Doctoral Thesis

A Generalized Approach to Population Synthesis

Author(s):
Müller, Kirill

Publication Date:
2017

Permanent Link:
https://doi.org/10.3929/ethz-b-000171586

Rights / License:
In Copyright - Non-Commercial Use Permitted
A GENERALIZED APPROACH TO POPULATION SYNTHESIS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

KIRILL MÜLLER

Diplom-Informatiker, Technische Universität Karlsruhe

born on 26.02.1979

citizen of

Germany

accepted on the recommendation of

Prof. Dr. Kay W. Axhausen, examiner
Prof. Dr. Philippe Toint, co-examiner
Prof. Dr. Hillel Bar-Gera, co-examiner

2017
Contents

Abstract xi

Zusammenfassung xiii

Acknowledgments xv

1 Introduction 1

2 Literature review 5

2.1 Introduction 5

2.2 Fitting 6

2.2.1 Iterative Proportional Fitting 8

2.2.2 Control dimensions and zoning 14

2.2.3 Sparse list 15

2.2.4 Zero-cell problem 17

2.2.5 Category reduction 17

2.3 Multi-level fitting 19

2.3.1 Iterative Proportional Updating 20

2.3.2 Entropy optimization 23

2.3.3 Relation matrix 24

2.4 Generation 25

2.4.1 Biased household selection 27

2.4.2 Synthesizing household-person relationships 27

2.4.3 Direct generation from household-level weights 28

2.5 Beyond reweighting 28

2.6 Conclusion 29

3 Multi-level fitting algorithms 31

3.1 Introduction 31
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Hierarchical Iterative Proportional Fitting</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Fitting households</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Fitting persons</td>
<td>36</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Fitting households and persons</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>A common framework</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Iterative Proportional Updating</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Entropy Maximization</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>Numerical issues</td>
<td>42</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Stability</td>
<td>42</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Convergence</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusion</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Survey calibration</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Generalized raking</td>
<td>49</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Notation</td>
<td>49</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Problem</td>
<td>50</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Solution</td>
<td>50</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Distance functions</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Equivalence</td>
<td>53</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Data structure and notation</td>
<td>53</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Objective function</td>
<td>53</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Application</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison</td>
<td>55</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>Comparing fitting algorithms</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Software</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Examples</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>Conclusion</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>Large-scale application of fitting algorithms</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Description of data</td>
<td>66</td>
</tr>
<tr>
<td>6.2</td>
<td>Zero-cell problem</td>
<td>71</td>
</tr>
<tr>
<td>6.3</td>
<td>Sampling error</td>
<td>72</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparing methods</td>
<td>74</td>
</tr>
<tr>
<td>6.5</td>
<td>Conclusion</td>
<td>78</td>
</tr>
</tbody>
</table>
## Appendices

| A | Adjusting the ratio of persons per household in Hierarchical IPF | 149 |
| B | The `MultiLevelIPF` package | 153 |
|   | `compute_margins` | 154 |
|   | `fitting_problem` | 155 |
|   | `flatten_ml_fit_problem` | 157 |
|   | `ml_fit` | 159 |
|   | `toy_example` | 162 |
| C | The `grake` package | 163 |
|   | `dss` | 164 |
|   | `gginv` | 167 |
| D | The `wrswoR` package | 169 |
|   | `sample_int_R` | 171 |
| E | The `mangow` package | 175 |
| F | The `RANN.L1` package | 177 |
| Curriculum Vitae | 181 |
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>IPF example: Problem</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>IPF example: Notation</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>IPF example: Solution</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>List-based IPF</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>IPU example</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Relation matrix</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Single-level fitting algorithms</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Multi-level fitting algorithms</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>Two possible reasons for non-convergence of IPF</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Survey calibration methods</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Histograms of category frequencies for control attributes</td>
<td>68</td>
</tr>
<tr>
<td>6.2</td>
<td>Histograms of category frequencies for auxiliary attributes</td>
<td>69</td>
</tr>
<tr>
<td>6.3</td>
<td>Population per zone or municipality classification</td>
<td>70</td>
</tr>
<tr>
<td>6.4</td>
<td>Sampling fraction needed to avoid the zero-cell problem in presence of a spatial component</td>
<td>72</td>
</tr>
<tr>
<td>6.5</td>
<td>Sampling error with different sampling fractions</td>
<td>73</td>
</tr>
<tr>
<td>6.6</td>
<td>SRMSE values after sampling</td>
<td>74</td>
</tr>
<tr>
<td>6.7</td>
<td>SRMSE values after fitting, aggregated over all controls</td>
<td>76</td>
</tr>
<tr>
<td>6.8</td>
<td>Detailed SRMSE values after fitting</td>
<td>77</td>
</tr>
<tr>
<td>7.1</td>
<td>Distribution of weights after survey calibration</td>
<td>88</td>
</tr>
<tr>
<td>7.2</td>
<td>Individual weights after survey calibration</td>
<td>89</td>
</tr>
<tr>
<td>7.3</td>
<td>Relative error of frequency and duration per activity type</td>
<td>94</td>
</tr>
<tr>
<td>7.4</td>
<td>Error of sociodemographic indicators, after matching</td>
<td>96</td>
</tr>
<tr>
<td>7.5</td>
<td>Error of travel indicators, after matching</td>
<td>98</td>
</tr>
<tr>
<td>7.6</td>
<td>Education activities with different age thresholds</td>
<td>101</td>
</tr>
<tr>
<td>7.7</td>
<td>Distorted weights</td>
<td>102</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Median run times</td>
<td>115</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparison of crank and expj run times</td>
<td>116</td>
</tr>
<tr>
<td>8.3</td>
<td>Comparison of R and expj run times for linear ascending weights</td>
<td>117</td>
</tr>
<tr>
<td>8.4</td>
<td>Break-even point of R and expj run times</td>
<td>118</td>
</tr>
<tr>
<td>8.5</td>
<td>Overhead of unsorted data vs. sorting costs</td>
<td>119</td>
</tr>
<tr>
<td>8.6</td>
<td>Schweder plot comparing the crank and R implementations</td>
<td>122</td>
</tr>
<tr>
<td>8.7</td>
<td>Schweder plot comparing the R implementation with a skewed version of itself</td>
<td>123</td>
</tr>
<tr>
<td>8.8</td>
<td>Combining p-values for a comprehensive test</td>
<td>124</td>
</tr>
<tr>
<td>8.9</td>
<td>Comparing each new code to the stock implementation</td>
<td>125</td>
</tr>
<tr>
<td>8.10</td>
<td>Comparing the stock implementation to a skewed version of itself</td>
<td>125</td>
</tr>
</tbody>
</table>
## List of Algorithms

<table>
<thead>
<tr>
<th>Procedure/Subroutine</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Procedure ML-FIT((H, P, C_a, C_h, \ldots, C_{\alpha}, C_{\beta}, \ldots))</td>
<td>...</td>
</tr>
<tr>
<td>3.2 Subroutine H-FIT((H, f_h, C_{a}, C_{b}, \ldots))</td>
<td>...</td>
</tr>
<tr>
<td>3.3 Subroutine H-ADJUST((H, f_h, x, C_x))</td>
<td>...</td>
</tr>
<tr>
<td>3.4 Subroutine P-FIT-HIPF((P, f_h, C_{\alpha}, C_{\beta}, \ldots))</td>
<td>...</td>
</tr>
<tr>
<td>3.5 Subroutine P-FIT((P, f_h, C_{\alpha}, C_{\beta}, \ldots))</td>
<td>...</td>
</tr>
<tr>
<td>3.6 Subroutine P-ADJUST-IPU((P, f_h, \xi, C_{\xi}))</td>
<td>...</td>
</tr>
<tr>
<td>3.7 Subroutine P-ADJUST-ENT((P, f_h, \xi, C_{\xi}))</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Function</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 sample((n, s, replace, p_i))</td>
<td>...</td>
</tr>
<tr>
<td>8.2 sample.rej((n, s, p_i))</td>
<td>...</td>
</tr>
<tr>
<td>8.3 sample.rank((n, s, p_i))</td>
<td>...</td>
</tr>
<tr>
<td>8.4 sample.expj((n, s, p_i))</td>
<td>...</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>List of population synthesizers reviewed</td>
</tr>
<tr>
<td>4.1</td>
<td>Notational equivalents for multi-level fitting and generalized raking</td>
</tr>
<tr>
<td>5.1</td>
<td>Feasible examples</td>
</tr>
<tr>
<td>5.2</td>
<td>Infeasible examples</td>
</tr>
<tr>
<td>6.1</td>
<td>Description of individual-level attributes in the Swiss census data</td>
</tr>
<tr>
<td>6.2</td>
<td>Description of group-level attributes in the Swiss census data</td>
</tr>
<tr>
<td>6.3</td>
<td>Sampling fraction needed to avoid the zero-cell problem</td>
</tr>
<tr>
<td>6.4</td>
<td>Experiment results</td>
</tr>
<tr>
<td>7.1</td>
<td>Frequency and duration by activity type, after calibration</td>
</tr>
<tr>
<td>7.2</td>
<td>Most frequent activity schedule types, after calibration</td>
</tr>
<tr>
<td>7.3</td>
<td>Frequency and duration by activity type, after matching</td>
</tr>
</tbody>
</table>
Abstract

Agent-based microsimulation model systems for transportation planning operate at the level of individual agents, and allow more detailed simulation and analysis than traditional four-step models. At the core of any agent-based model is the synthetic population, a disaggregate representation of the agents in the study area. While population synthesis of human individuals has been explored broadly, the generation of household populations for the simulation of entire households has been studied only recently in the field of transport planning. This thesis focuses on methods for generating synthetic populations of households that satisfy exogenous constraints at both household and person levels.

A novel method, Hierarchical Iterative Proportional Fitting, and two known methods, Iterative Proportional Updating and Entropy Optimization, are presented in an algorithmic framework. The demonstrated equivalence of Entropy Optimization to Generalized Raking, a method from survey statistics, provides access to strong theoretical results and allows application in a broader scope. In particular, for the first time a fast feasibility check for the exogenous constraints is available. A generic cross-platform open-source implementation is presented. Empirical results suggest that Generalized Raking outperforms existing methods: the synthetic population can be devised with less computational effort and better represents the input data.

This method is then applied to generate a synthetic population with activity chains for entire Switzerland for 2030. The activity chains are reweighted to match the assumed frequency of the particular activity types. Generalized Raking allowed obtaining a suitable set of activity chains with only little modeling effort.

Weighted random sampling without replacement is another important method for generating synthetic populations. A fairly recent efficient algorithm is described, and its open-source implementation with an empirical validation is presented.
Zusammenfassung


Gewichtetes Ziehen ohne Zurücklegen ist eine weitere für die Erzeu-
Zusammenfassung

gung synthetischer Populationen wichtige Methode. Ein vergleichsweise junger schneller Algorithmus wird beschrieben, und dessen Open-Source-Implementierung und eine zugehörige empirische Validierung wird vorgestellt.
Acknowledgments

I gratefully acknowledge the financial support of the Swiss National Science Fund (project 138270), of the SustainCity project (SustainCity, 2011), and of the sponsors of the THELMA project (THELMA, 2013). I would like to thank Prof. Kay W. Axhausen for his support, for the freedom he gave me for pursuing twisted paths, and his patience for the time it took me to walk these paths. Thanks to my co-advisors, Prof. Philippe Toint and Prof. Hillel Bar-Gera, for helpful feedback on earlier versions of the manuscript, and to Prof. Kai Nagel and Prof. Philippe Toint for providing the opportunity to work with their groups in Berlin and Namur, respectively.

Thanks to Prof. Andreas Alfons and Dr. Gregory Jefferis for the collaboration on the grake and RANN.L1 packages, and to Pieter Fourie and Prof. Johan Joubert for testing and contributing to early versions of the MultiLevelIPF package. I drew a lot of inspiration from discussions with Prof. Theo Arentze, Prof. Ty Frazier, Prof. Gunnar Flötteröd, and Dr. Ingo Waschkies.

Most computations presented here were made with the amazing R platform for statistical computing (R Core Team, 2017). The dplyr (Wickham and Francois, 2016) and tidyr packages (Wickham, 2017) helped processing the data; the plots were created with ggplot2 (Wickham, 2009) and rendered with tikzDevice (Sharpsteen and Bracken, 2016). Chapters 5, 6 and 8 were created entirely using knitr (Xie, 2016; 2015; 2014) and rmarkdown (Allaire et al., 2017). The BatchJobs and BatchExperiment packages (Bischl et al., 2015) helped parallelizing the experiments in Chapter 8, and the remake package (FitzJohn, 2017) helped organize the workflow in Chapter 6.

I always had a great time at the IVT thanks to my awesome fellow colleagues, I would like to thank especially Alex, Andy, Basil, Boris, Christof, Christoph, Claude, David, Felix, Georgios, Henrik, Ilka, Konrad, Lara, Matthias, Maxim, Michael, Milos, Nadine, Patrick, Pieter, Rashid, Sebastian, Thibaut, and Vero, for their kind help and exciting discussions.
Acknowledgments

Thanks to Patrick, Adrian, Bernhard, and Peter, who took care of the IT infrastructure and implemented numerous feature requests. The operations team around Edith, Elisabeth, Jenny, Ruth, and Valérie, rendered non-existent all administrative problems. I have also received a tremendous amount of invaluable help from the fantastic StackOverflow and R online communities.

Last but not least, I owe my beloved family for their support and their understanding when I was unavailable to them, especially during the more stressful periods during the research and the preparation of this manuscript. Thank you for your love.
Chapter 1

Introduction

Agent-based microsimulation model systems for transportation planning have come into widespread use (Horni et al., 2016; Barthelemy and Toint, 2015; Bradley et al., 2010; Beckx et al., 2009; Roorda et al., 2008; Bhat et al., 2004; Ben-Akiva et al., 2002; de Palma and Marchal, 2002; Bowman and Ben-Akiva, 2001; Mahmassani et al., 1995). These model systems operate at the level of individual agents, i.e., persons in households. The overall performance of the modeled transport system emerges from the interactions of all agents in the system. A detailed activity log is collected for each agent, which allows fine-grained analyses on various spatial, temporal, and attribute aggregations. In contrast to traditional four-step models, the aggregation levels do not need to be fixed beforehand.

Beyond behavioral models and constraints that define the agents’ behavior, agent-based model systems require a description of each agent’s attributes and location. The synthetic population is composed from several datasets related to the study area, because a single dataset with localized individual data for all persons in the study area with all necessary attributes is virtually never available to the modeler. It is not necessary that each agent corresponds to a true person, only that the population of agents is statistically consistent with data that describe the study area.

