SIMULATIONS OF POWER ANALYSIS FOR COMPLEX EXPERIMENTAL DESIGNS IN R

Giulio Costantini
14/05/2021

Preliminary Operations

Download slides here: https://bit.ly/33IOro8

These slides assume that a recent version of R (4.0.5) and Rstudio (1.4.1103) are installed on your computer.

- R https://cran.r-project.org/.
- Rstudio https://www.rstudio.com/

Here I include a list of required packages and the code to automatically install and load them. Command *set.seed* ensures that all simulation results replicate exactly each time the slides are compiled.

```
if(!require("pacman")) install.packages("pacman")
pacman::p_load("MASS", "psych", "dplyr", "ggplot2", "reshape2",
               "powerMediation", "bmem", "shiny", "lme4",
               "lmerTest", "simr", "prepdat")
set.seed(1)
load("data/BKP.RData")
```

The slides embed R code to get all results shown. However, I have pre-run and saved some results that would require a long time to run. Set eval = TRUE in an Rstudio chunk if you want to run it.

Most of this presentation (particularly the part about mediation) is based on Perugini, Gallucci, & Costantini (2018).

Power and Simulation

The idea behind using simulation for power analysis is quite straightforward. If power of a **statistical test** is the probability of successfully rejecting H0 if H1 is true, one can determine power by

1. defining the expected values of the population parameters under H1
2. generating a sample of size N from the population parameters
3. testing the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. estimating power as the proportion of simulated samples in which H0 is rejected

Power and Simulation

The necessary ingredients are thus

- a set of values of population parameters under H1 (~effect size),
- a method for simulating data,
- a statistical test for deciding whether to reject H0.

Todays roadmap

1. Power analysis for a **two-sample between subject t-test**. This is of course not useful per se (analytic methods work nicely), but it allows understanding the general logic behind power simulations and it gives you the instruments for implementing your own simulations.
2. Power anlaysis for **mediation**.
3. Power analysis for **mixed models**

Between-subject t-test

Power for a t.test in R - analytic solution

In R, we can estimate power for a two-sample with function *power.t.test*. Given three among \(\alpha \), *effect size* (in terms of difference between means and sd), *sample size*, and *power*, this function returns the fourth element. For example, the following code estimates power in a between-subject two-sample t-test under the following conditions:

- A difference between means equal to *delta* = 0.50 and a standard deviation of *sd* = 1 (corresponding to a Cohen's *d* = .50)
- A *two-sided* t-test at the conventional level \(\alpha = .05 \).
- *n* = 64 subjects by group

```
power.t.test(sig.level = .05,
             delta = .5,
             sd = 1,
             n = 64,
             power = NULL)
```

Power for a t.test in R - analytic solution

```
power.t.test(sig.level = .05,
             delta = .5,
             sd = 1,
             n = 64,
             power = NULL)
```

```
     Two-sample t test power calculation

              n = 64
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.8014586
    alternative = two.sided

NOTE: n is number in *each* group
```

Steps

Let's try and reproduce the same results using simulation!

1. define the expected values of the population parameters under H1
2. generate a sample of size N from the population parameters
3. test the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. estimate power as the proportion of simulated samples in which H0 is rejected

Steps

Let's try and reproduce the same results using simulation!

1. **define the expected values of the population parameters under H1**
2. generate a sample of size N from the population parameters
3. test the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. estimate power as the proportion of simulated samples in which H0 is rejected

Define population parameters under H1 (1)

Let's assume a bivariate normal distribution with *SDs* = 1 and *cor* = 0, one variable has *M* = 0 and the other has *M* = 0.5.

The following code creates a vector of means called **mu** and a covariance matrix called **Sigma** that describe our population parameters.

```
mu <- c(0, .5)
Sigma <- matrix(c(1, 0,
                  0, 1),
                ncol = 2)
```

Define population parameters under H1 (1)

```
mu
```

```
[1] 0.0 0.5
```

```
Sigma
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Steps

1. define the expected values of the population parameters under H1
2. **generate a sample of size N from the population parameters**
3. test the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. estimate power as the proportion of simulated samples in which H0 is rejected

Generate a sample (1)

R offers several packages and functions to simulate data from a multivariate normal distribution.

For example, function **mvrnorm** in package *MASS* requires as input a sample size *n*, a vector of means *mu*, and a covariance matrix *Sigma* as those that we have just created (type *?mvrnorm* in the R console for info). We can save the simulated data in a matrix called **simData**.

