# Simulation-based sample size estimation - Tutorial

*Zoltan Kekecs*

*1/9/2021*

Abstract: In this tutorial we will cover the basic concepts related to simulation-based power analysis, and will demonstrate it's use through four examples.

## Loading reqired packages

```
library(MASS) # for murnorm()
library(lme4) # for lmer()
```

```
## Loading required package: Matrix
```

```
library(cAIC4) # for cAIC()
```

```
## Loading required package: stats4
```

## Simulating data

Simulation means that we generate data randomly based on some characteristics or assumption.

There are many functions in Base R that serve this purpose. Here are the basic ones:

**rbinom()** generates numbers from a binomial distribution with a given probability distribution

```
rbinom(n = 10, size = 1, prob = 0.5)
```

```
##  [1] 1 1 0 0 1 1 0 1 1 0
```

**sample()** draws "cases" from a "bag" of possible cases with given probability for drawing each case

```
sample(c("heads", "tails"), prob = c(0.3, 0.7), replace = T, size = 10) # draws "cases" from a bag of p
```

```
##  [1] "tails" "tails" "tails" "tails" "tails" "tails" "tails" "tails" "heads"
## [10] "tails"
```

**rnorm()** generates numbers from a univariate normal distribution with a given mean and sd

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

```
##  [1] -0.17584551 -1.85745706 -0.15161558 -0.02683101 -1.23305878  0.97917705
##  [7]  1.42326652  1.14156121 -0.54838597  0.46349027
```

## Using simulated data

People who are good in probability theory might be able to mathematically calculate the probability of certain events. For all other people, there is the brute force method. Simulation allows you to rest your brain and let the law of big numbers spit out a response for your question.

For example, let's say you have flipped a coin 10 times and the result is 9 "heads" and 1 "tails". Based on this results you suspect that the coin is biased, and you want to know what is the probability to get at least 9 heads with a fair coin.

You can take the "high road" and calculate this using a mathematical formula, like this:

```
1-pbinom(9-1, 10, 0.5)
```

```
## [1] 0.01074219
```

(We basically get the same answer when looking at the p-value of the binomial test below.)

```
binom.test(x = 9, n = 10, p = 0.5, alternative = "greater")
```

```
##
##  Exact binomial test
##
## data:  9 and 10
## number of successes = 9, number of trials = 10, p-value = 0.01074
## alternative hypothesis: true probability of success is greater than 0.5
## 95 percent confidence interval:
##  0.6058367 1.0000000
## sample estimates:
## probability of success
##                    0.9
```

Or you go take the "low road" and just flip a fair coin over and over again, and see how many times you get 9 or more heads in 10 coinflip series.

In the example below we simulate that we do 100.000 coin flip series, and in each series we flip the coin 10 times. then, we simply calculate the proportion of series in which we observed 9 or more "heads".

```
number_of_series =100000
size_of_series = 10
probability_of_heads = 0.5

number_of_heads = rbinom(n = number_of_series, size = size_of_series, prob = probability_of_heads)

sum(number_of_heads > 8)/100000 # proportion of coin-flip series with 9 or more "heads"
```

```
## [1] 0.01092
```

## Using simulation to find out the power of a research protocol

We can use the same logic to find out the power of a research protocol (design + analysis plan).

The **power** of a research protocol is the probability that when using that protocol we will correctly reject the null-hypothesis (H0), when the alternative hypothesis is true. In other words, statistical power is the probability that the protocol is able to detect the hypothesized effect (the alternative hypothesis is true)

### Steps to do simulation-based power analysis

1) **simulate realistic data with the effect**: use a code that generates data that is realistic to be observed in the population of interest using the research protocol
2) **run the statistical test**: run the proposed statistical test and note the decision drawn based on the test, for example "reject H0" (in other words, "support H1"), "retain H0".
3) **iterate**: repeat steps 1) and 2) a large number of times
4) **calculate the proportion of "support H1" decisions**: calculate the proportion of simulations in which the alternative hypothesis was supported, this is the "empirical", or observed, power of the protocol.

## Example 1: Coin flips

Sticking with the above example, we may be interested in the probability of being able to detect if a coin is biased when flipping the coin 10 times and using the one-tailed binomial test to reject testing the alternative hypothesis that heads are more likely than tails.

