

Robust Small Area Estimation: a Vignette

Tobias Schoch*

July 1, 2011

Contents

1	Introduction	2
2	Getting started	2
3	Setting up a model	3
3.1	Exploring the data	3
3.2	We set up our first model	5
4	(Robust) Estimation	6
4.1	Maximum likelihood estimation	6
4.1.1	When the ML estimator does not converge	7
4.1.2	When the mixed linear model is not appropriate	7
4.2	Huber-type M -estimation	9
4.2.1	Default mode	9
4.2.2	Safe mode	11
4.3	S -estimation	12
5	Robust prediction	12
6	Utility functions	15
6.1	Synthetic data generation	15
6.1.1	Uncontaminated data	15
6.1.2	Contaminated data	17
6.1.3	Unbalanced data	18
A	About rsae	18
B	Comparing the ML estimates of rsae with those of nlme	18
C	Package installation on Linux systems	19

*Contact: Tobias Schoch, University of Applied Sciences Northwestern Switzerland, Riggensbachstrasse 16, CH-4600 Olten, Switzerland; e-mail: tobias.schoch@fhnw.ch; phone: +41 62 286 01 47.

1 Introduction

The **rsae** package offers functions to estimate

- model A, area-level SAE model, aka Fay-Herriot model,
- model B, basic unit-level SAE model,

by either maximum likelihood or robust methods. The classification of the models (A or B) follows the proposal of [Rao \(2003\)](#). Currently, the package includes the following robust (i.e. bounded-influence estimating equation) methods for model A:

- Huber-type M -estimation, using a slight generalization of Huber's Proposal 2 to estimate the variance components; cf., [Richardson and Welsh \(1995\)](#), [Sinha and Rao \(2009\)](#), [Schoch \(2011\)](#),
- *variance/MSE estimation [testing phase; floating-point arithmetic problems]*
- *robust high-breakdown point S -estimation [testing phase, expected release for SAE conference 2011],*
- *robust simple and minimal-iterative methods using fast-LTS and a MED-MAD strategy.[under development]*

(Robust estimating methods for model B will be released in the near future)

NOTE: In order to use the full functionality of **rsae**, I recommend to install the **robust-base** package.

2 Getting started

I will not attempt to provide another introduction to R. There are several excellent resources intended to accomplish this task.

Once R is running, the installation of additional packages is straightforward. A platform-independent way to install the **rsae** package is to type

```
> install.package("rsae")
```

in the console (note that the character ">" is not part of the command; it denotes the command prompt of the command-line interface, indicating that R is ready to read your commands). Provided your computer has a proper internet connection (and you have sufficient writing permissions on your system), the installation of the package should proceed automatically (at least on Windows and Apple computers, because binary sources are available from CRAN). On POSIX compliant OS, e.g., AIX, HP-UX, Solaris, etc. and mostly POSIX-compliant systems such as Linux, OpenSolaris, etc. installation is more involved (see [Appendix C](#) for more details on the installation process).

Once the **rsae** package is installed, we need to load it to the current session.¹

¹Note that we have to load a package in every session where we need its functionality, whereas installation is needed a single time on your computer.

```
> library(rsae)
```

The model fitting exercise with `rsae` takes always three (or more) steps. Namely,

- setting up a `saemodel`, given some data,
- fitting the `saemodel` by robust methods,
- (robustly) predicting the random effects and the area-specific means, given the robustly fitted model.

These steps will be discussed in subsequent Sections.

3 Setting up a model

First of all, we have to set up a model. We use the `landsat` data from [Battese et al. \(1988\)](#), the landmark paper on the basic unit-level SAE model (here: units=segments, each about 250 hectares; areas=Counties). Those readers who are not familiar with the `landsat` can type `help(landsat)` in the R console in order to get a description of the data.

3.1 Exploring the data

First, we have to load the `landsat` data into the workspace.

```
> data(landsat)
```

Next, we will explore the data. The `names` command reports the names of the variables in the `landsat` data.

```
> names(landsat)
```

```
[1] "SegmentsInCounty" "SegmentID"      "HACorn"
[4] "HASoybeans"      "PixelsCorn"     "PixelsSoybeans"
[7] "MeanPixelsCorn"  "MeanPixelsSoybeans" "outlier"
[10] "CountyName"
```

Get a table (i.e., an aggregation) of the variable `CountyName`.

```
> table(landsat$CountyName)
```

Cerro Gordo	Hamilton	Worth	Humboldt	Franklin	Pocahontas
1	1	1	2	3	3
Winnebago	Wright	Webster	Hancock	Kossuth	Hardin
3	3	4	5	5	6

