Power Calculations and Simulations - R lab

Introduction

Why are power calculations important?

Power influences many design aspects, including what research questions to pursue, how many treatment arms to employ, and even more fundamentally, whether or not to proceed with a potential research project. For example, it may be that a remedial education program boosts tests scores by 20 percent when comparing treatment and control groups, but due to limited power, the RCT is unable to detect this true effect (with 95% confidence). However, we can estimate whether a given design is likely to be able to detect a reasonable effect size ex ante, allowing us to properly manage partner expectations and make the most of limited research resources.

This exercise will cover two ways of calculating power for a RCT :

1. The conventional parametric method, and
2. A non-parametric "simulation" method.

Questions to consider before running power calculations

- What is the main specification (e.g. regression) we plan to run? (It doesn't have to be fully baked, but the more "baked" it is, the more precise we can make your power estimates.)
- What do we expect to be the mean of the outcome in the control group?
- How about the standard deviation (SD) of the outcome in control group? (If your outcome is binary, we can estimate this without baseline data by assuming this Bernoulli random variable takes a binomial distribution.)
- What sample sizes are feasible?
- What effect sizes could the intervention reasonably cause?
- What is the smallest, cost-effective effect size that we are interested in? (We often arrive at a reasonable answer to this question through discussions with partner organizations and literature reviews.)

Please install these packages if you don't have them:

```
install.packages(c("haven", "ICC", "randomizr", "multiwayvcov", "lmtest", "knitr",
    "devtools", "Hmisc"), dependencies = TRUE, INSTALL_opts = c("--no-lock"))
devtools::install_github("vikjam/pwrcalc")
```

Once you installed the packages, load them to get started:

```
library(haven)
library(ICC)
library(randomizr)
library(multiwayvcov)
library(lmtest)
library(knitr)
library(pwrcalc)
library(Hmisc)
```

Let's start with a simple parametric example.

Example 1. Basic Parametric Example

Before we get started make sure you've set your directory properly. You can check your directory with `getwd` and if you need to change it you can set it with `setwd`:

```
getwd()
setwd("D:/R/J-PAL Power Tutorial")
```

The `balsakhi` dataset is provided with the package `pwrclc`. We'll use this dataset to estimate the control mean:

```
data(balsakhi)
```

Let us view the `balsakhi`data that we have just loaded:

```
View(balsakhi)
```

For all parametric power calculations, we'll assume a conventional 95% confidence interval and 80% power.

What do we expect the mean and standard deviation of the outcome to be in the control group?

```
control_mean <- mean(subset(balsakhi$post_totnorm, balsakhi$bal == 0), na.rm = TRUE)
control_sd <- sd(subset(balsakhi$post_totnorm, balsakhi$bal == 0), na.rm = TRUE)
```

Before proceeding lets check the values of `control_sd` and `control_mean`:

```
control_sd
```

```
## [1] 1.15142
```

```
control_mean
```

```
## [1] 0.4288781
```

Note: Since power calculations are usually done prior to a study, we often use baseline/pilot data on the study population, or government statistics for a comparable population, to get an approximate for this outcome in the control group.

Let's say, based on other studies, that we expect an effect size of a tenth of a standard deviation. Now let's calculate the sample size given that we know the likely effect size:

```
expected_effect <- control_sd/10
treated_mean <- expected_effect + control_mean
```

`treat` in the Stata exercise is equivalent to `treated_mean` in this R exercise. `control` is the Stata exercise is equivalent to `control_mean` in this R exercise.

Let us check the variables that we have:

```
treated_mean
```

```
## [1] 0.5440201
```

```
control_mean
```

```
## [1] 0.4288781
```

```
expected_effect
```

```
## [1] 0.115142
```

```
control_sd
```

```
## [1] 1.15142
```

We can take a look at the package `pwrclc` documentation now to understand the function `twomeans`. The function calculates the sample sizes for two-sample test based on means and standard deviations of the two samples.

