Power and Simulation

```
library(ggplot2)
library(dplyr)
library(pwr)
library(paramtest)
```

Overview

The **statistical power** of a hypothesis test is the probability that the data will lead to the rejection of the null hypothesis, if a particular version of the alternate hypothesis is in fact true. Power calculations are an important part of study design - being able to compute power means that one can answer the questions, "How much data do I need to gather in order to reliably notice the relationship that I think might exist?" or, "With the data available, what effect size will I reliably be able to notice?" But power calculations based on all but relatively simple statistical models can be horribly complicated or even impossible. Simulation allows one to avoid these complicated or impossible calculations with a bit of programming.

Power calculations that can be done analytically

Some power calculations can be solved with calculus. For these, there are off the shelf functions for computing power. Without loading any packages, you can use `power.t.test`, `power.prop.test` and `power.anova.test`. The `pwr` package gives a few more tools, such as `pwr.f2.test` for linear models.

For a first example, we'll look at a t-test for the difference in means between two groups. Consider the following scenario:

- y is a numeric variable.
- x is a binary categorical variable. The two levels of x are a and b.
- Write μ_a and σ_a for the mean and standard deviation of y when $x = a$, and similarly μ_b and σ_b for when $x = b$.
- The pooled standard deviation σ_{pooled} is the standard deviation of y, ignoring x.
- We want to know if y depends on x, so we will perform a t-test: `t.test(y~x)`.
- We will collect a sample: n_a observations where $x = a$ and n_b observations where $x = b$. How large do n_a and n_b have to be so that if $\mu_a \neq \mu_b$ our t-test will detect it?

An key step in setting up a power calculation is determining a particular version of the alternative hypothesis that we are interested in detecting, in other words, a suitable value for the effect size. (The effect size is our way of quantifying the difference between the null hypothesis and the particular version of the alternative hypothesis to be considered. Power calculations can also be used to determine effect size if all other parameters are known, but this is less frequently done.) The effect size for a t-test used by `pwr.t.test` is the difference in means divided by the pooled standard deviation, known as Cohen's d, after Cohen (1988).

$$d = \frac{\mu_a - \mu_b}{\sigma_{pooled}}$$

We want to choose an effect size that is large enough to be interesting, and to be detectable at reasonable power with an achievable sample size, but not so large that it exceeds values that you would find interesting or useful, or is unlikely to really exist.

To use any of the standard functions for computing power, specify the parameters you wish to fix, and the function will compute the one you leave unspecified. (Default significance level is 0.05, this does not have to be specified. If you want to compute the significance level where a certain power is achieved with fixed sample and effect sizes, you have to explicitly specify `sig.level=NULL`.)

