# MCLS Workshop: Computing Statistical Power in R Using Simulation

Hugo Lortie-Forgues

2023-06-04

## Installing and Loading Packages

Here I install and load a number of useful and highly popular R packages. Note that we only have to install the packages once, but we need to load them every time we use R. Remove the hashtag to install the packages.

```
# install.packages("tidyverse")
# install.packages("MASS")
# install.packages("broom")
# install.packages("lme4")
# install.packages("lmerTest")

library(tidyverse) #A helpful collection of R packages designed for data science.
library(MASS) #To create correlated variables
library(broom) #To more easily extract information from R outputs
library(lme4) #For mixed-effect models
library(lmerTest) #To obtain pvalues in mixed effect models
```

## Useful Functions & Concepts Used Throughout the Workshop

### Function: c() to combine

In R, the *c()* function is used to combine objects into a single vector or list.

Here are a few examples to illustrate the usage of the c() function:

```
vector <- c(1, 2, 3, 4, 5)
vector
```

```
## [1] 1 2 3 4 5
```

In this case, the c() function combines the individual elements 1, 2, 3, 4, and 5 into a single vector [1, 2, 3, 4, 5].

Combining multiple vectors into one:

```r
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
combined_vector <- c(vector1, vector2)
combined_vector
```

```
## [1] 1 2 3 4 5 6
```

Here, the c() function combines the elements of vector1 and vector2 into a single vector [1, 2, 3, 4, 5, 6].

## Function: rep() to replicate

The *rep()* function is used to create repeated copies of a vector, a value, or a set of values.

When using *times*, you specify the total number of repetitions of the entire object or vector.

```r
vector <- c(1, 2, 3)
rep(vector, times = 3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

When using *each*, you specify the number of repetitions for each element in the object or vector.

```r
vector <- c(1, 2, 3)
rep(vector, each = 3)
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

## Exercises:

- Using the functions shown above, write the following sequences of numbers
- 1 2 1 2 1 2 1 2
- 1 1 2 2 1 1 2 2
- 1 2 3 4 1 2 3 4 5 6 7 8 5 6 7 8

## Function: rnorm()

The function *rnorm()* is very useful; it allows us to draw a random sample of values from a normal distribution with a given mean and standard deviation. The function takes three arguments:

- n: the size of the sample
- mean: the mean of the normal distribution from which the sample will be drawn
- sd: the sd of the normal distribution from which the sample will be drawn

Importantly: every time we use the function, R returns a *different* sample of n values. Try with the code below, which randomly draws 10 values from a normal distribution with a mean of 0 and a SD of 1.

```
rnorm(n = 10, mean = 0, sd = 1)
```

```
##  [1]  0.7031012  1.0443090  0.4773204  0.4777502 -0.1081739 -0.9643979
##  [7]  1.1246452  0.8485686 -0.1413788 -1.7501463
```

Note that every time you execute the command, you obtain a different sample of data.

```
normal_Data <- rnorm(n = 100, mean = 0, sd = 1)

#Should be close to 0 (i.e., the mean of the population from which the sample is drawn)
mean(normal_Data)
```