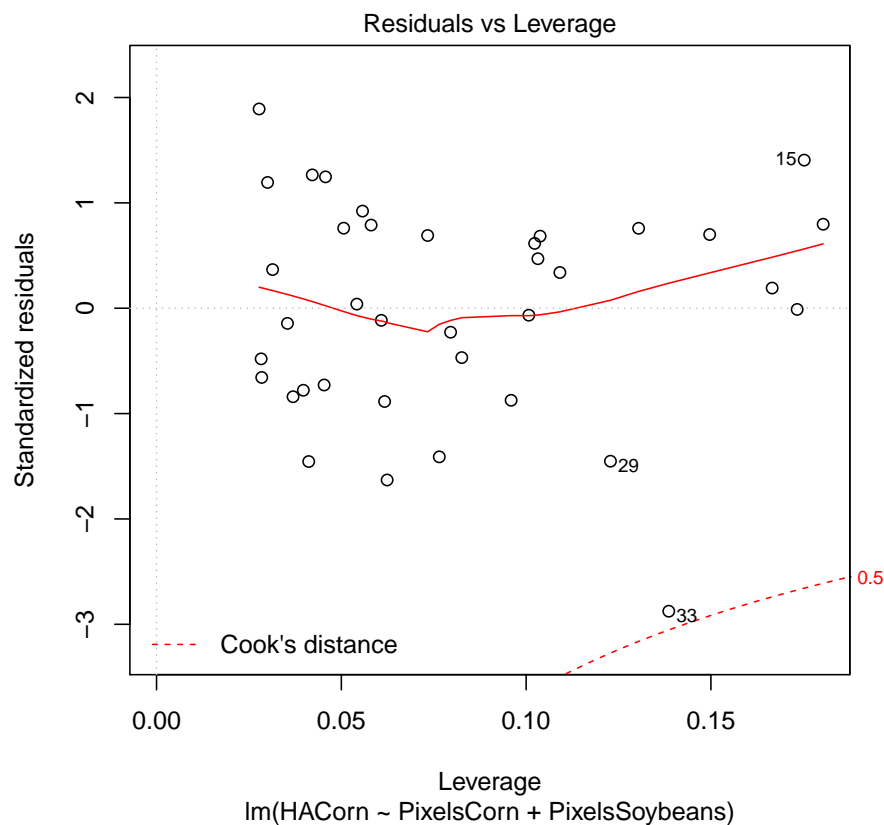
So, there are twelve areas – the smallest areas (Cerro Gordo, Hamilton, and Worth) contain one unit, the largest area, Hardin, involves six units.

Next, we want to have a look at observations no. 30 to 35 and some interesting variables.

```
> landsat[30:35, c("HACorn", "PixelsCorn", "PixelsSoybeans", "CountyName",
+ "outlier")]
```

	HACorn	PixelsCorn	PixelsSoybeans	CountyName	outlier
30	122.66	342	182	Kossuth	FALSE
31	104.21	294	179	Kossuth	FALSE
32	88.59	220	262	Hardin	FALSE
33	88.59	340	87	Hardin	TRUE
34	165.35	355	160	Hardin	FALSE
35	104.00	261	221	Hardin	FALSE

Note that I added the variable `outlier` to the original data. This variable flags observation no. 33 as outlier. This is in line with [Battese et al. \(1988\)](#), who indicate that for both observations no. 32 and 33, the interviewed farm operators reported for HACorn the same value. Despite the same number of estimated hectares under corn (i.e., 88.59), the readings for `PixelsCorn` are very different for obs. no. 32 and 33. To be precise, we would call observation no. 33 a leverage point rather than an outlier.



The following figure is a display of the (standardized) residuals of the linear model against the leverage. Note that obs. no. 33 is very close the the bound on the values of Cook's distance.

```
> linmodel <- lm(HACorn ~ PixelsCorn + PixelsSoybeans, data = landsat)
> plot(linmodel, 5)
```

Note that [Sinha and Rao \(2009\)](#), on the other hand, included this bad leverage point in their simulation exercise and obtained completely different estimates.

3.2 We set up our first model

Having explored the data, we consider setting up the model. The (first of their) model writes ([Battese et al., 1988](#))

$$HACorn_{i,j} = \alpha + \beta_1 \cdot PixelsCorn_{i,j} + \beta_2 \cdot PixelsSoybeans_{i,j} + u_i + e_{i,j},$$

where $j = 1, \dots, n_i$, $i = 1, \dots, 12$.

The `rsae` package provides the function `saemodel` to set up a model. The model is specified by mainly three arguments: (1) the `formula` argument defines the fixed-effect part (the `~` operator separates dependent and independent variables), (2) the `area` argument specifies the area-level random effect (here, the variable `CountyName` serves as area identifier; note the leading `~` operator), (3) the `data` argument specifies the data (a `data.frame`). The only "difficulty" here is that we use `data=subset(landsat, subset=(outlier==FALSE))` to include only non-outlying observations, instead of `data=landsat`. We call our model `bhfmmodel` (Battese, Harter, and Fuller).

```
> bhfmmodel <- saemodel(formula = HACorn ~ PixelsCorn + PixelsSoybeans,
+   area = ~CountyName, data = subset(landsat, subset = (outlier ==
+   FALSE)))
```

(Note that in R, a `formula` object (evaluated by `model.frame` and `model.matrix`) contains always an intercept term. In the case we want to fit the above model without an intercept, we have to change the RHS of the formula part to look like this: `-1 + PixelsCorn + PixelsSoybeans`).