Suppose we want to detect a difference of 4 between two groups (e.g., control and treatment). For example, we anticipate the control group mean being 12 and the treatment group mean being 16. In addition, suppose the standard deviation of each group is 5. We can calculate the sample size required with `pwrcalc`:

```
twomeans(m1 = 12, m2 = 16, sd = 5)
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 12
##              m2 = 16
##              n1 = 25
##              n2 = 25
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Now, we can calculate the sample sizes for the `balsakhi` dataset:

```
twomeans = twomeans(m1 = control_mean, m2 = treated_mean, sd = control_sd)
twomeans
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 1570
##              n2 = 1570
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Say, instead, we knew the sample size and wanted to calculate the Minimum Detectable Effect Size (MDE). You can manually calculate the effect size:

```
sample_n <- twomeans$n1 + twomeans$n2  # this takes sample sizes n1 and n2 from the previous code chunk
mde <- (0.842 + 1.96) * sqrt(1/0.25) * sqrt(1/sample_n) * control_sd
mde
```

```
## [1] 0.1151508
```

The effect size should be .1151. Verify you get this result before proceeding. Let's compare this to the likely effect size that went into the first sample size calculation:

```
expected_effect
```

```
## [1] 0.115142
```

As this shows, it doesn't matter which we start with - sample size or effect size.

Some other questions to answer before calculating power:

- Will this study be cluster-randomized?
- Will our main specification include controls (i.e. lagged dependent variables)?
- Do we expect only part of the treatment group to take-up the intervention? If so, are we interested in estimating the local average treatment effect?

We'll address each of these questions one at a time to see how they affect power.

Example 2. Building Intuition

Now, let us get a better intuition on how a larger or smaller sample size affects our power to pick up an effect.

Say our anticipated effect size is smaller than originally thought; how much larger would we need to make the sample in order to still pick up an effect?

Let's try an effect size that is half as large:

```
smaller_expected_effect <- expected_effect/2
smaller_treated_mean <- smaller_expected_effect + control_mean
twomeans(m1 = control_mean, m2 = smaller_treated_mean, sd = control_sd)
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.4864491
##              n1 = 6280
##              n2 = 6280
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Observation: The minimum sample required is four times as large.

Let's try an effect size that is a third of the original:

```
smaller_expected_effect <- expected_effect/3
smaller_treated_mean <- smaller_expected_effect + control_mean
twomeans(m1 = control_mean, m2 = smaller_treated_mean, sd = control_sd)
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
```

```
##              m2 = 0.4672588
##              n1 = 14128
##              n2 = 14128
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Observation: The minimum sample required is now nine times as large.

Remember: If our MDE decreases by a factor of X, the required sample size increases by the square of X!

You can verify this from the other side i.e. look at the impact on the MDE of increasing your sample size by a factor of X. Say X is 4:

```
new_sample <- 4 * (3142)  # Global n of the Stata exercise
new_mde <- (0.842 + 1.96) * sqrt(1/0.25) * sqrt(1/new_sample) * control_sd
new_mde/mde
```

```
## [1] 0.4998408
```

Example 3. Parametric Power Calculation with Controls

Now, say we plan to control for baseline covariates in our main specification. The inclusion of these controls will improve our power, since they explain some of the variance in our outcome. For example, including prior test scores on the right-hand side as controls when our left-hand side variable is test scores during the study period, can improve power if prior test scores predict future test scores. Economists will often control for a lagged dependent variable (recorded prior to randomization) if it makes a first order improvement in power.

To see how potential controls affect power, we would ideally have access to a sample data set (e.g. historical or pilot data). With these data, we would want to regress Y_i (the outcome) on X_i (the controls) to evaluate how much variance is explained by the set of covariates we plan to include. With access to historical data, for example, this would involve regressing last year's test scores (Y_i = Y_t-1) on test scores from the year before (X_i = Y_t-2).