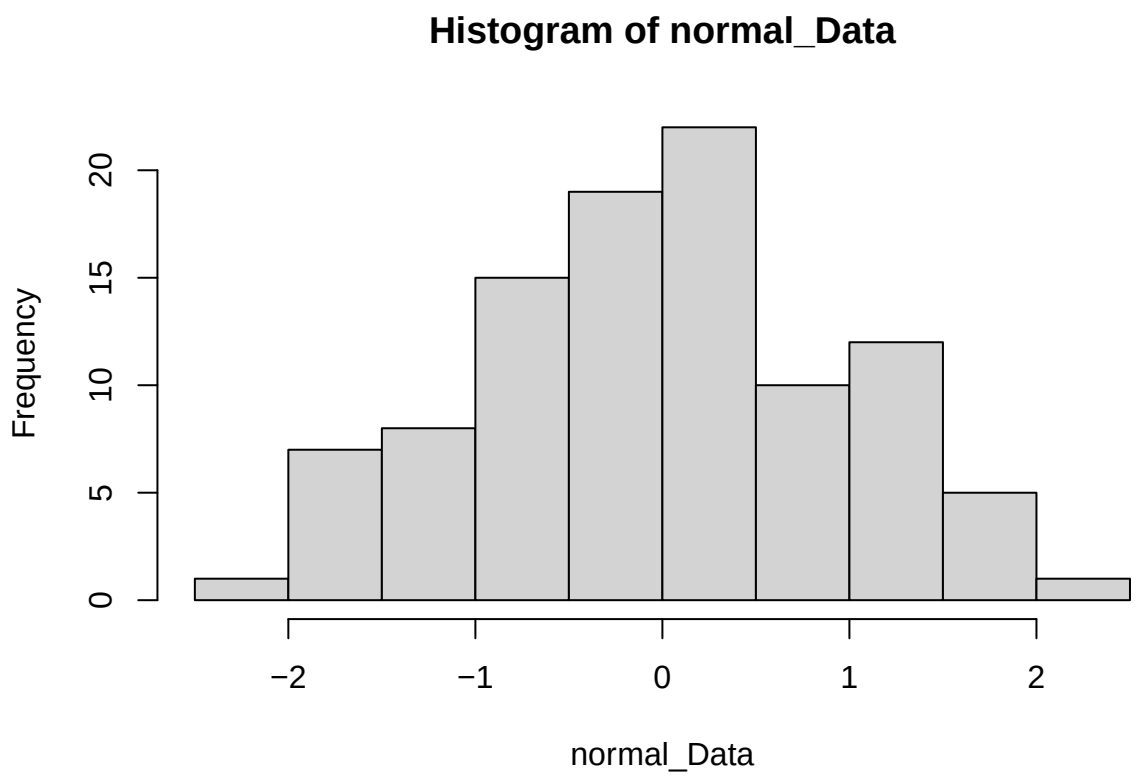
```
## [1] -0.02340089
```

```
#Should be close to 1 (i.e., the SD of the population from which the sample is drawn)
sd(normal_Data)
```

```
## [1] 0.9991313
```

The data in the *normal_Data* variable should also be approximately normal.

```
hist(normal_Data)
```




Note that there are comparable functions available for a range of distributions. For example:

- *rt()* returns values randomly sampled from a t-distribution
- *rlnorm()* returns values randomly sampled from a Log-normal distribution
- *rpois* returns values randomly sampled from a Poisson distribution

## Concept: For loop

The for loop is a type of looping construct that is used to execute a block of code repeatedly a specified number of times (n). For example, in the following, I print the word "hello" 10 times.

```
for (i in 1:10){

  print("hello")

}
```

```
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
## [1] "hello"
```

## Concept: Storing information in a for loop

It is often important to record information in a for loop. We can use a variable to store the information as the loop iterates. In the following, I create a container variable (named "container") that stores the value that "i" takes in the different iterations of the loop.

```
container <- NULL #Creating an empty container

for(i in 1:10){
  container <- c(container, i)
}

container #Here I print the values contained in the variable "container".
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Note: we can count the number of values that meets a certain condition. For example, here I count how many values in the variable "container" are below or equal to 5, and divide by the number of values in the container.

```
sum(container <= 5)/length(container)
```

```
## [1] 0.5
```

We will often use this method in the workshop to count the number of p-values that are below 0.05 (i.e., the number of p-values that are significant).

### Concept: Extracting information from an R output

R often reports lots of information. For example, when using the function *cor.test()*, R outputs the following information:

```
v1 <- rnorm(n = 10000, mean = 0, sd = 1)
v2 <- rnorm(n = 10000, mean = 0, sd = 1)

cor.test(v1,v2, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  v1 and v2
## t = 1.3459, df = 9998, p-value = 0.1784
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.006142435  0.033050606
## sample estimates:
##        cor
## 0.01345925
```

It is often useful to automatically extract specific information from an output. In the following, I extract the p-value from the output. I use the function *tidy* from the *broom* package.

