EUROPEAN COMMISSION

Community Research

SEVENTH FRAMEWORK PROGRAMME

AMELI

Advanced Methodology for European Laeken Indicators

Deliverable 6.1

The AMELI Simulation Study

Version: 2011

Andreas Alfons, Jan Pablo Burgard, Peter Filzmoser, Beat Hulliger, Jan-Philipp Kolb, Stefan Kraft, Ralf Münnich, Tobias Schoch and Matthias Templ

The project **FP7–SSH–2007–217322 AMELI** is supported by European Commission funding from the Seventh Framework Programme for Research.

http://ameli.surveystatistics.net/

Contributors to deliverable 6.1

Chapter 1: Jan-Philipp Kolb and Ralf Münnich University of Trier.

Chapter 2: Andreas Alfons and Peter Filzmoser and Stefan Kraft and Matthias Templ, Vienna University of Technology; Jan-Philipp Kolb and Ralf Münnich University of Trier.

Chapter 3: Andreas Alfons and Peter Filzmoser and Stefan Kraft and Matthias Templ, Vienna University of Technology.

Chapter 4: Pablo Burgard and Ralf Münnich, University of Trier; Beat Hulliger and Tobias Schoch, University of Applied Sciences North-Western Switzerland.

Chapter 5: Andreas Alfons and Peter Filzmoser and Stefan Kraft and Matthias Templ, Vienna University of Technology; Jan-Philipp Kolb and Ralf Münnich, University of Trier.

Main responsibility

Andreas Alfons, Matthias Templ, Vienna University of Technology

Evaluators

Internal expert: Risto Lehtonen, University of Helsinki.

Aim and objectives of deliverable 6.1

The aim of work package 6 is to yield a basis for a comparative simulation study in. This shall allow at investigating the methodology developed in WPs 2, 3, and 4 in a close to reality environment based on realistic datasets. The present deliverable describes the design of simulation studies in this context and the applied simulation framework within the project.

Contents

Chapter 1

Introduction

The aim of the AMELI project is it to develop advanced methodology for measuring poverty and social cohesion in Europe. To do so, extensive simulation is used to evaluate the developed methodology. However, in larger research projects such as AMELI, with many partners from different institutions, guidelines regarding, e.g., simulation designs, survey designs, contamination, missing data models or evaluation criteria are required. Otherwise the obtained results may be incomparable, thus making it impossible to draw meaningful conclusions.

Member states of the European Union and other European countries use the so-called *indicators on social exclusion and poverty* to measure poverty and social cohesion. The subset of these indicators based on EU-SILC (*European Union Statistics on Income and Living Conditions*) is of particular interest for the AMELI project. Within the project, the methodology for estimating the indicators on social exclusion and poverty is improved directly by using robust approaches or small area estimation, but also by enhancing the data quality with, e.g. appropriate imputation or outlier detection methods. In addition, suitable variance estimation methodology is developed.

In Workpackage 6, the methods developed in Workpackages 2, 3 and 4 are investigated and evaluated. The basis of the simulation study are synthetic populations for selected European countries, which are generated from original EU-SILC samples provided by Eurostat. Hence, Workpackage 5 has an impact on Workpackage 6, as it is necessary to be aware of the peculiarities of the data to generate high quality synthetic populations. Using the results from Workpackage 6, a thorough analysis of the impact of different situations on different procedures involved in the estimation of the indicators on social exclusion and poverty is conducted in Workpackage 7. Additionally, suitable visualization methods for simulation results are developed in Workpackage 8. The output from Workpackages 7 and 8 is in turn used by Workpackage 9 to give best practice recommendations in order to support policy makers.

The DACSEIS project has a strong impact on the AMELI project regarding simulation and acts as a starting point for further development. Münnich et al. (2003b) describe the design of the simulation studies within the DACSEIS project.

Most notably, the simulation studies in the AMELI project are based on repeatedly drawing samples from finite populations. Since registers containing population data are not available for most countries, synthetic populations need to be generated from samples. For detailed information on the generation of the synthetic population data used in the simulations, the reader is referred to Alfons et al. (2011d).

The design of the simulation study should be as close to reality as possible, to be able to give best practice recommendations for the real life survey. Besides having realistic populations, this includes using the true sampling designs, as well as realistic outlier and missing data models. An important question is the treatment of contamination and non-response. Including those in the population is the most realistic scenario, but results in an unpredictable number of outliers and missing values in the samples. On the one hand only including them in the samples provides maximum control over the amount of outliers and missing values, which is sometimes required for evaluating their influence on the methodology, but on the other hand this approach does not reflect the real processes. This trade-off between close to reality simulation designs and more practical approaches needs to be considered.

A general introduction to simulation studies is given in section 2.1. Afterwards the design of the classical simulations within the AMELI project is presented, before the general design of the simulation studies concerning outlyingness and missingness is addressed in section 2.3. Chapter 3 then describes the R package `simFrame` (Alfons et al., 2010; Alfons, 2011), which is a general framework for statistical simulation. Other simulation engines used within the AMELI project are discussed in Chapter 4. Finally, Chapter 5 gives a summary of the developments.

Chapter 2

Design-based simulations

Simulations are the attempt to adjust real life situations in a controllable environment, so that results can be reproduced. Thus, if simulation methods are applied, it is the target to reconstruct real life situations, but the starting points can be different. If it is not exactly clear which starting point is the most realistic one, therefore a scenario analysis might be helpful. Although one special scenario can be very unlikely, it is sometimes useful to know what can happen in an extreme situation. The sample is drawn only once but there are many values to be estimated. Without a simulation it is not possible to evaluate the quality of an estimator.

Often different factors can influence estimations. It is important for the validity of simulations to include the structures of the real world into the basis, the synthetic universe and into the process, in other words the simulation. Here, the manner how a sample is drawn or the type of observations, which can get into the sample, is meant for instance. As an example, extreme observations can influence the mean income very much for example. The basic principle of a design-based simulation is to evaluate the estimators in the context of one synthetic population which is assumed to be the *real* population and to apply realistic sampling methods on this population.

The general concept of the simulation study should be as close to reality as possible. Starting from population data, samples are drawn repeatedly using the real life sampling methods. In each simulation run, the quantities of interest (in most cases, the indicators on social exclusion and poverty or their variances) are estimated from the corresponding sample. The results of all simulation runs are then combined to form a distribution. Finally, certain evaluation criteria are used to compare the estimates with the true values. Different simulation studies are weighted against each other on the basis of evaluation and quality criteria.

2.1 General introduction

As the word design-based simulations suggests, realistic survey designs take center stage. The next step is then to look at results which stem from different starting points. Therefore, comparable criteria for the evaluation have to be found whenever possible.

The simulation set-up within the AMELI project is based on experiences form the FP5 projects EUREDIT and DACSEIS (see for example Hulliger 2011 for the introduction of missingness and contamination mechanisms). In most of the simulations within the AMELI project design-based simulations are applied. This implies, that draws from a previously generated population are realized.

2.2 Classical studies

The European statistics Code of Practice which was published in 2005 has the objective to improve the confidence in the integrity of European official statistics. It should ameliorate the possibility to compare statistics, guaranty that only the most relevant statistical concepts are applied and that the information is complete. Therefore it is necessary to evaluate the process of data acquisition up to the computation of statistics on poverty and social exclusion.

It is the target of the AMELI project to support this evaluation. One important part of the code of practice is the demand for information about the accuracy of estimations. Here it is the question how to measure the accuracy in the case of the indicators on social exclusion and poverty. Normally, two types of errors affect the accuracy of the statistics. The first one is the sampling error which can be represented by the standard error, the confidence interval or the effective sample size n_{eff}. The second one is the non-sampling error which do result from non-response and frame errors.

Therefore, one focus of the simulations within the AMELI process was to address variance estimation of indicators on social exclusion and poverty, on national and local scale, like NUTS 2, by using EU-SILC data. Because of the nonlinear nature of these indicators, adequate variance estimation methods like resampling and linearization techniques are employed. To empirically explore the quality and accuracy measurement of indicators on social exclusion and poverty a Monte Carlo study, based on the AMELIA data set, is conducted. In this Monte Carlo studies, single-stage and two-stage cluster designs are applied which are further described in deliverable 7.1.

The focus thus lies, in contrast to the simulation studies described in section 2.3.1, not so much on outlier scenarios but more on the realization of close to reality survey designs. The whole simulation study within the AMELI project is complemented here with the interplay between complex survey designs, which are applied in EU-SILC, and the estimation of point and variance estimators for small regional or contentual sub-groups. It is not easy to control simultaneously for all influences (outliers, missing values for small areas), the computational costs are to high to control for this. Therefore, the AMELI team decided to run different simulations which do complement each other.

In figure 2.1 an overview of the situation at hand is given. The true population data is indicated with the table which is visible at the upper left hand side of the graphic. In reality samples are drawn from this population data and distributions are based on this sample (shown symbolically on the left bottom). However it is only a synthetic image of the true population, influenced by e.g. the EU-SILC samples of 2005 and 2006 and further informations like knowledge about the possible values for statistics on poverty and social exclusion. The samples are then drawn from this synthetic population, with a survey design which is assumed to be as close as possible to reality. The drawing is repeated R-times and the results for the estimators of statistics on poverty and social exclusion and their accuracy are summarized. The results of this procedure should then be comparable to those which would arise from a sample drawn from the real population in Europe.