### 1) simulate realistic data with the effect

In order to give an answer to this question, we need to specify the extent of the bias we want to be detect. Let's say that the true probability of getting heads is 0.6 (instead of 0.5 that is expected by chance).

We can write a function that will simulate the data with this effect. The function below generates 10 data points from a binomial distribution where the probability of getting 1 is 0.6.

```
data_simulation_function <- function(number_of_flips, probability_of_heads){
  simulated_data = rbinom(number_of_flips, size = 1, prob = probability_of_heads)
  return(simulated_data)
}
```

### 2) run the statistical test

We will need to run the statistical test on the simulated data. It is important that we run the statistical test that we actually propose to do on the real data once we get the data. (So if you have important moderators you might have to simulate them as well.)

If we would like to use a one-tailed binomial test for statistical inference, we need to write a function that performs that on the data. Importantly, we need to store the decision or inference made based on this test in an object for later use. I store the decision "H1" in the "decision" object if the p-value is lower than 0.05, otherwise, the decision is "Inconclusive" (since in classical NHST, you can either reject the null or say that we don't have enough evidence yet to reject the null).

```
analysis_function <- function(data){
  number_of_successes = sum(data)
  number_of_observations = length(data)

  test_result = binom.test(x = number_of_successes, n = number_of_observations, p = 0.5, alternative =

  decision = if(test_result$p.value < 0.05){"H1"} else {"Inconclusive"}

  return(decision)
}
```

### 3) iterate

We need to be able to iterate this simulation-analysis process many times. The easiest way to do this is to write a new function that integrates the simulation and the analysis in one function:

```
simulation_and_analysis_function <- function(number_of_flips, probability_of_heads){
  simulated_data = data_simulation_function(number_of_flips = number_of_flips, probability_of_heads = p
  decision = analysis_function(data = simulated_data)
  return(decision)
}
```

Now that we have our integrated function, we can simply tell R to iterate this function many times.

In the code below I first specify the number of times the simulation-analysis process should be repeated. In simulation-based data analysis, we usually use **10.000 simulations** to make a decision, but this depends on the precision you want to achieve. You can play around with different iteration numbers and see how the result varies. In general, the more iterations, to less variability there should be between the results of two runs. If an event of interest is very rare (for example false positives), you might need more iterations to get a more precise estimate of exactly how rare they are.

I use the replicate() function to repeatedly run the simulation_and_analysis_function() and gather the results in one object. I save the results, the decisions in each simulation in an object called "all_decisions". If

you print the all_decisions object you will find that it contains the decisions (either "H1" or "Inconclusive") in a character vector.

```
number_of_iterations = 10000

all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips =
```

#### 4) calculate power

Finally, we calculate the proportion of times the procedure correctly supported the alternative hypothesis ("H1").

```
# statistical power

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.0464
```

The result should produce a number close to 0.046. This is the statistical power, indicating that the protocol was only successful at detecting the effect (the bias of the coin) in 4.6% of the 10.000 times we run it.

You can verify the result you got by running a "post-hoc" power calculation using GPower, a free power analysis tool: https://www.psychologie.hhu.de/arbeitsgruppen/allgemeine-psychologie-und-arbeitspsychologie/gpower.html. G*power returns a comparable result with power estimated as 0.046 (Test family: Exact, Statistical test: Proportion: Sign test (binomial test), effect size is 0.1 since the extent of the bias is 0.1 above the expected 0.5 probability by chance, alpha level = 0.05, smaple size = 10)

Based on this we need to realize that the procedure of flipping the coin only 10 times is not very effective at discerning a slightly biased coin (p = 0.6) from an unbiased coin using the binomial test.

## Optimize the protocol

There are a number of ways to improve power of the protocol in a real-life study. The most common approach is to increase the number of observations, and we are going to use this here for the sake of simplicity, but it is important to realize that there are many methods for improving power and adjusting sample size targets is not the only one, nor is it the best method.

## What is the optimal sample size?

One of the questions we can answer using this approach is **"What is the optimal sample size?"** for our study.

Let's see what would happen if we observed 100 coin flips.

When setting up the data_simulation_function we specified "number_of_flips" as one of the adjustable parameters. This is generally a good idea when writing the data simulation function, so you can adjust sample size.

