Model selection

Parsimonious model selection

Daniela Palleschi

2024-02-09

Table of contents

Learning Objectives

Today we will…

- apply remedies for nonconvergence
- reduce our RES with a data-driven approach
- compare a parsimonious model to maximal and intercept-only models

Resources

- this lecture covers
  - Sections 10.3-5 in Sonderegger (2023)
  - Section 15.7.3 'Convergence Issues' in Winter (2019)
  - Brauer & Curtin (2018)
  - Meteyard & Davies (2020)
- we will continue using the data from Biondo et al. (2022)

Set-up

```
# suppress scientific notation
options(scipen=999)
```

```
library(broman)
# function to format p-values
format_pval <- function(pval){
    dplyr::case_when(
        pval < .001 ~ "< .001",
        pval < .01 ~ "< .01",
        pval < .05 ~ "< .05",
        TRUE ~ broman::myround(pval, 3)
    )
}
```

Load packages

```
# load libraries
pacman::p_load(
               tidyverse,
               here,
               janitor,
               # new packages for mixed models:
               lme4,
               lmerTest,
               broom.mixed,
               lattice)
```

```
lmer <- lmerTest::lmer
```

Load data

- data from Biondo et al. (2022)

```
df_biondo <-
  read_csv(here("data", "Biondo.Soilemezidi.Mancini_dataset_ET.csv"),
           locale = locale(encoding = "Latin1") ## for special characters in Spanish
```

```
            ) |>
    clean_names() |>
    mutate(gramm = ifelse(gramm == "0", "ungramm", "gramm")) |>
    mutate_if(is.character,as_factor) |> # all character variables as factors
    droplevels() |>
    filter(adv_type == "Deic")
```

Set contrasts

```
  contrasts(df_biondo$verb_t) <- c(-0.5,+0.5)
  contrasts(df_biondo$gramm) <- c(-0.5,+0.5)
```

```
  contrasts(df_biondo$verb_t)
```

```
       [,1]
Past   -0.5
Future  0.5
```

```
  contrasts(df_biondo$gramm)
```

```
         [,1]
gramm    -0.5
ungramm   0.5
```

Start maximal

- model structure should be decided *a priori*
  - included fixed (predictors and covariates) and random effects

Maximal model

- starting point: most maximal model structure justified by your design
  - if this converges, great!
  - if it doesn't, what does this mean and what should we do?

```
fit_verb_fp_mm <- lmer(log(fp) ~ verb_t*gramm +
                        (1 + verb_t*gramm|sj) +
                        (1 + verb_t*gramm|item),
                      data = df_biondo,
                      subset = roi == 4)
```

- we get a warning of singular fit

Convergence issues

- "*Convergence is not a metric of model quality*" (Sonderegger, 2023, p. 365, Box 10.2)
  - convergence does not always indicate "overfitting" or "overparameterisation"
  - can also be due to optimizer choice
    - since default optimizer was changed to `nloptwrap` from `bobyqa`, there seem to be more 'false positive' convergence warnings
- false-positive convergence: you get a convergence warning, but changing the optimizer and/or iteration count does not produce a warning
- false-negative convergence: you do not get a warning, but your variance-covariance matrix might indicate overfitting

Nonconvergence remedies

- unfortunately there is no one "right" way to deal with convergence issues
  - important is to transparently report and justify your method
- Table 17 in Brauer & Curtin (2018) (p. 404) suggests 20 remedies, whittled down to 10 suggestions in Sonderegger (2023)

Intrusive vs. Non-intrusive remedies

- non-intrusive remedies amount to checking/adjusting data and model specifications
- intrusive remedies involve reducing random effects structure
  - there are different schools of thought
    - random-intercepts only: increased Type I error rate = overconfident estimates
    - maximal-but-singular-fit model (Barr et al., 2013): reduces power = underconfident estimates

Table 10.1
Possible fixes for non-convergent (non-intrusive + intrusive) and singular models (intrusive only), ordered by which to try first (adapted from Brauer and Curtin 2018). Fixes 2(a) and 2(b) are tied.