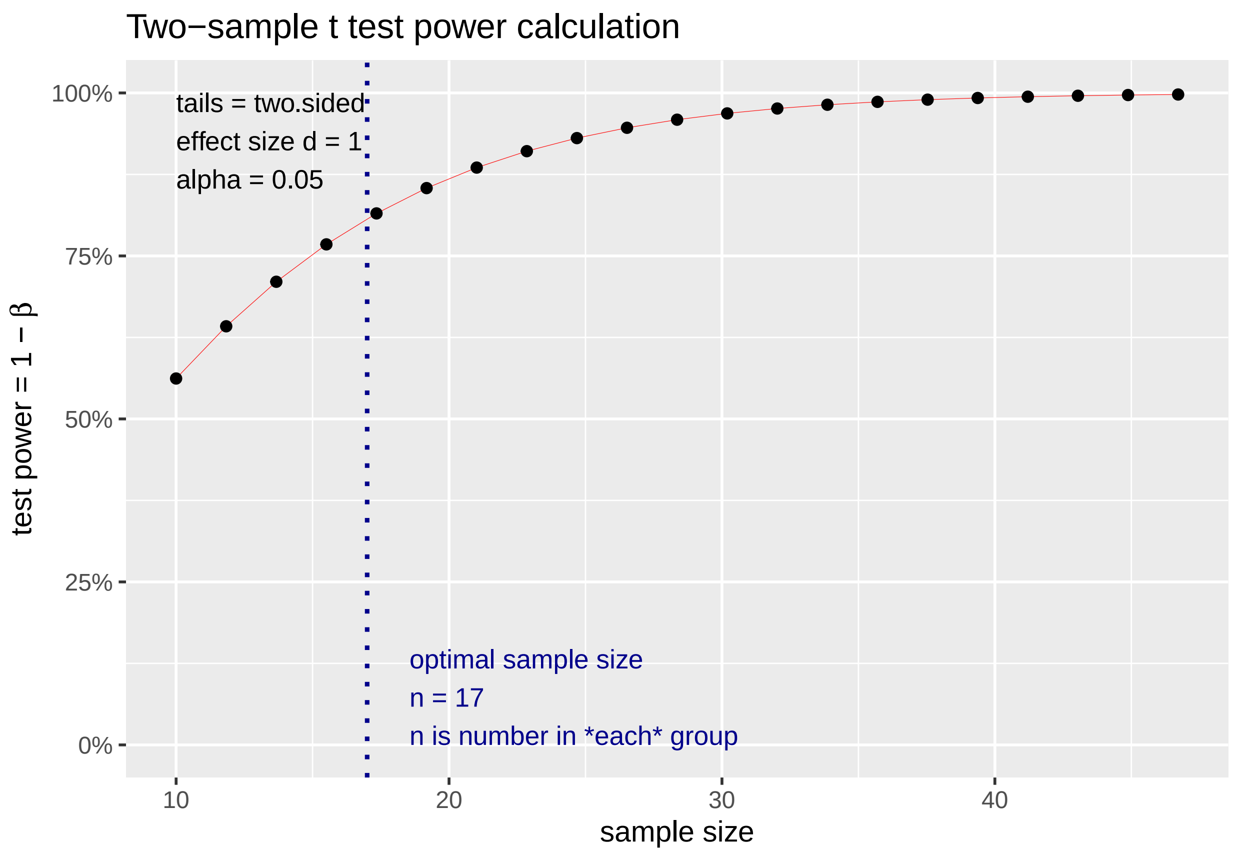
Example: Compute the required sample size for a t-test with an effect size of 1 and a desired power of 80%.

```
tpow<-pwr.t.test(d=1, power = 0.8)
tpow
```

```
##
##      Two-sample t test power calculation
##
##               n = 16.71472
##               d = 1
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

We can also plot the result of this test, giving a bit more information about how our parameter of interest (sample size) affects the power.

```
plot(tpow)
```



(1) Suppose that y is normally distributed with a mean of 50 and a standard deviation of 6. Let $x = a$ be the default, and let $x = b$ correspond to a sub-population. Our research question is if this sub-population has, on average, a different value for y. If μ_b is 53 or more, we would like to collect enough data so that

we have at least a 90% chance of getting a statistically significant result from a *t*-test. How much data do we need to collect?

Simulating a *t*-test

To see how to use simulation to compute power, we can simulate to confirm the result from the example above.

```r
d <- 1 #effect size
N <- 17 #number per group
num_sim <- 5000 #number of t tests to simulate
sig <- logical(num_sim) #initialize
for(i in 1:num_sim){
  sim <- data.frame(group = c(rep("a",N), rep("b",N)), #simulate data
                    value = c(rnorm(N,0,1), rnorm(N,d,1)))
  ttest <- t.test(value~group, data=sim) #test for difference
  p <- ttest$p.value #extract p value
  sig[i] <- (p < 0.05) #record significance
}
cat("power =",mean(sig)) #compute and report power
```

```
## power = 0.7984
```

This is quite close to the claimed power of 80%. If we want to find the sample size $N = 17$ that gives this power, as we did using the analytic test above, we can run a series of simulations with increasing sample size until a power of 80% is reached. Be advised, this can take a while (about a minute on my machine for the following code). If you have a slower computer, you might try decreasing the number of simulations of the target power before you run the following code.

```r
d <- 1 #effect size
num_sim <- 5000 #number of t tests to simulate for each N
pow <- 0 #initialize power
N <- 1 #initialize sample size
while(pow<0.8){
  N <- N+1 #increment sample size
  sig <- logical(num_sim) #initialize significance
  for(i in 1:num_sim){
    sim <- data.frame(group = c(rep("a",N), rep("b",N)), #simulate data
                      value = c(rnorm(N,0,1), rnorm(N,d,1)))
    ttest <- t.test(value~group, data=sim) #test for difference
    p <- ttest$p.value #extract p value
    sig[i] <- (p < 0.05) #record significance
  }
  pow <- mean(sig) #compute power
}
cat("sample size =",N) #report result
```

```
## sample size = 17
```

Using the `paramtest` package

Simulating can be made a bit easier by using the `paramtest` package. The code that follows is taken from the `paramtest` vignette *Simulating Power*. Run `vignette("Simulating-Power")` to see the entire vignette.

First we create a function that generates random data based on parameters and then tests those generated data.

