Snijders Example for Bootstrap

Yichi Zhang, Winnie Wing-Yee Tse

2022-10-11

- add ICC, R^2
- wild bootstrap

```
library(tidyverse)
library(lme4)
library(bootmlm)
library(broom.mixed)
library(CR2)
library(lmeresampler)
# library()
```

```
temp <- tempfile()
download.file("https://www.stats.ox.ac.uk/~snijders/mlbook2_r_dat.zip", temp)
sj_dat <- read.table(unz(temp, "mlbook2_r.dat"), header = TRUE)
```

Assumption checking

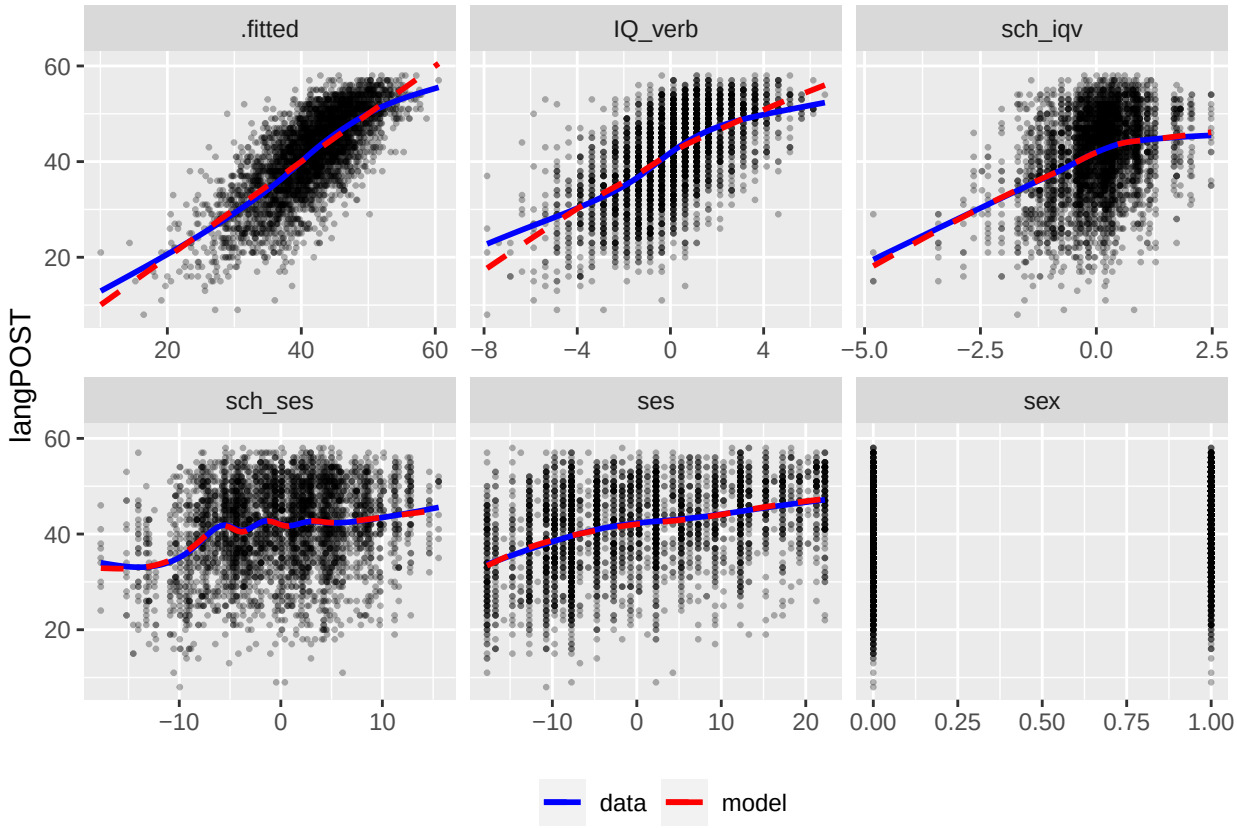
```
mmps_lmer <- function(object) {
  plot_df <- object@frame
  form <- formula(object)
  xvar <- attr(attr(plot_df, "terms"), "varnames.fixed")[-1]
  plot_df$.fitted_x <- fitted(object)
  plot_df$.fitted <- plot_df$.fitted_x
  plot_df$.rowid <- seq_len(nrow(plot_df))
  plot_df_long <- reshape(plot_df, direction = "long",
                          varying = c(xvar, ".fitted_x"),
                          v.names = "xvar",
                          idvar = ".rowid")
  plot_df_long$varname <- rep(c(xvar, ".fitted"),
                              each = nrow(plot_df))
  ggplot(
    data = plot_df_long,
    aes_string(x = "xvar", y = paste(form[[2]]))
  ) +
    geom_point(size = 0.5, alpha = 0.3) +
    geom_smooth(aes(col = "data"), se = FALSE) +
    geom_smooth(aes(y = .fitted, col = "model"),
      linetype = "dashed", se = FALSE
    ) +
```

```
    facet_wrap(~ varname, scales = "free_x") +
    labs(color = NULL, x = NULL) +
    scale_color_manual(values = c("data" = "blue",
                                  "model" = "red")) +
    theme(legend.position = "bottom")
}
```

Normality

```
m0 <- lmer(langPOST ~ IQ_verb*ses + sex + sch_iqv*sch_ses + (1 | schoolnr),
           data = sj_dat)
mmps_lmer(m0)
```

