Power Analysis by Simulation using R and simglm

Abstract

Power is a task that is commonly done prior to collecting data for a primary study. In most cases closed-form solutions are used to estimate power which may statistical assumptions to be able to perform the computations, for example assume residuals are normally distributed. In real-world data, these statistical assumptions may not hold, therefore estimates of power when these assumptions are assumed will likely be inflated. Power by simulation is another way to compute power estimates and offers significant flexibility to the user to explore the impact of various statistical assumption violations may have on power. This tutorial uses the `simglm` R package to perform the power by simulation. The simglm package provides a framework to simulate data from generalized linear mixed models which includes a wide variety of models. In addition, functions to perform replications and to compute power estimate summaries are available for users to take advantage of. Two worked examples are shown, one for a two-sample t-test and another within a repeated measures or longitudinal framework.

Keywords: power; simulation; R; simglm

Word count: 4700

Power Analysis by Simulation using R and simglm

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 (Hoenig & Heisey, 2001; Thomas, 1997). 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 (Faul, Erdfelder, Lang, & Buchner, 2007), PowerUp! (Dong, Kelcey, Maynard, & Spybrook, 2015), or Optimal Design (Raudenbush et al., 2011). There are also packages in statistical software programs such as pwr (Champely, 2018), WebPower (Zhang & Mai, 2018), or stats which is a part of base R (R Core Team, 2019) 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.

The tutorial will use the following packages

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

Two-Sample t-test Example

A two-sample t-test is a statistical technique that compares whether two means are statistically different from one another. This analysis is commonly done when two groups are measured on a single outcome and the mean of these two groups are compared to one another. As an example, the males and females may be compared on their average reading score or average depression score after a treatment.

Power for a two-sample design using the t-test can be done with the following code for a single effect size, a 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
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 the code `power[1]`, which returns a value of 0.04. There are numerous power values however and figures are a great way to show all of these values in a relatively succict way. The code below creates a data frame that includes the sample size and subsequent power values then uses the ggplot2 (Wickham, 2016) package to visualize the power estimates. A horizontal line is drawn at a power level of 0.80, a common standard, and a vertical line is drawn when the curve hits this threshold which happens to occur at a sample size of 700. This sample size does not represent the total sample size, rather it represents the sample size for each group, assuming balanced groups and other commonly made statistical assumptions for the two-sample t-test including homogeneity of variance, normally distributed residuals, independent observations, and equal group sizes (the last is not a statistical assumption, but rather an assumption made by the `power.t.test` function).

```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. In these cases, descriptive power analyses are often conducted that vary the effect size as well as the sample size to explore the power estimates for different effect sizes across a variety of sample sizes.

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. These conditions are created using the `expand.grid` function which

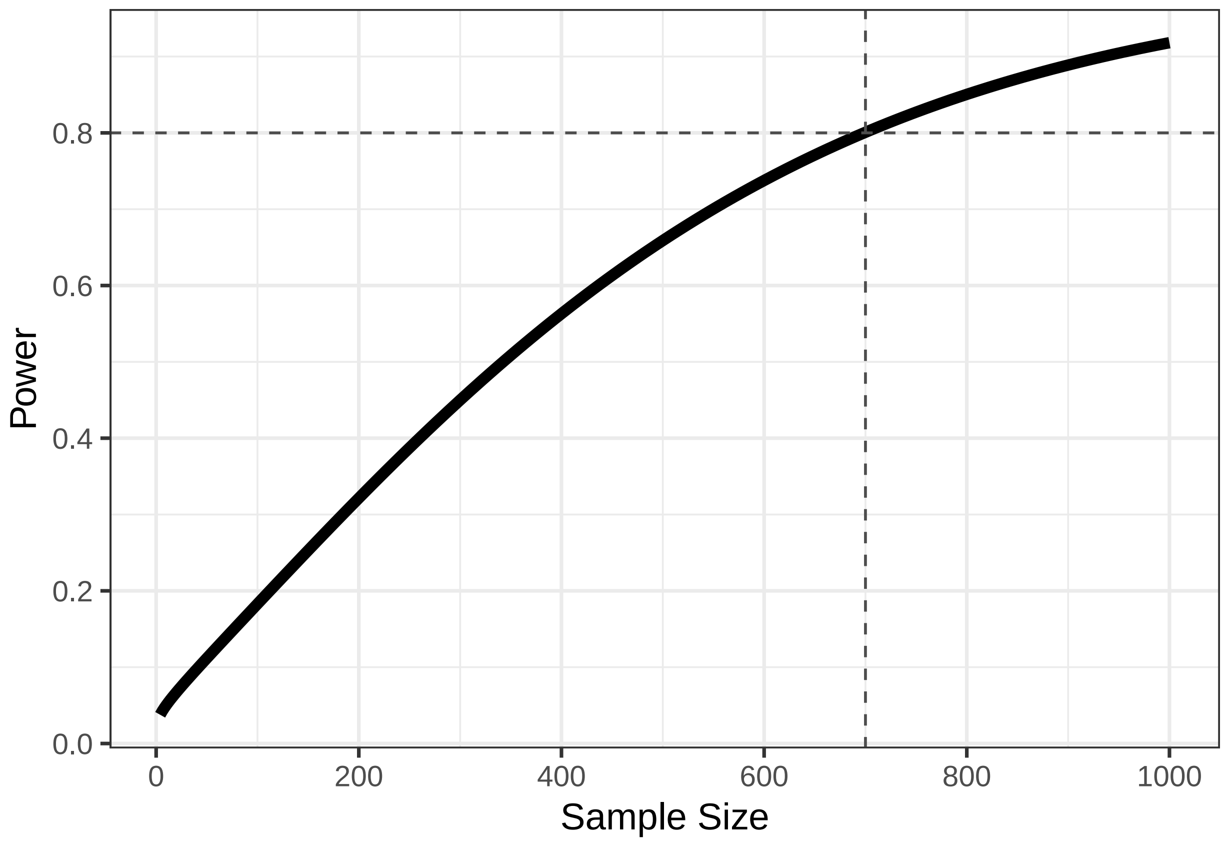


Figure 1. Power estimates for an effect size of 0.15 in the two-sample design.

creates a row for each unique value of the variables specified. These conditions are then passed to the `power.t.test` function one at a time to create power estimates for each of the data conditions.

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

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)
```

