Sample size calculation in R

Martin Otava

11 januari 2018 (11:13:24)

SBS example

This example has been used in class and we will use it in several calculations below. We set the framework as experiment to investigate effect of hypothetical drug on blood pressure 1. Two samples of independent patients 2. Compare systolic blood pressure (SBP) in a treatment group and control group 3. SBP is measured using a standard sphygmomanometer 4. Treatment is expected to reduce SBP

Question is how many patients should we include in control group and test group.

For calculations below, we will assume following choices:

1. Significance level to control Type I error set to 0.05
2. Power of 80%
3. Variance of 400 (standard deviation 20)
4. Effect size of smallest effect of interest to be reduction of 15

Z-test manually

Let us use the formula for one sample Z-test explicitly:

1. `zA` quantile of standard normal distribution related to Type I error
2. `zB` quantile of standard normal distribution related to power (Type II error)
3. `sigma^2` variance of the distribution
4. `delta` effect size that we are interested to detect

```
sampleSizeZtest <-  function(alpha, sigma, beta, delta){
    zA <-  qnorm(p = alpha, mean = 0, sd = 1)
    zB <-  qnorm(p = beta, mean = 0, sd = 1)
    sampleSize <- 2*(zA-zB)^2*(sigma^2/delta^2)
    return(sampleSize)
}
```

For SBS example, we will have

```
sampleSizeZtest(alpha = 0.05, sigma=20, beta=0.85, delta=15)
```

```
## [1] 25.56196
```

Note that we have slightly different result than in class (26 instead of 27). The reason is that we did not consider known variance in class, while Z-test does have this assumtion. Hence, there is less information needed to be estimated, so it is natural that less samples are needed here.

Core options

R core library `stats` is fully reliable tool and very simple to use, but it offers solution for several basic sample size calculations frameworks.

t test

```
power.t.test(n = , delta = 15, sig.level = 0.05, sd = 20, power = 0.85, type = "two.sample",
        alternative = "one.sided", strict = TRUE)
```

```
##
##      Two-sample t test power calculation
##
##               n = 26.26614
##           delta = 15
##              sd = 20
##       sig.level = 0.05
##           power = 0.85
##     alternative = one.sided
##
## NOTE: n is number in *each* group
```

Notice that we fill in all the items except the `n` that we wish to calculate. This is the function that was used to obtain 27 patients per group as result for SBP data set. Indeed, we do not obtain value of 27 precisely, but 26.26614. Reasonable practice is to round the result up to next higher integer value. If we wish to know exact of the power achieved with 27 patients, we can proceed with filling in `n` and leaving `power` empty:

```
power.t.test(n = 27, delta = 15, sig.level = 0.05, sd = 20, power = NULL, type = "two.sample",
        alternative = "one.sided", strict = TRUE)
```

```
##
##      Two-sample t test power calculation
##
##               n = 27
##           delta = 15
##              sd = 20
##       sig.level = 0.05
##           power = 0.8587416
##     alternative = one.sided
##
## NOTE: n is number in *each* group
```

We reach power of almost 86%.

In classical situations, `strict` intepretation is used that assumes successful rejection regardless sign of the effect. Hence, even situations when true effect is positive, but large negative effect is observed, are considered as good result from power perspective. It is worth to investigate performance of this option. For details, refer to help of the function:

```
?power.t.test
```

Binary data

In case of comparison of two proportions, i.e. probabilties of success of two uniform or binary distributions, not only difference between proportions p1 and p2, but also actual value is important. That is caused by dependence between mean and variance p1(1-p1)+p2(1-p2).

Therefore, although the absolute value of difference is 0.3 in both following examples, we obtain different sample size:

```r
power.prop.test(n = NULL, p1 = 0.1, p2 = 0.4, sig.level = 0.05, power = 0.80)
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 31.49838
##              p1 = 0.1
##              p2 = 0.4
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

```r
power.prop.test(n = NULL, p1 = 0.5, p2 = 0.8, sig.level = 0.05, power = 0.80)
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 38.48004
##              p1 = 0.5
##              p2 = 0.8
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Note that it won't be case in following examples and we will obtain same sample size, due to symmetry around 0.5

```r
power.prop.test(n = NULL, p1 = 0.1, p2 = 0.4, sig.level = 0.05, power = 0.80)
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 31.49838
##              p1 = 0.1
##              p2 = 0.4
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

```r
power.prop.test(n = NULL, p1 = 0.6, p2 = 0.9, sig.level = 0.05, power = 0.80)
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 31.49838
##              p1 = 0.6
##              p2 = 0.9
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

