Replication Code For Simulation Studies

Manuel Koller

Abstract

Instructions how to replicate several simulation studies made to study properties of the robust scoring equations estimator (RSE). All the required code and the simulation results are included. The code is written in a way that makes sure that the stored simulation results are still reproduced by the current code.

Keywords: robust statistics, mixed-effects model, hierarchical model, ANOVA, R, crossed, random effect, simulation study.

1. Introduction

This is a technical document that should make it easy for others to replicate the results of the simulation studies presented in Koller (2013) and Koller and Stahel (2022). For the sake of brevity, the full descriptions of the studies are not duplicated here. Please refer to the cited document. The same goes for the code itself. Instead of printing long listings here that are not very useful, **robustlmm** provides a function that opens the the corresponding script in the user's editor. All code is implemented in R (R Core Team 2022).

In the next section, we give an overview of the included simulation studies and how to access the code. In Section 3 we will cover the general structure of the code that is used to run each simulation study. Section 4 shows a list of R packages that were used in the simulation studies' code. The charts for each simulation study are then shown in turn. Simulation studies that have not been published elsewhere are described in detail while others are just referenced. Section 5 contains the results for the Sensitivity curves simulation study, consistency and efficiency for varying tuning parameters of the ψ-functions in Section 6, breakdown in Section 7, convergence for increasing number of observations in Section 9, and, robustness and empirical test coverage in Section 10. Finally, Section 12 shows the output of `sessionInfo()` that lists all versions of packages that were loaded when the simulation studies were run.

2. The code

The code for each simulation study is shipped as script file with **robustlmm**. To see it, run a variation of following command in R or click the script name in this document to open the file on github.

```
R> robustlmm::viewCopyOfSimulationStudy("sensitivityCurves.R")
```

This will create a copy of the `sensitivityCurves.R` script in the local working directory and open the script for editing and running.

As CRAN does not allow built package archives to be more than 5 MB in size, the version of the package on CRAN does not contain the full simulation results. Instead, only aggregated simulation results are included. A version of the package with the full results is available on github. You can install said version using the following command.

```
R> remotes::install_github("kollerma/robustlmm", "full-results")
```

The following simulation studies are provided:

- Section 5: `sensitivityCurves.R` reproduces Figure 4.1 in Koller (2013) and Figure 1.1 in Koller and Stahel (2022).
- Section 6: `consistencyAndEfficiencyDiagonal.R` reproduces Figure 4.2, Figure 4.3 and Figure 4.4 in Koller (2013).
- Section 6.2: `consistencyAndEfficiencyBlockDiagonal.R` a previously unpublished simulation study, the analogue of the previous script but for the block diagonal case.
- Section 9: `convergence.R` a previously unpublished simulation study, in which the effect of increasing sample size is studied.
- Section 7: `breakdown.R` reproduces Figure 4.5 in Koller (2013).
- Section 8: `breakdownMC.R` a Monte-Carlo extension of the breakdown study that compares the new RANSAC and redescending-ψ wrappers under several contamination strategies per design.
- Section 10: `robustnessDiagonal.R` replicates the simulation study in Section 4.2 in Koller and Stahel (2022) (Figure 2 and Figure 3)
- Section 10.2: `robustnessBlockDiagonal.R` replicates the simulation study in Section 4.4 in Koller and Stahel (2022) (Figure 4 and Figure 5).

- Section 11: the studies backing the methods new in the 3.5.0 release — `inferenceStudy.R` (Section 11.1), `leverageBreakdown.R` and `leverageGES.R` (Section 11.2), `mallowsEfficiency.R` and `mallowsEfficiencyDim.R` (Section 11.3), `ransacBasin.R` and `ransacConsensus.R` (Section 11.4), and `structuredCovariances.R` (Section 11.5).

`consistencyAndEfficiencyBlockDiagonal.R`, `convergence.R` and `breakdownMC.R` have not been published elsewhere, so a description of the study and a discussion are included in this document.

3. General structure of code for each presented study

All the simulation studies presented here follow the same pattern, consisting, broadly speaking, of the following four steps:

3.1 Generation of datasets which will be fitted using various methods,

3.2 fitting each dataset using each method and extracting all relevant information from the fitted models,

3.3 preparing one or more datasets of results suitable for plotting and finally,

3.4 plotting the results.

The **robustlmm** R package provides a set of methods that cover all of these steps and make it easy to run a simulation study. Full documentation is available in the help for each function. The simplest way to get started is probably by working through one of the scripts that are provided in **robustlmm**. Each of those scripts is discussed in Section 5 and later.

We will now go though each of the four steps and provide a few pointers how to use the provided functions.

3.1. Generation of datasets

The methods provided in **robustlmm** assume that the datasets come in a specific format. Besides the datasets themselves, one also has to provide all the information that needs to be passed to each of the fitting functions, e.g., the formula that specifies the model.

There are four functions available in **robustlmm**:

1. `createDatasetsFromList` converts a list of generated datasets into the required format.

2. `generateAnovaDatasets` create a ANOVA type balanced dataset with a variety of fixed and random grouping variables.

3. `generateMixedEffectDatasets` creates datasets using parameterized bootstrap off a base dataset that was prepared using `prepareMixedEffectDataset`.

4. `generateSensitivityCurveDatasets` creates datasets where one or more observations are changed more and more away from the original values in order to show how sensitive a method is to changes in the data.

All the functions produce a list that contains all the ingredients used later on. The resulting list contains functions that can produce a dataset for a given dataset index and other functions that are useful to combine multiple datasets together.

`createDatasetsFromList` is the most flexible way of creating a dataset list, the only limitation is that one can specify only one formula per dataset list. The trade off for the flexibility is that `createDatasetsFromList` is not as memory efficient as the two `generateAnovaDatasets` and `generateMixedEffectDatasets` functions.

Both `generateAnovaDatasets` and `generateMixedEffectDatasets` store as little redundant information as possible and only generate the full dataset on demand.

3.2. Fitting each dataset and extracting the relevant information

After the datasets have been simulated / generated, the next step is to apply each of the methods to each of the datasets. Then results need to be extracted into a memory efficient form and stored on disk for later re-use.

For each method that should be used for fitting the datasets, one has to provide a function that takes the dataset list produced in the previous step as the first argument. **robustlmm** comes with many pre-defined methods, see `?fitDatasets`. These functions are essentially wrappers that prepare some arguments needed for the fitting function and then call `lapplyDatasets` which does the actual work.

The fitted objects are preferably not stored on disk as these are usually very large when saved. To reduce the amount of disk space required, the fitted objects are passed through `processFit`. `processFit` is a S3 generic function, so it should be easy to add additional implementations for packages that are not supported by **robustlmm** already.

Except for debugging, it is usually not needed to call one of the `fitDatasets` and `processFit` functions directly. This is done either by `processDatasetsInParallel` or `processFile`.

For smaller simulation studies that can be run on a single machine, as the ones presented in this vignette, the function `processDatasetsInParallel` does all the work. The

datasets are split into chunks and then run in parallel using `parLapply` of the **parallel** package (R Core Team 2022).

Larger simulation studies or more computationally expensive methods may require to be run on a compute server. This is possible, but requires a bit more work from the user. First, one has to use `saveDatasets` to split the full list of datasets into chunks. For each of the chunks of datasets, a file is created. Second, these files need to be distributed to each of the computing machines. Third, there the files are processed using `processFile`. Fourth and finally, the collected results are merged using `loadAndMergePartialResults`.

To ensure that the results stay valid over a long period of time (for example when R or packages are updated), one can repeat this step with `checkProcessed` set to true. Then the first dataset is fit using all the methods and the newly produced results are compared with the stored results. This is also useful to weed out hidden dependencies on the random number generator seed.

To satisfy even stricter hard disk space limits (such as when submitting a package to CRAN), the argument `createMinimalSaveFile` can be used. This creates a file with the processed results of only the first three generated datasets. This subset of results can be used to run the code for the next step.

3.3. Prepare results for plotting

The results are stored in a list with several matrices. Using `cbind` these can be converted into a data frame.

If the full results are not permanently stored and only the minimal results are kept (see last paragraph in previous section), this step will have to include some code to load the aggreated data from disk and verify that the data before aggreation hasn't changed. It is good practice to keep running the aggregation code on the partial results anyway. While this does not protect against all possible problems, this at least will make sure that the aggregation code is valid code and can be run.

3.4. Plotting the results

This step is entirely up to the user. This vignette uses **ggplot2** (Wickham 2016), but this is just the author's preference.

4. List of Functions From Other Packages

Several R packages were used to run the simulation studies presented here. Many functions from R 's base packages (R Core Team 2022) were used – they are not listed

here.

Models were fit using code from **robustlmm** Koller (2016), **lme4** (Bates, Mächler, Bolker, and Walker 2015), **robustvarComp** (Agostinelli and Yohai 2019), **heavy** (no longer on CRAN) (Osorio and F. 2019), **lqmm** (Geraci 2014). Support for `rlme` from **rlme** (no longer on CRAN) (Bilgic, Susmann, and McKean 2018) is also available, but it was not included in any simulation studies as all studied datasets were balanced.

The function `hubers` from **MASS** (Venables and Ripley 2002) is used to compute robust mean and scale estimates when aggregating simulation results. The function `geom_pointpath` from **ggh4x** (van den Brand 2021) is used to draw lines of type `b`. Colors are taken from palettes provided by **RColorBrewer** (Neuwirth 2022). The skewed t-distribution is implemented in **skewt** (King and Anderson 2021). We ensure file names are file system compatible using `path_sanitize` from **fs** (Hester, Wickham, and Csárdi 2021). The function `arrange` from **dplyr** (Wickham, François, Henry, and Müller 2022) is used to work around a bug in **robustvarComp**. The function `melt` from **reshape2** (Wickham 2007) is used to convert `data.frame` from long to wide format. The function `facet_grid2` from **ggh4x** (van den Brand 2021) is used to create faceted plots in grid format that all have different y axes.

5. Sensitivity Curves

This section reproduces the Figure 4.1 in Koller (2013) and Figure 1 in Koller and Stahel (2022). The plots have been defined in `sensitivityCurves.R`. Method *lme* corresponds to `fitDataset_lmer`, *RSEn* to `fitDatasets_rlmer_DAStau_noAdj` and *RSEa* to `fitDatasets_rlmer_DAStau`.

The plots are shown in Figure 1.

6. Consistency and Efficiency

6.1. Diagonal Case

This section reproduces Figure 4.2, Figure 4.3 and Figure 4.4 in Koller (2013). The plots have been defined in `consistencyAndEfficiencyDiagonal.R`. Method *lme* corresponds to `fitDataset_lmer`, *RSEn* to `fitDatasets_rlmer_DAStau_noAdj` and *RSEa* to `fitDatasets_rlmer_DAStau`. The tuning parameters are set as shown in Table 1.

The plots are shown in Figure 2 and Figure 3.

6.2. Block Diagonal Case

Table 1: Tuning parameters k used for the smoothed ψ-function, s is always set to 10, diagonal case.

`rho.e`	0.5	1.345	2	5
`rho.sigma.e` (RSEn)	0.5	1.345	2	5
`rho.sigma.e` (RSEa)	1.47	2.18	2.9	5.03
`rho.b`	0.5	1.345	2	5
`rho.sigma.b`	1.47	2.18	2.9	5.03

This section contains a previously unpublished simulation study. The study is the analogue of study presented in the previous section, but for a model with a block diagonal random effects covariance matrix. The study is based upon the Sleep Study dataset that is also used in Section 10.2. The Sleep Study dataset is available in **lme4** (Bates *et al.* 2015) and was originally published by Belenky, Wesensten, Thorne, Thomas, Sing, Redmond, Russo, and Balkin (2003).

The tuning parameters for the smoothed Huber ψ-function used in this study are as indicated in the charts for `rho.e`. For `rho.sigma.e`, the squared robustness weights are used (`psi2propII`). For `rho.sigma.b` the same function is used as for `rho.b`. The tuning parameters are set as shown in Table 2.

Table 2: Tuning parameters k used for the smoothed ψ-function, s is always set to 10, block diagonal case.

`rho.e`	0.5	1.345	2	5
`rho.sigma.e` (RSEn)	0.5	1.345	2	5
`rho.sigma.e` (RSEa)	1.47	2.18	2.9	5.03
`rho.b`	2.17	5.14	8.44	34.21
`rho.sigma.b`	2.17	5.14	8.44	34.21

The plots are shown in Figure 4 and Figure 5. The plots have been defined in `consistencyAndEfficiencyBlockDiagonal.R`.

