**Current state of the matter**

This patiently explains why post-hoc power is bad. Interestingly, the manuscript aims to be understood by scientists of all backgrounds, not only statistical. It uses appealing examples and develops them in a didactic, orderly manner, so a clinician (like myself) can grasp those concepts.

*Also, the second paragraph begins using the word “Dismayingly”. How graphic.*

**What can we do about it?**

My two cents to this discussion abut **post-hoc power** come below in the form of a small R tutorial. It shows (yet again), using the rationale in the paper, how it *doesn’t make sense* to get power calculations after hypothesis testing.

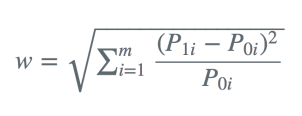
I hope this can help those peeps that need a more ‘*hands-on*’ (rather than purely abstract) approach to learning stats! R serves as a great, flexible tool for this!.

**Observed power is a function of the p-value**

*(One determines the other)*

TL;DR: nonsignificant p values always correspond to low observed powers.

This piece of code is inspired by the one written by Lakens, only that it is done with p-values from a Chi-Square test, making use of the pwr::pwr.chisq.test() function to get the power of the study. The only problem is that the effect size in this function is given by w, but fortunately *w* can be estimated from our contingency table using this formula:



* P0i= the proportion in cell i posited by the null hypothesis,
* P1i= the proportion in cell i posited by the alternate hypothesis and reflects the effect for that cell, and
* m= the number of cells.

The reason I say “estimate” instead of to “calculate” the range of *w* values, is that we will use the sample proportions (observed, expected) instead of the population values (which we don’t know). Regardless effect size estimation in this case the results would be the same, so for didactic purposes this should be fine.

**Defining the problem: genomic data**

Let’s start with a simplified case for the simplest possible contingency table: a 2×2 table featuring an exposure and the affected status in a **case-control study**:

* The Antithrombin-III-Hamilton disease is characterized by recurrent thromboembolic events
* Caused by a DNA mutation exchanging Guanine (G) for Adenine (A) in the first base of codon 382 from the AT-III gene. This impairs its serine protease reactivity, thus being less effective blocking thrombus formation.
* Thus, we can consider the “A” allele as the exposure, “G” as the absence of exposure, and the “cases” would be those with recurrent thromboembolic events.

We cannot simulate from a rnorm distribution like in t-tests. To simulate datasets with different table proportions we will set a range of different frequency combinations for two of the cells:

library(tidyverse)

# Set basic parameters

lower\_prob\_range=0.3

upper\_prob\_range=0.7

prob\_step=0.01

probabilities <- seq(from = lower\_prob\_range, to = upper\_prob\_range, by = prob\_step)

# Create dataframe with all the possible effect sizes

mydata0 <- expand.grid(probabilities,probabilities) %>%

tibble::as\_tibble() %>%

rename(

`prob of exposure in cases` = Var1,

`prob of exposure in controls` = Var2

)

Next, we define a function that does all the calculations and stores the results in a list (the code is as inelegant and explicit as possible):

power\_against\_pvalue <- function(probabilities\_expo\_case=0.5, probabilities\_expo\_control=0.5, N=1000, exposure="exposed", unexposed="unexposed"){

require(pwr)

require(magrittr)

table\_exposure <- c(exposure, unexposed)

# Make a data frame

cases <- tibble(

status = "case",

allele = sample( table\_exposure, N, replace=TRUE, prob=c(probabilities\_expo\_case, 1-probabilities\_expo\_case) )

)

controls <- tibble(

status = "control",

allele = sample( table\_exposure, N, replace=TRUE, prob=c(probabilities\_expo\_control, 1-probabilities\_expo\_control) )

)

study <- cases %>%

bind\_rows(controls)

# Observed counts

.TableOC <- table(study$status, study$allele)

# Observed proportions (row)

.TableOPr <- prop.table(.TableOC, 1) # proportions by row

# Observed proportions (table)

.TableOP <- prop.table(.TableOC) # proportions by table

# Chi-squared test

xsq\_text <- .TableOC %>% chisq.test()

# Expected table counts:

.TableEC <- xsq\_text$expected

# Expected table proportions:

.TableEP <- prop.table(.TableEC) # proportions by table

# get p-value

xsq\_pvalue <- xsq\_text$p.value

# Observed counts:

H\_n <- count\_observed\_ctrls\_unexpo <- .TableOC[2,2]

