Introducing fastpos: A Fast R Implementation to Find the Critical Point of Stability for a Correlation

by Johannes Titz

Abstract The R package **fastpos** provides a fast algorithm to estimate the required sample size for a Pearson correlation to *stabilize* (Schönbrodt and Perugini 2013). The stability approach is an innovative alternative to other means of sample size planning, such as power analysis. Although the approach is young, it has already attracted much interest in the research community. Still, to date, there exists no easy way to use the stability approach because there is no analytical solution and a simulation approach is computationally expensive with a quadratic time complexity. The presented package overcomes this limitation by speeding up the calculation of correlations and achieving linear time complexity. For typical parameters, the theoretical speedup is around a factor of 250, which was empirically confirmed in a comparison with the original implementation `corEvol`. This speedup allows practitioners to use the stability approach to plan for sample size and theoreticians to further explore the method.

1 Sample size planning with the stability approach

Sample size planning is one of the most crucial steps before conducting an empirical study. The approach-avoidance conflict lies in the desire for reliable conclusions, but the unwillingness to spend resources for large samples. To balance benefit and cost there exist three more or less established paths: power analysis (e.g. Cohen 1988), accuracy in parameter estimation [AIPE; e.g. Maxwell, Kelley, and Rausch (2008)] and interval based accuracy methods (Algina and Olejnik 2003). Recently, a fourth way was introduced: stability (Schönbrodt and Perugini 2013). The general idea of this approach is to determine the sample size at which a certain percentage of studies will fall into an priori specified interval and stay in this interval if the sample size is increased further. For instance, if the population correlation is 0.5, one can define the limits to be 0.4 and 0.6. Given these constraints, what sample size is required to guarantee, with a certain probability (e.g. 90%), that the correlation coefficient will not drop below 0.4 or rise above 0.6 if more participants are added. This sample size is also referred to as the *critical point of stability* for the specific parameters. The stability approach is promising because it (1) focuses on the effect size instead of significance and (2) is fairly intuitive. Indeed, the interest in the method is growing, evident in more than 1500 citations of the original publication. But a proper software package for the stability approach is still missing.

When the concept was introduced, the authors presented a collection of R scripts (`corEvol`, available at a github repository: `https://github.com/nicebread/corEvol`) to derive a sample size table for certain parameters. This implementation is too slow to plan the sample size for an individual study as it can take hours to get reliable results. In this article a faster implementation of the stability approach is introduced available in the R package **fastpos** with the function `find_critical_pos`.

2 Model and implementations

The general model can be shortly described as follows: Define a population correlation ρ, the corridor of stability with lower limit l and upper limit u and a confidence $1-\alpha$. Now, pairs of values from a bivariate normal distribution with correlation ρ are drawn. In a first step n_{min} pairs are drawn, to which, repeatedly, one more pair is added so that the sample size n is sequentially increased by 1. For every n the correlation r_n is calculated. The point of stability n_{pos} can be described as:

$$n_{\text{pos}} = \min \{n \in \mathbb{N} | l \leq r_m \leq u, \forall m \geq n\} \tag{1}$$

Meaning that the corridor of stability is not left again after the point of stability has been crossed and that the corridor of stability was just entered at the point of stability. Note that n_{pos} is a random variable that has to be evaluated with respect to the normal bivariate distribution. The *critical* point of stability is the quantile $1-\alpha$ of the probability density function of n_{pos}. It is possible to calculate the transition probabilities of entering, leaving or staying in the corridor of stability for two neighboring sample sizes n and $n+1$. But, so far, no analytical solution to calculate the critical point of stability has been proposed.

Instead, Schönbrodt and Perugini (2013) set up a Monte Carlo simulation to produce a sample size table for some parameter combinations. In a simulation a maximum sample size $n_{\max}$ has to be chosen. Then, for every n from $n_{\min}$ to $n_{\max}$ the correlation can be calculated. The point of stability for one simulation study can again be described by the above condition. From many of such studies, the critical point of stability can be estimated for the desired confidence.