```
tidy(cor.test(v1,v2, method = "pearson"))$p.value[1]
```

```
## [1] 0.1783612
```

## Example 1: Using simulation to calculate power for an Independent Sample t-test

In the following, I compute the power for an Independent Sample t-test, with a $N$ of 15, and a $d$ of 0.2

```
#Sample size
n_per_group <- 15

#Parameters
MeanGr1 <- 0
SDGr1 <- 1
MeanGr2 <- 0.2
SDGr2 <- 1

#Container
pvalues <- NULL


for(i in 1:1000){

  group1 <- rnorm(n = n_per_group, mean = MeanGr1, sd = SDGr1)
```

```
  group2 <- rnorm(n = n_per_group, mean = MeanGr2, sd = SDGr2)

  score <- c(group1, group2)
  groupID <- rep(c("gr1", "gr2"), each = n_per_group)

  MyData <- data.frame(groupID, score)


  t_model <- t.test(score ~ groupID, data = MyData)
  pvalues <- c(pvalues, tidy(t_model)$p.value[1])

}

#Power
sum(pvalues <= 0.05)/length(pvalues)
```

```
## [1] 0.08
```

### Exercises:

1. Modify the example above to effect size of d = 0.5, and compare what you obtain with G*Power.
2. Increase the sample size to 50 per group and see what happens to the power?
3. What happens to power if the sample size of the two groups is very different?
4. Modify the example above to conduct a Wilcoxon rank-sum test (non-parametric version of the t-test) Use the function **wilcox.test()**
5. Increase the number of simulated examples from 1000 to 10000 for more accurate results (but a longer running time).
6. Compute the power if the null hypothesis is true (i.e., the means of both groups are equal)
7. Draw a histogram of the p-value obtained when the null hypothesis is true. You can use **hist(pvalues)** to draw the histogram.
8. (Advanced) Modify the example so that the data is not coming from a normal distribution (you can use the **rlnorm()** function to generate data from a log-normal distribution).More info here

## Example 2: One-Way ANOVA

We can easily adapt Example 1 to compute the power for a one-way ANOVA. In the following, I compute the power for a one-way ANOVA with 3 groups.

```
#Sample Size
n_per_group <- 15

#Parameters
MeanGr1 <- 0
SDGr1 <- 1
MeanGr2 <- 0.2
SDGr2 <- 1
```

```r
MeanGr3 <- 0
SDGr3 <- 1

#Container
pvalues <- NULL

for(i in 1:1000){

  group1 <- rnorm(n = n_per_group, mean = MeanGr1, sd = SDGr1)
  group2 <- rnorm(n = n_per_group, mean = MeanGr2, sd = SDGr2)
  group3 <- rnorm(n = n_per_group, mean = MeanGr3, sd = SDGr3)

  score <- c(group1, group2, group3)
  groupID <- as.factor(rep(c("gr1", "gr2", "gr3"), each = n_per_group))

  MyData <- data.frame(groupID, score)

  anova.model <- aov(score ~ groupID, data = MyData)

  pvalues <- c(pvalues, tidy(anova.model)$p.value[1])

}

sum(pvalues <= 0.05)/length(pvalues)
```

```
## [1] 0.084
```

### Exercises:

- Explore how having widely different variance (or SD) between groups influences Type 1 error associated with a one-way ANOVA. Note that homogeneity of variances (i.e., assumption of equal variances, or homoscedasticity assumption) is an assumption of one-way ANOVA.

## Example 3: Correlation

In the following, I present a way to create correlated variables. I use the function *mvnorm* from the *MASS* package (installed and loaded earlier). This method can be easily expanded to create multiple correlated variables.

```r
#Sample size
n_observation <- 100

#Parameters
r_v1v2 <- 0.9 # value of correlation between v1 and v2 in the population

pop_rs <- matrix(c(1.00, r_v1v2,
                   r_v1v2, 1.00), nrow = 2, ncol = 2)


allVar <- mvrnorm(n = n_observation, mu = c(0,0), pop_rs)
var1 <- allVar[,1]
```

```r
var2 <- allVar[,2]

MyData <- data.frame(var1, var2)

cor.test(var1,var2, data = MyData)
```

```
##
##  Pearson's product-moment correlation
##
## data:  var1 and var2
## t = 25.101, df = 98, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8979151 0.9526198
## sample estimates:
##       cor
## 0.9302653
```

Here I'm using the code to examine the statistical power for a Pearson correlation with n = 100, and r = 0.3.

