arguments) %>%
  generate_response(simulation_arguments)
```

```
##    X.Intercept. sex    sex1 level1_id      error fixed_outcome
## 1             1   1    male         1  2.2866454          0.15
## 2             1   1    male         2 -1.3888607          0.15
## 3             1   0  female         3 -0.2787888          0.00
## 4             1   0  female         4 -0.1333213          0.00
## 5             1   0  female         5  0.6359504          0.00
## 6             1   0  female         6 -0.2842529          0.00
## 7             1   1    male         7 -2.6564554          0.15
## 8             1   0  female         8 -2.4404669          0.00
## 9             1   1    male         9  1.3201133          0.15
## 10            1   0  female        10 -0.3066386          0.00
## 11            1   1    male        11 -1.7813084          0.15
## 12            1   0  female        12 -0.1719174          0.00
## 13            1   1    male        13  1.2146747          0.15
## 14            1   1    male        14  1.8951935          0.15
## 15            1   0  female        15 -0.4304691          0.00
## 16            1   0  female        16 -0.2572694          0.00
## 17            1   0  female        17 -1.7631631          0.00
## 18            1   0  female        18  0.4600974          0.00
## 19            1   1    male        19 -0.6399949          0.15
## 20            1   1    male        20  0.4554501          0.15
##    random_effects          y
## 1               0  2.4366454
## 2               0 -1.2388607
## 3               0 -0.2787888
## 4               0 -0.1333213
## 5               0  0.6359504
## 6               0 -0.2842529
## 7               0 -2.5064554
## 8               0 -2.4404669
## 9               0  1.4701133
## 10              0 -0.3066386
## 11              0 -1.6313084
## 12              0 -0.1719174
## 13              0  1.3646747
## 14              0  2.0451935
## 15              0 -0.4304691
## 16              0 -0.2572694
## 17              0 -1.7631631
## 18              0  0.4600974
## 19              0 -0.4899949
## 20              0  0.6054501
```

Using `simglm` to generate power

```
plan(multiprocess)


simulation_arguments <- list(
  formula = y ~ 1 + sex,
```

```r
  fixed = list(sex = list(var_type = 'factor',
                          levels = c('male', 'female'))),
  sample_size = 20,
  error = list(variance = 1),
  reg_weights = c(0, .15),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + sex,
                   model_function = 'lm'),
  extract_coefficients = TRUE
)

replicate_sim <- replicate_simulation(simulation_arguments)
```

```r
replicate_sim %>%
  compute_statistics(simulation_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE)
```

```
## # A tibble: 2 x 8
##   term  avg_estimate power avg_test_stat param_estimate_~ avg_standard_er~
##   <chr>        <dbl> <dbl>         <dbl>            <dbl>            <dbl>
## 1 (Int~       0.0114 0.082        0.0351            0.329            0.324
## 2 sex         0.132  0.077        0.302             0.461            0.457
## # ... with 2 more variables: precision_ratio <dbl>, replications <dbl>
```

```r
simulation_arguments <- list(
  formula = y ~ 1 + sex,
  fixed = list(sex = list(var_type = 'factor',
                          levels = c('male', 'female'))),
  error = list(variance = 1),
  reg_weights = c(0, .15),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + sex,
                   model_function = 'lm'),
  power = list(
    dist = 'qnorm',
    alpha = .05
  ),
  extract_coefficients = TRUE,
  vary_arguments = list(
    sample_size = seq(20, 2000, 20)
  )
)