Discussion

The results are similar to the diagonal case presented in the previous section. It is crucial though that the tuning parameters for `rho.b` and `rho.sigma.b` are increased, otherwise the methods are not consistent (as for $k = 0.5$ where even the larger tuning parameters are not sufficient). This is because squared Mahalanobis distances are used for blocks of dimension 2 and larger and not simple residuals or predicted random effects as in the one dimensional case. The expected values of the distances grows by the dimension of a block, so one has to adjust tuning parameters per block size.

7. Breakdown

This section reproduces Figure 4.5 in Koller (2013). The plots have been defined in `breakdown.R`. Method *lme* corresponds to `fitDataset_lmer`, *RSEn* to `fitDatasets_rlmer_DAStau_noAdj` and *RSEa* to `fitDatasets_rlmer_DAStau`.

The plot is shown in Figure 6.

8. Monte-Carlo Breakdown Sweep

This section presents a Monte-Carlo extension of the breakdown study from Section 7. While `breakdown.R` shows a single trajectory under extreme observation-level contamination ($|y_i| \cdot 10^6$), the study here repeats the breakdown experiment many times under several distinct *contamination strategies* and compares the additional fitting methods provided by the wrappers `fitDatasets_rlmer_DAStau_bisq`, `fitDatasets_rlmer_DAStau_sizeOBR`, `fitDatasets_rlmer_ransac` and `fitDatasets_rlmer_ransac_bisq` against the classical and default robust baselines.

8.1. Setup

Three default **robustlmm** designs are used:

- `Dyestuff` (one-way, 6 batches $\times 5$ replicates)
- `Penicillin` (crossed, 24 plates $\times 6$ samples)
- `sleepstudy` (block-diagonal, 18 subjects $\times 10$ days, with random intercept and slope)

For each design, several contamination strategies that differ in *which* observations or groups are corrupted are tested (Table 3). The contamination intensity $\epsilon \in \{0, 0.05, 0.10, 0.15, 0.20, 0.30,$ is the fraction of observations (or, for RE-level strategies, the fraction of groups) that are contaminated. The contamination shifts the corresponding y values by $10 \cdot \sigma_e$ (where σ_e is the true residual scale, taken from the canonical `lmer` fit on the uncontaminated data).

For each (design, strategy, method, ϵ) combination, 30 random datasets are generated by parametric bootstrap from the canonical `lmer` fit, contaminated, and re-fit. A fit is "broken" on indicator A if any of the following indicator components hold:

- *beta*: $|\hat{\beta}_j - \beta_j| > 5 \cdot \mathrm{SE}_{\mathrm{lmer}}(\hat{\beta}_j)$ for some j
- *sigma_e*: $\hat{\sigma}_e / \sigma_e \notin [1/5, 5]$

- *sigma_b*: $\hat{\sigma}_{b,k}/\sigma_{b,k} \notin [1/5, 5]$ for some k
- *rho* (sleepstudy only): $|\hat{\rho}| > 0.99$

"Any indicator" is the OR of those four. The breakdown rate is the fraction of replicates for which the indicator is true.

Table 3: Contamination strategies per design.

Design	Strategy	Description
Dyestuff	`within`	Saturate batch 1, then 2, ...
	`round_robin`	One obs per batch in turn
	`random_obs`	Uniform random obs
	`re_shift`	Shift all obs of $\lceil \epsilon \cdot 6 \rceil$ batches
Penicillin	`within_plate`, `within_sample`	Within-plate (resp. sample) saturation
	`round_robin_plate`, `round_robin_sample`	Round-robin across plates (resp. samples)
	`random_obs`	Uniform random obs
	`re_shift_plate`, `re_shift_sample`	RE-level shift on plate (resp. sample) intercepts
sleepstudy	`within_subject`	Saturate subject 1, then 2, ...
	`round_robin`	One obs per subject in turn
	`high_leverage`	Highest-`Days` obs first
	`random_obs`	Uniform random obs
	`re_shift`	Shift all obs of $\lceil \epsilon \cdot 18 \rceil$ subjects
	`re_shift_slope`	Subject slope shift, centered on `Days`

8.2. Code

```
Loaded cached results: 21840 rows from datasets_breakdownMC-results.Rdata
```

9. Convergence

This section contains the details on a simulation study we ran to answer a referee's comment about the convergence of the estimator for increasing sample sizes. The plots have been defined in `convergence.R`. Method *lme* corresponds to `fitDataset_lmer`, *RSEn* to `fitDatasets_rlmer_DAStau_noAdj` and *RSEa* to `fitDatasets_rlmer_DAStau`.

9.1. Setup

We simulated a balanced dataset with varying number of subjects and replicates per subject. For both, we varied the numbers between 5, 10, 20 and 50. We simulated all

16 combinations. The model we used is

$$Y_{hi} = \beta_0 + \beta_{\text{continuous}} x_{\text{continuous}} + \beta_{\text{binary}} x_{\text{binary}} + B_h + \varepsilon_{hi}$$

where $x_{\text{continuous}}$ is a continuous variable generated by a uniform distribution between 0 and 1, x_{binary} is a binary variable that takes 0 and 1 with equal probability, $\varepsilon_{hi} \sim \mathcal{N}(0, \sigma^2)$ and $B_h \sim \mathcal{N}(0, \sigma_b^2)$, $h = 1, \ldots, \text{nSubject}$ and $i = 1, \ldots, \text{nReplicates}$.

The other simulation settings are chosen as described in sections 1.4 and 1.4.2 of the manuscript. We show summary statistics (location and scale) computed by using Huber's Proposal 2 using `hubers` as provided by the `MASS` R package (Venables and Ripley 2002). We also compute the empirical efficiencies by dividing the scale of the classical estimator by the scale of the robust estimator.

9.2. Discussion

The simulation results are shown in Figure 10, Figure 11 and Figure 12 for the N/N case and Figure 13, Figure 14 and Figure 15 for the t3/t3 case.

Except for small differences in the case with just 5 subjects, both the classic estimator and our proposed method in the manuscript behave in the same way. Our method is tuned for 95% asymptotic efficiency at the central (N/N) model. The tuning works as expected and is independent of the number of subjects or replicates within subjects, as can be seen in the Figure 11. The empirical efficiency shown in Figure 12 reiterates this point. The RSEa method has been tuned for 95% asymptotic efficiency, and this efficiency is surpassed most of the time. Only in the 5 subjects / 5 replicates case the empirical efficiency drops to 80%. The empirical efficiency of RSEn is lower, as it is tuned for higher robustness for the scale estimates.

The charts for the t3/t3 case shown in Figure 13, Figure 14 and Figure 15 also show expected behavior. Neither method shows a large bias for the betas as the contamination is symmetric. Our method shows less bias for $\hat{\sigma}$ and $\hat{\sigma}_b$ as expected for a robust method. The empirical efficiency is higher for our method throughout, except for one the 5 subjects / 5 replicates case where the empirical efficiency for $\hat{\sigma}_b$ comes out slightly lower. The higher robustness for the scale estimates pays off with a higher empirical efficiency for RSEn than RSEa.

10. Robustness and Empirical Test Coverage

10.1. Diagonal Case

This section replicates the simulation study in Section 4.2 in Koller and Stahel (2022) (Figure 2 and Figure 3). The plots have been defined in `robustnessDiagonal.R`. Method

lme corresponds to `fitDataset_lmer`, *RSEn* to `fitDatasets_rlmer_DAStau_noAdj` and *RSEa* to `fitDatasets_rlmer_DAStau`.

The plots are shown in Figure 16, Figure 17.

10.2. Block Diagonal Case

This section replicates the simulation study in Section 4.4 in Koller and Stahel (2022) (Figure 4 and Figure 5). The plots have been defined in `robustnessBlockDiagonal.R`.

The plots are shown in Figure 18, Figure 19.

Figure 1: Sensitivity curves for a balanced one-way dataset with 10 groups of 5 observations. *lme* is the classical estimator, *RSEn* is the RSE estimator for the smoothed Huber ψ-function where the tuning parameter k for `rho.sigma.e` is the same as for `rho.e` ($k = 1.345$), similar for `rho.sigma.b` and `rho.b`. *RSEa* is the same as RSEn, but the tuning parameter for `rho.sigma.e` and `rho.sigma.b` are both adjusted ($k = 2.28$). Plots from top to bottom correspond to `plot_shiftFirstObservation`, `plot_shiftFirstGroup`, and `plot_scaleFirstGroup`.

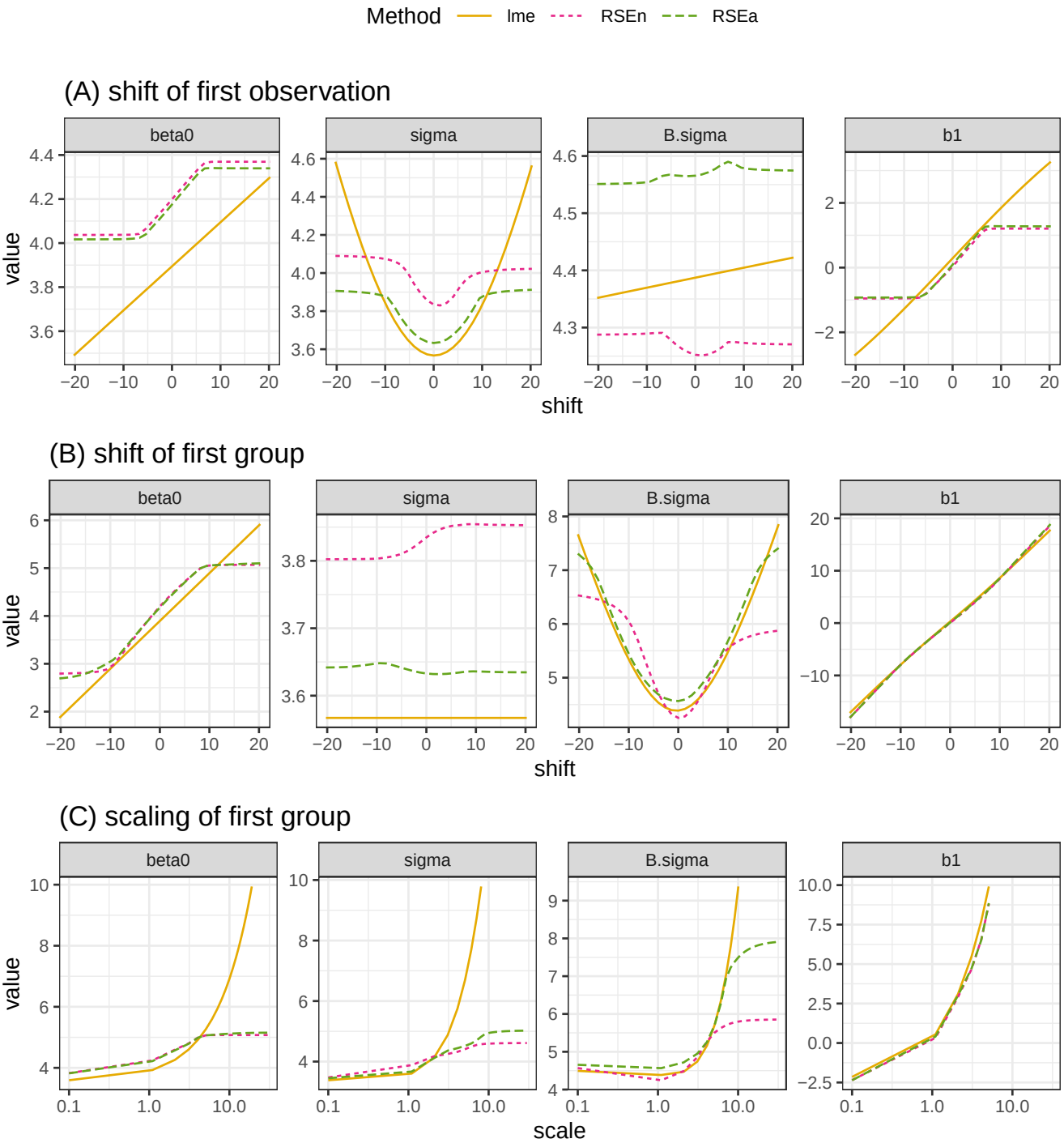


Figure 2: Mean values and quartiles of 1000 fits of randomly generated, balanced one-way designs with 20 groups and 20 observations per group. The tuning parameters for `rho.e` are shown on the horizontal axis, the corresponding tuning parameters for the other functions are shown in Table 1. The black line indicates the true values. The yellow line shows the classical fit (solid: mean, dashed: quartiles). Plot corresponds to `plot_consistencyDiagonal`.