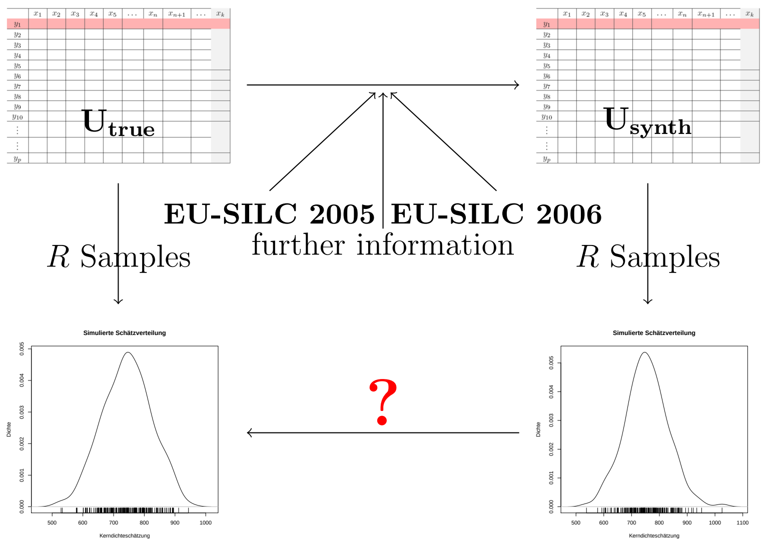

Figure 2.1: Overview general simulations

However distortions can occur because the synthetic population is finite. It is very difficult to foresee extreme situations in reality. This problem gets more serious if results for smaller entities have to be computed. Then outliers can have a big influence on the result.

Salvati et al. (2010) describe design-based simulations as more interesting than model-based simulations. They justify this statement with the fact that design-based simulations constitute a more realistic representation of the small area estimation problem than simulations based on theoretically generated populations. This is due to the fact that differences between the small areas are fixed effectively (cf. Salvati et al. 2010 p. 22).

The fact that a design-based simulation can lead to very astonishing and interesting results is for example shown in Hulliger and Münnich (2006). The multitude of survey designs, scenarios and estimators results in an even higher number of simulation results. This leads to the challenge to handle big amounts of data. Therefore, it is useful to define a simulation framework and a strategy how to get from the starting positon to meaningful results. The focus of simulation studies can be different. It can be the target to test the interplay between the survey design and e.g. small area estimation. Here survey designs are applied and estimates for small areas are produced for these samples.

2.3 Design of the simulation study on outlier and non-response

In Section 2.3.1 general issues regarding the outline of the simulation study on outlyingness and missingness are discussed. While section 2.4 addresses sampling methods, contamination and missing data models are described in Sections 2.4.1 and 2.4.2, respectively. Note that the description of the simulation study in this document is of a general nature. For detailed information of the sampling methods, contamination and missing data models, as well as evaluation criteria used within the AMELI project, the reader is referred to Alfons et al. (2011a).

2.3.1 The simulation design for Outlier and missingness simulations

Before the design of the simulation study is discussed, the notion of *representative* and *nonrepresentative* outliers (Chambers, 1986) is introduced for better understanding of the role of outliers in survey statistics. *Representative* outliers in a sample are observations whose values have been recorded correctly and cannot be regarded as unique in the population. Therefore, these observations contain some relevant information for estimating quantities of interest. *Nonrepresentative* outliers, on the other hand, contain values that are either incorrect or unique to the specific population element. In the latter case, they need to be considered in the simulation process, i.e., they should be excluded from the estimation of quantities of interest or assigned very low weights. If they exist due to errors in the sample data, they need to be corrected in the editing process, thus this case is not relevant for our purposes.

Concerning outliers and non-response, the most realistic perception of the world is that they exist on the population level. Whether a person has an extremely high income or is not willing to respond to certain questions of a survey does not depend on whether that person is actually in the sample. Hence the most realistic simulation design is to insert contamination and non-response into the population (see Figure 2.2 for a graphical representation). A disadvantage of this approach is that the number of outliers or missing values in the samples is unpredictable.

If properties of the considered estimators is the main focus of a simulation, or if outlier detection methods are investigated, maximum control over the amount of contaminated observations is necessary for a thorough evaluation. The same principle with respect to the amount of missing values applies if the treatment of incomplete data is the main interest. This problem can be solved by adding contamination and non-response to the samples instead of the population. While this approach may not truly reflect the real processes, it may be more practical from a statistician's point of view in the aforementioned situations. Nevertheless, it should be noted that adding contamination and non-response to samples comes with increasing computational costs for an increasing number of simulation runs. Figure 2.3 visualizes the general outline of such a simulation design.

A synthetic population is generated from an original sample such that the properties of the underlying sample are reflected. Hence, also non-response in the population may be

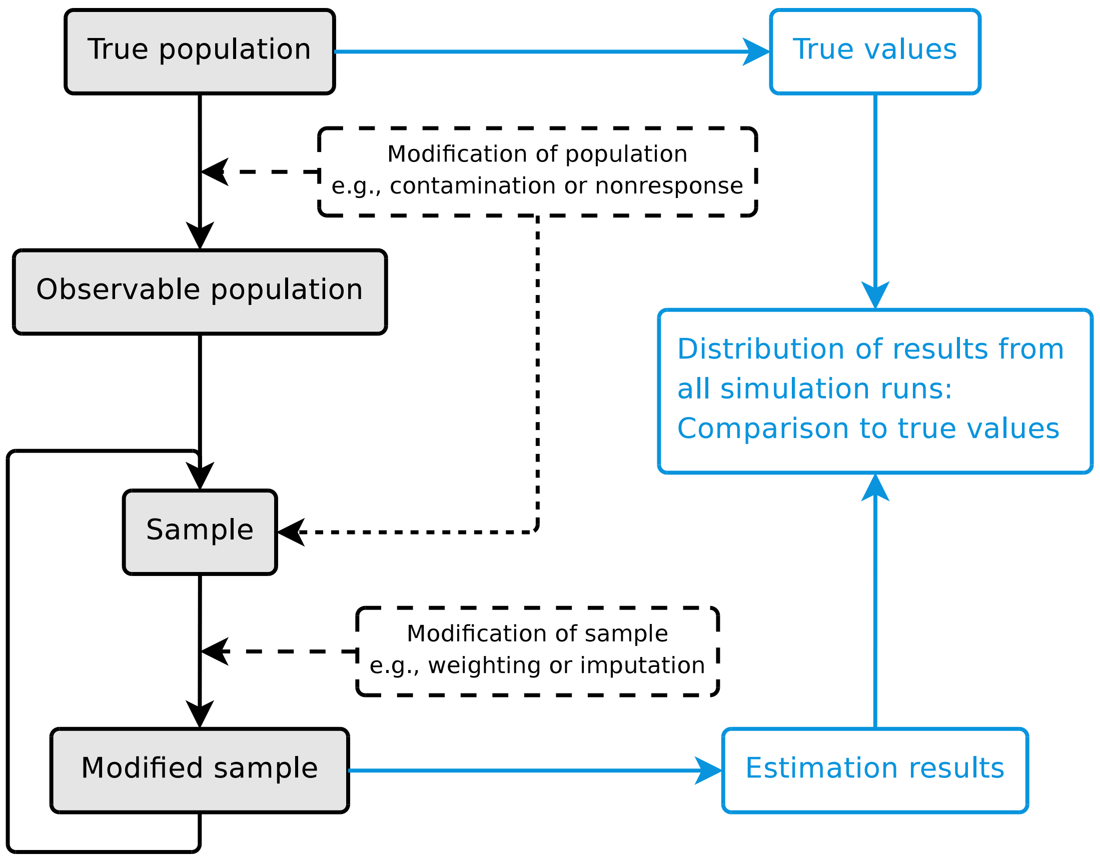
True population
True values
Modification of population
e.g., contamination or nonresponse
Observable population
Distribution of results from
all simulation runs:
Comparison to true values
Sample
Modification of sample
e.g., weighting or imputation
Modified sample
Estimation results

Figure 2.2: Outline of the most realistic simulation design (cf. Münnich et al., 2003b).

generated with a model from the underlying sample (see Section 2.4.2). However, the situation is much more complicated with contamination. While non-response is clearly visible in a sample (a missing value is a missing value is a missing value), the contaminated observations are not known beforehand. Outlier detection methods may be applied, but the results still leave some uncertainty. Even if outliers are detected, it is still necessary to find a model for their distribution, which may or may not depend on the majority of the observations. Simply experimenting with different proportions of outliers and different distributions for the contamination might be a more practical approach. In any case, the simulation study should not be confined to only the most realistic contamination and missing data models. The better the behavior of the developed methodology is known, the better the best practice recommendations are at the end of the day.