From this regression, we are interested in the residual standard deviation of the outcome variables, or the variance of the outcome that is NOT explained by controls. This residual SD becomes the new SD we include in our parametric power calculations.

Part 1: We have pilot/historical data.

Using `balsakhi` data, this would be:

```
fit <- lm(post_totnorm ~ pre_totnorm, data = balsakhi, subset = bal == 0)
summary(fit)
```

```
##
## Call:
## lm(formula = post_totnorm ~ pre_totnorm, data = balsakhi, subset = bal ==
##     0)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6007 -0.5264 -0.0086  0.5312  3.1473
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.40028    0.01247   32.10   <2e-16 ***
## pre_totnorm  0.80506    0.01243   64.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.821 on 4340 degrees of freedom
##   (866 observations deleted due to missingness)
## Multiple R-squared:  0.4916, Adjusted R-squared:  0.4915
## F-statistic:  4197 on 1 and 4340 DF,  p-value: < 2.2e-16
```

```
balsakhi$y_predicted <- fit$coefficients["pre_totnorm"] * balsakhi$pre_totnorm +
    fit$coefficients["(Intercept)"]
balsakhi$residual_predicted <- balsakhi$post_totnorm - balsakhi$y_predicted
res_control_sd <- sd(balsakhi$residual_predicted, na.rm = TRUE)
res_control_sd
```

```
## [1] 0.8254881
```

An alternate longer way to do the same is:

```
control_subset <- subset(balsakhi, bal == 0)
fit <- lm(post_totnorm ~ pre_totnorm, data = control_subset)
```

If we knew the effect size and wanted to know the sample size needed:

```
twomeans(m1 = control_mean, m2 = treated_mean, sd = signif(res_control_sd, digits = 6))
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 807
##              n2 = 807
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Questions:

1. What percent of the variance of study period test scores is explained by test scores at baseline? (Hint: Look at the R^2 statistic of the regression.)
2. How does this affect our sample size (compared to not including controls)?
3. How about our MDE?

Part 2: We do not have pilot/historical data for our study population, but plan to include controls.

We must first guess the percentage of variance in outcome variables that we expect to be explained by controls. (This can be done by looking at other data-sets with similar outcomes, etc.)

Say our controls explain 49% of the outcome. We can then calculate the residual standard deviation as follows:

```
expected_res_control_sd <- sqrt(0.51 * control_sd^2)
twomeans(m1 = control_mean, m2 = treated_mean, sd = expected_res_control_sd)
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 801
##              n2 = 801
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

We can also trace out an MDE "range" using various assumptions for how much of the variance of the outcome is explained by covariates:

```
for (proportion in seq(0.5, 0.9, 0.1)) {
    expected_res_control_sd <- sqrt(proportion * control_sd^2)
    cat("\nThe assumed proportion of variance explained by covariates = ", proportion,
        "\n")
    print(twomeans(m1 = control_mean, m2 = treated_mean, sd = expected_res_control_sd))
}
```

```
##
## The assumed proportion of variance explained by covariates =  0.5
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 785
##              n2 = 785
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
##
##
## The assumed proportion of variance explained by covariates =  0.6
##
```

```
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 942
##              n2 = 942
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
##
##
## The assumed proportion of variance explained by covariates =  0.7
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 1099
##              n2 = 1099
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
##
##
## The assumed proportion of variance explained by covariates =  0.8
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.5440201
##              n1 = 1256
##              n2 = 1256
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
##
##
## The assumed proportion of variance explained by covariates =  0.9
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
```

```
##              m2 = 0.5440201
##              n1 = 1413
##              n2 = 1413
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Please note that the code snippet above is the equivalent of the `controlling_for_covariates.do` file in Stata. There is no separate R script.

Example 4. Parametric Power Calculation for Cluster-RCTs

Many designs randomize at the group level instead of at the individual level. For such designs, we need to adjust our power calculations so that they incorporate the fact that individuals within the same group may be subject to similar shocks, and thereby have correlated outcomes. Duflo et al. presents a modified parametric approach, which takes into account the intra-cluster correlation (ICC) that arises from randomization at the group level.

We can think of cluster-RCTs as follows:

- When ICC = 0, then our `N` is effectively the number of individuals in the study.
- When ICC = 1, then our `N` is effectively just the number of clusters.
- Usually the ICC lies somewhere between 0 and 1, requiring that we adjust our power calculations to account for this.

Below we adjust R's power estimates based on Duflo et al.'s model.

Note: This model assumes that all clusters are of the same size and have the same number of individuals. It's usually okay if this is violated in reality, but you would not want to use these adjustments if groups are dramatically different in size (e.g. group one has 10 individuals, group two has 1,000 individuals.) More on this model is explained in Duflo et al.'s article "Using Randomization in Development Economics Research: A Toolkit."

Part 1: Calculating MDE

First, let's calculate the intra-cluster correlation (ICC) which measures how correlated the error terms of individuals in the same cluster are:

```
control_subset <- subset(balsakhi, bal == 0 & !is.na(divid) & !is.na(post_totnorm))
control_subset$divid = as.factor(control_subset$divid)
icc <- ICCest(divid, post_totnorm, data = control_subset)
rho <- icc$ICC
```

Now, let's specify the number of individuals in each cluster:

```
m <- 53
```

and the number of clusters (as documented in the Balsakhi experiment):

```
j <- 193
```

Let us assume 95% confidence intervals and 80% power:

```
t_stat <- 1.96 + 0.842
```

and 50% of the study population is assigned to treatment and 50% to control:

```
P <- 0.5
```

Now, we have Duflo et al.'s power adjustment:

```
mde <- t_stat * sqrt(1/(P * (1 - P) * j)) * sqrt(rho + (1 - rho)/m) * control_sd
mde_ldv <- t_stat * sqrt(1/(P * (1 - P) * j)) * sqrt(rho + (1 - rho)/m) * res_control_sd
```

And lastly, the total `N` of our study and the number who are treated, respectively:

```
n <- j * m
treated <- n * P

mdes <- c(0.05, 0.8, rho, j, m, n, treated, control_mean, control_sd, mde, 0.05,
    0.8, rho, j, m, n, treated, control_mean, control_sd, mde_ldv)
table_names <- list(c("No_controls", "Control_LDV"), c("Signif", "Power", "ICC",
    "Clusters", "Cluster_size", "N", "Treated", "Cntrl_mn", "Cntrl_SD", "MDE"))
mde_table <- matrix(data = mdes, nrow = 2, ncol = 10, byrow = 2, dimnames = table_names)
mde_table
```

```
##             Signif Power       ICC Clusters Cluster_size     N Treated
## No_controls   0.05   0.8 0.1554845      193           53 10229  5114.5
## Control_LDV   0.05   0.8 0.1554845      193           53 10229  5114.5
##              Cntrl_mn Cntrl_SD       MDE
## No_controls 0.4288781  1.15142 0.1923013
## Control_LDV 0.4288781  1.15142 0.1378667
```

Part 2: Calculating Sample Size

In Stata, `sampclus` command can be used in combination with the `sampsi` command to incorporate clustering into our sample size calculations. A more parsimonious alternative is the `clustersampsi` command. However, note that `clustersampsi` rounds the group means to the tenth place after the decimal, making for a less precise calculation.

For R, please refer to the `pwrclc` documentation for the function `clustered` that adjust for the number of individuals per cluster:

```
twomeans(m1 = 12, m2 = 16, sd = 5) %>% clustered(obsclus = 10, rho = 0.3)
```

```
##
##      Two-sample t-test power calculation
##
##                          m1 = 12
##                          m2 = 16
##            n1 (unadjusted) = 25
##            n2 (unadjusted) = 25
##                         rho = 0.3
##        Average per cluster = 10
## Minimum number of clusters = 19
##              n1 (adjusted) = 93
##              n2 (adjusted) = 93
##                  sig.level = 0.05
##                      power = 0.8
##                alternative = two.sided
##
```