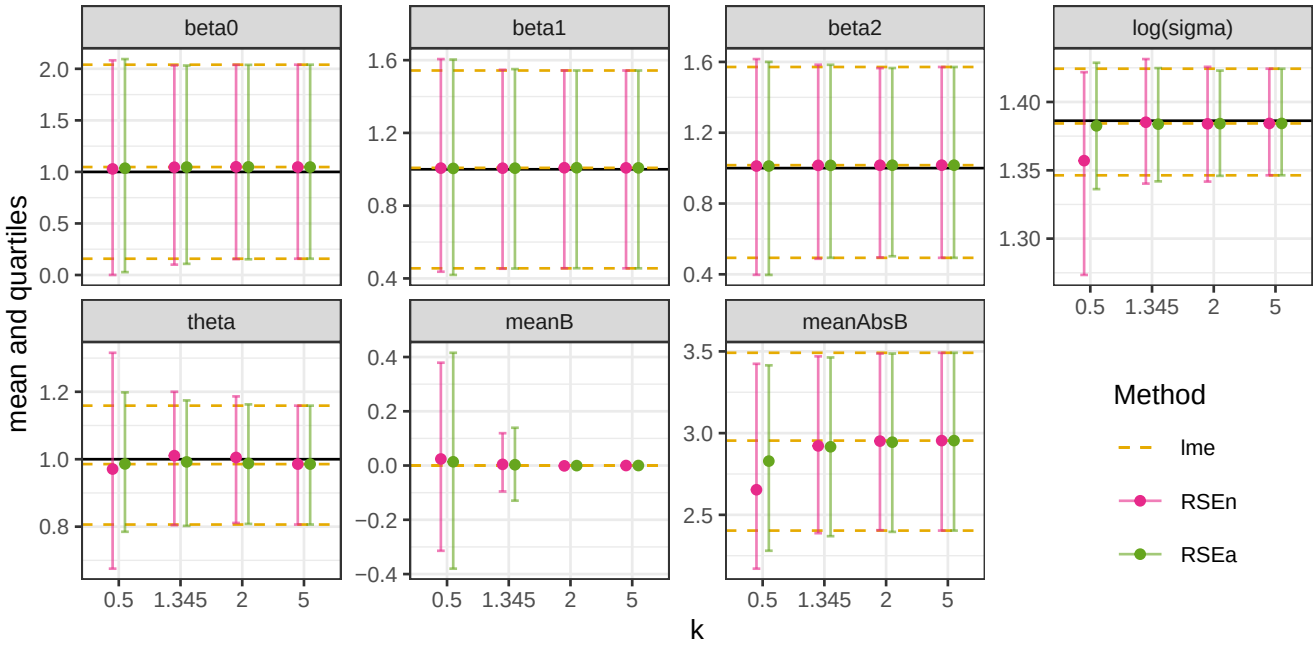


Figure 3: For the same setup as shown in Figure 2, the empirical efficiencies were computed. The efficiency was computed on the bias corrected estimates. The parameters are shown in facet columns, while the rows correspond to the same choice of ψ-functions, both are indicated in the facet strip. Plot corresponds to `plot_efficiencyDiagonal`.

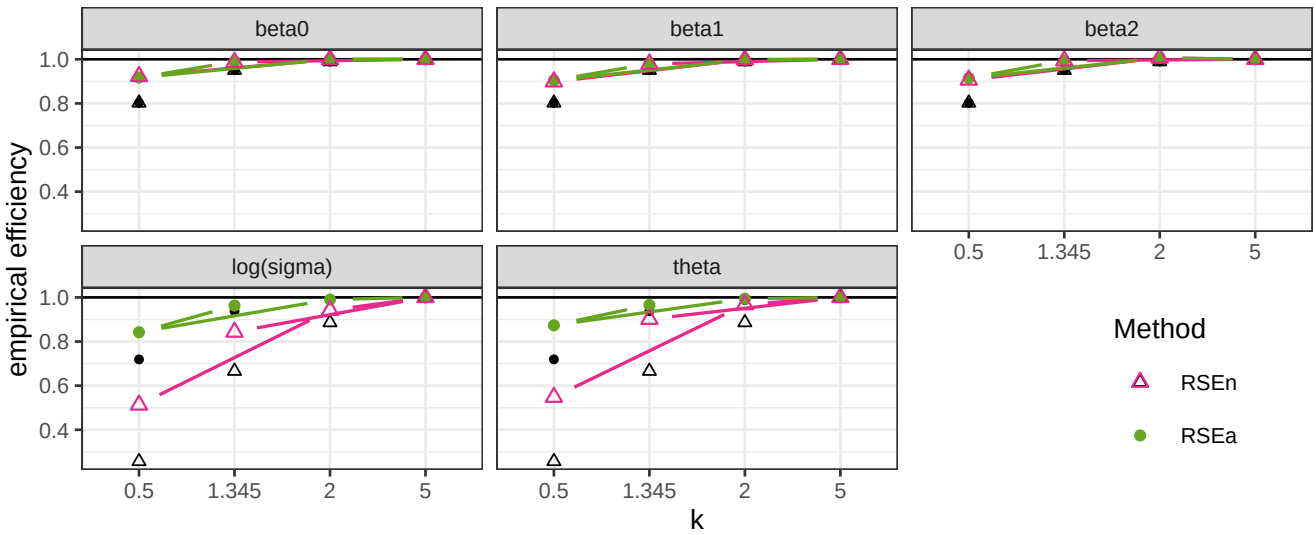


Figure 4: Mean values and quartiles of 1000 fits of parametric bootstrap samples of the Sleep Study dataset. The tuning parameters for `rho.e` are shown on the horizontal axis, the corresponding tuning parameters for the other functions are shown in Table 2. The black line indicates the true values. The yellow line shows the classical fit (solid: mean, dashed: quartiles). Plot corresponds to `plot_consistencyBlockDiagonal`.

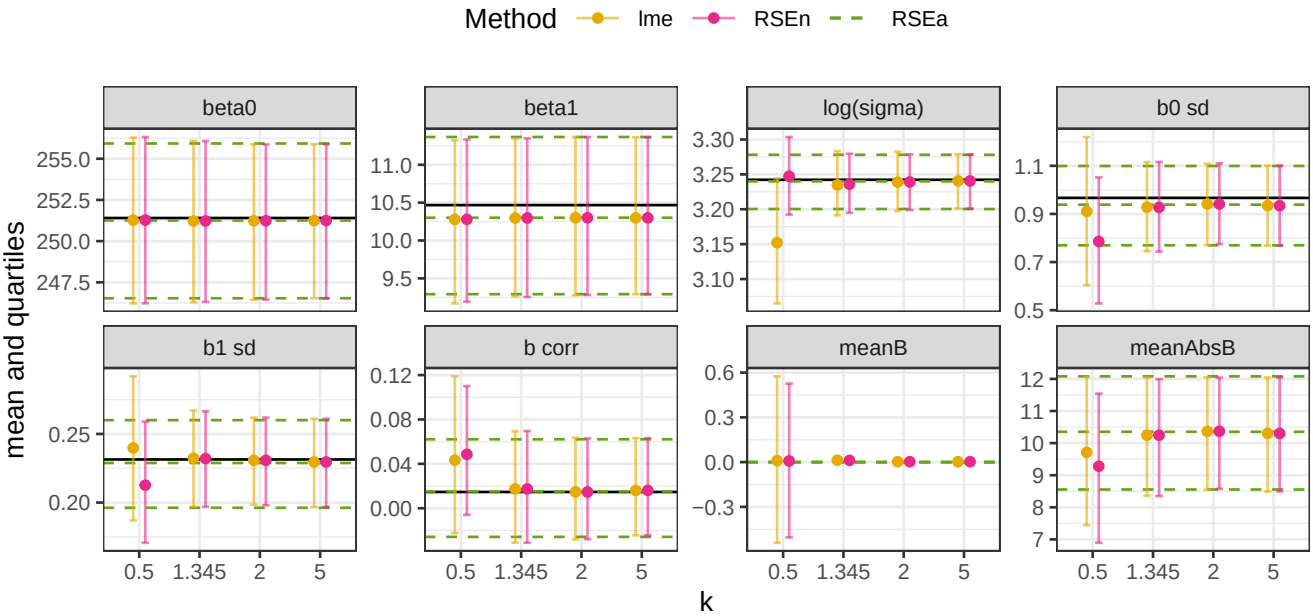


Figure 5: For the same setup as shown in Figure 4, the empirical efficiencies were computed. The efficiency was computed on the bias corrected estimates. The parameters are shown in facet columns, while the rows correspond to the same choice of ψ-functions, both are indicated in the facet strip. Plot corresponds to `plot_efficiencyBlockDiagonal`.

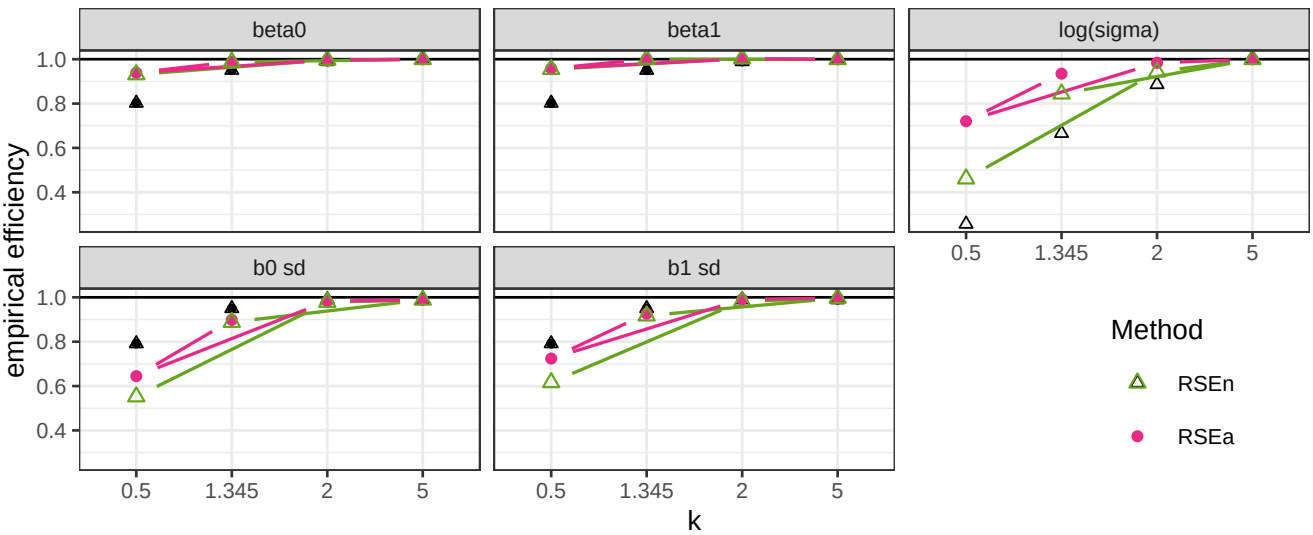


Figure 6: Example of breakdown for a one-way random model with 20 groups and 5 observations per group. Group after group, one observation after another was replaced by its absolute value multiplied by 10^6. *lme* is the classical estimator, *RSEn* is the RSE estimator for the smoothed Huber ψ-function where the tuning parameter k for `rho.sigma.e` is the same as for `rho.e` ($k = 1.345$), similar for `rho.sigma.b` and `rho.b`. *RSEa* is the same as RSEn, but the tuning parameter for `rho.sigma.e` and `rho.sigma.b` are both adjusted ($k = 2.28$). Plot corresponds to `plot_breakdown`.