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips =

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.6241
```

Power improved considerably, but it is still below commonly used standards on the field.

Another way of approaching this issue is to see what would be the number of observations that would result in the desired power. For example, let's say we would like to achieve power of 0.9 (90% chance to detect H1 if it is true).

We can adjust the sample size until by trial and error, we see that power the desired level.

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 1

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.8607
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 3

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.9654
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 1

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.9065
```

It seems that 220 observations would result in a power close to 0.9. This is close to the result we get from GPower if we switch to Apriori: compute required sample size, and specify a desired power of 0.9, which returs a total sample size of 213.

## What is the smallest detectable effect?

Another questions that can be answered by this approach is **"What is the smallest effect that my protocol can still reliably detect?"**. It is often the case that our resources are limited and we can only collect a certain amount of observations. We may be interested in what is the smallest effect we can hope to detect with our constraints in mind.

To answer this question, instead of adjusting the number of observations, we can adjust the effect size in our function, until we see that the observed power gets close to the desired power. This is the reason we specified the effect size (probability_of_heads) as an adjustable parameter in our function.

Lets say that we only have time to flip the coin 30 times and we need to make a decision at that point. What is the amount of bias that we are able to detect with this protocol with a 90% probability?

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 3

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.7305
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 3

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.973
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 3

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.897
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function(number_of_flips = 3

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.9112
```

Based on the results it seems that if we can only flip the coin 30 times, we can only hope to detect reliably (with 0.9 power) if the coin is so biased that it lands on heads with somewhere around 0.75-0.76 probability. this is verified by GPower if we switch the type of power analysis to "Sensitivity" and enter alpha = 0.05, power = 0.9, and sample size = 30, it returns an effect size of 0.252, which matches our estimation (0.5 + 0.252 = 0.752).

## Example 2: SuperIQ drug test

Here is another example for another research protocol: Let's say we would like to test the effectiveness of an experimental drug (superIQ) at improving cognitive performance. We want to randomly sample from the general public, participants randomly to two groups, one group gets the drug superIQ, while the other gets placebo, and we compare the average IQ score of the two groups after taking the drug.

Currently we have funding to run the study with N = 100 participants, and we suspect that the drug might increase IQ by 5 points. We would like to answer the following questions:

a) What is the power of our protocol with these parameters?
b) What would be the optimal sample size to achieve 90% power?
c) What is the minimal effect size that we can still detect with total N = 100 with 90% chance?

We will follow the same steps as above: **1. Simulate > 2. Analyze > 3. Iterate > 4. Calculate**.

### 1) Simulate

We know a lot about IQ in the general public, so we can hope to set up a realistic data generation function. We know that in the general public the IQ average is 100, and the SD of IQ is 15. We also know that IQ is normally distributed in this population. so we will use these parameters in the rnorm() function to generate data, and then add the effect.

```
data_simulation_function_group_means <- function(total_N, effect_of_drug_on_IQ){

  IQ_placebo_group = rnorm(n = total_N/2, mean = 100, sd = 15)
  IQ_treatment_group = rnorm(n = total_N/2, mean = 100+effect_of_drug_on_IQ, sd = 15)
  IQ_all = c(IQ_placebo_group, IQ_treatment_group)

  groups_labels = rep(c("placebo_group", "treatment_group"), each = total_N/2)

  simulated_data = data.frame(IQ = IQ_all, group = groups_labels)

  return(simulated_data)
}
```

### 2) Analyze

The proposed analysis is a one-tailed t.test (we hypothetise that the IQ in the treatment group will be higher than in the placebo control group). We write an analysis function.

```
analysis_function_group_means <- function(data){

  p_value = t.test(IQ ~ group, data = data, alternative = "less")$p.value

  decision = if(p_value < 0.05){"H1"} else {"Inconclusive"}

  return(decision)
}
```

### 3) Iterate

Now we integrate the data simulation and the analysis function into one.

```
simulation_and_analysis_function_group_means <- function(total_N, effect_of_drug_on_IQ){
  simulated_data = data_simulation_function_group_means(total_N = total_N, effect_of_drug_on_IQ = effec
  decision = analysis_function_group_means(data = simulated_data)
  return(decision)
}
```

We can run the simulation and analysis multiple times using this function. We entered total_N = 100 and an effect size of 5 in raw IQ units. Note that I use 1000 iterations here for the code to run faster, but when yu prepare work for publication, you should run at least 10000 simulations to get more precise answers. It is common to use lower iteration numbers initially when we fine-tune our protocol and code, and once we are fully satisfied, we would run the final simulation with the highest iteration number.