In the original implementation, the correlations were calculated from scratch for each n, using the function `cor` from `stats`. This is slow as several millions of correlations have to be calculated for a reliable estimate. The correlations at n and $n+1$ only differ by one pair of values, which can be exploited for speed. Take the sum formula for the correlation coefficient at a specific sample size n:

$$r_n = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}} \tag{2}$$

Several sums are calculated, each consisting of adding up n terms. In `corEvol` this is done for every sample size from the minimum to the maximum one. Thus, the total number of added terms for one sum is:

$$\sum_{n=n_{\min}}^{n_{\max}} n = \sum_{n=1}^{n_{\max}} n - \sum_{n=1}^{n_{\min}-1} n = \frac{n_{\max}(n_{\max}+1)}{2} - \frac{(n_{\min}-1)(n_{\min}-1+1)}{2} \tag{3}$$

The variable $n_{\min}$ can be ignored as it is usually a small value and could even be set to 2. Furthermore, the number of sums in the correlation formula will be the same for every algorithm and is a constant. Dropping constant factors and lower order terms, the time complexity of the described algorithm is $\mathcal{O}(n_{\max}^2)$.

In contrast, fastpos calculates the correlation for the maximum sample size first. This requires to add $n_{\max}$ numbers for one sum. Then it subtracts one value from this sum to find the correlation for the sample size $n_{\max} - 1$, which happens repeatedly until the minimum sample size is reached (or the corridor is left). In the worst case, the total number of terms for one sum amounts to:

$$n_{\max} + n_{\max} - n_{\min} \tag{4}$$

Again, dropping constant factors and lower order terms, the time complexity of this algorithm is $\mathcal{O}(n_{\max})$. The ratio between the two approaches is:

$$\frac{n_{\max}(n_{\max}+1) - (n_{\min}-1)\,n_{\min}}{4n_{\max} - 2n_{\min}} \tag{5}$$

For the typically used $n_{\max}$ of 1,000 and $n_{\min}$ of 20, a speedup of about 250 can be expected. From a theoretical perspective it is also interesting to study the stability approach with larger values of $n_{\max}$, for which the difference becomes even more pronounced.

The theoretical speedup is only an approximation for several reasons. First, one can stop the algorithm when the corridor is left the first time, which is done in fastpos but not in `corEvol`. Second, the main function of fastpos was written in C++ (via Rcpp, Eddelbuettel et al. 2022), which is much faster than normal R. At the same time, the algorithms contain many more steps than just calculating correlations. For instance, setting up the population with a specific ρ takes some time since it usually consists of a million value pairs. The interface functions to setup the simulations also play a role, especially when the algorithm itself is very fast. Thus, it is necessary to study the speed benefit empirically. But before running a benchmark it will be useful to show (1) how to use fastpos in general and (2) that it produces the same estimates as `corEvol`.

3 How to use fastpos

For a simple illustration, imagine you plan an empirical study and believe the population correlation is 0.6. You would be happy to find a *stable* correlation between 0.5 and 0.7 with a probability of 80%. What this means is that there is an 80% chance of finding a correlation between 0.5 and 0.7 and by adding more participants this corridor is not left again. In fastpos you can run:

```
library(fastpos)
set.seed(20200219)
find_critical_pos(rho = 0.6, precision_absolute = 0.1, confidence_levels = .8,
                  sample_size_min = 20, sample_size_max = 1e3, n_studies = 1e4)
```

```
#>   rho_pop pos.80% sample_size_min sample_size_max lower_limit upper_limit
#> 1     0.6     104              20            1000         0.5         0.7
#>   n_studies n_not_breached precision_absolute precision_relative
#> 1     10000              0                0.1                 NA
```

This loads the package, sets a seed for reproducibility, and runs the simulation with default parameters (except for the ones specifically set). A progress bar is displayed if run in interactive mode. The result is a critical point of stability of 104.

The main function of the package `find_critical_pos` will usually suffice for most use cases. Its parameters are documented in detail in the package. The population correlation (`rho`) and the number of simulation studies (`n_studies`) is self-explanatory. The chosen precision (`precision_absolute`) of 0.1 (i.e. the half-width) will result in the desired corridor between 0.5 and 0.7. There is also a convenience argument to set the precision as a relative value, `precision_relative`, which will override `precision_absolute`. For instance, `precision_relative = 0.1` produces an interval of $\rho \pm \rho \cdot 0.1$. Alternatively, one can also provide the lower and upper limit of the corridor directly via `lower_limit` and `upper_limit`. This is especially useful if the corridor is not symmetric. Notable, most parameters can also take vectors so it is possible to run multiple simulations for different `rho` values (and corresponding other parameters) at once.

The parameter `confidence_levels` defines the quantile corresponding to the critical point of stability. This parameter can be a single value or a vector, but is fixed for all `rho` values. If different confidence levels are of interest, providing them as a vector saves a lot of resources because one simulation can be used to calculate the critical points of stability for all confidence levels.