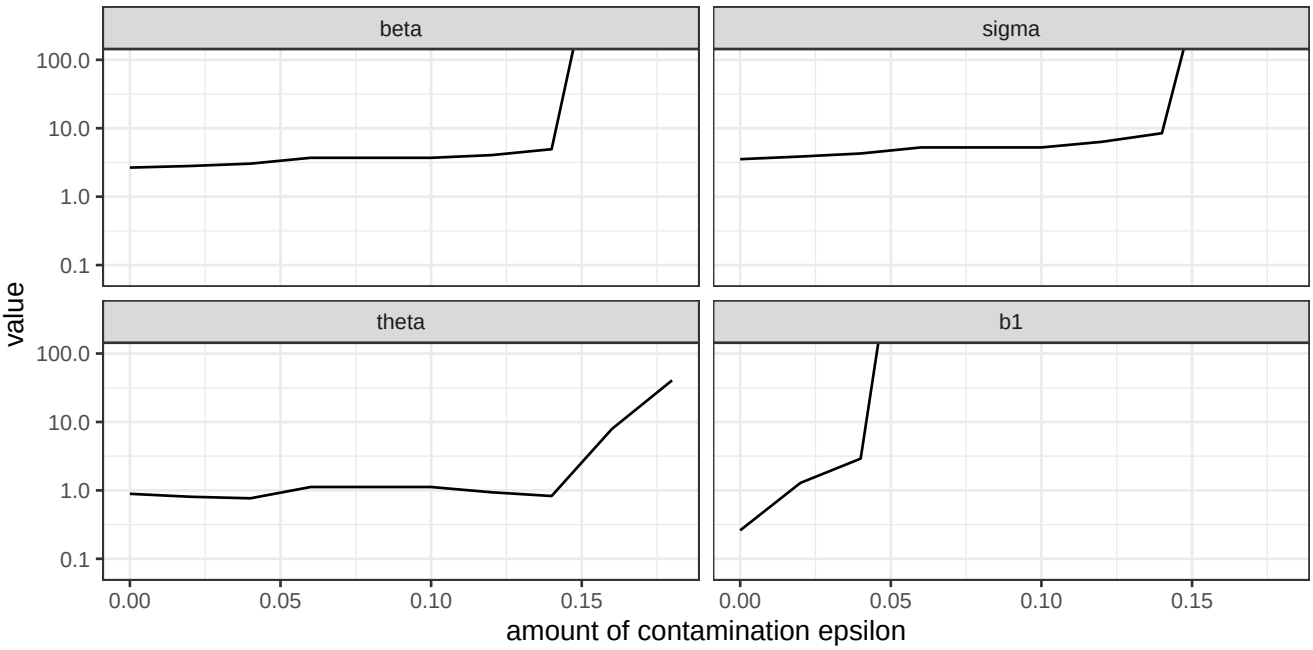


Figure 7: Breakdown rate (any indicator) for Dyestuff across the four contamination strategies, as a function of ϵ (fraction of contaminated observations or groups). 30 replicates per cell.

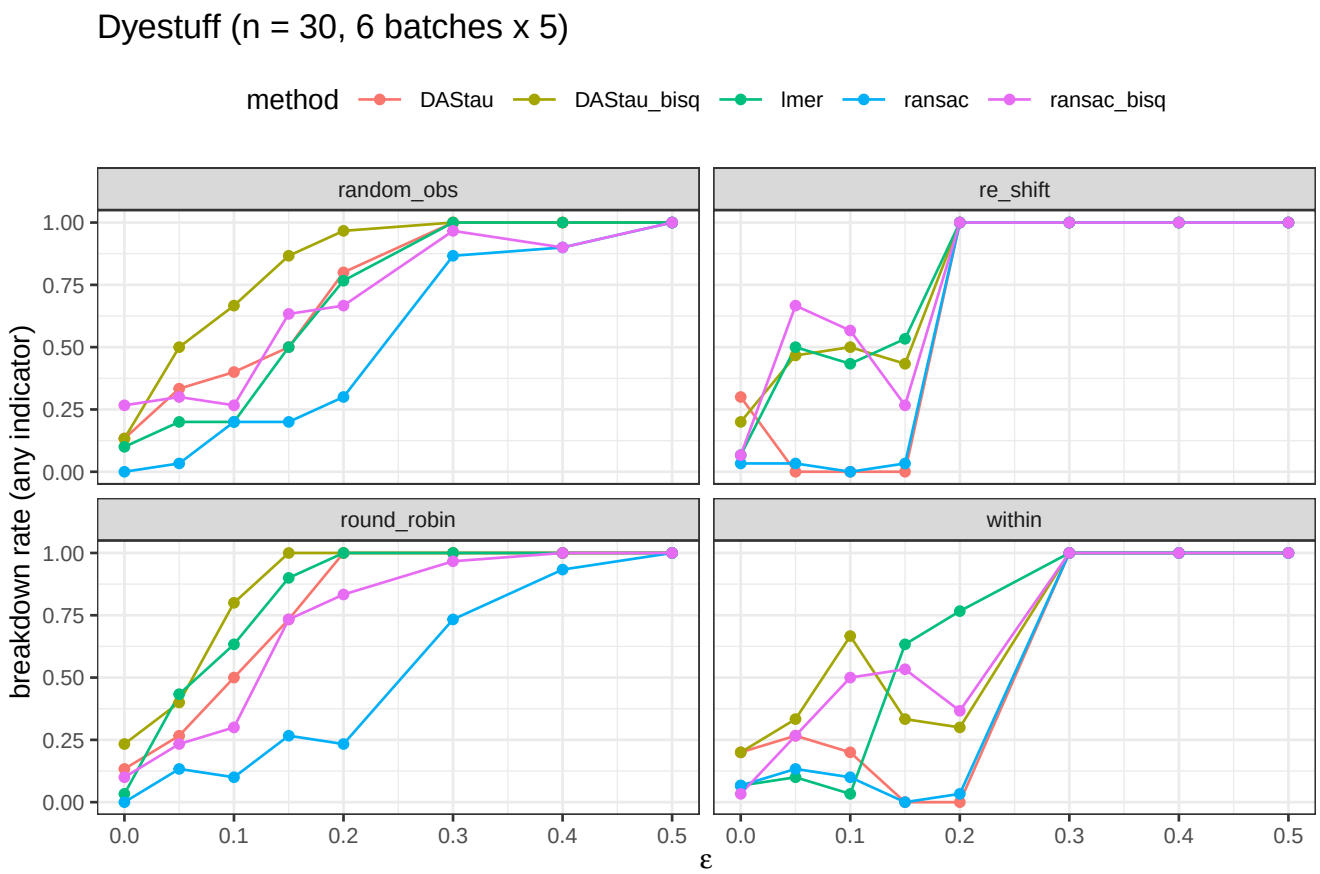


Figure 8: Breakdown rate for Penicillin (crossed random effects).

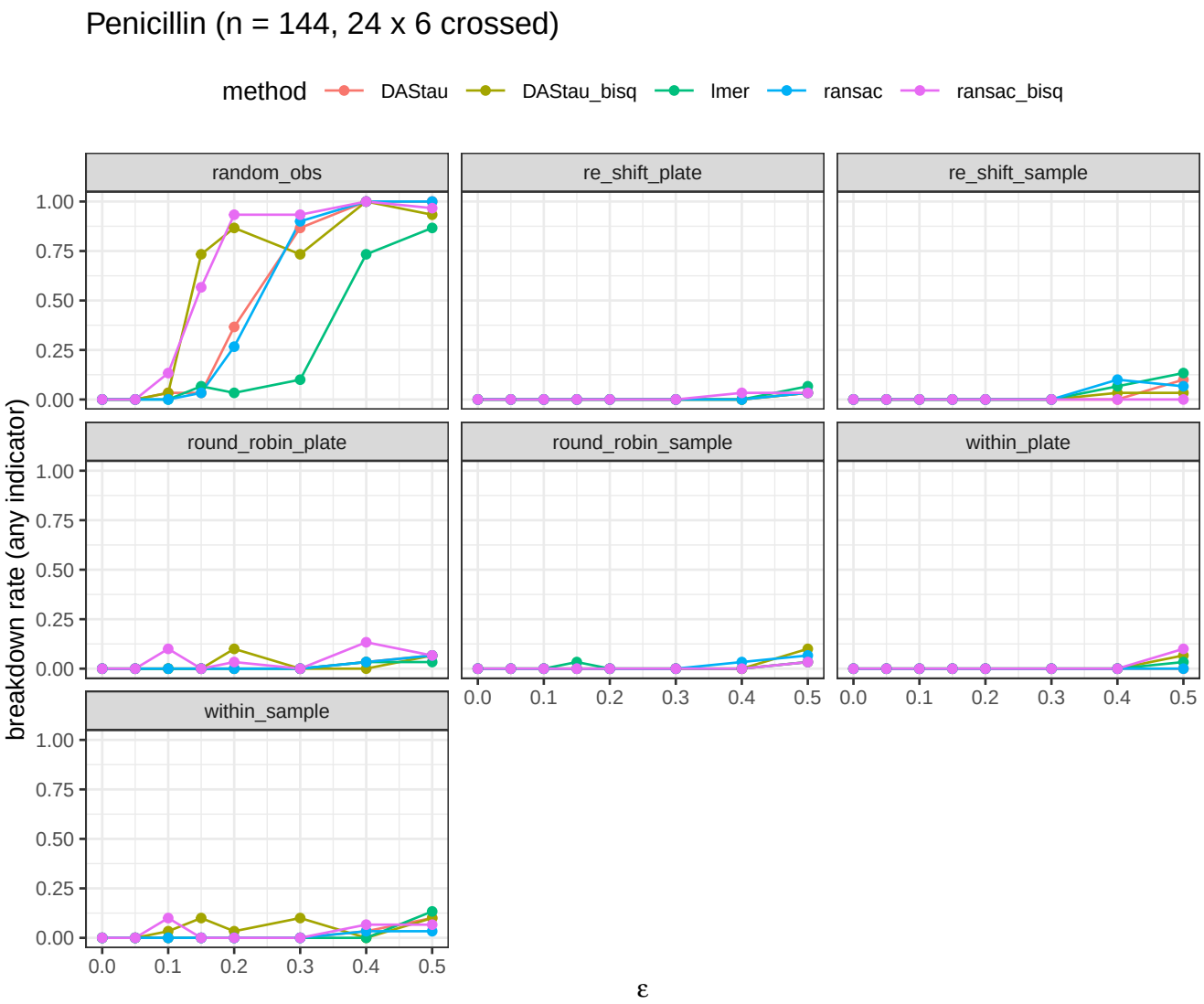


Figure 9: Breakdown rate for sleepstudy (block-diagonal random effects with correlation ρ).

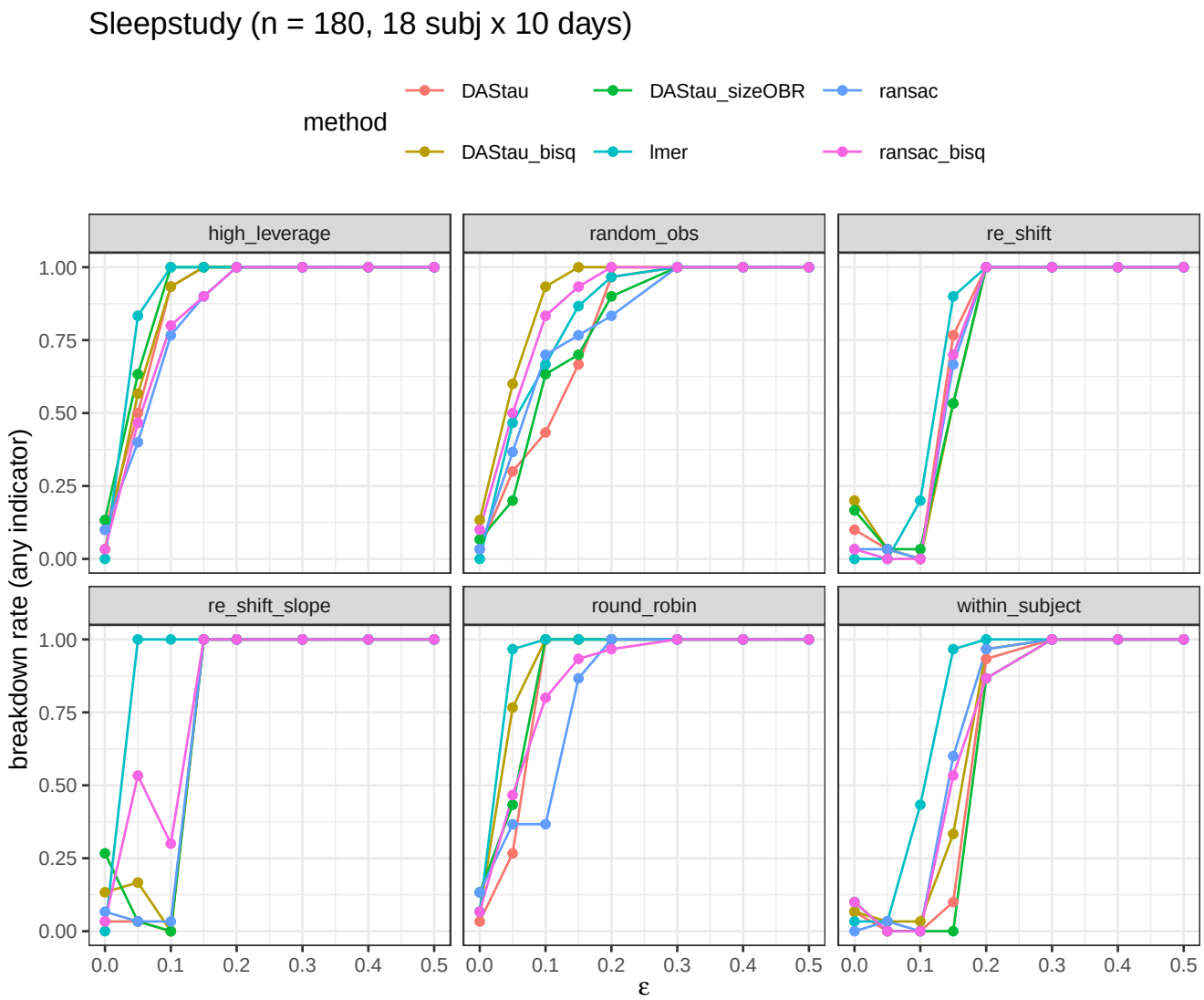


Figure 10: Convergence simulation study, N/N case, bias, estimated as the robust mean computed using Huber's Proposal 2 from 1000 fits. *lme* is the classical estimator, *RSEn* is the RSE estimator for the smoothed Huber ψ-function where the tuning parameter k for `rho.sigma.e` is the same as for `rho.e` ($k = 1.345$), similar for `rho.sigma.b` and `rho.b`. *RSEa* is the same as RSEn, but the tuning parameter for `rho.sigma.e` and `rho.sigma.b` are both adjusted ($k = 2.28$). Plot corresponds to `plot_convergence_N_N_bias`.