```r
#Sample Size
n_observation <- 100

#Parameters
r_v1v2 <- 0.3
pop_rs <- matrix(c(1.00, r_v1v2,
                   r_v1v2, 1.00), nrow = 2, ncol = 2)

#Container
pvalues <- NULL

for(i in 1:1000){

  allVar <- mvrnorm(n = n_observation, mu = c(0,0), pop_rs)
  var1 <- allVar[,1]
  var2 <- allVar[,2]

  MyData <- data.frame(var1, var2)

  cor_model <- cor.test(var1, var2, method = "pearson", data = MyData)

  pvalues <- c(pvalues, tidy(cor_model)$p.value[1])
}

sum(pvalues <= 0.05)/length(pvalues)
```

```
## [1] 0.874
```

## Exercises:

- Instead of a Pearson correlation, use a Spearman correlation (i.e., **method = "spearman"**). Is the statistical power similar?

- What happens to the power if we dichotomise var2 using a median split? *var2 <- ifelse(var2 > median(var2),1,0)*
- What happens to the power if we introduce random noise in the measurement of a variable (i.e., reduce the reliability of a measure)?
- What is the mean value of correlations that have met the criteria for statistical significance? How does it compare to the correlation value in the population?

## Example 4: Paired Sample t-test

The following code computes the statistical power for a paired sample t-test. Note that we have to specify the correlation between the two time-points, which here is set to r = 0.5.

```
##Sample Size
n_participant <- 10

#Population Mean for Each Cell in the Design
mean_time1 <- 0
mean_time2 <- 0.5

#Common Population Standard Deviation
sd_all <- 1

#Common population correlation among within-subjects factors
r_within <- 0.5

pop_rs <- matrix(c(1.00, r_within,
                   r_within, 1.00), nrow = 2, ncol = 2)

#Container
pvalues <- NULL

for(i in 1:1000){

  allVar<-mvrnorm(n_participant, mu = c(0,0), pop_rs, empirical = FALSE)

  time1 <- (allVar[,1] * sd_all) + mean_time1
  time2 <- (allVar[,2] * sd_all) + mean_time2

  Value <- c(time1, time2)

  Condition <- rep(c("T1", "T2"), each = n_participant)
  ID <- as.factor(rep(1:n_participant, times = 2))

  MyData <- data.frame(ID, Condition, Value)

  t_model <- t.test(Value ~ Condition, data = MyData, paired = TRUE)
  pvalues <- c(pvalues, tidy(t_model)$p.value[1])

}

#Power
sum(pvalues <= 0.05)/length(pvalues)
```

```
## [1] 0.309
```

### Exercise:

- What happens to the power of the test when the correlation between the two time-point decreases?
- What if you incorrectly use an Independent Sample t-test to analyse this data?

## Example 5: Repeated-measure ANOVA (three levels: T1, T2, T3)

The following code computes the power for a repeated-measure ANOVA. Just like with the paired-sample t-test, we have to specify the correlation between the different levels of the repeated measure (here we assume that they are the same, r = 0.20. Not doing so may violate the sphericity assumption of repeated-measures ANOVA).

```
##Sample Size
n_participant <- 10

#Population Mean for Each Cell in the Design
mean_time1 <- 3
mean_time2 <- 4
mean_time3 <- 4

#Common Population Standard Deviation
sd_all <- 1

#Common population correlation among within-subjects factors
r_within <- 0.2

rvalues <- matrix(c(1.00, r_within, r_within,
                    r_within, 1.00, r_within,
                    r_within, r_within, 1.00), nrow = 3, ncol = 3)

#Container
pvalues <- NULL

for(i in 1:100){

  allVar<-mvrnorm(n_participant, mu = c(0,0,0), rvalues, empirical = FALSE)
  time1 <- (allVar[,1] * sd_all) + mean_time1
  time2 <- (allVar[,2] * sd_all) + mean_time2
  time3 <- (allVar[,3] * sd_all) + mean_time3

  Value <- c(time1, time2, time3)
  Time <- rep(c("T1", "T2", "T3"), each = n_participant)
  ID <- as.factor(rep(1:n_participant, times = 3))