D\_n <- count\_observed\_cases\_unexpo <- .TableOC[1,2]

H\_e <- count\_observed\_ctrls\_expo <- .TableOC[2,1] # A is the exposure

D\_e <- count\_observed\_cases\_expo <- .TableOC[1,1] # A is the exposure

# Get the OR

OR <- ( D\_e / H\_e ) / ( D\_n / H\_n )

# Observed table proportions

pr\_observed\_ctrls\_unexpo <- .TableOP[2,2]

pr\_observed\_case\_unexpo <- .TableOP[1,2]

pr\_observed\_ctrls\_expo <- .TableOP[2,1]

pr\_observed\_case\_expo <- .TableOP[1,1]

# Expected table proportions:

pr\_expected\_ctrls\_unexpo <- .TableEP[2,2]

pr\_expected\_case\_unexpo <- .TableEP[1,2]

pr\_expected\_ctrls\_expo <- .TableEP[2,1]

pr\_expected\_case\_expo <- .TableEP[1,1]

# Cohen's w

w <- (

(

(pr\_expected\_ctrls\_unexpo - pr\_observed\_ctrls\_unexpo)^2 / pr\_expected\_ctrls\_unexpo

) + (

(pr\_expected\_ctrls\_expo - pr\_observed\_ctrls\_expo)^2 / pr\_expected\_ctrls\_expo

) + (

(pr\_expected\_case\_unexpo - pr\_observed\_case\_unexpo)^2 / pr\_expected\_case\_unexpo

) + (

(pr\_expected\_case\_expo - pr\_observed\_case\_expo)^2 / pr\_expected\_case\_expo

)

)^(1/2)

# Power

pw <- pwr.chisq.test(

w = w,

N = N,

df = 1,

sig.level = 0.05,

power = NULL

)

# Final result

result0 <- list(

prob\_case\_in\_exposed = probabilities\_expo\_case,

prob\_control\_in\_exposed = probabilities\_expo\_control,

N = N,

w = w,

power = pw$power,

p\_value = xsq\_pvalue,

OR = OR,

observed\_counts = .TableOC,

expected\_counts = .TableEC,

observed\_proportions = .TableOP,

expected\_proportions = .TableEP

)

result0

}

# A helper function to extract the results into new columns

power\_against\_pvalue\_extractor <- function(power\_versus\_pvalue, extracted=NULL){

if(is.null(extracted)){

result <- power\_versus\_pvalue

}else{

result <- power\_versus\_pvalue[[extracted]]

}

result

}

We can execute this function and see the output:

power\_against\_pvalue(

probabilities\_expo\_case=0.5,

probabilities\_expo\_control=0.5,

N=1000,

exposure="A=exposed",

unexposed="G=unexposed"

)

$prob\_case\_in\_exposed

[1] 0.5

$prob\_control\_in\_exposed

[1] 0.5

$N

[1] 1000

$w

[1] 0.02300609

$power

[1] 0.1124906

$p\_value

[1] 0.3250515

$OR

[1] 1.096436

$observed\_counts

A=exposed G=unexposed

case 523 477

control 500 500

$expected\_counts

A=exposed G=unexposed

case 511.5 488.5

control 511.5 488.5

$observed\_proportions

A=exposed G=unexposed

case 0.2615 0.2385

control 0.2500 0.2500

$expected\_proportions

A=exposed G=unexposed

case 0.25575 0.24425

control 0.25575 0.24425

And now we can map this function to all the combinations of proportions we had stored in our data frame (using the wonderful purrr package):

mydata <- mydata0 %>%

# calculate

mutate(

power\_versus\_pvalue = purrr::map2(

`prob of exposure in cases`, `prob of exposure in controls`,

~power\_against\_pvalue(.x, .y, N=1000, exposure="A=exposed", unexposed="G=unexposed")

)

) %>%

# extract elements

mutate(

power = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="power"),

pvalue = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="p\_value"),

OR = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="OR"),

cohen\_w = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="w")

) %>%

unnest(cols = c(power, pvalue, OR, cohen\_w))

mydata %>% select(-power\_versus\_pvalue, -cohen\_w) %>% head(3) %>% kable()

| **prob of exposure in cases** | **prob of exposure in controls** | **power** | **pvalue** | **OR** |
| --- | --- | --- | --- | --- |
| 0.30 | 0.3 | 0.0500000 | 1.0000000 | 1.000000 |
| 0.31 | 0.3 | 0.1439164 | 0.2275967 | 1.128824 |
| 0.32 | 0.3 | 0.1547361 | 0.2025910 | 1.138560 |