2.4 Sampling and weighting

In order to achive a realistic portrayal of the real-life processes in the simulation study, the true sampling methods and weighting schemes should be used for the respective countries. Others may be used additionally to get a more complecte picture of the behavior of the methodology in different situations. However, there is a large number of sampling methods described in the literature regarding survey statistics, so it is necessary to select a few representative methods to be used in the simulations. The effects of simple methods on the

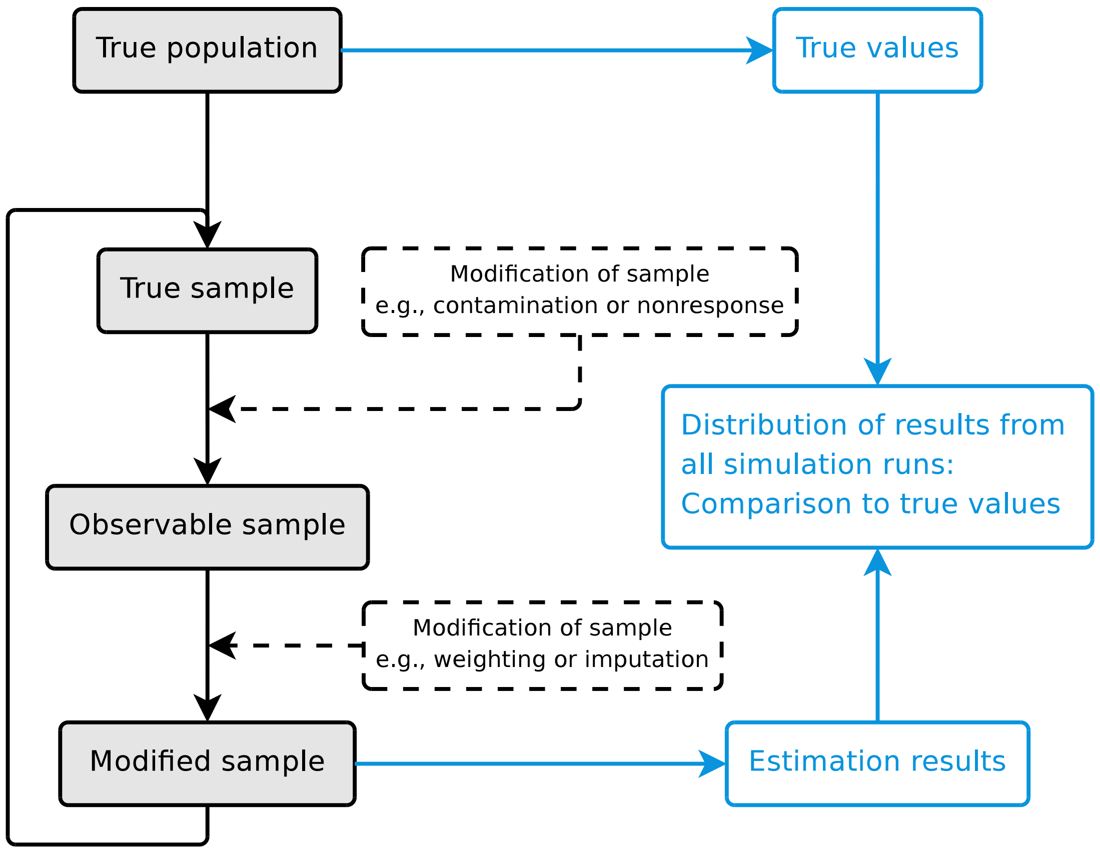

Figure 2.3: Outline of the most practical simulation design for the AMELI project.

outcome might provide some insight, thus simple random sampling should be considered. Nevertheless, unequal probability sampling is frequently used in reality and should be covered in the simulations. Since each household is linked to a region in EU-SILC data, stratified sampling should certainly be investigated. Furthermore, more advanced methods such as multistage sampling or balanced sampling may be of interest. In any case, The designs to be tested should cover the most frequently used in EU-SILC (as reported by WP5).

2.4.1 Contamination models

Outlyingness in simulations should be modeled as a two-step process (Béguin and Hulliger, 2008; Hulliger and Schoch, 2009). The first step is to select observations to be contaminated, the second is to model the distribution of the outliers.

To discuss the process in more detail, let n be the number of observations, p the number of variables, and let $\boldsymbol{x}_i = (x_{i1}, \dots, x_{ip})$, $i = 1, \dots, n$, denote the observations.

1. Let O_i, $i = 1, \dots, n$, be an indicator whether an observation is an outlier ($O_i = 1$) or not ($O_i = 0$). The situation that the probability distribution of O_i does not depend on any other variables, i.e., that

$$P(O_i = 1 | x_{i1}, \dots, x_{ip}) = P(O_i = 1), \qquad i = 1, \dots, n \tag{2.1}$$

may be called *outlying completely at random* (OCAR). If Equation (2.1) is violated, i.e., if the probability distribution of O_i depends on observed information, the situation may be called *outlying at random* (OAR). In the case of EU-SILC data, the probability of outlying income may depend on, e.g., economic status or age. In order to simulate practical situations, OAR should therefore be considered.

2. Once the observations to be contaminated have been fixed, the distribution of the outliers needs to be defined. Again, this distribution may or may not depend on the observed information. Let $I_c := \{i = 1, \dots, n : O_i = 1\}$ be the index set of the observations to be contaminated, and let $\boldsymbol{x}_i^* = (x_{i1}^*, \dots, x_{ip}^*)$ denote the original (i.e., non-contaminated) values of $\boldsymbol{x}_i$, $i \in I_c$. If the distribution does not depend on the original values, i.e., if

$$\boldsymbol{x}_i \sim F(x_1, \dots, x_p), \qquad i \in I_c, \tag{2.2}$$

the outliers may be called *contaminated completely at random* (CCAR). An example is to generate the contamination with a (multivariate) normal distribution with fixed location and scatter. On the other hand, if the distribution depends on the original values, i.e., if

$$\boldsymbol{x}_i \sim F(x_1, \dots, x_p, x_{i1}^*, \dots, x_{ip}^*), \qquad i \in I_c, \tag{2.3}$$

the outliers may be called *contaminated at random* (CAR). A simple example for CAR situations is the scaling of variables with a certain factor, e.g., 100 or 1000. Regarding income data in EU-SILC, it is easily conceivable that the distribution of outliers depends on some underlying information.

Realistic outlier scenarios should be investigated in the simulation study in order to give recommendations for practical applications. Hence, using OAR-NCAR situations is of high interest (cf. the simulation example in BÉGUIN and HULLIGER, 2008). Nevertheless, it is also important to know how the developed methodology behaves in simpler (e.g., OCAR-CCAR) situations.

2.4.2 Missing data models

This part is structured as follows: first, Section 2.4.2 covers different mechanisms that generate missing data. Afterwards, two approaches for handling non-response in simulations are discussed in Section 2.4.2.

Missing data mechanisms

Three important cases of processes responsible for generating missing values are commonly distinguished in the missing data literature (see, e.g., LITTLE and RUBIN, 2002), based on the original ideas by RUBIN (1976).

Let $\boldsymbol{X} = (\boldsymbol{X}_{obs}, \boldsymbol{X}_{miss})$ denote the data, where $\boldsymbol{X}_{obs}$ and $\boldsymbol{X}_{miss}$ are the observed and missing parts, respectively.

Then the missing values are *missing at random* (MAR) if

$$f(\boldsymbol{M}|\boldsymbol{X},\boldsymbol{\phi}) = f(\boldsymbol{M}|\boldsymbol{X}_{obs},\boldsymbol{\phi}), \tag{2.4}$$

i.e., the probability of missingness does not depend on the missing part $\boldsymbol{X}_{miss}$. The important special case of MAR called *missing completely at random* (MCAR) is given if the probability of missingness does not depend on the observed part $\boldsymbol{X}_{obs}$ either, which translates to the equation

$$P(\boldsymbol{X}_{miss}|\boldsymbol{X}) = P(\boldsymbol{X}_{miss}). \tag{2.5}$$

On the other hand, the missing values are said to be *missing not at random* (MNAR) if Equation (2.4) is violated and missingness is in some way related to the outcome variables, i.e., the probability of missingness depends on $\boldsymbol{X}_{miss}$. This can be written as

$$P(\boldsymbol{X}_{miss}|\boldsymbol{X}) = P(\boldsymbol{X}_{miss}|(\boldsymbol{X}_{obs},\boldsymbol{X}_{miss})). \tag{2.6}$$

Hence, in the latter case, the missing values cannot be fully explained by the observed part of the data.

A motivational example for the different missing data mechanisms, which is also a suitable illustration for EU-SILC data, is given in Little and Rubin (2002). Consider two variables *age* and *income*, with missing values in income. If the probability of missingness is the same for all individuals, regardless of their age or income, then the data are MCAR. If the probability that income is missing varies according to the age of the respondent, but does not vary according to the income of respondents with the same age, then the data are MAR. If the probability that income is recorded varies according to income for those with the same age, then the data are MNAR. Note that MNAR *cannot* be detected in practice, as this would require knowledge of the missing values themselves.

The performance of imputation methods usually depends, among other things, on the multivariate structure of the missing values. In the simulation study, realistic missing data scenarios should be investigated. Therefore, the existing real life EU-SILC samples need to be studied. The R package `VIM` (Templ and Filzmoser, 2008; Templ et al., 2011), which has been developed in Workpackage 8, contains visualization tools that allow not only to detect the missing value mechanisms (MAR or MCAR), but also to gain insight into the quality and various other aspects of the data at the same time.

Adding non-response in simulations

Choosing variables in which missing values should be inserted is the first step of adding nonresponse to population or sample data. These variables will in the following be referred to as *target* variables. In the simulations within the AMELI project, the target variables typically are (some of) the income variables.