Suppose we have generated several different models (e.g., using different independent variables), and they all reside in the current workspace. It may be difficult to figure out which of them is the `bhfmmodel` (except you have adopted unique naming conventions). For situations like this, type the name of the model to get some information.

```
> bhfmmodel
```

```
SAE MODEL TYPE: B (J.N.K. Rao's classification)
```

```
---
```

```
FIXED EFFECTS: HACorn ~ (Intercept) + PixelsCorn + PixelsSoybeans
```

```
AREA-SPECIFIC RANDOM EFFECTS: CountyName
```

If you need to know more about a particular model, you may use the `summary` method.

```
> summary(bhfmmodel)
```

Model summary:

```
saemodel(formula = HACorn ~ PixelsCorn + PixelsSoybeans, area = ~CountyName,
  data = subset(landsat, subset = (outlier == FALSE)))
---
No. of areas: 12
No. of obs.: 36
Smallest area: 1 units
Largest area: 5 units
```

The important thing to note is that `summary` reminds you of the fact that the model was generated using only non-outlying observations.

4 (Robust) Estimation

Having set up our model, we consider estimating the parameters of the Gaussian core model by different methods. All fitting is done using the following workhorse function

```
fitsaemodel(method, model, ...)
```

Depending on the arguments delivered, `fitsaemodel` decides what method to use. The decision is based on the `method` argument.

4.1 Maximum likelihood estimation

To start with, we compute the usual maximum likelihood (ML) estimates. Therefore, we call `fitsaemodel` with `method="ml"` (the ML methods does not need any additional [...] arguments).

```
> mlfit <- fitsaemodel("ml", bhfmodel)
```

Type name of the fit, i.e. `mlfit`, to get a display of the model fit.

```
> mlfit
```

ESTIMATES OF SAE-MODEL (model type B)

Method: Maximum likelihood estimation

Fixed effects

Model: HACorn ~ (Intercept) + PixelsCorn + PixelsSoybeans

Coefficients:

(Intercept)	PixelsCorn	PixelsSoybeans
50.968	0.329	-0.134

Random effects

Model: ~1 | CountyName

(Intercept)	Residual
-------------	----------

Std. Dev. 11.0	11.7
----------------	------

```

---
Number of Observations:  36
Number of Areas:  12

```

Note that the layout of the output is similar to the one of the `lme` (linear mixed-effects model) function in the `nlme` package (cf. [Pinheiro et al., 2011](#)). In particular, the output on the random-effects part mimicks what the `print` methods reports for `lme` (i.e., `~1|Countyname`).

To learn more about the fit, we may call the `summary` method. The properties of the model summary will be discussed in the context of Huber-type M -estimation (see below).

4.1.1 When the ML estimator does not converge

Now, we deliberately set the number of outer-loop iterations to one, in order to show the model report, when the algorithm fails to converge.

```
> fitsaemodel("ml", bhfmodel, niter = 1)
```

```
THE METHOD DID NOT CONVERGE!
```

```

---
1) use summary() of your fitted model to learn more
2) study the documentation using the command ?fitsaemodel
3) you may call fitsaemodel with 'init' equal to (either) 'lts'
   or 's' (this works also for ML, though it may no be very efficient)
4) if it still does not converge, the last resort is to modify
   'acc' and/or 'niter' (and hope and pray)

```

Obviously, `niter=1` is a bad choice. Now, it may happen that the ML-method of `fitsaemodel` does not converge (even if the parameters have not been modified). A remedy is to initialize the ML method by a regression S -estimator (sic!), calling `fitsaemodel` with `init="s"`. Obviously, this approach is not optimal in terms of computing time. Nonetheless, by this specification, `fitsaemodel` enters the "safe mode" of robust Huber-type M -estimation and applies several checks whether the algorithm behaves well (these checks are ignored in the default mode).

4.1.2 When the mixed linear model is not appropriate

Suppose that our data do not have area-specific variation. Notably, we shall generate data based on the linear model,

$$y_{i,j} = (1, x_{i,j})^T \boldsymbol{\beta} + e_{i,j}, \quad (1)$$

where $x_{i,j} \sim N(0, 1)$, $e_{i,j} \sim N(0, 1)$, $\boldsymbol{\beta} = (1, 1)^T$, $n_i = n = 10$, $\forall i = 1, \dots, 10$ (balanced data). The following code generates the data.

```

> set.seed(12345)
> n <- 200
> beta <- c(1, 1)

```

```

> cst <- rep(1, n)
> x <- rnorm(n)
> y <- as.matrix(cbind(cst, x)) %*% beta + rnorm(n)
> areaid <- rep(1:10, each = 10)
> df <- data.frame(y = y, x = x, areaid = areaid)

```

The OLS fit of our linear model is

```

> lm(y ~ x, data = df)

```

Call:

```

lm(formula = y ~ x, data = df)

```

Coefficients:

(Intercept)	x
1.080	1.034

Then, we set up the `saemodel` with the identical data (i.e., `df`)

```

> fakelm <- saemodel(y ~ x, area = ~areaid, data = df)

```

and consider the following model fit

```

> fitsaemodel("ml", fakelm)

```

ESTIMATES OF SAE-MODEL (model type B)

Method: Maximum likelihood estimation

Fixed effects

Model: $y \sim (\text{Intercept}) + x$

Coefficients:

(Intercept)	x
1.08	1.03

Random effects

Model: $\sim 1 \mid \text{areaid}$

NOTE THAT THE VARIANCE OF THE AREA-LEVEL RANDOM EFFECT IS ALMOST ZERO! DO YOU REALLY NEED THE RANDOM EFFECT? IF SO, GO AHEAD. HOWEVER, YOU SHOULD CONSIDER FITTING A (ROBUST) GLS MODEL.