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

Indeed, the variance of both examples is same: $0.1*(1-0.1)+0.4*(1-0.4) = 0.33 = 0.9*(1-0.9)+0.6*(1-0.6)$

ANOVA data

In case of ANOVA, following paramters are needed:

1. Number of groups: important for correct specification of F distribution
2. Between variability: parameter determined by assumed effect size
3. Within variability: nuisance parameter for testing the classical ANOVA hypothesis of means equality

```
power.anova.test(groups = 5, n = NULL, between.var = 1, within.var = 5, sig.level = 0.05, power = 0.9)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##           groups = 5
##                n = 20.23026
##     between.var = 1
##      within.var = 5
##       sig.level = 0.05
##           power = 0.9
##
## NOTE: n is number in each group
```

Note that in case of five groups, between variability does not depend only on maximal difference, but on actual means of all groups. For example, we will achieve different results for following effect sizes:

```
power.anova.test(groups = 5, n = NULL, between.var = var(c(10, 10, 10, 10, 15)), within.var = 5, sig.lev
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##           groups = 5
##                n = 4.909918
##     between.var = 5
##      within.var = 5
##       sig.level = 0.05
##           power = 0.9
##
## NOTE: n is number in each group
```

```
power.anova.test(groups = 5, n = NULL, between.var = var(c(10, 15, 10, 10, 15)), within.var = 5, sig.lev
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##           groups = 5
##                n = 3.666669
##     between.var = 7.5
##      within.var = 5
##       sig.level = 0.05
##           power = 0.9
##
## NOTE: n is number in each group
```

The difference is quite high in relative sense given that small sample size is sufficient. Let us see what happens if we would have used second setting for calculations, but we are really interested in detecting the first setting as well:

```
power.anova.test(groups = 4, n = 4, between.var = var(c(10, 10, 10, 10, 15)), within.var = 5, sig.level
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##          groups = 4
##               n = 4
##     between.var = 5
##      within.var = 5
##       sig.level = 0.05
##           power = 0.6927136
##
## NOTE: n is number in each group
```

Power of detecting this type of setting, i.e. one group only being different from the rest, is actually only 69% instead of 90%. In practice, reasonable approach would be to clearly identify the settings of interest and then run calculation across all of them, selecting the maximum. In case of no prior information, we can always test all the settings and keep maximum needed.

Specialized libraries

For more advanced situations, specialized libraries needs to be used. There are many options in R regarding the calculations of sample size, some of them more general, while others field and context specific. In general, caution is needed while using third party libraries and thorough study of manual and codes is recommended.

longpower: Correlated data

The core function of this library is `lmmpower`. However, it is rather complex function and it is not that easy to understand it fully. Checking the help files is recommended:

```
?lmmpower
```

Note that the help file uses sentence "in the pilot estimate of the parameter of interest"; which is not correct, the effect size of interest should be always used, not pilot study result

```
lmmpower(delta = 1.5, t = seq(0, 1.5, 0.25),
    sig2.i = 55, sig2.s = 24, sig2.e = 10, cov.s.i = 0.8*sqrt(55)*sqrt(24), power = 0.80)
```

```
##
##      Power for longitudinal linear model with random slope (Edland, 2009)
##
##               n = 207.3101
##           delta = 1.5
##          sig2.s = 24
##          sig2.e = 10
##       sig.level = 0.05
##               t = 0.00, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50
##           power = 0.8
##     alternative = two.sided
##
##  NOTE: n is number in *each* group
```

The parameter of interest in this example is `delta` and value of 1.5. Variance and covariance structure are estimated form the pilot study and covariance structure is rather complex in this example. Note that `t` represents sampling points for longitudinal studies and that sample size may depend on this structure, so correct specification is important.

Alternative parametrization is possible with `beta` standing for pilot estimate of placebo effect and effect size is determined by `pct.change` as $delta = 1.5 = 5 * 0.3 = beta * pct.change$. Therefore, result is exactly the same

```
lmmpower(beta = 5, pct.change = 0.30, t = seq(0, 1.5, 0.25),
    sig2.i = 55, sig2.s = 24, sig2.e = 10, cov.s.i=0.8*sqrt(55)*sqrt(24), power = 0.80)
```

```
##
##      Power for longitudinal linear model with random slope (Edland, 2009)
##
##               n = 207.3101
##           delta = 1.5
##          sig2.s = 24
##          sig2.e = 10
##       sig.level = 0.05
##               t = 0.00, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50
##           power = 0.8
##     alternative = two.sided
##            beta = 5
##
##  NOTE: n is number in *each* group
```