The generation of synthetic populations of human individuals has been explored broadly. In the seminal paper by Beckman et al. (1996), the Iterative Proportional Fitting (IPF) procedure (Deming and Stephan, 1940) is suggested to reweight a sample of persons to exogenous constraints. However, personal decisions are affected by the individual household situation—the behavior of an independent person is different from that of a parent or of an elderly person living with a partner (Jones et al., 1983). To
simulate these interactions, replicating the proper household structure is a major requirement for the synthetic population.

The literature review in Chapter 2 focuses on procedures that extend the approach by Beckman et al. (1996) to household structures. IPF generates maximum-likelihood weights for a contingency table, but it is not obvious how the MLE property weights carry over to IPF extensions that work on a hierarchical sample, like Iterative Proportional Updating (IPU, Ye et al., 2009), or to heuristics that reconstruct households based on a reweighting computed by IPF. In the published literature, more focus is put on empirical results than on mathematical rigor. In addition, implementations of the published methods are either unavailable, not portable, or restricted to specially formatted input data. This thesis aims at laying a more solid foundation for reweighting-based population synthesis methods, and presents a cross-platform, generic open-source implementation of the methods described here.

Hierarchical Iterative Proportional Fitting (HIPF, Müller and Axhausen, 2011a) is a fitting algorithm based on IPF that operates on a hierarchical sample. Bar-Gera et al. (2009), and later Lee and Fu (2011), propose direct optimization-based estimation of weights with optimal entropy. Chapter 3 establishes a common algorithmic framework for IPU, HIPF, and entropy optimization, in which the differences between the three algorithms are reduced to a minimum.

The original application of IPF was the estimation of weights for a survey that satisfy exogenous marginal totals. Since then, methods supporting continuous variables, and also combinations of continuous and categorical variables and hierarchical samples, have become available, such as generalized raking (Deville and Särndal, 1992). Equivalence of Entropy Maximization to a special case of generalized raking is shown in Chapter 4. This provides access to strong theoretical results, which ultimately allow application in a broader scope and reduction of computational effort. An open-source reimplementation of these algorithms for the popular statistical computing environment R (R Core Team, 2017), and an experimental evaluation based on a toy dataset, are presented in Chapter 5. These algorithms are applied to samples drawn from the full population census of Switzerland in Chapter 6.

An example for a different application of generalized raking is the update of relative frequencies of activities to reflect assumptions on future behavior.
Chapter 7 describes a case study where a synthetic population for entire Switzerland based on census data is augmented with adapted activity chains. Weighted random sampling without replacement is an important computational primitive for generating synthetic populations. However, its R implementation is too slow for large instances. An efficient algorithm (Efraimidis and Spirakis, 2006) and its open-source implementation for R, including an empirical validation of its correctness, are presented in Chapter 8.

Mathematical details and technical documentation are included as an appendix.
Chapter 2

Literature review

This chapter reviews existing population synthesis procedures. The review focuses on reweighting methods that estimate a weight for each observation in a population sample, and use these weights to generate the synthetic population. Many are based on the Iterative Proportional Fitting algorithm, its detailed description here helps analyzing the individual contributions of the reviewed population synthesis procedures. Chapters 3 and 4 describe further generalizations of the IPF algorithm to the case of hierarchical samples, which are then compared in Chapters 5 and 6.

2.1 Introduction

One of the most acknowledged efforts to generate a population of human agents based on statistical microdata is the original work of Beckman et al. (1996). The principal idea of the reweighting method is to reweight a population sample to satisfy additional constraints given by census cross-tabulations (fitting stage), and then derive the actual agent population in proportion to these weights (generation stage). Using a population sample allows the inclusion of more detailed attributes than available from the cross-tabulations alone.

Since then, the method has been refined to address issues like simultaneous control at multiple hierarchy levels, categorization detail, convergence, and memory requirements (Pritchard and Miller, 2012; Auld and Mohammadian, 2010; Ye et al., 2009; Srinivasan et al., 2008; Arentze et al., 2007; Guo and Bhat, 2007; Bar-Gera et al., 2009; Lee and Fu, 2011). This chapter reviews the population synthesizers mentioned above, describes common

---

1 This chapter is based on a peer-reviewed conference paper (Müller and Axhausen, 2011b).
features and analyzes and evaluates their methodological characteristics. The accuracy or goodness-of-fit of the populations synthesized by the various methods are not discussed here, in particular because different measures of goodness-of-fit for different input and output data are applied to each. Also, the review focuses on the methodological improvements, and less on the actual case studies where the methods were applied.

Table 2.1 is an overview that briefly describes the improvement over existing approaches for each synthesizer. This list is not exhaustive: Among others, the TRANSIMS population synthesizer described in (Hobeika, 2005), and a synthesizer for the Swiss population by Frick and Axhausen (2004), also belong to the class of reweighting methods but are not discussed further. Barthelemy and Toint (2012) use a variant of reweighting to generate a pool of individuals which are later grouped to households using combinatorial optimization. Beyond agent-based modeling, synthetic populations are useful for computing small-area statistics in the domain of spatial microsimulation, see (Hermes and Poulsen, 2012) for a review.

The next two sections describe single-level fitting and multi-level fitting, respectively. The subsequent section describes the generation stage. All three sections highlight similarities and differences between the synthesis procedures. Section 2.5 briefly discusses alternative methods such as combinatorial optimization and synthetic reconstruction. This chapter concludes with a summary and a brief overview of the following chapters.

2.2 Fitting

The purpose of the fitting stage is to fit a disaggregate sample of agents (called reference sample) to aggregate constraints (referred to as control totals or controls). This section presents algorithms that fit only one level of aggregation, i.e., only persons or only households. These algorithms postpone control for the other level of aggregation to the generation stage. The next section is dedicated to algorithms that simultaneously fit for both levels of aggregation.

For household-person synthesis, the reference sample contains demographic data for a representative subsample of the population, usually anonymized, and sometimes taken at a point in time different to the simulation base year. In the vast majority of cases the attributes are categorical.
Table 2.1: List of population synthesis procedures reviewed

<table>
<thead>
<tr>
<th>Name/Model</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ILUTE**  | Fitting: “Sparse list” data structure  
Generation: Connecting persons to households  
(Pritchard and Miller, 2012)  
Toronto, Canada  
pages 15 and 27 |
| **CEMDAP** | Fitting: Focus on zero-cell problem  
Generation: Household selection aware of person-level constraints (Guo and Bhat, 2007)  
Dallas-Fort Worth, Texas, USA  
pages 17 and 27 |
| **PopSynWin** | Fitting: On-the-fly categorization (Auld et al., 2009)  
Generation: Household selection aware of person-level constraints (Auld and Mohammadian, 2010)  
Chicago, Illinois, USA (generic)  
pages 17 and 27 |
| **ENT**    | Fitting: Simultaneous fitting to person and household constraints, relaxed formulation (Bar-Gera et al., 2009)  
Maricopa County, Arizona, USA (generic)  
page 23 |
| **PopGen** | Fitting: Simultaneous fitting to person and household constraints (Ye et al., 2009)  
Maricopa County, Arizona, USA (generic)  
page 20 |
| **ALBATROSS** | Fitting: Converting distribution of individuals to distribution of households (Arentze et al., 2007, 2000)  
Netherlands  
page 24 |
| **FSUMTS** | Generation: Deterministic household selection (Srinivasan et al., 2008)  
Tampa Bay and South-East Florida, Florida, USA  
page 27 |
The control totals are given for a selection of attributes, the control variables, present in the reference sample. For each category, the target value specifies the desired number of persons or households per category. The joint distribution of control variables in the sample is referred to as seed. Typically, the reference sample is obtained from survey data such as a population census or a transportation microcensus, whereas the control totals are obtained from readily available aggregate data.

### 2.2.1 Iterative Proportional Fitting

The Iterative Proportional Fitting procedure (IPF) estimates a multi-dimensional distribution of control variables with the following two properties: (a) the sum over any category equals the corresponding target value, and (b) the correlation structure of the seed is retained. IPF has been first described by Deming and Stephan (1940) and is also known as matrix raking, RAS method, or matrix scaling. Among all distributions that satisfy the control totals, the resulting table is the most similar one to the initial distribution with respect to information gain (also known as relative entropy or Kullback-Leibler divergence) (Ireland and Kullback, 1968). This conforms to the principle of Minimum Discrimination Information (Kullback, 1959): New evidence (the control totals) should add the least possible amount of new information to the hypothesis (the seed).

Since this algorithm is of vital importance for population synthesis using synthetic reconstruction, two-dimensional IPF is described verbally, formally, and with a numerical example. Extensions to the more general multi-dimensional case are sketched. The example used throughout this chapter considers the distribution of work status (non-working, part-time and full-time) and age (four categories) in a small sample population (Fig. 2.1(a)). The control totals exhibit a slight overall population growth and a considerable shift towards full-time occupation (Fig. 2.1(b)).

**Algorithm** For a two-way contingency table, as shown in Fig. 2.2, the value of the interior cell in row \(i\) and column \(j\) is denoted by \(n_{ij}\), and the row and column sums as \(n_{i\bullet}\) and \(n_{\bullet j}\), respectively:

\[
\sum_m n_{im} = n_{i\bullet} \quad \text{for all } i, \quad \text{and} \quad \sum_m n_{mj} = n_{\bullet j} \quad \text{for all } j.
\]
The row and column target values are denoted by $r_i$ and $c_j$, respectively. Fitting to the control totals consists of computing $n_{ij}^*$ subject to

$$n_{i*}^* = r_i \text{ for all } i, \text{ and } n_{*j}^* = c_j \text{ for all } j.$$ 

While there are infinitely many solutions to this system of equations in general, the IPF solution is unique. IPF computes a sequence of tabulations, denoted by $n_{ij}^{(k)}$, starting with the seed at $k := 0$:

$$n_{ij}^{(0)} := n_{ij} \text{ for all } i, j.$$  \hfill (2.1)
Chapter 2. Literature review

Figure 2.2: Structure of a two-way contingency table

The sequence is computed using the following assignment:

\[
\begin{align*}
n_{ij}^{(k+1)} &= n_{ij}^{(k)} \cdot \begin{cases} 
  r_i \div n_{i\cdot}^{(k)} & \text{if } k \text{ even}, \\
  c_j \div n_{\cdot j}^{(k)} & \text{if } k \text{ odd.}
\end{cases} 
\end{align*}
\]

For \( k \) even, the scaling factor amounts to the discrepancy of observed row sums versus row target values. Scaling makes the row totals of the table agree to the row controls; however, in general, column totals fail to match the column controls after this adjustment. In the subsequent iteration, column totals are forced to agree with the column controls, distorting the fit for the rows. The target matrix \( n_{ij}^* \) is the limit of the sequence for \( k \to \infty \), if it exists:

\[
n_{ij}^* := \lim_{k \to \infty} n_{ij}^{(k)}.
\]

In practice, the procedure is aborted as soon as the scaling factors are very close to unity or the target values are reached within a specified tolerance.

The limit \( n_{ij}^* \) is also a stationary point of the sequence given in Eq. (2.2), and therefore satisfies both row and column controls. As the IPF procedure uniformly rescales one column or row in each iteration, no additional information on the interaction between columns and rows is added in the process.
In the general case of more than two dimensions, a multi-dimensional array with one dimension per variable is used. All variables are processed in a round-robin fashion. For each variable, all strata corresponding to each target value are considered. All cells belonging to the same stratum are rescaled uniformly so that their sum equals the target value. This procedure also applies to multi-dimensional control totals, only that the strata become smaller the more dimensions a control total covers, which may trigger the zero-cell problem discussed in Section 2.2.4.

For the numerical example, the contingency table is extended by two rows and columns, containing the target values and the scaling factors (Fig. 2.3(a)). The result of the first two IPF iterations is shown in Figs. 2.3(b) and 2.3(c). Convergence within the given precision is achieved after 60 iterations (Fig. 2.3(d)). As the result suggests, in general, IPF yields fractional weights. Integer counts are obtained in the generation stage, described in Section 2.4.

An example for the more general case is the extension by an additional gender attribute, which is used in control totals of age by gender, and work status by gender. Both the original and the extended example, and most following, will be repeated in Chapter 5, where multi-level fitting algorithms are compared.

**Mathematical properties**

Stephan (1942) effectively proves that the IPF target matrix is of the form

\[ n_{ij}^* = q \cdot a_i \cdot b_j \cdot n_{ij} \] (2.3)

(with \( \prod a_i = \prod b_j = 1 \)), provided that all cells in the initial table are strictly larger than zero. In other words, the value of the fitted cell equals the value of the original cell, scaled by the row and column effects \( a_i \) and \( b_j \) and an overall scaling factor \( q \). The decomposition into row and column effects means that IPF does not introduce any new second- or higher-order interactions not present in the seed. With nonzero factors, logarithmizing the cell values of the IPF target matrix (2.3) yields

\[ \ln n_{ij}^* = \mu + \alpha_i + \beta_j + \ln n_{ij}, \]

with \( \mu = \ln q, \alpha_i = \ln a_i, \) and \( \beta_j = \ln b_j \). This can be interpreted as a result of a log-linear analysis of \( n_{ij}^* \). All second-order interactions are obtained...
Figure 2.3: Solution process for the Iterative Proportional Fitting procedure

(a) Contingency table initialized for IPF

<table>
<thead>
<tr>
<th>0–14</th>
<th>15–34</th>
<th>35–64</th>
<th>65+</th>
<th>$n_{i\bullet}^{(k)}$</th>
<th>$r_i$</th>
<th>$r_i/n_{i\bullet}^{(k)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>23</td>
<td>35</td>
<td>74</td>
<td>205</td>
<td>124</td>
<td>0.60</td>
</tr>
<tr>
<td>0</td>
<td>42</td>
<td>17</td>
<td>15</td>
<td>74</td>
<td>83</td>
<td>1.12</td>
</tr>
<tr>
<td>0</td>
<td>60</td>
<td>65</td>
<td>2</td>
<td>127</td>
<td>227</td>
<td>1.79</td>
</tr>
</tbody>
</table>