A similar process is used to create a data frame with the conditions and power estimates to be visualized. Figure 2 shows the three power curves based on the three effects sizes of 0.10, 0.15, and 0.25. You'll notice that for the effect size of 0.10, estimated power never reaches the desired threshold of 0.80 indicating that larger sample sizes for each group would be needed. In addition, power estimates for an effect size of 0.25 reaches the threshold of 0.80 with around 250 individuals in each group. Having sample sizes much larger than that results in much smaller changes to the estimated power.

```r
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') +
  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 and can replicate the simulation procedure to perform a power analysis. In order to use the `simglm` package, the two-sample t-test described above needs to be reformulated into an equivalent form using the general linear model or more specifically a linear regression model.

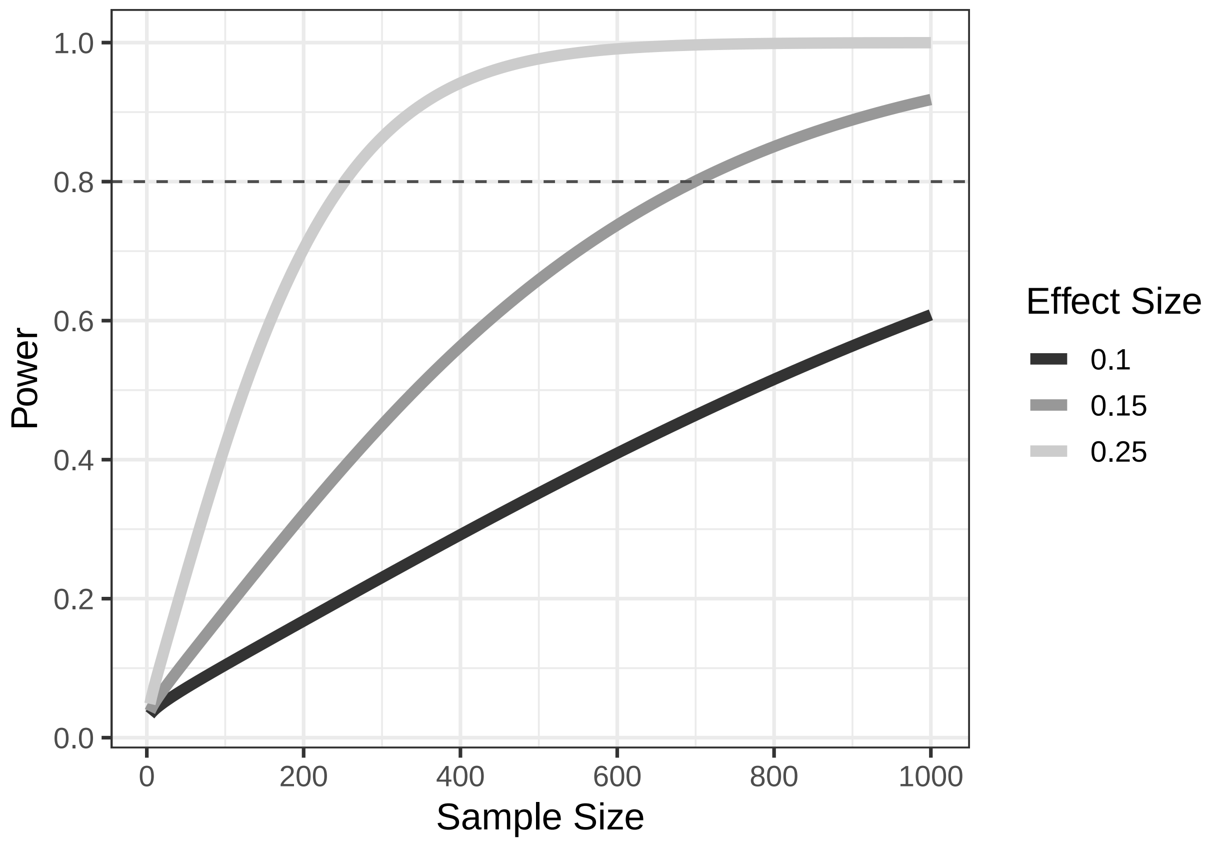


Figure 2. Three power curves representing power estimates across three different effect sizes and various sample sizes.

The linear regression model would look like the following:

$$Y_i = \beta_0 + \beta_1 group_i + \epsilon_i \tag{1}$$

where Y_i is the outcome of interest (i.e. reading or depression scores reference above) for individual i, $group_i$ is a fixed indicator variable that would have a score of 1 if an individual belonged to group 1 otherwise would have a score of 0, and ϵ_i would represent random sampling error. With the linear regression model set up this way, β_0 would represent the intercept and would be interpreted as the mean of the group coded as 0 in the $group_i$ variable. The second coefficient, β_1, would be the variable of interest and represent the mean difference between the two groups. The mean of the second group could be found by taking $\beta_0 + \beta_1$. The primary benefit of approaching the two-sample t-test in this fashion is the model specification flexiblity, in particular if statistical controls or interactions would want to be explored.

The following code is an example of simulating data using the `simglm` package. The code consists of two parts, the first part specifies the parameters for the simulation in a named list, named `simulation_arguments` below. For the two-sample linear regression model shown above the following named elements need to be included, the model formula (which mimics the linear regression formula depicted above without the error), the fixed component representing the group variable (depicted as sex below), the sample size (i.e. how many individuals), the size of random error variance, and the regression weights. The regression weights (named `reg_weights` below) represent the values for β_0 and β_1 discussed above. Exploring the code below, this would mean that the mean for group 1 would be assumed to be 0 and the mean difference between the two groups is 0.15 (equivalently here the mean of group 2 would be $0 + 0.15 = 0.15$). Since the error variance is specified to equal 1, the data generated mimics a standardized example and the interpretation of the second regression weight (i.e. $\beta_1 = 0.15$) would be similar to the effect size cohen's d.

Finally, the fixed argument in some cases will be the most difficult argument to specify and often requires the most code. In this example, only the sex variable needs to be defined how this should be simulated. The code for the fixed portion identifies that this variable should be a factor variable, using the `var_type` argument and the levels of the sex variable (i.e. male and female) are specified. This ensures that the data labels are created directly rather than just numeric values and can be helpful to ensure the creation of the two groups. Internally during the simulation process this variable with character labels would be turned into a dichotomous indicator variable that is defined above in Equation (1).