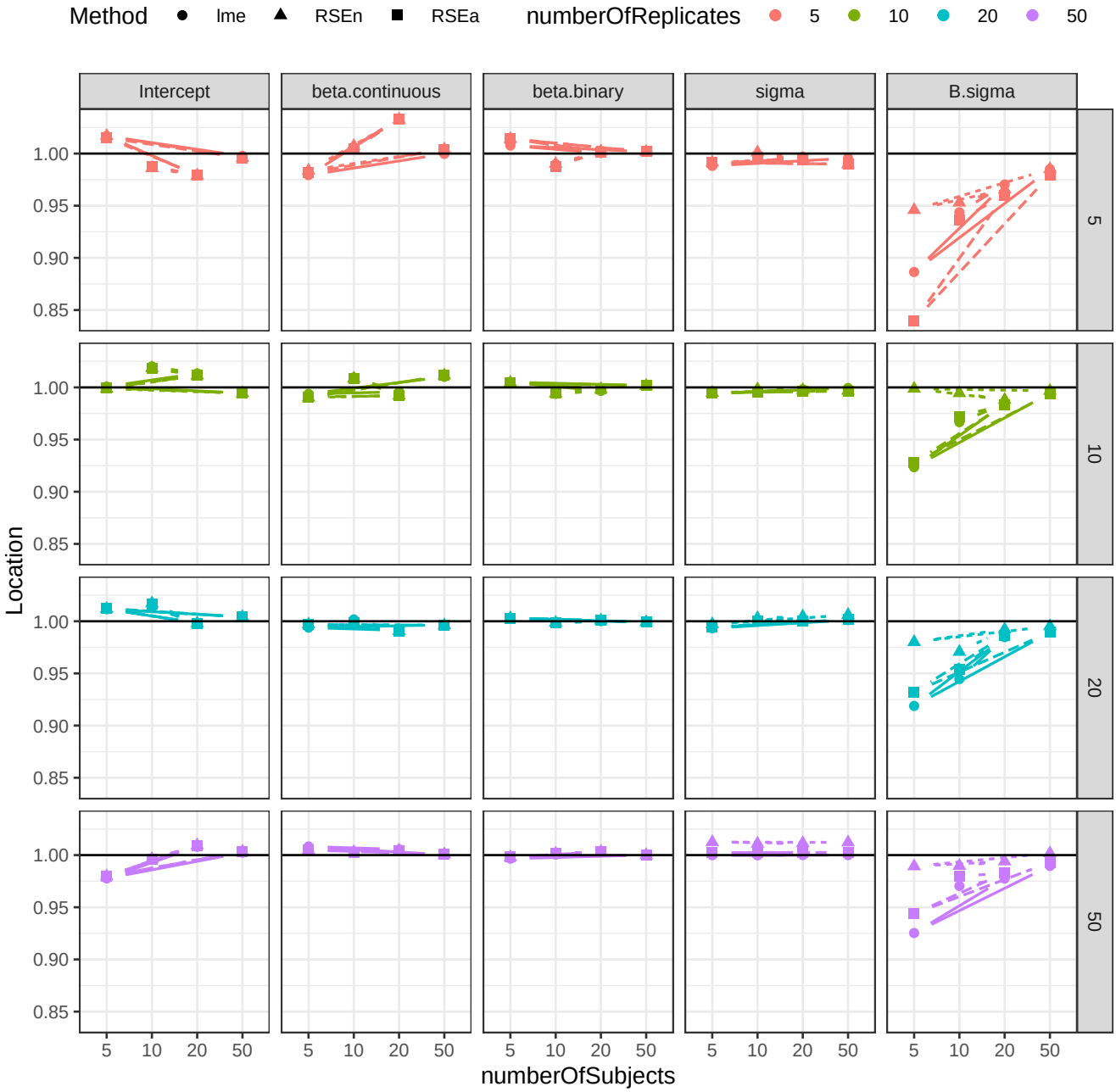


Figure 11: For the same setup as shown in Figure 10, but showing the robust scale. Plot corresponds to `plot_convergence_N_N_scale`.

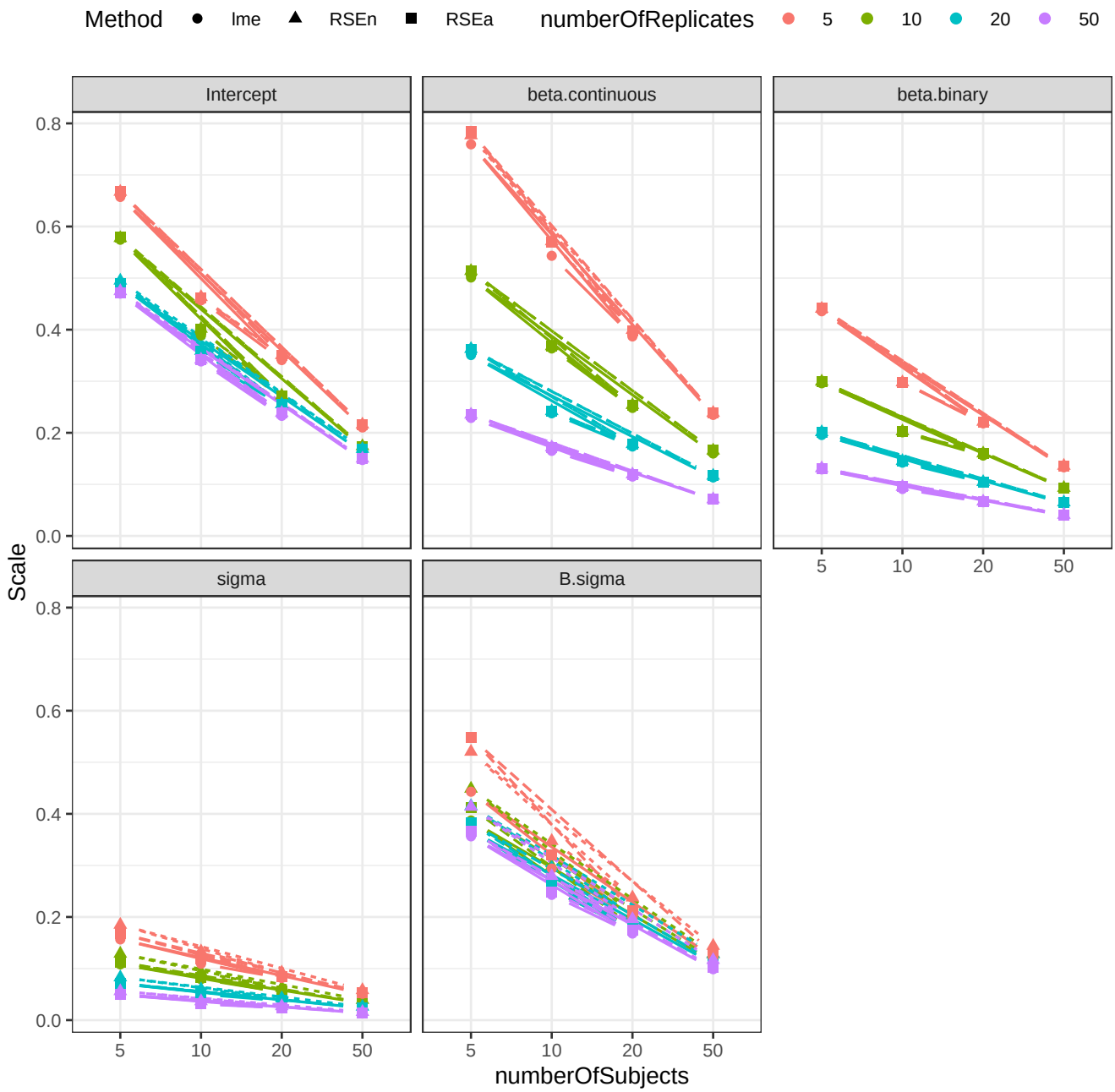


Figure 12: For the same setup as shown in Figure 10, but showing the empirical efficiency, computed by dividing the scale of the classical estimator by the one of the robust estimator. Plot corresponds to `plot_convergence_N_N_efficiency`.

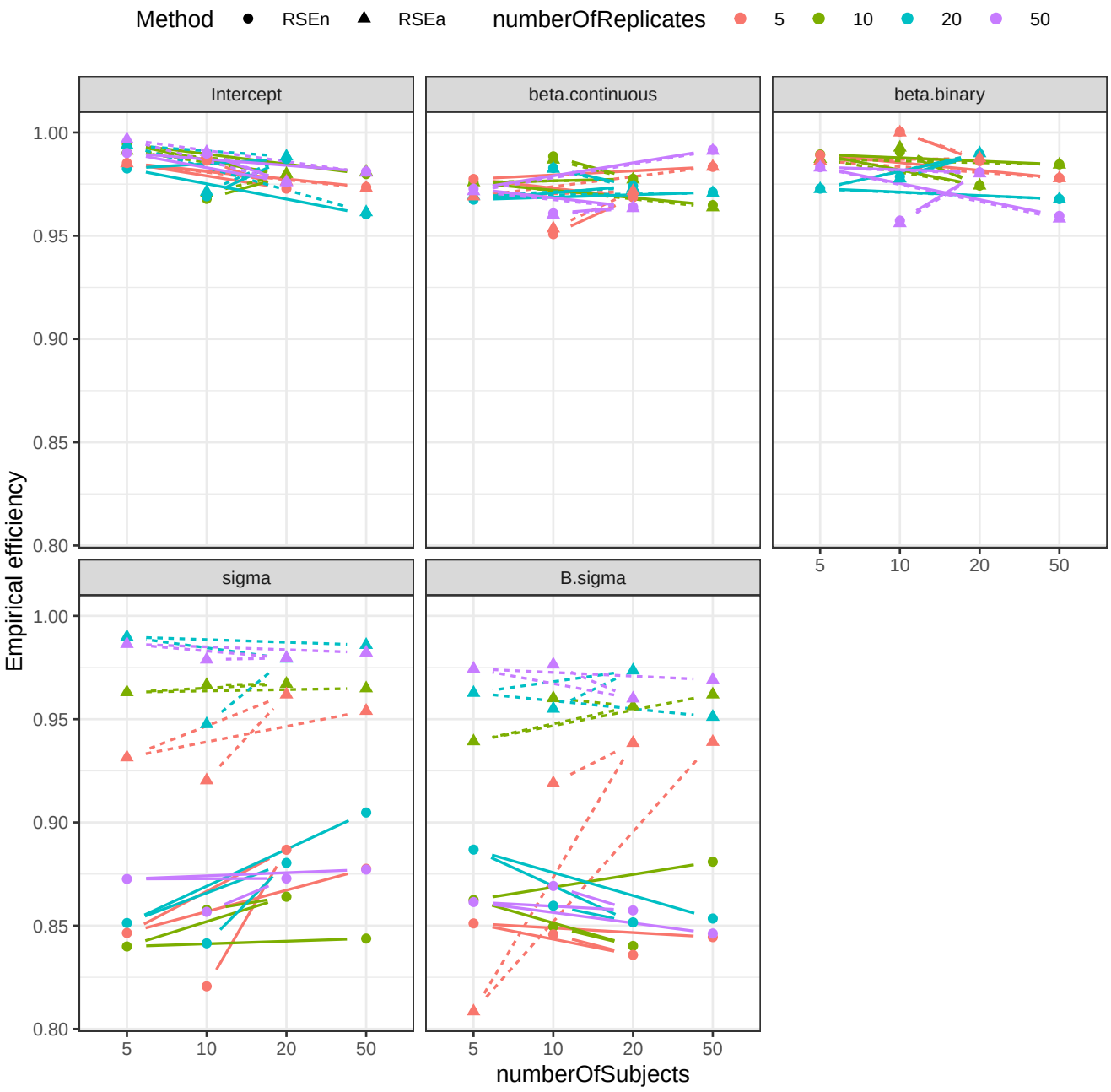


Figure 13: Convergence simulation study, t3/t3 case, bias, estimated as the robust mean computed using Huber's Proposal 2 from 1000 fits. *lme* is the classical estimator, *RSEn* is the RSE estimator for the smoothed Huber ψ-function where the tuning parameter k for `rho.sigma.e` is the same as for `rho.e` ($k = 1.345$), similar for `rho.sigma.b` and `rho.b`. *RSEa* is the same as RSEn, but the tuning parameter for `rho.sigma.e` and `rho.sigma.b` are both adjusted ($k = 2.28$). Plot corresponds to `plot_convergence_t3_t3_bias`.