1. Nonintrusive
   a. Check your data and model
   b. Standardize predictors (center, possibly scale)
   c. Increase number of iterations
   d. Change the optimizer
   e. Give the optimizer better start values
2. Intrusive
   a. Remove random effects involving control predictors (must not be in interactions with critical predictors)
   b. Selectively remove random-effect correlations: for control predictors, then correlations that are probably close to 0
   c. Remove random intercept (leaving slope terms in)
   d. Remove random slopes for critical predictors

Figure 1: From Sonderegger (2023), p. 366

    - ∗ data-driven approach (Bates et al., 2015): can lose the forest for the trees, e.g., removing random slopes for predictors of interest

- each strategy has its drawback
  - important is to choose your strategy *a priori* and transparently report and justify your strategy
  - even better: share/publish your data and code, which should be reproducible

`?convergence`

- type `?convergence` in the Console and read the vignette
  - what suggestions does it make?
- compare this to `?isSingular`

Non-intrusive methods

- check your data structure/variables
  - check model assumptions (e.g., normality, missing transformations of variables)
  - check your RES is justified by your experimental design/data structure

  - centre your predictors (e.g., sum contrasts, or centring/standardizing) to reduce multicollinearity; reduces collinearity in the random effects (a possible source of nonconvergence)
  - check observations per cell (e.g., is there a participant very few observations, or few observations per one condition? Should be at least >5 per cell)
- alter model controls:
  - increase iterations
  - check optimizer

Check optimzer

- optimizer
  - `lme4::allFit(model)` (can take a while to run)

```
all_fit_verb_fp_mm <- allFit(fit_verb_fp_mm)
# bobyqa : boundary (singular) fit: see help('isSingular')
# [OK]
# Nelder_Mead : [OK]
# nlminbwrap : boundary (singular) fit: see help('isSingular')
# [OK]
# nmkbw : [OK]
# optimx.L-BFGS-B : boundary (singular) fit: see help('isSingular')
# [OK]
# nloptwrap.NLOPT_LN_NELDERMEAD : boundary (singular) fit: see help('isSingular')
# [OK]
# nloptwrap.NLOPT_LN_BOBYQA : boundary (singular) fit: see help('isSingular')
# [OK]
# There were 11 warnings (use warnings() to see them)
```

Optimizers

- default optimizer for `lmer()` is `nloptwrap`, formerly `bobyqa` (Bound Optimization by Quaradric Approximiation)
  - usually changing to `bobyqa` helps
- see `?lmerControl` for more info
- if fits are very similar (or all optimizeres except the default), the nonconvergent fit was a false positive

- it's safe to use the new optimizer

```
summary(all_fit_verb_fp_mm)$llik
```

```
                     bobyqa                     Nelder_Mead
                  -2105.109                       -2179.479
                 nlminbwrap                            nmkbw
                  -2105.106                       -2105.109
            optimx.L-BFGS-B nloptwrap.NLOPT_LN_NELDERMEAD
                  -2105.106                       -2105.106
  nloptwrap.NLOPT_LN_BOBYQA
                  -2105.106
```

```
summary(all_fit_verb_fp_mm)$fixef
```

```
                              (Intercept)    verb_t1      gramm1 verb_t1:gramm1
bobyqa                          5.956403 0.06170602 0.003369634    -0.01418865
Nelder_Mead                     5.956350 0.06188102 0.003488675    -0.01397531
nlminbwrap                      5.956403 0.06170726 0.003369637    -0.01419047
nmkbw                           5.956404 0.06170653 0.003369153    -0.01419036
optimx.L-BFGS-B                 5.956403 0.06170717 0.003369787    -0.01419044
nloptwrap.NLOPT_LN_NELDERMEAD   5.956403 0.06170725 0.003369649    -0.01419046
nloptwrap.NLOPT_LN_BOBYQA       5.956403 0.06170771 0.003369203    -0.01419184
```

Increase iterations

- and/or increase number of iterations
  - default is 10 000 (`1e5` in scientific notation)
  - you can try 20 000, 100 000, etc.
  - this usually helps with larger data or models with complex RES

```
# check n of iterations
fit_verb_fp_mm@optinfo$feval
```

```
[1] 2318
```