```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)
)
```

A series of commands are used for the generation of the simulated data, `simulate_fixed`, `simulate_error`, and `generate_response`. In general these function take the simulation arguments defined in the previous code chunk as the primary argument and the `%>%` pipe operator is used to chain the commands together which passes the result from the previous step to the first function argument which for the simglm package is always the data. The `simulate_fixed` function generates the fixed portion of the model which for the model shown in Equation (1) representing the intercept and sex variables respectively. The `simulate_error` function generates the random sampling error and `generate_response` takes the results from the first two functions to generate the outcome.

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

Table 1 shows the first six rows of the simulated data. The output shows the values for the sex variable which would represent the variable of interest and the outcome variable here is represented as the last column labeled y. The random error values are also given in the error column. It is useful to explore the simulated data prior to going to simulation to ensure that was in simulated is what was specified. In addition, the output in Table **??** shows the the mean difference of 0.15 would indicate that males are 0.15 standard units higher than females. If the opposite would want to be created, the parameter in the `reg_weights` argument could be made negative (i.e. -0.15) which would indicate males have a smaller

Table 1

The output from the simulated data generated with simglm

X.Intercept.	sex	sex1	level1_id	error	fixed_outcome	random_effects	y
1	1	male	1	2.29	0.15	0	2.44
1	1	male	2	-1.39	0.15	0	-1.24
1	0	female	3	-0.28	0.00	0	-0.28
1	0	female	4	-0.13	0.00	0	-0.13
1	0	female	5	0.64	0.00	0	0.64
1	0	female	6	-0.28	0.00	0	-0.28

mean.

Using `simglm` to generate power

Very few adjustments are need to generate power using the `simglm` package once the simulation code is added. The number of replications need to be specified, a specification of what kind of model should be fitted, and whether to extract the model coefficients. The number of replications indicate how many times the data will be simulated and subsequently how many times model estimates will be obtained. In general, larger is better and will increase precision in the power estimates that are obtained. The number of replications can be thought of as similar to sample size in primary studies. In the example below, the number of replications is set to 1000 which is a minimum value that I would recommend and was used here as a trade-off between time to run the power analysis and precision of the power estimates. The more replications that are run, the longer the power analysis will take to run.

The model to be fitted is specified with the names `model_fit` argument. Only the `model_function` needs to be specified within the `model_fit` list to indicate which R model function should be used. The additional argument of model formula is also shown below as this gives the user additional flexibility and would allow the user to explore the impact of model misspecification (i.e. not including a model variable that is present in the population). If the model formula is omitted from the `model_fit` argument, the same model formula as the simulated model is specified. Finally, the `extract_coefficients` is commonly specified as TRUE to indicate that the estimated coefficients from the fitting of models from `model_fit` will be returned. If the `extract_coefficients` argument is missing from the simulation arguments, then the fitted model object is returned which would need to be processed to extract relavent information.

```r
plan(multiprocess)

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),
  replications = 1000,
  model_fit = list(formula = y ~ 1 + sex,
                   model_function = 'lm'),
  extract_coefficients = TRUE
)

replicate_sim <- replicate_simulation(simulation_arguments)
```

Finally, one last function is needed to compute power and other useful summary statistics based on the analysis. The `compute_statistics` function is used to compute

Table 2

Statistical power and precision output obtained from the simglm package.

Term	Avg Est	Power	Avg TS	Est SD	Avg SE	Precision	Repl
(Intercept)	0.01	0.08	0.04	0.33	0.32	1.02	1000
sex	0.13	0.08	0.30	0.46	0.46	1.01	1000

these summary statistics from the replicated simulation results. The primary arguments for this function include the simulation arguments and which statistics to be returned. In the example below, power and precision are set to TRUE to return those statistics and type I error statistics are omitted as power is of most interest here. For simulation studies to explore methodological impacts of assumption violations, type I error may be of interest to users.

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

Table 2 shows the output from the power analysis with simglm. The output shows the information for power and precision. In the output, if power estimates are of most concern, the columns "Avg Est" and "Power" are most relavent which provide estimates for the average estimates and the number of binary hypotheses correctly rejected across all replications. The additional information is more for reference, but contain information on the average test statistic, parameter estimate standard deviation, average standard error, precision (estimated standard deviation / average standard error), and number of replications. In the example shown above with only a sample size of 20 individuals, the estimated power was 0.08 meaning that the null hypothesis would only be correctly rejected 8% of the time.

Varying arguments with `simglm`

The example with `simglm` above only considered power for a single sample size, but a power-curve was shown above that reflected various sample sizes and various effect sizes simultaneously. This can be achieved in a single analysis by using the `vary_arguments` argument to the simulation arguments. As shown in the code below, the `sample_size` argument is removed from the main simulation arguments and it is instead specified within the `vary_arguments` argument. The sample size is generated to indicate that the sample size should be evaluated for the sequence starting at 20, ending at 2000, and incrementing by 20. When specifying arguments to vary, the default behavior is to do a factorial design meaning that all combinations are considered and replicated as part of the power analysis. More specifically for this example, this would mean that sample sizes of 20, 40, 60, up to 2000 will be run and replicated 1000 times.

Finally, an optional argument named, `power`, is shown that provides further user customization to how power is calculated. For example, the power arguments defined below indicate that a standard normal critical value with an alpha value of 0.05 and the default behavior is to assume a two-tailed hypothesis test. Therefore, the test statistics estimated from the model fitting will be compared to a critical value of ± 1.96. This was used here for its simplicity, but it is acknowledged that this may result in conservative estimates of power for smaller sample sizes. In these cases a t-distribution may be more appropriate.