```
# create user-defined function to generate and analyze data
t_func <- function(simNum, N, d) {
    x1 <- rnorm(N, 0, 1)
    x2 <- rnorm(N, d, 1)

    t <- t.test(x1, x2, var.equal=TRUE)  # run t-test on generated data
    stat <- t$statistic
    p <- t$p.value

    return(c(t=stat, p=p, sig=(p < .05)))
        # return a named vector with the results we want to keep
}
```

Now we can use the `run_test` function to do the simulation.

```
power_ttest <- run_test(t_func, n.iter=5000,
                        output='data.frame', N=17, d=1)  # simulate data
```

```
## Running 5,000 tests...
```

```
results(power_ttest) %>%
    summarise(power=mean(sig))
```

```
##    power
## 1 0.8004
```

A great advantage of the `paramtest` package is the ability to vary parameters with the `grid_search` and `random_search` commands.

```
# give 'params' a list of parameters we want to vary;
# testing at sample sizes from 5, 10, 15 and 20.
power_ttest_vary <- grid_search(t_func, params=list(N=c(5,10,15,20)),
    n.iter=5000, output='data.frame', d=1)
```

```
## Running 20,000 tests...
```

```
results(power_ttest_vary) %>%
    group_by(N.test) %>%
    summarise(power=mean(sig))
```

```
## # A tibble: 4 x 2
##   N.test power
##    <dbl> <dbl>
## 1      5 0.296
## 2     10 0.553
## 3     15 0.749
## 4     20 0.866
```

Another advantage is that if we want to vary across two separate parameters, that is easy to do as well:

```
# varying N and Cohen's d
power_ttest_vary2 <- grid_search(t_func, params=list(N=c(25, 50, 100),
                                                     d=c(.2, .5)),
    n.iter=5000, output='data.frame')
```

```
## Running 30,000 tests...
```

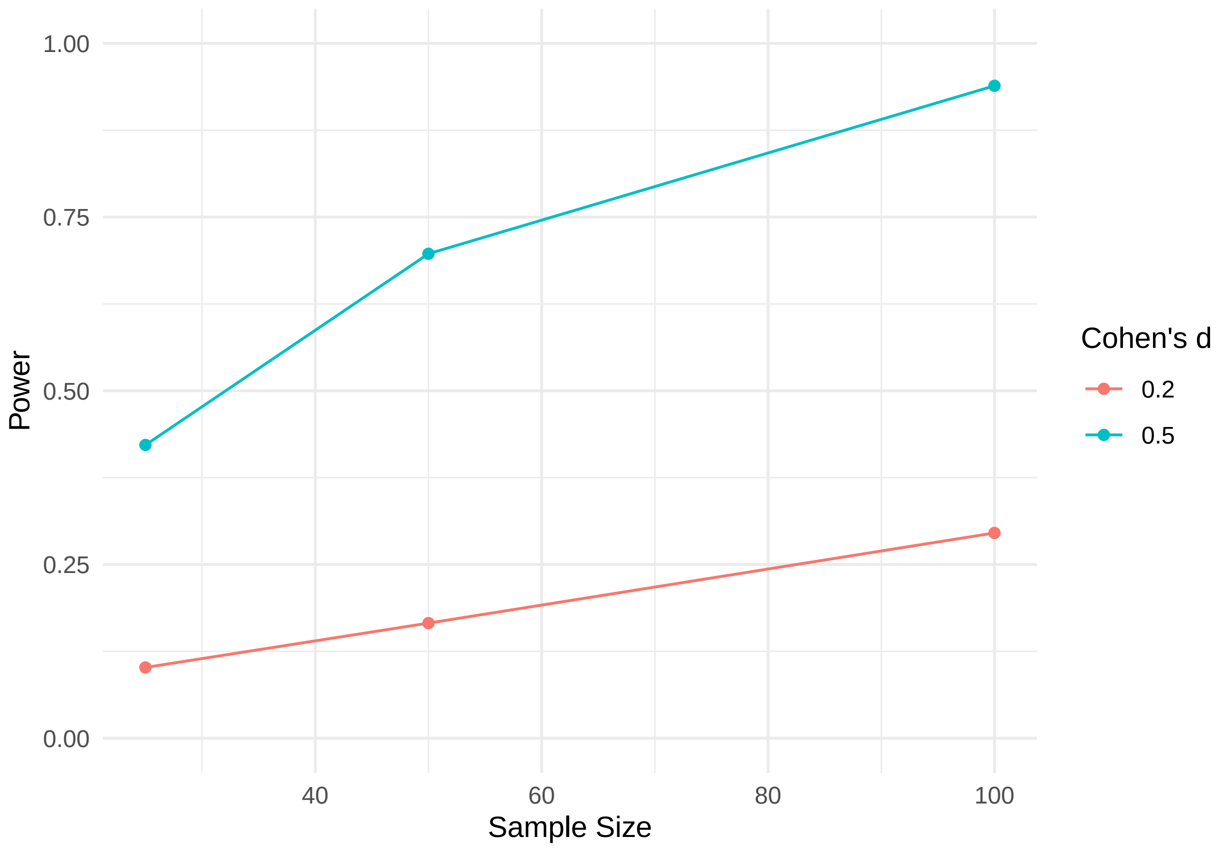
```
power <- results(power_ttest_vary2) %>%
    group_by(N.test, d.test) %>%
    summarise(power=mean(sig))
```