```
## NOTE: m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Let us calculate the sample size:

```
smaller_treated_mean2 <- control_mean + mde
twomeans(m1 = control_mean, m2 = smaller_treated_mean2, sd = control_sd) %>% clustered(obsclus = m,
    rho = rho)
```

```
##
##      Two-sample t-test power calculation
##
##                          m1 = 0.4288781
##                          m2 = 0.6211794
##            n1 (unadjusted) = 563
##            n2 (unadjusted) = 563
##                         rho = 0.1554845
##        Average per cluster = 53
## Minimum number of clusters = 194
##              n1 (adjusted) = 5115
##              n2 (adjusted) = 5115
##                  sig.level = 0.05
##                      power = 0.8
##                alternative = two.sided
##
## NOTE: m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

If we include the lagged dependent variable as a control we have:

```
smaller_treated_mean2_ldv <- control_mean + mde_ldv
twomeans(m1 = control_mean, m2 = smaller_treated_mean2_ldv, sd = res_control_sd) %>%
    clustered(obsclus = m, rho = rho)
```

```
##
##      Two-sample t-test power calculation
##
##                          m1 = 0.4288781
##                          m2 = 0.5667448
##            n1 (unadjusted) = 563
##            n2 (unadjusted) = 563
##                         rho = 0.1554845
##        Average per cluster = 53
## Minimum number of clusters = 194
##              n1 (adjusted) = 5115
##              n2 (adjusted) = 5115
##                  sig.level = 0.05
##                      power = 0.8
##                alternative = two.sided
##
## NOTE: m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Note that again it doesn't matter if we start with the MDE, or with the sample size.

Questions:

1. Why do we have to adjust power for clustering when running a cluster-RCT?
2. Assuming ICC > 0, does adding a new cluster of 5 individuals or adding 5 individuals to already-existing clusters give us more power to detect effects?

Example 5. Parametric Power Calculation with Partial Take-up

In randomized designs, it is common that there is partial take-up of the intervention. For example, in the Oregon Health Insurance Experiment, the offer to apply for health insurance was associated with only a 25 percentage point increase in take-up of health insurance. When take-up is not 100 percent, researchers are often interested in the answers to second stage questions, such as what the average effect of becoming insured is on health care utilization.

Below, we provide code that adjusts R's power to take into account that only some individuals in the treatment group take up the intervention.

The measure of take-up that we care about is "effective take-up", or the percentage of individuals in the treatment group that takes up the intervention MINUS the percentage of individuals in the control group that takes up.

Let us say 90% take up in the treatment group, and 10% do so in the control group. We then have an effective take-up rate of 80%:

```
eff_tu <- 0.9 - 0.1
```

Now we calculate the adjusted MDE by multiplying the unadjusted effect size times the effective take-up rate:

```
adjusted_mde <- ((0.842 + 1.96) * sqrt(1/0.25) * sqrt(1/n) * control_sd) * eff_tu
adjusted_mde  # n is the original sample size
```

```
## [1] 0.05103936
```

If we want to estimate sample size, we follow a similar procedure:

```
treated_mean_adjusted <- control_mean + adjusted_mde
twomeans(m1 = control_mean, m2 = treated_mean_adjusted, sd = control_sd)
```

```
##
##      Two-sample t-test power calculation
##
##              m1 = 0.4288781
##              m2 = 0.4799175
##              n1 = 7990
##              n2 = 7990
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE:
## m1 and m2 are the means of group 1 and 2, respectively.
## n1 and n2 are the obs. of group 1 and 2, respectively.
```

Note that unlike with the MDE, partial take-up affects sample size quadratically; we divide estimates of sample size by the square of the effective take-up rate!

Note: For more on partial take-up and how it affects estimation of the power to detect local average treatment effects, please see the aforementioned article by Duflo et al.

