Applications of Statistical Simulation in the Case of EU-SILC: Using the R Package simFrame

Andreas Alfons
Erasmus University Rotterdam

Matthias Templ
Zurich University of Applied Sciences

Peter Filzmoser
Vienna University of Technology

Abstract

This paper demonstrates the use of **simFrame** for various simulation designs in a practical application with EU-SILC data. It presents the full functionality of the framework regarding sampling designs, contamination models, missing data mechanisms and performing simulations separately on different domains. Due to the use of control objects, switching from one simulation design to another requires only minimal changes in the code. Using bespoke R code, on the other hand, changing the code to switch between simulation designs would require much greater effort. Furthermore, parallel computing with **simFrame** is demonstrated.

Keywords: R, statistical simulation, EU-SILC.

1. Introduction

This is an updated version of a supplementary paper to "An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**" (Alfons, Templ, and Filzmoser 2010d) and demonstrates the use of **simFrame** (Alfons 2013) in R (R Development Core Team 2010) for various simulation designs in a practical application. It extends the example for design-based simulation in Alfons *et al.* (2010d) (Example 6.1). Different simulation designs in terms of sampling, contamination and missing data are thereby investigated to present the strengths of the framework.

Note that the paper is supplementary material and is supposed to be read after studying vignette `"simFrame-intro"` (an updated version of Alfons *et al.* 2010d). It does not give a detailed discussion about the motivation for the framework, nor does it describe the design or implementation of the package. Instead it is focused on showing its full functionality for design-based simulation in additional code examples with brief explanations. However, model-based simulation is not considered here.

The European Union Statistics on Income and Living Conditions (EU-SILC) is panel survey conducted in EU member states and other European countries and serves as basis for measuring risk-of-poverty and social cohesion in Europe. An important indicator calculated from this survey is the *Gini coefficient*, which is a well-known measure of inequality. In the following examples, the standard estimation method (EU-SILC 2004) is compared to two semiparametric methods under different simulation designs. The two semiparametric approaches are based on fitting a Pareto distribution (e.g., Kleiber and Kotz 2003) to the upper tail of the

data. In the first approach, the classical Hill estimator (Hill 1975) is used to estimate the shape parameter of the Pareto distribution, while the second uses the robust partial density component (PDC) estimator (Vandewalle, Beirlant, Christmann, and Hubert 2007). All these methods are implemented in the R package **laeken** (Alfons, Holzer, and Templ 2010a). For a more detailed discussion on Pareto tail modeling in the case of the Gini coefficient and a related measure of inequality, the reader is referred to Alfons, Templ, Filzmoser, and Holzer (2010e).

The example data set of **simFrame** is used as population data throughout the paper. It consists of 58 654 observations from 25 000 households and was synthetically generated from Austrian EU-SILC survey data from 2006 using the data simulation methodology by Alfons, Kraft, Templ, and Filzmoser (2010b), which is implemented R package **simPopulation** (Alfons and Kraft 2010).

2. Application of different simulation designs to EU-SILC

First, the required packages and the data set need to be loaded.

```
R> library("simFrame")
R> library("laeken")
R> data("eusilcP")
```

Then, the function to be run in every iteration is defined. Its argument `k` determines the number of households whose income is modeled by a Pareto distribution. Since the Gini coefficient is calculated based on an equivalized household income, all individuals of a household in the upper tail receive the same value.

```
R> sim <- function(x, k) {
+     x <- x[!is.na(x$eqIncome), ]
+     g <- gini(x$eqIncome, x$.weight)$value
+     eqIncHill <- fitPareto(x$eqIncome, k = k,
+         method = "thetaHill", groups = x$hid)
+     gHill <- gini(eqIncHill, x$.weight)$value
+     eqIncPDC <- fitPareto(x$eqIncome, k = k,
+         method = "thetaPDC", groups = x$hid)
+     gPDC <- gini(eqIncPDC, x$.weight)$value
+     c(standard = g, Hill = gHill, PDC = gPDC)
+ }
```

This function is used in the following examples, which are designed to exhibit the strengths of the framework. In order to change from one simulation design to another, all there is to do is to define or modify control objects and supply them to the function `runSimulation()`.