```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)
```

The compute_statistics function is again used to generate summary statistics. Internally, the arguments that are varied are used as grouping variables when computing these summary statistics, therefore power values will be computed for all unique values for arguments that were varied. In addition, some post-processing is done to turn the sample size values from a factor variable to numeric for easier use later on and only the power values for the sex variable are returned as this is the power effect of interest.

```
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')
```

The tabular results could be explored by viewing the `power_results` object directly, however this would be more difficult to interpret quickly. Therefore, the result can then be explored visually to generate a power curve across the sample sizes for this single effect size. The results mimic was would found in Figure 1 where sample sizes for each group was approximately 700.

```
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 = min(filter(power_results, power >= 0.8)$power), linetype = 2,
  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 sex effect and the associated p-values based on these.

The `model_results` object is a list across sample size conditions, therefore this is first combined into a dataframe and then plotted with ggplot2. The resulting figure shows the increased precision as sample size increases. More specifically, the standard error is decreasing, but on average the parameter estimate is centered around 0.15.

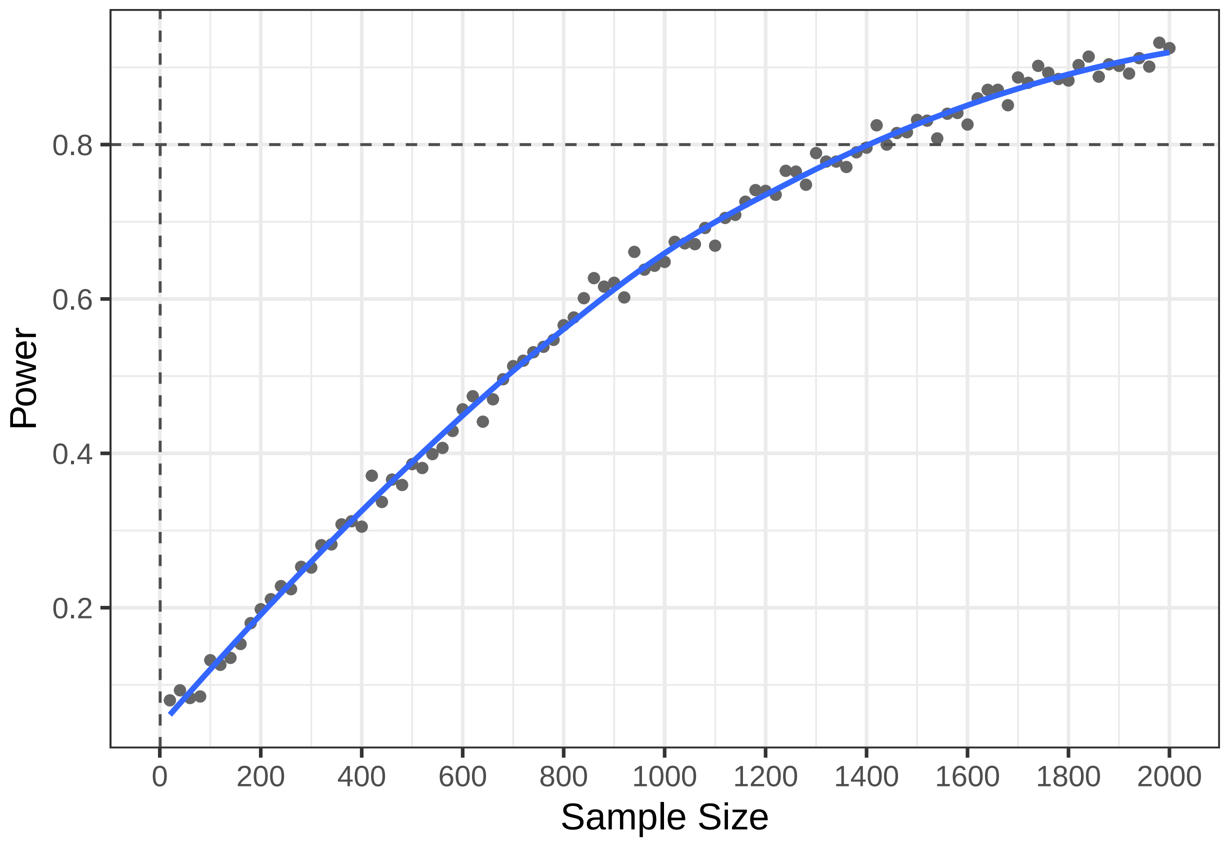


Figure 3. A single power curve for varying sample size conditions with a single effect size using simglm.

```r
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()
```

