Issues for the Calculation of Bayes Factors

John Maindonald

5-7 minutes

Choice of prior and of analysis method.

To keep the discussion simple, I propose to limit attention to the one-sample comparison, using as an example the `sleep` dataset from the `datasets` package. When the needed account is taken of pairing, this better illustrates issues that arise for comparison between different priors and parameter settings than when the analysis (wrongly) treats this as a two-sample problem.

Normal prior versus Cauchy prior

What has guided the choice of a normal prior as the default, mostly, in the vignettes? Why prefer it to use of a Cauchy prior? As implemented in the `BayesFactor` package (Morey and Rouder 2018), this uses a numerical approximation that avoids the convergence issues that arise for Markov Chain Monte Carlo. I cannot see why one would want to use Cauchy as implemented in `bayestestR` (Makowski, Ben-Shachar, and Lüdecke 2019). I would use `bayestestR` with a Normal prior only if provided with a good reason to prefer, in a specific analysis, that to Cauchy. What reasons might be adduced? The choice of prior, and issues with convergence, appear to me to trump differences that arise from different approaches to scaling. Accordingly, I regard these as secondary.

Use of `bayestestR` with a Cauchy prior

```
library(rstanarm)
library(bayestestR)
library(logspline)
sleep1 <- data.frame(ID=levels(sleep$ID),
                     diffs=with(sleep, extra[group==1]-
extra[group==2]))
modelc <- stan_glm(formula = diffs~1, data = sleep1,
prior=cauchy(),
                   chains = 10, iter = 5000, warmup = 1000,
verbose=FALSE,
                   refresh=FALSE)
  ## NB: The argument `refresh=FALSE` deserves to be better
advertised
bf0 <- bayestestR::bf_parameters(modelc, null = 0)
  ## returned 18.9; 17.4; no convergence (see below); 20.7
print(c('This run:'=exp(as.data.frame(bf0)[['log_BF']])),
quote=F)
```

```
## This run:
##  31.69117
```

```
bf30 <- bayestestR::bf_parameters(modelc, null =
c(-0.5,0.5))
print(c('This run:'=exp(as.data.frame(bf30)[['log_BF']])),
quote=F)
```

```
## This run:
##   15.2739
```

```
  ## returned run 1: 8.9, run 2: 9.9; no convergence (see
below)
  ## Sampling priors, please wait...
  ## Error in oldlogspline(x) : * no convergence
## Setting of autoscale appears not to make much difference
```

An issue is that convergence failures are relatively common.
It then becomes a matter of trying again. When there is convergence, the Bayes Factors that are returned appear fairly consistent. As will be seen, the same level of consistency appears harder to achieve when a Normal prior is specified.

Use of `bayestestR` with a Normal prior

```
modeln <- stan_glm(formula = diffs~1, data = sleep1,
verbose=FALSE,
   prior = normal(), chains = 10, iter = 5000, warmup =
1000,
   refresh=FALSE)
bfn0 <- bayestestR::bf_parameters(modeln, null = 0)
print(c('This run:'=exp(as.data.frame(bfn0)[['log_BF']])),
quote=F)
```

```
## This run:
##  24.48475
```

```
  ## (autoscale=FALSE) 30.0; 29.8; 28.3; 24.3; 20.2; 24.6
  ## (autoscale=TRUE) 35.5; 20.1; 27.4; 30.2; 20.5; 25.4
```

```
bfn30 <- bayestestR::bf_parameters(modeln, null =
c(-0.5,0.5))
print(c('This run:'=exp(as.data.frame(bfn30)[['log_BF']])),
quote=F)
```

```
## This run:
##  13.73161
```

```
  ## returned 17.4; 17.5 17.6, 16.0.
```

Note that the Bayes Factor corresponding to a point NULL was in each case greater than 28.

The upper limit given in Sellke, Bayarri, and Berger (2001) for a p-value that equals 0.00283 is:

$$-1/(exp(1) * 0.00283 * log(0.00283)) = 22.15$$

Does the Sellke formula apply, for the normal prior as implemented in `bayestestR`? If so, why does the calculated Bayes Factor that is returned usually greater than the Sellke upper limit? If `chains` and/or `iter` and/or `warmup` need to be increased, is the best recourse just to, e.g., double them all.

Comparison with `BayesFactor` (Cauchy prior)

There is no variation of consequence from one run to another.

```
library(BayesFactor)
## Start by checking different made by rscale setting, first
for NULL
rnames <- c('medium','wide','ultrawide')
bfval <- numeric(3)
for (i in 1:3)
 bfval[i] <- with(sleep1, ttestBF(sleep1$diffs,
nullInterval=NULL,
                                  rscale=rnames[i]))
setNames(round(bfval,2), rnames)
```

```
##    medium      wide ultrawide
##     17.26     18.42     18.02
```

```
## For use of a NULL interval, note that a standardized
dfference of
## Effect size that corresponds to a 30-minute difference
eff <- mean(sleep1$diffs)*30/60/sqrt(var(sleep1$diffs))
bfval <- matrix(nrow=2, ncol=3)
for (j in 1:3){
 bfall <- with(sleep1,
      ttestBF(sleep1$diffs, nullInterval=c(-eff,eff),
rscale=rnames[j]))
 bfval[, j] <- as.data.frame(bfall)[['bf']]
 }
setNames(apply(bfval, 2, function(x)round(x[2]/x[1],2)),
rnames)
```

```
##    medium      wide ultrawide
##      5.02      4.34      3.65
```

Packages noted in the Bayesian task view

There is a wide choice. Which of them have functions that calculate Bayes Factors?

References

Makowski, Dominique, Mattan S. Ben-Shachar, and Daniel Lüdecke. 2019. "bayestestR: Describing Effects and Their Uncertainty, Existence and Significance Within the Bayesian Framework." *Journal of Open Source Software* 4 (40): 1541. https://doi.org/10.21105/joss.01541.

Morey, Richard D., and Jeffrey N. Rouder. 2018. *BayesFactor: Computation of Bayes Factors for Common Designs*. https://CRAN.R-project.org/package=BayesFactor.

Sellke, Thomas, MJ Bayarri, and James O Berger. 2001. "Calibration of ρ Values for Testing Precise Null Hypotheses." *The American Statistician* 55 (1): 62–71.