2.1. Basic simulation design

In this basic simulation design, 100 samples of 1500 households are drawn using simple random sampling. Note that the `setup()` function is not used to permanently store the samples in an object. This is simply not necessary, since the population is rather small and the

sampling method is straightforward. Furthermore, the Pareto distribution is fitted to the 175 households with the largest equivalized income.

```
R> set.seed(12345)
R> sc <- SampleControl(grouping = "hid", size = 1500, k = 100)
R> results <- runSimulation(eusilcP, sc, fun = sim, k = 175)
```

In order to inspect the simulation results, methods for several frequently used generic functions are implemented. Besides `head()`, `tail()` and `summary()` methods, a method for computing summary statistics with `aggregate()` is available. By default, the mean is used as summary statistic. Moreover, the `plot()` method selects a suitable graphical representation of the simulation results automatically. A reference line for the true value can thereby be added as well.

```
R> head(results)

  Run Sample standard     Hill      PDC
1   1      1 26.69834 26.79820 27.62520
2   2      2 26.29901 25.98625 26.84920
3   3      3 26.18608 26.24073 27.62158
4   4      4 26.07079 26.06536 27.05907
5   5      5 26.09216 26.22237 25.57402
6   6      6 27.09757 26.93235 25.33397

R> aggregate(results)

standard     Hill      PDC
26.67425 26.71005 26.68249

R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

Figure 1 shows the resulting box plots of the simulation results for the basic simulation design. While the PDC estimator comes with larger variability, all three methods are on average quite close to the true population value. This is also an indication that the choice of the number of households for fitting the Pareto distribution is suitable.

2.2. Using stratified sampling

The most frequently used sampling designs in official statistics are implemented in **simFrame**. In order to switch to another sampling design, only the corresponding control object needs to be changed. In this example, stratified sampling by region is performed. The sample sizes for the different strata are specified by using a vector for the slot `size` of the control object.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 100)
R> results <- runSimulation(eusilcP, sc, fun = sim, k = 175)
```

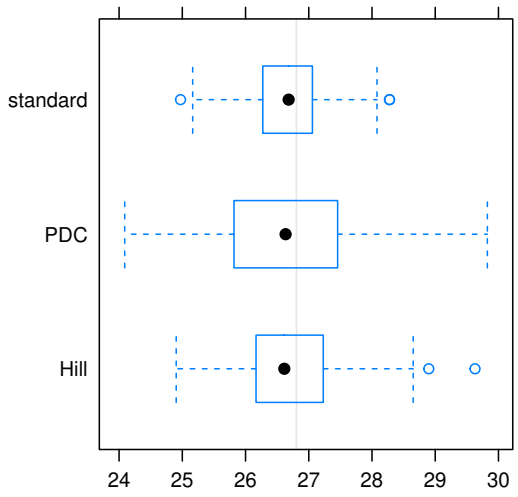

Figure 1: Simulation results for the basic simulation design.

As before, the simulation results are inspected with `head()` and `aggregate()`. A plot of the simulation results is produced as well.

```
R> head(results)

  Run Sample standard     Hill      PDC
1   1      1 26.52834 25.72292 26.12931
2   2      2 27.00022 27.38283 26.29495
3   3      3 27.17566 26.63600 28.42359
4   4      4 26.52469 26.75186 26.41880
5   5      5 26.83442 26.78308 26.03441
6   6      6 27.38513 27.10651 26.24364

R> aggregate(results)

standard     Hill      PDC
26.82340 26.82473 26.83603

R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

Figure 2 contains the plot of the simulation results for the simulation design with stratified sampling. The results are very similar to those from the basic simulation design with simple random sampling. On average, all three investigated methods are quite close to the true population value.

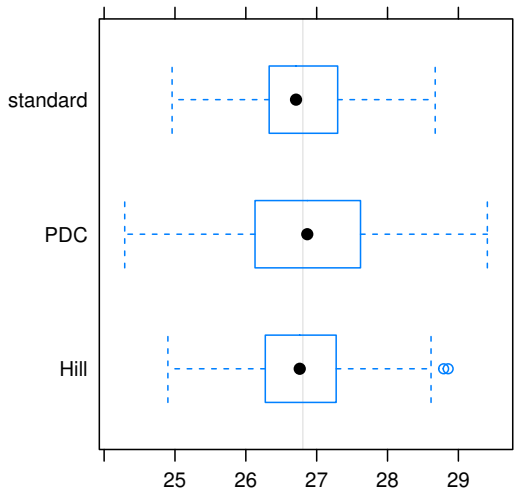

Figure 2: Simulation results for the simulation design with stratified sampling.

2.3. Adding contamination

When evaluating robust methods in simulation studies, contamination needs to be added to the data to study the influence of these outliers on the robust estimators and their classical counterparts. In **simFrame**, contamination is specified by defining a control object. Various contamination models are thereby implemented in the framework. Keep in mind that the term *contamination* is used in a technical sense here (see Alfons *et al.* 2010d; Alfons, Templ, and Filzmoser 2010c, for an exact definition) and that contamination is modeled as a two step process (see also Béguin and Hulliger 2008; Hulliger and Schoch 2009). In this example, 0.5% of the households are selected to be contaminated using simple random sampling. The equivalized income of the selected households is then drawn from a normal distribution with mean $\mu = 500\,000$ and standard deviation $\sigma = 10\,000$.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 100)
R> cc <- DCARContControl(target = "eqIncome", epsilon = 0.005,
+     grouping = "hid", dots = list(mean = 500000, sd = 10000))
R> results <- runSimulation(eusilcP, sc,
+     contControl = cc, fun = sim, k = 175)
```