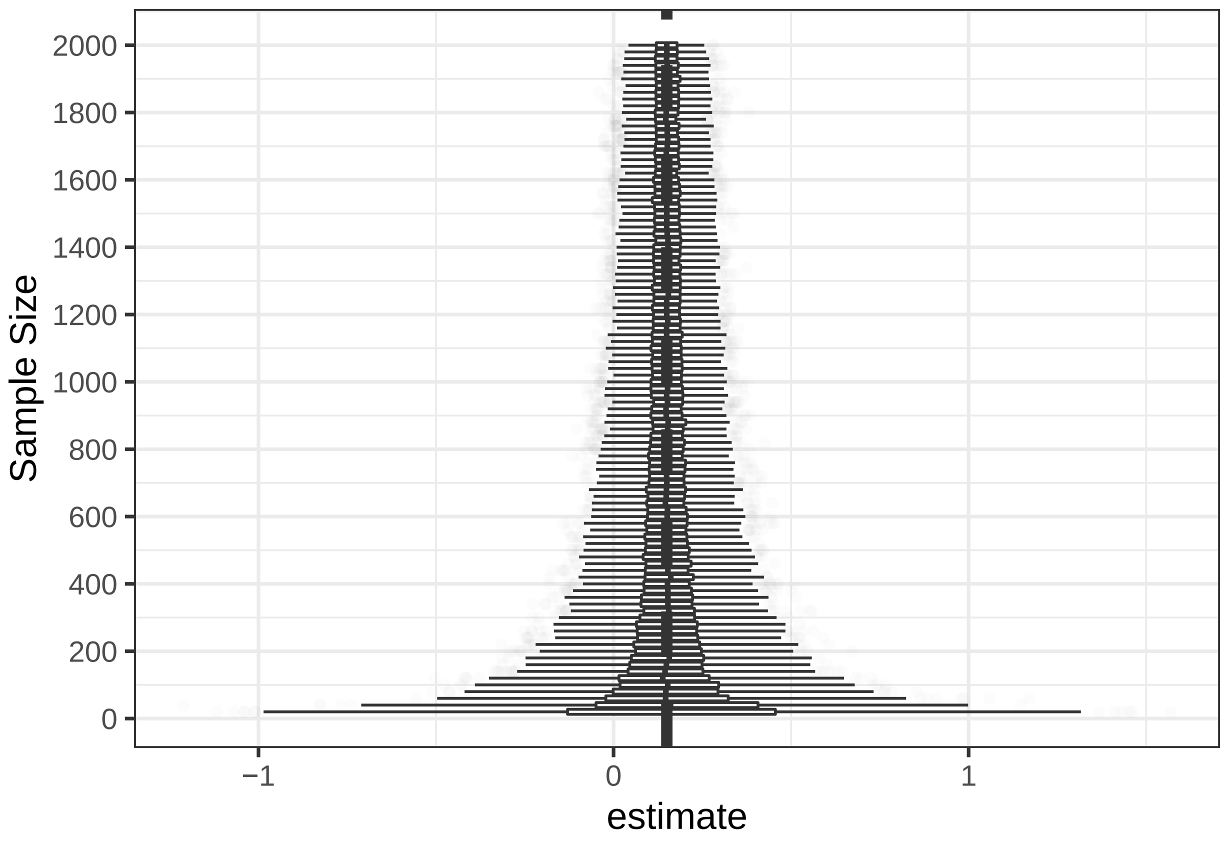
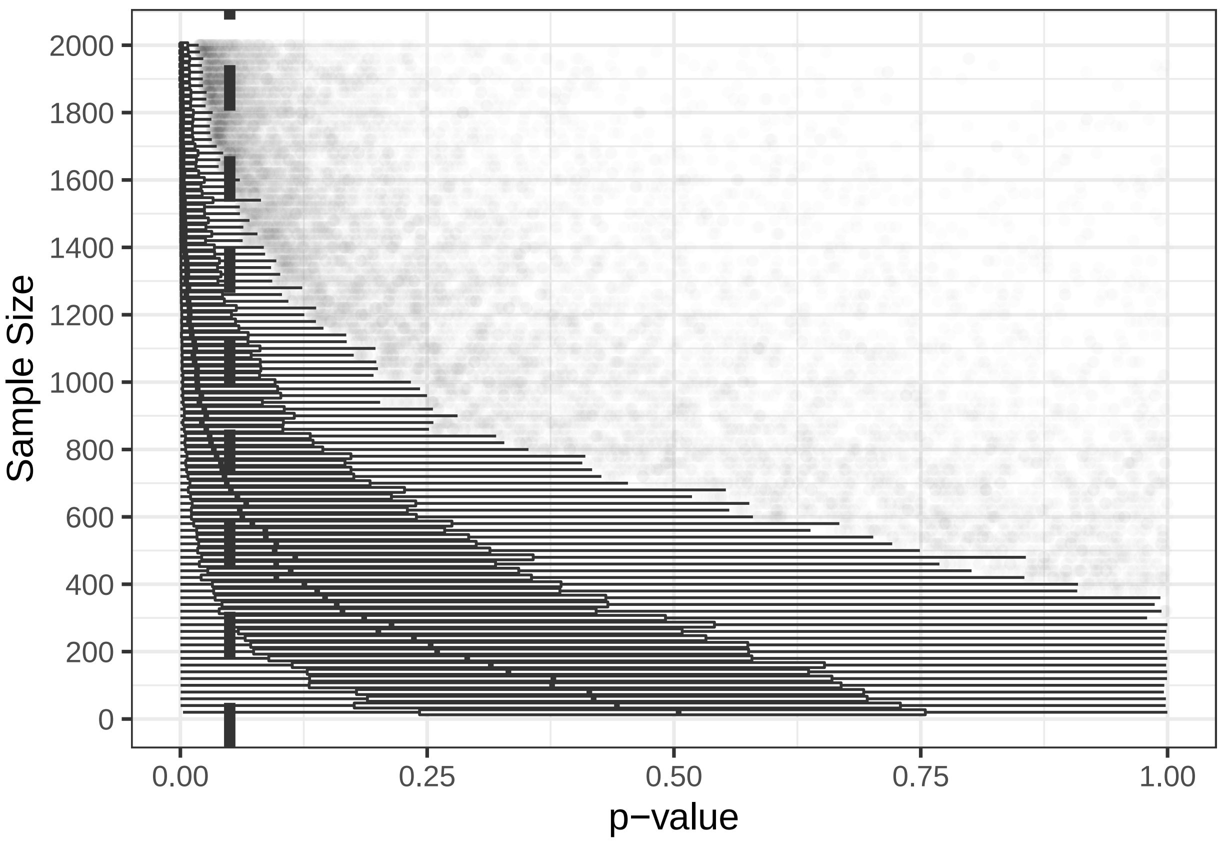


Figure 4. Distribution of parameter estimates for the sex effect by sample size.