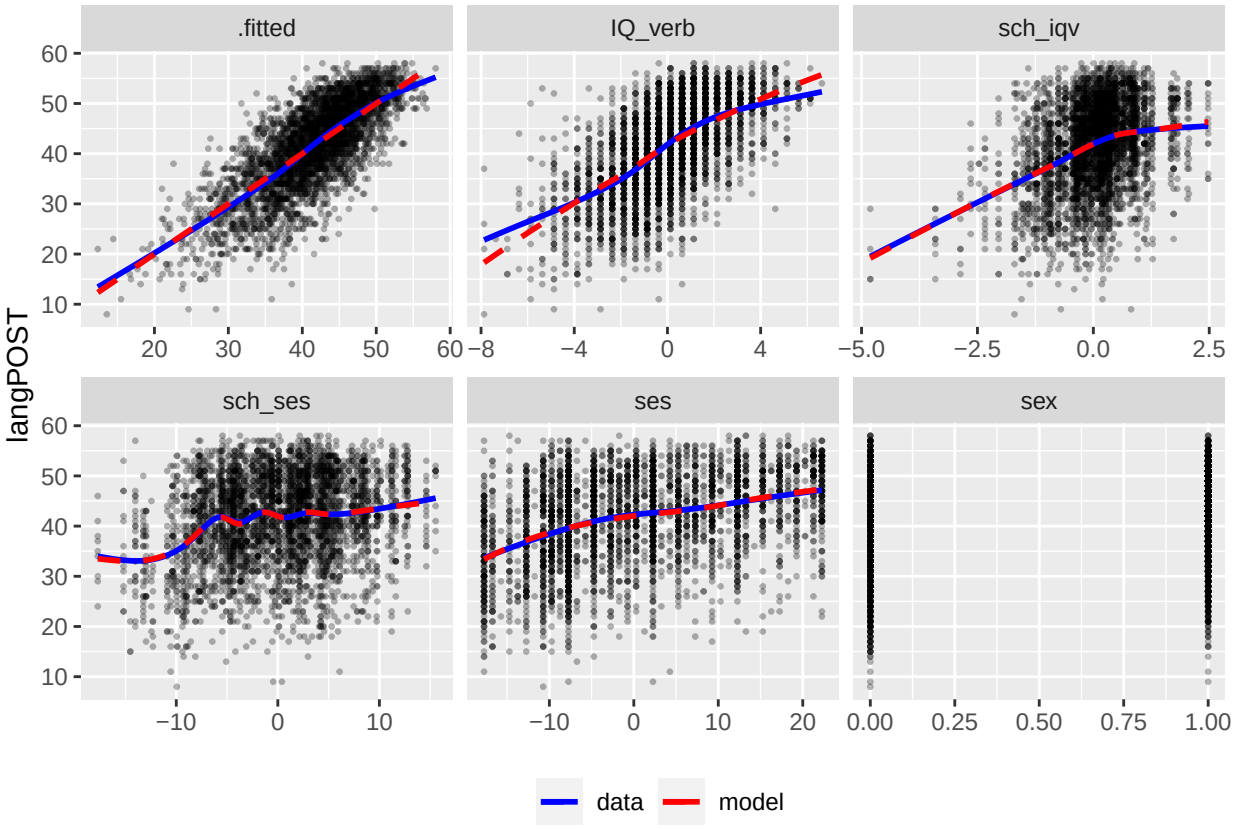
```
## Warning: Computation failed in `stat_smooth()`:
## x has insufficient unique values to support 10 knots: reduce k.
## Computation failed in `stat_smooth()`:
## x has insufficient unique values to support 10 knots: reduce k.
```




```
m1 <- lmer(langPOST ~ IQ_verb*ses + sex + sch_iqv*sch_ses + (IQ_verb | schoolnr),
           data = sj_dat)
mmps_lmer(m1)
```

```
## Warning: Computation failed in `stat_smooth()`:
```

```
## x has insufficient unique values to support 10 knots: reduce k.
## Computation failed in `stat_smooth()`:
## x has insufficient unique values to support 10 knots: reduce k.
```



Heteroscedasticity

- add Mark's graphs

```
ncvMLM(lmer(langPOST ~ IQ_verb + (1 | schoolnr), data = sj_dat))
```

```
## [1] 2.563923e-13
```

```
ncvMLM(lmer(langPOST ~ ses + (1 | schoolnr), data = sj_dat))
```

```
## [1] 6.491533e-09
```

```
ncvMLM(lmer(langPOST ~ sex + (1 | schoolnr), data = sj_dat))
```

```
## [1] 0.007255365
```

```
ncvMLM(lmer(langPOST ~ sch_iqv + (1 | schoolnr), data = sj_dat))
```

```
## [1] 8.605473e-05
```

```
ncvMLM(lmer(langPOST ~ sch_ses + (1 | schoolnr), data = sj_dat))
```

```
## [1] 0.03955194
```

```
ncvMLM(lmer(langPOST ~ IQ_verb * ses + (1 | schoolnr),
            data = sj_dat))
```

```
## [1] 1.937051e-16
```

```
ncvMLM(lmer(langPOST ~ sch_iqv * sch_ses + (1 | schoolnr),
            data = sj_dat))
```

```
## [1] 0.001259076
```

```
ncvMLM(lmer(langPOST ~ IQ_verb * ses + sex + sch_iqv * sch_ses + (1 | schoolnr),
            data = sj_dat))
```

```
## [1] 7.320264e-16
```

Bootstrapping for Fixed Effect

```
fix_eff <- function(x) x@beta[2]
fix_eff(m1)
```

```
## [1] 2.24959
```

Parametric Bootstrap

We can run parametric bootstrap, which essentially call the `lme4::bootMer()` function. It's usually recommended to have a large number of bootstrap samples (R), especially for CIs with higher confidence levels. For illustrative purpose I will use $R = 999$, but in general 1,999 or more is recommended

```
system.time(
  boo_par <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "parametric")
)
```

```
##    user  system elapsed
##  40.698   0.506  41.478
```

```
boo_par
```

```
##
## PARAMETRIC BOOTSTRAP
##
##
```

```
## Call:
## lme4::bootMer(x = x, FUN = FUN, nsim = nsim, seed = seed, use.u = FALSE,
##     type = "parametric", verbose = FALSE)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1*  2.24959 0.001656451  0.06126456
```

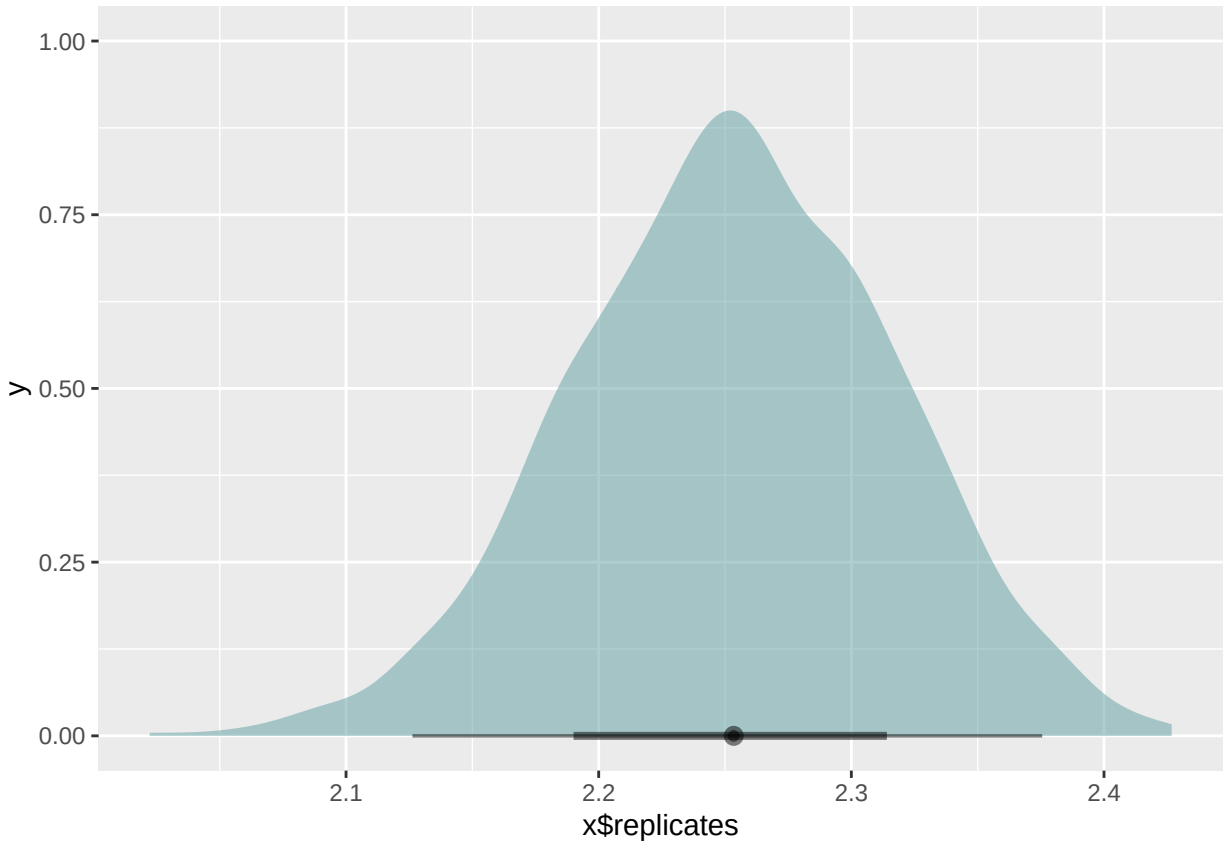
```
## using lmeresampler package
system.time(
  lmer_par <- parametric_bootstrap(m1, .f = fix_eff, B = 999)
)
```

```
##    user  system elapsed
##  40.904   0.511  41.779
```