The `head()`, `aggregate()` and `plot()` methods are again used to take a look at the simulation results. Note that a column is added that indicates the contamination level used.

```
R> head(results)
```

```
  Run Sample Epsilon standard     Hill      PDC
1   1      1   0.005 37.28679 29.25113 26.07553
```

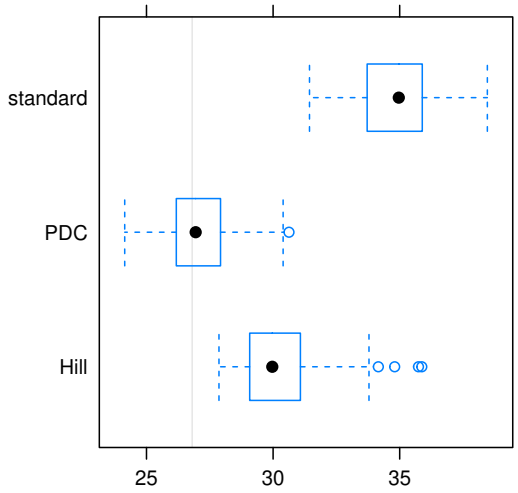

Figure 3: Simulation results for the simulation design with stratified sampling and contamination.

```
2   2      2   0.005 35.60126 32.35751 26.08581
3   3      3   0.005 35.96872 32.39944 28.47702
4   4      4   0.005 32.45425 29.62245 26.23159
5   5      5   0.005 32.68330 30.51681 25.63735
6   6      6   0.005 35.70180 34.80147 26.32128
```

```
R> aggregate(results)

  Epsilon standard     Hill      PDC
1   0.005 34.83684 30.29145 27.13244
```

```
R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

In Figure 3, the resulting box plots are presented. The figure shows that such a small amount of contamination is enough to completely corrupt the standard estimation of the Gini coefficient. Using the classical Hill estimator to fit the Pareto distribution is still highly influenced by the outliers, whereas the PDC estimator leads to very accurate results.

2.4. Performing simulations separately on different domains

Data sets from official statistics typically contain strong heterogeneities, therefore indicators are usually computed for subsets of the data as well. Hence it is often of interest to investigate the behavior of indicators on different subsets in simulation studies. In **simFrame**, this can be done by simply specifying the `design` argument of the function `runSimulation()`. In

the case of extending the example from the previous section, the framework then splits the samples, inserts contamination into each subset and calls the supplied function for these subsets automatically. With bespoke R code, the user would need to take care of this with a loop-like structure such as a `for` loop or a function from the `apply` family.

In the following example, the simulations are performed separately for each gender. It should be noted that the value of `k` for the Pareto distribution is thus changed to 125. This is the same as Example 6.1 from Alfons *et al.* (2010d), except that a control object for sampling is supplied to `runSimulation()` instead of setting up the samples beforehand and storing them in an object.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 100)
R> cc <- DCARContControl(target = "eqIncome", epsilon = 0.005,
+     grouping = "hid", dots = list(mean = 500000, sd = 10000))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     design = "gender", fun = sim, k = 125)
```