  MyData <- data.frame(ID, Time, Value)

  fitAnova<-aov(Value~Time + Error(ID/Time), data=MyData)
```

```
  pvalues <- c(pvalues, tidy(fitAnova)$p.value[2])

}

#Power
sum(pvalues <= 0.05)/length(pvalues)
```

```
## [1] 0.64
```

### Exercise:

- (Advanced) What happens to the power if you were treating participants as a random effect?

## Example 6: Factorial ANOVA (Two Between-Subject Variables)

The following code computes the power for a factorial ANOVA with two between-subject variables). The analysis yields 3 p-values, one for the main effect of variable 1, one for the main effect of variable 2, and one for the interaction between variable 1 and variable 2.

```
#Sample Size
n_per_cell <- 10

#Population Mean for Each Cell in the Design
mean_a1b1 <- 3
mean_a1b2 <- 2
mean_a2b1 <- 4
mean_a2b2 <- 5

#Common Population Standard Deviation
sd_all <- 1

#Container
pvalues_IV_a <- NULL
pvalues_IV_b <- NULL
pvalues_Interaction <- NULL

for(i in 1:100){

  IV_a <- as.factor(rep(c("a1", "a2"), each = n_per_cell*2))
  IV_b <- as.factor(rep(c("b1", "b2", "b1", "b2"), each = n_per_cell))

  a1b1 <- rnorm(n_per_cell, mean = mean_a1b1, sd = sd_all)
  a1b2 <- rnorm(n_per_cell, mean = mean_a1b2, sd = sd_all)
  a2b1 <- rnorm(n_per_cell, mean = mean_a2b1, sd = sd_all)
  a2b2 <- rnorm(n_per_cell, mean = mean_a2b2, sd = sd_all)

  DV <- c(a1b1, a1b2, a2b1, a2b2)

  MyData <- data.frame(IV_a, IV_b, DV)

  fitAnova <- aov(DV ~ IV_a * IV_b, data = MyData)
```

```
  pvalues_IV_a <- c(pvalues_IV_a, tidy(fitAnova)$p.value[1])
  pvalues_IV_b <- c(pvalues_IV_b, tidy(fitAnova)$p.value[2])
  pvalues_Interaction <- c(pvalues_Interaction, tidy(fitAnova)$p.value[3])

}

#Power IV a
sum(pvalues_IV_a <= 0.05)/length(pvalues_IV_a)
```

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

```
#Power IV b
sum(pvalues_IV_b <= 0.05)/length(pvalues_IV_b)
```

```
## [1] 0.02
```

```
#Power Interaction a x b
sum(pvalues_Interaction <= 0.05)/length(pvalues_Interaction)
```

```
## [1] 0.87
```

### Exercise:

- Compute the power associated with different patterns of interaction (e.g., Reversal vs Attenuation)
- Is it possible to have a significant interaction, but no significant main effects?

## Example 7: Mixed-design ANOVA (One Between-Subject and One Within-Subject Variable).

The following code computes the power associated with a Mixed-design ANOVA. Like in the repeated-measure ANOVA, we have to specify the correlation between the different levels of the Within-Subject Variable. Here we assume that the correlation is the same between each level, and we set it to r = 0.2.

```
#Sample Size
n_per_cell <- 10
n_total <- 20

#Population Mean for Each Cell in the Design
mean_a1_Time1 <- 5
mean_a1_Time2 <- 10
mean_a2_Time1 <- 15
mean_a2_Time2 <- 20

#Common correlation among within-subjects factors
corr <- 0.2

#Common Standard Deviation
```

```r
sd_var <- 10

#Containers
pvalueBetween <- NULL
pvalueWithin <- NULL
pvalueInteraction <- NULL

for(i in 1:100){

  rvalues <- matrix(c(1.00, corr, 0.00, 0.00,
                      corr, 1.00, 0.00, 0.00,
                      0.00, 0.00, 1.00, corr,
                      0.00, 0.00, corr, 1.00), nrow = 4, ncol = 4)

  allVar<-mvrnorm(n_per_cell, mu = c(0,0,0,0), rvalues, empirical = FALSE)

  a1_Time1 <- (allVar[,1] * sd_var) + mean_a1_Time1
  a1_Time2 <- (allVar[,2] * sd_var) + mean_a1_Time2
  a2_Time1 <- (allVar[,3] * sd_var) + mean_a2_Time1
  a2_Time2 <- (allVar[,4] * sd_var) + mean_a2_Time2