$\sum_i n_{i\bullet}^{(k)} = 73 + 125 + 117 + 91 = 406$

$\sum_j c_j = 88 + 132 + 115 + 99 = 434$

$\sum_i r_i = 205 + 74 + 127 = 406$

$c_j/n_{i\bullet}^{(k)} = 1.21, 1.06, 0.98, 1.09$

(b) Contingency table after fitting for the row controls

<table>
<thead>
<tr>
<th>0–14</th>
<th>15–34</th>
<th>35–64</th>
<th>65+</th>
<th>$n_{i\bullet}^{(k)}$</th>
<th>$r_i$</th>
<th>$r_i/n_{i\bullet}^{(k)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>44.16</td>
<td>13.91</td>
<td>21.17</td>
<td>44.76</td>
<td>124</td>
<td>124</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>47.11</td>
<td>19.07</td>
<td>16.82</td>
<td>83</td>
<td>83</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>107.24</td>
<td>116.18</td>
<td>3.57</td>
<td>227</td>
<td>227</td>
<td>1</td>
</tr>
</tbody>
</table>

$\sum_i n_{i\bullet}^{(k)} = 44.16 + 168.26 + 156.42 + 65.16 = 434$

$\sum_j c_j = 88 + 132 + 115 + 99 = 434$

$c_j/n_{i\bullet}^{(k)} = 1.99, 0.78, 0.74, 1.52$
Figure 2.3: Solution process for the Iterative Proportional Fitting procedure (continued)

(c) Contingency table after fitting for the column controls

<table>
<thead>
<tr>
<th>$k = 2$</th>
<th>$j$ (age)</th>
<th>$n^{(k)}_{i*}$</th>
<th>$r_i$</th>
<th>$r_i / n^{(k)}_{i*}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0–14</td>
<td>15–34</td>
<td>35–64</td>
<td>65+</td>
</tr>
<tr>
<td>$i$ (wk. stat.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○</td>
<td>88.00</td>
<td>10.91</td>
<td>15.56</td>
<td>68.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>182.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.68</td>
</tr>
<tr>
<td>●</td>
<td>0.00</td>
<td>36.96</td>
<td>14.02</td>
<td>25.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.08</td>
</tr>
<tr>
<td>●</td>
<td>0.00</td>
<td>84.13</td>
<td>85.42</td>
<td>5.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>174.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.30</td>
</tr>
</tbody>
</table>

(d) Contingency table upon convergence

<table>
<thead>
<tr>
<th>$k \rightarrow \infty$</th>
<th>$j$ (age)</th>
<th>$n^{(k)}_{i*}$</th>
<th>$r_i$</th>
<th>$r_i / n^{(k)}_{i*}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0–14</td>
<td>15–34</td>
<td>35–64</td>
<td>65+</td>
</tr>
<tr>
<td>$i$ (wk. stat.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○</td>
<td>88.00</td>
<td>1.83</td>
<td>2.52</td>
<td>31.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>●</td>
<td>0.00</td>
<td>25.22</td>
<td>9.27</td>
<td>48.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>●</td>
<td>0.00</td>
<td>104.95</td>
<td>103.21</td>
<td>18.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>227</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

| $n^{(k)}_{*j}$ | 88  | 132  | 115  | 99   |
| $c_j$         | 88  | 132  | 115  | 99   |
| $c_j \div n^{(k)}_{*j}$ | 1   | 1    | 1    | 1    |
from the seed, while the control totals influence only the main, row, and column effects. Therefore, population synthesis with IPF is not much different to estimating a loglinear model for the data in the fitting stage, and then simulating this model in the generation stage. A brief outlook on more general modeling approaches is presented in Section 2.5.

Convergence criteria for IPF have been studied in depth. Pukelsheim (2013), adapted from (Pukelsheim and Simeone, 2009), summarizes the past efforts and provide a proof of convergence with necessary and sufficient conditions for the two-dimensional case. If all cells are nonzero, IPF converges if and only if the sum of the row totals equals the sum of the column totals. In presence of zero cells, convergence also depends on their arrangement in the matrix. A column $j$ is called connected to a row $i$ if $n_{ij} > 0$; IPF obtains a solution if and only if the sum of the target values for each subset of rows is less or equal to the sum of the target values for all columns connected to at least one of these rows. In practice, for the application of population synthesis, convergence problems mostly occur if an entire row or column is zero, and the corresponding target value is nonzero. This is an obvious violation of the convergence criterion, and discussed further in Section 2.2.4.

If the sum of the row totals differs from the sum of the column totals, a fit is not possible at all. IPF cannot converge in this case. In the case of multi-dimensional controls, this problem is even more subtle: If a variable is used for more than one control total, collapsing these control totals must result in identical target values for that variable. Contradicting controls often occur when using data from different sources. The data must be prepared in advance so that the grand totals, and also all relevant subtotals, match (Rich and Mulalic, 2012). Barthelemy and Toint (2012) suggest rescaling offending controls to the assumed true grand total.

2.2.2 Control dimensions and zoning

As described before, IPF supports multi-dimensional control totals, and so do most of the reviewed population synthesizers. The exceptions are PopGen and ALBATROSS: Their special data structures, further described in Section 2.3, do not provide straightforward support for multi-dimensional controls.

Most synthesizers work with a spatial hierarchy: A region is subdivided
into several zones, and one or several regions define the study area. Control totals are usually provided at zone level, but the reference sample is given region-wise.

For the ILUTE synthesizer (Salvini and Miller, 2005), two approaches to zone-level synthesis are compared by Pritchard and Miller (2012). In the zone-by-zone approach, IPF is first run at regional level using controls aggregated for the region, and the result becomes the seed for further IPF runs that compute the disaggregate zone-wise population. In contrast, the multizone approach synthesizes all zones simultaneously by adding a zone dimension to each control. The multizone approach slightly improves the fit; however, it also requires substantially more memory. All other synthesizers implement the simpler zone-by-zone approach.

### 2.2.3 Sparse list

Memory requirements for the contingency table were a major problem for many previous synthesizers: The contingency table grows exponentially with the number of attributes, its size equals the product of the category counts of all attributes. A large contingency table is inherently sparse: The number of nonzero values is at most as large as the size of the reference sample. This calls for a more efficient data structure.

Williamson et al. (1998) recommend a list-based representation in the context of population synthesis. As the reference sample is usually given as a list of observations, each specified as a tuple with one element per attribute, it can be used without further treatment. The ILUTE synthesizer implements a variant of IPF that operates directly on that list by attaching a non-negative real-valued expansion factor, denoted by $f$, to each observation. Every operation of classical IPF can be translated into a reweighting of expansion factors, allowing the algorithm to work entirely on the list-based representation.

In the following, the equivalence of list-based and classical IPF is demonstrated for the two-dimensional case, the argument can be easily generalized to more than two dimensions. Figure 2.4 repeats the numerical example shown in Fig. 2.3 (Section 2.2.1) with the list-based approach and the notation used here.
Formally, the disaggregate reference sample is given as a finite sequence of category tuples

\[(X_x)_{x=1}^n = ((i_1, j_1, \ldots), \ldots),\]

with control variables \(i\) and \(j\), and arbitrary payload. For a given combination of categories \((i, j)\), the set \(X_{ij}\) of the reference sample observations that fall into these categories is defined as follows:

\[X_{ij} := \{x : (i_x, j_x, \ldots) = (i, j, \ldots)\}.\]

An expansion factor \(f_x\) is defined for each element of the sequence. In addition, the sum \(F_{ij}\) of all expansion factors for given categories \(i, j\) is defined:

\[F_{ij} := \sum_{x \in X_{ij}} f_x.\]

For the first iteration, at \(k = 0\), the expansion factor is initialized to unity: \(f_x^{(0)} := 1\). The following update assignment for \(f\) is the equivalent to the original IPF update in Eq. (2.2) on page 10:

\[f_x^{(k+1)} := f_x^{(k)} \cdot \begin{cases} r_i \div \sum_m F_{i,m}^{(k)} & \text{if } k \text{ even}, \\ c_j \div \sum_m F_{m,j}^{(k)} & \text{if } k \text{ odd}. \end{cases}\]

It is easy to verify that the identity \(n_{ij}^{(k)} = F_{ij}^{(k)}\) is satisfied for all \(k\), cf. the initialization and update rules for IPF in Eqs. (2.1) and (2.2). Essentially, the above invariant defines a translation between expansion factors in the list-based IPF and cell values in the classical IPF. A detailed description can be found in (Pritchard, 2008, Section 4.2.1).

In Fig. 2.4, seed entries with common work status and age attributes are shown in condensed form: One row in the table corresponds to multiple entries in the list. Figures 2.4(a) and 2.4(b) show the first and second iteration, respectively. In Fig. 2.4(b), the list is sorted by age only to simplify the presentation. The result upon convergence (rightmost column in Fig. 2.4(b)) is the same as with classical IPF.

For large attribute spaces and detailed categorization schemes, memory
consumption is greatly reduced. The memory required by the list-based representation is only proportional to the size of the reference sample and the number of attributes; it does not depend on categorization detail. For the categorization scheme used by the ILUTE model, the sparse list representation cuts down memory requirements to 0.2 % for the zone-by-zone and to 0.07 % for the multizone approach (Pritchard and Miller, 2012).

The following two convenient properties of list-based IPF are noteworthy as well. First, all attributes present in the reference sample are preserved, not only those controlled for. Second, the method natively supports simultaneous fitting against controls with different categorization detail (e.g., if the age attribute is given in a coarse categorization for one control, and in a more fine-grained categorization for another).

### 2.2.4 Zero-cell problem

For detailed control totals with many categories, a category that has no representative in the reference sample may have a non-zero target value. This may also occur when dealing with small geographies (Beckman et al., 1996). In this case, a division by zero occurs during the execution of IPF—the outcome of the algorithm is undefined. This is referred to as the zero-cell problem. The simplest solution is to initialize the false zero cells with an arbitrarily small value. This assures convergence but introduces bias (Guo and Bhat, 2007). For this reason, other solutions are desirable.

PopSynWin reduces the occurrence of zero cells using the category reduction routine described in Section 2.2.5. For the CEMDAP synthesizer (Pinjari et al., 2006), Guo and Bhat (2007) also suggest manual category reduction as a preprocessing step.

PopGen fills false zero cells in the zone-level seed with an estimate computed from the region-level seed (Ye et al., 2009). Similarly, the FSUMTS synthesizer “borrows” from another, arbitrarily selected, area to fill false zero cells (Srinivasan et al., 2008).

### 2.2.5 Category reduction

Among the reviewed population synthesizers, only ILUTE supports arbitrarily detailed categorization of attributes. All other procedures resort to
coarser categorization or reduction of the number of variables to keep the contingency table at reasonable size.

*PopSynWin* implements automatic aggregation of categories for interval-scale attributes. Each category for which the corresponding target value does not exceed a user-specified percentage threshold is merged with a neighboring category. Recategorization is performed separately for each region. Apart from decreasing the number of categories, this procedure also reduces the occurrence of false zero cells (see above).

According to Auld et al. (2008), category reduction potentially worsens
the quality of the synthesized population. Nevertheless, this is a feasible remedy against the zero-cell problem. If memory consumption is the only concern, a sparse list seems to be a better approach.

2.3 Multi-level fitting

In the previous section, various single-level fitting algorithms were considered. The current section is dedicated to algorithms that simultaneously fit for control totals at multiple levels of aggregation—for generating synthetic
human households, the fitting algorithm simultaneously controls at the person and household level.

Three approaches are shown that simultaneously control for both person- and household-level attributes, which is not possible with IPF: an algorithm similar to IPF developed by Ye et al. (2009), entropy optimization (Bar-Gera et al., 2009; Lee and Fu, 2011), and a formulation as an IPF problem on a special structure as shown by Arentze et al. (2007). Chapters 3 and 4 further expand on multi-level fitting techniques.

2.3.1 Iterative Proportional Updating

Unlike most of the synthesis procedures reviewed, PopGen uses a fitting procedure different to IPF. The new approach, named Iterative Proportional Updating (IPU), simultaneously controls for multiple hierarchy levels during the fitting procedure. The proposed algorithm has many parallels to the sparse list variant of IPF. In what follows, this algorithm is described, and a numerical example is provided for the simplest case of one household-level and one person-level control.

Similarly to list-based IPF in Section 2.2.3, the seed (at household level) is given as a pair of finite sequences of category tuples. The sequences \((H_h)_{h=1}^n\) and \((P_p)_{p=1}^m\) describe the households or the persons, respectively:

\[
(H_h)_{h=1}^n = (((a_1, b_1, \ldots), \ldots, (a_n, b_n, \ldots)) \text{ with } H_h \in \alpha \times \beta \times \ldots,

(P_p)_{p=1}^m = ((\alpha_1, \beta_1, \ldots, h_1), \ldots) \text{ with } P_p \in \alpha \times \beta \times \cdots \times \{1, \ldots, n\}.
\]

For each person index \(p\), the special attribute \(h_p\) specifies the household index of that person. The description below is limited to two control variables only, one at the person and one at the household level, but the method allows for more control variables.

In a first step, the person-level sequence is collapsed to the household level. For each category in the person-level controls, the number of persons within this category is counted:

\[
n_{ha} := |\{p \in \{1, \ldots, m\} : \alpha_p = \alpha \land h_p = h\}|.
\]
For consistency, a similar conversion is applied to the household-level attributes as well. The counts can be either zero or one here:

\[
    n_{ha} := \begin{cases} 
    1 : a = a_h, \\ 
    0 : \text{otherwise}. 
    \end{cases}
\]

As with list-based IPF, an expansion factor \( f \) is attached to each household and initialized to unity for the first iteration. The weighted sum \( F_\lambda \) of all expansion factors relevant for a household- or person-level category \( \lambda \in a \sqcup \alpha \) is defined as

\[
    F_\lambda := \sum_h (f_h \cdot n_{ha}).
\]

The target values are denoted by \( C_\alpha \) for the household level and \( C_\alpha \) for the person level. IPU determines expansion factors \( f^* \) so that the control totals for both household and person levels are satisfied:

\[
    F^*_\lambda = C_\lambda \text{ for all } \lambda \in a \sqcup \alpha. \tag{2.4}
\]

The algorithm proceeds in iterations. All categories \( \lambda \), at both person and household levels, are considered in a round-robin fashion. Like with IPF, the ratio between target value and currently estimated persons or households is used as a scaling factor. However, the expansion factors are updated only for those households that contribute to the current category \( \lambda \):

\[
    f_h^{(k+1)} := f_h^{(k)} \cdot \begin{cases} 
    e_m \div F_\lambda^{(k)} & : n_{ha} > 0, \\ 
    1 & : \text{otherwise.}
    \end{cases}
\]

Note that, when only household-level attributes are controlled for, expansion factors are rescaled exactly as in list-based IPF. Thus, IPU can be seen as an extension to IPF.