Below, the results are inspected using `head()` and `aggregate()`. The `aggregate()` method thereby computes the summary statistic for each subset automatically. Also the `plot()` method displays the results for the different subsets in different panels by taking advantage of the **lattice** system (Sarkar 2008, 2010). In order to compute the true values for each subset, the function `simSapply()` is used.

```
R> head(results)
```

```
  Run Sample Epsilon gender standard     Hill      PDC
1   1      1   0.005   male 33.61311 27.42126 24.97945
2   1      1   0.005 female 41.22478 31.97406 28.12122
3   2      2   0.005   male 35.18056 28.05930 24.49684
4   2      2   0.005 female 39.52626 33.87831 27.54628
5   3      3   0.005   male 34.48228 30.41719 26.05132
6   3      3   0.005 female 33.60189 30.64527 27.17976
```

```
R> aggregate(results)
```

```
  Epsilon gender standard     Hill      PDC
1   0.005   male 33.35098 29.15329 26.36809
2   0.005 female 35.51221 31.38108 28.02747
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

The resulting plots are shown in Figure 4, which is the same as Figure 2 in Alfons *et al.* (2010d). Clearly, the PDC estimator leads to excellent results for both subsets, while the two classical approaches are in both cases highly influenced by the outliers.

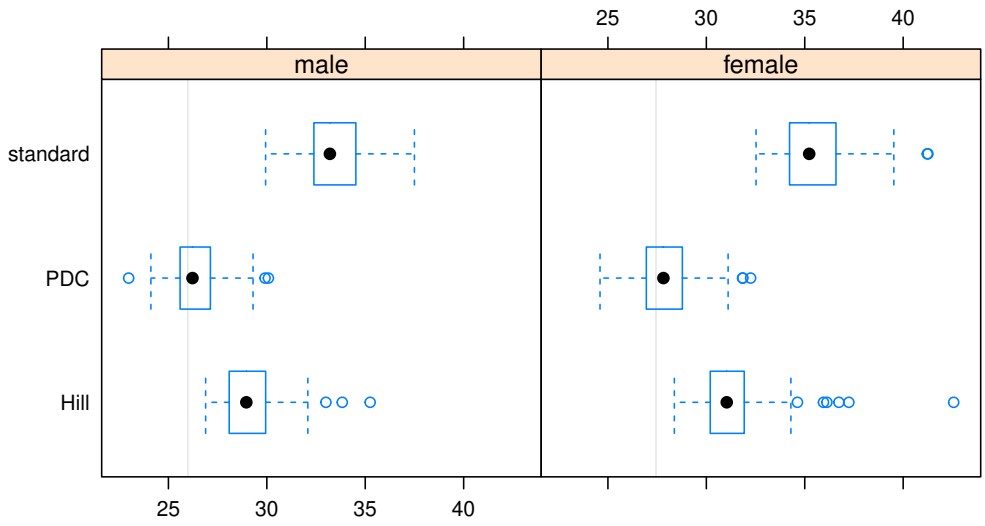

Figure 4: Simulation results for the simulation design with stratified sampling, contamination and performing the simulations separately for each gender.

2.5. Using multiple contamination levels

To get a more complete picture of the behavior of robust methods, more than one level of contamination is typically investigated in simulation studies. The only necessary modification of the code is to use a vector of contamination levels as the slot `epsilon` of the contamination control object. In this example, the contamination level is varied from 0% to 1% in steps of 0.25%. With bespoke R code, the user would have to add another loop-like structure to the code and collect the results in a suitable data structure. In **simFrame**, this is handled internally by the framework.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 100)
R> cc <- DCARContControl(target = "eqIncome",
+     epsilon = c(0, 0.0025, 0.005, 0.0075, 0.01),
+     dots = list(mean = 500000, sd = 10000))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     design = "gender", fun = sim, k = 125)
```

The simulation results are inspected as usual. Note that the `aggregate()` method in this case returns values for each combination of contamination level and gender.

```
R> head(results)

  Run Sample Epsilon gender standard     Hill      PDC
1   1      1  0.0000   male 25.75505 25.04945 24.16866
```