Example 6. Non-parametric Power Simulations

Non-parametric power simulations do better than parametric power calculations when we have access to good data (historical, baseline, or pilot) on our study population. From these data, we can simulate a fake dataset that assumes the treatment has no effect and then see what effects we are powered to detect, by looking at the simulated 95% confidence interval around our null effect. We should expect that any effect greater than this confidence interval would be detected by our study.

In particular, power simulations do not require the assumption that the sampling distribution of your Beta coefficient(s) of interest takes a normal distribution in your (finite) sample. You may be particularly worried about this assumption (of parametric power calculations) if your sample is very small.

To do non-parametric power simulations we need to create a (reasonable) "fake" or simulated dataset. For example, if you have baseline data for the 3 months prior to a 12 month trial, then a reasonable way to expand this dataset would be to simply randomly draw days with replacement until you have 365 days in your dataset. Similarly, you could use historical data from the two years prior to the study to estimate the confidence interval around a null effect in the past year, with data on your outcome variable from two years ago serving as controls.

Let's do an example using `balsakhi` data to make this procedure more clear.

Step 1. Upload pre-period data

For this example, we expand study data for the control group of the Balsakhi experiment by 2, since this is similar to what would be available from historical data:

```
control_subset <- subset(balsakhi, bal == 0)
simulated = control_subset
simulated$studentid <- (-1) * simulated$studentid
simulated <- rbind.data.frame(control_subset, simulated)

keep_cols <- c("studentid", "pre_tot", "mid_tot", "post_tot", "divid", "pre_totnorm",
    "mid_totnorm", "post_totnorm")
simulated <- simulated[, keep_cols, drop = FALSE]
simulated <- simulated[order(simulated$studentid), ]

nrow(simulated)
```

```
## [1] 10416
```

```
summary.data.frame(simulated)
```

```
##    studentid            pre_tot         mid_tot          post_tot
##  Min.   :-64041343   Min.   : 0.00   Min.   :  0.00   Min.   : 0.00
##  1st Qu.:-12041566   1st Qu.:14.00   1st Qu.: 24.00   1st Qu.:20.00
##  Median :        0   Median :29.00   Median : 43.00   Median :38.00
##  Mean   :        0   Mean   :32.13   Mean   : 44.46   Mean   :40.88
##  3rd Qu.: 12041566   3rd Qu.:48.00   3rd Qu.: 65.00   3rd Qu.:61.00
##  Max.   : 64041343   Max.   :94.00   Max.   :234.00   Max.   :98.00
##                                      NA's   :1264     NA's   :1732
##      divid        pre_totnorm       mid_totnorm       post_totnorm
##  Min.   :31130   Min.   :-1.9114   Min.   :-1.9114   Min.   :-1.9114
##  1st Qu.:33440   1st Qu.:-0.7920   1st Qu.:-0.3418   1st Qu.:-0.4886
##  Median :41070   Median :-0.1851   Median : 0.5230   Median : 0.3207
##  Mean   :38704   Mean   : 0.0000   Mean   : 0.5970   Mean   : 0.4289
##  3rd Qu.:43250   3rd Qu.: 0.7045   3rd Qu.: 1.5131   3rd Qu.: 1.3228
##  Max.   :46400   Max.   : 3.4059   Max.   : 9.2180   Max.   : 3.7094
```

```
##                                    NA's   :1264      NA's   :1732
```

```
write.csv(simulated, "bal_power_data_r.csv", row.names = FALSE, quote = FALSE)
```

Step 2. Write randomization code as you plan to randomize

&

Step 3. Write simulation code.

Your randomization code should take into account how you plan to randomize, including your plan to stratify, etc.