`lmerControl()`

```
fit_verb_fp_mm <- lmer(log(fp) ~ verb_t*gramm +
                       (1 + verb_t*gramm|sj) +
                       (1 + verb_t*gramm|item),
                     data = df_biondo,
                     subset = roi == 4,
                     control = lmerControl(optimizer = "bobyqa",
                                           optCtrl = list(maxfun = 2e5))
)
```

- or you can just 'update' the model to save some syntax

```
fit_verb_fp_mm <- update(fit_verb_fp_mm,
                         control = lmerControl(optimizer = "bobyqa",
                                               optCtrl = list(maxfun = 2e5)))
```

```
boundary (singular) fit: see help('isSingular')
```

```
Warning: Model failed to converge with 1 negative eigenvalue: -5.3e-01
```

Removing parameters

- still won't converge?
  - it's time to consider intrusive remedies: removing random effects parameters

Intrusive methods

- nonconvergence in maximal models is often due to overfitting
  - i.e., the model is overly complex given your data
  - this is typically due to an overly complex random effects structure
- if the non-intrusive methods don't lead to convergence, the problem is likely overfitting

Parsimonious vs. maximal

- there are different camps on how to deal with this issue
- I personally follow the suggestions in Bates et al. (2015) (for now)
    1. run random effects Principal Components Analysis (`summary(rePCA(model))`, `lme4` package)
        - informs by how many parameters our model is overfit
    2. check variance-covariance matrix (`VarCorr(model)`)
    3. remove parameters with very high or low Correlation terms and/or much lower variance compared to other terms
    4. fit simplified model
    5. wash, rinse, repeat
- we'll practice this method today, but keep in mind that it's up to you to decide and justify which method you use

Random effects Principal Components Analysis

- gives us a ranking of all parameters ('components') in our RES per unit

```
summary(rePCA(fit_verb_fp_mm))
```

```
$item
Importance of components:
                          [,1]   [,2]    [,3]                      [,4]
Standard deviation     0.3638 0.2493 0.08366 0.00000000000000000004965
Proportion of Variance 0.6567 0.3085 0.03474 0.00000000000000000000000
Cumulative Proportion  0.6567 0.9653 1.00000 1.00000000000000000000000

$sj
Importance of components:
                          [,1]    [,2]        [,3]         [,4]
Standard deviation     0.6490 0.01470 0.000007463 0.0000001104
Proportion of Variance 0.9995 0.00051 0.000000000 0.0000000000
Cumulative Proportion  0.9995 1.00000 1.000000000 1.0000000000
```

- important is the Cumulative Proportion

  - how much of the cumulative variance explained by all the by-unit parameters does this one parameter contribute?
  - we see for item, the first component accounts for 66% of the variance explained, and the next contributes an additional 31%, and the next 3%
  - so two components account for roughly 97% of variance explained by our RES
  - in other words, we can remove one component for sure, and possibly another
  - we could potentially remove 3 components from participant

Variance-covariance matrix

- so we can remove 2 parameters from item and participant
  - so either the varying intercept, or slope for tense, grammaticality, or their interaction
- we can check this with `VarCorr(fit_verb_fp_mm)`

```
VarCorr(fit_verb_fp_mm)
```

```
Groups   Name           Std.Dev. Corr
item     (Intercept)    0.139189
         verb_t1        0.055890  0.488
         gramm1         0.022569 -0.109 -0.921
         verb_t1:gramm1 0.095313 -0.283  0.456 -0.646
sj       (Intercept)    0.257535
         verb_t1        0.018297 0.974
         gramm1         0.012055 0.960  0.872
         verb_t1:gramm1 0.017731 0.990  0.933  0.990
Residual                0.399095
```

- for item I would remove `gramm` because it has the lowest variance, and has a pretty high correlation with `verb_t` (which is unlikely to be true)
- I would also remove `gramm` for participant for the same reason, as well as its high correlation with the intercept and `verb_t`