We can see the power that corresponds to a p-value of 0.05:

mydata %>%

filter(

pvalue > 0.049 & pvalue < 0.051

) %>%

select(-power\_versus\_pvalue, -cohen\_w) %>% kable()

| **prob of exposure in cases** | **prob of exposure in controls** | **power** | **pvalue** | **OR** |
| --- | --- | --- | --- | --- |
| 0.40 | 0.44 | 0.2925585 | 0.0508424 | 0.8339090 |
| 0.46 | 0.50 | 0.2962194 | 0.0490027 | 0.8350416 |
| 0.48 | 0.50 | 0.2960685 | 0.0490725 | 0.8351346 |
| 0.65 | 0.69 | 0.2930812 | 0.0508489 | 0.8263437 |

The power corresponding to a p-value of 0.05 is between 0.29 and 0.30 (exact results will vary each time code runs; for didactic purposes I won’t set.seed here). In Hoenig and Lakens examples, -using different tests- the correspondence was between p=0.05 and a power of 0.5.

We can see how plotting power against p-value (regardless of sample size, you can try with a different one) always yield the same relationship:

power\_vs\_pvalue <- mydata %>%

ggplot(

aes(

x=power,

y=pvalue,

colour=pvalue

)

) +

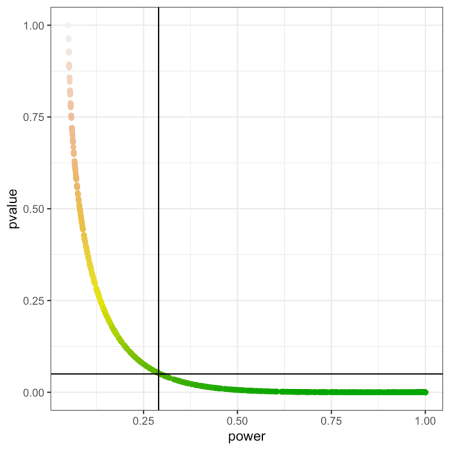
geom\_point() +

geom\_hline(yintercept=0.05) + geom\_vline(xintercept=0.29) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

power\_vs\_pvalue



**Conclusion**

Once you set your data and statistical test to compute a p-value, your power is already fixed. Hence, power doesn’t add information and cannot be further interpreted, as higher p-values will ***always*** correspond to low powers.

**Bonus: Plotting power & p-pvalue against OR and Cohen’s w**

Here are 4 graphs corresponding to 1000 patients. Unlike the ‘power versus p-value’ representation, those are influenced by sample size. With **bigger sample sizes**:

* As effect size increases, power increases are more steep and p-value decreases more quickly,
* OR values that correspond either to *p-values under 0.05* or *power values over 0.8* are closer to 1.

or\_vs\_pvalue <- mydata %>%

ggplot(

aes(x=OR, y=pvalue, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.05) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

w\_vs\_pvalue <- mydata %>%

ggplot(

aes(x=cohen\_w, y=pvalue, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.05) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

or\_vs\_power <- mydata %>%

ggplot(

aes(x=OR, y=power, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.8) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

w\_vs\_power <- mydata %>%

ggplot(

aes(x=cohen\_w, y=power, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.8) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

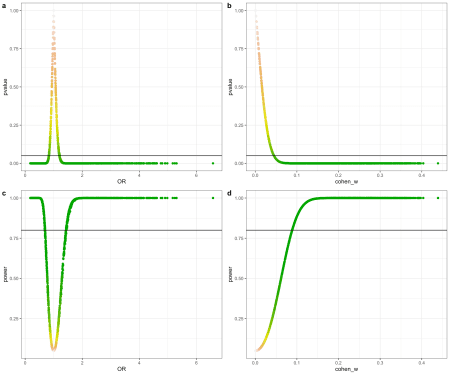
cowplots <- list(or\_vs\_pvalue, w\_vs\_pvalue, or\_vs\_power, w\_vs\_power)

mylabels <- letters[1:length(cowplots)]