  Group <- as.factor(rep(c("A1", "A2"), each = n_per_cell*2))

  Time <- as.factor(rep(c("T1", "T2", "T1", "T2"), each = n_per_cell))

  Value <- c(a1_Time1, a1_Time2, a2_Time1, a2_Time2)

  #This creates a vector of 40 numbers: c(1 to 10, 1 to 10, 11 to 20, 11 to 20)
  ID <- as.factor(c(rep(1:n_per_cell, times = 2), rep((n_per_cell+1):n_total, times = 2)))

  MyData <- data.frame(ID, Group, Time, Value)

  rptAnova<-aov(Value~(Group*Time)+Error(ID/Time), data=MyData)

  pvalueBetween <- c(pvalueBetween, tidy(rptAnova)$p.value[1])
  pvalueWithin <- c(pvalueWithin, tidy(rptAnova)$p.value[3])
  pvalueInteraction <- c(pvalueInteraction, tidy(rptAnova)$p.value[4])

}

#Power Between Factor A
sum(pvalueBetween <= 0.05)/length(pvalueBetween)
```

```
## [1] 0.78
```

```r
#Power Within Factor Time
sum(pvalueWithin <= 0.05)/length(pvalueWithin)
```

```
## [1] 0.39
```

```r
#Power Interaction A x Time
sum(pvalueInteraction <= 0.05)/length(pvalueInteraction)
```

```
## [1] 0.07
```

## Example 8: Simple Regression

The following example computes the power for a simple linear regression. There are two p-values (one for the intercept, one for the slope), and as such, two values of power.

```
#Sample Size
n_observation <- 100

#Parameters
b0 <- 1 #Intercept
b1 <- 0.3 #Slope
res_se <- 2 #Residual Standard Deviation (or Residual Standard Error)

#Container
pvalues_b0 <- NULL
pvalues_b1 <- NULL
pvalue_F <- NULL


for(i in 1:1000){
  x <- rnorm(n = n_observation, mean = 0, sd = 1)
  y <- b0 + b1*x + rnorm(n = n_observation, mean = 0, sd = res_se)

  MyData <- data.frame(x, y)

  fit <- lm(y ~ x, data = MyData)
  pvalues_b0 <- c(pvalues_b0, tidy(fit)$p.value[1])
  pvalues_b1 <- c(pvalues_b1, tidy(fit)$p.value[2])
  pvalue_F <- c(pvalue_F, glance(fit)$p.value)

}

#Power B0
sum(pvalues_b0 <= 0.05)/length(pvalues_b0)
```

```
## [1] 0.999
```

```
#Power B1
sum(pvalues_b1 <= 0.05)/length(pvalues_b1)
```

```
## [1] 0.309
```

```
#Power F
sum(pvalue_F <= 0.05)/length(pvalue_F)
```

```
## [1] 0.309
```

## Example 9: Multiple Regression with Interaction

The following code computes the power for a multiple regression with an interaction term.

```
##Sample Size
n_observation <- 80

#population parameters
b0 <- 1 #Intercept
b1 <- 0.2 #Slope b1
b2 <- 0.3 #Slope b2
b3 <- 0.4 #Slope b3 (interaction)
res_se <- 1 #Residual Standard Deviation (or Residual Standard Error)

#containers
pvalues_b0 <- NULL
pvalues_b1 <- NULL
pvalues_b2 <- NULL
pvalues_b3 <- NULL
pvalue_F <- NULL

for(i in 1:100){
  x1 <- rnorm(n_observation, mean = 0, sd = 1)
  x2 <- rnorm(n_observation, mean = 0, sd = 1)
  y <- b0 + b1*x1 + b2*x2 + b3*x1*x2 + rnorm(n_observation, mean = 0, sd = res_se)