```
simData <- MASS::mvrnorm(n = 64,
                         mu = mu,
                         Sigma = Sigma)
```

We can use function *head* to inspect the first 6 cases of **simData**.

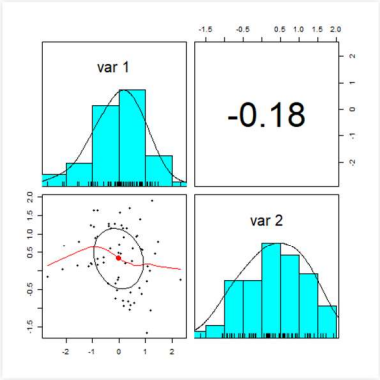
```
head(simData)
```

```
           [,1]       [,2]
[1,]  0.08607881  1.2309363
[2,] -0.58638186  0.2259920
[3,] -0.81503297  0.9497791
[4,]  0.68771552  0.9534433
[5,]  1.05743079 -1.6429211
[6,] -0.80565896  0.1548838
```

Generate a sample (2)

Function pairs.panels allows visualizing the data as well as their correlation

```
pairs.panels(simData)
```



We can use function *colMeans* to compute the means of the two variables. *round* is used to round the result to two decimals.

```
colMeans(simData) %>%
  round(2)
```

```
[1] -0.03  0.34
```

Tip. "%>%" is a function in package *dplyr* that allows using the output of a function as the first input of the subsequent function, resulting in tidier coding.

Steps

1. define the expected values of the population parameters under H1
2. generate a sample of size N from the population parameters
3. **test the significance of the target effect using the preferred statistical method**
4. replicate steps 2 and 3 a large number of times
5. estimate power as the proportion of simulated samples in which H0 is rejected

Test for significance (1)

A t.test can be performed using function *t.test*. In the following code, the results are saved to a variable called *tt*. Setting *var.equal = TRUE* ensures that a Student t-test is performed as opposed to a Welch t.test (for more information type *?t.test*)

```
tt <- t.test(x = simData[,1],
             y = simData[,2],
             alternative = "two.sided",
             var.equal = TRUE)
tt
```

```
    Two Sample t-test

data:  simData[, 1] and simData[, 2]
t = -2.3616, df = 126, p-value = 0.01973
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.67671412 -0.05965236
sample estimates:
  mean of x   mean of y
-0.02574118  0.34244206
```

Test for significance (2)

We can extract the p-value from variable *tt* using the following code

```
tt$p.value
```

```
[1] 0.01972947
```

We can contrast it to the conventional value $\alpha = .05$ to check whether the statistical test suggests to reject H0 (**TRUE**) or not (**FALSE**).

```
tt$p.value < .05
```

```
[1] TRUE
```

Notice that in R (as in other languages) the logicals *FALSE* and *TRUE* can be treated as if they were numeric values of 0 and 1 respectively. To count instances in which some operation gives a result *TRUE*, one can just sum logicals.

Steps

Let's try and reproduce the same results using simulation!

1. define the expected values of the population parameters under H1
2. generate a sample of size N from the population parameters
3. test the significance of the target effect using the preferred statistical method
4. **replicate steps 2 and 3 a large number of times**
5. estimate power as the proportion of simulated samples in which H0 is rejected

Replicate steps 2 and 3 (1)

Now let's replicate steps 2 and 3 1000 times. This time we will save our p-values in a vector, called *p.values*

First we create an empty vector called *p.values*

```
p.values <- c()
```

Replicate steps 2 and 3 (2)

Second, we wrap the code we developed so far within a for loop with $B = 1000$ replications. The last row within the loop stores the p-value in vector p.values.