Instead of such specification, you can actually use directly pilot data set.However, you need to be *very* careful what the function takes from the data besides variance-covariance structure. In example below, `beta` is actually estimated from the pilot data, which is typically not appropriate, since effect of interest (not one observed in pilot data), should be used for sample size calculations.

```
data(sleepstudy)
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
lmmpower(fm1, pct.change = 0.30, t = seq(0,9,1), power = 0.80)
```

```
##
##      Power for longitudinal linear model with random slope (Edland, 2009)
##
##               n = 68.46993
##           delta = 3.140186
##          sig2.s = 35.07166
##          sig2.e = 654.941
##       sig.level = 0.05
##               t = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
##           power = 0.8
##     alternative = two.sided
##        delta.CI = 2.231279, 4.049093
##            Days = 10.46729
##         Days CI = 7.437595, 13.496977
##            n.CI = 41.18083, 135.61351
##
##  NOTE: n is number in *each* group
```

However, if you specify anything in the function itself, the pilot estimate won't be used.

```
lmmpower(fm1, delta = 1.5, t = seq(0,9,1), power = 0.80)
```

```
##
##      Power for longitudinal linear model with random slope (Edland, 2009)
##
##               n = 300.0738
##           delta = 1.5
##          sig2.s = 35.07166
##          sig2.e = 654.941
##       sig.level = 0.05
##               t = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
##           power = 0.8
##     alternative = two.sided
##        delta.CI = 1.065834, 1.934166
##            Days = 10.46729
##         Days CI = 7.437595, 13.496977
##            n.CI = 180.4776, 594.3348
##
##  NOTE: n is number in *each* group
```

```
lmmpower(fm1, delta = 1.5, t = seq(0,9,1), power = 0.80, sig2.i = 10, sig2.s = 5, sig2.e = 7)
```

```
##
##      Power for longitudinal linear model with random slope (Edland, 2009)
##
##               n = 35.47588
##           delta = 1.5
##          sig2.s = 5
##          sig2.e = 7
##       sig.level = 0.05
##               t = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
##           power = 0.8
##     alternative = two.sided
##        delta.CI = 1.065834, 1.934166
##            Days = 10.46729
##         Days CI = 7.437595, 13.496977
##            n.CI = 21.33675, 70.26455
##
##  NOTE: n is number in *each* group
```

Instead of `lmer`, `lme` function can be used instead as input:

```
# random intercept and slope
#fm2 <- lme(Reaction ~ Days, random = ~Days|Subject, sleepstudy)
#lmmpower(fm2, delta = 1.5, t = seq(0,9,1), power = 0.80)
# random intercept only
#fm3 <- lme(Reaction ~ Days, random=~1|Subject, sleepstudy)
#lmmpower(fm3, delta = 1.5, t = seq(0,9,1), power = 0.80)
```

Similarly other functions can be used:

```
fm4 <- gee(Reaction ~ Days, id = Subject,
           data = sleepstudy,
           corstr = "exchangeable")
lmmpower(fm4, pct.change = 0.30, t = seq(0,9,1), power = 0.80)
```

The help file assists greatly in understanding `lmmpower` function and contains references to the formulas used for calculations behind: Diggle (2002) and Liu and Liang (1997).

For example, you may wish to compare slopes using approach of Diggle:

```
diggle.linear.power(n = NULL, delta = 0.5, t = seq(0,1.5,0.25), sigma2 = 1,
R = 0.8, sig.level = 0.05, power = 0.8, alternative = "two.sided", tol = .Machine$double.eps^2)
```

It is strongly recommended to consult help files, vignette and/or underlying references before using this library:

```
?lmmpower
browseVignettes(package = "longpower")
```

pwr library

This library is rather example of how librarires in open source community should not look like. Although it offers interesting tools and extends on `stats` library, it is almost impossible to use it successfully without consulting respective book. Problem is that tranformed effect sizes are used and they are not described sufficiently in the help files. Therefore, original reference needs to be consulted: Cohen (1988). Cohen's book is rather famous in some disciplines and therefore `pwr` library is often cited as the library for sample size calculation in `R`. However, I have doubts on its clarity and also value of some of the proposed tests. Let us look at multiple cases in detail.

t test

Effect size is $d = (mu1 - mu2)/sigma = delta/sigma$. Hence, one sample t-test is

```
pwr.t.test(d = -15/20, power = 0.85, sig.level = 0.05, type = "two.sample", alternative= "less")
```

```
##
##      Two-sample t test power calculation
##
##               n = 26.26614
##               d = -0.75
##       sig.level = 0.05
##           power = 0.85
##     alternative = less
##
## NOTE: n is number in *each* group
```

Note that this code gives same result for SBP example, as `stats` library. However, we need to know that `delta` is actually ratio of effect size and standard deviation.