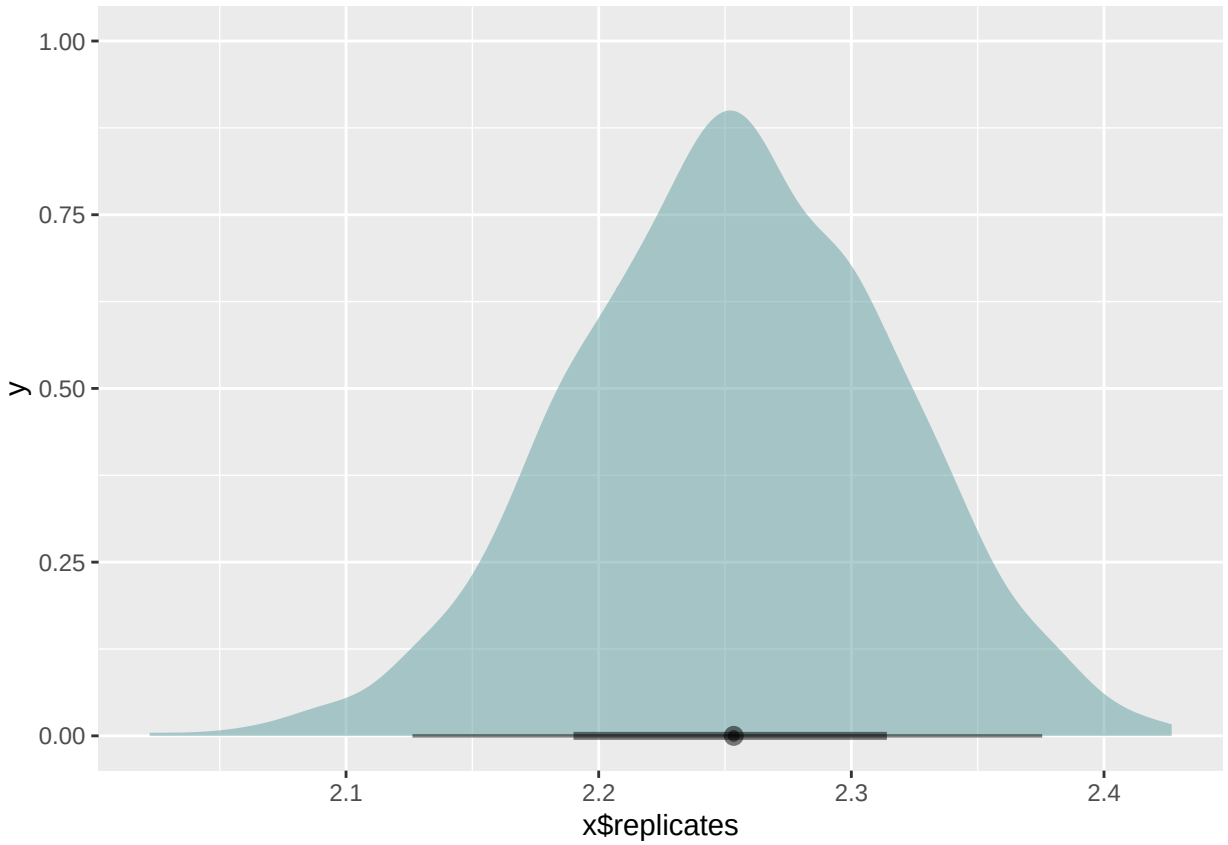
```
summary(lmer_par)
```

```
## Bootstrap type: parametric
##
## Number of resamples: 999
##
##   observed rep.mean        se       bias
## 1  2.24959 2.252917 0.0630829 0.00332728
##
## There were 206 messages, 0 warnings, and 0 errors.
##
## The most commonly occurring message was: boundary (singular) fit: see help('isSingular')
```