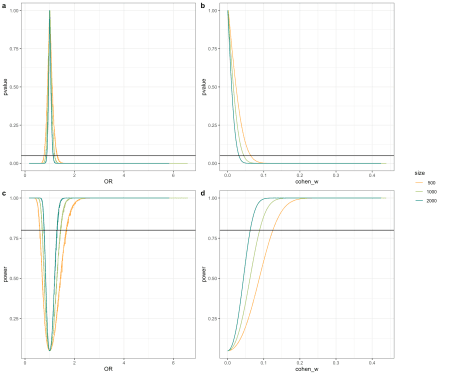
composed\_figure <- cowplot::plot\_grid(plotlist = cowplots,

labels = mylabels, nrow = 2, align = "h")

composed\_figure



**Different sample sizes**



**This .Rmd source code**

title: 'Playing with post-hoc power with R - why we shouldn’t do it'

author: 'A. Baluja'

date: '2020-01-04'

slug: playing-with-post-hoc-power

categories:

- R

tags:

- '#rstats'

- Statistical

- research

- '#tutorial'

- medicine

- power

twitterImg: ./images/posthoc.png

keywords:

- tech

thumbnailImagePosition: left

thumbnailImage: ./images/posthoc.png

metaAlignment: center

disable\_comments: false

output:

blogdown::html\_page:

toc: false

fig\_width: 8

css: "/css/my-style.css"

editor\_options:

chunk\_output\_type: console

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(collapse = TRUE, echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE, comment=" ")

# https:\/\/aurora-mareviv.github.io\/talesofr

```

```{r results='hide', echo=FALSE}

# Installs missing libraries on render!

list.of.packages <- c("rmarkdown", "Rcpp", "knitr", "tidyverse", "pwr", "kableExtra", "cowplot", "gridGraphics")

new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]

if(length(new.packages)) install.packages(new.packages, repos='https://cran.rstudio.com/')

```

```{r echo=FALSE}

wdir <- getwd()

# la carpeta donde guardo los datos

datadir <- paste(wdir, "/data", sep="")

# y la carpeta inmediatamente por encima

wdirRoot <- gsub("\\content\\/post", "", wdir)

```

```{r fun\_kabler, echo=FALSE}

# Kable Extra formatter

require(magrittr)

kable <- function(x){

y <- x %>%

knitr::kable("html", escape = F, align = "c") %>%

kableExtra::kable\_styling("striped", full\_width = F)