```
B <- 1000
for(i in 1:B)
{
  simData <- mvrnorm(n = 64,
                     mu = mu,
                     Sigma = Sigma)

  tt <- t.test(x = simData[,1],
               y = simData[,2],
               alternative = "two.sided",
               var.equal = TRUE)

  p.values <- c(p.values, tt$p.value)
}
```

Note. Albeit R offers faster alternatives to *for* loops (e.g., *lapply*, *apply* etc.), I use loops here, because what they do is immediately clear.

Steps

1. define the expected values of the population parameters under H1
2. generate a sample of size N from the population parameters
3. test the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. **estimate power as the proportion of simulated samples in which H0 is rejected**

Estimate power

We can estimate power simply as the proportion of simulated samples in which H0 is rejected. If we use the conventional alpha level of .05, power is simply the proportion of our $B = 1000$ cases in which $p < .05$.

$$\frac{\sum_{i = 1}^B (p_i < .05)}{B}$$

We expect this number to be close to our the value of .80 that we estimated using the analytic procedure above. The higher B, the higher the accuracy of our power estimate

```
sum(p.values < .05) / B
```

```
[1] 0.807
```

Extensions

Given three elements among power, sample size, effect size, and alpha level, one can get the fourth. We used simulation to estimate power from the other three elements. In Gpower, this corresponds to **Post hoc** power.

The other two most widespread scenarios are:

- **A priori** power analysis: N is unknown.
- **Sensitivity** power analysis: effect size is unknown.

The simplest solution is performing a simulation for post-hoc power analysis under different scenarios, including different sample sizes and/or effect sizes and then inspecting which scenarios offer enough power.

Of course, this can easily become computationally cumbersome.

Extensions: A priori power, N unknown (1)

In the following code, we explore several possible values of N (from 10 to 120 in steps of 10).

We first create a vector **N** which includes the desired sample sizes, each one repeated *B* times using function *rep*. The *for* loop is now around N, each time data are simulated with a different sample size $n \in N$.

```
p.values <- c()
B <- 1000
N <- seq(from = 10, to = 120, by = 10) %>%
  rep(each = B)

for(n in N)
{
  simData <- mvrnorm(n = n, # <- notice the varying sample size
                     mu = mu,
                     Sigma = Sigma)

  tt <- t.test(x = simData[,1],
               y = simData[,2],
               alternative = "two.sided",
               var.equal = TRUE)

  p.values <- c(p.values, tt$p.value)
}
```

Extensions: A priori power, N unknown (2)

We can assemble the results in a dataframe.

Within the dataframe, we include a variable, called **Sig.res**, indicating whether each result turned out significant (p < \(\alpha \)) or not.

```
results <- data.frame(N, p.values)
results <- mutate(results, sig.res = p.values < .05)
head(results)
```

```
   N  p.values sig.res
1 10 0.3122438   FALSE
2 10 0.5559785   FALSE
3 10 0.2883535   FALSE
4 10 0.0217494    TRUE
5 10 0.4288805   FALSE
6 10 0.2542485   FALSE
```

Tip. *mutate* is a function in the R package *dplyr* that computes new variables in a dataframe.

Extensions: A priori power, N unknown (3)

We can compute power as the proportion of times a significant result is observed. However, this time we need to compute power separately for each N. We save the result in a dataframe called **smm**.

```
smm <- group_by(results, N) %>%
  summarize(power.sim = mean(sig.res))
smm
```

Tip. Function *group_by* in package *dplyr*, when combined with *summarize*, allows performing summary statistics on groups of cases. In this case, the grouping variable is N and the summary statistic is simply the mean of sig.res, which is equivalent to the proportion of times a result turned out significant.

Extensions: A priori power, N unknown (3)

```
smm <- group_by(results, N) %>%
  summarize(power.sim = mean(sig.res))
smm
```

```
# A tibble: 12 x 2
       N power.sim
 * <dbl>     <dbl>
 1    10     0.182
 2    20     0.303
 3    30     0.477
 4    40     0.634
 5    50     0.696
 6    60     0.792
 7    70     0.827
 8    80     0.874
 9    90     0.908
10   100     0.941
11   110     0.949
12   120     0.976
```

Extensions: A priori power, N unknown (4)

Let's contrast our results with those obtained analytically using *power.t.test* Notice that power.t.test can take as input also a vector for each parameter. In this case, it will estimate the unknown parameter under all conditions

```
an.res <- power.t.test(sig.level = .05,
                                    delta = .5,
                                    sd = 1,
                                    n = smm$N)
smm$power.analytic <- an.res$power
smm
```

