The sse-Package: Commented Examples

Thomas Fabbro

27. March 2019

```
> library(sse)
```

1 Starting with a minimal example

How sensitive is the estimation of the sample size to changes in the effect size?

In a first step we crate an object of class powPar. We investigate the power for a range of effect sizes (0.5 to 1.5) and a range of sample sizes (20 to 60).

```
> psi <- powPar(theta = seq(from = 0.5, to = 1.5, by = 0.05),
+               n = seq(from = 20, to = 60, by = 10))
```

In a second step we write a function that calculates the power. The function should take only one argument, the object of class powPar we created in the last step. Two methods called n and theta allow to extract the elements of the object during evaluation.

```
> powFun <- function(psi)
+ {
+   return(power.t.test(n = n(psi)/2,
+                       delta = theta(psi),
+                       sig.level = 0.05)$power)
+ }
```

In the third step we create an object of class powCalc using a function with the same name. The function powCalc evaluates the function powFun step by step for all combinations of n and theta in the object psi.

```
> calc <- powCalc(psi, statistic = powFun)
```

The "hidden" result of the last steps is a matrix that contains the power of all combinations of n and theta. Now we need to define for which theta we would like to know the sample size with a certain power.

```
> pow <- powEx(x = calc, theta = 1, power = 0.9)
```

Now we are done and can extract the information we need. But before we start we have a look at the relationship between the power and the sample size for the theta we are mainly interested in.

```
> inspect(pow)
```

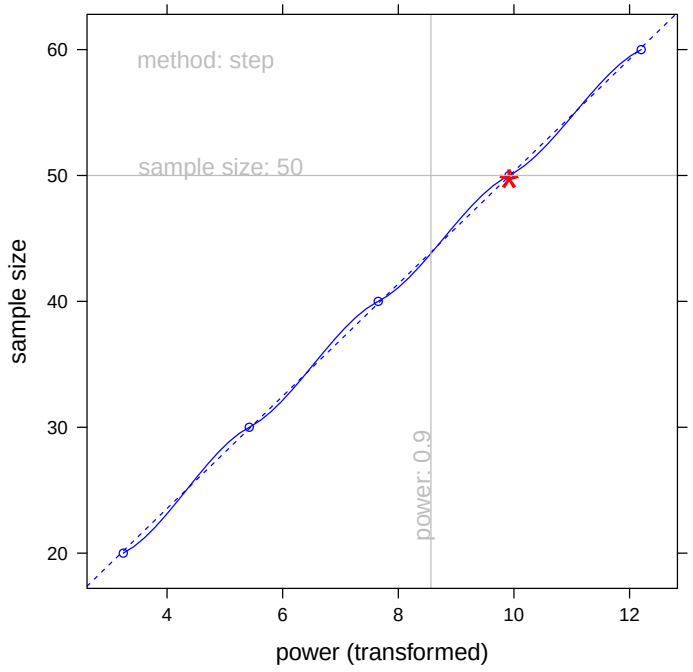

Figure 1: Inspection plot of the minimal example: We see on the x-axis the observed range of power and on the y-axis the range of sample sizes. The dots represent the individual values that were calculated. The red star shows which sample size, 50, was chosen for reporting. The method "step" always takes the first sample size that is larger than the power chosen.

We can see for which sample size the power was calculated. In this case we decided that the sample size was evaluated for 20, 30 ,40 ... but not e.g. for 42. We can now update the object and choose a new range of `n` and only the new elements will be evaluated.

```
> pow <- update(pow, n = seq(from = 20, to = 60, by = 2))
```

Finally we can generate a "power plot" showing the sensitivity of the sample size estimation to the effect size, theta (figure 2).

```
> plot(pow,
+      xlab = "Effect size",
+      ylab = "Total sample size")
```

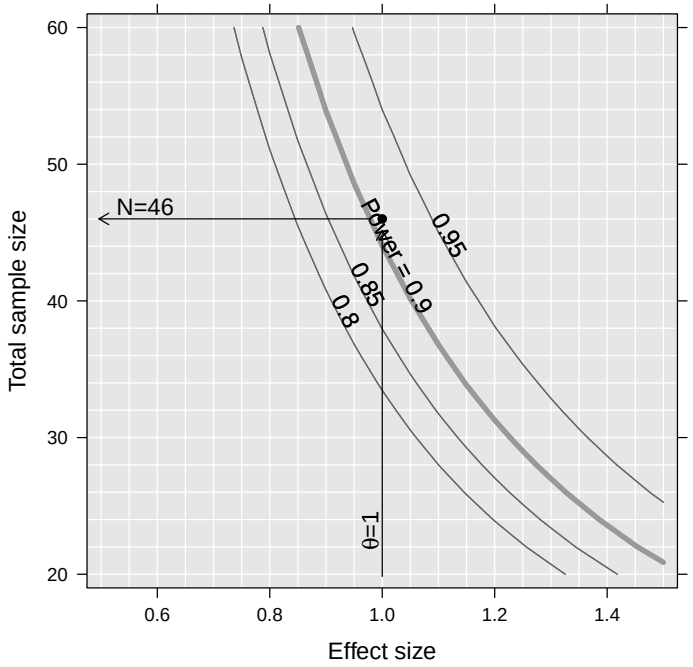

Figure 2: "Power plot" of the minimal example: Showing how sensitive the estimated sample size is with respect to the effect size. The arrows indicate how to read the figure for the given example.