  MyData <- data.frame(x1, x2, y)

  fit <- lm(y ~ x1 * x2, data = MyData)

  pvalues_b0 <- c(pvalues_b0, tidy(fit)$p.value[1])
  pvalues_b1 <- c(pvalues_b1, tidy(fit)$p.value[2])
  pvalues_b2 <- c(pvalues_b2, tidy(fit)$p.value[3])
  pvalues_b3 <- c(pvalues_b3, tidy(fit)$p.value[4])
  pvalue_F <- c(pvalue_F, glance(fit)$p.value)

}

#Power B0
sum(pvalues_b0 <= 0.05)/length(pvalues_b0)
```

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

```
#Power B1
sum(pvalues_b1 <= 0.05)/length(pvalues_b1)
```

```
## [1] 0.35
```

```
#Power B2
sum(pvalues_b2 <= 0.05)/length(pvalues_b2)
```

```
## [1] 0.66
```

```r
#Power B3 (interaction)
sum(pvalues_b3 <= 0.05)/length(pvalues_b3)
```

```
## [1] 0.9
```

```r
#Power F
sum(pvalue_F <= 0.05)/length(pvalue_F)
```

```
## [1] 0.96
```

### Exercise:

- If all betas are 0 in the population, what is the probability that at least one estimated beta will significant?

## Example 10: Regression with Nested Data Using a Mixed-Effect Model

The following code computes power for a nested design, a typical design in educational RCTs where randomization is done at the classroom level.

```r
#Sample Size
N_Cluster <- 40
N_per_Cluster <- 100
N_Total <- N_per_Cluster * N_Cluster

#Parameters
effectSD <- 0.45
var_Cluster <- 1
var_Error <- 2
totalSD <- sqrt(var_Cluster + var_Error)
#population ICC
#var_Cluster/(var_Cluster + var_Error)

#Containers
pvalues <- NULL

for(i in 1:1000){#Increase for more accurate results (but a MUCH longer running time).

  #Condition Effect
  condition_ID <- as.factor(rep(c(1, 2), each = N_Total/2))
  condition_effect <- rep(c(0, (effectSD * totalSD)), each = N_Total/2)

  #Cluster Effect
  cluster_ID <- as.factor(rep(1:N_Cluster, each = N_per_Cluster))
  cluster_effect_values <- rnorm(n = N_Cluster, mean = 0, sd = sqrt(var_Cluster))
  cluster_effect <- rep(cluster_effect_values, each = N_per_Cluster)

  #Participant Effect
```

```
  individual_effect <- rnorm(n = N_Total, mean = 0, sd = sqrt(var_Error))

  #Dependent Variable
  y <- condition_effect + cluster_effect + individual_effect

  #Analysis
  MyData <- data.frame(condition_ID,
                       cluster_ID,
                       y)

  fit.rdm <- lmer(y ~ condition_ID + (1|cluster_ID), data = MyData)

  pvalues <- c(pvalues, summary(fit.rdm)$coefficients[2,5])

}
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00524864 (tol = 0.002, component 1)
```