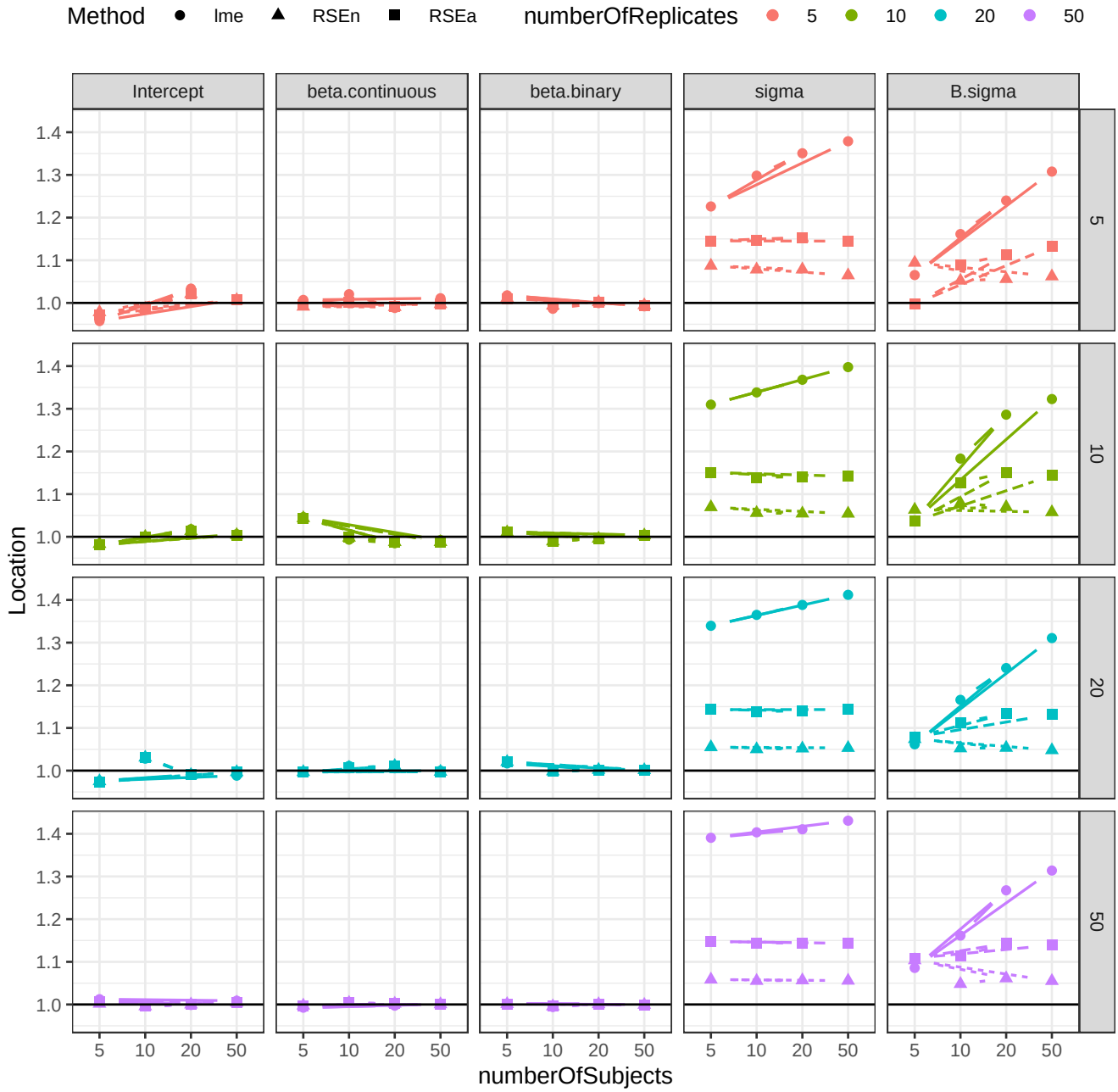


Figure 14: For the same setup as shown in Figure 13, but showing the robust scale. Plot corresponds to `plot_convergence_t3_t3_scale`.

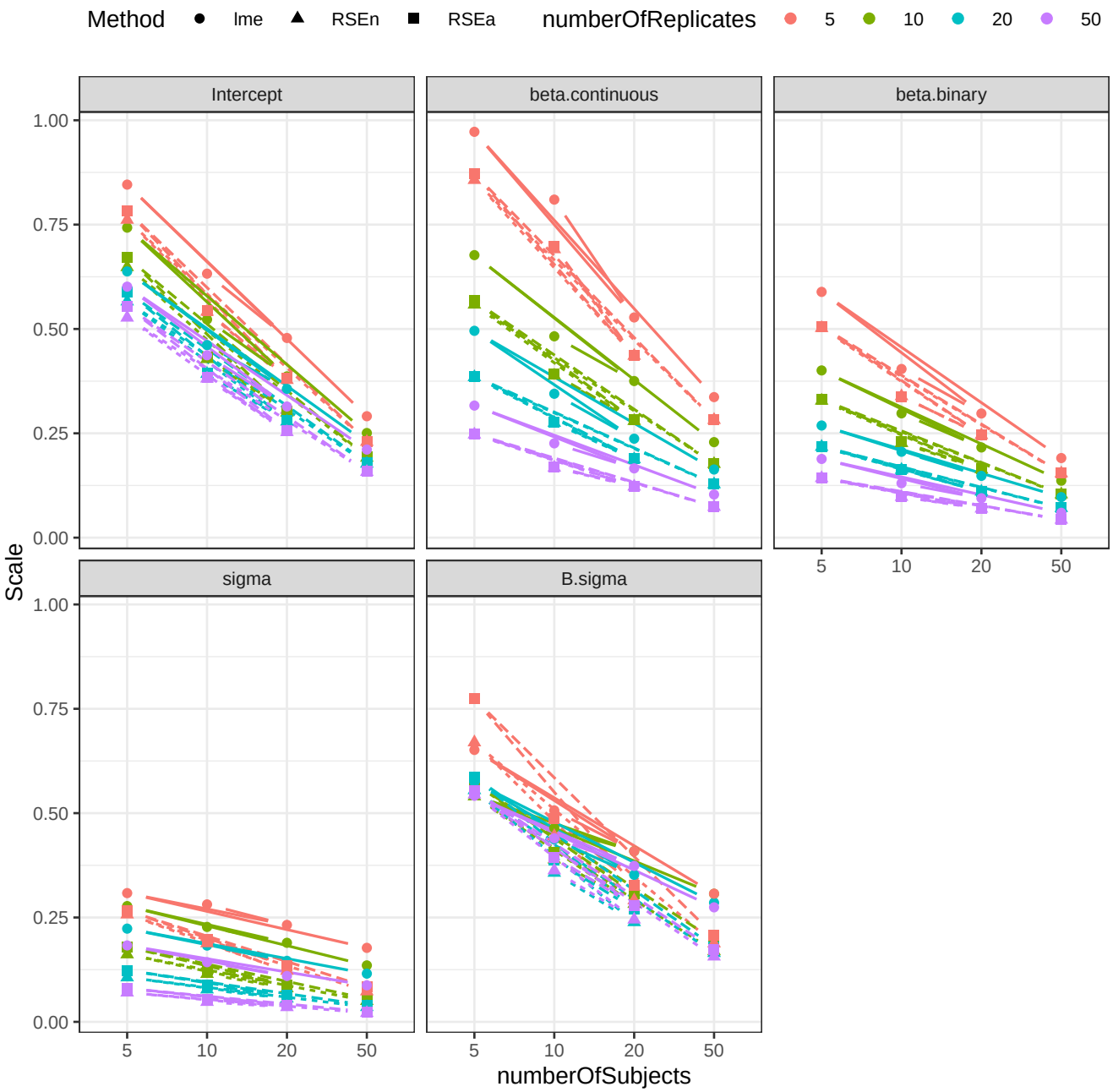


Figure 15: For the same setup as shown in Figure 13, but showing the empirical efficiency, computed by dividing the scale of the classical estimator by the one of the robust estimator. Plot corresponds to `plot_convergence_t3_t3_efficiency`.

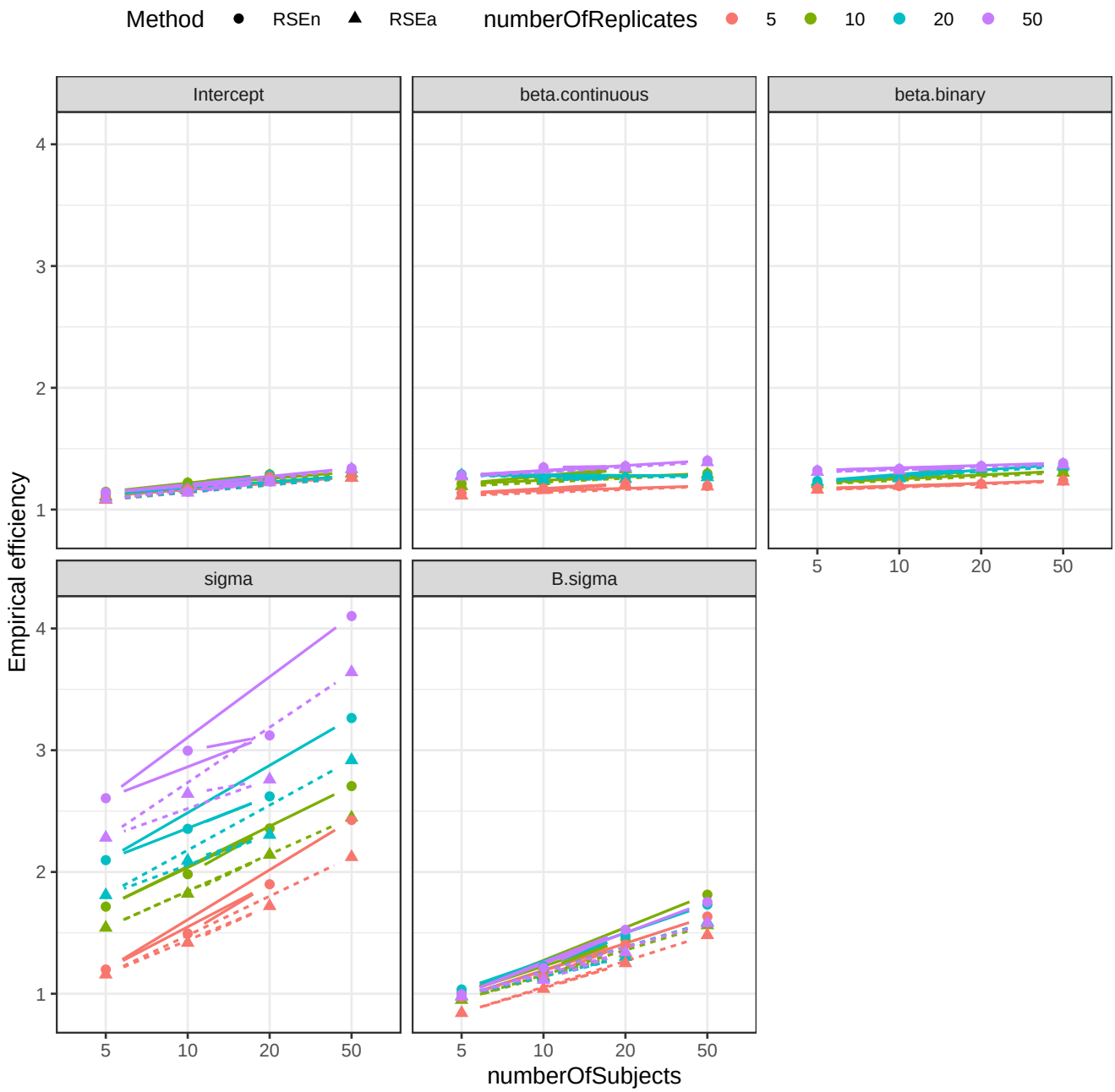


Figure 16: Simulation results for the diagonal case. The left column shows a robust location estimate of the simulated estimates for the diverse methods and the five parameters $\beta_0, \beta_1, \beta_2, \sigma, \sigma_b$. The true values are indicated by gray horizontal lines. Deviations from them thus represent biases. The right column shows robust scale parameters—measures of simulated standard errors—in an analogous way. Plot corresponds to `plot_robustnessDiagonal`.