model_results <- replicate_simulation(simulation_arguments)
```

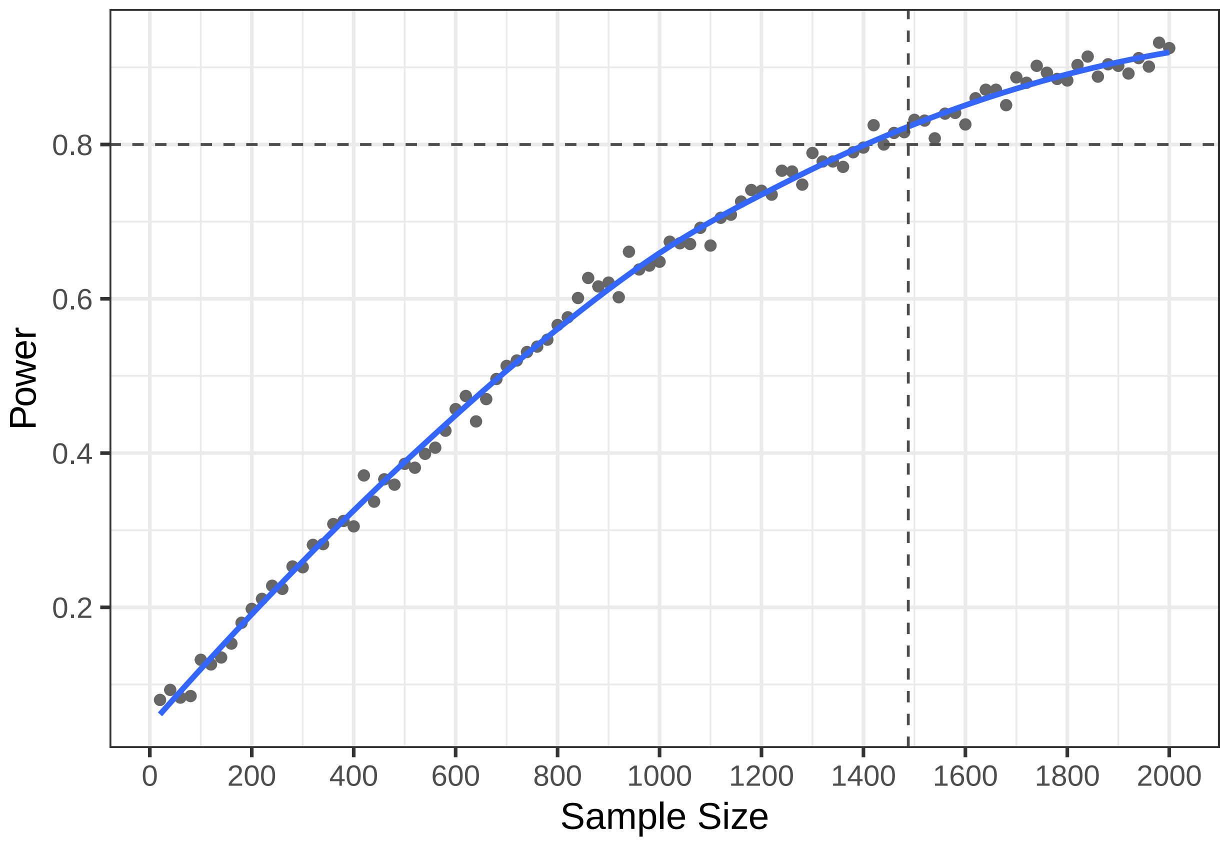
```
power_results <- model_results %>%
  compute_statistics(simulation_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE)
```

```
power_results <- power_results %>%
  ungroup() %>%
  mutate(sample_size = as.numeric(as.character(sample_size))) %>%
  arrange(sample_size) %>%
  filter(term == 'sex')
head(power_results, n = 10)
```