A similar figure can be created showing the distribution of p-values, which gives a similar picture. There is much more variation when the sample sizes are small compared to larger sample sizes which reflects the smaller standard errors.

```r
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)) +
  ylab("p-value") +
  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 `power.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.

To simulate heterogeneity an additional argument to the simulation arguments is added called `heterogeneity`. This argument species the group to which the heterogeneity is applied, in this case the group/sex variable and the specific variances for the values of that variable. In this example, there are only two groups therefore only two variances need to be supplied. The variance for the second group is set to be about 8 times that of the first group, a ratio that exceeds a common heuristic of 3 times variance ratio indicating a violation of the statistical assumption homogeneity of variance.

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

These results are then processed similarly to above and combined with the power results that assumed a constant variance across the two groups. Figure 5 shows the power estimates contrasting results with and without heterogeneity. The effect of heterogeneity is quite stark with significantly reduced power estimates when there is heterogeneity. Smaller variance ratios could be considered to explore smaller ratios and the results impact on power estimates.

```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')

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) +
  scale_x_continuous("Sample Size", breaks = seq(0, 2000, 200)) +
  scale_y_continuous("Power", breaks = seq(0, 1, .2)) +
  theme_bw(base_size = 14)
```

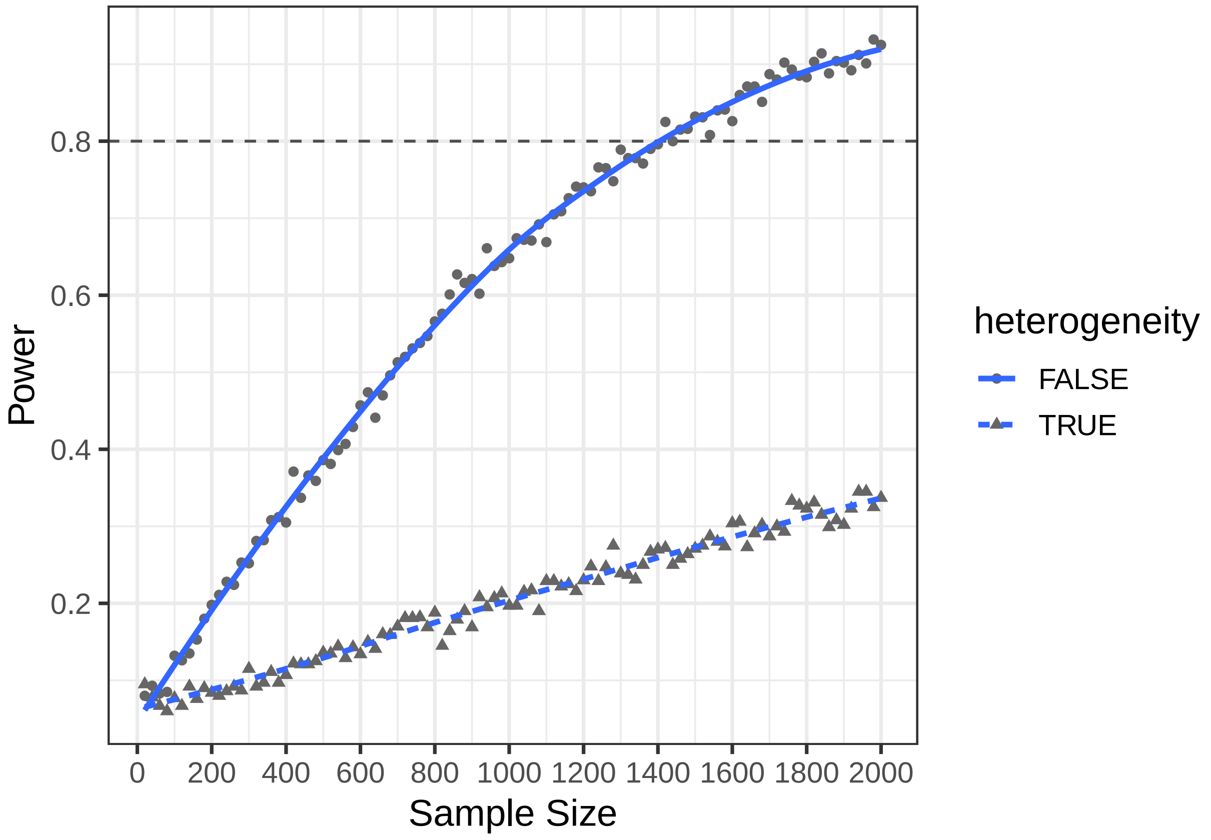


Figure 5. Power estimates for a single effect size (0.15) by differing sample sizes and whether there is variance heterogeneity.

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) (Fitzmaurice, Laird, & Ware, 2012), sometimes known as the hierarchical linear model (HLM) (Raudenbush & Bryk, 2002) or multilevel model (Goldstein, 2011).

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.

The code below also introduces a new argument to the fixed and randomeffect arguments, called `var_level` which indicates at which level the variable belongs to. The default value is level 1 which would represent the repeated measures and level 2 would represent individuals. In this case, the sex variable and the random effects need to be specified at level 2 or the individual level (i.e. these values should be constant for an individual not vary within an individual). This is done for these variables by setting `var_level = 2`. Similarly, the sample sizes are specified for level 1 and level 2 separately in the `sample_size` argument with `level1 = 10` and `level2 = 500` respectively.

```
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)
```

The data structure for this is not shown directly as it would look very similar to that shown in Table 1.

Power for Repeated Measures

The structure for power for repeated measures is very similar to that found with when doing power for the two-sample t-test. The addition of number of replications, model fitting instructions, and whether to extract model coefficients are specified with the `replications`, `model_fit`, and `extract_coefficients` simulation arguments respectively.

```
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),
```

Table 3

Statistical power and precision output obtained from the repeated measures example.

Term	Avg Est	Power	Avg TS	Est SD	Avg SE	Precision	Repl
(Intercept)	4.0	1.00	54.60	0.07	0.07	1.02	1000
sex	0.3	0.82	2.89	0.11	0.10	1.04	1000
time	0.4	1.00	8.94	0.04	0.05	0.96	1000
time:sex	0.2	0.87	3.11	0.06	0.06	0.96	1000

```
                      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)
```

The `compute_statistics` function is again used to summarize the power estimates by the terms in the model. Power and precision summary statistics are returned directly and summarized in Table 3. Power estimates reveal that power is above 0.8 for all variables in the model.

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

Varying simulation conditions. Sample size is now varied to explore the impact that different level 1 and level 2 sample sizes have on power estimates. The `vary_arguments` simulation argument is used to control this behavior where the level 1 sample size was specified to either be 5 or 8 measurement occasions and the level 2 sample size was varied to be either 50, 150, or 250 individuals.