y

}

```

<!-- \*(A half-baked post that I had some time ago pending finishing)\* -->

## Current state of the matter

The reason for bringing this here is that I witnessed an interesting exchange some time ago, regarding one article and their use of post-hoc power, pinpointed by @[ADAlthousePhD](https://twitter.com/ADAlthousePhD){target="\_blank"}:

<center>

`r htmltools::HTML("{{< tweet 1123205494968074240 >}}")`

`r htmltools::HTML("{{< tweet 1123206831885692928 >}}")`

`r htmltools::HTML("{{< tweet 1123575393481629697 >}}")`

</center>

The tweets refer also to this great post: \*\*[Observed power, and what to do if your editor asks for post-hoc power analyses](http://daniellakens.blogspot.com/2014/12/observed-power-and-what-to-do-if-your.html){target="\_blank"}\*\*, written by [Dani&euml;l Lakens](https://twitter.com/lakens){target="\_blank"} in which this issue is discussed.

At first, I wasn't too interested in this topic (to be honest)<!-- from an statistical point of view-->; but then I read [the above mentioned study](https://www.sciencedirect.com/science/article/pii/S0022480419301854?via%3Dihub){target="\_blank"}, showcasing post-hoc calculations, and a few others that were \*\*spreading \*and\* being cited (even worse)\*\* in a field very, very close to mine: \*\*surgery\*\*. That's when it got personal.

Like almost a year ago, I came across this beautiful paper: \*\*[The Abuse of Power: The Pervasive Fallacy of Power Calculations for Data Analysis](https://www.vims.edu/people/hoenig\_jm/pubs/hoenig2.pdf){target="\_blank"}\*\* by [John Hoenig et al.](https://www.researchgate.net/profile/John\_Hoenig){target="\_blank"}

This paper patiently explains why post-hoc power is bad. Interestingly, the manuscript aims to be understood by scientists of all backgrounds, not only statistical. It uses appealing examples and develops them in a didactic, orderly manner, so a clinician (like myself) can grasp those concepts.

\*Also, the second paragraph begins using the word "Dismayingly". How graphic.\*

## What can we do about it?

My two cents to this discussion abut \*\*post-hoc power\*\* come below in the form of a small R tutorial. It shows (yet again), using the rationale in the paper, how it \*doesn't make sense\* to get power calculations after hypothesis testing.

I hope this can help those peeps that need a more '\*hands-on\*' (rather than purely abstract) approach to learning stats! R serves as a great, flexible tool for this!.

### Observed power is a function of the p-value

\*(One determines the other)\* <!--\*(they're the same thing)\*-->

<div class="well alert alert-info text-center">

TL;DR: nonsignificant p values always correspond to low observed powers.

</div>

This piece of code is inspired by the one written by Lakens, only that it is done with p-values from a Chi-Square test, making use of the `pwr::pwr.chisq.test()` function to get the power of the study. The only problem is that the effect size in this function is given by `w`, but fortunately \*w\* can be estimated from our contingency table using this formula (from Cohen's [book](https://www.amazon.com/Statistical-Power-Analysis-Behavioral-Sciences/dp/0805802835){target="blank"}, p.216):

$w = \sqrt{ \sum\_{i=1}^{m} \dfrac{ (P\_{1i} - P\_{0i})^2 } {P\_{0i}} }$

- $P\_{0i}$ = the proportion in cell i posited by the null hypothesis,

- $P\_{1i}$ = the proportion in cell i posited by the alternate hypothesis and reflects the effect for that cell, and

- $m$ = the number of cells.

The reason I say "estimate" instead of to "calculate" the range of \*w\* values, is that we will use the sample proportions (observed, expected) instead of the population values (which we don't know). Regardless effect size estimation in this case the results would be the same, so for didactic purposes this should be fine.

#### Defining the problem: genomic data

Let's start with a simplified case for the simplest possible contingency table: a 2x2 table featuring an exposure and the affected status in a \*\*case-control study\*\*:

- The Antithrombin-III-Hamilton disease is characterized by recurrent thromboembolic events (PMID: [3179438](https://www.ncbi.nlm.nih.gov/pubmed/3179438){target="blank"}).

- Caused by a DNA mutation exchanging Guanine (G) for Adenine (A) in the first base of codon 382 from the AT-III gene. This impairs its serine protease reactivity, thus being less effective blocking thrombus formation.

- Thus, we can consider the "A" allele as the exposure, "G" as the absence of exposure, and the "cases" would be those with recurrent thromboembolic events.

We cannot simulate from a `rnorm` distribution like in t-tests. To simulate datasets with different table proportions we will set a range of different frequency combinations for two of the cells:

```{r power\_parameters\_data}

library(tidyverse)

# Set basic parameters

lower\_prob\_range=0.3

upper\_prob\_range=0.7

prob\_step=0.01

probabilities <- seq(from = lower\_prob\_range, to = upper\_prob\_range, by = prob\_step)

# Create dataframe with all the possible effect sizes

mydata0 <- expand.grid(probabilities,probabilities) %>%

tibble::as\_tibble() %>%

rename(

`prob of exposure in cases` = Var1,

`prob of exposure in controls` = Var2

)

```

Next, we define a function that does all the calculations and stores the results in a list (the code is as inelegant and explicit as possible):

```{r fun\_power\_against\_pvalue}

power\_against\_pvalue <- function(probabilities\_expo\_case=0.5, probabilities\_expo\_control=0.5, N=1000, exposure="exposed", unexposed="unexposed"){

require(pwr)

require(magrittr)

table\_exposure <- c(exposure, unexposed)

# Make a data frame

cases <- tibble(

status = "case",

allele = sample( table\_exposure, N, replace=TRUE, prob=c(probabilities\_expo\_case, 1-probabilities\_expo\_case) )

)

controls <- tibble(

status = "control",

allele = sample( table\_exposure, N, replace=TRUE, prob=c(probabilities\_expo\_control, 1-probabilities\_expo\_control) )

)

study <- cases %>%

bind\_rows(controls)

# Observed counts

.TableOC <- table(study$status, study$allele)

# Observed proportions (row)

.TableOPr <- prop.table(.TableOC, 1) # proportions by row

# Observed proportions (table)

.TableOP <- prop.table(.TableOC) # proportions by table

# Chi-squared test

xsq\_text <- .TableOC %>% chisq.test()