```
# A tibble: 12 x 3
       N power.sim power.analytic
   <dbl>     <dbl>          <dbl>
 1    10     0.182          0.184
 2    20     0.303          0.338
 3    30     0.477          0.478
 4    40     0.634          0.598
 5    50     0.696          0.697
 6    60     0.792          0.775
 7    70     0.827          0.836
 8    80     0.874          0.882
 9    90     0.908          0.916
10   100     0.941          0.940
11   110     0.949          0.958
12   120     0.976          0.971
```

Extensions: A priori power, N unknown (5)

We can also plot the results. First we can convert the smm data in long format using function *melt* in package *reshape2* and save the result in a new dataframe, called **smm_melt**.

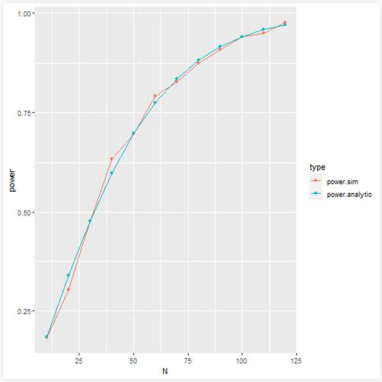
```
smm_melt <- melt(smm,
                 id.vars = "N",
                 value.name = "power",
                 variable.name = "type")
head(smm_melt)
```

```
   N      type power
1 10 power.sim 0.182
2 20 power.sim 0.303
3 30 power.sim 0.477
4 40 power.sim 0.634
5 50 power.sim 0.696
6 60 power.sim 0.792
```

Extensions: A priori power, N unknown (6)

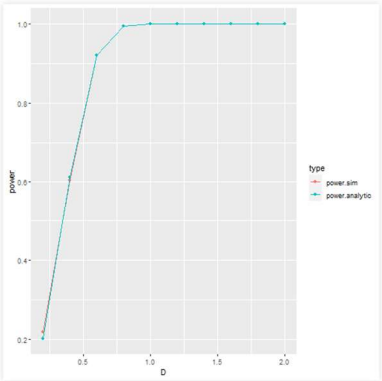
Second, we can visualize them in ggplot.

```
ggplot(smm_melt, aes(x = N, y = power, color = type)) +
  geom_point() +
  geom_line()
```



Extensions: Sensitivity, explore different effect sizes

Homework. Explore different values of Cohen's *d* (es. .2, .4, .6, ..., 2), by keeping the sample size fixed to N = 64 in each group. You should be able to produce the plot showed here.



- Tip 1: Most code is just copy-paste from above.
- Tip 2: If the standard deviation is equal to 1, Cohen's d is just the difference between the means of the two groups (e.g., if you assign a group mean = 0 and the other = 1, you get *d* = 1).
- Tip 3: in R there are always many ways to do something. If you want to see how I did it, just check the code running behind this slide and you'll find a solution. Your code does not need to be identical to mine as long as you get the correct result.

Extensions to other analyses

The method that we have just seen is not limited to the between-subject t-test, but it can be applied in any situation in which we have the necessary ingredients

- A set of values of population parameters under H1 (~effect size),
- a method for simulating data,
- a statistical test for deciding whether to reject H0.

Furthermore, R packages have been implemented that save you the burden of programming simulations from scratch. In the next slides, we will see some examples of how this logic can be applied to more complex situations.

- Mediation analysis
- Mixed models

Mediation analysis

Refresher on mediation analysis (1)

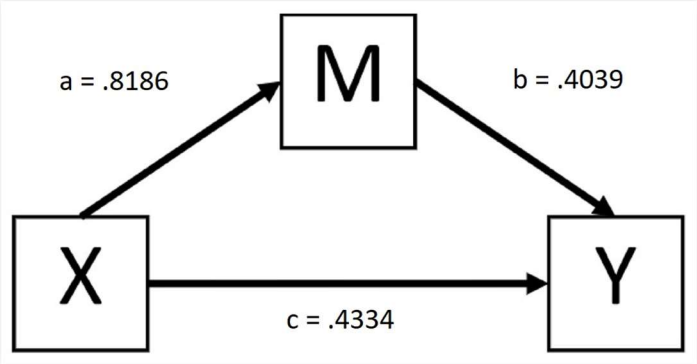
In mediation analysis, we are interested in the indirect effect that a predictor X has on a response Y through a mediator M. Within Baron & Kenny's (1986) framework, the indirect effect can be estimated as the product of two path coefficients, $a*b$. We will assume that all variables are standardized with $M = 0$ and $SD = 1$.

Coefficient a is the slope of a simple linear regression

$M = aX$

Coefficients b and c are the slopes of a multiple regression

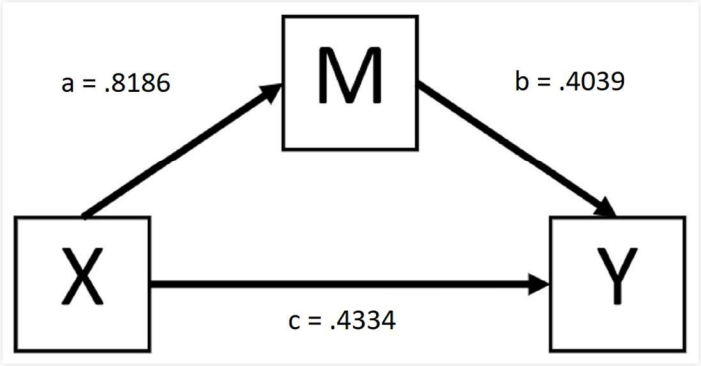
$Y = bM + cX$



Refresher on mediation analysis (2)

In the example below, the indirect effect is \(.8186 * .4039 = .3306 \). This means that, when X increases of 1*SD*, Y increases through M of .3306 *SDs* and it increases of additional .4334 *SDs* independent of M.

Under the assumption that \(a*b \) is normally distributed, the indirect effect can be tested through Sobel's (1982) test. Power for Sobel test can be determined analytically, for example with package **powerMediation** (Qiu, 2017).



Power for mediation analysis - analytic solution (1)

function *powerMediation.Sobel* in package **powerMediation** computes power for a Sobel test. However, the parameters to set are not very intuitive. Input parameters are

- **n**, the sample size
- **theta.1a**, path coefficient a (in our case, .8186)
- **lambda.1a**, path coefficient b (in our case, .4039)
- **sigma.x** and **sigma.y**, the *SDs* of X and M respectively (in our case, both = 1, because we assume standardized variables).
- **sigma.epsilon**, which is the standard deviation (σ_ϵ) of the error term (ϵ) in the multiple regression $Y = b_1 X + b_2 M + \epsilon$. sigma.epsilon can be determined given a, b, and c using the following formula

$$\sigma_\epsilon = \sqrt{1-(b^2+c^2+2abc)}$$

In our case, this amounts to $$\sigma_\epsilon = \sqrt{1-(.4039^2+.4334^2+2*.8186*.4039*.4334)} = .6020$$

Power for mediation analysis - analytic solution (2)

If we plug the arguments in function ssMediation.Sobel, we get a power of .96