For the numerical example shown in Fig. 2.5, the same seed at person level is used as for the IPF example in Section 2.2.1. Work status is now distinguished only between “non-workers” (⊙) and “full-time” (●), and a new household-level attribute is added that indicates whether the household has at least one car available (□) or not (□). The same target values for the work status as in the initial example are used; the “car” controls exhibit
Figure 2.5: Example for the Iterative Proportional Updating procedure

<table>
<thead>
<tr>
<th>$y$</th>
<th>$n_{y\alpha}$</th>
<th>$n_{y\alpha}$</th>
<th>$k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..22</td>
<td>2/1</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>1.65</td>
<td>1.64</td>
<td>1.60</td>
</tr>
<tr>
<td>23..43</td>
<td>1/1</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>1.65</td>
<td>1.64</td>
<td>1.60</td>
</tr>
<tr>
<td>44..64</td>
<td>3/0</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.74</td>
<td>0.73</td>
<td>0.33</td>
</tr>
<tr>
<td>65..80</td>
<td>2/0</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.38</td>
<td>0.38</td>
<td>0.19</td>
</tr>
<tr>
<td>81..96</td>
<td>2/1</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>0.86</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>97..108</td>
<td>1/0</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.38</td>
<td>0.38</td>
<td>0.19</td>
</tr>
<tr>
<td>109..119</td>
<td>2/0</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.74</td>
<td>0.73</td>
<td>0.33</td>
</tr>
<tr>
<td>120..128</td>
<td>1/0</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.74</td>
<td>0.73</td>
<td>0.33</td>
</tr>
<tr>
<td>129..136</td>
<td>1/2</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>0.86</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>137..144</td>
<td>1/2</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>1.65</td>
<td>1.64</td>
<td>1.60</td>
</tr>
<tr>
<td>145..151</td>
<td>1/1</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>1.65</td>
<td>0.86</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>152..158</td>
<td>3/0</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>0.74</td>
<td>0.74</td>
<td>0.38</td>
<td>0.38</td>
<td>0.19</td>
</tr>
<tr>
<td>159..164</td>
<td>0/1</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>2.23</td>
<td>2.23</td>
<td>2.21</td>
<td>3.51</td>
</tr>
<tr>
<td>165..170</td>
<td>0/2</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>2.23</td>
<td>2.23</td>
<td>2.21</td>
<td>3.51</td>
</tr>
<tr>
<td>171..173</td>
<td>0/1</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>2.23</td>
<td>1.16</td>
<td>1.16</td>
<td>2.80</td>
</tr>
<tr>
<td>174..175</td>
<td>0/3</td>
<td>0/1</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>2.23</td>
<td>2.23</td>
<td>2.21</td>
<td>3.51</td>
</tr>
<tr>
<td>176</td>
<td>0/2</td>
<td>1/0</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>2.23</td>
<td>1.16</td>
<td>1.16</td>
<td>2.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$F^{(k)}_{n}$</th>
<th>279.00</th>
<th>101.70</th>
<th>86.22</th>
<th>146.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{n}$</td>
<td>207</td>
<td>227</td>
<td>45</td>
<td>145</td>
</tr>
<tr>
<td>$c_{n} \div F^{(k)}_{n}$</td>
<td>0.74</td>
<td>2.23</td>
<td>0.52</td>
<td>0.99</td>
</tr>
</tbody>
</table>
a slight preference for car ownership. As with the example for list-based IPF, the table is shown in condensed form. Ye et al. (2009) also provide a geometric explanation of the algorithm, however, a proof of convergence is missing.

In the basic formulation, fitting against multi-dimensional controls requires one column per target value; memory requirements increase exponentially with each control dimension. To avoid this, the algorithm can be implemented to operate directly on the hierarchical reference sample, similarly to the sparse list approach for classical IPF.

2.3.2 Entropy optimization

The Ent algorithm seeks to obtain the desired household weights by solving a constrained optimization problem. Relative entropy (Kullback and Leibler, 1951) has been widely applied in urban and regional modeling as a measure of similarity between distributions (e.g., Wilson, 1970), and can be used as objective function for the problem at hand. Ent directly optimizes an information-based similarity metric of the weights, and therefore introduces the least possible amount of new information—just like IPF.

This subsection describes the algorithms presented by Bar-Gera et al. (2009) and Lee and Fu (2011). Both use objective functions based on relative entropy. In (Bar-Gera et al., 2009), the optimization problem is formulated as follows: Minimize

$$\sum_y f_y \left( \ln \left( \frac{f_y}{f_y^{(0)}} \right) - 1 \right)$$

subject to Eq. (2.4), the same constraints as used for IPU. Lee and Fu (2011) suggest optimizing relative entropy instead:

$$I = \sum_y f_y \ln \left( \frac{f_y}{f_y^{(0)}} \right).$$

Here, $f_y^{(0)}$ denotes the prior (initial) weights of the reference sample. Note that the difference between the objective functions (2.5) and (2.6) vanishes if there is at least one household-level constraint, because in this case the sum of all expansion factors $\sum_y f_y$ is fixed. In addition, Bar-Gera et al. (2009) present a relaxed formulation where deviation from target values is
optionally penalized in the objective function instead of specifying hard constraints.

To compute optimal weights, Bar-Gera et al. (2009) suggest iterative adjustment of the weights, so that after each adjustment one target value is satisfied. A direct solution can be obtained through numerical methods. Lee and Fu (2011) suggest the Broyden-Fletcher-Goldfarb-Shanno algorithm (see, e.g., Nocedal and Wright, 2006), other methods may be suited better for this kind of problem. Experiments suggest that this approach clearly outperforms iterative adjustment of the weights, however no formal runtime measurements have been carried out so far.

2.3.3 Relation matrix

ALBATROSS uses the concept of a relation matrix to estimate a composition of households that perfectly matches person-level constraints. The relation matrix is a specially formed contingency table that helps estimating the distribution of single and male-female households for a human population given a distribution of individuals. This section presents the structure of the simplest relation matrix, and the two relation matrices used in Arentze et al. (2007).

The simplest relation matrix is a $2 \times 2$ matrix as shown in Fig. 2.6(a). The total number of females and males is denoted by $X$ and $Y$, while $X_a$ and $Y_a$ denote the number of independently living females and males, respectively. The row controls fix the total number of females, $X$, in the first row, and the total number of males that live independently, $Y_a$, in the second row. Conversely, the column controls define the number of males, $Y$, and single females, $X_a$. The interior cells of the table contain household counts: The top left cell represents the number of couples $H_{♀♂}$, while the bottom left and top right cells denote the number of single-person male or female households, $H_{♂}$ and $H_{♀}$, respectively. The bottom-right cell is a structural zero: A household consisting of two persons living independently is a contradiction in itself.

Due to the (simplified) assumption that each individual is either member of a two-person household or living independently, the following equation must hold:

$$H = X + Y_a = Y + X_a.$$
not true if the control totals are further disaggregated. Arentze et al. (2007) define two relation matrices (shown in Figure 2.6(b)) by disaggregating the first row and first column by age and work status, respectively, introducing 14 new target values, seven per gender: $X_1 \ldots X_7$ and $Y_1 \ldots Y_7$. These matrices are extended by one column and one row for persons that live in a household but are not head thereof (e.g., children or retired parents), controlled by $X_w$ and $Y_w$. The age relation matrix contains further structural zeros, as children cannot act as head of a household.

As with classical IPF, the relation matrices are initialized with data from the reference sample. After performing IPF on these matrices, the resulting distributions are used as controls for another IPF run on a household-level seed.

### 2.4 Generation

In the generation stage, a disaggregate set of persons and households with attributes required by the microsimulation model is constructed. The result of the preceding fitting procedure, shown in the previous two sections, is a fractional expansion factor for each household or person in the reference sample. The generation of the synthetic population is based on these expansion factors. According to Bowman (2009c), the result of the generation stage is a population where each member of the generated population has clearly defined attributes, the sum within each control category matches the corresponding control total, and the interactions present in the reference sample can be observed in the generated population. In practice, this is achieved by cloning observations from the reference sample as often as required by the control totals. Section 2.5 sketches alternative approaches.

The remainder of this section presents several selection algorithms applicable to results of single-level fitting algorithms where the weights satisfy either household-level or person-level controls but not both: Alterations to the household selection procedure independently proposed by Auld and Mohammadian (2010), Guo and Bhat (2007), and Srinivasan et al. (2008), and household construction introduced by Pritchard and Miller (2012). Finally, the advantage of multi-level fitting is highlighted, where the selection procedures are much more straightforward. Enrichment with
Figure 2.6: Relation matrix

(a) Computing household-level constraints from person-level constraints

<table>
<thead>
<tr>
<th></th>
<th>total ♂</th>
<th>independent ♂</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>total ♂</td>
<td>H♀♂</td>
<td>H♀</td>
<td>X</td>
</tr>
<tr>
<td>independent ♂</td>
<td>H♂</td>
<td>0</td>
<td>Yₐ</td>
</tr>
<tr>
<td>Σ</td>
<td>Y</td>
<td>Xₐ</td>
<td>H</td>
</tr>
</tbody>
</table>

(b) Computing household-level constraints by age and work status

<table>
<thead>
<tr>
<th>age</th>
<th>0–14</th>
<th>15–34</th>
<th>35–64</th>
<th>65+</th>
<th>indep.</th>
<th>cohab.</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>♂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0–14</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15–34</td>
<td></td>
<td>2-head</td>
<td>1-head</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35–64</td>
<td></td>
<td>♂+♂</td>
<td>♂</td>
<td>hh-s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂ indep.</td>
<td>0</td>
<td>1-head</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>♂ cohab.</td>
<td></td>
<td>♂ cohab.</td>
<td>♂ cohab.</td>
<td>hh-s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td>Y₁</td>
<td>Y₂</td>
<td>Y₃</td>
<td>Y₄</td>
<td>Xₐ</td>
<td>X₆</td>
<td>H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>work status</th>
<th></th>
<th>♂</th>
<th>♂</th>
<th>♂</th>
<th>indep.</th>
<th>cohab.</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>♂</td>
<td>2-head</td>
<td>1-head</td>
<td>♂ cohabs.</td>
<td>X₅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂ cohabs.</td>
<td>♂+♂ households hh-s</td>
<td>♂ cohabs.</td>
<td>X₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂ indep.</td>
<td>1-head ♂ cohabs.</td>
<td>♂ cohabs.</td>
<td>X₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂ cohab.</td>
<td>♂ cohabs.</td>
<td>♂ cohabs.</td>
<td>Yₐ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td>Y₅</td>
<td>Y₆</td>
<td>Y₇</td>
<td>Xₐ</td>
<td>X₆</td>
<td>H</td>
<td></td>
</tr>
</tbody>
</table>

Source: Arentze et al. (2007)
further information, such as refined geographic placement at the sub-zone level or additional attributes, is beyond the scope of this chapter.

2.4.1 Biased household selection

For household selection, both PopSynWin and CEMDAP use stochastic procedures that favor households consisting of persons in categories still underrepresented in the generated population. As a result, after completion of the selection procedure, person-level controls are matched, albeit approximately (Auld and Mohammadian, 2010; Guo and Bhat, 2007).

The FSUMTS synthesizer uses a similar, entirely deterministic approach. A fitness value, defined for each household, is used as choice criterion. The fitness value is a measure for the adherence to both household- and person-level constraints, given the already selected households. After computing fitness values for all households, the household with the largest fitness value is selected. This process is repeated until all fitness values fall below zero (Srinivasan et al., 2008).

In all three approaches, every household’s selection probability potentially changes after each iteration, which may result in prohibitive run times. Only PopSynWin addresses this issue by implementing a heuristic that skips most recomputations of the selection probability (Auld and Mohammadian, 2010). The nonincreasing nature of the fitness values for the FSUMTS synthesizer might allow a reduction of computational effort, but the authors do not provide run times nor implementation details.

2.4.2 Synthesizing household-person relationships

The reference sample available for the ILUTE model is unique in the sense that it does not contain links between households and persons (Pritchard and Miller, 2012). Household- and person-level distributions are estimated independently; they are subsequently fitted against each other to ensure consistency. It is only in the generation stage that persons are assigned to households.

Households are repeatedly selected according to their selection probabilities. Each member of the current household is then drawn from the subset of eligible persons according to the selection probabilities at person
level. This ensures approximate consistency with the control totals at both household and person levels.

The synthesis of relationships can be carried out as described above even if the household-person relationships are present in the input data. According to Pritchard and Miller (2012), one should consider whether variation in household composition outweighs the drawback of purely synthetic household-person relationships.

### 2.4.3 Direct generation from household-level weights

PopGen and ALBATROSS simultaneously control at person and household level during the fitting stage. The additional effort pays off for the generation stage: For the construction of the final population, the households only need to be selected according to their estimated expansion factor. Feasible selection schemes include

- Monte Carlo random sampling with replacement, where the selection probability is proportional to the expansion factor (Ye et al., 2009),
- combinatorial optimization, using the estimated weights as an initial solution or as selection probability in a genetic algorithm (Williamson et al., 1998) or in a mixed-integer linear program (Choupani and Mamdoohi, 2015)
- the “Truncate-Replicate-Sample” algorithm by Lovelace and Ballas (2013), which is computationally cheaper than combinatorial optimization but maintains population totals better than simple random sampling.

The latter benefits from a computationally efficient implementation of random sampling without replacement, which is presented in Chapter 8.

### 2.5 Beyond reweighting

For completeness, alternative approaches which were not considered in this chapter are briefly mentioned here.

Combinatorial optimization techniques such as integer programming, genetic algorithms, or simulated annealing, can be used at either the fitting or the generating stage or both. Barthelemy and Toint (2012) employ tabu search to combine households from a pool of individuals. The use of combinatorial optimization for the purpose of generating a population
of firms is evaluated in Ryan et al. (2009). Voas and Williamson (2000) generate a synthetic microdataset to predict the spatial distribution of a group of persons which is not directly available from the data.

All reviewed synthesis procedures copy observations from the reference sample to the generated population, if necessary more than once. This may amplify spurious correlations in the sample if the attribute space is large in relation to the size of the reference sample. As noted in Section 2.2.1, IPF in the fitting stage is essentially equivalent to estimating a loglinear model for the multivariate data. Adding heterogeneity to the generated population on both household and person levels can be achieved by substituting IPF with other models, such as Markov Chain Monte Carlo (Farooq et al., 2015; Farooq et al., 2013), Bayesian networks (Sun and Erath, 2015; Pfeifhofer, 2014), copulas (Kao et al., 2012), or multivariate density estimation (Hamada et al., 2015).