This power simulation program does the following:

(a) specifies main regression equation,

(b) simulates treatment assignment and runs main regressions 1000 times,

(c) summarizes the results from these regressions into an easy-to-read matrix.(left as an exercise)

To reduce the run-time, we have set this program to run over 1 iteration, but if you are truly calculating power via simulation, you should set this at 1000:

```
alpha <- 0.05  # Standard significance level
sims <- 1  # Number of simulations to conduct
```

Initialize a matrix to collect results. The matrix can be filed up by users according to their choice. and has been left as an exercise. For hint see the end of the document:

```
colnames <- c("pvalue", "tstat", "control_mean", "control_sd", "beta", "se", "ci_low",
    "ci_high")
results <- matrix(nrow = sims, ncol = 8)
colnames(results) <- colnames
```

Part 1. Basic MDE (clustering on divid because data is from C-RCT)

Loop to conduct experiments `sims` times over. Output is shown for `sims` = 1:

```
source("C:\\Users\\sabhyag\\Documents\\Training\\R lab\\Power_R\\R scripts\\Clustered_SE.R")  #make sur
for (i in 1:sims) {
    # We do a clustered random assignment using divid as the cluster
    simulated$treatment <- cluster_ra(clusters = simulated$divid)
    # View(simulated)

    # Do analysis (Simple regression)
    fit <- lm(post_totnorm ~ treatment, data = simulated, na.action = na.exclude)
    # call for clustered SE by divid
    output <- cluster_summary(fit, simulated$divid)
    print(output)
}
```

```
## [[1]]
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.501637   0.067493  7.4324 1.168e-13 ***
## treatment   -0.138379   0.103001 -1.3435    0.1792
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## [[2]]
## Wald test
##
## Model 1: post_totnorm ~ treatment
## Model 2: post_totnorm ~ 1
##   Res.Df Df      F Pr(>F)
## 1   8682
## 2   8683 -1 1.8049 0.1792
##
## [[3]]
##                Estimate    LowerCI    UpperCI
## (Intercept)  0.5016372  0.3693340 0.63394047
## treatment   -0.1383794 -0.3402854 0.06352672
```

Part 2. Controlling for LDV

Loop to conduct experiments `sims` times over. Output is shown for `sims` = 1:

```
for (i in 1:sims) {
    # We do a clustered random assignment using divid as the cluster
    simulated$treatment <- cluster_ra(clusters = simulated$divid)

    # Do analysis (Simple regression)
    fit <- lm(post_totnorm ~ pre_totnorm + treatment, data = simulated, na.action = na.exclude)
    # call for clsutered SE by divid

    output <- cluster_summary(fit, simulated$divid)
    print(output)
}
```

```
## [[1]]
##
## t test of coefficients:
##
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.386797   0.042994  8.9965   <2e-16 ***
## pre_totnorm 0.805338   0.025431 31.6676   <2e-16 ***
## treatment   0.026143   0.075375  0.3468   0.7287
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## [[2]]
## Wald test
##
## Model 1: post_totnorm ~ pre_totnorm + treatment
## Model 2: post_totnorm ~ 1
##   Res.Df Df      F    Pr(>F)
## 1   8681
## 2   8683 -2 519.03 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## [[3]]
##                Estimate    LowerCI   UpperCI
## (Intercept) 0.38679660  0.3025181 0.4710751
## pre_totnorm 0.80533772  0.7554870 0.8551885
## treatment   0.02614259 -0.1216100 0.1738951
```

Since the `balsakhi` dataset has many observations, we find that our simulated power estimates of MDE are very similar to those found through parametric estimates. To see this, compare Example 4 (parts 1 & 2) to Example 6 (parts 1 & 2). As you can see, it is much easier to cluster or control for covariates using this method; you simply run the regression specification that you intend to use at the analysis stage.

Similarly, if you wanted to estimate the (local) average treatment effect in the presence of partial take-up of the intervention, then you can simply make basic assumptions on the effective take-up rate, and run your two-stage least squares regression.

Generally speaking, parametric power calculations are great for back-of-the-envelope calculations of power, while non-parametric simulations provide more precise estimates of power when you have good baseline data.