Random insertion of missing values MCAR situations can be generated by selecting observations for every target variable with simple random sampling. Using unequal probability sampling makes it possible to account for MAR or MNAR situations. The probability weights may thereby be derived from one or more variables. In any case, different missing value rates may be used for the different target variables. More difficult situations require special treatment, though. An important example is that non-response in one variable may only occur for observations with non-response in another variable.

Response propensity models A completely different approach for generating MAR situations is to use a logit model, which can be obtained from the simulation's underlying real life sample. Sensible predictors for income components in EU-SILC are, e.g., age, gender, region and economic status. With the imputed original sample, even MNAR situations may be generated. More details on response propensity models with several applications can be found in Münnich et al. (2003a). The advantage of this method is that it is a more analytical approach for generating realistic non-response scenarios. However, the resulting number of missing values is not predictable. Thus, response propensity models are more suitable for adding non-response to population data than to samples, as having maximum control over the number of missing values is the motivation for the latter.

Chapter 3

The R package simFrame

Simulation studies in research projects such as AMELI require a precise outline. If different partners use, e.g., different contamination or missing data models, the results may be incomparable. A software framework for statistical simulation may thus contribute its share to avoid such problems. For this purpose, the R package `simFrame` (Alfons et al., 2010; Alfons, 2011) has been developed. The object-oriented implementation with S4 classes and methods (Chambers, 1998, 2008a) gives maximum control over input and output and provides clear interfaces for user-defined extensions. Moreover, the framework allows a wide range of simulation designs to be used with only a little programming.

One of the main goals of the AMELI project is to improve the methodology for the indicators on social exclusion and poverty under typical data problems such as outliers and missing data. The package `simFrame` therefore allows to add certain proportions of outliers or non-response. In addition, depending on the structure of the simulation results, an appropriate plot method is selected automatically.

The rest of the chapter is organized as follows. Section 3.1 contains a brief description of the package contents. How to extend the framework is outlined in Section 3.2, while Section 3.3 presents extensions implemented specifically for the AMELI project. Last but not least, an example for the usage of the package is provided in Section 3.4. A more detailed description of the package is given in Alfons et al. (2010).

3.1 Package contents

A cornerstone of the AMELI project is that all software tools are developed in the open source statistical environment R (R Development Core Team, 2011a) and will be made freely available to the statistical community. In addition to being open source, one of the main reasons that R has been chosen as platform is that it includes a well-developed programming language and provides interfaces to many others, including the fast low-level languages C/C++ and Fortran. Most of `simFrame` is implemented with the object-oriented S4 system, with the exception of some utility functions and some C++ code. *Control classes* determine the sequence of events in the simulations and in most cases provide the interfaces for extending the framework (see Section 3.2 for the latter).

Guidelines for the simulation study in AMELI may thus be defined in terms of specific control classes and parameter settings. A slightly simplified UML class diagram (e.g., FOWLER, 2003) of `simFrame` is shown in Figure 3.1.

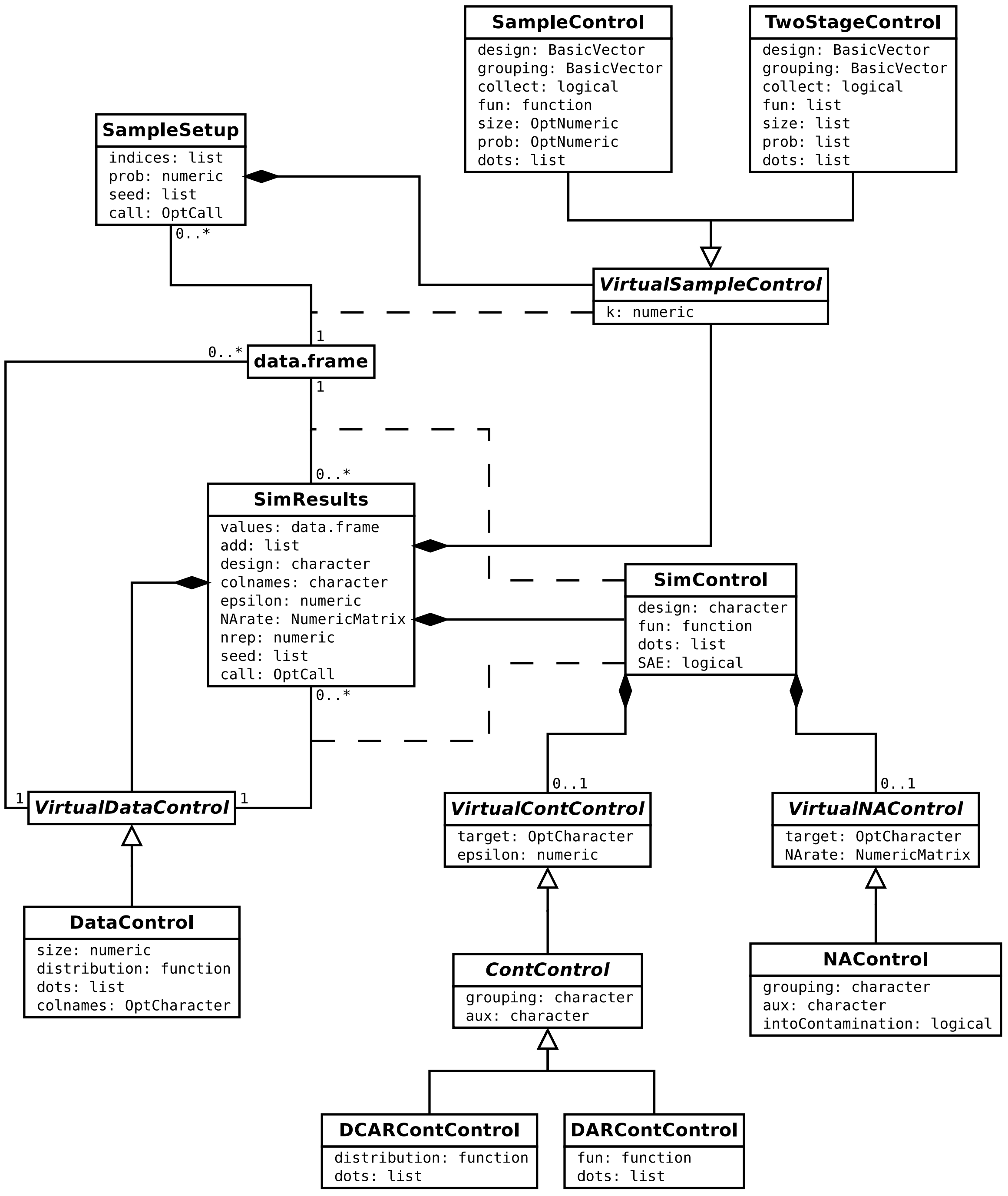

Figure 3.1: Slightly simplified UML class diagram of `simFrame`.

3.1.1 Data handling

To store data, the data frame concept is used in `simFrame`, with some restrictions on the column names. For design-based simulations, the sample weights are stored in a variable called `.weight`. Furthermore, when parts of the data are contaminated, it is necessary to store which observations have been contaminated (e.g., to evaluate outlier detection methods). The corresponding variable is called `.contaminated`. Hence these two variable names should be avoided in the data frame, otherwise the respective columns will be overwritten.

The generic function `generate(control, ...)` is available to generate data from a distribution model. Currently only the control class `DataControl` is implemented in `simFrame` (see also Figure 3.1). It allows to specify the number of observations to draw from the distribution, a function for generating the data, e.g., `rmvnorm()` from package `mvtnorm` (Genz and Bretz, 2009; Genz et al., 2010) for a multivariate normal distribution, and a list of additional arguments to be passed to this function. For user convenience, the name of the control class may also be supplied as a character string (the default is `DataControl`), in which case the slots of the control object may be supplied as arguments.

3.1.2 Sampling

One of the most important design principles of `simFrame` is that the indices giving each sample are obtained before the actual simulations are performed. This process is in the following referred to as *setting up* the samples and has two advantages. First, the samples can be stored permanently for faster reproduction of the simulation results or for other simulations based on the same population. Second, overall computation time may be reduced dramatically in certain situations. In stratified sampling, for example, the actual stratification needs to be performed only once.

The multiple samples are obtained with the generic function `setup(x, control, ...)` and stored in an object of class `SampleSetup`, which contains all information required for drawing the samples at a later stage. The basic control class `SampleControl` is highly flexible and may be used for stratified sampling as well as sampling of whole groups rather than individuals with a specified sampling method. Furthermore, the number of items to sample and/or probability weights need to be specified. More complex two-stage sampling designs can be applied with the control class `TwoStageControl`. It allows to specify the desired sampling method for each of the two stages. In addition, the PSUs and SSUs need to be supplied, with the default SSUs being the observations themselves. It is also possible to use stratification in the first sampling stage. See also Figure 3.1 for all slots of the classes involved in the sampling procedure. An important feature is that the each sample is set up using the function `try()` to avoid losing all samples if an error occurs. Furthermore, the control class for `setup()` may be defined by supplying its name as a character string, allowing the slots to be specified as arguments.

Numerous approaches for unequal probability sampling without replacement from finite populations have been proposed in the literature to date (see, e.g. Tillé, 2006), and many of them are implemented in the R package `sampling` (Tillé and Matei, 2009). Wrappers for the corresponding functions to be used with `simFrame` can easily be defined (see Listing 3.3 in Section 3.2.2). Unfortunately, the implementation of the functions in `sampling` is not performant. While the package is excellent for teaching purposes, it cannot be used for the close to reality simulations within AMELI. As a remedy, fast C++ implementations of Midzuno sampling (MIDZUNO, 1952), Tillé sampling (TILLÉ, 1996; DEVILLE and TILLÉ, 1998) and Brewer sampling (BREWER, 1975) are available in package `simFrame`.