	(Intercept)	Residual
Std. Dev.	0.000	0.955

Number of Observations: 200

Number of Areas: 10

The report indicates that the random-effect variance is close to zero or equal to zero. Therefore, the MLM model is not appropriate.

4.2 Huber-type M -estimation

Huber-type M -estimation is the recommended estimation method for situations where the response variable is supposed to be (moderately) contaminated. The M -estimators downweight the influence of outlying observations (in the vector of responses) on the estimates. Notably, they bound the influence of outliers. But, no attempt is made to limit the effect of leverage points (see above). In principle, one may adapt generalized regression M -estimators (GM) to the family of linear mixed-level models in order to deal with influential observations/leverage (i.e., to bound also the influence of the design matrix). This has been done by Richardson (1997). However, in terms of numerical properties, the Schweppe- and Mallows-type weighted GM -estimators turned out to be extremely unstable (Richardson, 1995). Therefore, we propose to use the S -estimator (with constrained parameter space) if the data are heavily contaminated and/or contain influential observations/leverage (see below).

Next, we discuss two different fitting modes for the Huber-type M -estimation method.

- If the response variable is supposed to be uncontaminated or contaminated by only a couple of outliers, I recommend to use the (very fast) **default mode**.
- If the response variable is moderately contaminated and/or if the **default-mode algorithm failed to converge**, I recommend the **safe mode**.

4.2.1 Default mode

The default-mode setup of the Huber-type M -estimation exercise is

```
> huberfit <- fitsaemodel("huberm", bhfmodel, k = 1.5)
```

where k denotes the robustness-tuning constant of the Huber ψ -function ($0 < k \leq \infty$; note that (in the limit) $k = \infty$ leads to the ML estimates). (**NOTE:** that in the simple location-scale model, the choice of $k=1.345$ leads to estimates which feature an asymptotic relative efficiency w.r.t. the ML estimate of approx. 95% at the true (uncontaminated) Gaussian core model. This property does **not** directly carry over to the estimates of mixed-linear models!)

```
> huberfit
```

```
ESTIMATES OF SAE-MODEL (model type B)
```

```
Method: Huber-type M-estimation
```

```
Robustness tuning constant: k = 1.5
```

```
---
```

```
Fixed effects
```

```
Model: HACorn ~ (Intercept) + PixelsCorn + PixelsSoybeans
```

```
Coefficients:
```

(Intercept)	PixelsCorn	PixelsSoybeans
50.285	0.328	-0.139

```
---
```

```
Random effects
```

```

Model: ~1 | CountyName
      (Intercept) Residual
Std. Dev. 12.0      12.4
---

```

```

Number of Observations: 36
Number of Areas: 12

```

To learn more about the fit, namely about the number of iterations and the robustness properties, we shall call the `summary` method.

```
> summary(huberfit)
```

```
SUMMARY: ESTIMATES OF SAE-MODEL (model type B)
```

```
Method: Huber-type M-estimation
```

```
Robustness tuning constant: k = 1.5
```

```
---
```

```
Degree of downweighting/winsorization:
```

```

              sum(wgt)/n
fixeff        0.984
residual var  0.972
area raneff var 0.984
---

```

```
User specified number of iterations (niter) and
numeric precision (acc):
```

```

              niter  acc
overall loop  4e+01 1e-05
fixeff        2e+02 1e-05
residual var  2e+02 1e-05
area raneff var 1e+02 1e-05
---

```

Number of runned EE-specific iterations in each call (given the user-defined specs), reported for each of the 12 overall iterations separately:

```

      fixeff  residual var  area raneff var
1       6      17          12
2       2       8          12
3      10      10          12
4      11       7          12
5       6       6          12
6       8       5          12
7       6       5          12
8       5       4          12
9       4       3          12
10      2       2          12

```

11	2	2	12
12	1	1	12

-EOF-

The output is by default separated into 4 blocks when the algorithm converged. In the case of non-convergence, the report is slightly adapted.

- Block 1 reports general information on the method and the robustness tuning constant(s).
- Block 2 reports the degree of down-weighting outlying residuals at the final iteration. The degree of down-weighting is reported for all estimating equations (EE) separately. In general $d = 1$, if no down-weighting took place. Conversely, the smaller d , the more outlying residuals have been down-weighted (and/or the same outliers are heavier down-weighted).
- Block 3 reports the default or user-specified max number of iterations, `niter`, and the numeric tolerance, `acc`, used in the termination rule of the algorithms.
- Block 4 reports the number of iterations that each of the estimating equation-specific (EE-specific) loops (aka inner loops) and the overall loop (aka outer loop) used to conform to the termination rule. Each row in the table represents a single run of the overall loop. In the example, the algorithm needed 7 overall loops. The entries of a row refer to the EE-specific number of iterations in a particular overall loop.