```
## # A tibble: 10 x 9
##    sample_size term  avg_estimate power avg_test_stat param_estimate_~
##          <dbl> <chr>        <dbl> <dbl>         <dbl>            <dbl>
##  1          20 sex          0.158 0.08          0.350            0.449
##  2          40 sex          0.168 0.093         0.542            0.323
##  3          60 sex          0.153 0.083         0.595            0.260
##  4          80 sex          0.146 0.085         0.652            0.217
##  5         100 sex          0.151 0.132         0.759            0.204
##  6         120 sex          0.141 0.126         0.772            0.187
##  7         140 sex          0.144 0.135         0.862            0.166
##  8         160 sex          0.148 0.153         0.939            0.152
##  9         180 sex          0.153 0.18          1.03             0.146
## 10         200 sex          0.154 0.198         1.10             0.141
## # ... with 3 more variables: avg_standard_error <dbl>,
## #   precision_ratio <dbl>, replications <dbl>
```

This result can then be explored visually to generate a power curve.

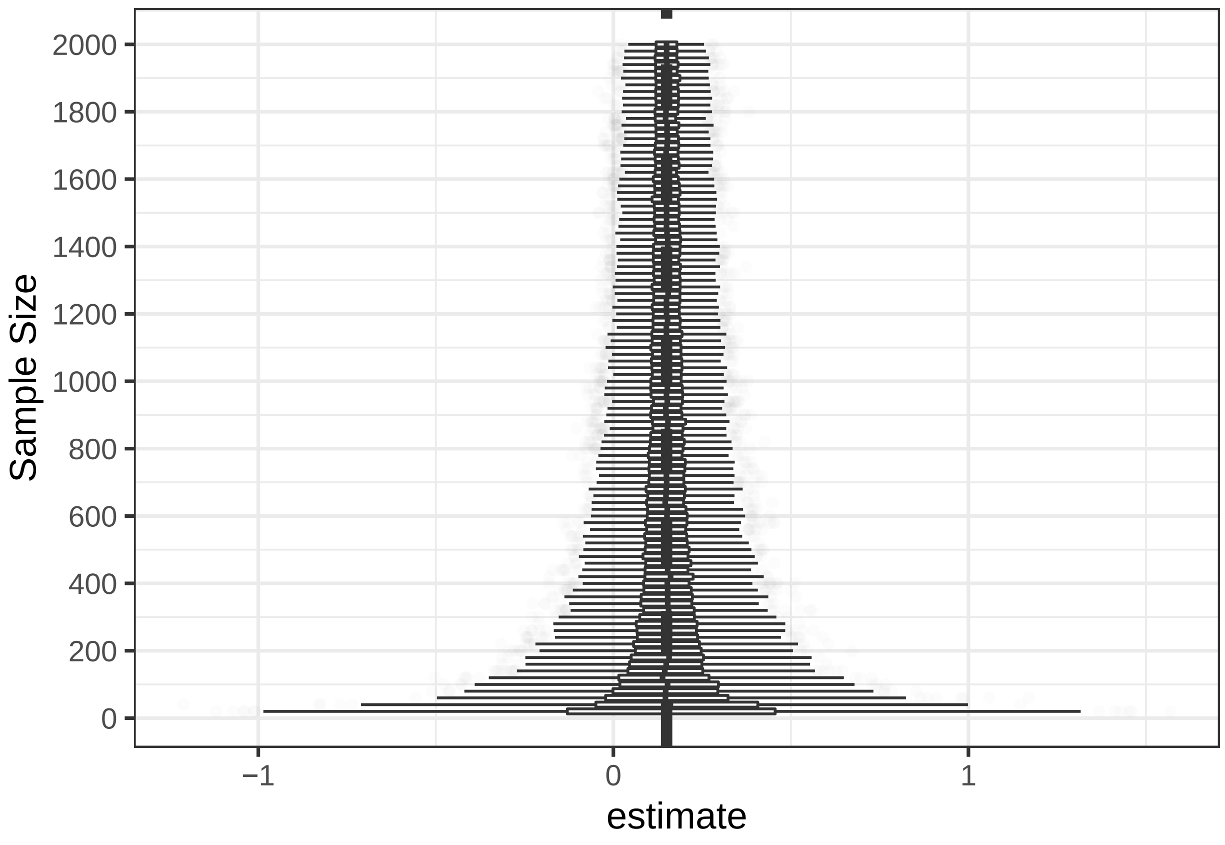
```r
ggplot(power_results, aes(x = sample_size, y = power)) +
  geom_point(size = 1.5, color = 'gray40') +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  geom_smooth(linetype = 1, size = 1, se = FALSE) +
  geom_vline(xintercept = 1488, linetype = 2, color = 'gray30') +
  scale_x_continuous("Sample Size", breaks = seq(0, 2000, 200)) +
  scale_y_continuous("Power", breaks = seq(0, 1, .2)) +
  theme_bw(base_size = 14)
```



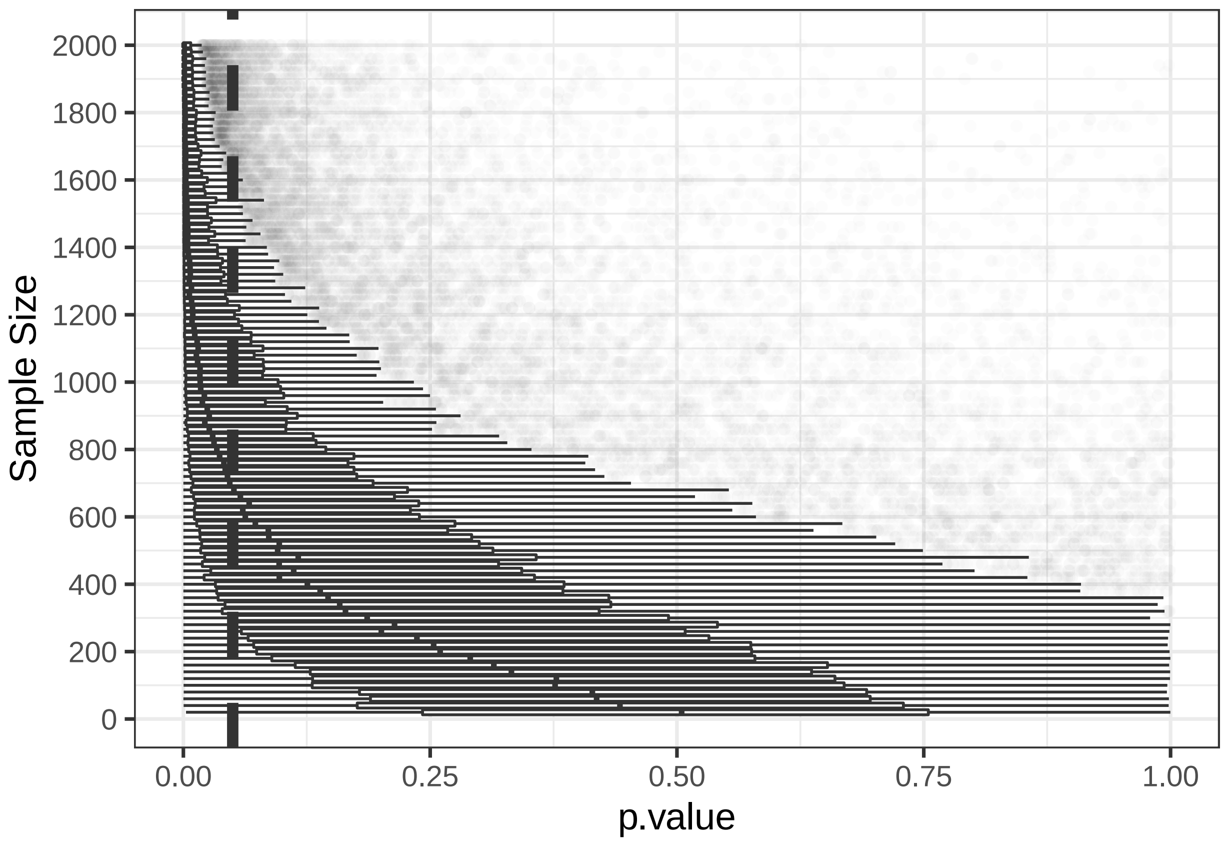
Visualizing estimates and p-values

Exploring the estimates and the p-values can be informative over and above the power graphics shown above. The model results are stored in the object, `model_results` shown above as a list for each replicated condition. This means that for each sample size, there are 1000 estimates for the group effect and the associated p-values based on these.