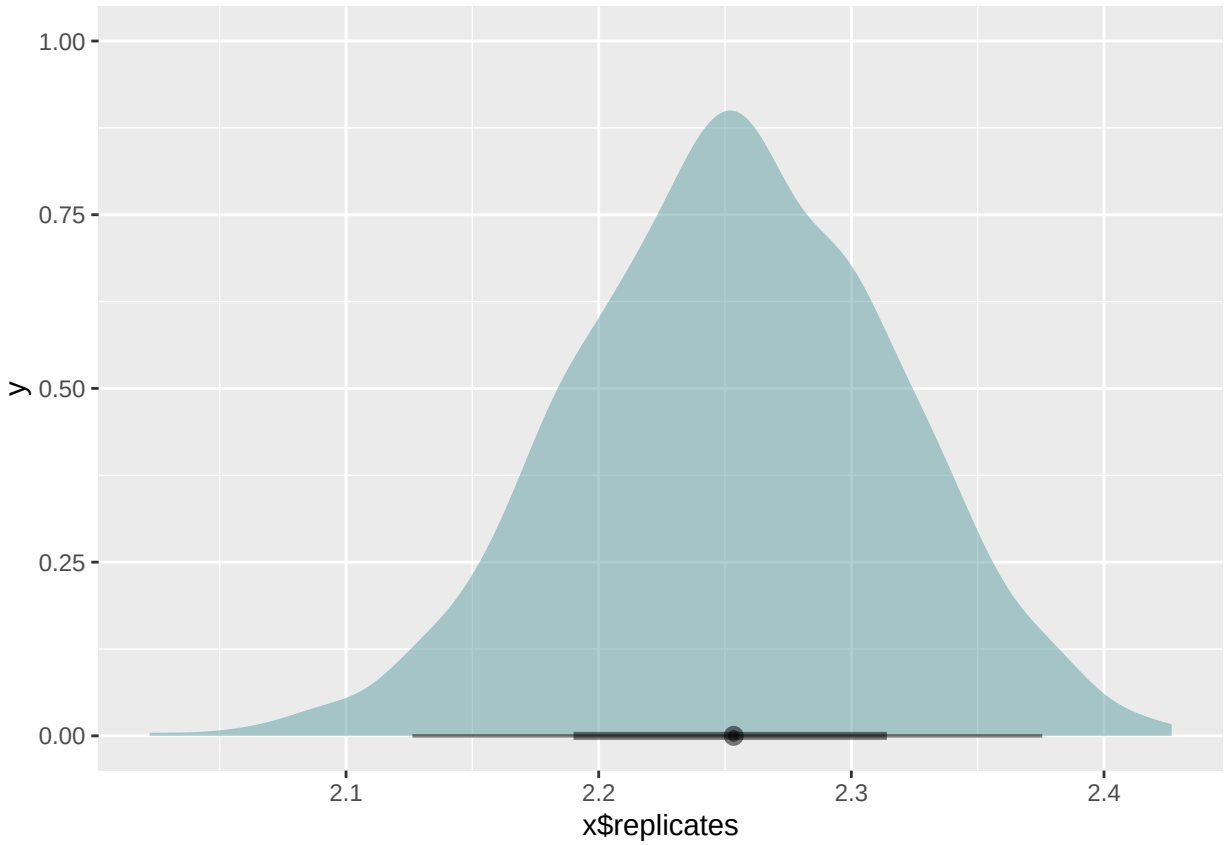
```
plot(lmer_par, "IQ_verb")
```



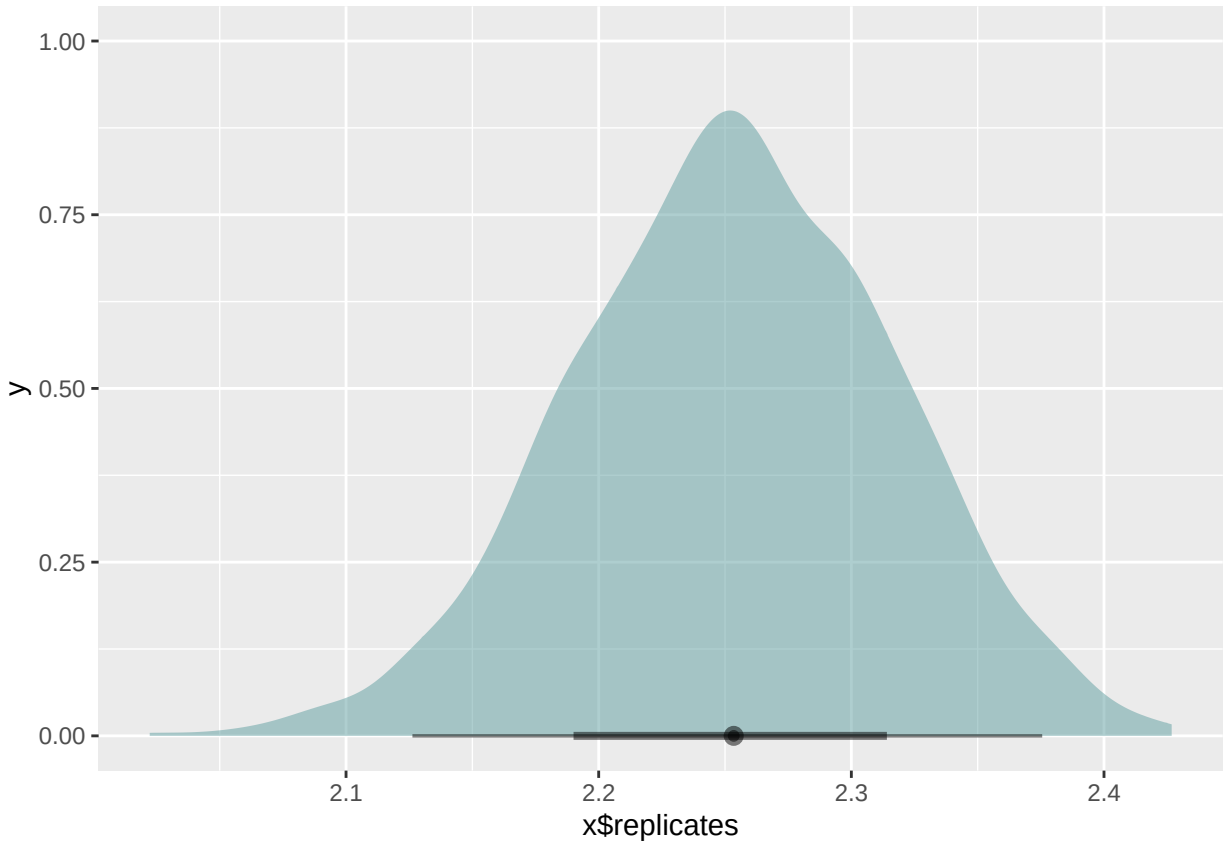

```
plot(lmer_par, "ses")
```



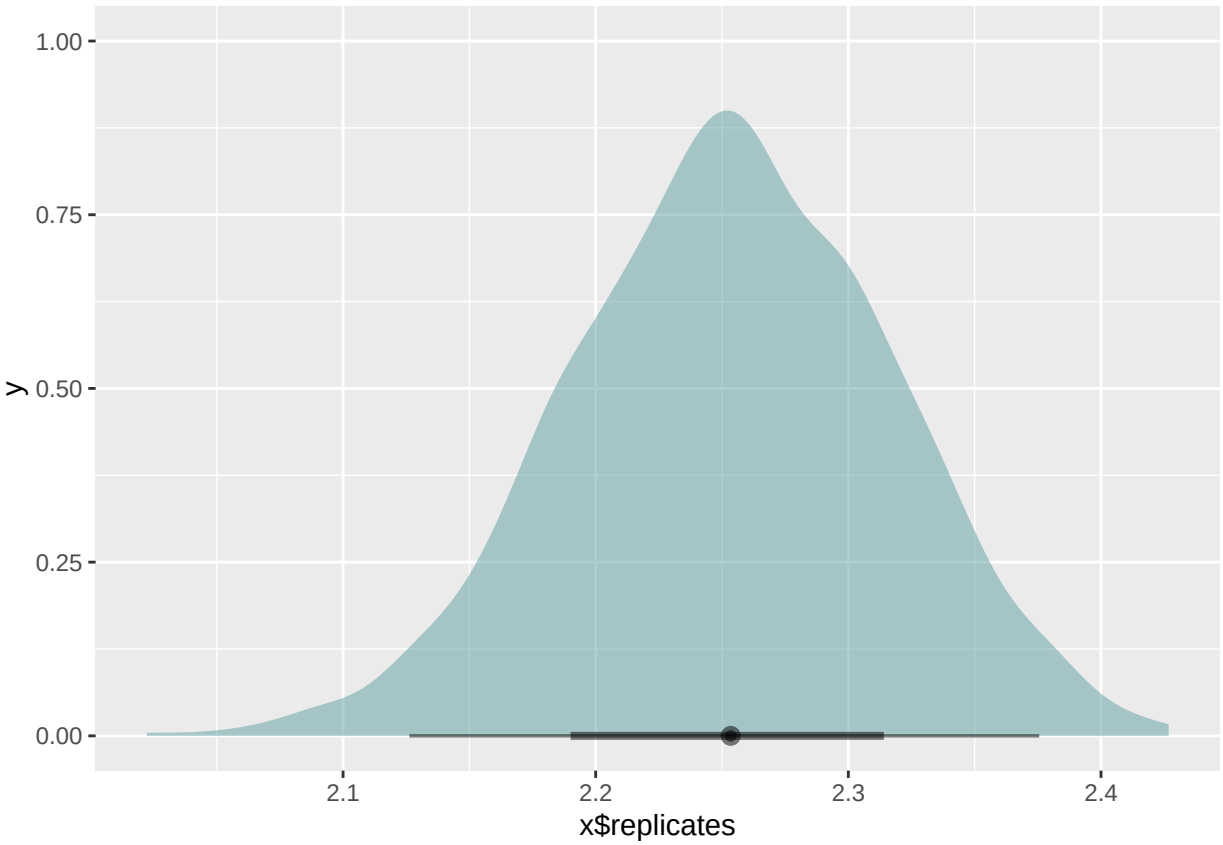

```
plot(lmer_par, "sex")
```




```
plot(lmer_par, "sch_iqv")
```




```
plot(lmer_par, "sch_ses")
```



As you can see, the *SE* for sex is estimated to be 0.0612646 with parametric bootstrap.