```
number_of_iterations = 1000

all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_
```

### 4) Calculate

We can now start answering the questions listed above:

#### a) What is the power of our protocol with its initial parameters (total_N = 100, effect size = 5)?

The analysis below indicates that the power is around 0.50. (Gpower returns comparable results, note that power is given in Cohen's d in Gpower, which is interpreted the effect size in standard deviation units. Since SD = 15 in this population, d is calculated as effect_of_drug_on_IQ/15. This translates to d = 0.333 in our initial case).

```
# statistical power

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.535
```

#### b) What would be the optimal sample size to achieve 90% power?

The analysis below indicates that the ideal sample size would be somewhere between 300 and 330 participants. (Gpower estimates 310 as the ideal sample size)

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.76
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.881
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.922
```

**c) What is the minimal effect size that we can still detect with total N = 100 with 90% chance?**

The analysis below indicates that this protocol can detect an effect size of 9 IQ points reliably (with roughly 90% chance). Gpower corroborates this: returning Cohen's d of 0.59 as the reliably detectable effect size, which translates to 0.59*15 = 8.85 IQ points.

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.834
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.957
```

```
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_group_means(total_

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.923
```

## Example 3: SuperIQ drug test, within-subjects variant

Now let's see an example with a within-subjects study design. We can imagine the same SuperIQ drug trial as described above, but with a two-staged measurement process. Everyne would undergo an IQ test before the intervention, then, people would get the drug, and do another IQ test 1 hour later. Here, we would expect that there would not only be a group difference between the groups, but also that IQ would increase compared to its previous level in the treatment group. I would also expect a slight increase in IQ in the placebo control group as well, due to the placebo effect and also because people get more familiar with the IQ test itself.

We can set out to answer the same questions:

a) What is the power of our protocol with these parameters?
b) What would be the optimal sample size to achieve 90% power?
c) What is the minimal effect size that we can still detect with total N = 100 with 90% chance?

As usual, we follow the same steps as above: **1. Simulate > 2. Analyze > 3. Iterate > 4. Calculate**.

### 1) Simulate

The difference in the data simulation code is that we need to account for the repeated measurements within each person.

We use the function mvrnorm() from the MASS package to generate correlated observations.

Note that aside from the previous assumptions about IQ, we also make the assumption that there is a very high correlation between a person's two consecutive IQ test resilts (r = 0.8).

```
data_simulation_function_within_means <- function(total_N, effect_of_drug_on_IQ){

r = 0.8
mean_IQ_baseline = 100
sd_IQ = 15
placebo_effect = 2
familiarity_effect = 1
drug_effect = effect_of_drug_on_IQ
```

```
IQ_pre = mvrnorm(n = total_N,
                      mu = c(0, 0),
                      Sigma = matrix(c(1, r,
                                       r, 1), nrow = 2))

IQ_pre_placebo_group = IQ_pre[1:(total_N/2),]
IQ_pre_treatment_group = IQ_pre[((total_N/2)+1):total_N,]

IQ_baseline_placebo_group = IQ_pre_placebo_group[,1]*sd_IQ + mean_IQ_baseline
IQ_posttest_placebo_group = IQ_pre_placebo_group[,2]*sd_IQ + mean_IQ_baseline + placebo_effect + familia


IQ_baseline_treatment_group = IQ_pre_treatment_group[,1]*sd_IQ+mean_IQ_baseline
IQ_posttest_treatment_group = IQ_pre_treatment_group[,2]*sd_IQ + mean_IQ_baseline + placebo_effect + fa

IQ_all = c(IQ_baseline_placebo_group, IQ_posttest_placebo_group, IQ_baseline_treatment_group, IQ_posttes

groups_labels = rep(c("placebo_group", "treatment_group"), each = total_N)

time_labels = rep(rep(c("baseline", "posttest"), each = total_N/2), 2)