The parameters `sample_size_min` and `sample_size_max` set the minimum and maximum sample size of one simulation study. As in `corEvol` they default to 20 and 1,000. This means a sample of 20 observation pairs is drawn from the population and step by step one more observation is added until the sample size of 1,000 is reached.

The output summarizes the individually set (and default) parameters as well as the critical point of stability of about 104. The value will change slightly from run to run because only 10,000 simulations are done here. In practice one can make a quick estimate with the default parameters and then increase the number of simulation studies for a more robust estimate. Under GNU/Linux one can also take advantage of the multicore support (parameter `n_cores`). This functionality is currently implemented via the **pbmcapply** package (Kuang, Kong, and Napolitano 2022), which is based on `parallel`.[1]

For another illustration let us reproduce Schönbrodt and Perugini (2013)'s oft-cited table of the critical points of stability for an absolute precision of 0.1 (meaning that the corridor will be $\rho \pm .1$). We take advantage of the vectorized input option by providing several ρ values at once. Furthermore, we increase the number of studies to 100,000 to get accurate estimates. To cache the simulation results we use **simpleCache** (Nagraj and Sheffield 2021):

```
library(simpleCache)
setCacheDir("titz_cache")
simpleCache("sim2", {find_critical_pos(rho = seq(.1, .7, .1), n_studies = 1e5)})
sim2
```

```
#>   rho_pop pos.80% pos.90% pos.95% sample_size_min sample_size_max lower_limit
#> 1     0.1     253     363     478              20            1000         0.0
#> 2     0.2     237     339     448              20            1000         0.1
#> 3     0.3     212     305     404              20            1000         0.2
#> 4     0.4     181     262     343              20            1000         0.3
#> 5     0.5     143     208     277              20            1000         0.4
#> 6     0.6     103     150     200              20            1000         0.5
#> 7     0.7      65      96     129              20            1000         0.6
#>   upper_limit n_studies n_not_breached precision_absolute precision_relative
#> 1         0.2     1e+05            139                0.1                 NA
#> 2         0.3     1e+05            102                0.1                 NA
#> 3         0.4     1e+05             43                0.1                 NA
#> 4         0.5     1e+05             15                0.1                 NA
#> 5         0.6     1e+05              5                0.1                 NA
#> 6         0.7     1e+05              0                0.1                 NA
#> 7         0.8     1e+05              0                0.1                 NA
```

The results are very close to the original publication (Schönbrodt and Perugini 2013). Note that a warning is shown because in some simulations the point of stability was not found. This is not too

[1]The multicore support will not be demonstrated here because it is difficult to create reproducible examples across different operating systems and number of cores.

surprising as one can easily imagine an extreme outlier study that, for instance, starts at a negative correlation with $n = 20$ and does not reach the specified corridor of stability at the maximum sample size of $n = 1,000$. There are different ways to handle these outliers, which will affect the estimate.

4 Handling outliers

When comparing the table from above with the one in Schönbrodt and Perugini (2013), one should notice that **fastpos** usually produces larger estimates. To illustrate this more reliably we need to increase the number of studies, so that random fluctuations are minimized. Here we will run 100 simulations with 1,000,000 studies each.[2]

```
simpleCache("sim3", {find_critical_pos(rho = rep(0.1, 100),
                                       sample_size_max = 1e3, n_studies = 1e6)})
```

A good summary of the data is the mean and the standard error of the distribution. Before calculating these statistics, we select only the points of stability from the result:

```
sim3 <- sim3[, c("pos.80%", "pos.90%", "pos.95%")]
colMeans(sim3)

#>  pos.80%  pos.90%  pos.95%
#> 253.2020 363.2100 477.5905

round(apply(sim3, 2, sd), 3)

#> pos.80% pos.90% pos.95%
#>   0.603   0.729   1.035
```

The average estimates are 253, 363 and 478 (with reasonably small standard errors), while in Schönbrodt and Perugini (2013) they are 252, 362, 470 and in Schönbrodt and Perugini (2018) 252, 360 and 474. Note that in every case **fastpos** gives a slightly larger estimate, which is not just a random fluctuation but related to the warning. In `corEvol`, if the corridor of stability is not reached, the respective study is ignored when calculating the critical point of stability. This leads to a systematic underestimation of the critical point of stability.