ANOVA

Effect size `f` is function of grouped variances against pooled variance:

```
pwr.anova.test(k = 5, f = 0.25, sig.level = 0.05, power = 0.8)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 5
##               n = 39.1534
##               f = 0.25
```

```
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: n is number in each group
```

Binary data

With p1 and p2 being proportions in each sample, effect size should be $abs(2 * asin(sqrt(p1)) - 2 * asin(sqrt(p2)))$.

```
p1 <- 0.1
p2 <- 0.4
(ES <- abs(2*asin(sqrt(p1)) - 2*asin(sqrt(p2))))
```

```
## [1] 0.7259373
```

```
pwr.2p.test(n =, h = ES, sig.level = 0.05, power = 0.80)
```

```
##
##      Difference of proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.7259373
##               n = 29.7878
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: same sample sizes
```

However, if we try to compare to `stats`, we obtain different result. It can be due to different approximation used, or I have found wrong formula in the book (or some other reasons). Without formula directly in help file, it is not possible to identify root cause without going through codes in detail.

```
power.prop.test(n = NULL, p1 = 0.1, p2 = 0.4, sig.level = 0.05, power = 0.80)
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 31.49838
##              p1 = 0.1
##              p2 = 0.4
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Correlation test

Correlation coefficient `r` is tested against hypothesis of being zero

```
pwr.r.test(r = 0.3, n = NULL, sig.level = 0.05, power = 0.85, alternative = "two.sided")
```

```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 95.85551
```

```
##               r = 0.3
##       sig.level = 0.05
##           power = 0.85
##     alternative = two.sided
```

Note that practical value of such test is disputable. Even highly significant result may have little pracical value, if the estimated correlation (point estimate or confidence interval) is not sufficiently high.

Linear regression

In this case, `f2` represents transformation of R^2 and `u` and `v` degrees of freedom.

```
pwr.f2.test(u = 5, v = 89, f2 = 0.1/(1-0.1), sig.level = 0.05)
```

```
##
##      Multiple regression power calculation
##
##               u = 5
##               v = 89
##              f2 = 0.1111111
##       sig.level = 0.05
##           power = 0.6735858
```

Chisq test

Effect size is function of counts under H_0 and H_1 in each cell

```
pwr.chisq.test(w = 0.289, df = (4-1), N = NULL, sig.level = 0.05, power = 0.8)
```

```
##
##      Chi squared power calculation
##
##               w = 0.289
##               N = 130.5368
##              df = 3
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: N is the number of observations
```

Examples of other useful libraries

1. `TrialSize` for sample size calculation in clinical research
2. `powerSurvEpi` for survival analysis in epidemiological studies
3. `asypow` very powerful library for calculations based on asymptotic methods
4. `powerTOST` power for equivalence studies
5. `clusterPower` power for cluster-randomized crossover trials

Simulations

Case study: Poisson data

Poisson data generally poses challenge in the sample size calculation due to fact that variance and mean are equal. Hence, pilot knowledge on variance immediately imply knowledge on effect size. There exists some

solution in library `Sequential` and also library `asypow` mentioned above can be used.

```
?SampleSize.Poisson
?asypow.n
```

However, the simulation approach can be very efficient solution for Poisson data.

Simple example: t-test

Let us start with simpler example below to demonstrate the essential principles of simulation approach towards sample size calculation. Indeed, in following setting, simulation would not be necessary since there is simple analytical solution.

First, let us choose the sample size to start with. It can be completely random choice, since the approach we are taking is iterative and we are just specifying starting condition.