ID_labels = c(rep(paste0("ID_", 1:(total_N/2)), 2), rep(paste0("ID_", ((total_N/2)+1):total_N), 2))

simulated_data = data.frame(ID = ID_labels, IQ = IQ_all, group = groups_labels, time = time_labels)

  return(simulated_data)
}
```

### 2) Analyze

In this classical pre-post mixed design we are looking for a significant group*time interaction. We use a mixed linear model with random intercept of participant ID, building two models where in the "full model"" we include group, time and their interaction as fixed effect predictors, while in the "control model" we only take into account the main effect of group and time without their interaction. If the full model has a lower cAIC by at least 2, the decision is to support "H1" (that the intervention was effective at increasing IQ).

We write a function that analyzes the data and makes this decision.

```
analysis_function_within_means <- function(data){

  mod_with_interaction = lmer(IQ ~ group * time + (1|ID), data = data)
  mod_without_interaction = lmer(IQ ~ group + time + (1|ID), data = data)

  cAIC_dif = cAIC(mod_without_interaction)$caic - cAIC(mod_with_interaction)$caic

  decision = if(cAIC_dif > 2){"H1"} else {"Inconclusive"}

  return(decision)
}
```

### 3) Iterate

Now we integrate the data simulation and the analysis function into one.

```r
simulation_and_analysis_function_within_means <- function(total_N, effect_of_drug_on_IQ){
  simulated_data = data_simulation_function_within_means(total_N = total_N, effect_of_drug_on_IQ = effe
  decision = analysis_function_within_means(data = simulated_data)
  return(decision)
}
```

We can run the simulation and analysis multiple times using this function. We entered total_N = 100 and an effect size of 5 in raw IQ units.

I use 1000 iterations here for the code to run faster, but still it runs for roughly 1 mins, due to the mixed models analysis used.

```r
number_of_iterations = 1000

all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_within_means(total
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00577269 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0346006 (tol = 0.002, component 1)
```

#### 4) Calculate

We can now start answering the questions listed above:

#### a) What is the power of our protocol with its initial parameters (total_N = 100, effect size = 5)?

The analysis below indicates that the power is around 0.83. (The simulation now takes into account a level of complexity that is harder to set up in Gpower, because both groups experience some effect here.)

```r
# statistical power

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.856
```

#### b) What would be the optimal sample size to achieve 90% power?

The optimal total sample size is somewhere around 120.

```r
# statistical power
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_within_means(total
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00613628 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00861708 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00666708 (tol = 0.002, component 1)
```

```r
sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.908
```

#### c) What is the minimal effect size that we can still detect with total N = 100 with 90% chance?

It seems that we can detect the effect reliably if it is at least 5.5 IQ points over and above the effect experienced in the other group.

```
# statistical power
all_decisions = replicate(n = number_of_iterations, simulation_and_analysis_function_within_means(total
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00540307 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00628758 (tol = 0.002, component 1)
```

```
sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.903
```

## Example 4: Linear regression-based simulation

In this study we would like to assess the effect of getting negative feedback about cognitive abilities on self reported stress. Let's say we are replicating a previous study where the researchers used the following protocol: People were told that they would complete a new type of IQ test. In the end (irrespective of actual performance) people were divided into 3 groups randomly. Group0 was simply told a score without anchoring it to other's performance, Group1 got the feedback that they performed poorly, both their score, and their time was below average, and Group2 got the feedback that they performed below average but their solution time for correct responses was better than average. After the intervention, participants were asked to indicate their perceived stress level on a 0-7 scale. The researchers reported a small effect for the feedback in Group1 compared to Group0, where Group1 reported higher stress levels. They also reported an even smaller effect in Group2 (again, the feedback increased stress to some extent), an even smaller effect in Group2. They also reported a gender-effect, where women reported lower stress levels overall, and an IQ effect, where those with higher IQs reported more stress. There was no interaction effect.

Now we would like to perform a replication of this study, and would like to see what is the ideal sample size for this study.

Fortunately, the authors published their regression analysis results in detail, so we can simulate realistic data.

They reported the following regression coefficients (b-values):

SD of stress in the population = 0.7,

intercept (Group0, Man) = 6,

group: Group1 = 0.3,

group: Group2 = 0.15,

gender: Woman = -0.4

IQ: -0.02

We don't have sub-group means, but fortunately, we can use the regression coefficients to reverse-engineer the data and make a realistic simulation.

### 1) Simulate

Note that in the code below we simulate the effects of the different predictors based on the regression coefficients reported by the researchers. These are paramteres that we feed into the data simulation function, so we can adjust them later during the analysis.

In the code below we only implement categorical predictors, but we could just as easily simulate the effect of a continuous predictor using it's regression coefficient, if we know the mean