To be able to extract the estimate from the power-object for the integration in Sweave documents there is a method called tex. The call

```
> tex(pow, type = "nEval")
```

```
[1] 46
```

returns the string "46" and can be used directly within the Sweave document e.g. with the LaTeX-command "Sexpr".

2 Using a resampling approach

To illustrate the resampling approach we take the same example as before but we compare the two groups with a Wilcoxon test. Note the difference in the function we define. Whereas the former function powFun returns the power of a combination of theta and n the function powFun.resample defined below returns a logical, indicating if the test was statistically significant or not.

```
> powFun.resample <- function(psi)
+ {
+   x <- rnorm(n(psi)/2)
+   y <- rnorm(n(psi)/2) + theta(psi)
+   return(wilcox.test(x = x, y = y)$p.value < 0.05)
+ }
```

Also the call of the function powCalc has to be changed slightly. Whereas in the minimal example the "power function" was evaluated only once per combination of theta and n it will now be evaluated several times [1] to be able to calculate the power as the proportion of significant evaluations.

```
> calc.resample <- powCalc(psi,
+                          statistic = powFun.resample,
+                          n.iter = 99)
```

The next two steps are the same as in the minimal example.

```
> pow.resample <- powEx(calc.resample, theta = 1, power = 0.9)
```

As in the minimal example we will now be able to inspect the estimation. To choose a sample size based on the evaluations there are two different methods (they can be addressed using the argument "method" of the function powEx). The default method for the resampling approach is called "lm". Therefor a linear regression is fit to the transformed values and used for estimating the sample size where the power is equal to the value asked. To see if the fit of the regression model is reasonable it is important to have a look at the inspection plot (figure 3).

```
> inspect(pow.resample)
```

[1] Note: the number of iterations (n.iter = 99) is to small and only for illustrative purpose!

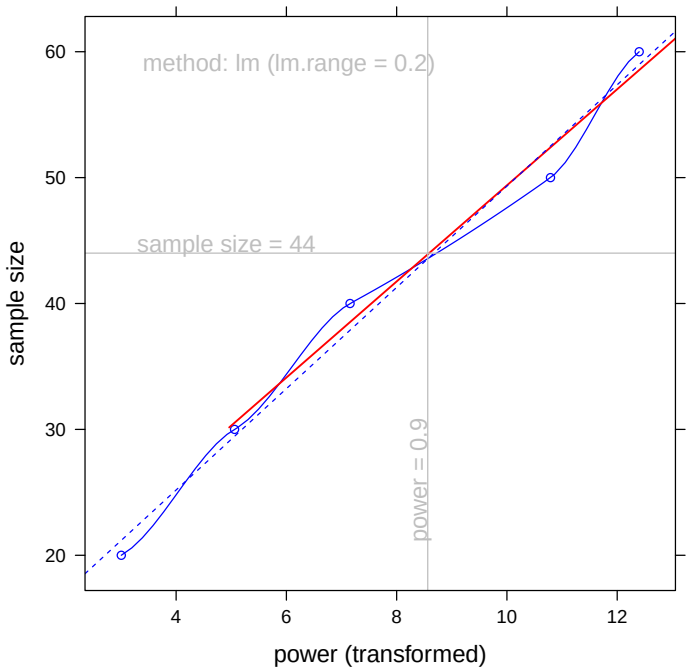

Figure 3: Inspection plot of the resampling example: We see on the x-axis the observed range of power and on the y-axis the range of sample sizes. The dots represent the individual values that were calculated. The red line shows the regression line used for estimation. By default only the data in the neighbourhood of the power in focus is used for estimating the regression line (the size of the neighbourhood can be chosen using the argument lm.range of the function powEx).

To get a more accurate estimation of the sample size for the chosen example it is possible to increase the number of iterations by e.g. a factor of five.

```
> pow.resample <- refine(pow.resample, factor = 5)
```

Not like the function update, the function refine increases the number of iterations only locally and not for the whole parameter space.

The power plot can be produced in exactly the same manner as in the minimal example. Especially useful for the resampling approach is the call

```
> tex(pow.resample, type = "sampling")
```

that returns after evaluation in LaTeXto: "$n_{i=1,...,5} = 20, ..., 60$". It allows to see the step width used for estimation. Also the call

```
> tex(pow.resample, type = "n.iter")
```

returning the number of iterations for each combination of theta and n might be useful.

3 Semiparametric Resampling (Theory)

If data from a pilot study is available, a semiparametric resampling can be used to account

- non-parametrically for available data, and
- parametrically for an effect size, e.g. a shift in location.