```
#  1. Fix sample size N
N <- 10 # choose one fixed N
```

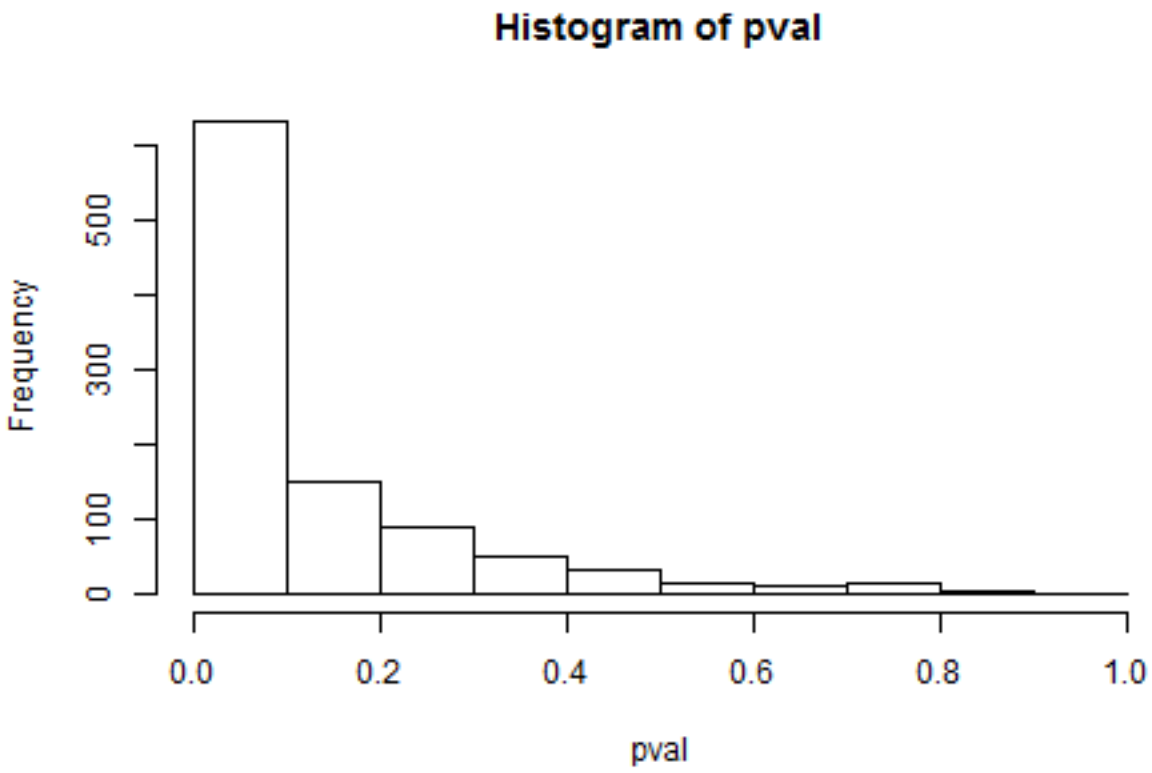
We have to choose significance level, effect size and nuisance parameters (variance).

```
# 2. Fix other parameters
delta = 15
sdChoice = 20
alpha = 0.05
```

The core part of the simulation approach is that we will simulate hundreds of data sets from distribution with exactly the specified effect size and then we check how many times we would reject appropriate statistical test. The portion of rejected tests is approximate power of the test under given effect size and sample size.

```
# 3. Simulate huge number of experiments
numberSimulation <- 1000
pval <- numeric(numberSimulation) # here we store p-value of 1000 tests

set.seed(1234) # set the seed for reproducible resuts
for (i in 1:numberSimulation){
    # we set any mean we wish, it does not matter, only important is to keep difference delta
  # [this holds for this particular setting of normal distribution and t-test]
  # we simulate from normal distribution
    controlGroup <- rnorm(N, mean = 120, sd = sdChoice)
    treatedGroup <- rnorm(N, mean = 120-delta, sd = sdChoice)
    # we perform the t-test on the data and keep the p-value
    pval[i] <- t.test(controlGroup,treatedGroup, alternative = "greater",
            mu = 0, paired = FALSE, var.equal = TRUE, conf.level = 1-alpha)$p.value
}
hist(pval)
```




```
# power translates to: how often we reject if true effect is delta?
# (if given significance level alpha = 0.05 an sd = 20)
```

Now, we can estimate the power if true effect size is as specified, assuming specified significance level and variance.

```
# 4. Estimate power
sum(pval<0.05)/numberSimulation
```

```
## [1] 0.478
```

```
# here we achieve power only 0.476
```

Estimated power is only 47.5%. Last step would be to increase `N` and run the whole code again. We repeat this procedure, until desired power is achieved.

Naturally, the whoel approach can be automated, so the change of power does not need to be done manually. Simple solution is running over loop of of N, given that it does not take too much computational time. Otherwise, more advanced iterative method can be applied.