Confidence interval

With parametric bootstrap there are three ways to construct confidence intervals via the `boot.ci()` function from the `boot` package: normal, basic, and percentile. We can use the following function:

```
boo_par_ci <- boot::boot.ci(boo_par, type = c("norm", "basic", "perc"),
                            index = 1L)
boo_par_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_par, type = c("norm", "basic", "perc"),
##     index = 1)
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   ( 2.128,  2.368 )   ( 2.120,  2.364 )   ( 2.135,  2.379 )
## Calculations and Intervals on Original Scale
```

```
print(lmer_par, ci = TRUE)
```

```
## Bootstrap type: parametric
##
## Number of resamples: 999
##
##   observed rep.mean        se       bias
## 1  2.24959 2.252917 0.0630829 0.00332728
##
## There were 206 messages, 0 warnings, and 0 errors.
##
## The most commonly occurring message was: boundary (singular) fit: see help('isSingular')
```

```
## # A tibble: 3 x 6
##   term  estimate lower upper type  level
##   <chr>    <dbl> <dbl> <dbl> <chr> <dbl>
## 1 ""        2.25  2.12  2.37 norm   0.95
## 2 ""        2.25  2.12  2.37 basic  0.95
## 3 ""        2.25  2.13  2.38 perc   0.95
```

```
confint(lmer_par, method = "perc", level = 0.95)
```

```
## # A tibble: 3 x 6
##   term  estimate lower upper type  level
##   <chr>    <dbl> <dbl> <dbl> <chr> <dbl>
## 1 ""        2.25  2.12  2.37 norm   0.95
## 2 ""        2.25  2.12  2.37 basic  0.95
## 3 ""        2.25  2.13  2.38 perc   0.95
```

Residual Bootstraps

Whereas parametric bootstrap resamples from independent normal distributions, residual bootstrap samples the residuals. Therefore, residual bootstrap is expected to be more robust to non-normality. `bootmlm` implements three methods for residual bootstrap: differentially reflated residual bootstrap, Carpenter-Goldstein-Rashbash's residual bootstrap (CGR; Carpenter et al., 2003), and transformational residual bootstrap by van der Leeden, Meijer, and Busin (2008). They are all motivated by the fact that the residuals, generally empirical bayes estimates (denoted as $\tilde{u}$ and $\tilde{e}$), are shrinkage estimates and have sampling variabilities much smaller than the population random effects, u and e.

The first residual bootstrap rescale $\tilde{u}$ and $\tilde{e}$ so that their sampling variabilities match those of u and e as implied by the model estimates. This can be obtained by

```
system.time(
  boo_res <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "residual")
)
```

```
##    user  system elapsed
##  41.261   0.880  42.301
```

```
boo_res
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "residual")
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1*  2.24959 0.01291016  0.06035725
```

```
## using lmeresampler package
system.time(
  lmer_resid <- resid_bootstrap(m1, .f = fix_eff, B = 199)
)
```

```
##    user  system elapsed
##   8.368   0.241   8.682
```

```
summary(lmer_resid)
```

```
## Bootstrap type: residual
##
## Number of resamples: 199
##
##   observed rep.mean         se        bias
## 1  2.24959 2.252681 0.05733712 0.003091372
##
## There were 26 messages, 0 warnings, and 0 errors.
##
## The most commonly occurring message was: boundary (singular) fit: see help('isSingular')
```

```
confint(lmer_resid, method = "prec", level = 0.95)
```

```
## # A tibble: 3 x 6
##   term  estimate lower upper type  level
##   <chr>    <dbl> <dbl> <dbl> <chr> <dbl>
## 1 ""        2.25  2.13  2.36 norm   0.95
## 2 ""        2.25  2.12  2.36 basic  0.95
## 3 ""        2.25  2.14  2.37 perc   0.95
```

As you can see, the *SE* for sex is estimated to be 0.0603573 with residual bootstrap.

The second method, CGR, rescale the sample covariance matrix of the *realized values* of the residuals to match the model-implied variance components. This can be obtained by

```
system.time(
  boo_cgr <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "residual_cgr")
)
```

```
##    user  system elapsed
##  41.083   0.914  42.185
```

```
boo_cgr
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "residual_cgr")
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1*  2.24959 -0.004011058  0.06390416
```

The *SE* is estimated to be 0.0639042 with CGR bootstrap.

The third method first transforms the OLS residuals, $\hat{r}_{ij} = y_{ij} - \boldsymbol{x}_{ij}\hat{\boldsymbol{\beta}}$, by the inverse of cholesky factor, $\boldsymbol{L}$, of the model-implied covariance matrix of $\boldsymbol{y}$, $\hat{\boldsymbol{V}}$, so that theoretically $\boldsymbol{L}^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$ should be independent and identically distributed. However, as the true sampling variance of $\hat{r}_{ij}$ is not $\boldsymbol{V}$, I also provide the option `corrected_trans = TRUE` to do the transformation using the theoretically sampling variability of $\hat{r}_{ij}$.

```
# Transformation according to V
system.time(
  boo_tra <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "residual_trans"))
```

```
##    user  system elapsed
##   41.26    0.94   42.52
```

```
boo_tra
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "residual_trans")
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1*  2.24959 0.0005751498   0.0595761
```

```
# Transformation according to the sampling variance of r
system.time(
  boo_trac <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "residual_trans",
                            corrected_trans = TRUE))
```

```
##    user  system elapsed
##  53.262   1.438  55.075
```

```
boo_trac
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "residual_trans",
##     corrected_trans = TRUE)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1*  2.24959 0.001348209  0.06161392
```

The *SE* is estimated to be 0.0595761 and 0.0616139 with and without corrections with the transformational residual bootstrap.

Confidence interval

With residual bootstrap methods there are four ways to construct confidence intervals via the `boot.ci()` function from the `boot` package, with the addition of the bias-corrected and accelarted bootstrap (BCa). We can use the following function:

```
# First need to compute the influence values
inf_val <- empinf_mer(m1, fix_eff, index = 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00269073 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0038289 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00557378 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00287811 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00220452 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00225211 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00205098 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00320949 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00583685 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00829394 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00615681 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00281505 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0040521 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00929698 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00219705 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00280959 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00567641 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00545159 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00547187 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00909515 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00458473 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00216328 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00784547 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00266385 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00568795 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00283698 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00559769 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00202329 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00217381 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00572734 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00495193 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00326787 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00864597 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00974135 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0033536 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00857357 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00478413 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00540099 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00636894 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00709296 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00316854 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00281797 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00208957 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00606181 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00769254 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00237967 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00673628 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00290125 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00396233 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0116139 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00228078 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00415663 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00481408 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00442276 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00260664 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00418914 (tol = 0.002, component 1)
```

```r
# Residual bootstrap
boo_res_ci <- boot::boot.ci(boo_res, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_res_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_res, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.118,  2.355 )   ( 2.113,  2.352 )
##
## Level     Percentile            BCa
## 95%   ( 2.147,  2.386 )   ( 2.114,  2.357 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```
# CGR
boo_cgr_ci <- boot::boot.ci(boo_cgr, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_cgr_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cgr, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.128,  2.379 )   ( 2.130,  2.372 )
##
## Level     Percentile            BCa
## 95%   ( 2.127,  2.369 )   ( 2.124,  2.366 )
## Calculations and Intervals on Original Scale
```

```
# Transformational (no correction)
boo_tra_ci <- boot::boot.ci(boo_tra, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_tra_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_tra, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.132,  2.366 )   ( 2.129,  2.366 )
##
## Level     Percentile            BCa
```

```
## 95%   ( 2.133,  2.370 )   ( 2.124,  2.357 )
## Calculations and Intervals on Original Scale
```

```
# Transformational (with correction)
boo_trac_ci <- boot::boot.ci(boo_trac, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
boo_trac_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_trac, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.127,  2.369 )   ( 2.130,  2.369 )
##
## Level     Percentile            BCa
## 95%   ( 2.130,  2.370 )   ( 2.117,  2.359 )
## Calculations and Intervals on Original Scale
```