Previously set up samples can then be drawn from the population with the generic function `draw(x, setup, ...)`. Note that it adds the column `.weight`, which contains the sample weights, to the resulting data set.

3.1.3 Contamination

The generic function `contaminate(object, control, ...)` is implemented to contaminate certain proportions of the data. Two general control classes are available in `simFrame`: `DCARContControl` and `DARContControl`. DCAR and DAR stand for *distributed completely at random* and *distributed at random*, respectively. This terminology was used earlier in the AMELI project before it was further refined to the terminology presented in Section 2.4.1 (CCAR, CAR and NCAR).

For both control classes, the contamination levels and the variables to be contaminated need to be supplied. Whole groups rather than individuals may be contaminated, and OAR situations instead of OCAR situations may be generated by specifying an auxiliary variable containing probability weights for selecting the items to be contaminated. The full hierarchy of control classes for contamination is outlined in Figure 3.1.

With `DCARContControl`, the contamination does not depend on the original values and is defined in the same way as for model-based data generation - by supplying a function to generate the data and, if necessary, a list of additional arguments to be passed to this function (see Section 3.1.1). The number of items to be generated will thereby be passed as the first argument. With `DARContControl`, on the other hand, the contamination depends on the original values. A function is required, too, but in this case the original values are passed as the first argument. As before, a list of additional arguments may be specified.

It should be noted that the two control classes `DCARContControl` and `DARContControl` are designed to be as general as possible. For the simulations within the AMELI project, specialized contamination models designed specifically for EU-SILC are used. These project-specific extensions of `simFrame` are presented in Section 3.3.

In order to be able to identify the contaminated observations, a logical variable called `.contaminated` is added to the data set. Furthermore, the control class for `contaminate()` may be specified as a character string, with `DCARContControl` being the default. This allows the slots to be supplied as arguments.

3.1.4 Insertion of missing values

Adding missing values follows a similar implementation as contaminating the data. With the control class `NAControl` for the generic function `setNA(x, control, ...)`, missing value rates may be selected individually for the variables in which missing values should be inserted. The same missing value rates are used for all specified variables if they are specified as a vector. However, if the missing value rates are given by a matrix, each column is used for the respective variable. Whole groups may be set as missing rather than individual values, and an auxiliary variable of probability weights may be specified to account for MAR or MNAR situations instead of MCAR situations (see Section 2.4.2). Note that it is possible to specify whether missing values should only be inserted into non-contaminated observations, or if they should also be inserted into contaminated observations. The latter is the more realistic scenario. An overview of the control class is given in Figure 3.1.

In addition, the generic function `setNA()` allows the control class to be specified as character string, which allows the slots to be supplied as arguments.

3.1.5 Running simulations

The function `runSimulation()` combines all elements of the package into one convenient interface for running simulation studies. For design-based simulation, a data frame and previously set up samples (or a control class for setting up samples) may be supplied. For model-based simulation, a control class for data generation and the number of replications may be passed.

How the simulations are performed is determined by the control class `SimControl` (see also Figure 3.1). Control classes for contamination and the insertion of missing values may be supplied. In design-based simulations, contamination and non-response may currently only be added to the samples rather than the population, to give maximum control over the amount of outliers or missing values. It is important to note that the methods themselves should never use this information, this only affects computing evaluation criteria. Furthermore, the simulations may be split into different domains, in which case it is possible to specify whether small area estimation will be used. The most important thing is, however, to specify a `function(x, ...)` for the simulation runs, with the following requirements:

- The function must return a numeric vector, or a list with components `values` (a numeric vector with the main simulation results) and `add` (additional results of any class).
- Its argument `x` should expect a data frame.
- The original data (i.e., the data before contamination and the insertion of missing values) can be passed via an argument called `orig`.
- In the case of splitting the simulations into different domains, the indices of units belonging to the current domain can be passed to the function via an argument called `domain`.

If the simulations are split into different domains, the behavior is as follows. For small area estimation, the whole data set (sample) is always passed to the supplied function, and contamination and missing values are added to the whole data (sample) as well. It is therefore required to use the `domain` argument to extract the current domain in the simulations. On the other hand, if small area estimation is not used, the data is split and simulations are performed on every domain separately, including contamination and the insertion of missing values.

The simulation results are stored in an object of class `SimResults` (see Figure 3.1). Similar to setting up multiple samples, the function for the simulation runs is evaluated using `try()`. Hence, not all results are lost if computations fail in any of the simulation runs.

Note that the slots of `SimControl` may also be specified as arguments for user convenience. A detailed example for design-based simulation is presented in Section 3.4, additional examples for model-based simulation and parallel computing can be found in ALFONS et al. (2010).

3.1.6 Visualization

Various plots for simulation results are implemented using package `lattice` (SARKAR, 2008, 2010). If the simulation study was divided into several domains, the results for each domain are displayed in a separate panel. The following plots are implemented:

`simBwplot`: A boxplot of the simulation results is produced.

`simDensityplot`: A kernel density plot of the simulation results is produced.

`simXyplot`: The average results are plotted against the contamination levels or missing value rates.

Reference lines for the true values can be added in all the plots mentioned above. A convenient feature is that the `plot()` method for the simulation results selects a suitable graphical representation automatically, depending on their structure. Examples are shown in Section 3.4 and ALFONS et al. (2010).

3.1.7 Parallel computing

Being an *embarrassingly parallel* process, statistical simulation may benefit highly from parallel computing, which is implemented in `simFrame` using package `snow` (ROSSINI et al., 2007; TIERNEY et al., 2008, 2009a). The functions `clusterSetup()` and `clusterRunSimulation()` are available for setting up multiple samples and running simulations on a cluster. It is important to note that all necessary packages and objects (e.g., functions, data sets or control objects) need to be made available on every worker process.

Reproducibility and the use of random number generators are slightly more complicated issues for parallel computing than in the sequential case. Simply using the standard random number generator on every worker process is not recommended. If the same seed is used on every worker, identical sequences of random numbers are generated, which in turn produces the same results on each worker. Using different seeds is not a proper solution to this problem, because there might still be some overlap in the sequences of random numbers. The remedy is to use random number streams, which can be initialized with the function `clusterSetupRNG()` from package `snow`. Two packages for creating random number streams are supported: `rlecuyer` (L'Ecuyer et al., 2002; Sevcikova and Rossini, 2009) and `rsprng` (Mascagni and Srinivasan, 2000; Li, 2010).

For more detailed information on parallel computing with `simFrame`, including an example, consult Alfons et al. (2010).

3.2 Extending the framework

The object-oriented implementation of `simFrame` provides clear interfaces for extensions by developers, which are described in the following.

3.2.1 Model-based data

Since the control class `DataControl` is very general, it is often sufficient to supply a user-implemented function for the desired data generation model. The first argument should thereby be the number of observations to be generated (see Listing 3.1).

In order to extend the framework by more specialized data generation models or for more convenient access to frequently used distribution models, a control class extending `VirtualDataControl` and the corresponding method for the generic function `generate()` may be defined (see Listing 3.2).

```
myDataGeneration <- function(size, ...) {
    # computations
}
```

Listing 3.1: Code skeleton for a data generation method.

```
setClass("MyDataControl",
    # class definition
    contains = "VirtualDataControl")

setMethod("generate",
    signature(control = "MyDataControl"),
    function(control) {
        # method definition
    })
```

Listing 3.2: Code skeleton for extending model-based data generation.

3.2.2 Sampling

The control classes `SampleControl` and `TwoStageControl` are highly flexible and allow for the most commonly used sampling designs in practice (see also Section 3.1.2). Thus implementing the desired sampling method for the simplest case of drawing items from a finite (sub-)population often suffices to extend the framework. Details on some restrictions for such a function can be found in Alfons et al. (2010) or on the R help file of the generic function `setup()` (which can be viewed by calling `?setup` or `help(setup)` on the R console). Listing 3.3 shows an example for how to define a simple wrapper function for Poisson sampling based on the implementation in package `sampling` (Tillé and Matei, 2009).

For more complex sampling procedures such as multistage sampling with more than two stages, the framework can be extended by implementing a control class extending `VirtualSampleControl` and the corresponding `setup()` method. The slot `k` of `VirtualSampleControl` is inherited (see Figure 3.1) and should determine the number of samples. In addition, the resulting object must be of class `SampleSetup`. In order to use parallel computing, a method for `clusterSetup()` needs to be defined. A code skeleton for such an extension is given in Listing 3.4.

```
myPoisson <- function(prob) {
    require(sampling)
    which(as.logical(UPpoisson(prob)))
}
```

Listing 3.3: User-defined function for Poisson sampling.

```
setClass("MySampleControl",
    # definition of additional properties
    contains = "VirtualSampleControl")

setMethod("setup",
    signature(x = "data.frame", control = "MySampleControl"),
    function(x, control) {
        # method definition
    })

setMethod("clusterSetup",
    signature(x = "data.frame", control = "MySampleControl"),
    function(cl, x, control) {
        # method definition
    })
```

Listing 3.4: Code skeleton for user-defined setup of multiple samples.