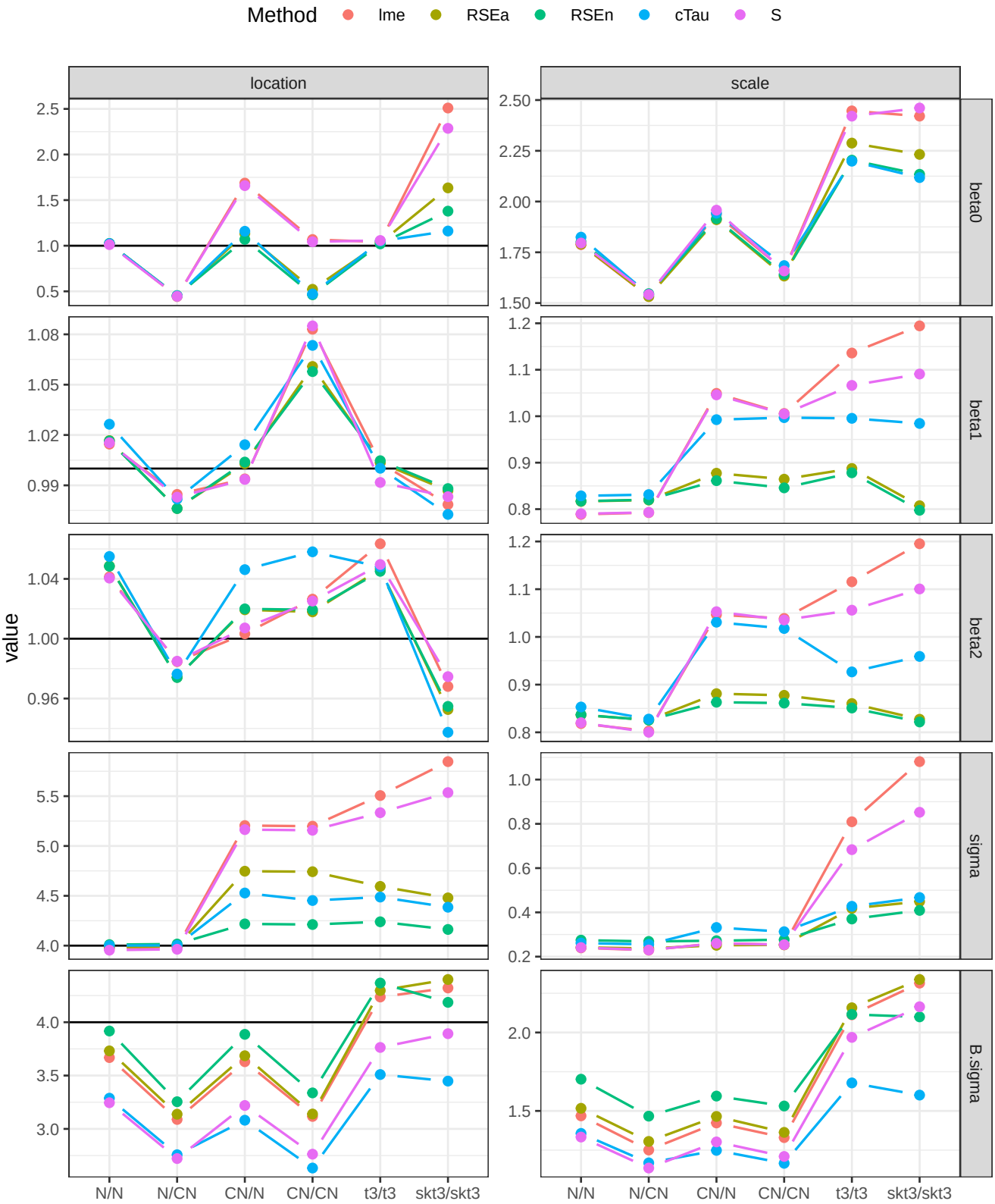


Figure 17: Simulation results for the diagonal case showing empirical coverage probabilities for the intercept β_0 (left), β_1 (middle) and β_2 (right). The expected level of 0.95 is shown by a gray line. Plot corresponds to `plot_coverageDiagonal`.

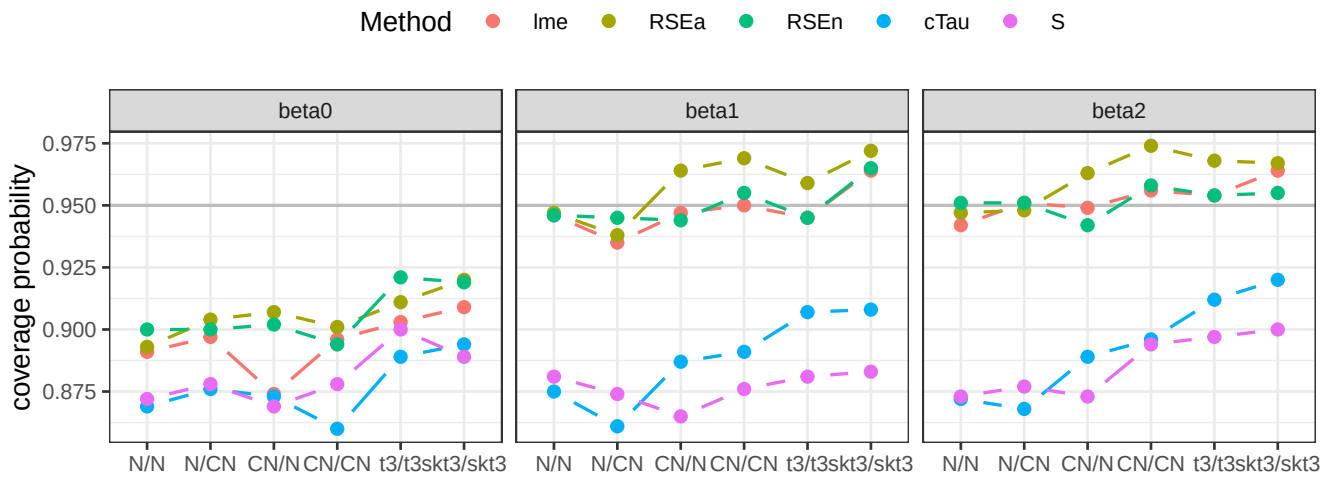


Figure 18: Simulation results for the block-diagonal case, shown as in Figure 16. Plot corresponds to `plot_robustnessBlockDiagonal`.

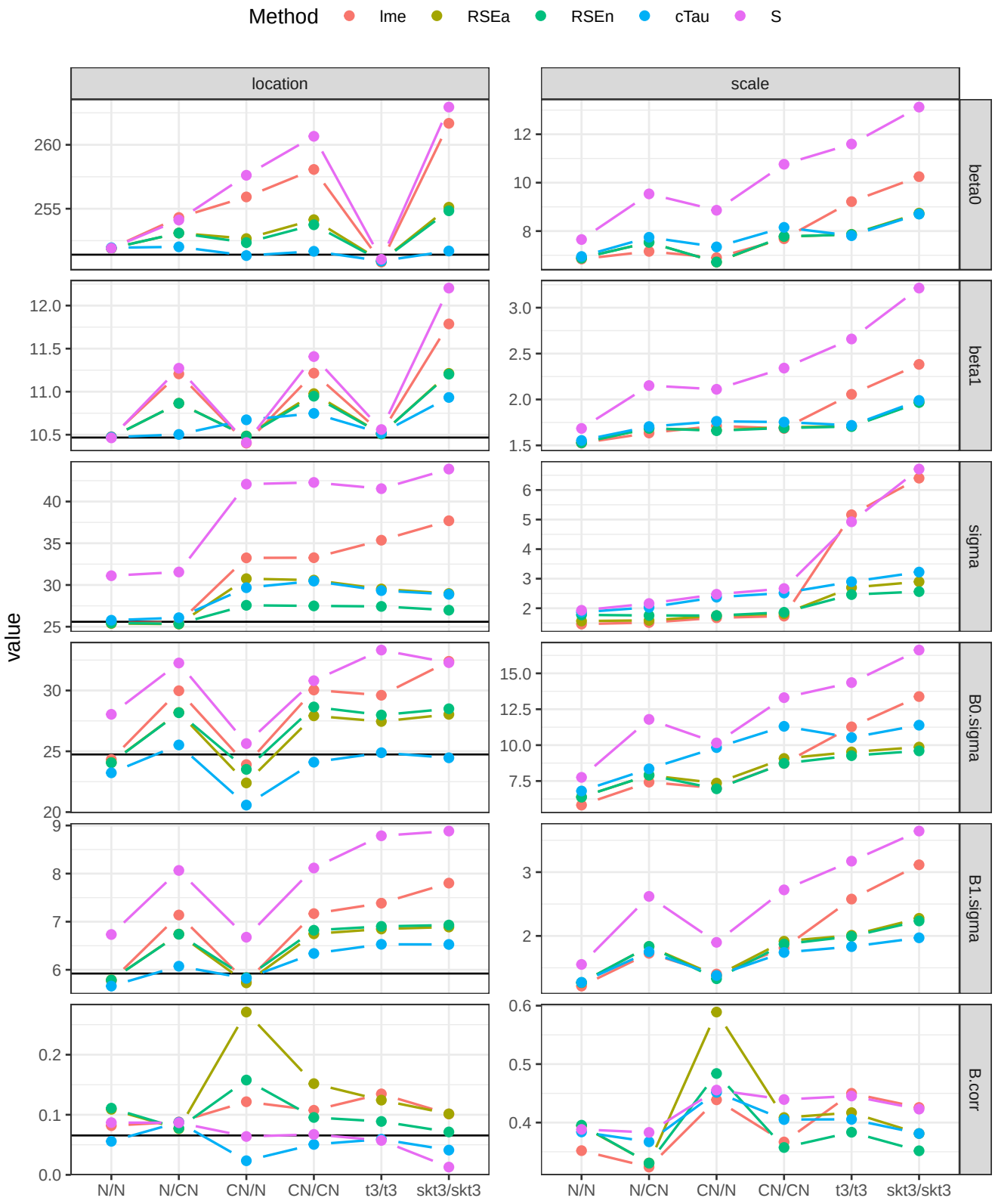
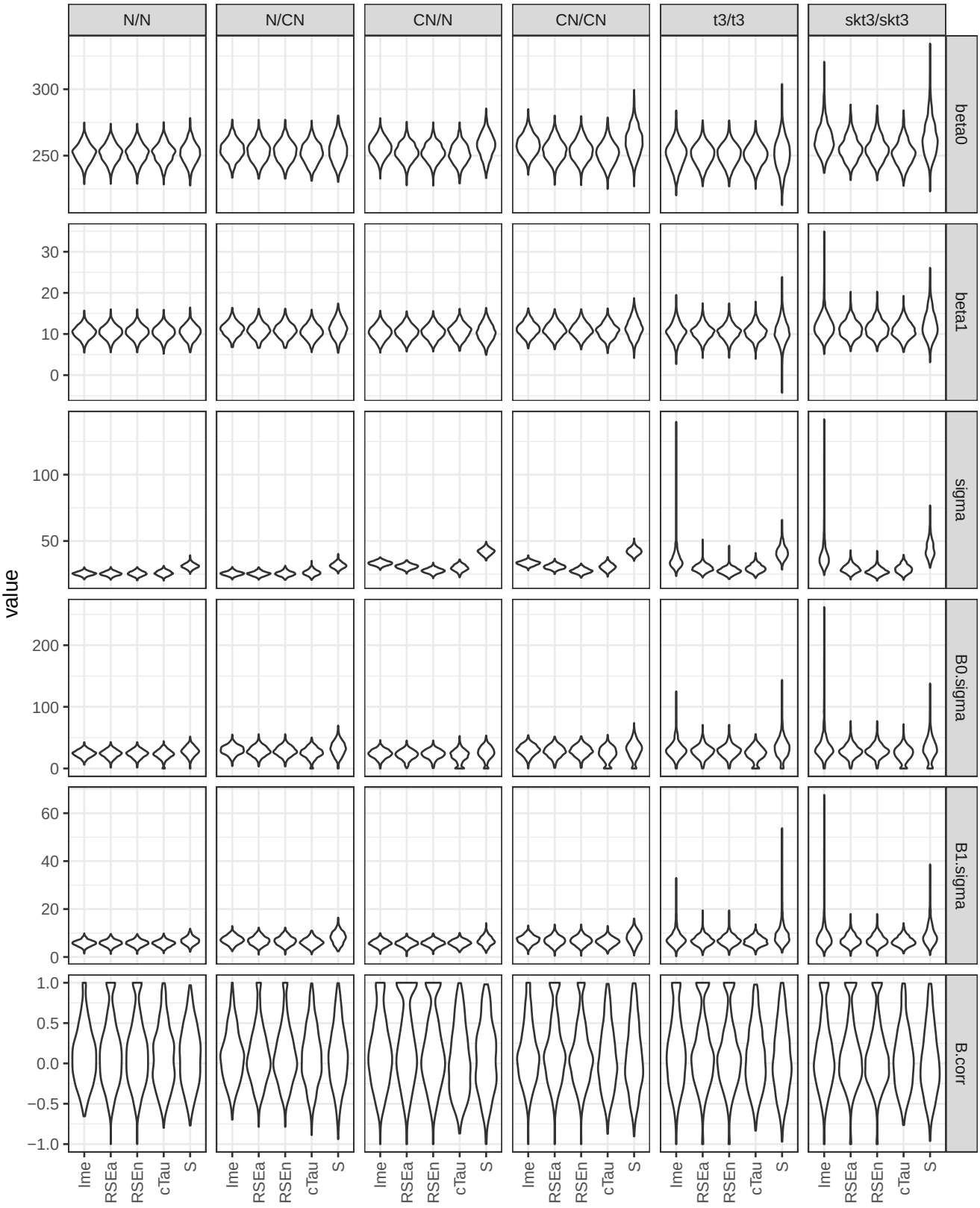