```
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)
```

The summary statistics from the different sample size conditions are summarized and save in the object, `long_power_nomiss`.

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

In addition to just sample size, missing data can also be added to the design with the simulation argument, `missing_data`. Dropout missing data can be common in longitudinal designs where data from individuals stops at some point in the data collection. For example, an individual may come for the first two measurement occasions, but the subsequent measurement occasions for that individual are missing. Arguments within the `missing_data` include the new outcome variable to include in the data object that has the missing data included called `new_outcome`, the proportion of missing data with `miss_prop`, the cluster variable called `clust_var`, and the type of missing data mechanism with `type`. To call dropout missing data set `type = "dropout"`. The simglm documentation contains information on additional missing data mechanisms that are currently supported. The last aspect that needs to be modified is to use the new outcome variable when specifying the `model_fit` simulation argument. More specifically, the outcome in the model fitting step should be `y_miss` instead of `y` to ensure that the missing data are dropped from the model fitting procedure.

```
sim_arguments <- list(
  formula = y ~ 1 + time + sex + time:sex + (1 + time | individual),
  reg_weights = c(4, 0.4, 0.3, 0.20),
```

```r
  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)
```

Upon simulation and model fitting, the power and precision summary statistics are calculated and the power estimates are combined for the missing and complete data conditions. Some processing to make the sample size data labels easier to read for plotting is also done.

```r
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)
```

Figures 6 and 7 show the power estimates by various sample size conditions and whether there was missing data for the sex and time by sex interaction effects respectively. The results show that the level 2 sample size has greater impacts on power compared to changes in the level 1 sample size. In addition, the dropout missing data has greater impacts on the time by sex interaction effect compared to the sex effect. This is not surprising as the sex effect is exploring changes in the intercept based on whether the individual is male or female. In general across all of the conditions, power does not reach the often desired level of 0.80, but is closer for the time by sex interaction with the largest sample sizes.

```r
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') +
  xlab("Sample Size") +
  scale_y_continuous("Power", breaks = seq(0, 1, .2), limits = c(0, 1)) +
  theme_bw(base_size = 14) +
  coord_flip()
```

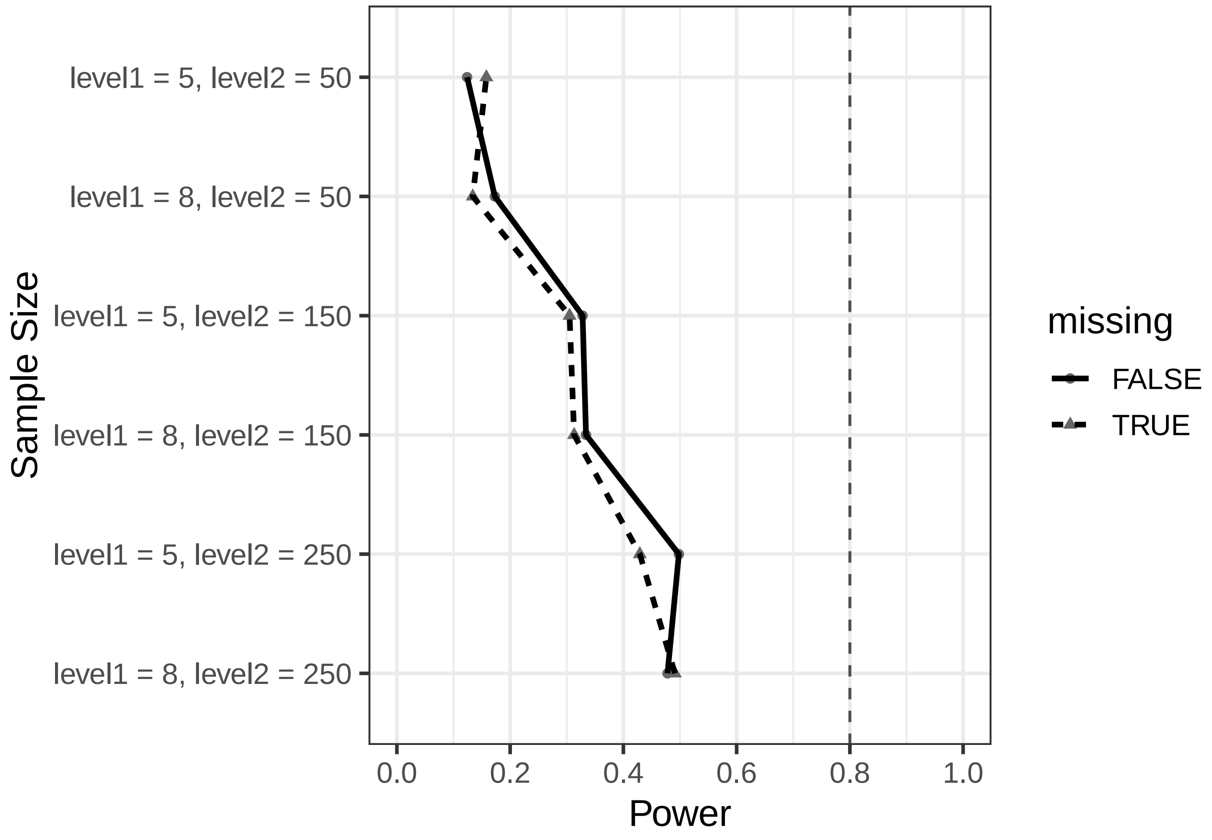


Figure 6. Power estimates for different sample sizes and whether missing data is included for the sex effect.