3.2.3 Contamination

Additional contamination models can be added to the framework by defining a control class inheriting from `VirtualContControl` and the corresponding method for the generic function `contaminate()` (see Listing 3.5). Note that the control class inherits the slots `target` and `epsilon` for selecting the target variable(s) and contamination level(s), respectively (see Figure 3.1). Furthermore, a logical variable `.contaminated` indicating the contaminated observations should be added to the resulting data set before it is returned, e.g., to be able to use that information for the evaluation of outlier detection methods.

3.2.4 Insertion of missing values

Last but not least, adding user-defined missing value models is done by implementing a control class extending the virtual class `VirtualNAControl` and the corresponding method for the generic function `setNA()` (see Listing 3.6). The slots `target` and `NArate` for selecting the target variable(s) and missing value rate(s), respectively, are inherited (see Figure 3.1).

```
setClass("MyContControl",
    # definition of additional properties
    contains = "VirtualContControl")

setMethod("contaminate",
    signature(x = "data.frame", control = "MyContControl"),
    function(x, control, i) {
        # method definition
    })
```

Listing 3.5: Code skeleton for user-defined contamination models.

```
setClass("MyNAControl",
    # definition of additional properties
    contains = "VirtualNAControl")

setMethod("setNA",
    signature(x = "data.frame", control = "MyNAControl"),
    function(x, control, i) {
        # method definition
    })
```

Listing 3.6: Code skeleton for user-defined missing value models.

3.3 Extensions for the AMELI project

For the AMELI project, specialized contamination models were designed specifically for EU-SILC data. A detailed description of these contamination models is given in ALFONS et al. (2011a). In the implementation in `simFrame`, the corresponding control classes extend the basic virtual class `VirtualContControl`. This virtual class consists of the following slots (see also Figure 3.1):

`target`: A character vector defining the variables to be contaminated, or `NULL` to contaminate all variables (except the additional ones generated internally).

`epsilon`: A numeric vector giving the contamination levels to be used in the simulation.

For code sharing purposes, the virtual class `UniContControl` is implemented for univariate contamination in EU-SILC data. It extends the basic virtual class `VirtualContControl` by the following additional slots, some of which were named in consistency with the built-in control classes of `simFrame`:

`grouping`: A character string specifying the name of the variable containing the household IDs in order to contaminate whole households rather than individuals.

`main`: A character string specifying the name of the variable containing the main income holder of each household.

`hsize`: A character string specifying the name of the variable giving the number of persons in each household.

`eqsize`: A character string specifying the name of the variable giving the equivalized household size according to the modified OECD scale (see EUROSTAT, 2004).

`design`: A character string specifying the name of a variable defining an OAR situation. If supplied, the outliers are drawn using a stratified sampling design. Consequently, the specified variable will be interpreted as a factor.

`prob`: A numeric vector giving the outlyingness probabilities for the different groups in OAR situations. The values need to be in the order of the corresponding factor levels and should sum up to 1.

In the simulations within the AMELI project, only CCAR and NCAR situations will be considered once the outliers are selected. The control classes `CCARContControl` and `NCARContControl` are implemented for CCAR and NCAR situations, respectively. Both classes extend the virtual class `UniContControl`. The class `CCARContControl` consists of the following slots in addition to the inherited ones:

`distribution`: A function for generating the data for the contamination, e.g., `rnorm()` for a normal distribution (the default).

`dots`: Additional arguments to be passed to the function in slot `distribution`.

On the other hand, the control class `NCARContControl` contains the following additional slots:

`fun`: A function generating the values of the contaminated data based on the original values.

`dots`: Additional arguments to be passed to the function in slot `fun`.

3.4 Example: Quintile share ratio

This section contains an example demonstrating the usage of `simFrame` for design-based simulation. The package contains synthetic EU-SILC data based on the Austrian sample from the year 2006, which is used as population data. In the data set, information on the personal level is available for 25 000 households. Note that this is an illustrative example only. Neither does the data represent the true population sizes of Austria and its regions, nor will a thorough analysis of presented methodology be provided.

Three methods for estimating the *quintile share ratio* (QSR) are compared in the following. The first is the standard estimation as described in Eurostat (2004). The other two are semiparametric approaches that fit a *Pareto* distribution (e.g. Kleiber and Kotz, 2003) to the upper tail of the data. While the *Hill* estimator uses the maximum likelihood principle to fit the shape of the Pareto distribution (Hill, 1975), the *partial density component* estimator (PDC; Vandewalle et al., 2007) follows a robust approach. Note that the semiparametric approaches flag values larger than the theoretical 99% quantile of the fitted distribution as nonrepresentative outliers. Since these are considered to be unique to the population, the sample weights of the flagged observations are then set to 1, and the weights of the remaining variables are calibrated accordingly. Details on the semiparametric methods can be found in Alfons et al. (2011b). All the aforementioned methods are implemented in the R package `laeken` (Alfons et al., 2011c).

The first step is of course to load the required functions and the data set. To ensure reproducibility of the results, the seed of the random number generator is set as well.

```
R> library(" simFrame")
R> source(" simFrameAMELI.R")
R> library(" laeken")
R> data(" eusilcP")
R> set.seed(1234)
```

Using stratified sampling by regions, combined with sampling of whole households rather than individuals, 100 samples from the population are set up with just one command. The sample sizes are thereby selected approximately proportional to the stratum sizes in the population.

```
R> set <- setup(eusilcP, design = " region", grouping = " hid",
+      size = c(192, 1108, 1406, 414, 813, 977, 401, 453, 236),
+      k = 100)
```

Contamination in this example is introduced by a OCAR-CCAR model (see Section 2.4.1). The contamination level is set to 0.1%, and the outliers are generated by a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $\mu = 500\,000$ and standard deviation $\sigma = 20\,000$.

```
R> cc <- CCARContControl(target ="eqIncome", epsilon = 0.001,
+     grouping = "hid", main = "main", hsize = "hsize", eqsize = "eqsize",
+     dots = list(mean = 5e+05, sd = 20000))
```

In the function for the simulation runs, the QSR is computed with the three methods. The Pareto distributions are thereby fit to the largest `k` values.

```
R> sim <- function(x, k) {
+     q <- qsr(x$eqIncome, x$.weight)$value
+     fitHill <- paretoTail(x$eqIncome, k = k, method = " thetaHill" ,
+         w = x$.weight, groups = x$hid)
+     wHill <- reweightOut(fitHill, calibVars(x$region))
+     qHill <- qsr(x$eqIncome, wHill)$value
+     fitPDC <- paretoTail(x$eqIncome, k = k, method = " thetaPDC\grqq ,
+         w = x$.weight, groups = x$hid)
+     wPDC <- reweightOut(fitPDC, calibVars(x$region))
+     qPDC <- qsr(x$eqIncome, wPDC)$value
+     c(standard = q, Hill = qHill, PDC = qPDC)
+ }
```

Running the simulation experiment is only one more command. In this example, the largest 300 observations are used for Pareto tail modeling.

```
R> results <- runSimulation(eusilcP, set, contControl = cc,
+     fun = sim, k = 300)
```

Finally, the true values of the QSR are computed in order to compare them to the simulation results. In the plot of the simulation results, the true values can be added as reference lines (see Figure 3.2).

```
R> tv <- qsr(eusilcP$eqIncome)
```

Figure 3.2 shows that the standard estimation of the QSR leads to corrupted results even for a small proportion of outliers. The classical Hill estimator for fitting the Pareto distribution is somewhat less influenced, but the robust PDC estimator still performs better with respect to bias. In any case, this example shows that `simFrame` allows to use complex simulation designs with only a few lines of code. Other examples for the usage of the package – including model-based simulation, parallel computing and different plots of the simulation results – can be found in ALFONS et al. (2010).

```
R> plot(results, true = tv, xlab = " Quintile share ratio" )
R> simDensityplot(results, true = tv, xlab = " Quintile share ratio" )
```

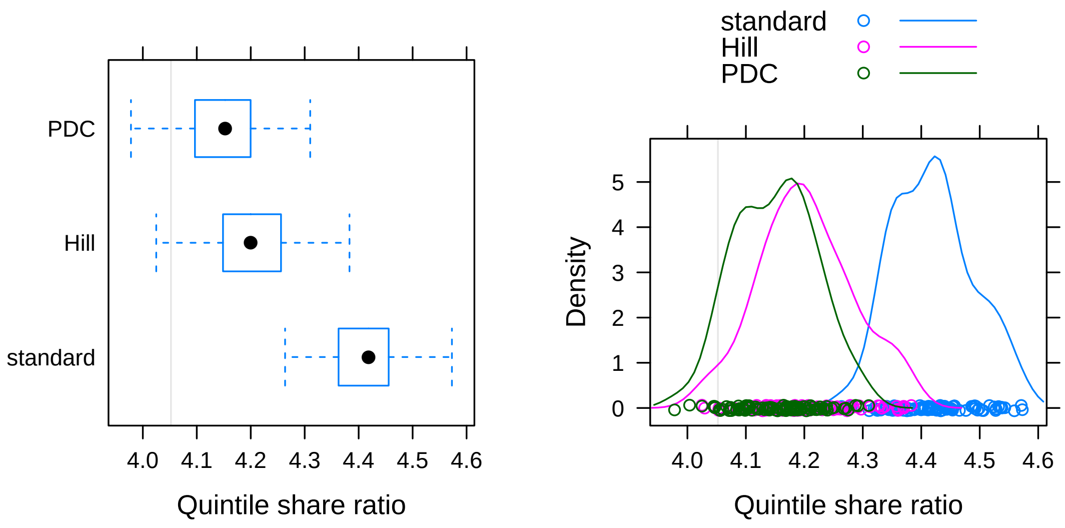

Figure 3.2: Simulation results from the illustrative example. *Left*: Default plot of results from a simulation study with one contamination level. *Right*: Kernel density plots of the simulation results.