```
## `summarise()` has grouped output by 'N.test'. You can override using the `.groups` argument.
```

```
print(power)
```

```
## # A tibble: 6 x 3
## # Groups:   N.test [3]
##   N.test d.test power
##    <dbl>  <dbl> <dbl>
## 1     25    0.2 0.102
## 2     25    0.5 0.422
## 3     50    0.2 0.166
## 4     50    0.5 0.697
## 5    100    0.2 0.295
## 6    100    0.5 0.939
```

```
ggplot(power, aes(x=N.test, y=power,
                  group=factor(d.test), color=factor(d.test))) +
    geom_point() +
    geom_line() +
    ylim(c(0, 1)) +
    labs(x='Sample Size', y='Power', colour="Cohen's d") +
    theme_minimal()
```



(2) Continuing with the scenario from the first exercise, where $\mu_a = 50$, $\sigma_{pooled} = 6$, find the power for sample sizes of 50, 75, and 100, and for μ_b 53, 55, and 60.

(3) The `pwr.f2.test` function can be used to compute the power for a linear model. Parameters are u,

the number of predictors and v, the degrees of freedom (sample size less the number of coefficients, including the intercept). So a model with formula `y~x1+x2` built with 100 data points would have $u = 2$ and $v = 97$. In general, to find sample size compute $u + v + 1$ and round up to the next whole number. The measure of effect size is `f2` and it is the ratio of variance explained by predictors to variance not explained by predictors: $f^2 = \frac{R^2}{1-R^2}$. Find the sample size needed to detect a linear effect with an R^2 of 0.5 with a power of 80%, in the scenario with two predictors: $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$.

(4) Hopefully your answer to the above question was 14. Examples of linear models with a mean of about 0.5 for R^2 can be created by setting $\beta_1 = 0.7$ and the residual variance to $1 - \beta_1^2$.

```
b0 <- 0
b1 <- 0.7
x <- rnorm(N, 0, 1)
y <- rnorm(N, b0 + b1*x, sqrt(1 - b1^2))
lm(y~x)
```

This can be used with `grid_search` to confirm the `pwr.f2.test` computation using simulation. The following function (adapted from the same vignette) creates simulated data for x and y and tests the linear relationship between them.

```
lm_test <- function(simNum, N, b1, b0=0, xm=0, xsd=1) {
    x <- rnorm(N, xm, xsd)
    y <- rnorm(N, b0 + b1*x, sqrt(1 - b1^2))  # var. approx. 1 after accounting
                                              # for explained variance by x
    model <- lm(y ~ x)

    # pull output from model
    est <- coef(summary(model))['x', 'Estimate']
    se <- coef(summary(model))['x', 'Std. Error']
    p <- coef(summary(model))['x', 'Pr(>|t|)']
    r2 <- summary(model)$r.squared
    f2 <- r2/(1-r2)

    return(c(b1=b1, xm=mean(x), xsd=sd(x), ym=mean(y), ysd=sd(y), est=est,
             se=se, p=p, r2=r2, f2=f2, sig=(p < .05)))
}
```

Use this and the `grid_search` function to confirm that a slope of 0.7 does lead to a R^2 of approximately 0.5, and that in this case, a sample size of 14 provides sufficient power. (Warning, this can take a minute or two to run.)