```
model_results_df <- bind_rows(model_results) %>%
  filter(term == 'sex') %>%
  mutate(sample_size = as.numeric(as.character(sample_size)))

ggplot(model_results_df, aes(x = sample_size, y = estimate)) +
  geom_boxplot(aes(group = sample_size), outlier.alpha = 0.01) +
  scale_x_continuous("Sample Size", breaks = seq(0, 2000, 200)) +
  theme_bw(base_size = 14) +
  geom_hline(yintercept = .15, linetype = 2, size = 2, color = 'gray20') +
  coord_flip()
```



```
ggplot(model_results_df, aes(x = sample_size, y = p.value)) +
  geom_boxplot(aes(group = sample_size), outlier.alpha = 0.01) +
  scale_x_continuous("Sample Size", breaks = seq(0, 2000, 200)) +
  theme_bw(base_size = 14) +
  geom_hline(yintercept = .05, linetype = 2, size = 2, color = 'gray20') +
  coord_flip()
```



Add Heterogeneity

Heterogeneity is a common phenomenon that can impact power and can occur when there are population variance differences across the two groups. In the above example, this would mean that there would be variance differences across males and females, perhaps males are more variable in the outcome. The `pwr.t.test()` function from before does not assume heterogeneity, however this is possible to incorporate when doing power from a simulation framework. Adding these conditions can help to mimic real world conditions and also maybe provide a better estimate of power.

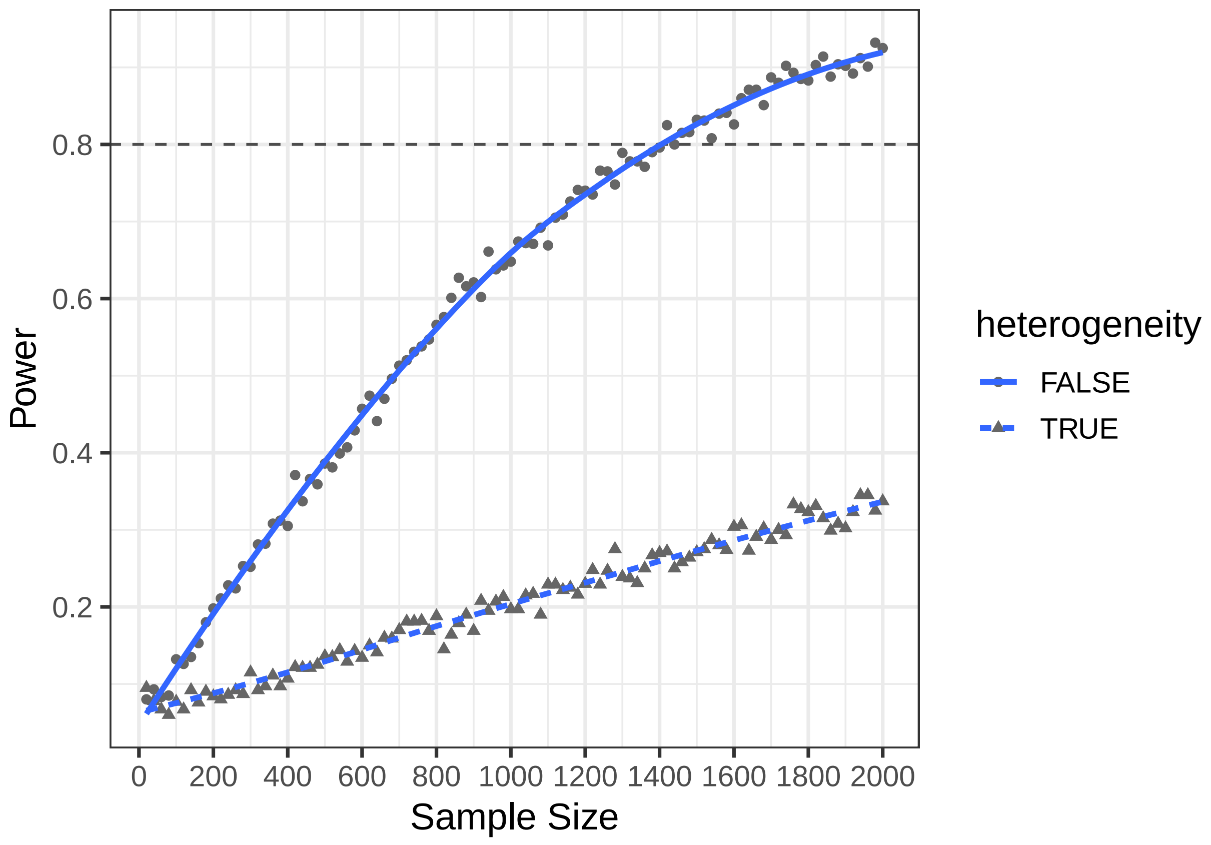
```
simulation_arguments <- list(
  formula = y ~ 1 + group,
  fixed = list(group = list(var_type = 'factor',
                            levels = c('male', 'female'))),
```