```
powerMediation.Sobel(n = 100,
                     theta.1a = .8186,
                     lambda.a = .4039,
                     sigma.x = 1,
                     sigma.m = 1,
                     sigma.epsilon = .6020)
```

```
[1] 0.9607992
```

Power for mediation analysis - analytic solution (3)

Function ssMediation.Sobel allows performing apriori power analysis, by specifying the desired **power** instead of the sample size **n**. The results show that for 80% power, in our case we would need at least 57 participants.

```
ssMediation.Sobel(power = .80,
                  theta.1a = .8186,
                  lambda.a = .4039,
                  sigma.x = 1,
                  sigma.m = 1,
                  sigma.epsilon = .6020)
```

```
[1] 56.71795
```

Power for mediation analysis - simulation in bmem (1)

The assumption that the sampling distribution of the indirect effect \(a*b \) is normal has been shown to be untenable and particularly problematic with small sample sizes. For this reason, the indirect effect is often tested using bootstrap (e.g., Preacher & Hayes, 2004).

If bootstrap is used, analytic formulas for power are not available. As we have learned, in cases like this one, simulation can help. Fortunately, we do not need to set up a simulation from scratch, as R package **bmem** implements what we need.

Steps

1. **define the expected values of the population parameters under H1**
2. generate a sample of size N from the population parameters
3. test the significance of the target effect using the preferred statistical method
4. replicate steps 2 and 3 a large number of times
5. estimate power as the proportion of simulated samples in which H0 is rejected

Power for mediation analysis - simulation in bmem (2)

First, we need to specify the values of population parameters under H1, using a syntax that is taken from the R package *lavaan* (a package for structural equation models; Rosseel, 2012). In short, a model is a text string, each new line specifies either regression ("~") or a variance ("~~"). *start()* is used for specifying population values for each coefficient under H1.