```
power_lm <- grid_search(lm_test, params=list(N=13:15, b1=c(0.6,0.7,0.8)),
                        n.iter=5000, output='data.frame')
results(power_lm) %>%
    group_by(N.test, b1) %>%
    summarise(power=mean(sig),
              r2 = mean(r2))
```

Power calculations in more complicated scenarios

Since most real world data doesn't fit perfectly into classical statistical analysis, it's great to be able to compute power by simulation in order to design a study.

Consider the pigweed example from the lecture. Here is the code that randomly generates and computes a total seed set for a plot, encapsulated in a function.

```r
pigweed_seeds <- function(L=30,
                          nparents=50, offspr_per_parent = 10, dispdist=2,
                          M=2.3, alpha=0.49,
                          b = 271.6, k = 0.569){

  parent_x = runif(nparents,min=0,max=L)
  parent_y = runif(nparents,min=0,max=L)
  noffspr = nparents*offspr_per_parent
  angle = runif(noffspr,min=0,max=2*pi)
  dist = rexp(noffspr,1/dispdist)

  offspr_x = rep(parent_x,each=offspr_per_parent)+cos(angle)*dist
  offspr_y = rep(parent_y,each=offspr_per_parent)+sin(angle)*dist
  pos <- cbind(offspr_x,offspr_y); ndist <- as.matrix(dist(pos))

  nbrcrowd = rowSums(ndist<2)-1
  ci = nbrcrowd*3

  mass_det=M/(1+ci)
  mass = rgamma(length(mass_det),scale=mass_det,shape=alpha)

  seed_det <- b*mass
  seed <- rnbinom(length(seed_det),mu=seed_det,size=k)

  total_seeds <- sum(seed)
  return(total_seeds)
}
```

Calling this function runs a simulation and reports the total number of seeds in the simulation.

```r
pigweed_seeds()
```

```
## [1] 12706
```

(5) This function is missing the comments from the lecture, which you may wish to expand on. Comment the code above defining `pigweed_seeds` to describe what each part of the model is doing.

To create a distribution of the expected seed set under these assumptions, we can just call this function in a `for` loop and record the output.

```r
num_sim <- 10000
expected_seeds <- numeric(num_sim)
for(i in 1:num_sim){
  expected_seeds[i] <- pigweed_seeds()
}
```

Now we can use this distribution to test if a seed set is unexpectedly large or small based on this model. Writing this test into a function designed to work with `paramtest` will allow us to compute power. We'll include M and b as adjustable parameters, since these have more clear biological meaning in terms of plant mass and seed production.

```r
pigweed_test <- function(simNum, M=2.3, b=271.6){
  test_seeds <- pigweed_seeds(M=M, b=b)
  p_greater <- mean(test_seeds < expected_seeds)
  p_less <- mean(test_seeds > expected_seeds)
  sig <- (2*p_greater<0.05)|(2*p_less<0.05)
  return(c(sig=sig, p_greater=p_greater, p_less=p_less))
}
```

Calling `pigweed_test` allows us to compare a plot with potentially alternate values of M and b to the expectation based on the model. For example, if the mass coefficient was increased to 3 and the seed production coefficient decreased to 250, we might see the following scenario, where approximately 80% of the plots in the simulated distribution have fewer seeds than the randomly generated test plot.

```
pigweed_test(M=3, b=250)
```

```
##       sig p_greater    p_less
##    0.0000    0.1814    0.8186
```

To evaluate the power of experiments where we create test plots with pigweed variants having different values of M and b, we can use `grid_search`. We'll only run 500 tests per combination of variables, since each test involves a simulation.