Random Effect Block Bootstrap

```
system.time(
  boo_reb <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "reb"))
```

```
##    user  system elapsed
##  49.041   1.272  50.516
```

```
boo_reb
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "reb")
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1*  2.24959 0.1084657  0.09763492
```

```
system.time(
  boo_rebs <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "reb",
                            reb_scale = TRUE))
```

```
##    user  system elapsed
##  42.275   1.067  43.521
```

```
boo_rebs
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "reb",
##     reb_scale = TRUE)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1*  2.24959 0.002170628  0.06295514
```

```
## using lmeresampler package
lmer_reb1 <- reb_bootstrap(m1, .f = fix_eff, B = 199, reb_type = 0)
## using lmeresampler package
lmer_reb2 <- reb_bootstrap(m1, .f = fix_eff, B = 199, reb_type = 1)
## using lmeresampler package
# lmer_reb3 <- parametric_bootstrap(m1, .f = fix_eff, B = 199, reb_type = 2)
```

Confidence interval

With case bootstrap the supported CIs are: normal, basic, and percentile, and BCa. We can use the following function:

```
# Only sampling clusters
boo_reb_ci <- boot::boot.ci(boo_reb, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
```

```
## Warning in norm.inter(t, adj.alpha): extreme order statistics used as endpoints
```

```
boo_reb_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_reb, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.950,  2.332 )   ( 1.943,  2.340 )
##
## Level     Percentile            BCa
## 95%   ( 2.160,  2.556 )   ( 2.097,  2.333 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

```
# Transformational (with correction)
boo_rebs_ci <- boot::boot.ci(boo_rebs, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
boo_rebs_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_rebs, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.124,  2.371 )   ( 2.124,  2.378 )
##
## Level     Percentile            BCa
## 95%   ( 2.121,  2.375 )   ( 2.096,  2.358 )
## Calculations and Intervals on Original Scale
```

Case Bootstrap

With case bootstrap, the observed *cases* are sampled with replacement. However, because of the multilevel structure, we need to resample the clusters. Optionally, we can then resample the cases within each cluster (using the `lv1_resample = TRUE` argument). Unlike the parametric and residual bootstrap methods, currently `bootmlm` only support the case bootstrap with two levels.

```
system.time(
  boo_cas <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "case"))
```

```
##    user  system elapsed
##  44.864   2.104  47.250
```

```
boo_cas
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "case")
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1*  2.24959 0.001969595  0.06475504
```

```
system.time(
  boo_cas1 <- bootstrap_mer(m1, fix_eff, nsim = 999L, type = "case",
                            lv1_resample = TRUE))
```

```
##    user  system elapsed
##  45.045   1.818  47.078
```

```
boo_cas1
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m1, FUN = fix_eff, nsim = 999, type = "case",
##     lv1_resample = TRUE)
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1*  2.24959 0.04545824  0.08641503
```

```
## using lmeresampler package
lmer_case <- case_bootstrap(m1, .f = fix_eff, B = 999, resample = c(FALSE, TRUE))
```

The *SE* for the ICC is estimated to be 0.064755 (only sampling clusters) and 0.086415 (sampling also cases) with case bootstrap.

Confidence interval

With case bootstrap the supported CIs are: normal, basic, and percentile, and BCa. We can use the following function:

```
# Only sampling clusters
boo_cas_ci <- boot::boot.ci(boo_cas, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_cas_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cas, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.121,  2.375 )   ( 2.124,  2.376 )
##
## Level     Percentile            BCa
## 95%   ( 2.123,  2.375 )   ( 2.111,  2.366 )
## Calculations and Intervals on Original Scale
```

```
# Transformational (with correction)
boo_cas1_ci <- boot::boot.ci(boo_cas1, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
```

```
## Warning in norm.inter(t, adj.alpha): extreme order statistics used as endpoints
```

```
boo_cas1_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cas1, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.035,  2.374 )   ( 2.044,  2.378 )
##
## Level     Percentile            BCa
## 95%   ( 2.122,  2.456 )   ( 2.017,  2.366 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

Summary

```
boo_names <- c("parametric", "residual", "cgr", "trans",
               "trans (cor)", "REB", "REB (scaled)",
               "case (cluster)", "case (c + i)")
boo_lst <- list(boo_par, boo_res, boo_cgr, boo_tra, boo_trac,
                boo_reb, boo_rebs, boo_cas, boo_cas1)
boo_ci_lst <- list(boo_par_ci, boo_res_ci, boo_cgr_ci, boo_tra_ci,
                   boo_trac_ci, boo_reb_ci, boo_rebs_ci, boo_cas_ci,
                   boo_cas1_ci)
get_ci <- function(boo_ci, type) {
  paste0("(", paste(comma(tail(boo_ci[[type]][1, ], 2L)), collapse = ", "), ")")
}

tab <- tibble(boot_type = boo_names, boo = boo_lst, boo_ci = boo_ci_lst) %>%
  mutate(sd = map_chr(boo, ~ comma(sd(.x$t))),
         normal = map_chr(boo_ci, ~ get_ci(.x, "normal")),
         basic = map_chr(boo_ci, ~ get_ci(.x, "basic")),
         percentile = map_chr(boo_ci, ~ get_ci(.x, "percent")),
         bca = map_chr(boo_ci, ~ get_ci(.x, "bca"))) %>%
  select(-boo, -boo_ci)
knitr::kable(
  tab
)
```

boot_type	sd	normal	basic	percentile	bca
parametric	0.06126	(2.128, 2.368)	(2.120, 2.364)	(2.135, 2.379)	(NULL)
residual	0.06036	(2.118, 2.355)	(2.113, 2.352)	(2.147, 2.386)	(2.114, 2.357)
cgr	0.0639	(2.128, 2.379)	(2.130, 2.372)	(2.127, 2.369)	(2.124, 2.366)

boot_type	sd	normal	basic	percentile	bca
trans	0.05958	(2.132, 2.366)	(2.129, 2.366)	(2.133, 2.370)	(2.124, 2.357)
trans (cor)	0.06161	(2.127, 2.369)	(2.130, 2.369)	(2.13, 2.37)	(2.117, 2.359)
REB	0.09763	(1.950, 2.332)	(1.943, 2.340)	(2.160, 2.556)	(2.097, 2.333)
REB (scaled)	0.06296	(2.124, 2.371)	(2.124, 2.378)	(2.121, 2.375)	(2.096, 2.358)
case (cluster)	0.06476	(2.121, 2.375)	(2.124, 2.376)	(2.123, 2.375)	(2.111, 2.366)
case (c + i)	0.08642	(2.035, 2.374)	(2.044, 2.378)	(2.122, 2.456)	(2.017, 2.366)