An idea by Davison and Hinkley, *Bootstrap Methods and their Application*, Cambridge Series in Statistical and Probabilistic Mathematics, 1997.

Calculating the power for the null hypothesis, $H_0 : \theta = 0$, using the Wilcoxon test, where θ is a shift in location and the underlying distribution F is unspecified.

1. Pool $F(g_1)$, and $F(g_2 - \hat{\theta})$ to $\hat{F}$, where $\hat{\theta}$ is the difference in sample medians between group one and two.
2. Sample n_1 subjects from $\hat{F}$.
3. Sample n_2 subjects from $\hat{F}$ and add θ to each.
4. Calculate a test statistic t^* with R repetitions.
5. Calculate the proportion of successful repetitions.

4 Comparing the Wilcoxon and t Test

The following example shows how sample size for a t Test and the Wilcoxon Test can be estimated based on semiparametric resampling.

```
> ##
> pilot.data <- rnorm(1000)
> ##
> psi <- powPar(F.hat = pilot.data,
+               delta = seq(from = 0.5, to = 1.5, by = 0.05),
+               n = seq(from = 20, to = 50, by = 2),
+               theta.name = "delta")
> ##
> powFun <- function(psi){
+    a <- sample(pp(psi, "F.hat"), size = n(psi)/2, replace = TRUE)
+    b <- sample(pp(psi, "F.hat"), size = n(psi)/2, replace = TRUE) + theta(psi)
+    w <- wilcox.test(a, b)$p.value < 0.05
+    t <- t.test(a, b)$p.value < 0.05
+    return(c(w = w, t = t))
```

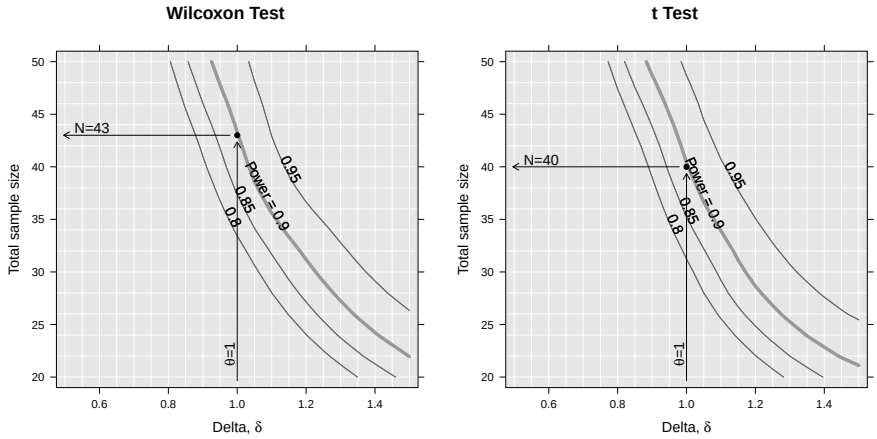

Figure 4: Comparing the sample size needed between Wilcoxon Test and t Test.

```
+    }
> calc <- powCalc(psi, statistic = powFun, n.iter = 99)
> pow.t <- powEx(calc, theta = 1, drop = 0.1, endpoint = "t")
> pow.w <- powEx(calc, theta = 1, drop = 0.1, endpoint = "w")

> plot(pow.w, smooth = 0.5,
+      xlab = expression(paste("Delta, ", delta)),
+      ylab = "Total sample size",
+      main = "Wilcoxon Test")

> plot(pow.t, smooth = 0.5,
+      xlab = expression(paste("Delta, ", delta)),
+      ylab = "Total sample size",
+      main = "t Test")
```

5 Some Ideas for Optimal Usage

- Choose a coarse but wide range for `n` and `theta` to start with. You can make it finer using the `update`-function and you can narrow it in the plot by using `xlim` and `ylim`. This might look like:

```
> ## as used in the vignette(examples)
> psi <- powPar(theta = seq(from = 0.5, to = 1.5, by = 0.5),
+               n = seq(from = 20, to = 200, by = 20))
> ##
> powFun.power <- function(psi)
+ {
```

```
+   return(power.t.test(n = n(psi)/2,
+                       delta = theta(psi),
+                       sig.level = 0.05)$power)
+ }
> ##
> calc.power <- powCalc(psi, statistic = powFun.power)
> ##
> pow.power <- powEx(calc.power, theta = 1, power = 0.9)
> ##
> plot(pow.power)
> ##
> calc.power.fine <- update(calc.power,
+                       n = c(seq(from = 20, to = 60, by = 2),
+                             seq(from = 80, to = 200, by = 20)),
+                       theta = seq(from = 0.5, to = 1.5, by = 0.05))
> ##
> pow.power.fine <- powEx(calc.power.fine, theta = 1, power = 0.9)
> ##
> plot(pow.power.fine,
+      xlim = c(0.79, 1.21),
+      ylim = c(28, 62)
+      )
```

- If you use resampling, increase `n.iter` using the `update`-function once you know how the plot looks like.