It is evident that the number of (inner-loop) iterations for `fixeff` and `residual var` become smaller, the higher the number of outer/overall loops. This is not (and will never be) the case for `area` `ranef` `var` because it is obtained using a different optimization method.

In general, we can feel confident if the number of inner-loop iterations decrease (in the case of `fixeff` and `residual var`). However, there are situations where the algorithm converges perfectly without featuring such a nice decrease in the number of inner-loop iterations.

4.2.2 Safe mode

The safe mode should be used when the data are supposed to be moderately contaminated and/or the algorithm in default mode failed to converge. The safe-mode algorithm is initialized by a high-breakdown-point regression estimator. However, it is only safe up to a certain degree of contamination (i.e., breakdown-point is rather low for M -estimators). In the presence of too many outliers, the M -estimator will break down.

NOTICE: in order to use the safe-mode functions, you need the **robustbase** package ([Rousseeuw et al., 2011](#)).

The safe mode is entered if one specifies (in `fitsaemodel`) either

- `init = "lts"` for initializing the method by a fast-LTS regression estimator ([Rousseeuw, 1984](#); [Rousseeuw and Van Driessen, 2006](#)),

- `init = "s"` for initializing the method by a regression S -estimator ([Rousseeuw and Yohai, 1984](#); [Salibian-Barrera and Yohai, 2006](#)).

(Also, the safe mode uses a couple of tests to check whether the estimates at consecutive iterations behave well. Notably, it prevents cycling estimates (i.e., the situation when the algorithm is trapped in a flip-flop), which typically occurs for very small robustness-tuning constants).

In general, the results of `fitsaemodel` with either `init = "s"` or `init = "lts"` are the same. For data with more than 50,000 observations, `init="s"` is considerably faster. The call is

```
> fitsaemodel("huberm", bhfmodel, k = 1.2, init = "s")

ESTIMATES OF SAE-MODEL (model type B)
Method:  Huber-type M-estimation
Robustness tuning constant:  k =  1.2
---
Fixed effects
Model:  HACorn ~ (Intercept) + PixelsCorn + PixelsSoybeans
Coefficients:
      (Intercept)      PixelsCorn  PixelsSoybeans
           57.134           0.312           -0.161
---
Random effects
Model:  ~1 | CountyName
              (Intercept)  Residual
Std. Dev.   13.1          12.9
---
Number of Observations:  36
Number of Areas:  12
```

Also, we can get more information calling the `summary` method.

4.3 S -estimation

[has to be written]

5 Robust prediction

Once the parameters of the Gaussian core model have been estimated robustly, we consider (robustly) predicting the random effects. [Sinha and Rao \(2009\)](#) proposed to solve the robustified mixed model equations ([Fellner, 1986](#)) by a Newton-Raphson-type updating scheme that is obtained from a Taylor-series expansion of the robustified mixed model equations. Consequently, computation is very involved.

However, [Schoch \(2011\)](#) showed that robust predictions of the random effects can be obtained far more easily. Namely, since the robustly estimated parameters of the core Gaussian model determine the model completely, prediction is straightforward.

The workhorse function for (robust) prediction is given by

```
robpredict(fit, k, areadata=NULL)
```

where `fit` is a fitted model (an object produced by `fitsaemodel`), `k` is the robustness-tuning constant (of the Huber ψ -function) for robust prediction. The robustness-tuning constant `k` does not necessarily be the same as the one used in `fitsaemodel`. Further, choosing `k` sufficiently large (e.g., `k=20000` should work in (almost) all situations), `robpredict` produces the usual EBLUP predictions. If `areadata=NULL` (the default setting), the prediction is based on the same data that have been used for the model fitting exercise (i.e., within-sample prediction). However, in the SAE context, we are usually interested in (robustly) predicting the small-area means. Therefore, we deliver the area-specific population means through `areadata`.

In the `landsat` example, the county-specific population means of pixels of the segments under corn and soybeans are recorded in the variables `MeanPixelsCorn` and `MeanPixelsSoybeans`, respectively. Note that each sample segment in a particular county has assigned the county-specific mean (in the `landsat` data). Therefore, each population mean of the variables `MeanPixelsCorn` and `MeanPixelsSoybeans` occurs n_i times. The unique county-specific are obtained using

```
> d <- unique(landsat[-33, c("MeanPixelsCorn", "MeanPixelsSoybeans",
+   "CountyName")])
> d <- cbind(rep(1, 12), d)
> rownames(d) <- d$CountyName
> d <- d[, 1:3]
```

Let us have a look at `d`.

```
> d
```

	rep(1, 12)	MeanPixelsCorn	MeanPixelsSoybeans
Cerro Gordo	1	295.29	189.70
Hamilton	1	300.40	196.65
Worth	1	289.60	205.28
Humboldt	1	290.74	220.22
Franklin	1	318.21	188.06
Pocahontas	1	257.17	247.13
Winnebago	1	291.77	185.37
Wright	1	301.26	221.36
Webster	1	262.17	247.09
Hancock	1	314.28	198.66
Kossuth	1	298.65	204.61
Hardin	1	325.99	177.05

The first column of `d` is a dummy variable (the `bhfmodel` has an intercept term). The second and third column represent the county-specific population means of segments under corn and soybeans (the rows of the above table are labeled with the county names).