```
#Power Condition Effect
sum(pvalues<=0.05)/length(pvalues)
```

```
## [1] 0.666
```

### Exercise:

- What happens to the power if you reduce the number of clusters, but increase the number of observations per cluster so that the total number of observations stays the same?

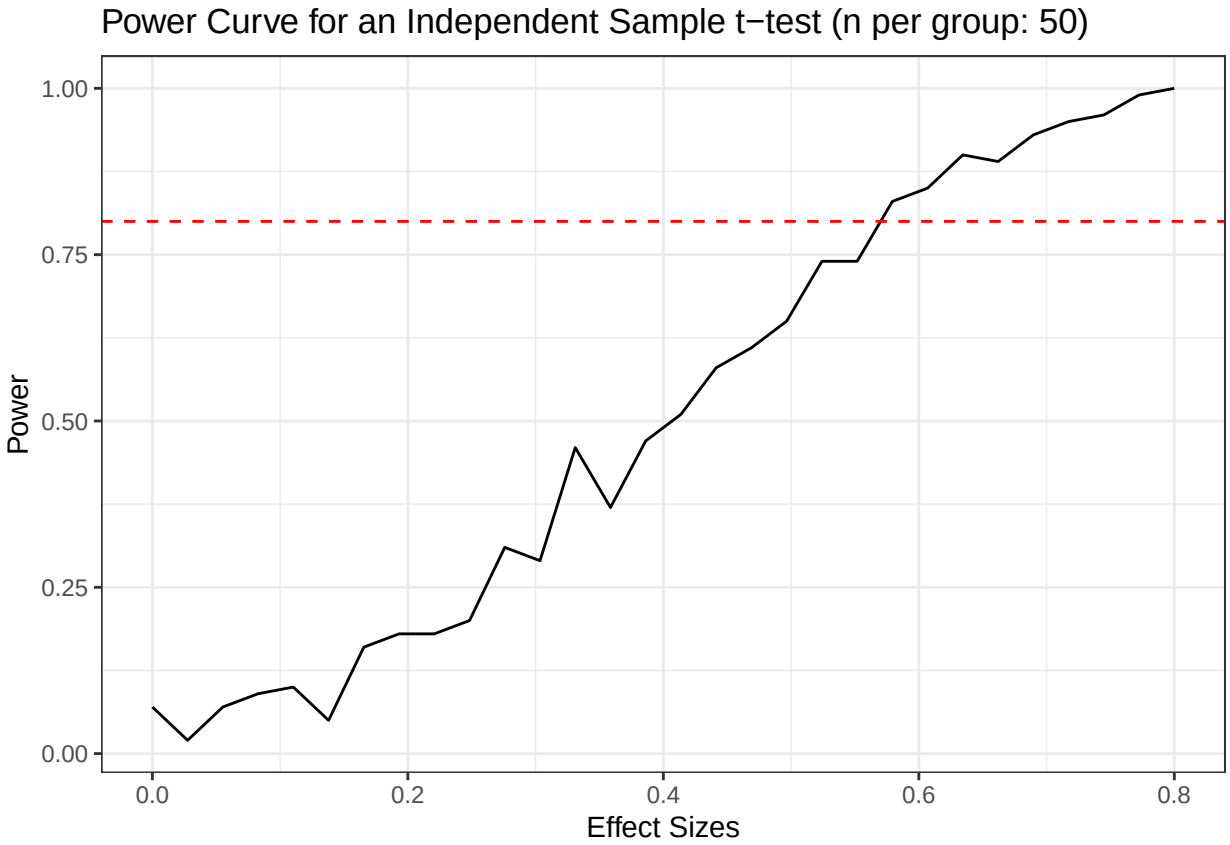
# APPENDIX

### Creating a power curve

In the following, I create a power curve (a curve displaying power for a range of effect sizes) for an independent sample t-test (i.e., Example 1). To do so, I embedded the code of Example 1 in a second for-loop that iterates over a range of population effect size. I then plot the power curve using ggplot. This code can easily be used with the other analyses presented in the workshop.

```
#Sample size
n_per_group <- 50

pop_es <- seq(from = 0, to = 0.8, length.out = 30) #creates sequences of numbers.

#Container power
powerValue <- NULL

for(es in pop_es){

#Container pvalues
pvalues <- NULL
```

```
  for(i in 1:100){

    group1 <- rnorm(n = n_per_group, mean = 0, sd = 1)
    group2 <- rnorm(n = n_per_group, mean = es, sd = 1)

    score <- c(group1, group2)
    groupID <- rep(c("gr1", "gr2"), each = n_per_group)

    MyData <- data.frame(groupID, score)
    t_model <- t.test(score ~ groupID, data = MyData)
    pvalues <- c(pvalues, tidy(t_model)$p.value[1])

  }

  #Power
  powerValue <- c(powerValue, sum(pvalues <= 0.05)/length(pvalues))

}

MyDataPower <- data.frame(pop_es, powerValue)

#Plot
ggplot(MyDataPower, aes(x=pop_es, y=powerValue)) +
  geom_line() +
  xlab("Effect Sizes")+
  ylab("Power")+
  theme_bw() +
  geom_hline(yintercept= 0.8, linetype="dashed", color = "red") +
  ggtitle("Power Curve for an Independent Sample t-test (n per group: 50)")
```


Power Curve for an Independent Sample t−test (n per group: 50)
Power
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
Effect Sizes