Bootstrapping for ICC

```
m_null <- lmer(langPOST ~ (1 | schoolnr),
               data = sj_dat)
```

The intraclass correlation is defined as

$$\rho = \frac{\tau}{\tau + \sigma^2} = \frac{1}{1 + \sigma^2/\tau} = \frac{1}{1 + \theta^{-2}},$$

where $\theta = \sqrt{\tau}/\sigma$ is the relative cholesky factor for the random intercept term used in `lme4`. Therefore, we can estimate the ICC as:

```
(icc0 <- 1 / (1 + getME(m_null, "theta")^(-2)))
```

```
## schoolnr.(Intercept)
##            0.2249341
```

So the ICC is quite large for this data set. However, it is important to also quantify the uncertainty of a point estimate. Although there are analytic methods to obtain *SE* and CI for ICC, a reliable alternative is to do bootstrapping. We first define the function for computing the test statistic:

```
icc <- function(x) 1 / (1 + x@theta^(-2))
icc(m_null)
```

```
## [1] 0.2249341
```

With the `bootmlm` package we can perform various bootstrap methods using the `bootstrap_mer()` function.

Parameteric Bootstrap

We can run parametric bootstrap, which essentially call the `lme4::bootMer()` function. It's usually recommended to have a large number of bootstrap samples (R), especially for CIs with higher confidence levels. For illustrative purpose I will use $R = 999$, but in general 1,999 or more is recommended

```
system.time(
  boo_par <- bootstrap_mer(m_null, icc, nsim = 999L, type = "parametric")
)
```

```
##    user  system elapsed
##   8.082   0.081   8.203
```

```
boo_par
```

```
##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## lme4::bootMer(x = x, FUN = FUN, nsim = nsim, seed = seed, use.u = FALSE,
##     type = "parametric", verbose = FALSE)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.2249341 -0.0001984226  0.02177171
```

As you can see, the *SE* for the ICC is estimated to be 0.0217717 with parametric bootstrap.

Confidence interval

With parametric bootstrap there are three ways to construct confidence intervals via the `boot.ci()` function from the `boot` package: normal, basic, and percentile. We can use the following function:

```
boo_par_ci <- boot::boot.ci(boo_par, type = c("norm", "basic", "perc"),
                            index = 1L)
boo_par_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_par, type = c("norm", "basic", "perc"),
##     index = 1)
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   ( 0.1825,  0.2678 )   ( 0.1821,  0.2678 )   ( 0.1821,  0.2677 )
## Calculations and Intervals on Original Scale
```

Residual Bootstraps

Whereas parametric bootstrap resamples from independent normal distributions, residual bootstrap samples the residuals. Therefore, residual bootstrap is expected to be more robust to non-normality. `bootmlm` implements three methods for residual bootstrap: differentially reflated residual bootstrap, Carpenter-Goldstein-Rashbash's residual bootstrap (CGR; Carpenter et al., 2003), and transformational residual bootstrap by van der Leeden, Meijer, and Busin (2008). They are all motivated by the fact that the residuals, generally empirical bayes estimates (denoted as $\tilde{u}$ and $\tilde{e}$), are shrinkage estimates and have sampling variabilities much smaller than the population random effects, u and e.

The first residual bootstrap rescale $\tilde{u}$ and $\tilde{e}$ so that their sampling variabilities match those of u and e as implied by the model estimates. This can be obtained by

```
system.time(
  boo_res <- bootstrap_mer(m_null, icc, nsim = 999L, type = "residual"))
```

```
##    user  system elapsed
##   8.715   0.129   8.885
```

```
boo_res
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "residual")
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.2249341 0.006034797  0.02445122
```

As you can see, the *SE* for the ICC is estimated to be 0.0244512 with residual bootstrap.

The second method, CGR, rescale the sample covariance matrix of the *realized values* of the residuals to match the model-implied variance components. This can be obtained by

```
system.time(
  boo_cgr <- bootstrap_mer(m_null, icc, nsim = 999L, type = "residual_cgr"))
```

```
##    user  system elapsed
##   8.420   0.057   8.499
```

```
boo_cgr
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "residual_cgr")
##
##
## Bootstrap Statistics :
##      original       bias    std. error
## t1* 0.2249341 0.0001941401  0.02209763
```

The *SE* is estimated to be 0.0220976 with CGR bootstrap.

The third method first transforms the OLS residuals, $\hat{r}_{ij} = y_{ij} - \boldsymbol{x}_{ij}\hat{\boldsymbol{\beta}}$, by the inverse of cholesky factor, $\boldsymbol{L}$, of the model-implied covariance matrix of $\boldsymbol{y}$, $\hat{\boldsymbol{V}}$, so that theoretically $\boldsymbol{L}^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$ should be independent and identically distributed. However, as the true sampling variance of $\hat{r}_{ij}$ is not $\boldsymbol{V}$, I also provide the option `corrected_trans = TRUE` to do the transformation using the theoretically sampling variability of $\hat{r}_{ij}$.

```
# Transformation according to V
system.time(
  boo_tra <- bootstrap_mer(m_null, icc, nsim = 999L, type = "residual_trans"))
```

```
##    user  system elapsed
##   8.383   0.055   8.466
```

```
boo_tra
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "residual_trans")
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.2249341 -0.00118075  0.02159347
```

```
# Transformation according to the sampling variance of r
system.time(
  boo_trac <- bootstrap_mer(m_null, icc, nsim = 999L, type = "residual_trans",
                            corrected_trans = TRUE))
```

```
##    user  system elapsed
##  20.402   0.525  21.106
```

```
boo_trac
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "residual_trans",
##     corrected_trans = TRUE)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.2249341 -0.0001916893  0.02102375
```

The *SE* is estimated to be 0.0215935 and 0.0210237 with and without corrections with the transformational residual bootstrap.