Sampling from the conditional distributions defined by census cross-tabulations, without a reference sample, is sometimes referred to as synthetic reconstruction (Williamson et al., 1998), and can also be viewed as a Bayesian network if the dependency structure is cycle-free. Alfons et al. (2011) suggest stepwise model-based imputation, which is implemented in the simPop extension for the R statistical software package (R Core Team, 2017).

2.6 Conclusion

Six population synthesis procedures used for various microsimulation models were reviewed. All of them use a variant of IPF for the fitting stage, and are able to synthesize a population that fits the control totals at both person and household levels. To overcome memory limitations with many attributes, detailed categorization, or multi-zone synthesis, the sparse list data structure seems promising.

Many synthesizers are tied to specific input data sets (e.g., PopGen is heavily geared towards US data). Source code seems to be available only for PopGen, in some cases not even binary executables are available upon request.

Control at both person and household level is implemented differently by all synthesizers. Several approaches to the generation stage are presented.
Biased stochastic or deterministic selection, and synthesis of household-person relationships, are useful if the weights obtained from the fitting stage are consistent with controls only at person or only at household level. The relation matrix is an elegant way to combine household-level and person-level distributions. A sample of households with detailed information on their constituent persons can be fitted against control totals at both person and household levels using the new IPU algorithm.

The simplicity of the generation stage given accurate household weights prompts further investigation of multi-level fitting methods. Chapter 3 presents two other algorithms similar to IPU in a common framework. As shown in Chapter 4, one of these algorithms is a special case of a reweighting technique commonly used in the domain of survey statistics. Chapter 5 presents an open-source implementation of multi-level fitting algorithms for the statistical software package R (R Core Team, 2017), which is applied to large datasets in Chapter 6.
Chapter 3

Multi-level fitting algorithms

This chapter describes a novel algorithm to fit a household sample with person information to given control totals. The surprising similarity of this algorithm to two other algorithms, despite seemingly different original formulations, is demonstrated at the pseudocode level. Convergence issues are discussed briefly. A link to survey statistics is established in the next chapter, simulation results are presented in Chapters 5 and 6.

3.1 Introduction

In multi-person households, other household members may affect personal decisions (Jones et al., 1983). Therefore, properly replicating the household structure is necessary to be able to simulate these interactions. This chapter focuses exclusively on algorithms that generate populations of persons grouped into households. The following data are assumed:

- a representative reference sample that contains the characteristics of households and all persons in each household
- control totals for selected categorical attributes on both household and person levels.

As described in the previous chapter, generating synthetic populations using the reweighting method consists of two principal stages: fitting and generation. In the deterministic fitting stage, a disaggregate sample of households

---

1Parts of this chapter are published as conference papers (Müller and Axhausen, 2011a, 2012).
households (called reference sample) is reweighted. This yields a positive, usually non-integer expansion factor (or weight) for each household. The reweighted reference sample corresponds to the full population of the study area and is required to satisfy aggregate constraints (referred to as control totals or controls). The constraints specify target values for categories of individuals or households. It is crucial that the new weights are similar to the original weights, so that the correlations observed in the reference sample are preserved to the widest extent possible. This expansion factor is used to construct persons and households in the subsequent generation stage, which usually is of stochastic nature.

Expansion factors that satisfy household-level control totals (without considering the controls at person level) can be computed using the well-known IPF algorithm (Deming and Stephan, 1940). In order to also satisfy the person-level controls, several studies suggest heuristics that adapt the generation stage so that households with persons in still underrepresented categories are preferred (Auld and Mohammadian, 2010; Srinivasan and Ma, 2009; Guo and Bhat, 2007). This approach, single-level fitting with biased selection, is sketched in Fig. 3.1, see also Section 2.4.1.

In contrast, multi-level algorithms compute household-level expansion factors that satisfy the controls at both household and person levels. The expansion factors define a probability distribution of the households, and contain all the information necessary to satisfy the control totals. Construction of the final population is possible using randomized rounding (Lovelace and Ballas, 2013) or even weighted random sampling with replacement, ignoring control totals. Figure 3.2 illustrates this strategy.

Household-level expansion factors, and the ability to ignore control totals during generation, are helpful for generating many instances of similar yet different populations, akin to multiple imputation (Rubin, 1987). In contrast, single-level fitting algorithms with biased selection still need to consider control totals in the generation stage. Furthermore, the biased selection has only limited control over the distortion of the weights. The remainder of this chapter focuses on multi-level fitting procedures only, in particular on Iterative Proportional Updating (IPU) by Ye et al. (2009), Entropy Optimization (ENT) independently by Bar-Gera et al. (2009) and Lee and Fu (2011) which both were introduced in the previous chapter, and Hierarchical Iterative Proportional Fitting (HIPF), which will be presented in this chapter.
3.2 Hierarchical Iterative Proportional Fitting

This section briefly summarizes the list-based version of the IPF algorithm as described by Pritchard and Miller (2009) for fitting a sample of households. The subsequent presentation of the HIPF algorithm uses list-based IPF as a building block.

3.2.1 Fitting households

The list-based version of IPF computes nonnegative fractional expansion factors for all households \( h \) of the reference sample. These expansion factors satisfy all control totals at the household level. In the following, the notation is defined briefly. It builds upon the notation used for Section 2.3.1.

Similarly to Section 2.2.3, the disaggregate reference sample is given as a finite sequence of category tuples

\[
(H_h)_{h=1}^n = ((a_1, b_1, \ldots), \ldots, (a_n, b_n, \ldots)) \text{ with } H_h \in a \times b \times \ldots.
\]

The sets of household indexes \( H_a \) and \( H_b \) for given categories \( a \in a \) or \( b \in b \) are therefore defined as follows:

\[
H_a := \{ h \in \{1, \ldots, n\} : (a_h, \ldots) = (a, \ldots) \},
\]

\[
H_b := \{ h \in \{1, \ldots, n\} : (\ldots, b_h, \ldots) = (\ldots, b, \ldots) \}.
\]

These and the following definitions can be generalized for more than two controlled attributes or for controlling several attributes jointly.

An expansion factor \( f_h \in \mathbb{R}_+ \) is defined for each household. In addition, the sums \( F_a \) and \( F_b \) of all expansion factors for households of categories \( a \) or \( b \) are defined:

\[
F_a := \sum_{h \in H_a} f_h, \quad F_b := \sum_{h \in H_b} f_h.
\]

The set of control totals for attribute \( a \) is denoted by \( C_a \), and the target
Figure 3.1: Illustration of single-level fitting algorithms

Control totals (person level)

Control totals (household level)

Synthetic population

Result of fitting

Reference sample

Chapter 3. Multi-level fitting algorithms
Figure 3.2: Illustration of multi-level fitting algorithms
value for category $a$ (of attribute $\mathbf{a}$) is denoted by $C_a$. It is assumed that $C_a$ and $C_b$ for all $a \in \mathbf{a}$ and $b \in \mathbf{b}$ are given. It is a trivial precondition for the convergence of IPF that the grand totals match for all attributes: $N = \sum_a C_a = \sum_b C_b$.

The objective of fitting at the household level can be formulated as follows:

$$F_x = C_x$$

for all categories $x$ of all controlled household attributes $\mathbf{x}$.

(3.1)

Algorithms 3.1 to 3.3 show the list-based IPF routine, which computes expansion factors that satisfy the constraints in Eq. (3.1), in the context of multi-level fitting. The main routine is shown in Algorithm 3.1. If person-level attributes are not controlled for (as in the case of the original list-based IPF procedure), the P-FIT call in line 3 is a no-op, and the information on persons (denoted by $P$, $C_\alpha$, and $C_\beta$, see Section 3.2.2) is not required. In any case, the H-FIT routine (Algorithm 3.2) is called repeatedly, which in turn invokes the H-ADJUST routine (Algorithm 3.3) for each category of each control attribute. The H-ADJUST routine rescales the expansion factors so that the current control category is satisfied perfectly.

The ML-FIT routine terminates as soon as the constraints (3.1) hold approximately. Convergence depends on the input data. Pukelsheim (2013) show necessary and sufficient conditions for the convergence of IPF, derivable directly from the input data. As a rule of thumb, using a large and diverse reference sample avoids convergence problems. In the following, it is assumed that the reference sample and the control totals are well-conditioned, see Section 3.4.2 for further discussion of convergence issues.

3.2.2 Fitting persons

The reference sample describes not only the households, but also the persons in these households. This information is provided by another finite sequence of attribute tuples:

$$(P_p)_p^m = ((\alpha_1, \beta_1, \ldots, h_1), \ldots) \text{ with } P_p \in \alpha \times \beta \times \cdots \times \{1, \ldots, n\}.$$  

For each person $p$, the special attribute $h_p$ specifies the household index of that person.
Algorithm 3.1 Procedure ML-FIT\((H, P, C_a, C_b, \ldots, C_{\alpha}, C_{\beta}, \ldots)\) – the main routine of the generic framework

Require: Reference sample \((H, P)\)

Require: Household-level control totals \(C_a, C_b, \ldots\)

Require: Person-level control totals \(C_{\alpha}, C_{\beta}, \ldots\)

Require: Prior weights \(f_h\) (default: uniform)

Ensure: Expansion factors \(f_h\) obeying all control totals

1: while convergence not reached do
2: \(f_h \leftarrow H\text{-FIT}(H, f_h, C_a, C_b, \ldots)\)
3: \(f_h \leftarrow P\text{-FIT}(P, f_h, C_{\alpha}, C_{\beta}, \ldots)\)
4: end while
5: return \(f_h\)

Algorithm 3.2 Subroutine H-FIT\((H, f_h, C_a, C_b, \ldots)\) – fitting at household level

Require: Households \(H\) from reference sample

Require: Current expansion factors \(f_h\)

Require: Household-level control totals \(C_a, C_b, \ldots\)

Ensure: Improved expansion factors \(f_h\) for all household-level controls \(C_x \in \{C_a, C_b, \ldots\}\)

for all household-level controls \(C_x \in \{C_a, C_b, \ldots\}\) do

for all categories \(x \in x\) do

\(f_h \leftarrow H\text{-ADJUST}(H, f_h, x, C_x)\)

end for

end for

return \(f_h\)

Algorithm 3.3 Subroutine H-ADJUST\((H, f_h, x, C_x)\) – fitting one category at household level

Require: Current household-level category \(x\)

Require: Value \(C_x\) of the control total for category \(x\)

Ensure: Expansion factors \(f_h\) that satisfy control \(C_x\)

\(r \leftarrow C_x \div F_x\)

\(f_h \leftarrow f_h \cdot r\) for all \(h \in H_x\)

return \(f_h\)
In analogy to the previous subsection, the sets $P_\alpha$ and $P_\beta$ are the set of person indexes that fall within category $\alpha$ or $\beta$:

$$P_\alpha := \{ p \in \{1, \ldots, m\} : (\alpha_p, \ldots) = (\alpha, \ldots) \},$$

$$P_\beta := \{ p \in \{1, \ldots, m\} : (\ldots, \beta_p, \ldots) = (\ldots, \beta, \ldots) \}.$$ 

Again, these and the following definitions can be generalized for more than two controlled attributes or for controlling several attributes jointly.

The sets of controls $C_\alpha$ and $C_\beta$, the target values $C_\alpha$ and $C_\beta$, and the grand total $\nu = \sum_\alpha C_\alpha = \sum_\beta C_\beta$ are also defined in the same way as for households. In addition, the sums $F_\alpha$ and $F_\beta$ of all expansion factors for persons of category $\alpha$ is defined:

$$F_\alpha := \sum_{p \in P_\alpha} f_{h_p}, \quad F_\beta := \sum_{p \in P_\beta} f_{h_p}.$$ 

For a household index $h$, the number of constituent members in that household is denoted by $p_h := |P_h|$. The following defines the objective for fitting at the person level:

$$F_\xi = C_\xi \text{ for all categories } \xi \text{ of all controlled person attributes } \xi.$$ (3.2)

The ML-FIT routine shown in Algorithm 3.1 can also be used to compute person-level expansion factors that satisfy only person-level control totals by supplying $P$, $C_\alpha$, and $C_\beta$ instead of $H$, $C_a$, and $C_b$. The simultaneous fitting for both levels of aggregation is discussed in the next subsection.

### 3.2.3 Fitting households and persons

The principal idea of the HIPF algorithm is to temporarily switch domains by converting household-level expansion factors to the person level, and vice versa. An expansion factor $\phi_p \in \mathbb{R}_+$ is defined for each person. The P-FIT routine (line 3 in Algorithm 3.1) could be defined as follows:

1. Convert the household-level expansion factors to person-level expansion factors: $\phi_p := f_{h_p}$.
2. Use H-FIT to compute person-level expansion factors that satisfy person-level control totals, with $\phi_p$ as initial values, and using $P$, $C_\alpha$, and $C_\beta$ instead of $H$, $C_a$, or $C_b$. 

38
Subroutine $P$-FIT-HIPF($P$, $f_h$, $C_\alpha$, $C_\beta$, ...) – person-level fitting for the HIPF algorithm

Require: Persons $P$ from reference sample
Require: Current expansion factors $f_h$
Require: Person-level control totals $C_\alpha$, $C_\beta$, ...
Ensure: Improved expansion factors $f_h$

1: $\phi_p \leftarrow f_{hp}$ for all $p \in P$
2: $\phi_p \leftarrow H$-FIT($P$, $\phi_p$, $C_\alpha$, $C_\beta$, ...)
3: $f_h \leftarrow p_h^{-1} \cdot \sum_{p \in P_h} \phi_p$ for all $h \in H$
4: modify $f_h$ so that $\sum hp_h f_h = \nu$ and $\sum h f_h = N$ (see Appendix A)
5: return $f_h$

3. Convert the person-level expansion factors to household-level expansion factors by averaging: $f_h := \frac{1}{p_h} \sum_{p \in P_h} \phi_p$.

Most likely, the resulting household-level expansion factors will lead to a misfit for both person- and household-level controls. However, the expansion factors contain information from both levels; one could expect that repeated execution of the procedure outlined above eventually converges to a solution that satisfies all controls.

In practice, one crucial step is missing to achieve convergence. The process described above does not control for the average number of persons per household, which is fixed implicitly by the ratio $\frac{\nu}{N}$ with $N = \sum_{a \in a} C_a$ and $\nu = \sum_{\alpha \in \alpha} C_\alpha$. This issue is resolved by tweaking the household-level expansion factors $f_h$ to fix the average number of persons per household. From the potentially infinite set of feasible solutions, the one that minimizes the relative entropy (Kullback and Leibler, 1951) is chosen. A detailed analysis of the underlying optimization problem is presented in Appendix A; notably, the resulting ratio of new vs. old expansion factors is the same for all households of the same size.