Questions:

1. How is "non-parametric" power different from "parametric" power?
2. How do our parametric and non-parametric estimates of power compare? (Hint: Compare Example 4 to Example 6.)
3. When would you want to run parametric power calculations?
4. Non- paramteric simulations

Answers

Ex. 3, part 1:

1. 49%
2. Including controls cuts sample size almost in half.
3. Reduces MDE to a lesser extent (~28%).

Ex. 4, part 2:

1. Assignment is only random at the cluster level; thus we must cluster our standard errors in our main specification. To this end, our power calculations must also take this into account, since, by clustering in our main specification, we will lose all precision gained from intra-cluster correlation in outcomes.
2. Adding a new cluster of 5 individuals, since in the limiting case of ICC = 1, adding 5 individuals to previous clusters would have no effect on power, while adding 5 individuals in a new cluster would increase our effective `N` by 1.

Ex. 6:

1. Non-parametric power simulations do not require the assumption that beta coefficients take a normal distribution (which follows from the central limit theorem), while parametric power calculations do.
2. Our power simulations in Example 6 find that we are powered to detect an ~.20 SD increase in test scores without controls and a ~.14 SD increase in test scores when including a lagged dependent variable. Using parametric methods, Example 4 illustrates these are .19 SD and .13 SD, respectively. Thus, we find very similar results using each method.
3. We would want to run parametric power calculations when we want a quick, rough estimate of power, or when we do not have access to high quality baseline data.

4. We would want to run non-parametric simulations when we have access to high-quality baseline data and have reason to believe parametric assumptions may be violated. For example, we would prefer to run simulations if our randomization is complex (i.e. multiple randomizations), have a small sample size, or are running complex specifications.

Hint for aggregating results of regression simluation

Example is shown to tabulate the results of `coeftest`. Others can be done similarly:

```
# Define a function that will extract coefficients during the simulation
extract_estimates <- function(output) {
    class(output) <- "matrix"
    df_out <- data.frame(output)
    df_out$coefficient <- rownames(df_out)
    rownames(df_out) <- NULL
    return(df_out)
}

# simulation

# Standard significance level
alpha <- 0.05
# Number of simulations to conduct
sims <- 5

coefs <- data.frame()
for (i in 1:sims) {

    # We do a clustered random assignment using divid as the cluster
    simulated$treatment <- cluster_ra(clusters = simulated$divid)
    # View(simulated)

    # Do analysis (Simple regression)
    fit <- lm(post_totnorm ~ treatment, data = simulated, na.action = na.exclude)
    # call for clsutered SE by divid
    sim_output <- cluster_summary(fit, simulated$divid)
    sim_coef <- extract_estimates(sim_output[[1]])
    sim_coef$sim = i
    coefs <- rbind.data.frame(coefs, sim_coef)
}
coefs
```

```
##       Estimate Std..Error    t.value     Pr...t.. coefficient sim
## 1   0.40327181 0.08223064  4.9041553 9.553327e-07 (Intercept)   1
## 2   0.05249410 0.10427910  0.5034000 6.146958e-01   treatment   1
## 3   0.50165842 0.06489146  7.7307305 1.188628e-14 (Intercept)   2
## 4  -0.14760025 0.10365668 -1.4239338 1.545016e-01   treatment   2
## 5   0.38951406 0.06953937  5.6013460 2.191840e-08 (Intercept)   3
## 6   0.07890982 0.10499321  0.7515707 4.523296e-01   treatment   3
## 7   0.35271006 0.06282947  5.6137678 2.040617e-08 (Intercept)   4
## 8   0.14951248 0.10277761  1.4547184 1.457834e-01   treatment   4
## 9   0.43517613 0.08099109  5.3731360 7.938423e-08 (Intercept)   5
## 10 -0.01200440 0.10637906 -0.1128455 9.101556e-01   treatment   5
```

Similar data frames for aggregation can be made for other objects returned by `super.cluster.fun`, namely `w`(wald test) and `ci`(confidence intervals). Instead of using a loop, the same functionality can be achieved

through `apply` family of functions.