```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()
```

Summary

This paper works through power examples in the traditional framework first, then explores how to generate power estimates through simulation for the two-sample t-test and repeated measures frameworks. When statistical assumptions held, the power estimates were similar between the traditional and simulation frameworks (see 1 3). This is not surprising given the assumptions that closed form power solutions make to estimate power.

The benefits of estimating power through a simulation framework occurs with the flexibility that the procedure has and the ability to mimic real world data conditions. In addition, there are models for which closed form solutions to estimate power are not as readily available, for example generalized linear models that have outcomes that are non-normally distributed and/or not continuous are more difficult to obtain closed form solutions for power.

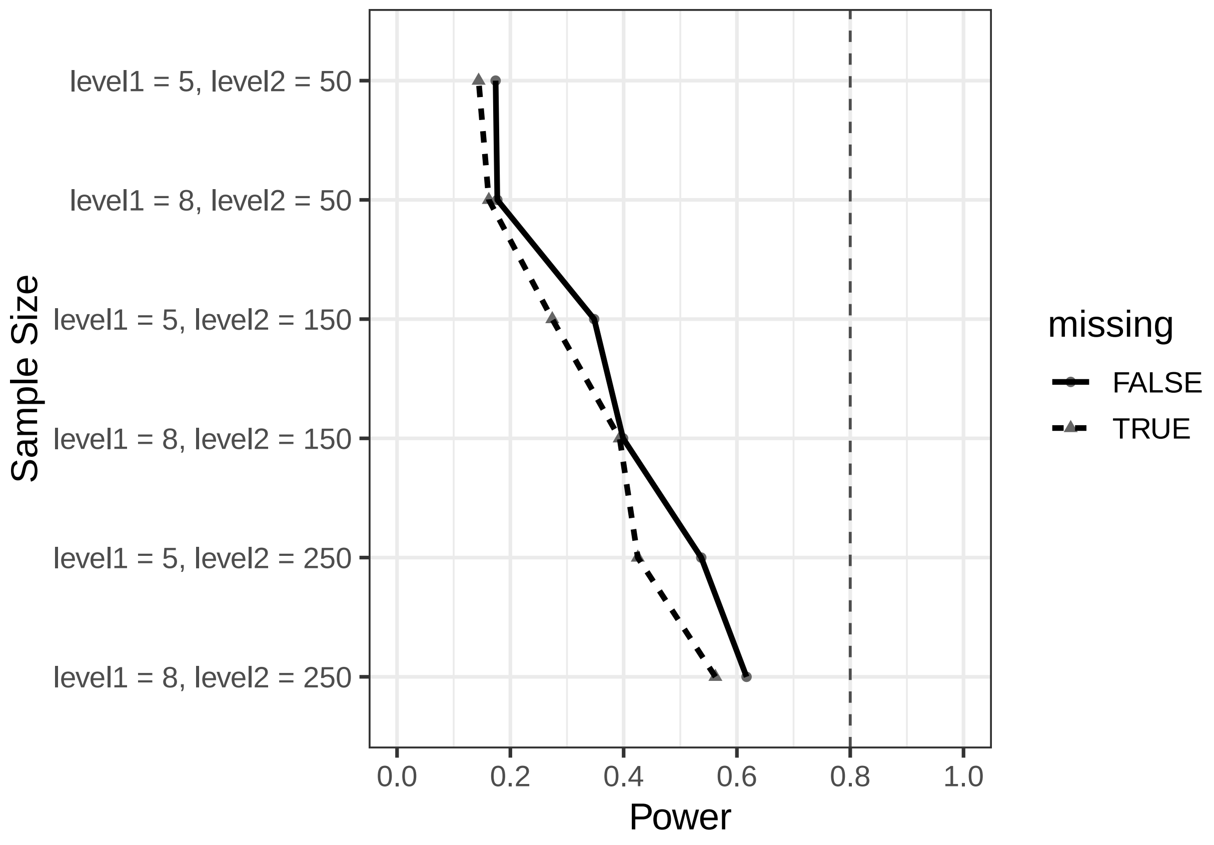


Figure 7. Power estimates for different sample sizes and whether missing data is included for the time by sex interaction.

In this tutorial, the `simglm` R package was used for power by simulation. The `simglm` package provides a framework to simulate data from a variety of generalized linear models including models that have nested or cross-classified data structures. The package also provides users ways to replicate the simulation design, vary simulation arguments, and estimate power with wrapper functions that do not require users to have to program the replications or data summarization. Users are directed to the `simglm` documentation, vignettes, and GitHub page (https://github.com/lebebr01/simglm) for additional examples and package options. The additional options include the ability to simulate models with binary or count outcomes, add additional missing data mechanisms, add additional levels of nested or cross-classifated data structures, and generate different types of variables.

References

Champely, S. (2018). *Pwr: Basic functions for power analysis.* Retrieved from https://CRAN.R-project.org/package=pwr

Dong, N., Kelcey, B., Maynard, R., & Spybrook, J. (2015). *PowerUp! Tool for power analysis.* Retrieved from https://www.causalevaluation.org/power-analysis.html

Faul, F., Erdfelder, E., Lang, A.-G., & Buchner, A. (2007). G* power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior Research Methods*, *39*(2), 175–191.

Fitzmaurice, G. M., Laird, N. M., & Ware, J. H. (2012). *Applied longitudinal analysis* (Vol. 998). John Wiley & Sons.

Goldstein, H. (2011). *Multilevel statistical models* (Vol. 922). John Wiley & Sons.

Hoenig, J. M., & Heisey, D. M. (2001). The abuse of power: The pervasive fallacy of power calculations for data analysis. *The American Statistician*, *55*(1), 19–24.

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

Raudenbush, S., Spybrook, J., Bloom, H., Congdon, R., Hill, C., & Martínez, A. (2011). *Optimal design software for multi-level and longitudinal research.* Retrieved from http://hlmsoft.net/od/

Raudenbush, S. W., & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods* (Vol. 1). Sage.

R Core Team. (2019). *R: A language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Thomas, L. (1997). Retrospective power analysis. *Conservation Biology*, *11*(1), 276–280.

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from http://ggplot2.org

Zhang, Z., & Mai, Y. (2018). *WebPower: Basic and advanced statistical power analysis*. Retrieved from https://CRAN.R-project.org/package=WebPower