Algorithm 3.4 shows a pseudocode for the $P$-FIT-HIPF routine, the HIPF version of the $P$-FIT routine, as described above. The proposed method, consisting of Algorithms 3.1 to 3.4, can be seen as a natural extension of the IPF algorithm (cf. $H$-ADJUST, Algorithm 3.3), because each step performs a groupwise uniform rescaling of expansion factors, where the grouping is defined by the currently “active” control or the household size.


Algorithm 3.5 Subroutine P-FIT($P, f_h, C_\alpha, C_\beta, \ldots$) – fitting at person level

Require: Persons $P$ from reference sample
Require: Current expansion factors $f_h$
Require: Person-level control totals $C_\alpha, C_\beta, \ldots$
Ensure: Improved expansion factors $f_h$

for all person-level controls $C_\xi \in \{C_\alpha, C_\beta, \ldots\}$ do
    for all categories $\xi \in \xi$ do
        $f_h \leftarrow$ P-ADJUST($P, f_h, \xi, C_\xi$)
    end for
end for

return $f_h$

3.3 A common framework

This section shows how the Iterative Proportional Updating (IPU) and Entropy Optimization (Ent) algorithms can be formulated in the algorithmic framework introduced in the previous section. Unlike HIPF, both IPU and Ent operate exclusively on household-level expansion factors and do not transform them to the person level. Therefore, the main difference is the implementation of the P-FIT routine. Both algorithms can be formulated using the pseudocode shown in Algorithm 3.5, which adjusts the expansion factors for all controls, just like H-FIT adjusts for controls at the household level. The IPU and Ent algorithms differ only by the implementation of the P-ADJUST routine.

3.3.1 Iterative Proportional Updating

The IPU algorithm uses the following adjustment at person level: Proportionally rescale the weights of all households that contain at least one person that falls into the current category $\xi$ (see Section 2.3.1). Expansion factors for households that do not contain such a person are not modified. The rescaling proportion for a household does not depend on the number of constituent persons that fall into the current category $\xi$. The procedure P-ADJUST-IPU in Algorithm 3.6 formalizes this.
Algorithm 3.6 Subroutine P-ADJUST-IPU\((P, f_h, \xi, C_\xi)\) – fitting one category at person level for the IPU algorithm

**Require:** Current person-level category \(\xi\)

**Require:** Value \(C_\xi\) of the control total for category \(\xi\)

**Ensure:** Expansion factors \(f_h\) that match control \(C_\xi\)

\[
H_\xi \leftarrow \{h_p : p \in P_\xi\}
\]

\[
r \leftarrow C_\xi \div F_\xi
\]

\[
f_h \leftarrow f_h \cdot r \text{ for all } h \in H_\xi
\]

return \(f_h\)

### 3.3.2 Entropy Maximization

The \(\text{Ent}\) algorithm computes weights that minimize the objective functions in Eq. (2.5) or Eq. (2.6) subject to the constraints in Eq. (2.4), cf. Section 2.3.2. Although this seems very different to IPU and HIPF, which perform iterative modifications to the weights, the \(\text{Ent}\) approach can also be fit into the framework shown in Algorithms 3.1 to 3.3 and 3.5: The argument is based on the objective function (2.5), which is equivalent to Eq. (2.6) if at least one household-level constraint is used. In the notation of this chapter, this means to optimize

\[
\sum_h f_h \left( \ln \left( \frac{f_h}{f_h^{(0)}} \right) - 1 \right)
\]

subject to the constraints in Eqs. (3.1) and (3.2).

Applying the method of Lagrange Multipliers to the constraints (3.1) and (3.2) and the objective function (3.3) yields the following auxiliary function with all \(f_h, \lambda_x, \lambda_\xi\) as parameters:

\[
\sum_h f_h \left( \ln \left( \frac{f_h}{f_h^{(0)}} \right) - 1 \right) + \sum_x \lambda_x (F_x - C_x) + \sum_\xi \lambda_\xi (F_\xi - C_\xi)
\]

Here, each \(\lambda\) corresponds to the target value for one category at household
or person level. Partial derivation of the auxiliary function with respect to each variable yields the following necessary condition for each $f_h$:

$$\ln\left(\frac{f_h}{f_h^{(0)}}\right) - \left(\lambda_{a_h} + \lambda_{p_h} + \ldots\right) - \sum_{p \in P_h} \left(\lambda_{\alpha_p} + \lambda_{\beta_p} + \ldots\right) = 0,$$

which can be rewritten as

$$f_h = f_h^{(0)} \cdot \exp\left(\left(\lambda_{a_h} + \lambda_{b_h} + \ldots\right) + \sum_{p \in P_h} \left(\lambda_{\alpha_p} + \lambda_{\beta_p} + \ldots\right)\right) \quad (3.4)$$

and plugged back into the constraints in Eqs. (3.1) and (3.2).

Bar-Gera et al. (2009) suggest adjusting the $\lambda$ in a round-robin fashion, so that after each adjustment the corresponding constraint from Eq. (3.1) or Eq. (3.2) is satisfied. Each adjustment step boils down to finding the unique positive real-valued root of a polynomial, which can be carried out using numerical methods. For the adjustment of $\lambda_a$ or $\lambda_b$ for a household-level constraint, this polynomial is just a linear equation, and the solution is equivalent to the one obtained by H-ADJUST. The procedure P-ADJUST-ENT in Algorithm 3.7 summarizes the adjustment for person-level constraints $\lambda_\alpha$ or $\lambda_\beta$. Note that the execution of line 5 in P-ADJUST-ENT is equivalent to recomputing the $f_h$ with the new $\lambda^\xi$ from the closed-form expression for $f_h$ in Eq. (3.4). A household’s weight is readjusted depending on how many persons of the given category it contains – the more matching persons, the stronger. Informally speaking, it “pays off” more to adjust a household with many matching persons.

### 3.4 Numerical issues

This section briefly discusses stability and convergence issues.

#### 3.4.1 Stability

Instead of updating the $f_h$ and potentially accumulating floating-point errors, a robust implementation of the algorithm should recompute $f_h$ from scratch for each iteration. For ENT, the estimable parameters $\lambda$ define the solution. It is possible to derive a similar set of estimable parameters for
Algorithm 3.7 Subroutine \textbf{P-ADJUST-ENT}(P, f_h, \xi, C_{\xi}) – fitting one category at person level for the Ent algorithm

\textbf{Require:} Current person-level category \(\xi\)

\textbf{Require:} Value \(C_{\xi}\) of the control total for category \(\xi\)

\textbf{Ensure:} Expansion factors \(f_h\) that match control \(C_{\xi}\)

1: Find \(\lambda'_{\xi}\) that satisfies the constraint corresponding to \(\xi\) in the constraint equations (3.2)
2: \(r \leftarrow \exp\left(\lambda'_{\xi} - \lambda_{\xi}\right)\)
3: \textbf{for all} \(h \in H\) \textbf{do}
4: \(i \leftarrow |P_h \cap P_{\xi}|\)
5: \(f_h \leftarrow f_h \cdot r^i\)
6: \textbf{end for}
7: \textbf{return} \(f_h\)

both IPU and HIPF by defining a parameter for each target value (and in the case of HIPF a parameter for each household size). The transformation of the algorithms from the weights space to the parameter space is beyond the scope of this chapter.

3.4.2 Convergence

While in general a multi-level fitting problem has arbitrarily many solutions, it is possible that a multi-level fitting problem does not have a solution at all. Two examples are described below. Both can also occur with IPF at a single level, and the matrix notation is used to depict them.

\textbf{Missing observations} If the target value for a category is nonzero, and there is no corresponding observation in the reference sample, a division by zero occurs during execution of IPF. This is also referred to as the \textit{zero-cell problem}, illustrated in Fig. 3.3(a). Various remedies have been suggested in the literature, such as introducing the missing observations with arbitrarily small weights, combining rare categories, or borrowing from other regions, see the previous chapter for an overview.

\textbf{Conflicting controls} If an observation is unique within two control categories, and the target values for these categories differ, this observation’s expansion factor will oscillate between the two target values.
Figure 3.3: Two possible reasons for non-convergence of IPF

(a) Missing observations ($C_a \neq 0$)

\[
\begin{array}{cccc}
\vdots & 0 & \vdots & \\
\vdots & 0 & \ddots & \\
\vdots & 0 & \ddots & \\
0 & \cdots & C_a & \cdots \\
\end{array}
\]

(b) Conflicting target values ($C_a \neq C_b$)

\[
\begin{array}{cccc}
\vdots & 0 & \vdots & \\
\vdots & 0 & \ddots & \\
0 & \cdots & 0 & 1 \\
0 & \cdots & 0 & \cdots & 0 & C_b \\
\vdots & 0 & \vdots & \\
\vdots & 0 & \ddots & \\
0 & \cdots & C_a & \cdots \\
\end{array}
\]

values. Figure 3.3(b) shows an example for the above case. More complicated settings are possible, even more so when taking into account person-level controls: For instance, the persons in a unique household will be replicated as often as the household, and this can be incompatible with person-level controls. Most of the methods presented in this chapter will happily accept
such malformed input but fail to converge. The only exception is the formulation of the \texttt{ENT} algorithm as a constrained optimization problem: Here, conflicting target values render the optimization problem intractable, and the nonlinear solver may offer a partial solution. \cite{Bar-Gera2009} suggest a relaxed formulation of the problem where deviation is tolerated for some controls.

Strictly speaking, if control totals are incompatible with the reference sample, then the reference sample is not representative of the region to which the control totals correspond. A reference sample with too few observations, or control totals with too many categories, can trigger the problems described above. Before tweaking target values, it seems worthwhile to consider if the reference sample can be enlarged or if fewer control categories are acceptable.

In any case, the reference sample and the control totals should refer to the same populations. Chapter 7 discusses an example where the reference sample does not cover persons aged 5 or below; the control totals must be corrected for differences between populations.

For the algorithms presented in this chapter, there seems to be no way to determine beforehand if convergence can be achieved for a given problem instance. The next chapter presents yet another approach, from the domain of survey statistics, where the existence of a solution can be gauged very quickly.

\section*{3.5 Conclusion}

This chapter discusses the problem of computing household-level expansion factors for a reference sample consisting of persons grouped to households, and control totals at both levels. Two general approaches to this problem are distinguished: single-level fitting with biased selection, and multi-level fitting.

A novel multi-level fitting algorithm, HIPF, is introduced. Just like IPF, it proceeds in iterations and performs a sequence of adjustments of the weights. Multi-level fitting is achieved by repeatedly switching between the household and the person domain. Each individual weight adjustment step adds only a minimum amount of new information. The notation of the
algorithm is geared towards a common framework that also accommodates two other similar algorithms, IPU and Ent.

In this framework, HIPF requires a specialized routine for adjusting weights to match person-level controls. In contrast, that part is almost identical for IPU and Ent: Just like the household-level controls, all person-level controls are considered and weights are adapted to satisfy the currently active control. The only difference between IPU and Ent lies in the procedure that adjusts weights for a given person-level control, and there only in the treatment of households where at least two persons have the same category for one of the person-level attributes. For each person-level category, IPU only discriminates if a household has at least one person that matches this category. In contrast, if a household contains two or more persons of the same category, Ent reweights this households stronger than those households with only one person of this category. Both algorithms do not change the weights of unaffected households.

The placement of HIPF, IPU and Ent in the common framework suggests an inherent similarity between these algorithms. This chapter focuses on a comparison at the algorithmic level, see Chapters 5 and 6 for simulation results and (Fourie and Müller, 2011) for an application to Singapore. Most methods are described in terms of the steps necessary to arrive at a solution; only the Ent algorithm can be defined as an optimization problem with a unique solution.

The next chapter shows that the Ent algorithm can be formulated as a special case of generalized raking, a technique used in survey statistics to correct surveys for sampling errors and nonresponse. Generalized raking can be formulated as an even simpler optimization problem, and allows a simple assessment if a solution exists, in contrast to the methods presented in this chapter.
Chapter 4

Survey calibration

The previous chapter presented three methods for the fitting stage of the multi-level fitting method in population synthesis. The methods boil down to computing weights for a data set with a hierarchical structure, such as persons in households. Weighting data sets is done frequently in the domain of survey statistics. This chapter presents generalized raking, a procedure that has been long known and used in that field but largely ignored by transportation planners. Equivalence of a special case of generalized raking to the ENT algorithm presented in the previous chapter is demonstrated formally. Finally the usefulness of generalized raking for the application of population synthesis is evaluated. Simulation results are presented in Chapters 5 and 6.

4.1 Introduction

Many treatments of synthetic population generation in the field of transport planning refer to the seminal paper by Beckman et al. (1996), which uses iterative proportional fitting (Deming and Stephan, 1940; Ireland and Kullback, 1968) as a cornerstone. The original application of IPF was the adjustment of a sample of the population to the known marginal totals. Since then, methodologies for the adjustment of survey samples to auxiliary information have evolved, and can be classified as follows:

- **Post-stratification** where the population is subdivided in disjoint strata for each of which the total is known,
- **Raking estimation** where the totals in each stratum are estimated beforehand using IPF.

1Parts of this chapter are published as a conference paper (Müller and Axhausen, 2012).
Regression estimation which seeks to obtain an unbiased linear model given known totals of the independent variables (Cassel et al., 1976).

Calibration estimation which offers an alternative view on regression estimation by weighting each observation (Deville and Särndal, 1992).

Generalized raking which presents further insights into, and a computational method for, calibration estimation (Deville et al., 1993).

Figure 4.1 shows a diagram that relates the methods to each other. Two methods are connected by an arrow if one is a superset of the other, e.g., calibration estimation can be configured to solve regression estimation problems.

The calibration estimators, as computed by generalized raking, are obtained by estimating weights for each observation in the sample so that known population totals are satisfied (Kim and Park, 2010). Furthermore, the sampling variance and nonresponse and noncoverage bias are reduced (Zhang, 2000). The methods above generally aim at improving the estimation of mean and variance of population totals of response variables, based on the assumption of a correlation between calibration and response variables.
The advances in the field of survey statistics do not seem to be widely applied within the transportation planning community, and in particular not within the population synthesis community. A notable exception is (Armoogum and Madre, 2002) which describes the usage of the generalized raking software for the calibration of a long-distance travel survey. The vast differences in terminology, notation, and perhaps application might be the reasons for this parallel development. This chapter closes that gap, therefore justifying the usage of theoretical results, algorithms and software implementations from survey statistics for the engineering problem at hand.