To illustrate this, we can use the lower level functions `create_pop` and `simulate_pos` to create a distribution of points of stability. In the following, the first line creates a population with a specific correlation and the second line produces several points of stability by drawing from this population. In contrast to the main function of the package (`find_critical_pos`), the function `simulate_pos` does not calculate quantiles, but only generates points of stability.

```
pop <- create_pop(rho = 0.1, size = 1e6)
simpleCache("sim4", {simulate_pos(x_pop = pop[, 1], y_pop = pop[, 2],
                                  n_studies = 1e6, sample_size_min = 20,
                                  sample_size_max = 1e3, replace = TRUE,
                                  lower_limit = 0, upper_limit = 0.2,
                                  progress = FALSE)})
```

There are two ways to calculate the quantiles of interest:

```
quantile(sim4, c(.8, .9, .95), na.rm = TRUE)

#> 80% 90% 95%
#> 252 361 473

sim4b <- ifelse(is.na(sim4), 1e3, sim4)
quantile(sim4b, c(.8, .9, .95))

#> 80% 90% 95%
#> 253 363 478
```

[2]It is worth noting that (with a single core) this simulation would take several weeks to complete with `corEvol` but only takes about 66 minutes with **fastpos**.

In the first calculation, the studies that did not reach the corridor of stability are ignored (like in corEvol), while in the second calculation it is assumed that the point of stability was reached at the maximum sample size. When repeating this simulation, the values will vary slightly but the second method will never produce smaller estimates. That the second method is more accurate can be tested by increasing the maximum sample size (to avoid studies that do not reach the corridor of stability). Here, we will set the maximum sample size to 5,000:

```
simpleCache("sim5", {find_critical_pos(rho = rep(0.1, 100),
                                       sample_size_max = 5e3,
                                       n_studies = 1e6)})
sim5 <- sim5[, c("pos.80%", "pos.90%", "pos.95%")]
colMeans(sim5)

#>  pos.80%  pos.90%  pos.95%
#> 253.3100 363.5100 478.4305

round(apply(sim5, 2, sd), 3)

#> pos.80% pos.90% pos.95%
#>   0.631   0.870   1.266
```

If every study reaches the point of stability, the estimates are 253, 364 and 478. When the maximum sample size is too small (as in the second to last simulation), **fastpos** is indeed closer to these estimates than corEvol. While the difference to corEvol might seem practically negligible, corEvol's estimates are clearly biased. Furthermore, depending on the parameters, the problem can become more severe. A very narrow corridor will lead to many studies not reaching the corridor, which corEvol will not even notice. On the other hand, **fastpos** will throw a warning, which should be taken seriously.

But even **fastpos** might underestimate the critical point of stability if the maximum sample size is too small: All estimates with a maximum sample size of 5,000 are slightly larger than the ones with a maximum sample size of 1,000. With a larger maximum sample size, there are more opportunities to leave the corridor again. At some point the probability of this event is very low because the corridor limits are too far away, but the probability is not 0. Thus, increasing the maximum sample size even further (here to 10,000) should lead to slightly larger estimates:

```
simpleCache("sim6", {find_critical_pos(rho = rep(0.1, 100),
                                       sample_size_max = 1e4,
                                       n_studies = 1e6)})
sim6 <- sim6[, c("pos.80%", "pos.90%", "pos.95%")]
colMeans(sim6)

#>  pos.80%  pos.90%  pos.95%
#> 253.4000 363.7000 478.7005

round(apply(sim6, 2, sd), 3)

#> pos.80% pos.90% pos.95%
#>   0.667   0.937   1.234
```

Indeed, all estimates are slightly larger but after rounding to a whole number only for the confidence of 95% the critical point of stability changes from 478 to 479. Furthermore, the randomness of the simulations permits such fluctuations since the standard errors are about 1. But note that all estimates increase when the maximum sample size changes from 1,000 to 5,000 and then to 10,000, which is a clear hint for a bias. Nonetheless, it appears unlikely that the estimates would increase much, when the maximum sample size grows further. The remaining problem is that the theoretical idea of stability assumes an *infinite* maximum sample size or, at least, that the maximum sample size is equal to the population size. It is therefore of some technical and practical interest to investigate the relationship between the maximum sample size and the critical point of stability in a dedicated simulation study with **fastpos**. Such a study would not be easy to approach with corEvol because of the quadratic time complexity. In the next section the speed difference between both packages is demonstrated empirically in a benchmark.