```
power_pigweed_test <- grid_search(pigweed_test,
                                  params = list(M=c(2,2.3,3),
                                                b=c(200,271.6,300)),
                                  n.iter=500, output = 'data.frame')
```

```
## Running 4,500 tests...
```

```
results(power_pigweed_test) %>%
  group_by(M.test, b.test) %>%
  summarise(power=mean(sig))
```

```
## `summarise()` has grouped output by 'M.test'. You can override using the `.groups` argument.
```

```
## # A tibble: 9 x 3
## # Groups:   M.test [3]
##   M.test b.test power
##    <dbl>  <dbl> <dbl>
## 1    2      200  0.476
## 2    2      272. 0.106
## 3    2      300  0.052
## 4    2.3    200  0.28
## 5    2.3    272. 0.036
## 6    2.3    300  0.056
## 7    3      200  0.046
## 8    3      272. 0.142
## 9    3      300  0.202
```

Apparently, these parameter shifts are not large enough to generate significant change in seed production very often.

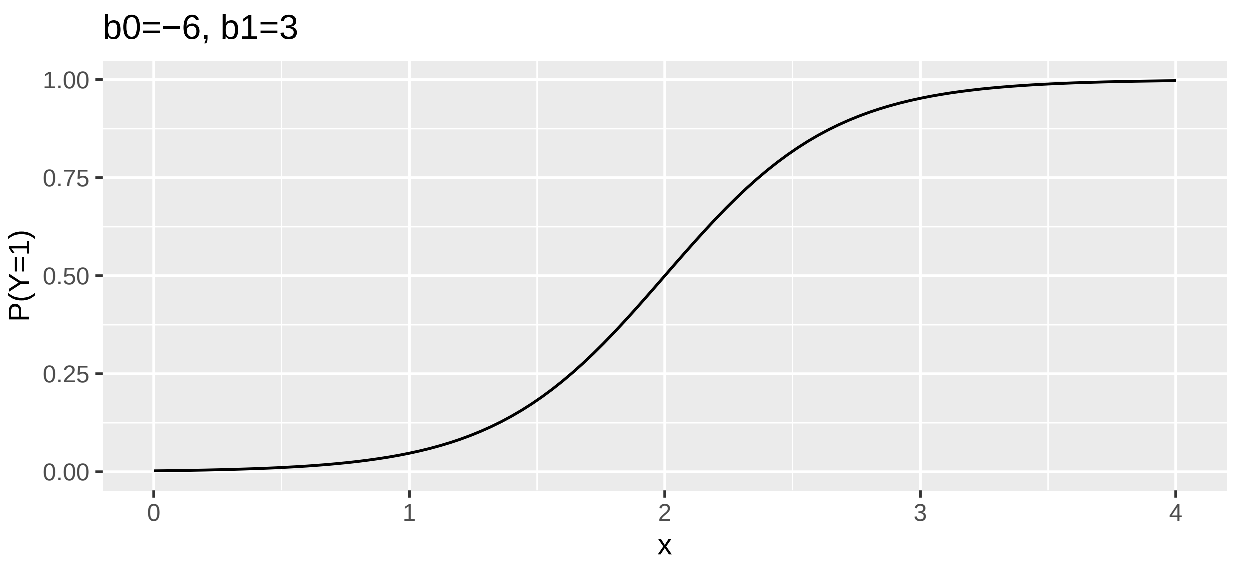
(6) Using the tools we have developed, determine values of M and b that would be likely (power over 50%) to produce plots with significantly different seed production.

Power for logistic regression

Recall that the signal part of a binomial glm is the logistic function

$$f(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

The inflection point is at $-\beta_0/\beta_1$, and the slope at this inflection point is $\beta_1/4$.



Logistic regression interprets $f(x)$ as the probability that the (binary categorical) response is "success" or $Y = 1$. The steepness of the logistic curve is therefore one way of quantifying the effect of the predictor on the response. A steeper curve means that smaller changes in x have more pronounced effects on the probability of $Y = 1$.

(7) The `lm_test` function above can be rewritten for a binomial glm:

```
binom_glm_test <- function(simNum, N, b1, b0=1, xm=0, xsd=1) {
  x <- rnorm(N, xm, xsd)
  y <- rbinom(N, size=1, prob = plogis(b0 + b1*x))
  model <- glm(y ~ x, family = binomial)

  # pull output from model
  est <- coef(summary(model))['x', 'Estimate']
  se <- coef(summary(model))['x', 'Std. Error']
  p <- coef(summary(model))['x', 'Pr(>|z|)']

  return(c(xm=mean(x), xsd=sd(x),
           est=est, se=se, p=p,
           sig=(p < .05)))
}
```

As explained above, effect size can be parameterized by the coefficient β_1 or `b1` of x in this model. Determine the sample size needed for 80% power if $\beta_1 = 1$.

References

Cohen, J. 1988. *Statistical Power Analysis for the Behavioral Sciences (2nd Ed.).* Routledge. https://doi.org/https://doi.org/10.4324/9780203771587.