Alternate model 1

- for now let's just remove the interaction term
  - for reproducibility reasons, do not delete the code for a model that did not converge
  - rather, write a comment on what decision was made (and why) for the new model

```
fit_verb_fp_m1 <- lmer(log(fp) ~ verb_t*gramm +
                        (1 + verb_t+gramm|sj) +
                        (1 + verb_t+gramm|item),
                      data = df_biondo,
                      subset = roi == 4,
                      control = lmerControl(optimizer = "bobyqa",
                                            optCtrl = list(maxfun = 2e5))
)
```

```
boundary (singular) fit: see help('isSingular')
```

rePCA()

```
summary(rePCA(fit_verb_fp_m1))
```

```
$item
Importance of components:
                          [,1]   [,2] [,3]
Standard deviation     0.3559 0.1291    0
Proportion of Variance 0.8837 0.1163    0
Cumulative Proportion  0.8837 1.0000    1

$sj
Importance of components:
                          [,1]         [,2] [,3]
Standard deviation     0.6465 0.0000004537    0
Proportion of Variance 1.0000 0.0000000000    0
Cumulative Proportion  1.0000 1.0000000000    1
```

VarCorr()

```
VarCorr(fit_verb_fp_m1)
```

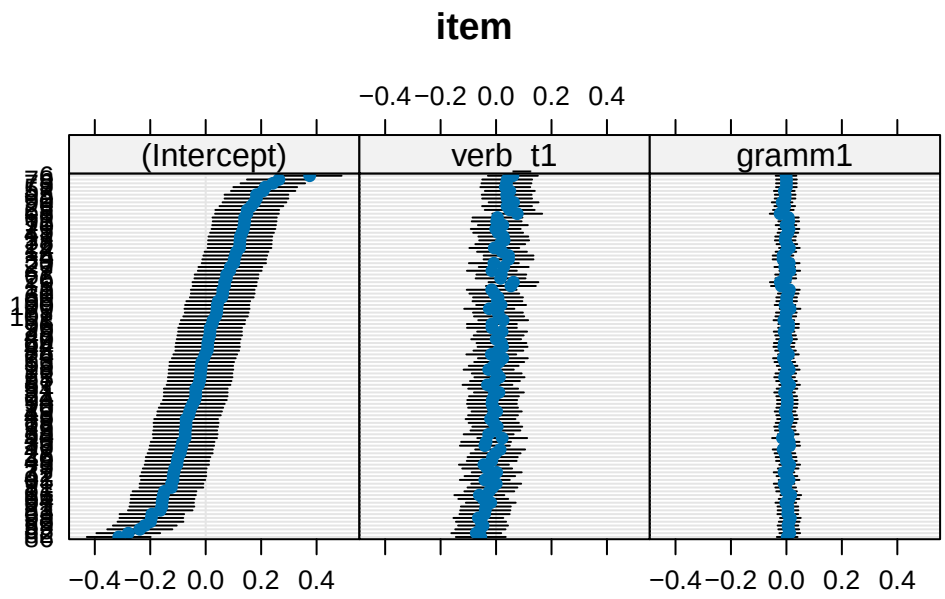
```
 Groups   Name        Std.Dev. Corr
 item     (Intercept) 0.139274
          verb_t1     0.055550  0.489
          gramm1      0.020747 -0.117 -0.924
 sj       (Intercept) 0.257657
          verb_t1     0.017584 1.000
```

```
          gramm1       0.011554 1.000  1.000
Residual               0.399869
```

- when we see Corr +/-1, this tells us there was an error computing correlations between parameters
  - it is an invitation to explore
- this is not plausible, and indicates overfitting in our model
  - we can remove all slopes from sj