```r
  error = list(variance = 1),
  heterogeneity = list(variable = 'group',
                       variance = c(1, 8)),
  reg_weights = c(0, .15),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + group,
                   model_function = 'lm'),
  power = list(
    dist = 'qnorm',
    alpha = .05
  ),
  extract_coefficients = TRUE,
  vary_arguments = list(
    sample_size = seq(20, 2000, 20)
  )
)


model_results_h <- replicate_simulation(simulation_arguments)
```

```r
power_results_h <- model_results_h %>%
  compute_statistics(simulation_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE) %>%
  ungroup() %>%
  mutate(sample_size = as.numeric(as.character(sample_size)),
         heterogeneity = TRUE) %>%
  arrange(sample_size) %>%
  filter(term == 'group')
```

```r
power_results <- power_results %>%
  mutate(heterogeneity = FALSE)

power_results_combined <- bind_rows(power_results,
                                    power_results_h)
```

```r
ggplot(power_results_combined, aes(x = sample_size, y = power,
                                 group = heterogeneity)) +
  geom_point(aes(shape = heterogeneity), size = 1.5, color = 'gray40') +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  geom_smooth(aes(linetype = heterogeneity), size = 1, se = FALSE) +
  #geom_vline(xintercept = 1488, linetype = 2, color = 'gray30') +
  scale_x_continuous("Sample Size", breaks = seq(0, 2000, 200)) +
  scale_y_continuous("Power", breaks = seq(0, 1, .2)) +
  theme_bw(base_size = 14)
```



Repeated Measures Example

For more complicated designs, closed form solutions are not always possible as ways to estimate statistical power or they make strong assumptions on the data that may not be reasonable given the data that are to be collected. The data simulation for the repeated measures portion adds a hierarchical data structure in which repeated measures are nested within individuals. These type of data add a correlation structure that needs to be modeled appropriately for valid inferences. Modern models to handle this dependency include the linear mixed model (LMM), sometimes known as the hierarchical linear model (HLM) or multilevel model.

This framework is an extension of the linear model considered above in the two group model. The extension includes the addition of random effects which represent subject specific deviations from the average trejectories. These random effects are what account for the dependency due to repeated measurements and nested data structure. In addition to the addition of random effects, two sample sizes need to be given, one representing the number of measurement occasions for each individual and another representing how individuals to generate. Therefore, in the balanced case (i.e. same number of measurement occasions for each individual), the total sample size (i.e. number of rows in the data) will be the number of measurement occasions times the number of individuals. This is a good initial check to ensure that the number of records are correct.