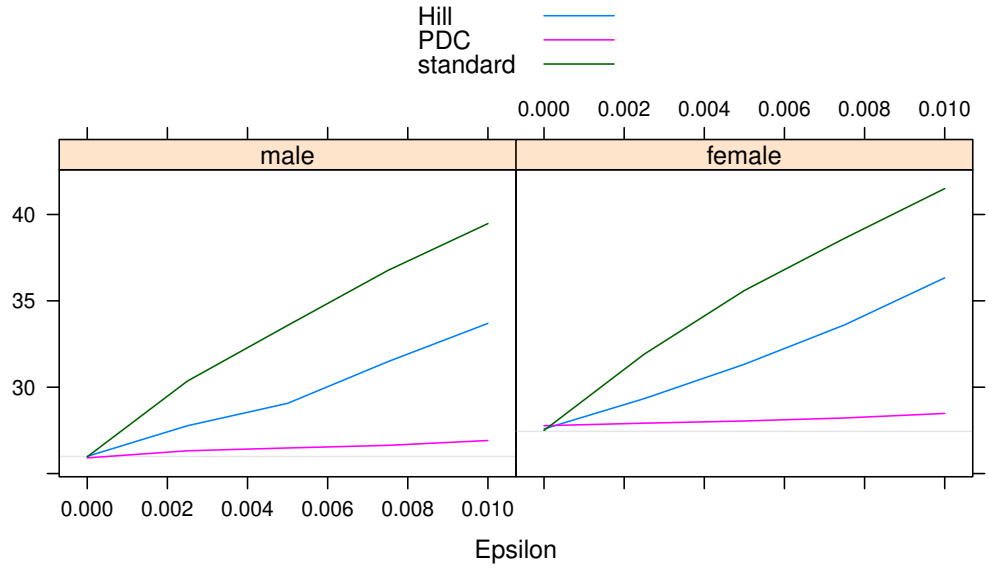

Figure 5: Simulation results for the simulation design with stratified sampling, multiple contamination levels and performing the simulations separately for each gender.

```
2   1      1  0.0000 female 27.16174 27.40417 26.01243
3   2      1  0.0025   male 30.03427 27.05686 24.01423
4   2      1  0.0025 female 31.84312 27.32846 26.21408
5   3      1  0.0050   male 33.82175 29.54768 25.49420
6   3      1  0.0050 female 34.88479 29.43231 26.44052
```

```
R> aggregate(results)
```

```
   Epsilon gender standard     Hill      PDC
1   0.0000   male 25.97113 25.99579 25.90674
2   0.0025   male 30.35443 27.76877 26.31829
3   0.0050   male 33.57469 29.06483 26.48133
4   0.0075   male 36.75641 31.47556 26.63262
5   0.0100   male 39.47215 33.69364 26.91326
6   0.0000 female 27.49801 27.58590 27.77009
7   0.0025 female 31.91151 29.33766 27.91714
8   0.0050 female 35.59702 31.32751 28.04418
9   0.0075 female 38.61865 33.60616 28.21860
10  0.0100 female 41.49725 36.32954 28.47958
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

If multiple contamination levels are used in a simulation study, the `plot()` method for the simulation results no longer produces box plots. Instead, the average results are plotted

against the corresponding contamination levels, as shown in Figure 5. The plots show how the classical estimators move away from the references line as the contamination level increases, while the values obtained with the PDC estimator remain quite accurate.

2.6. Inserting missing values

Survey data almost always contain a considerable amount of missing values. In close-to-reality simulation studies, the variability due to missing data therefore needs to be considered. Three types of missing data mechanisms are commonly distinguished in the literature (e.g., Little and Rubin 2002): missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR). All three missing data mechanisms are implemented in the framework.

In the following example, missing values are inserted into the equivalized household income of non-contaminated households with MCAR, i.e., the households whose values are going to be set to `NA` are selected using simple random sampling. In order to compare the scenario without missing values to a scenario with missing values, the missing value rates 0% and 5% are used. In the latter case, the missing values are simply disregarded for fitting the Pareto distribution and estimating the Gini coefficient. Furthermore, the number of samples is reduced to 50 and only the contamination levels 0%, 0.5% and 1% are investigated to keep the computation time of this motivational example low.

With **simFrame**, only a control object for missing data needs to be defined and supplied to `runSimulation()`, the rest is done automatically by the framework. To apply these changes to a simulation study implemented with bespoke R code, yet another loop-like structure for the different missing value rates as well as changes in the data structure for the simulation results would be necessary.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 50)
R> cc <- DCARContControl(target = "eqIncome",
+     epsilon = c(0, 0.005, 0.01), dots = list(mean = 500000, sd = 10000))
R> nc <- NAControl(target = "eqIncome", NArate = c(0, 0.05))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     NAControl = nc, design = "gender", fun = sim, k = 125)
```

As always, the `head()`, `aggregate()` and `plot()` methods are used to take a look at the simulation results. It should be noted that a column is added to the results that indicates the missing value rate used and that `aggregate()` in this example returns a value for each combination of contamination level, missing value rate and gender.

```
R> head(results)
```

```
  Run Sample Epsilon NArate gender standard     Hill      PDC
1   1      1   0.000   0.00   male 25.75505 25.69991 24.27128
2   1      1   0.000   0.00 female 27.16174 27.05703 26.63831
3   2      1   0.000   0.05   male 25.71583 25.22013 23.67258
4   2      1   0.000   0.05 female 27.15366 27.46721 26.27870
```

```
5   3      1   0.005   0.00   male 34.00351 29.12058 25.07296
6   3      1   0.005   0.00 female 35.08796 31.13506 27.34334

R> aggregate(results)

   Epsilon NArate gender standard     Hill      PDC
1    0.000   0.00   male 26.10984 26.34945 26.27024
2    0.005   0.00   male 33.70130 29.24899 26.85938
3    0.010   0.00   male 39.61988 33.54584 27.16191
4    0.000   0.05   male 26.09869 26.18319 26.05483
5    0.005   0.05   male 34.03376 29.15478 26.56508
6    0.010   0.05   male 40.18770 34.79180 26.95822
7    0.000   0.00 female 27.61056 27.60727 27.72856
8    0.005   0.00 female 35.70845 31.45121 28.13061
9    0.010   0.00 female 41.63594 37.11569 28.68557
10   0.000   0.05 female 27.62300 27.70838 27.79562
11   0.005   0.05 female 36.08729 31.21939 28.23553
12   0.010   0.05 female 42.21347 36.76705 28.51822

R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

If multiple contamination levels and multiple missing value rates are used in the simulation study, conditional plots are produced by the `plot()` method for the simulation results. Figure 6 shows the resulting plots for this example. The bottom panels illustrate the scenario without missing values, while the scenario with 5% missing values is displayed in the top panels. In this case, there is not much of a difference in the results for the two scenarios.

2.7. Parallel computing

Statistical simulation is an *embarrassingly parallel* procedure, hence parallel computing can drastically reduce the computational costs. Since version 0.5.0, parallel computing in **simFrame** is implemented using **parallel**, which is part of the R base distribution since version 2.14.0. Only minimal additional programming effort is required to adapt the code from the previous example: to initialize the computer cluster, to ensure that all packages and objects are available on each worker process, to use the function `clusterRunSimulation()` instead of `runSimulation()` and to stop the computer cluster after the simulations. In addition, random number streams (e.g., L'Ecuyer, Simard, Chen, and Kelton 2002) should be used instead of the built-in random number generator.