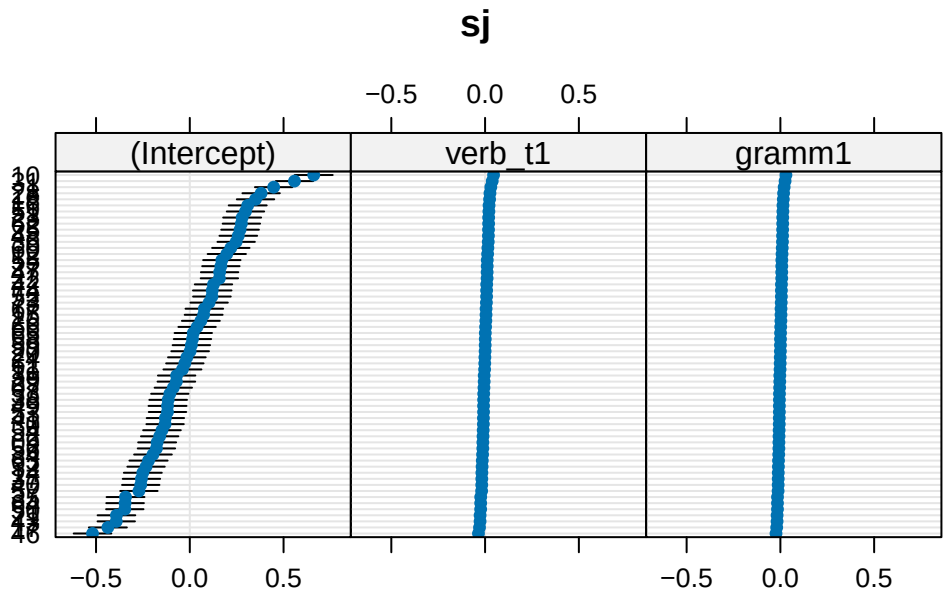
by-item random effects

```
lattice::dotplot(ranef(fit_verb_fp_m1))$item
```



by-participant random effects (with +1 correlations)

```
lattice::dotplot(ranef(fit_verb_fp_m1))$sj
```



Alternate model 2

```
fit_verb_fp_m2 <- lmer(log(fp) ~ verb_t*gramm +
                       (1 |sj) +
                       (1 + verb_t+gramm|item),
                     data = df_biondo,
                     subset = roi == 4,
                     control = lmerControl(optimizer = "bobyqa",
                                           optCtrl = list(maxfun = 2e5))
)
```

```
boundary (singular) fit: see help('isSingular')
```

rePCA()

```
summary(rePCA(fit_verb_fp_m2))
```

```
$item
Importance of components:
                          [,1]   [,2] [,3]
Standard deviation     0.3559 0.1297    0
Proportion of Variance 0.8827 0.1173    0
```

```
Cumulative Proportion  0.8827 1.0000    1

$sj
Importance of components:
                          [,1]
Standard deviation     0.6441
Proportion of Variance 1.0000
Cumulative Proportion  1.0000
```

VarCorr()

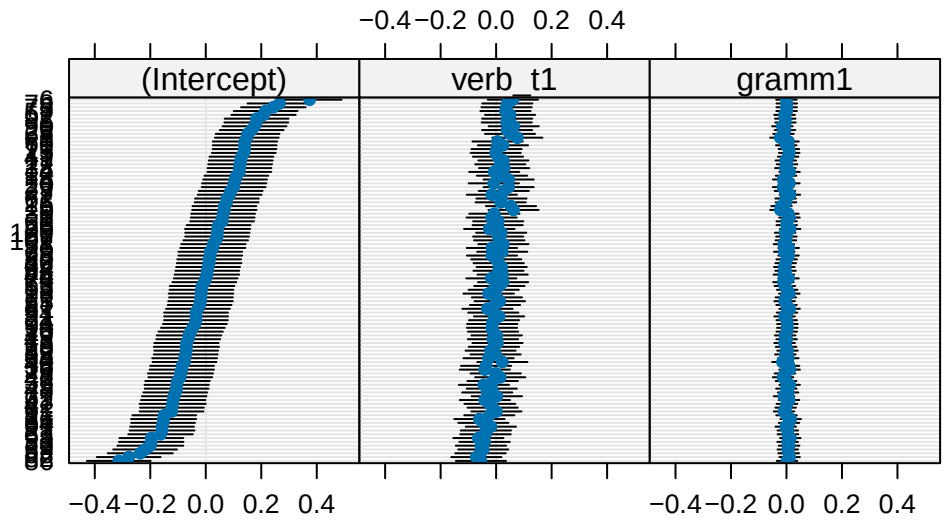
```
VarCorr(fit_verb_fp_m2)
```

```
 Groups   Name        Std.Dev. Corr
 item     (Intercept) 0.139364
          verb_t1     0.055805  0.485
          gramm1      0.020546 -0.097 -0.917
 sj       (Intercept) 0.257648
 Residual             0.399995
```