Next, we consider predicting the random and fixed effects and the county means. Note that we choose `k=10000`. This leads to results that are equivalent to those of the standard EBLUP, and therefore not robust.

```
> pr <- robpredict(mlfit, k = 10000, areadata = d)
```

The results are

```
> pr
```

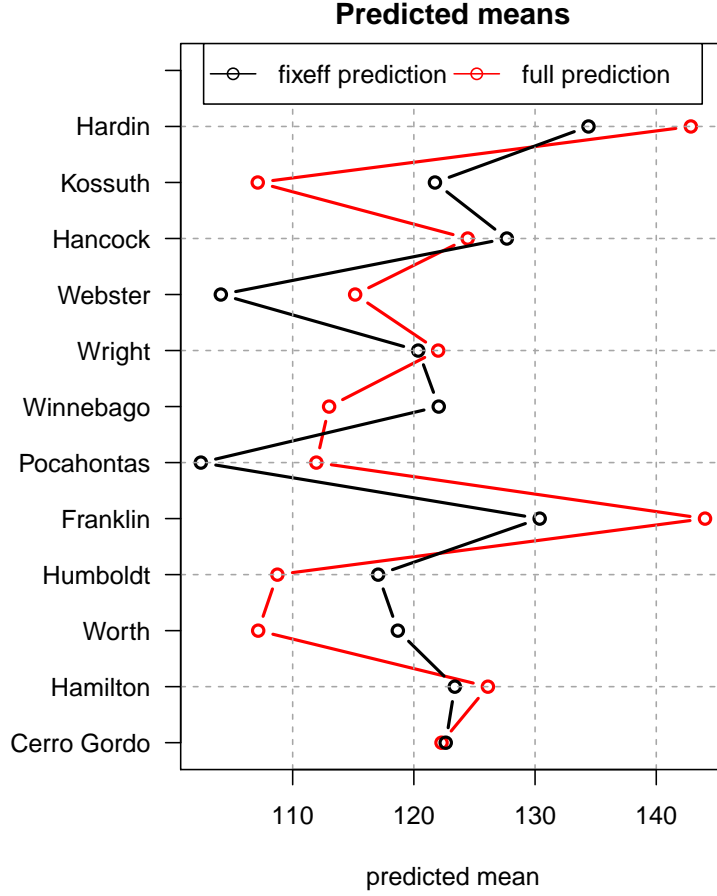
Robustly Estimated/Predicted Area-Level Means:

	raneff	fixeff	predicted mean
Cerro Gordo	-0.348	122.629	122.281
Hamilton	2.731	123.379	126.110
Worth	-11.522	118.677	107.154
Humboldt	-8.313	117.053	108.741
Franklin	13.641	130.380	144.021
Pocahontas	9.529	102.425	111.954
Winnebago	-9.043	122.052	113.009
Wright	1.648	120.358	122.006
Webster	11.082	104.073	115.155
Hancock	-3.229	127.671	124.442
Kossuth	-14.621	121.740	107.119
Hardin	8.445	134.408	142.853

(robustness tuning constant k = 10000)

and a visual display of the predicted county means

```
> plot(pr)
```



6 Utility functions

6.1 Synthetic data generation

The `rsae` package is shipped with a device to generate synthetic data (balanced and unbalanced) of basic unit-level model. This tool is particularly useful when comparing the behavior of different methods (e.g., in simulation studies) because it enables the user to generate data tailored for specific problems. In particular, `makedata` is able to generate outlier-contaminated data.

6.1.1 Uncontaminated data

The core model is given by

$$y_{i,j} = \alpha + \mathbf{x}_{i,j}^T \boldsymbol{\beta} + u_i + e_{i,j}, \quad j = 1, \dots, n_i, \quad i = 1, \dots, g, \quad (2)$$

where $y_{i,j}$ is the response variable, $\mathbf{x}_{i,j}$ is a p -vector of design variables, u_i is an area-specific random effect, and $e_{i,j}$ is a model error. The total number of observations is $n = \sum_{i=1}^g n_i$. Moreover, we assume that u_i and $e_{i,j}$ are independent and that

$$\mathbf{x}_{i,j} \sim MVN(\mathbf{0}, \mathbf{I}_p), \quad (3)$$

and

$$e_{i,j} \sim N(0, v_e), \quad u_i \sim N(0, v_u), \quad (4)$$

where \mathbf{I}_i is the $(p \times p)$ identity matrix, $j = 1, \dots, n_i$; $i = 1, \dots, g$.

The default values of `makedata` for balanced data, i.e., $n_i = n, \forall i = 1, \dots, g$ are reported in Table 1.