```
nvec <- seq(15, 30, by = 1)
numberSimulation <- 1000
pval <- numeric(numberSimulation)

sampleSizeCalculations <- numeric(length(nvec))
names(sampleSizeCalculations) <- nvec

set.seed(1234)
```

```
for (j in 1:length(nvec)){
    for (i in 1:numberSimulation){
        controlGroup <- rnorm(nvec[j], mean = 120, sd = 20)
        treatedGroup <- rnorm(nvec[j], mean = 120 - 15, sd = 20)
        pval[i] <- t.test(controlGroup,treatedGroup, alternative = "greater",
                mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95)$p.value
    }
    sampleSizeCalculations[j] <- sum(pval < 0.05)/numberSimulation
}
# power for all sample sizes between 15 and 30
sampleSizeCalculations
```

```
##    15    16    17    18    19    20    21    22    23    24    25    26
## 0.630 0.643 0.720 0.721 0.738 0.751 0.759 0.791 0.805 0.827 0.848 0.847
##    27    28    29    30
## 0.860 0.870 0.912 0.890
```

```
which(sampleSizeCalculations>0.85)
```

```
## 27 28 29 30
## 13 14 15 16
```

The result suggests that we wshould use 27 observations per group. As expected, the result correspond to output of automated function from `stats` library.

```
power.t.test(n=, delta=15, sig.level=0.05, sd=20, power=0.85, type = "two.sample",
        alternative="one.sided", strict=TRUE)
```

```
##
##      Two-sample t test power calculation
##
##               n = 26.26614
##           delta = 15
##              sd = 20
##       sig.level = 0.05
##           power = 0.85
##     alternative = one.sided
##
## NOTE: n is number in *each* group
```

Poisson distribution simulation

Our task will be to compare two samples with different parameter lambda. We start with random choice of `N` and proceed with choice of significance level. However, although we are primarily interested in difference, we need to specify actual values of two means. There is no way around this for Poisson distribution. There is clear paradox, becuase if we know these values, we do not need experiment. Hence, the suggested strategy can be to run following code for smallest and largetst considered possible value of parameter (maybe based on some pilot evaluation) and then select sample size appropriatelly. Another option can be to run medium size pilot experiment to get reasonable estimate effect for control group and then specify treated group per desired effect size.

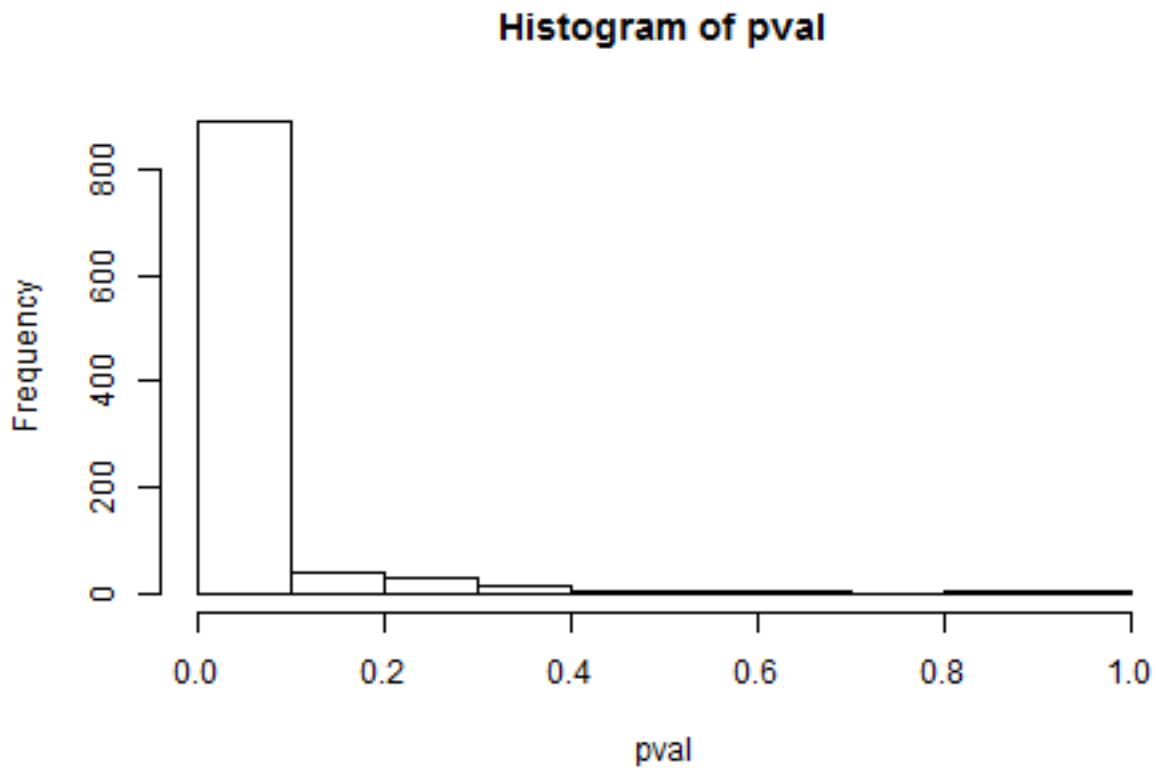
Note that similar approach would need to be considered for Binary distribution.