- by-item slopes for **gramm** and **verb_t** are highly correlated
- **gramm** has least variance, so let's remove it

by-item random effects

```
lattice::dotplot(ranef(fit_verb_fp_m2))$item
```



Alternate model 3

```
fit_verb_fp_m3 <- lmer(log(fp) ~ verb_t*gramm +
                       (1 |sj) +
                       (1 + verb_t|item),
                     data = df_biondo,
                     subset = roi == 4,
                     control = lmerControl(optimizer = "bobyqa",
                                           optCtrl = list(maxfun = 2e5))
)
```

- converged!

`rePCA()`

```
summary(rePCA(fit_verb_fp_m3))
```

```
$item
Importance of components:
                          [,1]    [,2]
Standard deviation     0.3553 0.10311
Proportion of Variance 0.9223 0.07768
Cumulative Proportion  0.9223 1.00000
```

```
$sj
Importance of components:
                          [,1]
Standard deviation     0.6438
Proportion of Variance 1.0000
Cumulative Proportion  1.0000
```

`VarCorr()`

```
VarCorr(fit_verb_fp_m3)
```

```
 Groups   Name        Std.Dev. Corr
 item     (Intercept) 0.139365
          verb_t1     0.050134 0.542
 sj       (Intercept) 0.257714
 Residual             0.400315
```

Alternate model 4

- but we might've also decided to remove `verb_t`, so let's run that model

```
fit_verb_fp_m4 <- lmer(log(fp) ~ verb_t*gramm +
                       (1 |sj) +
                       (1 + gramm|item),
                     data = df_biondo,
                     subset = roi == 4,
                     control = lmerControl(optimizer = "bobyqa",
                                           optCtrl = list(maxfun = 2e5))
)
```

```
boundary (singular) fit: see help('isSingular')
```

- does not converge, so we're justified in keeping by-item `verb_t` slopes

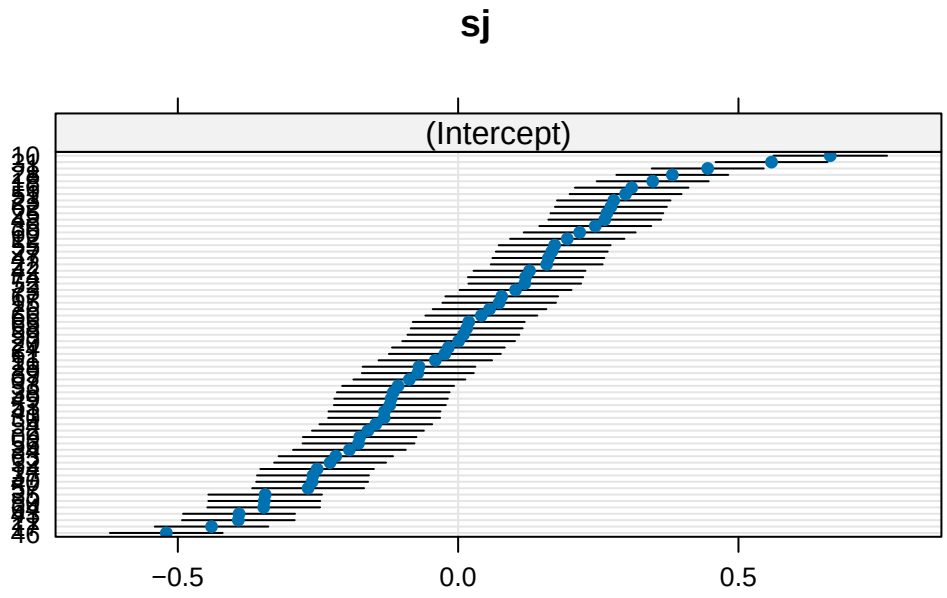
Final model

- the final model name should be some sort of convention to make your life easier
  - so remove model index

```
fit_verb_fp <- fit_verb_fp_m3
```