# Expected table counts:

.TableEC <- xsq\_text$expected

# Expected table proportions:

.TableEP <- prop.table(.TableEC) # proportions by table

# get p-value

xsq\_pvalue <- xsq\_text$p.value

# Observed counts:

H\_n <- count\_observed\_ctrls\_unexpo <- .TableOC[2,2]

D\_n <- count\_observed\_cases\_unexpo <- .TableOC[1,2]

H\_e <- count\_observed\_ctrls\_expo <- .TableOC[2,1] # A is the exposure

D\_e <- count\_observed\_cases\_expo <- .TableOC[1,1] # A is the exposure

# Get the OR

OR <- ( D\_e / H\_e ) / ( D\_n / H\_n )

# Observed table proportions

pr\_observed\_ctrls\_unexpo <- .TableOP[2,2]

pr\_observed\_case\_unexpo <- .TableOP[1,2]

pr\_observed\_ctrls\_expo <- .TableOP[2,1]

pr\_observed\_case\_expo <- .TableOP[1,1]

# Expected table proportions:

pr\_expected\_ctrls\_unexpo <- .TableEP[2,2]

pr\_expected\_case\_unexpo <- .TableEP[1,2]

pr\_expected\_ctrls\_expo <- .TableEP[2,1]

pr\_expected\_case\_expo <- .TableEP[1,1]

# Cohen's w

w <- (

(

(pr\_expected\_ctrls\_unexpo - pr\_observed\_ctrls\_unexpo)^2 / pr\_expected\_ctrls\_unexpo

) + (

(pr\_expected\_ctrls\_expo - pr\_observed\_ctrls\_expo)^2 / pr\_expected\_ctrls\_expo

) + (

(pr\_expected\_case\_unexpo - pr\_observed\_case\_unexpo)^2 / pr\_expected\_case\_unexpo

) + (

(pr\_expected\_case\_expo - pr\_observed\_case\_expo)^2 / pr\_expected\_case\_expo

)

)^(1/2)

# Power

pw <- pwr.chisq.test(

w = w,

N = N,

df = 1,

sig.level = 0.05,

power = NULL

)

# Final result

result0 <- list(

prob\_case\_in\_exposed = probabilities\_expo\_case,

prob\_control\_in\_exposed = probabilities\_expo\_control,

N = N,

w = w,

power = pw$power,

p\_value = xsq\_pvalue,

OR = OR,

observed\_counts = .TableOC,

expected\_counts = .TableEC,

observed\_proportions = .TableOP,

expected\_proportions = .TableEP

)

result0

}

# A helper function to extract the results into new columns

power\_against\_pvalue\_extractor <- function(power\_versus\_pvalue, extracted=NULL){

if(is.null(extracted)){

result <- power\_versus\_pvalue

}else{

result <- power\_versus\_pvalue[[extracted]]

}

result

}

```

We can execute this function and see the output:

```{r basic\_output}

power\_against\_pvalue(

probabilities\_expo\_case=0.5,

probabilities\_expo\_control=0.5,

N=1000,

exposure="A=exposed",

unexposed="G=unexposed"

)

```

And now we can map this function to all the combinations of proportions we had stored in our data frame (using the wonderful `purrr` package):

```{r mapped\_output}

mydata <- mydata0 %>%

# calculate

mutate(

power\_versus\_pvalue = purrr::map2(

`prob of exposure in cases`, `prob of exposure in controls`,

~power\_against\_pvalue(.x, .y, N=1000, exposure="A=exposed", unexposed="G=unexposed")

)

) %>%

# extract elements

mutate(

power = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="power"),

pvalue = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="p\_value"),

OR = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="OR"),

cohen\_w = purrr::map(power\_versus\_pvalue, power\_against\_pvalue\_extractor, extracted="w")

) %>%

unnest(cols = c(power, pvalue, OR, cohen\_w))

```

```{r print\_output}

mydata %>% select(-power\_versus\_pvalue, -cohen\_w) %>% head(3) %>% kable()

```

&nbsp;

We can see the power that corresponds to a p-value of 0.05:

```{r select\_output}

mydata %>%

filter(

pvalue > 0.049 & pvalue < 0.051

) %>%

select(-power\_versus\_pvalue, -cohen\_w) %>% kable()

```

&nbsp;

The power corresponding to a p-value of 0.05 is between 0.29 and 0.30 (exact results will vary each time code runs; for didactic purposes I won't `set.seed` here). In Hoenig and Lakens examples, -using different tests- the correspondence was between p=0.05 and a power of 0.5.

We can see how plotting power against p-value (regardless of sample size, you can try with a different one) always yield the same relationship:

```{r plot\_output, fig.height=5, fig.width=5}

power\_vs\_pvalue <- mydata %>%

ggplot(

aes(

x=power,

y=pvalue,

colour=pvalue

)

) +

geom\_point() +

geom\_hline(yintercept=0.05) + geom\_vline(xintercept=0.29) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

power\_vs\_pvalue

```

&nbsp;

## Conclusion

Once you set your data and statistical test to compute a p-value, your power is already fixed. Hence, power doesn't add information and cannot be further interpreted, as higher p-values will \*\*\*always\*\*\* correspond to low powers.

[Hoenig's paper](https://www.vims.edu/people/hoenig\_jm/pubs/hoenig2.pdf){target="\_blank"} elaborates on more reasons why calculated (post-hoc) power cannot be interpreted (hence used). The Discussion section definitely worths reading.

## Bonus

#### Plotting power & p-pvalue against OR and Cohen's \*w\*

Here are 4 graphs corresponding to 1000 patients. Unlike the 'power versus p-value' representation, those are influenced by sample size. With \*\*bigger sample sizes\*\*:

- As effect size increases, power increases are more steep and p-value decreases more quickly,

- OR values that correspond either to \*p-values under 0.05\* or \*power values over 0.8\* are closer to 1.

```{r plot\_output2, fig.height=10, fig.width=12}

or\_vs\_pvalue <- mydata %>%

ggplot(

aes(x=OR, y=pvalue, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.05) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

w\_vs\_pvalue <- mydata %>%

ggplot(

aes(x=cohen\_w, y=pvalue, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.05) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

or\_vs\_power <- mydata %>%

ggplot(

aes(x=OR, y=power, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.8) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

w\_vs\_power <- mydata %>%

ggplot(

aes(x=cohen\_w, y=power, colour=pvalue)

) +

geom\_point() +

geom\_hline(yintercept=0.8) +

scale\_colour\_gradientn(colours = terrain.colors(5)) +

theme\_bw() + theme(legend.position = "none")

cowplots <- list(or\_vs\_pvalue, w\_vs\_pvalue, or\_vs\_power, w\_vs\_power)

mylabels <- letters[1:length(cowplots)]

composed\_figure <- cowplot::plot\_grid(plotlist = cowplots,

labels = mylabels, nrow = 2, align = "h")

composed\_figure

```

&nbsp;

##### Different sample sizes

```{r additional\_data\_samsizes, echo=FALSE}

mydata <- mydata %>%

# calculate 500

mutate(

power\_versus\_pvalue\_500 = purrr::map2(

`prob of exposure in cases`, `prob of exposure in controls`,

~power\_against\_pvalue(.x, .y, N=500, exposure="A=exposed", unexposed="G=unexposed")

)

) %>%

# extract elements

mutate(

power\_500 = purrr::map(power\_versus\_pvalue\_500, power\_against\_pvalue\_extractor, extracted="power"),

pvalue\_500 = purrr::map(power\_versus\_pvalue\_500, power\_against\_pvalue\_extractor, extracted="p\_value"),

OR\_500 = purrr::map(power\_versus\_pvalue\_500, power\_against\_pvalue\_extractor, extracted="OR"),

cohen\_w\_500 = purrr::map(power\_versus\_pvalue\_500, power\_against\_pvalue\_extractor, extracted="w")

) %>%

unnest(cols = c(power\_500, pvalue\_500, OR\_500, cohen\_w\_500)) %>%

# calculate 2000

mutate(

power\_versus\_pvalue\_2000 = purrr::map2(

`prob of exposure in cases`, `prob of exposure in controls`,

~power\_against\_pvalue(.x, .y, N=2000, exposure="A=exposed", unexposed="G=unexposed")

)

) %>%

# extract elements

mutate(

power\_2000 = purrr::map(power\_versus\_pvalue\_2000, power\_against\_pvalue\_extractor, extracted="power"),

pvalue\_2000 = purrr::map(power\_versus\_pvalue\_2000, power\_against\_pvalue\_extractor, extracted="p\_value"),