Chapter 4

Other simulation environments

A simulation framework was installed in Trier to handle computer intensive calculations with big data amounts. In this Condor environment it is difficult to apply the simFrame Package. To use the Condor simulation framework it is necessary to store the variables in groups with specific names. The simFrame package is good for teaching purposes but cannot be applied for all of the close to reality simulations within AMELI.

4.1 Condor based simulations

Large scaled Monte Carlo studies need a huge amount of calculation time. In order to obtain results in more or less responsive time (2-3 days) a parallelization of the task is needed. In survey statistics the computations may be split e.g. by samples such that blocks of samples are parallel. In Order to do such parallel simulations, multiple CPU are needed. In Trier there are two such computing clusters, an internal high security cluster and an external cluster with access capabilities over Internet.

The internal cluster is used mainly for computation on highly confidential data like Census data or EU-SILC data sets. It has no connection to the Internet whatsoever and only a very limited amount of persons have access to it. It consist of 14 computing servers, a managing server, and a file server. The computing server have overall 304 cores and 1280 GB RAM. The file server is mirrored almost instantly and has 16 TB storage space.

The external cluster is used only with public data sets. Access is restricted to authorized persons only. It can be accessed from all over the world via Internet thorough an encrypted connection. It consists of 5 computing servers with overall 56 cores and 192 GB RAM, a managing server and a file server with 16 TB storage space.

In order to be capable to use all this cores full time simultaneously, a job queuing system called Condor (Thain et al., 2005) is installed on the managing and computing servers. The Condor system on the managing server queries empty slots from the computing servers. If a computing server has an empty slot (e.g., another job just finished) the managing server submits the next job in the queue to fill this slot. Thus, the user only has to submit the jobs once into the queue and the Condor system cares about the optimal distribution to the computing servers. This saves a lot of time and speeds up computations. Often the data sets needed for computations are rather large data set and the simulation results are also quite large. In order to save storage space, the data sets and the results are stored centrally on the file server.

4.2 DBsim: Simulation from Databases

With `DBsim` we present a simple yet powerful simulation framework for the R statistical computing language (R Development Core Team, 2011b; Chambers, 2008b), which is similar to `simFrame` (see above) and is explicitly designed for parallel computation on several nodes while storing the data on a database (Schoch, 2010). Further, `DBsim` is designed to work explicitly with survey (see Lumley, 2004, 2010), outliers and missing values of all kinds. Obviously, this is not the only parallel computation framework for R (see Schmidberger et al. (2009) for a recent comprehensive overview). `DBsim` is the ideal candidate for the tuple `[simulation jobs; computing clusters]` that features the folowing characteristics

- multiple, homogeneous CPUs (with shared memory),
- homogeneous simulation tasks (typical for design-based simulations),
- huge datasets or large, but different datasets for each single simulation run.

For applications that meet only the first two characteristics, the ideal candidate is the R-package `multicore` (see Urbanek, 2009). This implementation draws heavily on the unix-type system function `fork` (and similar wrappers around unix processes) and is thus (at the moment) only available for POSIX-compliant OS (see also `fork`; Warnes (2007). Essentially, computing frameworks that are based on the `fork` mechanism are in general very efficient in situations where all computation tasks act on the same data (or different but smaller datasets without danger that R runs out of memory) because `fork` invokes child processes with the same data and connections (i.e., stdin, stdout, and stderr) using the copy-on-work principle.

Users with a heterogeneous computing environment (e.g., different servers with different OS) may choose the R-package `snow` (see Tierney et al., 2009b) because it provides a setting for distributed memory parallel systems (see also Condor, above).

4.2.1 Conceptual Idea

Instead of letting `R` manage all the storage tasks (viz. `simFrame`), `DBsim` lets `R` and `MySQL` (or any similar relational database system) do what they can do best: all computations are done in `R` whereas `MySQL` is concerned with efficiently organizing and delivering data of all kind. It's as simple as that.

Moreover, `DBsim` is based on the following key assumptions/slogans:

- The purpose of the simulation is to analyze the performance of statistical methods (in terms of several criteria) in the context of different outlyingness and missingness patterns and other measurements problems, given the sampling design and the sample size.
- The important thing to note is that both the population and the sample design are predetermined and serve as fixed parameters in each simulation. This setting accounts for the practical needs in official statistics to find the best tuned methods among all methods given the implemented sampling design. And not vice versa. (This does not mean that only a single sampling design needs to be considered. It only means that comparison of different methods needs to be conditional on the sampling design)
- As a result of the preceding assumptions, the population and sampling design are hold fixed and cannot be modified during a `DBsim` simulation. Consequently, and because all simulation runs are homogeneous and independent, `DBsim` cuts the overall simulation into pieces and distributes them to the nodes. Each computing node has its own DB-connection and requests its data for a particular simulation run. Having finished the computation task, each node writes the results independently to the DB. Since no master coordinates the simulation, `DBsim` has minimal computing power loss due to communication.

4.2.2 Requirements

The DBsim framework requires the following software requirements

- MySQL database sever/client (version 5.0; MySQL (2010)),
- R (R Foundation for Statistical Computing, see R Development Core Team (2011b)),
- R-Package DBI (James, 2009),
- R-Package RMySQL (James and DebRoy, 2010)
- R-Package sets (Meyer and Hornik, 2010),
- R-Package survey (Lumley, 2010).

For more details, see Schoch (2010).

Chapter 5

Summary

The objective of Workpackage 6 is to investigate the methodology developed in Workpackages 2, 3 and 4 in a close to reality environment based on realistic data sets from selected European countries. With these results, Workpackage 7 will provide a thorough analysis of the impact of different situations on the indicators on social exclusion and poverty. Workpackage 9 will then use this output to support policy making with best practice recommendations.

However, the lack of real population data and the complexity of the real processes involved in contamination and non-response make completely realistic scenarios virtually impossible, but the key word is *best* practice recommendations. The work load of the Workpackage can be divided into two major tasks:

- to develop a suitable mechanism for generating population data as basis for the simulations, and
- to provide an outline of the simulation process and simplify the application of common simulation guidelines.

One problem with generating population data from real life samples is that the structure of the original data needs to be represented in the synthetic population, but statistical nondisclosure must under no circumstances be compromised. The first step of data generation is thus to set up the household structure using only regional information, the number of persons in the households as well as their gender and age class. This avoids disclosure problems since no other auxiliary information of the individuals in the real life sample is used and all categories after cross tabulation of these variables are far away to be unique in the sample. Further variables of the synthetic population can then be generated with statistical models. It has been demonstrated that this approach succeeds in reflecting the dependencies among variables and the heterogeneities between subgroups. Therefore, the populations created in this manner are perfectly suitable for close to reality simulation.

Concerning the simulation design, special emphasis is placed on typical data problems such as outliers and non-response. A general question is whether these should be included on the population level or in the samples. While the first approach may be more realistic, it results in an unpredictable number of outliers or missing values in the samples and may thus be suitable if these issues are not the main interest of the simulations. Nevertheless, to evaluate procedures such as outlier identification methods or imputation methods, maximum control over the amount of outliers or missing values is required. In these situations, a more practical approach might be to include them in the samples. Different contamination and non-response models are thereby discussed. Furthermore, various evaluation criteria for different types of simulations are proposed.

In order to simplify using common guidelines in simulation experiments, a software framework has been developed in the R package `simFrame`. It allows the use a wide range of simulation designs with a minimal effort of programming. In addition, the object-oriented implementation provides clear interfaces for further extensions.

Bibliography

Alfons, A. (**2011**): `simFrame`: Simulation framework. R package version 0.4.1.
URL http://CRAN.R-project.org/package=simFrame

Alfons, A., **Bruch, C.**, **Filzmoser, P.**, **Graf, M.**, **Graf, E.**, **Kolb, J.-P.**, **Lehtonen, R.**, **Hulliger, B.**, **Lussmann, D.**, **Meraner, A.**, **Münnich, R.**, **Nedyalkova, D.**, **Schoch, T.**, **Templ, M.**, **Valaste, M.**, **Veijanen, A.** and **Zins, S.** (**2011**a): *Report on the Analysis of the Simulation Results*. Deliverable 7.1, AMELI project.
URL http://ameli.surveystatistics.net

Alfons, A., **Filzmoser, P.**, **Hulliger, B.**, **Meraner, A.**, **Schoch, T.** and **Templ, M.** (**2011**b): *Robust methods for Laeken indicators*. Deliverable 4.2, AMELI project.
URL http://ameli.surveystatistics.net

Alfons, A., **Holzer, J.** and **Templ, M.** (**2011**c): `laeken`: Laeken indicators for measuring social cohesion. R package version 0.2.1.
URL http://CRAN.R-project.org/package=laeken

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2010**): *An object-oriented framework for statistical simulation: The R package simFrame*. Journal of Statistical Software, 37 (3), pp. 1–36.
URL http://www.jstatsoft.org/v37/i03/

Alfons, A., **Templ, M.**, **Filzmoser, P.**, **Kraft, S.**, **Hulliger, B.**, **Kolb, J.-P.** and **Münnich, R.** (**2011**d): *Synthetic data generation of SILC data*. Deliverable 6.2, AMELI project.
URL http://ameli.surveystatistics.net

Béguin, C. and **Hulliger, B.** (**2008**): *The BACON-EEM algorithm for multivariate outlier detection in incomplete survey data*. Survey Methodology, 34 (1), pp. 91–103.