Figure 19: Simulated distribution of estimates in the block-diagonal case. The green horizontal line marks the true value. Plot corresponds to `plot_violinBlockDiagonal`.



11. Studies new in the 3.5.0 release

The 3.5.0 release adds a number of new methods (robust inference, influence diagnostics, a RANSAC initial estimator, Mallows design weights and structured random-effect covariances; see `vignette("robustlmm-3.5.0")`). Each is backed by one or more simulation studies. The scripts are available in the package sources on GitHub and are linked below; the worked, -reproducible headline numbers appear in the `robustlmm-3.5.0` vignette. This section describes each study's design and conclusions. Because the full result objects exceed the CRAN size limit, the figures are not rebuilt here; install the `full-results` version (Section 2) to regenerate them.

11.1. Inference: coverage, size and power

`inferenceStudy.R` evaluates the finite-sample behaviour of the new inference methods (the cluster-sandwich `vcov`, the Wald and bootstrap `confint`, and the robust `anova`). It sweeps the number of groups J and several error- and random-effect-contamination strategies, and records, for each method: the empirical coverage of the nominal 95% confidence intervals, the Type-I error of the tests at nominal 5%, and the power under a fixed-effect alternative.

Conclusions. On clean Gaussian data all methods are well-calibrated. Under error-distribution contamination the robust Wald intervals are markedly shorter than the classical ones at matched coverage, and the robust Wald `anova` is more powerful than the classical likelihood-ratio test while holding its size. The cluster-sandwich covariance is a large-J device: at small J it under-covers and the sandwich-based tests become anti-conservative, returning to nominal only for larger J; `vcov_sandwich` warns below 20 clusters, and the conclusions are distilled into the `?vcov_sandwich` and `?anova.rlmerMod` caveats. The variance-component bootstrap test (`anova(f0, f1, test = "boot")`) calibrates correctly under error contamination but is anti-conservative when whole groups are contaminated, which motivates the cluster-level influence diagnostics below.

11.2. Leverage breakdown and gross-error sensitivity

The ψ-functions of the RSE bound the influence of outliers in the response but not of high-leverage design points. `leverageBreakdown.R` contaminates a fraction of the observations by moving them onto a spurious regression line at a large robust distance in covariate space, and compares the slope bias of `lmer`, the plain RSE and the Mallows-weighted RSE (`design.weights = "mcd"`) as the contamination fraction grows. `leverageGES.R` complements it by tracing the gross-error sensitivity — the supremum of the (standardised) influence as a single design point is moved out along a covariate.

Conclusions. The plain RSE breaks down with `lmer` under leverage contamination (good-leverage outliers have small residuals, so the ψ-functions do not down-weight them), whereas the Mallows-weighted fit stays close to the clean-data estimate, with several-fold smaller slope bias at moderate contamination. The gross-error sensitivity of the plain RSE is unbounded in the leverage direction, while the Mallows weights make it bounded.

11.3. Efficiency cost of the Mallows weights

`mallowsEfficiency.R` quantifies the price of the leverage protection at the clean Gaussian model across the tuning grid ($\gamma \in \{1, 2\}$, cut-off $\in \{0.95, 0.975, 0.99\}$), reporting the relative efficiency of $\hat{\beta}$ against `lmer` (the MLE) and against the plain RSE. `mallowsEfficiencyDim.R` repeats this as the number of continuous covariates grows, to check that the χ^2-calibrated cut-off keeps the false-down-weighting rate roughly constant with dimension.

Conclusions. At the default tuning ($\gamma = 1$, cut-off 0.975) the clean-model cost is about 0.8% in the variance of $\hat{\beta}$ relative to the plain RSE (over 2000 replicates), with only about 2.8% of clean observations down-weighted; the cost is stable as the covariate dimension grows and vanishes for designs without continuous covariates. Only the most aggressive corner of the grid ($\gamma = 2$, cut-off 0.95) costs appreciably more, which is why the gentler default is shipped.

11.4. RANSAC: basin radius and multi-start consensus

`ransacBasin.R` separates the two failure modes the `rlmer_ransac` diagnostics watch for: a *fixed-effect basin* condition (a redescending `rho.e` only keeps the high-breakdown RANSAC start if the start lies within the bisquare support-preservation radius r^*) and a distinct *random-effects* attractor in which the fitted RE correlation collapses to a phony $|\hat{\rho}| \to 1$. `ransacConsensus.R` then evaluates the multi-start consensus (`rlmer_ransac(n_starts = m)`) under several error/RE contamination generators.

Conclusions. The basin radius is genuinely a fixed-effect property and the phony-correlation failure is a separate random-effects phenomenon, so the two are reported by separate diagnostics. Running `rlmer` from the several best distinct RANSAC starts and keeping the interior, lowest-residual-scale fit markedly reduces the rate of phony $|\hat{\rho}| \to 1$ solutions relative to a single start, at essentially unchanged slope accuracy — recovering the good solution when the single best start falls into the phony attractor.

11.5. Structured random-effect covariances

`structuredCovariances.R` studies the new `cs` and `ar1` covariance structures. Balanced

repeated-measures datasets (generated with `generateRepeatedMeasuresDatasets`) are fitted classically (`lmer`) and robustly (`rlmer`) under both clean data and random-effects-level contamination, and the recovered common correlation and marginal standard deviations are compared with the truth.

Conclusions. On clean data both fits recover the truth, so the projection-based fitting is consistent. Under contamination of 10% of the subjects the classical common correlation collapses toward zero (from a true 0.50 to 0.11 for `cs` and from 0.60 to 0.12 for `ar1`) and the marginal standard deviations inflate, whereas the robust structured fit stays near the truth (correlation 0.42 and 0.56). The `cs`/`ar1` fitting is experimental (see `vignette("robustlmm-3.5.0")`).

12. Session Info

- R version 4.2.1 (2022-06-23), `aarch64-apple-darwin20`
- Running under: `macOS Monterey 12.4`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: lme4 1.1-29, Matrix 1.4-1, robustlmm 3.0
- Loaded via a namespace (and not attached): boot 1.3-28, cellWise 2.2.6, cli 3.3.0, codetools 0.2-18, colorspace 2.0-3, compiler 4.2.1, crayon 1.5.1, DEoptimR 1.0-11, dplyr 1.0.9, ellipsis 0.3.2, emmeans 1.7.5, estimability 1.3, fansi 1.0.3, fastGHQuad 1.0.1, fit.models 0.64, generics 0.1.3, ggplot2 3.3.6, glue 1.6.2, grid 4.2.1, gridExtra 2.3, GSE 4.2, gtable 0.3.0, heavy 0.38.196, lattice 0.20-45, lifecycle 1.0.1, lqmm 1.5.8, magrittr 2.0.3, MASS 7.3-57, matrixStats 0.62.0, minqa 1.2.4, munsell 0.5.0, mvtnorm 1.1-3, nlme 3.1-158, nloptr 2.0.3, numDeriv 2016.8-1.1, parallel 4.2.1, pcaPP 2.0-1, pillar 1.7.0, pkgconfig 2.0.3, plyr 1.8.7, purrr 0.3.4, R6 2.5.1, Rcpp 1.0.8.3, reshape2 1.4.4, rlang 1.0.3, robust 0.7-0, robustbase 0.95-0, robustvarComp 0.1-5, rrcov 1.7-0, scales 1.2.0, SparseGrid 0.8.2, splines 4.2.1, stats4 4.2.1, stringi 1.7.6, stringr 1.4.0, svd 0.5.1, tibble 3.1.7, tidyselect 1.1.2, tools 4.2.1, utf8 1.2.2, vctrs 0.4.1, xtable 1.8-4

References

Agostinelli C, Yohai VJ (2019). *robustvarComp: Robust estimation for Variance Component Models*. R package version 0.1-5, URL https://CRAN.R-project.org/package=robustvarComp.

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

Belenky G, Wesensten NJ, Thorne DR, Thomas ML, Sing HC, Redmond DP, Russo MB, Balkin TJ (2003). "Patterns of Performance Degradation and Restoration During Sleep Restriction and Subsequent Recovery: A Sleep Dose-response Study." *Journal of Sleep Research*, **12**, 1–12.

Bilgic Y, Susmann H, McKean J (2018). *rlme: Rank-Based Estimation and Prediction in Random Effects Nested Models*. R package version 0.5, URL https://CRAN.R-project.org/package=rlme.

Geraci M (2014). "Linear Quantile Mixed Models: The lqmm Package for Laplace Quantile Regression." *Journal of Statistical Software*, **57**(13), 1–29. doi:10.18637/jss.v057.i13.

Hester J, Wickham H, Csárdi G (2021). *fs: Cross-Platform File System Operations Based on 'libuv'*. R package version 1.5.2, URL https://CRAN.R-project.org/package=fs.

King R, Anderson E (2021). *skewt: The Skewed Student-t Distribution*. R package version 1.0, URL https://CRAN.R-project.org/package=skewt.

Koller M (2013). "Robust Estimation of Linear Mixed Models." Diss., ETH Zürich, Nr. 20997, 2013, URL https://doi.org/10.3929/ethz-a-007632241.

Koller M (2016). "robustlmm: An R Package for Robust Estimation of Linear Mixed-Effects Models." *Journal of Statistical Software*, **75**(6), 1–24. doi:10.18637/jss.v075.i06.

Koller M, Stahel WA (2022). "Robust Estimation of General Linear Mixed Effects Models." In M Yi, K Nordhausen (eds.), *Robust and Multivariate Statistical Methods: Festschrift in Honor of David E. Tyler*, pp. 297–322. Springer, Cham.

Neuwirth E (2022). *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-3, URL https://CRAN.R-project.org/package=RColorBrewer.

Osorio, F (2019). *heavy: Robust estimation using heavy-tailed distributions*. R package version 0.38.196, URL https://CRAN.R-project.org/package=heavy.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

van den Brand T (2021). *ggh4x: Hacks for 'ggplot2'*. R package version 0.2.1, URL https://CRAN.R-project.org/package=ggh4x.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL https://www.stats.ox.ac.uk/pub/MASS4/.

Wickham H (2007). "Reshaping Data with the reshape Package." *Journal of Statistical Software*, **21**(12), 1–20. URL https://www.jstatsoft.org/v21/i12/.

Wickham H (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Wickham H, François R, Henry L, Müller K (2022). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.9, URL https://CRAN.R-project.org/package=dplyr.

Affiliation:

Manuel Koller
E-mail: kollerma@proton.me