In the remainder of the chapter, generalized raking is described in some detail, and equivalence of a special case of generalized raking to the ENT algorithm from the previous chapter is shown. Before concluding, the application of generalized raking for population synthesis is evaluated in comparison with the ENT method.

4.2 Generalized raking

After introducing the notation, which closely follows (Deville et al., 1993), the problem is formulated as a constrained optimization problem, and a solution is sketched. The optimization problem can be defined using different distance functions, which are discussed at the end of this section.

4.2.1 Notation

Let $U = \{1, \ldots, k, \ldots, N\}$ be the finite population with a response variable $y_k \in \mathbb{R}$ and auxiliary variables $x_k = (x_{k1}, \ldots, x_{kj}, \ldots, x_{kJ}) \in \mathbb{R}^J$ for each member of the population. The (vector-valued) population total $t_x := \sum_{k \in U} x_k$ is known accurately. Of this population, both response and auxiliary variables are known only for a random subsample $s \subset U$ of the population, for which inclusion probabilities $\pi_k = P(k \in s)$ are given. The inclusion probabilities are determined by the survey’s sampling design, and allow a straightforward derivation of design weights $d_k := 1/\pi_k \in \mathbb{R}$ using inverse probability weighting (Horvitz and Thompson, 1952). If the inclusion probabilities are correct and there are no missing values, applying the design weights to the data yields unbiased estimates. In general, the sample is much larger than the number of auxiliary variables.
4.2.2 Problem

The objective is to find new weights \( w_k \in \mathbb{R} \) that satisfy the calibration equation which requires that the weighted sum of the auxiliary vectors equals the population total:

\[
\sum_{k \in s} w_k x_k = t_x. \tag{4.1}
\]

The new weights define a calibration estimator \( \hat{Y}_w = \sum_{k \in s} w_k y_k \) of the population total of the response variable (Kim and Park, 2010), and must be close to the design weights, so that this estimator can be expected to be unbiased (Deville and Särndal, 1992). Similarity of weights is assessed with the help of a distance function \( G(w_k/d_k) \). This function must be nonnegative, strictly convex, and twice continuously differentiable. Furthermore, \( G(1) = G'(1) = 0 \) and \( G''(1) = 1 \) are required. The overall distance measure is given by

\[
\Delta = \sum_{k \in s} d_k G(w_k/d_k). \tag{4.2}
\]

The goal is to minimize \( \Delta \) while fulfilling the calibration equation (4.1). Section 4.2.4 discusses useful distance functions.

4.2.3 Solution

Deville et al. (1993) apply the method of Lagrange multipliers to derive a solution for the constrained optimization problem. The calibration equation (4.1) defines \( J \) constraints. Combining these constraints with \( \Delta \) from the distance definition (4.2) leads to the following function (in \( w_k \)) for which the minimum is sought:

\[
\sum_{k \in s} d_k G(w_k/d_k) - \lambda^\top \left( \sum_{k \in s} w_k x_k - t_x \right).
\]
Here, \( \lambda = (\lambda_1, \ldots, \lambda_J) \) is a Lagrange multiplier. Partial differentiation with respect to each \( w_k \) leads to \( |s| \) equations, one for each \( w_k \), with \( g = G' \):

\[
g(w_k/d_k) - x_k^T \lambda = 0.
\]

The inverse \( F = g^{-1} \) exists because of the strict convexity of \( G \). This leads to the following identity:

\[
w_k = d_k F(x_k^T \lambda).
\]

Plugging this back into the calibration equation (4.1) leads to the following system with \( \lambda \) as the only unknown:

\[
\sum_{k \in s} d_k F(x_k^T \lambda)x_k = t_x.
\] (4.3)

This system is solved by setting

\[
\phi(\lambda) = \sum_{k \in s} d_k \left( F(x_k^T \lambda) - 1 \right) x_k,
\] (4.4)

and finding the root of \( \phi(\lambda) - m_x \), with \( m_x = t_x - \sum_{k \in s} d_k x_k \), using Newton’s method. Iteration proceeds until convergence using

\[
\lambda_{i+1} := \lambda_i + \phi'(\lambda_i)^{-1} (m_x - \phi(\lambda_i)),
\] (4.5)

with

\[
\phi'(\lambda) = \sum_{k \in s} d_k F'(x_k^T \lambda)x_k x_k^T.
\] (4.6)

Equation (4.6) is obtained by applying the double chain rule of differentiation to the definition of \( \phi \) in Eq. (4.4).

The computational costs are dominated by the evaluation and inversion of \( \phi'(\lambda) \), which is a matrix of size \( J \times J \). A sequence of vector and matrix operations can be used to compute \( \phi'(\lambda) \) from its definition (4.6). To avoid issues with singular matrices, which can occur because of limited numeric precision or correlations in the auxiliary variables, the Moore-Penrose pseudoinverse can be used as inversion operator (Ben-Israel, 1965).
However, in this case, convergence to a result satisfying the equation system (4.3) is not guaranteed.

### 4.2.4 Distance functions

Several distance functions $G$ (and their corresponding $F$) are suggested by Deville et al. (1993). The **linear method** uses $G(x) = \frac{1}{2}(x-1)^2$ and $F(u) = 1 + u$. The resulting ratio of final vs. design weights squared is minimal. Because the derivative of $F$ is constant here, the linear method allows solving the equation system (4.3) in just one iteration. The resulting weights can be negative, which makes them difficult to interpret. Feasibility check is an important practical application of the linear method: If it fails to find a solution, the problem can be assumed ill-conditioned for all practical purposes. A proof is beyond the scope of this chapter, however. Conversely, if the linear method yields a solution without or with only few negative weights, there is hope that there exists a solution with only positive weights, which then can be found using one of the methods described below.

The **multiplicative method** defines $G(x) = x \log x - x + 1$ and $F(u) = \exp(u)$, therefore all weights must be strictly positive. Here, the final weights have minimal relative entropy to the design weights. Indeed, this method is equivalent to the **ENT** algorithm presented in the previous chapter, as shown in the next section. A variation is the **logit method** with a similar distance function, which is however restricted to values between given bounds $L$ and $U$ with $L < 1 < U$, and evaluates to infinity for arguments outside this range. Finally, the **restricted linear method** is a similar variation of the linear method that forces all weights between given $L$ and $U$. For both the logit and the restricted linear methods, the corresponding $F(u)$ remain nondecreasing and differentiable in $(L, U)$, however the update step (4.5) must be adapted to ensure that all weights stay in the allowed range.

Apart from the obvious disadvantage of the linear method, there seems to be no clear consensus as to which method to prefer in practice. The multiplicative method seems to be optimal from an information-theoretic perspective, but tends to generate more extreme weights. While individual weights might change considerably depending on the distance function, the estimates based on these weights usually do not vary very much (Deville et al., 1993).
4.3 Equivalence

This section shows the equivalence of the Ent algorithm from the previous chapter to generalized raking with the multiplicative method. The two algorithms are defined for different data structures, use different notations, optimize for slightly different objective functions, and were designed for different applications. The necessary transformations are presented below.

4.3.1 Data structure and notation

For a household survey with categorical variables only, $U$ corresponds to the entire population of the study area at the household level. Each household is represented by a vector of auxiliary variables. For each categorical household and person attribute $x$ or $\xi$, several columns in the vector of auxiliary variables are allocated, one for each category $x \in x$ or $\xi \in \xi$. Columns that correspond to household-level categories are indicators: The value is one if the household is in the corresponding category, and zero if it is not. A column for a person-level category indicates the count of household persons of that category. Accordingly, each column in the vector of population totals denotes to the total number of households or persons in the corresponding category. Table 4.1 shows a map between notations used for multi-level fitting and generalized raking.

This transformation, which is also used in (Bar-Gera et al., 2009), converts a hierarchical structure of categorical data to a matrix of integers which nevertheless contains all the data necessary to solve the problem at hand. To control for person counts, it is sufficient to record the number of corresponding persons in each households. Note that the inverse transformation is not unique in general: If more than one attribute at person level is controlled, the link between the person-level attributes is lost.

4.3.2 Objective function

This subsection shows that the objective functions used for generalized raking with the multiplicative method (4.2) and for the Ent method (3.3 on page 41) are equivalent. Without loss of generality, $s \subseteq \{1, \ldots, n\}$ can be fixed for consistency with the notation in the previous chapter. It is easy to verify that the calibration equation (4.1) is equivalent to the constraints in
Table 4.1: Notational equivalents for multi-level fitting and generalized raking

<table>
<thead>
<tr>
<th>Object</th>
<th>Multi-level fitting</th>
<th>Generalized raking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universe</td>
<td>—</td>
<td>( U )</td>
</tr>
<tr>
<td>Sample of households</td>
<td>( H )</td>
<td>( s )</td>
</tr>
<tr>
<td>Sample of persons</td>
<td>( P )</td>
<td>—</td>
</tr>
<tr>
<td>Household index</td>
<td>( h )</td>
<td>( k )</td>
</tr>
<tr>
<td>Person index</td>
<td>( p )</td>
<td>—</td>
</tr>
<tr>
<td>Household weights</td>
<td>( f_h )</td>
<td>( w_k )</td>
</tr>
<tr>
<td>Person weights</td>
<td>( f_p )</td>
<td>—</td>
</tr>
<tr>
<td>Household of a person</td>
<td>( h_p )</td>
<td>—</td>
</tr>
<tr>
<td>Design weights</td>
<td>( f_h^{(0)} )</td>
<td>( d_k )</td>
</tr>
<tr>
<td>Household attributes</td>
<td>((a_h, b_h, \ldots))</td>
<td>( x_k = (x_{k1}, \ldots, x_{kJ}) )</td>
</tr>
<tr>
<td>Person attributes</td>
<td>((\alpha_p, \beta_p, \ldots))</td>
<td>( t_x )</td>
</tr>
<tr>
<td>Household controls</td>
<td>( C_x1, C_x2, \ldots )</td>
<td>( t_x )</td>
</tr>
<tr>
<td>Person controls</td>
<td>( C_{\xi1}, C_{\xi2}, \ldots )</td>
<td>( t_x )</td>
</tr>
<tr>
<td>One household attribute</td>
<td>( x_h \in {1 \ldots</td>
<td>x</td>
</tr>
<tr>
<td>One person attribute</td>
<td>( \xi_p \in {1 \ldots</td>
<td>\xi</td>
</tr>
</tbody>
</table>

Eqs. (3.1) and (3.2) (on pages 36 and 38) after applying the transformations in the previous subsection. Both ensure that the weighted frequency of each category (at either person or household level) is fixed, using the respective notation.

The multiplicative method uses \( G(x) = x \log x - x + 1 \) as distance function. Using the distance definition (4.2), the distance becomes

\[
\Delta = \sum_{k \in s} d_k((w_k/d_k) \log(w_k/d_k) - (w_k/d_k) + 1)
\]

\[
= \sum_{k \in s} (w_k \log(w_k/d_k) - w_k + d_k)
\]

\[
= \sum_{k \in s} w_k (\log(w_k/d_k) - 1) + \sum_{k \in s} d_k
\]

\[
= \sum_{k \in s} w_k (\log(w_k/d_k) - 1) + \text{const.}
\]
The identities hold because the design weights $d_k$ are fixed and positive. The constant does not change the location of the minimum of the objective function. This shows the claimed equivalence.

### 4.3.3 Application

The original application of generalized raking is the estimation of population totals for a variable of interest. By using a household identifier as categorical “variable of interest”, the weights computed by generalized raking can be mapped back to the original data structure. The calibration totals are satisfied exactly by the weights, and in expectation by sampling $|U| = N$ items from the distribution defined by these weights. A disaggregate synthetic population can be derived in a subsequent generation stage as in the previous chapter. This justifies the use of survey calibration for generating synthetic populations.

Because the basic data structure used for generalized raking is a simple real-valued matrix, continuous or natural-valued attributes (e.g., number of cars, income, . . .) can also be controlled for. Either the population total or the population mean must be provided as control for such attributes.

As outlined in Section 4.2.4, generalized raking with the linear method allows to determine quickly if a given problem has a solution. Even for relatively large problems with $J \approx 100$ and $|s| \approx 10^6$, the computation of the weights requires only a few seconds on a modern workstation. This can be used to manually adjust the problem until it eventually becomes feasible, e.g., by adapting the control totals, or collapsing or removing control categories.

### 4.4 Comparison

As seen in the previous section, the methods presented by Bar-Gera et al. (2009) (here abbreviated by ENT-COORD) and Lee and Fu (2011) (here: ENT-BFGS) are essentially equivalent to generalized raking with the multiplicative method (here: GR-MULT). A few subtle differences are described below.

All three methods define a constrained optimization problem that minimizes the relative entropy between initial and final weights subject to constraints defined by the control totals. The optimization problems are
almost equivalent, only Ent-BFGS uses a slightly different formulation (compare the constraints (2.5) and (2.6) on page 22), which is equivalent to that of Ent-coord and GR-mult if at least the total number of households is fixed.

The methods for solving this optimization problem are quite different. Ent-coord employs a coordinate search, Ent-BFGS suggests the Broyden-Fletcher-Goldfarb-Shanno algorithm that operates on an approximation of the Hessian matrix, and GR-mult suggests using Newton’s method and also provides a closed-form expression for the Jacobian matrix. All methods search for a suitable \( \lambda \) vector instead of operating on the weights. Deville et al. (1993) mention the coordinate search, and also demonstrate that it is equivalent to the IPF algorithm when the vector of auxiliary variables corresponds to expanded categorical variables.

Ent-coord defines a relaxed formulation where the deviation from the control totals is also optimized for, instead of fixing it to zero. This allows finding a solution that approximately satisfies the control totals if no solution for the strict formulation exists. The other two methods support only exact solutions. However, generalized raking can also be performed with a simpler distance function (the linear method), which allows a very fast assessment if a solution with positive weights is likely to exist.

Finally, perhaps the strongest advantage of the GR-mult method is the very general formulation. The method is defined for very simple data structures—matrices and vectors of real numbers, and the original problem, the weighting of a household survey, is adapted to that data structure. In contrast, most other methods discussed in the previous chapters are tailored to a hierarchical data structure with only categorical variables. For instance, adding support for continuous attributes to the multi-level fitting methods shown in the previous chapter seems to be difficult at least, whereas generalized raking easily allows using continuous attributes for calibration.