```r
sim_arguments <- list(
  formula = y ~ 1 + time + sex + time:sex + (1 + time | individual),
  reg_weights = c(4, 0.4, 0.3, 0.20),
  error = list(variance = 1),
  fixed = list(time = list(var_type = 'time'),
               sex = list(var_type = 'factor', levels = c('male', 'female'),
                          var_level = 2)),
  randomeffect = list(int_individual = list(variance = 1, var_level = 2),
                      time_individual = list(variance = 0.5, var_level = 2)),
  sample_size = list(level1 = 10, level2 = 500)
)

longitudinal_data <- sim_arguments %>%
  simulate_fixed(data = NULL, .) %>%
  simulate_randomeffect(sim_arguments) %>%
  simulate_error(sim_arguments) %>%
  generate_response(sim_arguments)
```

Power for Repeated Measures

```r
sim_arguments <- list(
  formula = y ~ 1 + time + sex + time:sex + (1 + time | individual),
  reg_weights = c(4, 0.4, 0.3, 0.20),
  error = list(variance = 1),
  fixed = list(time = list(var_type = 'time'),
               sex = list(var_type = 'factor', levels = c('male', 'female'),
                          var_level = 2)),
  randomeffect = list(int_individual = list(variance = 1, var_level = 2),
                      time_individual = list(variance = 0.5, var_level = 2)),
  sample_size = list(level1 = 10, level2 = 500),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + time + sex + time:sex +
                     (1 + time | individual),
                   model_function = 'lmer'),
  extract_coefficients = TRUE
)

long_models <- replicate_simulation(sim_arguments)
```

```r
long_models %>%
  compute_statistics(sim_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE)
```

```
## # A tibble: 4 x 8
##   term  avg_estimate power avg_test_stat param_estimate_~ avg_standard_er~
##   <fct>        <dbl> <dbl>         <dbl>            <dbl>            <dbl>
## 1 (Int~        4.00  1            54.6           0.0749           0.0734
## 2 sex          0.300 0.819         2.89          0.108            0.104
## 3 time         0.403 1             8.94          0.0434           0.0452
## 4 time~        0.199 0.872         3.11          0.0613           0.0639
## # ... with 2 more variables: precision_ratio <dbl>, replications <dbl>
```

```
sim_arguments <- list(
  formula = y ~ 1 + time + sex + time:sex + (1 + time | individual),
  reg_weights = c(4, 0.4, 0.3, 0.20),
  error = list(variance = 1),
  fixed = list(time = list(var_type = 'time'),
               sex = list(var_type = 'factor', levels = c('male', 'female'),
                          var_level = 2)),
  randomeffect = list(int_individual = list(variance = 1, var_level = 2),
                      time_individual = list(variance = 0.5, var_level = 2)),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + time + sex + time:sex +
                     (1 + time | individual),
                   model_function = 'lmer'),
  extract_coefficients = TRUE,
  vary_arguments = list(
    sample_size = list(list(level1 = 5, level2 = 50),
                       list(level1 = 5, level2 = 150),
                       list(level1 = 5, level2 = 250),
                       list(level1 = 8, level2 = 50),
                       list(level1 = 8, level2 = 150),
                       list(level1 = 8, level2 = 250))
  )
)