5 Benchmark

corEvol was written as a script for a simulation study and cannot be simply called via a function in a package. Thus, a helper function will be used that sources the script files. To make the benchmark reproducible, the original repository corEvol was forked and a benchmark branch created. With git and a shell installed, the following tries to update the repository in the corEvol folder. If this is unsuccessful (the folder does not exist), the repository is cloned.

```
git -C corEvol pull || git clone --single-branch --branch benchmark \
  https://github.com/johannes-titz/corEvol
```

Alternatively, you can download the required files from the supplementary material of this article.

For corEvol, two files are sourced for the benchmark. The first file generates the simulations and the second calculates the critical point of stability. In corEvol a simulation run takes a lot of time and thus it is not practical to run it too many times. But since the expected speed difference between both implementations is substantial, this should not be a concern. Here, ten repetitions were done with the **microbenchmark** (Mersmann 2021) package. The code was run on a Dell Server r6515 with an AMD EPYC 7302P CPU. Only one core was used to not confound the result with the specific parallel implementation.

```
library(microbenchmark)
corevol <- function() {
  setwd("corEvol")
  source("01-simdata.R")
  source("02-analyse.R")
  setwd("../")
}
fastpos <- function() {
  find_critical_pos(rho = .1, sample_size_max = 1e3, n_studies = 1e4,
                    progress = FALSE)
}
simpleCache("bm", {microbenchmark(corevol = corevol(), fastpos = fastpos(),
                                  times = 10, unit = "s")})
summary(bm)
```

```
#>      expr         min         lq        mean     median          uq         max
#> 1 corevol 350.4551133 352.642384 355.7306834 355.221224 358.2854179 365.8751542
#> 2 fastpos   0.5692708   0.579922   0.6215005   0.596842   0.6066209   0.8496276
#>   neval cld
#> 1    10   b
#> 2    10  a
```

For the chosen parameters, **fastpos** is about 572 times faster than corEvol, for which there are two main reasons: (1) **fastpos** is built around a C++ function via **Rcpp** and (2) this function does not calculate every calculation from scratch, but only calculates the difference between the correlation at sample size n and $n-1$ via the sum formula of the Pearson correlation (see Equation (2)). There are some other factors that might play a role, but they cannot account for the large difference found. For instance, setting up a population takes quite long in corEvol (about 17s), but compared to the 6 minutes required overall, this is only a small fraction. There are other parts of the corEvol code that are fated to be slow, but again, a speedup by a factor of 572 cannot be achieved by improving these parts. The presented benchmark is not comprehensive, but still demonstrates that **fastpos** can be used with no significant waiting time for a typical scenario, while for corEvol this is not the case.

Another benchmark on a local i5-3320 2.6 GHz CPU from 2012 resulted in means of 1.5s for **fastpos** and 603s for corEvol giving a speedup of around 400. Thus, even on older CPUs and single-cored **fastpos** delivers almost instantly for default parameters.

6 Other effect sizes

The focus of **fastpos** is on the Pearson correlation as the effect size. In principle the stability approach can be extended to all sorts of effect sizes or even other statistical parameters. Since the original authors studied the Pearson correlation, it made sense to improve the algorithm for this specific use case. But mathematical shortcuts as in Equation (2) should also exist for other effect sizes and might be implemented in the future.

A simple alternative for applying the method to other effect sizes is to convert these effects to the Pearson correlation. Such conversions are very common in meta-analyses, where a consistent effect size must be used across all studies to calculate a meaningful average effect. Standard approximate conversion formulas can be found in text books on research methods (Borenstein et al. 2021; Sedlmeier and Renkewitz 2018). Several packages in R also provide these conversions. For instance, **effectsize** (Ben-Shachar, Lüdecke, and Makowski 2020) includes the functions d_to_r and r_to_d. d_to_r is based on the approximation $r = \frac{d}{d^2+4}$, which should only be used for equal group sizes. As an example, consider $d = 0.5$ between two equally sized groups and a corridor with limits of 0.4 and 0.6.

```
r <- effectsize::d_to_r(0.5)
lower_limit <- effectsize::d_to_r(0.4)
upper_limit <- effectsize::d_to_r(0.6)
simpleCache("sim7", {find_critical_pos(rho = r, sample_size_max = 11e3,
                                       n_studies = 1e5,
                                       lower_limit = lower_limit,
                                       upper_limit = upper_limit)})
sim7
```

```
#>     rho_pop pos.80% pos.90% pos.95% sample_size_min sample_size_max lower_limit
#> 1 0.2425356    1119    1606    2108              20           11000   0.1961161
#>   upper_limit n_studies n_not_breached precision_absolute precision_relative
#> 1   0.2873479     1e+05              0                 NA                 NA
```

The corresponding Pearson correlation for $d = 0.5 \pm 0.1$ is about 0.24, with very narrow and slightly asymmetric limits (0.20 to 0.29). The critical point of stability is 2108 for a confidence level of 95%.