Table 1: Default setup of `makedata` (balanced data)

n	g	v_e	v_u	α	β
4	20	1	1	1	1

Now, call the `makedata` function

```
> mymodel <- makedata()
```

to generate the data. Note that `makedata` has been called without arguments, implying that all arguments are given by the default value. However, you are well advised to specify the random seed (default: `seed=1024`) to meet our needs. Otherwise, your computer generates the same random numbers every time `makedata()` is called.

By typing the name of the generated model (here `mymodel`) in the console, we obtain (through the `print` method) some characteristics of the synthetic data. This feature is particularly useful if we have several different models in the workspace (and have some troubles figuring out what the models actually do).

```
> mymodel
```

```
SAE MODEL TYPE: B (J.N.K. Rao's classification)
```

```
DATA: Synthetic, simulated data
```

```
MODEL:
```

```
  y_ij = intercept + sum_k[ beta_k * x_kij] + v_i + e_ij
with
  each x_kij ~ N(0, 1), k=1,...,2
  v_i ~ N(0, 1)
  e_ij ~ N(0, 1)
```

In addition, there is a `summary` method.

```
> summary(mymodel)
```

```
Model summary:
```

```
makedata()
```

```
---
```

```
No. of areas: 20
```

```
No. of obs.: 80
```

```
Balanced data, each area has 4 units
```


6.1.2 Contaminated data

The `makedata` allows to draw the model errors $e_{i,j}$ and/or random effects u_i from a (contamination) mixture distribution.

Contamination of the law of $e_{i,j}$

```
> makedata(ve.epsilon = 0.1)
```

SAE MODEL TYPE: B (J.N.K. Rao's classification)

DATA: Synthetic, simulated data

MODEL:

```
  y_ij = intercept + sum_k[ beta_k * x_kij] + v_i + e_ij
with
  each x_kij ~ N(0, 1), k=1,...,2
  v_i ~ N(0, 1)
  e_ij ~ (0.9)*N(0, 1) + 0.1*N(0, 41)
```

Contamination of the law of u_j

```
> makedata(vu.epsilon = 0.1)
```

SAE MODEL TYPE: B (J.N.K. Rao's classification)

DATA: Synthetic, simulated data

MODEL:

```
  y_ij = intercept + sum_k[ beta_k * x_kij] + v_i + e_ij
with
  each x_kij ~ N(0, 1), k=1,...,2
  v_i ~ (0.9)*N(0, 1) + 0.1*N(0, 41)
  e_ij ~ N(0, 1)
```

and both

```
> makedata(vu.epsilon = 0.1, ve.epsilon = 0.1)
```

SAE MODEL TYPE: B (J.N.K. Rao's classification)

DATA: Synthetic, simulated data

MODEL:

```
  y_ij = intercept + sum_k[ beta_k * x_kij] + v_i + e_ij
with
  each x_kij ~ N(0, 1), k=1,...,2
  v_i ~ (0.9)*N(0, 1) + 0.1*N(0, 41)
  e_ij ~ (0.9)*N(0, 1) + 0.1*N(0, 41)
```

It goes without saying that we may change other parameters of the `makedata` function. For instance, we may generate contaminated, unbalanced data; see below.

6.1.3 Unbalanced data

Further, we can generate unbalanced data very easily. First, we have to set both arguments `n` and `g` in `makedata` equal to `NULL` (since `n` is not a constant for unbalanced data). Next, we have to tell `makedata` to produce unbalanced data on grounds of a vector of area identifiers (ID; all units in a particular area are assigned an area-specific identifier code). By way of example, suppose we want to generate 16 units that reside in 5 areas. The following code snippet defines the vector identifiers.

```
> my_area_id = c(1, 1, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 5)
```

Thus, the vector of area size is $nsize^T = (n_1, \dots, n_5) = (5, 2, 3, 4, 1)^T$. (Note that the IDs do not have to be sorted (as in the display); any permutation of the IDs is a valid argument. The number of areas is determined from the number of unique elements.)

The data are generated by the following command.

```
> mymodelub <- makedata(n = NULL, g = NULL, areaID = my_area_id)
```

A About rsae

```
> citation("rsae")
```

To cite package 'rsae' in publications use:

Tobias Schoch (2011). `rsae`: Robust Small Area Estimation. R package version 0.1-2.

A BibTeX entry for LaTeX users is

```
@Manual{,
  title = {rsae: Robust Small Area Estimation},
  author = {Tobias Schoch},
  year = {2011},
  note = {R package version 0.1-2},
}
```

B Comparing the ML estimates of rsae with those of nlme

For ease of comparability, we report (again) the maximum likelihood estimates obtained by `fitsaemodel`.

```
> fitsaemodel("ml", bhfmodel)
```

```
ESTIMATES OF SAE-MODEL (model type B)
Method: Maximum likelihood estimation
---
Fixed effects
```

```

Model: HACorn ~ (Intercept) + PixelsCorn + PixelsSoybeans
Coefficients:
  (Intercept)      PixelsCorn  PixelsSoybeans
        50.968          0.329         -0.134
---
Random effects
Model: ~1 | CountyName
      (Intercept)  Residual
Std. Dev.   11.0      11.7
---
Number of Observations: 36
Number of Areas: 12