```
R> cl <- makeCluster(2, type="PSOCK")
R> clusterEvalQ(cl, {
+         library("simFrame")
+         library("laeken")
+         data("eusilcP")
+     })
R> clusterSetRNGStream(cl, iseed=12345)
```

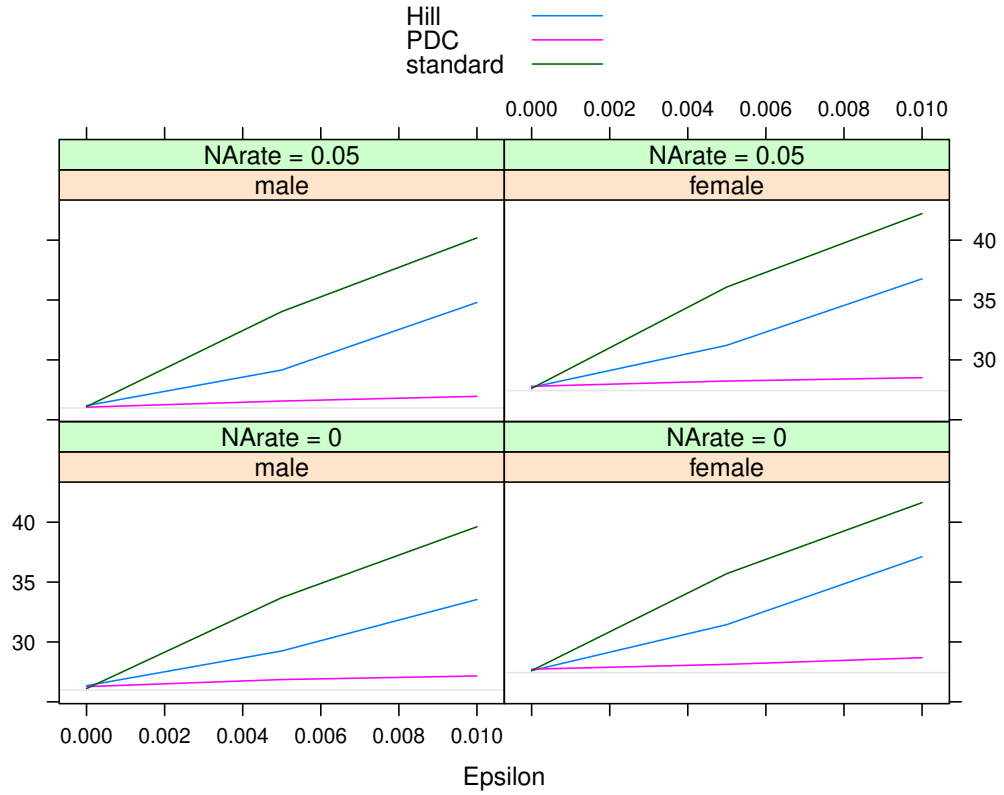

Figure 6: Simulation results for the simulation design with stratified sampling, multiple contamination levels, multiple missing value rates and performing the simulations separately for each gender.

```
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100), k = 50)
R> cc <- DCARContControl(target = "eqIncome",
+     epsilon = c(0, 0.005, 0.01), dots = list(mean = 500000, sd = 10000))
R> nc <- NAControl(target = "eqIncome", NArate = c(0, 0.05))
R> clusterExport(cl, c("sc", "cc", "nc", "sim"))
R> results <- clusterRunSimulation(cl, eusilcP, sc,
+     contControl = cc, NAControl = nc, design = "gender",
+     fun = sim, k = 125)
R> stopCluster(cl)
```

When the parallel computations are finished and the simulation results are obtained, they can be inspected as usual.