long_model_nomiss <- replicate_simulation(sim_arguments)
```

```
long_power_nomiss <- long_model_nomiss %>%
  compute_statistics(sim_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE)
```

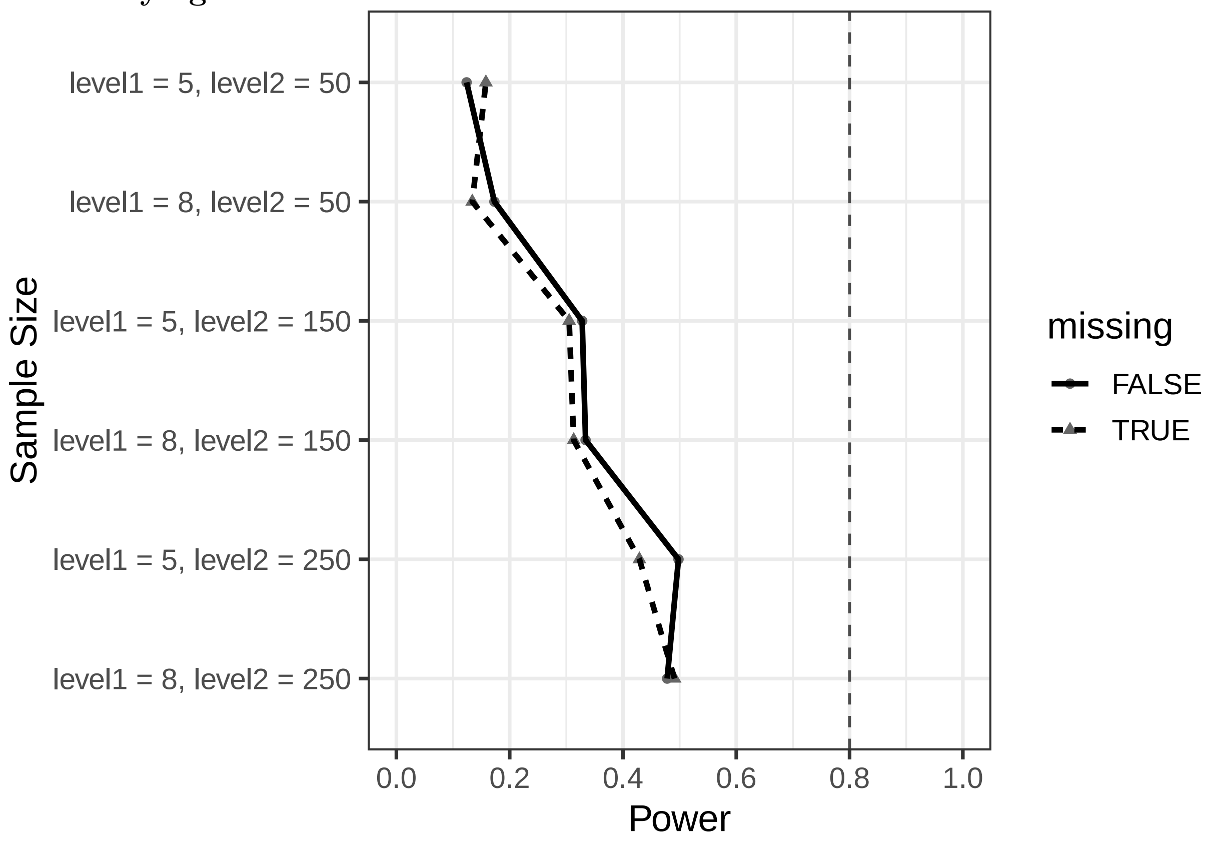
```
sim_arguments <- list(
  formula = y ~ 1 + time + sex + time:sex + (1 + time | individual),
  reg_weights = c(4, 0.4, 0.3, 0.20),
  error = list(variance = 1),
  fixed = list(time = list(var_type = 'time'),
               sex = list(var_type = 'factor', levels = c('male', 'female'),
                          var_level = 2)),
  randomeffect = list(int_individual = list(variance = 1, var_level = 2),
                      time_individual = list(variance = 0.5, var_level = 2)),
  replications = 1000,
  model_fit = list(formula = y_miss ~ 1 + time + sex + time:sex +
                     (1 + time | individual),
                   model_function = 'lmer'),
  missing_data = list(new_outcome = 'y_miss', miss_prop = .20,
                      clust_var = 'individual', type = 'dropout'),
  extract_coefficients = TRUE,
  vary_arguments = list(
    sample_size = list(list(level1 = 5, level2 = 50),
                       list(level1 = 5, level2 = 150),
                       list(level1 = 5, level2 = 250),
                       list(level1 = 8, level2 = 50),
                       list(level1 = 8, level2 = 150),
                       list(level1 = 8, level2 = 250))
  )
)