```
data_simulation_function_regression = function(intercept, sd_outcome, coefficient_group1, coefficient_g
```

```
  outcome_at_intercept = rnorm(mean = intercept, sd = sd_outcome, n = N_per_group*3)
  IQ = rnorm(mean = 100, sd = 15, n = N_per_group*3)
  gender = rbinom(n = N_per_group*3, size = 1, prob = 0.5)
  group = rep(c("group0", "group1", "group2"), each = N_per_group)

  # Dummy coding group

    group_1 = group
    group_2 = group

    group_1[which(group == "group1")] = 1
    group_1[which(group != "group1")] = 0

    group_2[which(group == "group2")] = 1
    group_2[which(group != "group2")] = 0

    group_1 = as.numeric(group_1 )
    group_2 = as.numeric(group_2 )


  outcome = outcome_at_intercept + gender * coefficient_gender_female + group_1 * coefficient_group1 + g

  data = data.frame(outcome = outcome, gender = gender, group = group, IQ = IQ)
  return(data)
}
```

**2) Analyze**

Here we specify the main analysis of interest as the significance test of the regression coefficient of the group1 (vs. group0, which is the default level). We make the decision based on the p-value corresponding to this coefficient.

```
data_analysis_function_regression <- function(data){
  mod_grp_full_sim = lm(outcome ~ group + gender + IQ, data =data)
  p_value = summary(mod_grp_full_sim)$coefficients[2,4]

  decision = if(p_value < 0.05){"H1"} else {"Inconclusive"}

  return(decision)
}
```

**3) Iterate**

Integrating the simulation and analysis function

```
simulation_and_analysis_function_regression <- function(intercept, sd_outcome, coefficient_group1, coefi
  simulated_data = data_simulation_function_regression(intercept, sd_outcome, coefficient_group1, coeffi
  decision = data_analysis_function_regression(data = simulated_data)
  return(decision)
}
```

And performing the iterated analysis.

```
number_iterations = 1000

all_decisions = replicate(n = number_iterations,
```

```
                          simulation_and_analysis_function_regression(
                                                    sd_outcome = 0.7,
                                                    intercept = 6,
                                                    coefficient_group1 = 0.3,
                                                    coefficient_group2 = 0.15,
                                                    coefficient_gender_female = -0.4,
                                                    coefficient_IQ = -0.02,
                                                    N_per_group = 80))
```

**4) Calculate**

With N = 80, the power is around 80%.

```
# statistical power

sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.758
```

The optimal sample size seems to be around N = 120.

```
all_decisions = replicate(n = number_iterations,
                          simulation_and_analysis_function_regression(
                                                    sd_outcome = 0.7,
                                                    intercept = 6,
                                                    coefficient_group1 = 0.3,
                                                    coefficient_group2 = 0.15,
                                                    coefficient_gender_female = -0.4,
                                                    coefficient_IQ = -0.02,
                                                    N_per_group = 90))
sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.819
```

```
all_decisions = replicate(n = number_iterations,
                          simulation_and_analysis_function_regression(
                                                    sd_outcome = 0.7,
                                                    intercept = 6,
                                                    coefficient_group1 = 0.3,
                                                    coefficient_group2 = 0.15,
                                                    coefficient_gender_female = -0.4,
                                                    coefficient_IQ = -0.02,
                                                    N_per_group = 110))
sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.891
```

```
all_decisions = replicate(n = number_iterations,
                          simulation_and_analysis_function_regression(
                                                    sd_outcome = 0.7,
                                                    intercept = 6,
                                                    coefficient_group1 = 0.3,
                                                    coefficient_group2 = 0.15,
                                                    coefficient_gender_female = -0.4,
                                                    coefficient_IQ = -0.02,
                                                    N_per_group = 120))
```

```r
sum(all_decisions == "H1")/length(all_decisions)
```

```
## [1] 0.919
```