```
R> head(results)
```

```
  Run Sample Epsilon NArate gender standard     Hill      PDC
1   1      1   0.000   0.00   male 25.75470 25.13417 24.04176
```

```
2   1      1   0.000   0.00 female 28.49954 29.03969 25.88112
3   2      1   0.000   0.05   male 26.03934 26.01137 23.71969
4   2      1   0.000   0.05 female 28.43866 28.14297 25.77251
5   3      1   0.005   0.00   male 33.35789 29.58219 24.26642
6   3      1   0.005   0.00 female 36.32355 31.15428 26.23692
```

```
R> aggregate(results)
```

```
   Epsilon NArate gender standard     Hill      PDC
1    0.000   0.00   male 25.94199 25.95362 25.87304
2    0.005   0.00   male 33.55200 29.40530 26.26338
3    0.010   0.00   male 39.39763 33.44937 26.52247
4    0.000   0.05   male 25.93731 26.07354 25.76448
5    0.005   0.05   male 33.91553 29.69189 26.30017
6    0.010   0.05   male 39.97217 33.17304 26.33333
7    0.000   0.00 female 27.42857 27.47925 27.38612
8    0.005   0.00 female 35.45745 30.87602 27.84644
9    0.010   0.00 female 41.45508 35.94360 28.74434
10   0.000   0.05 female 27.42720 27.45564 27.32308
11   0.005   0.05 female 35.84247 31.28447 28.03765
12   0.010   0.05 female 42.00047 36.30806 28.24898
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

Figure 7 shows the simulation results obtained with parallel computing. The plots are, of course, very similar to the plots for the previous example in Figure 6, since the design of the simulation studies is the same.

3. Conclusions

In this paper, the use of the R package **simFrame** for different simulation designs has been demonstrated in a practical application. The full functionality of the framework for design-based simulation has been presented in various code examples. These examples showed that the framework allows researchers to make use of a wide range of simulation designs with only a few lines of code. In order to switch from one simulation design to another, only control objects need to be defined or modified. Even moving from basic to highly complex designs therefore requires only minimal changes to the code. With bespoke R code, such modifications would often need a considerable amount of programming. Furthermore, parallel computing with **simFrame** can easily be done based on package **parallel**.

Besides the functionality for carrying out simulation studies, methods for several frequently used generic functions are available for inspecting or summarizing the simulation results. Most notably, a suitable plot method of the simulation results is selected automatically depending on their structure.

Due to this flexibility, **simFrame** is widely applicable for gaining insight into the quality of statistical methods and is a valuable addition to a researcher's toolbox.

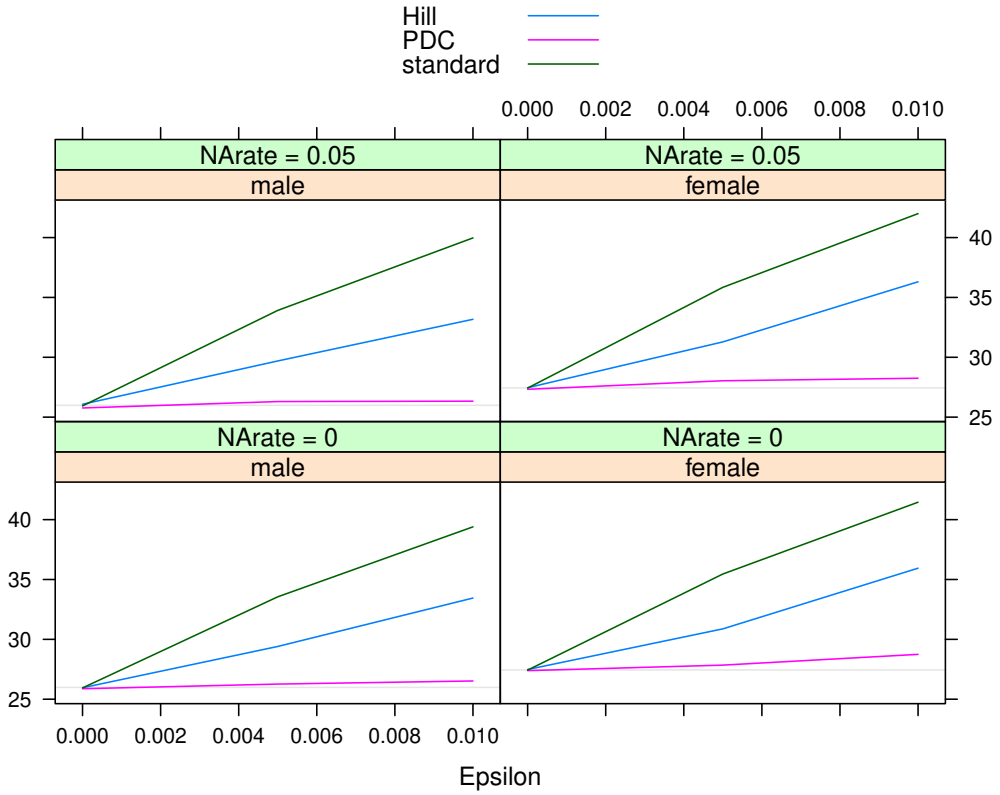

Figure 7: Simulation results obtained by parallel computing for the simulation design with stratified sampling, multiple contamination levels, multiple missing value rates and performing the simulations separately for each gender.