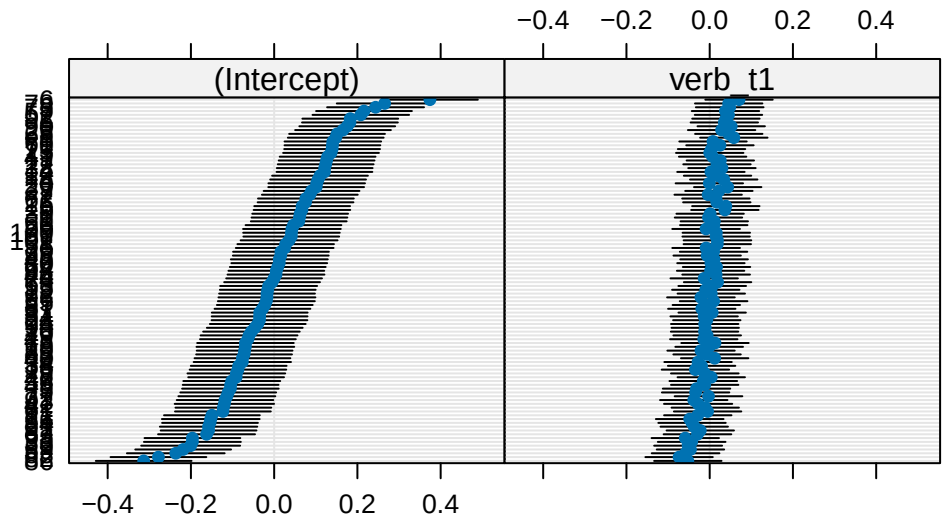
by-item random effects

```
lattice::dotplot(ranef(fit_verb_fp))$sj
```



by-participant random effects

```
lattice::dotplot(ranef(fit_verb_fp))$item
```



summary()

```
  summary(fit_verb_fp)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: log(fp) ~ verb_t * gramm + (1 | sj) + (1 + verb_t | item)
   Data: df_biondo
Control: lmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 200000))
 Subset: roi == 4

REML criterion at convergence: 4216.2

Scaled residuals:
    Min      1Q  Median      3Q     Max
-4.1758 -0.6096 -0.0227  0.6060  4.0568

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 item     (Intercept) 0.019423 0.13936
          verb_t1     0.002513 0.05013  0.54
 sj       (Intercept) 0.066417 0.25771
 Residual             0.160252 0.40032
Number of obs: 3795, groups:  item, 96; sj, 60
```

```
Fixed effects:
                  Estimate  Std. Error          df t value             Pr(>|t|)
(Intercept)       5.956384    0.036763   79.243183 162.021 < 0.0000000000000002
verb_t1           0.061733    0.013971   93.410602   4.419            0.0000267
gramm1            0.003298    0.012999 3544.451823   0.254                 0.80
verb_t1:gramm1   -0.014380    0.025998 3544.762347  -0.553                 0.58

(Intercept)    ***
verb_t1        ***
gramm1
verb_t1:gramm1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
            (Intr) vrb_t1 gramm1
verb_t1      0.077
gramm1       0.000 -0.002
vrb_t1:grm1  0.000  0.002  0.000
```

- IMPORTANTLY, only look at the fixed effects after you've got your final model!!!!
  - i.e., run model -> convergence error -> `rePCA()` + `VarCorr()` -> run model -> … -> converges -> only NOW run `summary(model)`

Comparing to 'bad' models

- let's compare our final model to our 'bad' models
  - random intercepts-only model (overconfident)
  - maximal model (underconfident)

Random-intercepts only

```
fit_verb_fp_intercepts <- lmer(log(fp) ~ verb_t*gramm +
                       (1 |sj) +
                       (1 |item),
                     data = df_biondo,
                     subset = roi == 4
)
```