```
# 1. Choose one fixed N
N <- 50
```

```r
# 2. Select parameters
control = 7
treated = 5.6
alpha = 0.05


# 3. Simulate huge number of experiments and test
numberSimulation <- 1000
pval <- numeric(numberSimulation)

set.seed(1234) # set the seed for reproducible resuts
for (i in 1:numberSimulation){
  # we simulate from Poisson distribution
  controlGroup <- rpois(N, lambda = control)
  treatedGroup <- rpois(N, lambda = treated)
  # we use GLM model for Poisson regression to test effect of treatment
  simData <- data.frame(response = c(controlGroup, treatedGroup), treatment = rep(c(0,1),
                                                                                  each = N))
  pval[i] <- summary(glm(response ~ treatment, data = simData, family=poisson()))$coeff["treatment", "P
}
hist(pval)
```




```r
# Estimate power
sum(pval < 0.05)/numberSimulation
```

```
## [1] 0.822
```

We achieved power of 82.2% in this case. If required power would be 85%, we may need to increase `N` slightly and rerun the code.

Design for more complex data set: Linear regression

Simulation approach gets more complicated, if we consider more complex statistical model and test. especially if covariance structure may play role, the simulation of data needs to be performed very carefully. However, the basic idea is always the same:

- Create the simulated data set with effect of interest with fixed 'N'
- Perform the test of interest
- Run in a loop many times and count how many times you reject to estimate power
- Adjust 'N' appropriately and iterate

Let us assume model $outcome = \beta_0 + \beta_1 * gender + \beta_2 * country + \beta_3 * age$. Let us say that we wish to mainly focus on effect of gender and effect size of interest is five, i.e. $\beta_1 = 5$.

We start with specifying the initial values for parameters. Be careful with meaning of specified sample size variable: if it refers to total sample size or per some group or combination of groups. See `n` below.

```
Nsimulations <- 1000
n <- 10 # per gender+country
sdChoice <- 10
effectSize <- 5
```

We continue with creating the data set.

```
gender <- rep(c(0,1), each = n*2)
country <- rep(rep(seq(0, 1), each = n), 2)
set.seed(1234)
age <- sample(seq(15, 50, by = 1), size = n*2*2, replace = TRUE)

# our structure so far
data.frame(gender = as.factor(gender), country = as.factor(country),  age = age)
```

```
##    gender country age
## 1       0       0  19
## 2       0       0  37
## 3       0       0  36
## 4       0       0  37
## 5       0       0  45
## 6       0       0  38
## 7       0       0  15
## 8       0       0  23
## 9       0       0  38
## 10      0       0  33
## 11      0       1  39
## 12      0       1  34
## 13      0       1  25
## 14      0       1  48
## 15      0       1  25
```

```
## 16      0       1  45
## 17      0       1  25
## 18      0       1  24
## 19      0       1  21
## 20      0       1  23
## 21      1       0  26
## 22      1       0  25
## 23      1       0  20
## 24      1       0  16
## 25      1       0  22
## 26      1       0  44
## 27      1       0  33
## 28      1       0  47
## 29      1       0  44
## 30      1       0  16
## 31      1       1  31
## 32      1       1  24
## 33      1       1  25
## 34      1       1  33
## 35      1       1  21
## 36      1       1  42
## 37      1       1  22
## 38      1       1  24
## 39      1       1  50
## 40      1       1  44
```

Note that for age, we have taken random values between some range. In practice, you would often have good idea of age distribution in your population and you can simulate from such distribution.

Now we will establish deterministic mean value based on model specified above. Note that we can actually specify any value for other parameters, because as we see later, they do not have any influence on result for gender.