OR\_2000 = purrr::map(power\_versus\_pvalue\_2000, power\_against\_pvalue\_extractor, extracted="OR"),

cohen\_w\_2000 = purrr::map(power\_versus\_pvalue\_2000, power\_against\_pvalue\_extractor, extracted="w")

) %>%

unnest(cols = c(power\_2000, pvalue\_2000, OR\_2000, cohen\_w\_2000))

deselected <- c(

"prob of exposure in cases", "prob of exposure in controls",

"power\_versus\_pvalue", "power\_versus\_pvalue\_500", "power\_versus\_pvalue\_2000"

)

mydata\_sizes0 <- mydata %>%

select(

-deselected

)

mydata\_sizes\_power <- mydata\_sizes0 %>%

select(starts\_with("power")) %>%

tidyr::gather(

power0, value\_power

)

mydata\_sizes\_pvalue <- mydata\_sizes0 %>%

select(starts\_with("pvalue")) %>%

tidyr::gather(

pvalue0, value\_pvalue

)

mydata\_sizes\_OR <- mydata\_sizes0 %>%

select(starts\_with("OR")) %>%

tidyr::gather(

OR0, value\_OR

)

mydata\_sizes\_cohen\_w <- mydata\_sizes0 %>%

select(starts\_with("cohen\_w")) %>%

tidyr::gather(

cohen\_w0, value\_cohen\_w

)

mydata\_sizes <- mydata\_sizes\_power %>%

bind\_cols(mydata\_sizes\_pvalue, mydata\_sizes\_OR, mydata\_sizes\_cohen\_w) %>%

mutate(

size = case\_when(

power0 == "power\_500" ~ " 500",

power0 == "power" ~ "1000",

power0 == "power\_2000" ~ "2000",

TRUE ~ NA\_character\_

)

) %>%

select(-ends\_with("0")) %>%

rename\_all(

list(

~stringr::str\_replace\_all(., 'value\_', '')

)

)

```

```{r plot\_output3, fig.height=10, fig.width=12, echo=FALSE}

discrete\_pal <- c(

"#ffa83d", #yellow

"#a4c166",

"#00857c"

)

cex <- 0.5

or\_vs\_pvalue0 <- mydata\_sizes %>%

ggplot(

aes(x=OR, y=pvalue, colour=size)

) +

# geom\_point(size = cex) +

geom\_line() +

geom\_hline(yintercept=0.05) +

scale\_colour\_manual(values=discrete\_pal) +

theme\_bw()

or\_vs\_pvalue <- or\_vs\_pvalue0+ theme(legend.position = "none")

w\_vs\_pvalue0 <- mydata\_sizes %>%

ggplot(

aes(x=cohen\_w, y=pvalue, colour=size)

) +

# geom\_point(size = cex) +

geom\_line() +

geom\_hline(yintercept=0.05) +

scale\_colour\_manual(values=discrete\_pal) +

theme\_bw()

w\_vs\_pvalue <- w\_vs\_pvalue0 + theme(legend.position = "none")

or\_vs\_power0 <- mydata\_sizes %>%

ggplot(

aes(x=OR, y=power, colour=size)

) +

# geom\_point(size = cex) +

geom\_line() +

geom\_hline(yintercept=0.8) +

scale\_colour\_manual(values=discrete\_pal) +

theme\_bw()

or\_vs\_power <- or\_vs\_power0 + theme(legend.position = "none")

w\_vs\_power0 <- mydata\_sizes %>%

ggplot(

aes(x=cohen\_w, y=power, colour=size)

) +

# geom\_point(size = cex) +

geom\_line() +

geom\_hline(yintercept=0.8) +

scale\_colour\_manual(values=discrete\_pal) +

theme\_bw()

w\_vs\_power <- w\_vs\_power0 + theme(legend.position = "none")

legend <- cowplot::get\_legend(

or\_vs\_pvalue0 + theme(legend.box.margin = margin(0, 0, 0, 9))

)

cowplots <- list(or\_vs\_pvalue, w\_vs\_pvalue, or\_vs\_power, w\_vs\_power)

mylabels <- letters[1:length(cowplots)]

composed\_figure\_sizes <- cowplot::plot\_grid(plotlist = cowplots,

labels = mylabels, nrow = 2, align = "h")

# composed\_figure\_sizes

library(gridGraphics)

cowplot::plot\_grid(composed\_figure\_sizes, legend, rel\_widths = c(3, .4))

```