```

These results are equal to those of the `lme` function in the **nlme** package (which may be called the industrial standard).

```

> require(nlme)
> nlme::lme(HACorn ~ PixelsCorn + PixelsSoybeans, random = ~1 |
+   CountyName, data = subset(landsat, subset = (outlier == FALSE)),
+   method = "ML")

```

```

Linear mixed-effects model fit by maximum likelihood
Data: subset(landsat, subset = (outlier == FALSE))
Log-likelihood: -147.0126
Fixed: HACorn ~ PixelsCorn + PixelsSoybeans
      (Intercept)      PixelsCorn  PixelsSoybeans
        50.9675892        0.3285805        -0.1337102

```

```

Random effects:
Formula: ~1 | CountyName
      (Intercept)  Residual
StdDev:   11.00298 11.71806

```

```

Number of Observations: 36
Number of Groups: 12

```

C Package installation on Linux systems

In this section, I give some details on the installation of **rsae** on a Linux powered system. These details essentially summarize my experience with **openSUSE 11.4** on a **x86_64** platform. Some of the details carry directly over to other (mostly-) POSIX-compliant systems, others do not. It is up to the user to modify these hints to fit on his/her system. Note the following:

- Make sure that you have installed **R-devel** (or **r-base-dev**) in addition to **R-base**; see [R-admin \(2011\)](#), chapters 2, 6 and Appendix A.1)

- The `rsae` package contains FORTRAN 90 code that must be compiled (and linked to R's BLAS and LAPACK) at installation. The code has been written and tested for `gfortran` (v. 4.5-1). The code does not use (to the best of my knowledge) any compiler-specific code. See [R-admin \(2011, Appendix B.8\)](#) for more details on FORTRAN compilers supported by R, and in what order they are selected if you have several compilers. Therefore, any compatible FORTRAN compiler should work. However, I highly recommend to use `gfortran`. (Note that if the C compiler that your current R installation knows of is `gcc 4` [the command `R CMD config CC` tells you what R uses; then check the version `gcc -v`], it will automatically choose `gfortran`)
- you need the system tool GNU `make`. By default, most Linux distributions are delivered with a copy of `make`, whereas `openSUSE 11.4` is not. You thus have to grab a version from the internet. See [R-admin \(2011, Appendix B.5\)](#) for more on `make`.

References

- Pinheiro, J., D. Bates, S. DebRoy, D. Sarkar and the R Development Core Team (2011): *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-100.
- Battese, G.E., R.M. Harter, and W.A. Fuller (1988): *An Error-Components Model for Prediction of County Crop Areas Using Survey and Satellite Data*, Journal of the American Statistical Association 83, pp. 28-36.
- Fellner, W. (1986): *Robust estimation of variance components*. Technometrics 28, pp. 51-60.
- R-admin (2011): *R Installation and Administration (version 2.13.0)*, ed. by: R Development Core Team, Vienna. URL <http://stat.ethz.ch/CRAN/doc/manuals/R-admin.html>.
- R-exts (2011): *R Writing R Extensions (version 2.13.0)*, ed. by: R Development Core Team, Vienna. URL <http://stat.ethz.ch/CRAN/doc/manuals/R-exts.html>.
- Rao, J.N.K.(2003): *Small Area Estimation*, New Work: John Wiley and Sons.
- Richardson, A.M. (1997): *Bounded Influence Estimation in the Mixed Linear Model*. Journal of the American Statistical Association 92, pp.151-161
- Richardson, A.M. (1995): *Some problems in estimation in mixed linear models*. PhD thesis, Australian National University.
- Richardson, A.M. and A.H. Welsh (1995) *Robust restricted maximum likelihood in mixed linear model*. Biometrics 51, pp. 1429-1439.
- Rousseeuw, P.J. (1984): *Least Median of Squares Regression*. Journal of the American Statistical Association 79, pp. 871-880
- Rousseeuw, P.J., and V.J. Yohai (1984): *Robust regression by means of S-estimators*, in: Robust and Nonlinear Time Series, Franke, J., W. Härdle, and R.D. Martin (eds.), Lecture Notes in Statistics 26, pp. 256-272, New York: Springer.

- Rousseeuw, P.J., and K. Van Driessen (2006): *Computing LTS Regression for Large Data Sets*, Data Mining and Knowledge Discovery 12, pp. 29-45.
- Rousseeuw, P.J., C. Croux, V. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, M. Maechler (2011): *robustbase: Basic Robust Statistics*. R package version 0.7-6. URL <http://CRAN.R-project.org/package=robustbase>
- Salibian-Barrera, M. and V.J. Yohai (2006): *A fast algorithm for S-regression estimates*. Journal of Computational and Graphical Statistics 15, pp. 414-427.
- Schoch, T. (2011) *The robust basic unit-level small area model. A simple and fast algorithm for large datasets*, in: Proceedings of the New Technologies and Techniques Conference (NTTS), EUROSTAT, Brussels.
- Sinha, S.K. and J.N.K. Rao (2009) *Robust small area estimation*. Canadian Journal of Statistics 37, pp. 381-399.