7 Summary

In this article, **fastpos**, a package for estimating the critical point of stability was introduced. The package is much faster than the original implementation and can be conveniently used for sample size planning as well as Monte Carlo simulation studies. While the original implementation ignores studies that do not reach the corridor of stability, **fastpos** takes them into account and gives a more conservative and more accurate estimate (i.e. a larger critical point of stability). From a practitioner's perspective, this detail might be negligible for typical parameters and relatively wide corridors. But from a statistical perspective, this detail is of relevance and further simulation studies are required to better understand the stability approach in general. Finally, a comparison to other methods of sample size planning would be of much interest and could influence how empirical scientists plan for sample size in the future. **fastpos** can be a useful tool to achieve these goals.

8 Acknowledgment

I want to thank Matthias Hörr and Thomas Schäfer for insightful discussions about the stability approach. Furthermore, I want to thank an anonymous reviewer for many helpful suggestions on how to improve the article and the package.

References

Algina, James, and Stephen Olejnik. 2003. "Sample Size Tables for Correlation Analysis with Applications in Partial Correlation and Multiple Regression Analysis." *Multivariate Behavioral Research* 38: 309–23. https://doi.org/10.1207/S15327906MBR3803_02.

Ben-Shachar, Mattan S., Daniel Lüdecke, and Dominique Makowski. 2020. "effectsize: Estimation of Effect Size Indices and Standardized Parameters." *Journal of Open Source Software* 5 (56): 2815. https://doi.org/10.21105/joss.02815.

Borenstein, Michael, Larry V Hedges, Julian PT Higgins, and Hannah R Rothstein. 2021. *Introduction to Meta-Analysis*. John Wiley & Sons.

Cohen, J. 1988. *Statistical Power Analysis for the Behavioral Sciences*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Eddelbuettel, Dirk, Romain Francois, JJ Allaire, Kevin Ushey, Qiang Kou, Nathan Russell, Inaki Ucar, Douglas Bates, and John Chambers. 2022. *Rcpp: Seamless r and c++ Integration*. https://CRAN.R-project.org/package=Rcpp.

Kuang, Kevin, Quyu Kong, and Francesco Napolitano. 2022. *pbmcapply: Tracking the Progress of Mc*pply with Progress Bar*. https://CRAN.R-project.org/package=pbmcapply.

Maxwell, S. E., K. Kelley, and J. R. Rausch. 2008. "Sample Size Planning for Statistical Power and Accuracy in Parameter Estimation." *Annual Review of Psychology* 59: 537–63. https://doi.org/10.1146/annurev.psych.59.103006.093735.

Mersmann, Olaf. 2021. *microbenchmark: Accurate Timing Functions*. https://CRAN.R-project.org/package=microbenchmark.

Nagraj, VP, and Nathan Sheffield. 2021. *simpleCache: Simply Caching r Objects*. https://CRAN.R-project.org/package=simpleCache.

Schönbrodt, F. D., and M. Perugini. 2013. "At What Sample Size Do Correlations Stabilize?" *Journal of Research in Personality* 47: 609–12. https://doi.org/10.1016/j.jrp.2013.05.009.

———. 2018. "Corrigendum to 'At What Sample Size Do Correlations Stabilize?' [J. Res. Pers. 47 (2013) 609–612]." *Journal of Research in Personality* 74: 194. https://doi.org/10.1016/j.jrp.2018.02.010.

Sedlmeier, Peter, and Frank Renkewitz. 2018. *Forschungsmethoden und Statistik für Psychologen und Sozialwissenschaftler*. 3rd ed. Hallbergmoos, Germany: Pearson Studium.

Johannes Titz
Chemnitz University of Technology
Department of Psychology
Chemnitz, Germany
https://johannestitz.com
ORCiD: 0000-0002-1102-5719
johannes.titz@psychologie.tu-chemnitz.de