```
model <-'
M ~ a*X + start(.8186)*X
Y ~ b*M + c*X + start(.4039)*M + start(.4334)*X
X ~~ start(1)*X
M ~~ start(1)*M
Y ~~ start(1)*Y'
```

- The first row specifies the regression $M = aX$
- The second row specifies the regression $Y = bM + cX$
- The last three rows only specify that all three variables have unitary variance

Steps

Package **bmem** takes care of implementing steps 2 to 5 for us with little programming.

1. define the expected values of the population parameters under H1
2. **generate a sample of size N from the population parameters**
3. **test the significance of the target effect using the preferred statistical method**
4. **replicate steps 2 and 3 a large number of times**
5. **estimate power as the proportion of simulated samples in which H0 is rejected**

Power for mediation analysis - simulation in bmem (3)

Once a model is specified, function **power.boot()** performs all the bootstrap simulation for us. We need to specify the model previously defined and the number of observations (**nobs**).

Argument *indirect* specifies the indirect effect for which we wish to get power, using the special assignment symbol (":="). In this case, we want to obtain power for the product of coefficients *a* and *b*.

```
power.result <- power.boot(model,
                           indirect = 'ab := a*b',
                           nobs = 100,
                           ncore = 7)
summary(power.result)
```

However, I recommend **NOT** to run the previous code, unless you can wait many hours. The code entails B = 1000 repetitions, each one involving 1000 Monte Carlo samples. I copy-paste the output in the next slide!

Power for mediation analysis - simulation in bmem (4)

bmem Output suggests that we have 96.6% power to detect the indirect effect.

```
Basic information:

  Estimation method                              ML
  Standard error                        standard
  Number of requested bootstrap             1000
  Number of requested replications          1000
  Number of successful replications         1000

                      True  Estimate      MSE      SD     Power Power.se Coverage
Regressions:
  M ~
    X         (a)     0.819    0.826    0.100    0.104    1.000    0.000    0.942
  Y ~
    M         (b)     0.404    0.402    0.101    0.105    0.966    0.006    0.933
    X         (c)     0.433    0.431    0.131    0.129    0.897    0.010    0.946

Variances:
    X                 1.000    0.989    0.136    0.135    1.000    0.000    0.927
    M                 1.000    0.978    0.135    0.135    1.000    0.000    0.921
    Y                 1.000    0.974    0.134    0.140    1.000    0.000    0.894

Indirect/Mediation effects:
    ab                0.331    0.332    0.094    0.095    0.966    0.006    0.928
```

Schoemann's approach (1)

An alternative for reducing computational requirements have been proposed by Schoemann (2017).

- First, instead of bootstrap confidence intervals, they use a faster alternative, the so-called Monte Carlo confidence intervals, which assume that \(a \) and \(b \) are normally distributed, without making assumptions on the distribution of their product (for details, see Preacher & Selig, 2012).
- Second, they use a varying sample size approach: Instead of fixing N for all simulated samples, a value of N is picked at random in the range of specified values. Each simulated sample has thus a different N.
- Third, the fact that a significant (vs. nonsignificant) value is predicted from N using a logistic regression approach. Power (i.e., the probability of obtaining a significant result) can be then predicted from the logistic regression equation given any sample size N within the range of interest.

Schoemann's approach (2)

The analysis is implemented in a Shiny app and requires no programming. The app can be found at this link

https://schoemanna.shinyapps.io/mc_power_med/

Or it can be run in Rstudio on one's computer, using the following code.

```
# runGitHub("mc_power_med", "schoam4")
```

This code assumes that package *shiny* has been previously loaded.

Schoemann's approach (3)

Instead of a, b, and c, one needs to specify correlations between X, Y, and M. These can be easily obtained using the following formulas.

- $r_{mx} = a$ in our case .8186
- $r_{yx} = c + a*b$ in our case .7640
- $r_{ym} = b + a*c$ in our case .7587

If we assume that the variables are all standardized, we can leave the values of standard deviation to the default value of 1.

Schoemann's approach (4)

In the Shiny app, one has to plug in the values computed and then select one of the two approaches

- *Set N, Find power* is equivalent to post-hoc power analysis in Gpower. In this case, a specific sample size is specified.
- *Set power, vary N* is equivalent to a priori power analysis in Gpower. In this case, a range of plausible sample size needs to be specified. More advanced models with two mediators are available in the Shiny app (for details, see Schoemann, 2017).