Acknowledgments

This work was partly funded by the European Union (represented by the European Commission) within the 7th framework programme for research (Theme 8, Socio-Economic Sciences and Humanities, Project AMELI (Advanced Methodology for European Laeken Indicators), Grant Agreement No. 217322). Visit http://ameli.surveystatistics.net for more information on the project.

References

Alfons A (2013). ***simFrame****: Simulation Framework*. R package version 0.5.4, URL https://CRAN.R-project.org/package=simFrame.

Alfons A, Holzer J, Templ M (2010a). ***laeken****: Laeken Indicators for Measuring Social Cohesion*. R package version 0.1.3, URL https://CRAN.R-project.org/package=laeken.

Alfons A, Kraft S (2010). ***simPopulation****: Simulation of Synthetic Populations for Sur-*

veys based on Sample Data. R package version 0.2, URL https://CRAN.R-project.org/package=simPopulation.

Alfons A, Kraft S, Templ M, Filzmoser P (2010b). "Simulation of Synthetic Population Data for Household Surveys with Application to EU-SILC." *Research Report CS-2010-1*, Department of Statistics and Probability Theory, Vienna University of Technology.

Alfons A, Templ M, Filzmoser P (2010c). "Contamination Models in the R Package **simFrame** for Statistical Simulation." In S Aivazian, P Filzmoser, Y Kharin (eds.), *Computer Data Analysis and Modeling: Complex Stochastic Data and Systems*, volume 2, pp. 178–181. Minsk. ISBN 978-985-476-848-9.

Alfons A, Templ M, Filzmoser P (2010d). "An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**." *Journal of Statistical Software*, **37**(3), 1–36. doi:10.18637/jss.v037.i03.

Alfons A, Templ M, Filzmoser P, Holzer J (2010e). "A Comparison of Robust Methods for Pareto Tail Modeling in the Case of Laeken Indicators." In C Borgelt, G González-Rodríguez, W Trutschnig, M Lubiano, M Gil, P Grzegorzewski, O Hryniewicz (eds.), *Combining Soft Computing and Statistical Methods in Data Analysis*, volume 77 of *Advances in Intelligent and Soft Computing*, pp. 17–24. Springer-Verlag, Heidelberg. ISBN 978-3-642-14745-6.

Béguin C, Hulliger B (2008). "The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data." *Survey Methodology*, **34**(1), 91–103.

EU-SILC (2004). "Common Cross-Sectional EU Indicators based on EU-SILC; the Gender Pay Gap." *EU-SILC 131-rev/04*, Working group on Statistics on Income and Living Conditions (EU-SILC), Eurostat, Luxembourg.

Hill B (1975). "A Simple General Approach to Inference about the Tail of a Distribution." *The Annals of Statistics*, **3**(5), 1163–1174.

Hulliger B, Schoch T (2009). "Robust Multivariate Imputation with Survey Data." 57th Session of the International Statistical Institute, Durban.

Kleiber C, Kotz S (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*. John Wiley & Sons, Hoboken. ISBN 0-471-15064-9.

L'Ecuyer P, Simard R, Chen E, Kelton W (2002). "An Object-Oriented Random-Number Package with Many Long Streams and Substreams." *Operations Research*, **50**(6), 1073–1075.

Little R, Rubin D (2002). *Statistical Analysis with Missing Data*. 2nd edition. John Wiley & Sons, New York. ISBN 0-471-18386-5.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL https://www.R-project.org.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York. ISBN 978-0-387-75968-5.

Sarkar D (2010). ***lattice**: Lattice Graphics*. R package version 0.19-13, URL https://CRAN.R-project.org/package=lattice.

Vandewalle B, Beirlant J, Christmann A, Hubert M (2007). "A Robust Estimator for the Tail Index of Pareto-Type Distributions." *Computational Statistics & Data Analysis*, **51**(12), 6252–6268.

Affiliation:

Andreas Alfons
Erasmus School of Economics
Erasmus University Rotterdam
Burgemeester Oudlaan 50
3062PA Rotterdam, Netherlands
E-mail: alfons@ese.eur.nl
URL: https://personal.eur.nl/alfons/