long_model_miss <- replicate_simulation(sim_arguments)
```

```
long_power_miss <- long_model_miss %>%
  compute_statistics(sim_arguments, power = TRUE,
                     type_1_error = FALSE, precision = TRUE)
```

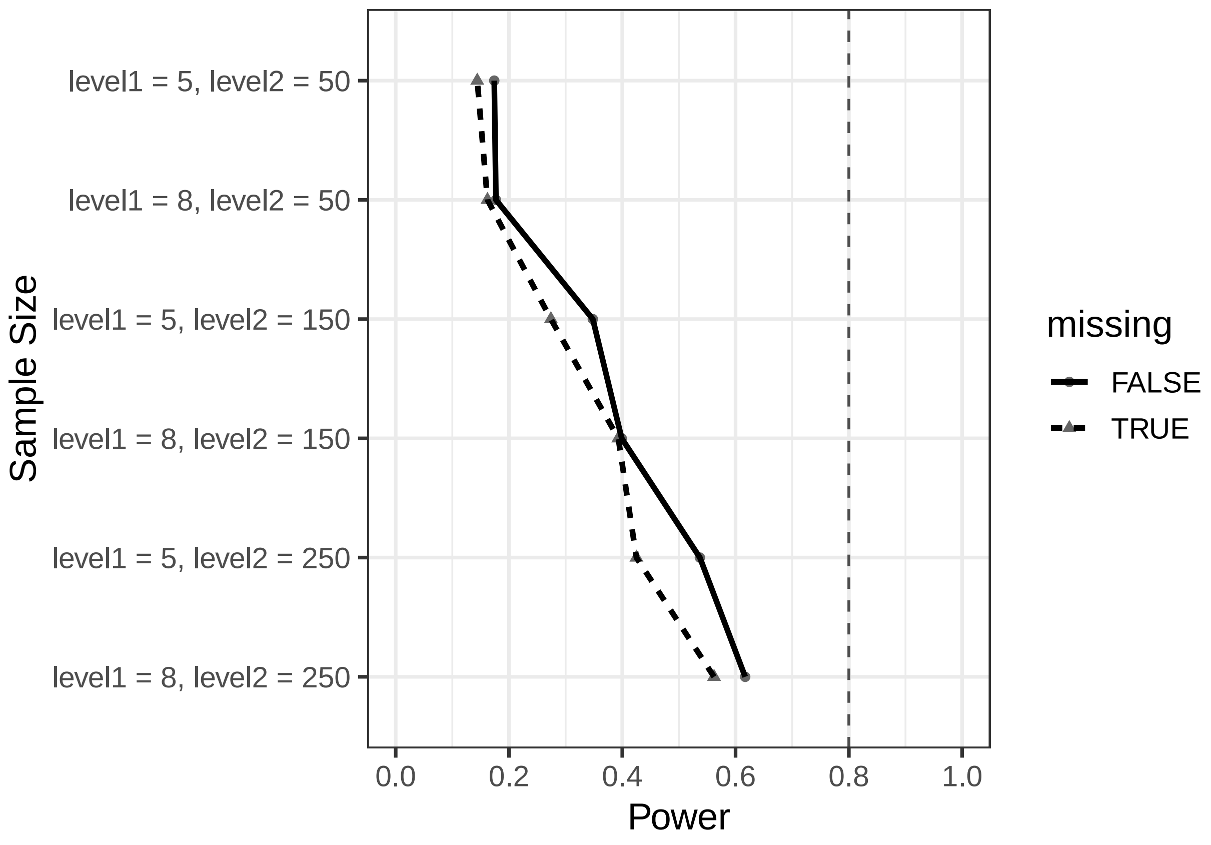
```
long_power_nomiss_h <- long_power_nomiss %>%
  ungroup() %>%
  mutate(sample_size = gsub("^list\\(|\\)$", "", as.character(sample_size)),
         missing = FALSE)

long_power_miss_h <- long_power_miss %>%
  ungroup() %>%
  mutate(sample_size = gsub("^list\\(|\\)$", "", as.character(sample_size)),
         missing = TRUE)

long_power_combined <- bind_rows(long_power_nomiss_h,
                                    long_power_miss_h)
```

```
long_power_sex <- filter(long_power_combined, term == 'sex')

ggplot(long_power_sex, aes(x = reorder(sample_size, desc(power)), y = power,
                           group = missing)) +
  geom_point(aes(shape = missing), size = 1.5, color = 'gray40') +
  geom_line(aes(linetype = missing), size = 1) +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  #geom_smooth(aes(linetype = missing), size = 1, se = FALSE) +
  #geom_vline(xintercept = 1488, linetype = 2, color = 'gray30') +
  xlab("Sample Size") +
  scale_y_continuous("Power", breaks = seq(0, 1, .2), limits = c(0, 1)) +
  theme_bw(base_size = 14) +
  coord_flip()
```

Varying simulation conditions.



```r
long_power_timesex <- filter(long_power_combined, term == 'time:sex')


ggplot(long_power_timesex, aes(x = reorder(sample_size, desc(power)), y = power,
                          group = missing)) +
  geom_point(aes(shape = missing), size = 1.5, color = 'gray40') +
  geom_line(aes(linetype = missing), size = 1) +
  geom_hline(yintercept = 0.8, linetype = 2, color = 'gray30') +
  #geom_smooth(aes(linetype = missing), size = 1, se = FALSE) +
  #geom_vline(xintercept = 1488, linetype = 2, color = 'gray30') +
  xlab("Sample Size") +
  scale_y_continuous("Power", breaks = seq(0, 1, .2), limits = c(0, 1)) +
  theme_bw(base_size = 14) +
  coord_flip()
```


level1 = 5, level2 = 50
level1 = 8, level2 = 50
level1 = 5, level2 = 150
level1 = 8, level2 = 150
level1 = 5, level2 = 250
level1 = 8, level2 = 250
Sample Size
0.0
0.2
0.4
0.6
0.8
1.0
Power
missing
FALSE
TRUE

Summary

References

LeBeau, B. (2019). *Simglm: Simulate models based on the generalized linear model.* Retrieved from https://CRAN.R-project.org/package=simglm