Brewer, K. (**1975**): *A simple procedure for sampling πpswor*. Australian Journal of Statistics, 17 (3), pp. 166–172.

Chambers, J. (**1998**): Programming with Data. New York: Springer, ISBN 0-387-98503-4.

Chambers, J. (**2008**a): Software for Data Analysis: Programming with R. New York: Springer, ISBN 978-0-387-75935-7.

Chambers, J. M. (**2008**b): Software for Data Analysis: Programming with R. Statistics and Computing. New York: Springer-Verlag.

Chambers, R. (**1986**): *Outlier robust finite population estimation.* Journal of the American Statistical Association, 81 (396), pp. 1063–1069.

Deville, J.-C. and **Tillé, Y.** (**1998**): *Unequal probability sampling without replacement through a splitting method.* Biometrika, 85 (1), pp. 89–101.

Eurostat (**2004**): *Common cross-sectional EU indicators based on EU-SILC; the gender pay gap.* EU-SILC 131-rev/04, Unit D-2: Living conditions and social protection, Directorate D: Single Market, Employment and Social statistics, Eurostat, Luxembourg.

Fowler, M. (**2003**): UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 3rd ed., ISBN 978-0-321-19368-1.

Genz, A. and **Bretz, F.** (**2009**): Computation of Multivariate Normal and t Probabilities, *Lecture Notes in Statistics*, vol. 195. New York: Springer, ISBN 978-3-642-01688-2.

Genz, A., **Bretz, F.**, **Miwa, T.**, **Mi, X.**, **Leisch, F.**, **Scheipl, F.** and **Hothorn, T.** (**2010**): `mvtnorm`: Multivariate normal and t distributions. R package version 0.9-92. URL http://CRAN.R-project.org/package=mvtnorm

Hill, B. (**1975**): *A simple general approach to inference about the tail of a distribution.* The Annals of Statistics, 3 (5), pp. 1163–1174.

Hulliger, B. (**2011**): *Main Results of the AMELI Simulation Study on Advanced Methods for Laeken Indicators.* Proceedings of the NTTS conference 2011.

Hulliger, B. and **Münnich, R.** (**2006**): *Variance estimation for complex surveys in the presence of outliers.* In Proceedings of the Section on Survey Research Methods, pp. 3153–3161, American Statistical Association.

Hulliger, B. and **Schoch, T.** (**2009**): *Robust multivariate imputation with survey data.* 57 Session of the International Statistical Institute, Durban.

James, D. A. (**2009**): R-Package. R Special Interest Group on Databases. R database interface. [CRAN version 0.2-5].

James, D. A. and **DebRoy, S.** (**2010**): R-Package. RMySQL. R interface to the MySQL database. [CRAN version 0.2-2].

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

L'Ecuyer, P., **Simard, R.**, **Chen, E.** and **Kelton, W.** (**2002**): *An object-oriented random-number package with many long streams and substreams.* Operations Research, 50 (6), pp. 1073–1075.

Li, N. (**2010**): `rsprng`: R interface to `SPRNG` (Scalable Parallel Random Number Generators). R package version 1.0.
URL http://CRAN.R-project.org/package=rsprng

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

Lumley, T. (**2004**): *Analysis of complex survey samples.* Journal of Statistical Software, 9, pp. 1–19.

Lumley, T. (**2010**): R-Package. Survey: Analysis of complex survey samples. [CRAN version 3.22].

Mascagni, M. and **Srinivasan, A.** (**2000**): *Algorithm 806: SPRNG: A scalable library for pseudorandom number generation.* ACM Transactions on Mathematical Software, 26 (3), pp. 436–461.

Meyer, D. and **Hornik, K.** (**2010**): R-Package. Sets, Generalized Sets, Customizable Sets and Intervals. [CRAN version 1.0-6].

Midzuno, H. (**1952**): *On the sampling system with probability proportional to sum of size.* Annals of the Institute of Statistical Mathematics, 3 (2), pp. 99–107.

Münnich, R., **Bihler, W.**, **Bjørnstad, J.**, **Li-Chun, Z.**, **Davidson, A.**, **Sardy, S.**, **Haslinger, A.**, **Knotterus, P.**, **Laaksonen, S.**, **Ohly, D.**, **Schürle, J.**, **Wiegert, R.**, **Oetliker, U.**, **Renfer, J.-P.**, **Quatember, A.**, **Skinner, C.** and **Berger, Y.** (**2003**a): *Data quality in complex surveys.* DACSEIS Deliverable D1.1, University of Tübingen.
URL http://www.dacseis.de

Münnich, R., **Schürle, J.**, **Bihler, W.**, **Boonstra, H.-J.**, **Knotterus, P.**, **Nieuwenbroek, N.**, **Haslinger, A.**, **Laaksonen, S.**, **Eckmair, D.**, **Quatember, A.**, **Wagner, H.**, **Renfer, J.-P.**, **Oetliker, U.** and **Wiegert, R.** (**2003**b): *Monte Carlo simulation study of European surveys.* DACSEIS Deliverables D3.1 and D3.2, University of Tübingen.
URL http://www.dacseis.de

MySQL (**2010**): MySQL 5.0 Reference Manual, Oracle USA, Inc., Redwood City, CA.

R Development Core Team (**2011**a): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

R Development Core Team (**2011**b): R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing. Vienna.

Rossini, A., **Tierney, L.** and **Li, N.** (**2007**): *Simple parallel statistical computing in R.* Journal of Computational and Graphical Statistics, 16 (2), pp. 399–420.

Rubin, D. (**1976**): *Inference and missing data.* Biometrika, 63 (3), pp. 581–592.

Salvati, N., **Chandra, H.**, **Giovanna Ranalli, M.** and **Chambers, R.** (**2010**): *Small area estimation using a nonparametric model-based direct estimator.* Computational Statistics & Data Analysis, 54 (9), pp. 2159–2171.

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

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

Schmidberger, M., **Morgan, M.**, **Hutchinson, F.**, **Eddelbuette, D.**, **Yu, H.**, **Tierney, L.** and **Mansmann, U.** (**2009**): *State of the art in parallel programming in R.* Journal of Statistical Software, 31.

Schoch, T. (**2010**): DBsim: A Superscalar, Database-Backed, Design-Based Framework for Finite Population Simulations in R. A Technical Report. Arbeitsberichte der Hochschule für Wirtschaft FHNW â€“ Nr. 21 (ISSN Nr. 1662-3266), Olten.

Sevcikova, H. and **Rossini, T.** (**2009**): `rlecuyer`: R interface to RNG with multiple streams. R package version 0.3-1.
URL http://CRAN.R-project.org/package=rlecuyer

Templ, M., **Alfons, A.** and **Kowarik, A.** (**2011**): VIM: Visualization and imputation of missing values. R package version 1.4.4.
URL http://CRAN.R-project.org/package=VIM

Templ, M. and **Filzmoser, P.** (**2008**): *Visualization of missing values using the R-package `VIM`.* Research Report CS-2008-1, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2008-1complete.pdf

Thain, D., **Tannenbaum, T.** and **Livny, M.** (**2005**): *Distributed computing in practice: the Condor experience.* Concurrency - Practice and Experience, 17 (2-4), pp. 323–356.

Tierney, L., **Rossini, A.** and **Li, N.** (**2009**a): *`snow`: A parallel computing framework for the R system.* International Journal of Parallel Programming, 37 (1), pp. 78–90.

Tierney, L., **Rossini, A.**, **Li, N.** and **Sevcikova, A.** (**2009**b): R-Package. Snow: Simple Network of Workstations. [CRAN version 0.3.-3].

Tierney, L., **Rossini, A.**, **Li, N.** and **Sevcikova, H.** (**2008**): `snow`: Simple network of workstations. R package version 0.3-3.
URL http://CRAN.R-project.org/package=snow

Tillé, Y. (**1996**): *An elimination procedure of unequal probability sampling without replacement.* Biometrika, 83 (1), pp. 238–241.

Tillé, Y. (**2006**): Sampling Algorithms. New York: Springer, ISBN 0-387-30814-8.

Tillé, Y. and **Matei, A.** (**2009**): `sampling`: Survey sampling. R package version 2.3.
URL http://CRAN.R-project.org/package=sampling

Urbanek, S. (**2009**): R-Package. Multicore. Parallel processing of R code on machines with multiple cores or CPUs. [CRAN version 0.1-3].

Vandewalle, B., **Beirlant, J.**, **Christmann, A.** and **Hubert, M.** (**2007**): *A robust estimator for the tail index of Pareto-type distributions.* Computational Statistics & Data Analysis, 51 (12), pp. 6252–6268.

Warnes, G. (2007): R-Package. fork: R functions for handling multiple processes. [CRAN version 1.2-2].