Confidence interval

With residual bootstrap methods there are four ways to construct confidence intervals via the `boot.ci()` function from the `boot` package, with the addition of the bias-corrected and accelarted bootstrap (BCa). We can use the following function:

```r
# First need to compute the influence values
inf_val <- empinf_mer(m_null, icc, index = 1)
# Residual bootstrap
boo_res_ci <- boot::boot.ci(boo_res, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_res_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_res, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1710,  0.2668 )   ( 0.1719,  0.2651 )
##
## Level     Percentile            BCa
## 95%   ( 0.1848,  0.2779 )   ( 0.1848,  0.2788 )
## Calculations and Intervals on Original Scale
```

```r
# CGR
boo_cgr_ci <- boot::boot.ci(boo_cgr, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_cgr_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cgr, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1814,  0.2681 )   ( 0.1823,  0.2695 )
##
## Level     Percentile            BCa
## 95%   ( 0.1803,  0.2676 )   ( 0.1901,  0.2817 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```r
# Transformational (no correction)
boo_tra_ci <- boot::boot.ci(boo_tra, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_tra_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
```

```
## boot::boot.ci(boot.out = boo_tra, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1838,  0.2684 )   ( 0.1796,  0.2701 )
##
## Level     Percentile            BCa
## 95%   ( 0.1798,  0.2702 )   ( 0.1916,  0.2860 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```
# Transformational (with correction)
boo_trac_ci <- boot::boot.ci(boo_trac, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
boo_trac_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_trac, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1839,  0.2663 )   ( 0.1843,  0.2660 )
##
## Level     Percentile            BCa
## 95%   ( 0.1838,  0.2655 )   ( 0.1923,  0.2847 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Random Effect Block Bootstrap

```
system.time(
  boo_reb <- bootstrap_mer(m_null, icc, nsim = 999L, type = "reb"))
```

```
##    user  system elapsed
##   9.506   0.102   9.667
```

```
boo_reb
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "reb")
##
```

```
##
## Bootstrap Statistics :
##      original     bias    std. error
## t1* 0.2249341 0.07178652   0.0286524
```

```
system.time(
  boo_rebs <- bootstrap_mer(m_null, icc, nsim = 999L, type = "reb",
                            reb_scale = TRUE))
```

```
##    user  system elapsed
##   9.548   0.089   9.685
```

```
boo_rebs
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "reb",
##     reb_scale = TRUE)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.2249341 0.002908254  0.02599828
```

Confidence interval

```
# Only sampling clusters
boo_reb_ci <- boot::boot.ci(boo_reb, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
```

```
## Warning in norm.inter(t, adj.alpha): extreme order statistics used as endpoints
```

```
boo_reb_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_reb, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.0970,  0.2093 )   ( 0.0928,  0.2073 )
##
## Level     Percentile            BCa
## 95%   ( 0.2426,  0.3571 )   ( 0.2129,  0.2129 )
```

```
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

```
# Transformational (with correction)
boo_rebs_ci <- boot::boot.ci(boo_rebs, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
boo_rebs_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_rebs, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1711,  0.2730 )   ( 0.1704,  0.2718 )
##
## Level     Percentile            BCa
## 95%   ( 0.1781,  0.2794 )   ( 0.1857,  0.2956 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Case Bootstrap

With case bootstrap, the observed *cases* are sampled with replacement. However, because of the multilevel structure, we need to resample the clusters. Optionally, we can then resample the cases within each cluster (using the `lv1_resample = TRUE` argument). Unlike the parametric and residual bootstrap methods, currently `bootmlm` only support the case bootstrap with two levels.

```
system.time(
  boo_cas <- bootstrap_mer(m_null, icc, nsim = 999L, type = "case"))
```

```
##    user  system elapsed
##  16.562   0.586  17.242
```

```
boo_cas
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "case")
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.2249341 -0.0005386712  0.02505031
```

```
system.time(
  boo_cas1 <- bootstrap_mer(m_null, icc, nsim = 999L, type = "case",
                            lv1_resample = TRUE))
```

```
##    user  system elapsed
##  17.355   0.517  17.940
```

```
boo_cas1
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## bootstrap_mer(x = m_null, FUN = icc, nsim = 999, type = "case",
##     lv1_resample = TRUE)
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1* 0.2249341 0.0512945  0.02882308
```

The *SE* for the ICC is estimated to be 0.0250503 (only sampling clusters) and 0.0288231 (sampling also cases) with case bootstrap.

Confidence interval

With case bootstrap the supported CIs are: normal, basic, and percentile, and BCa. We can use the following function:

```
# Only sampling clusters
boo_cas_ci <- boot::boot.ci(boo_cas, type = c("norm", "basic", "perc", "bca"),
                            index = 1L, L = inf_val)
boo_cas_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cas, type = c("norm", "basic", "perc",
##     "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1764,  0.2746 )   ( 0.1710,  0.2707 )
##
## Level     Percentile            BCa
## 95%   ( 0.1791,  0.2789 )   ( 0.1903,  0.3287 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```
# Transformational (with correction)
boo_cas1_ci <- boot::boot.ci(boo_cas1, type = c("norm", "basic", "perc", "bca"),
                             index = 1L, L = inf_val)
```

```
## Warning in norm.inter(t, adj.alpha): extreme order statistics used as endpoints
```

```
boo_cas1_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boo_cas1, type = c("norm", "basic",
##     "perc", "bca"), index = 1, L = inf_val)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1171,  0.2301 )   ( 0.1164,  0.2297 )
##
## Level     Percentile            BCa
## 95%   ( 0.2202,  0.3334 )   ( 0.1852,  0.2306 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

Summary

```
boo_names <- c("parametric", "residual", "cgr", "trans",
               "trans (cor)", "REB", "REB (scaled)",
               "case (cluster)", "case (c + i)")
boo_lst <- list(boo_par, boo_res, boo_cgr, boo_tra, boo_trac,
                boo_reb, boo_rebs, boo_cas, boo_cas1)
boo_ci_lst <- list(boo_par_ci, boo_res_ci, boo_cgr_ci, boo_tra_ci,
                   boo_trac_ci, boo_reb_ci, boo_rebs_ci, boo_cas_ci,
                   boo_cas1_ci)
get_ci <- function(boo_ci, type) {
  paste0("(", paste(comma(tail(boo_ci[[type]][1, ], 2L)), collapse = ", "), ")")
}
tab <- tibble(boot_type = boo_names, boo = boo_lst, boo_ci = boo_ci_lst) %>%
  mutate(sd = map_chr(boo, ~ comma(sd(.x$t))),
         normal = map_chr(boo_ci, ~ get_ci(.x, "normal")),
         basic = map_chr(boo_ci, ~ get_ci(.x, "basic")),
         percentile = map_chr(boo_ci, ~ get_ci(.x, "percent")),
         bca = map_chr(boo_ci, ~ get_ci(.x, "bca"))) %>%
  select(-boo, -boo_ci)
knitr::kable(tab)
```

boot_type	sd	normal	basic	percentile	bca
parametric	0.02177	(0.1825, 0.2678)	(0.1821, 0.2678)	(0.1821, 0.2677)	(NULL)
residual	0.02445	(0.1710, 0.2668)	(0.1719, 0.2651)	(0.1848, 0.2779)	(0.1848, 0.2788)
cgr	0.0221	(0.1814, 0.2681)	(0.1823, 0.2695)	(0.1803, 0.2676)	(0.1901, 0.2817)
trans	0.02159	(0.1838, 0.2684)	(0.1796, 0.2701)	(0.1798, 0.2702)	(0.1916, 0.2860)
trans (cor)	0.02102	(0.1839, 0.2663)	(0.1843, 0.2660)	(0.1838, 0.2655)	(0.1923, 0.2847)
REB	0.02865	(0.09699, 0.20931)	(0.09281, 0.20729)	(0.2426, 0.3571)	(0.2129, 0.2129)
REB (scaled)	0.026	(0.1711, 0.2730)	(0.1704, 0.2718)	(0.1781, 0.2794)	(0.1857, 0.2956)
case (cluster)	0.02505	(0.1764, 0.2746)	(0.1710, 0.2707)	(0.1791, 0.2789)	(0.1903, 0.3287)
case (c + i)	0.02882	(0.1171, 0.2301)	(0.1164, 0.2297)	(0.2202, 0.3334)	(0.1852, 0.2306)