```
meanResponse <- 5  + effectSize*gender - 2*country + 0.7* age

# we add response for each observation
data.frame(gender = as.factor(gender), country = as.factor(country),  age = age,
           meanResponse = meanResponse)
```

```
##    gender country age meanResponse
## 1       0       0  19         18.3
## 2       0       0  37         30.9
## 3       0       0  36         30.2
## 4       0       0  37         30.9
## 5       0       0  45         36.5
## 6       0       0  38         31.6
## 7       0       0  15         15.5
## 8       0       0  23         21.1
## 9       0       0  38         31.6
## 10      0       0  33         28.1
## 11      0       1  39         30.3
## 12      0       1  34         26.8
## 13      0       1  25         20.5
## 14      0       1  48         36.6
## 15      0       1  25         20.5
## 16      0       1  45         34.5
```

```
## 17      0       1  25         20.5
## 18      0       1  24         19.8
## 19      0       1  21         17.7
## 20      0       1  23         19.1
## 21      1       0  26         28.2
## 22      1       0  25         27.5
## 23      1       0  20         24.0
## 24      1       0  16         21.2
## 25      1       0  22         25.4
## 26      1       0  44         40.8
## 27      1       0  33         33.1
## 28      1       0  47         42.9
## 29      1       0  44         40.8
## 30      1       0  16         21.2
## 31      1       1  31         29.7
## 32      1       1  24         24.8
## 33      1       1  25         25.5
## 34      1       1  33         31.1
## 35      1       1  21         22.7
## 36      1       1  42         37.4
## 37      1       1  22         23.4
## 38      1       1  24         24.8
## 39      1       1  50         43.0
## 40      1       1  44         38.8
```

Note that we have performed these steps outside of the loop, becuase there has not been any randomness so far. Adding the random error to the observations is the first step that needs to be performed within the loop.

```
set.seed(1234)
pvaluesSim <- numeric(Nsimulations)
for (i in 1:Nsimulations){
  # simulated response is the mean response from model plus the random error
    response <- meanResponse + rnorm(length(meanResponse ), 0, sd = sdChoice)
    # the final data set used for each simulation step contains random error as well
    simulatedDataSet <- data.frame(gender = as.factor(gender), country = as.factor(country ),
        response = response, age = age)
    # keep the p-value for testing for gender
    pvaluesSim[i] <- summary(lm(response ~ gender + country + age, data = simulatedDataSet))$coeff["gen
}

# Estimate the power
sum(pvaluesSim < 0.05)/Nsimulations
```

```
## [1] 0.344
```

The estimated power is only 34.4%, so we will need to increase the `n` per group. Note that the result depends on `seed` specified above. Hence, it is important to track the seed and note that changing the seed may have small influence on reported result, in this case, we obtain 34.1%. How large the difference is indeed depends on amount of simulated data sets that we have used.

```
set.seed(5678)
pvaluesSim <- numeric(Nsimulations)
for (i in 1:Nsimulations){
  # simulated response is the mean response from model plus the random error
    response <- meanResponse + rnorm(length(meanResponse ), 0, sd = sdChoice)
    # the final data set used for each simulation step contains random error as well
```

```
    simulatedDataSet <- data.frame(gender = as.factor(gender), country = as.factor(country ),
        response = response, age = age)
    # keep the p-value for testing for gender
    pvaluesSim[i] <- summary(lm(response ~ gender + country + age, data = simulatedDataSet))$coeff["gen
}

# Estimate the power
sum(pvaluesSim < 0.05)/Nsimulations
```

```
## [1] 0.341
```

Finally, let us check how we will change the if we change the parameters for other variables. There may be small difference even if same seed is used, becuase the data set is different in this case.

```
gender <- rep(c(0,1), each = n*2)
country <- rep(rep(seq(0, 1), each = n), 2)
set.seed(1234)
age <- sample(seq(15,50,by=1), size = n*2*2, replace = TRUE)

# change the values of betas below:
meanResponse <- -3 + effectSize*gender - 0.1*country + 12* age
set.seed(1234)
pvaluesSim <- numeric(Nsimulations)
for (i in 1:Nsimulations){
  response <- meanResponse + rnorm(length(meanResponse ), 0, sd = sdChoice)
  simulatedDataSet <- data.frame(gender = as.factor(gender), country = as.factor(country ),
                                 response = response, age = age)
  pvaluesSim[i] <- summary(lm(response ~ gender + country + age, data = simulatedDataSet))$coeff["gende
}
sum(pvaluesSim < 0.05)/Nsimulations
```

```
## [1] 0.344
```

Very important note is that statement above only hold if main effects solely are used. If there are interactions, the result depends both on variable of interest as on values of all related variables.