This very general formulation of generalized raking also allows for very generic software implementations, such as the survey (Lumley, 2017, 2010), sampling (Tillé and Matei, 2016), and grake (Appendix C) extensions for the R statistical software package (R Core Team, 2017) or the SAS macro CALMAR by Deville et al. (1993). To date, the transformation of the problem as shown in Section 4.3.1 remained the responsibility of the user. The MultiLevelIPF package presented in the next chapter...
4.5 Conclusion

This chapter presents generalized raking, a technique to estimate weights for surveys so that known totals for auxiliary variables are satisfied. A special case of generalized raking is equivalent to the Ent algorithm from the previous chapter. In the generic case, the method becomes only more powerful: The existence of a solution can be checked quickly, and calibration against continuous or natural-valued attributes is supported. Just like with multi-level fitting, the calibration totals are required to be consistent with the reference sample to allow for a solution. In practice, this seems to require manual readjustment of the calibration totals based on the trustworthiness of the corresponding data sources.

Generalized raking is well established in the field of survey statistics, with strong theoretical results and readily available software implementations. Therefore, it is a viable alternative to the multi-level fitting methods presented in the previous chapter. While the latter are specialized algorithms designed for a specific data structure, generalized raking operates on simple data structures—real-valued vectors and matrices—and instead adapts the specific data structure to the problem. (A similar approach will be shown in Chapter 8 for nearest-neighbor computations on mixed-type data.) Further research seems necessary to fully transfer the state of the art in survey calibration to the domain of population synthesis for microsimulations.

The results of this chapter are of purely theoretical nature. The next chapter presents an implementation for the multi-level fitting techniques introduced in the previous chapter, and for generalized raking, as an extension for the R statistical software package (R Core Team, 2017). This implementation is used to conduct various tests using real-world data.
in Chapter 6, in order to assess the practical differences of the different methods.
Chapter 5

Comparing fitting algorithms

The algorithms described in Chapters 3 and 4 are implemented in the open-source MultiLevelIPF extension for the R statistical software package (R Core Team, 2017). This chapter describes the main features of the software, and presents experimental results for synthetic datasets. The next chapter presents simulation results for large real-world data.

5.1 Software

The MultiLevelIPF package (Müller, 2016) is a fully tested, open-source, pure R implementation of the algorithms presented in Chapters 3 and 4. The common generic interface allows a fair assessment of their performance relative to each other using a variety of input data. Automated tests allow maintaining and extending the code without fear of breaking functionality. With source code and test data readily available, reproducibility is ensured. The following algorithms are implemented:

**HIPF** Hierarchical Iterative Proportional Fitting (Section 3.2)

**IPU** Iterative Proportional Updating (Section 3.3.1)

**Ent** Entropy Optimization (Section 3.3.2), using a derivative-free spectral approach for solving nonlinear systems of equations from the BB package (Varadhan and Gilbert, 2009)

**GR** Generalized raking (Chapter 4), with the linear, raking, and logit distance functions (Section 4.2.4), using the grake package (Appendix C).
The fitting routines accept simple tables (data frames) for the reference sample and the control totals. The transformation to a matrix of count values (see Section 4.3.1) is performed automatically, along with several important technical consistency checks (see also Section 3.4.2).

- Are attribute names and categories consistent between reference sample and control totals?
- Are there any missing values in the reference sample or in the control totals?
- Do all target values have at least one corresponding observation in the reference sample?
- Are target values specified for all relevant categories?
- Are any target values zero? (In this case, observations are removed from the reference sample with a warning.)
- Are there any conflicting control totals?

While some care has been taken to provide an efficient implementation by using vectorized operations, precomputation, and sparse matrices, run time was not the major focus. Therefore, instead of comparing run times, the number of iterations necessary to achieve a prespecified accuracy is measured. For HIPF and IPU, an iteration consists of using each target value once to update the reference sample. For the Ent algorithm, the iterations of the underlying nonlinear optimization implementation are indicated. One iteration of GR corresponds to one step of Newton’s method (see Section 4.2.3).

More detailed technical documentation can be found in Appendix B.

5.2 Examples

The MultiLevelIPF package also contains several test examples:

**Tiny** The example shown in (Ye et al., 2009), 23 persons, two controls: car availability (🚗/🚗) at household level, and work status (●/●/●/●/●) at person level.

**Single** The example from Chapter 2 (see Fig. 2.5 on page 22), 406 persons, two controls: car availability (🚗/🚗) at household level, and work status (●/●/●/●/●) at person level.

**Separate** Same as Single, plus gender (♂/♀) at person level. Each observation in the original reference sample is repeated twice, the reference
sample now contains 812 persons. The gender has been assigned at random. The total number of individuals per gender ($\delta/\varphi$) are given as additional controls.

**Joint** Same as Separate, now controlling jointly for gender and work status ($\bullet \times \delta/\varnothing \times \delta/\bullet \times \varphi/\varnothing \times \varphi$).

**Separate-Grouped** Same as Single, plus two groups $X$ and $Y$. Each observation in the reference sample is repeated for each group, the reference sample now contains 812 persons. The total number of individuals per group ($p_X/p_Y$) and households per group ($h_X/h_Y$) are given as additional controls.

**Joint-Grouped** Same as Separate-Grouped, now controlling groupwise for both car availability ($\boxplus \times h_X/\square \times h_X/\boxplus \times h_Y/\square \times h_Y$) and work status ($\bullet \times h_X/\varnothing \times h_X/\bullet \times h_Y/\varnothing \times h_Y$).

**Conflict** A minimal example with conflicting target values (see Section 3.4.2): Two observations with one attribute at each household and person levels, only one category. The target value equals 1 for households, and 2 for persons.

Table 5.1 shows the results for the first four example datasets. All algorithms were able to find a solution for these problems. The IPU solution for the Tiny problem is identical to the solution shown in (Ye et al., 2009), and the HIPF and Ent solutions for the Single problem are identical to those shown in (Müller and Axhausen, 2011a).

Only GR with the linear distance function creates negative weights, and requires only one iteration (see Section 4.2.4). The results of the Ent and GR-raking algorithms are identical (as shown analytically in Section 4.3), however, GR requires considerably fewer iterations.

As expected, GR-linear generates weights with minimal squared error, whereas the weights computed by GR-raking (and Ent) are optimal with respect to the $G$ statistic, which is closely related to relative entropy. The weights computed by GR-logit algorithm are a compromise between information gain and sum of squared error. Both $G$ statistic and sum of squared errors are computed against unity weights.

For these small examples, the GR algorithm outperforms the other algorithms by a considerable margin, both in terms of efficiency and quality of the result. Compared to GR, both the $G$ statistic and the sum of squared errors are somewhat larger for HIPF, and even more so for IPU. (Only in the Joint example, the $G$ statistic is slightly smaller for IPU than for
Table 5.1: Feasible examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Weights</th>
<th>$G$</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny</td>
<td>HIPF</td>
<td>118</td>
<td>[2.8, 26]</td>
<td>43.26</td>
<td>523.2</td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td>873</td>
<td>[2.6, 26]</td>
<td>63.44</td>
<td>738.4</td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td>189</td>
<td>[2.6, 26]</td>
<td>43.11</td>
<td>513.1</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td>6</td>
<td>[2.6, 26]</td>
<td>43.11</td>
<td>513.1</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td>8</td>
<td>[2.5, 26]</td>
<td>43.12</td>
<td>510.7</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td>1</td>
<td>[0.3, 25]</td>
<td>47.91</td>
<td>492.4</td>
</tr>
<tr>
<td>Single</td>
<td>HIPF</td>
<td>27</td>
<td>[0.26, 3.9]</td>
<td>109.2</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td>186</td>
<td>[0.2, 3.5]</td>
<td>129.7</td>
<td>151.4</td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td>84</td>
<td>[0.25, 5.7]</td>
<td>97.42</td>
<td>136.3</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td>6</td>
<td>[0.25, 5.7]</td>
<td>97.42</td>
<td>136.3</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td>11</td>
<td>[0.22, 4.9]</td>
<td>97.93</td>
<td>127.9</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td>1</td>
<td>[–0.3, 3.4]</td>
<td>–</td>
<td>113.3</td>
</tr>
<tr>
<td>Separate</td>
<td>HIPF</td>
<td>28</td>
<td>[0.2, 4.3]</td>
<td>234.6</td>
<td>295.3</td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td>187</td>
<td>[0.13, 4.3]</td>
<td>285.5</td>
<td>338.6</td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td>102</td>
<td>[0.2, 6.5]</td>
<td>206.5</td>
<td>293.1</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td>6</td>
<td>[0.2, 6.5]</td>
<td>206.5</td>
<td>293.1</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td>11</td>
<td>[0.17, 5.4]</td>
<td>207.8</td>
<td>271.4</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td>1</td>
<td>[–0.53, 3.6]</td>
<td>–</td>
<td>239</td>
</tr>
<tr>
<td>Joint</td>
<td>HIPF</td>
<td>28</td>
<td>[0.19, 4.3]</td>
<td>234.2</td>
<td>293.1</td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td>97</td>
<td>[0.24, 5.1]</td>
<td>233.7</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td>107</td>
<td>[0.2, 6.5]</td>
<td>206.5</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td>6</td>
<td>[0.2, 6.5]</td>
<td>206.5</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td>11</td>
<td>[0.17, 5.4]</td>
<td>207.8</td>
<td>271.5</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td>1</td>
<td>[–0.49, 3.6]</td>
<td>–</td>
<td>239.3</td>
</tr>
</tbody>
</table>

HIPF, but still considerably larger than that for GR-raking.) Unsurprisingly, Newton’s algorithm used in GR-raking requires far fewer iterations than the general-purpose optimization algorithm used for Ent. HIPF consistently requires fewer iterations than IPU for these examples.
Table 5.2: Infeasible examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Algorithm</th>
<th>Measure</th>
<th>Iterations</th>
<th>Weights</th>
<th>G</th>
<th>SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep.-Gr.</td>
<td>HIPF</td>
<td></td>
<td>(73)</td>
<td>[0.24, 4.2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td></td>
<td>(71)</td>
<td>[0.2, 3.7]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td></td>
<td>143</td>
<td>[0.25, 6]</td>
<td>202.3</td>
<td>285.7</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td></td>
<td>6</td>
<td>[0.25, 6]</td>
<td>202.3</td>
<td>285.7</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td></td>
<td>11</td>
<td>[0.22, 5.1]</td>
<td>203.7</td>
<td>265</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td></td>
<td>1</td>
<td>[-0.27, 3.5]</td>
<td>–</td>
<td>233.4</td>
</tr>
<tr>
<td>Joint-Gr.</td>
<td>HIPF</td>
<td></td>
<td>(74)</td>
<td>[0.24, 4.1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td></td>
<td>368</td>
<td>[0.094, 3.5]</td>
<td>297.7</td>
<td>313.1</td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td></td>
<td>279</td>
<td>[0.25, 6.4]</td>
<td>205.3</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td></td>
<td>6</td>
<td>[0.25, 6.4]</td>
<td>205.3</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td></td>
<td>11</td>
<td>[0.22, 5.3]</td>
<td>206.6</td>
<td>268.1</td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td></td>
<td>1</td>
<td>[-0.3, 3.6]</td>
<td>–</td>
<td>235.9</td>
</tr>
<tr>
<td>Conflict</td>
<td>HIPF</td>
<td></td>
<td>(1)</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IPU</td>
<td></td>
<td>(3)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ent</td>
<td></td>
<td>(106)</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GR-raking</td>
<td></td>
<td>(501)</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GR-logit</td>
<td></td>
<td>(501)</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GR-linear</td>
<td></td>
<td>(1)</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2 shows the results for the last three datasets. If a solution satisfying all controls could not be obtained, the number of iterations is given in parentheses, and the error is given in braces as relative deviation from the target values. Neither HIPF nor IPU can solve the Grouped examples, although the problems are perfectly feasible, as the results for the other algorithms suggest. The error remains in the range of a few percent in this example.

As expected, the Conflict problem cannot be solved. HIPF and IPU return weights that perfectly satisfy one of the controls, while the other algorithms return in-between weights that minimize the overall error for all target values. In particular, the GR algorithms encounter a singular matrix
for which only the Moore-Penrose pseudoinverse but not a regular inverse can be computed (see Section 4.2.3).

Notably, Ent and the GR algorithms (except GR-linear) exceed their iteration threshold before giving up. However, feasibility can also be checked with the GR-linear method which still requires only one iteration, and the GR and Ent implementations can be fixed easily to detect that the state between two iterations has not changed, as already done for HIPF and IPU.

## 5.3 Conclusion

The demonstrated examples allow a glance on the performance of the various methods presented in the previous chapters. The results suggest that GR outperforms the other methods by a margin. The next chapter applies these methods on reference samples and controls derived from the full Swiss population census. Another real-world application, which also makes use of features unique to GR, is shown in Chapter 7.
Chapter 6

Large-scale application of fitting algorithms

This chapter presents a practical application of the MultiLevelIPF package introduced in the previous chapter. The performance of the different methods described earlier in this thesis is assessed with a simulation study. The Swiss population census 2000 (Swiss Federal Statistical Office (BFS), 2000) is taken as ground truth, and various attempts are made to reconstruct a synthetic population with both reference sample and control totals derived from this ground truth. The following research questions are addressed:

- What sample sizes are needed to avoid the zero cell problem (cf. Sections 2.2.4 and 3.4.2)?
- Does the inclusion of a spatial component, or joint control for attributes, improve the goodness of fit?
- How do the three methods IPU, HIPF, and GR (with the multiplicative method, cf. Section 4.2.4) compare with respect to convergence, goodness of fit, and computation time?

This chapter focuses solely on the fitting methods: the goodness of fit is evaluated on the estimated weights, and not on a discrete population. No location (beyond the location attribute available in the data) is assigned to the agents. To allow for comparing the different multi-level fitting methods, fitting continuous variables is not discussed in this chapter; the next chapter describes a case study that makes use of this unique feature of GR.

After a description of the data, the sample sizes necessary to avoid the zero-cell problem for various combinations of controls are determined. The error that results from sampling alone is assessed, and is compared to the error introduced by the different methods.
Table 6.1: Description of individual-level attributes in the Swiss census data

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Kind</th>
<th>Description</th>
<th>Categories/Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSON_ID</td>
<td>ID</td>
<td>ID of the person</td>
<td>5–7597703</td>
</tr>
<tr>
<td>GESL</td>
<td>control</td>
<td>gender</td>
<td>2</td>
</tr>
<tr>
<td>HMAT</td>
<td>control</td>
<td>nationality</td>
<td>2</td>
</tr>
<tr>
<td>ZIVL</td>
<td>control</td>
<td>marital status</td>
<td>4</td>
</tr>
<tr>
<td>VALTJ</td>
<td>control</td>
<td>age</td>
<td>0–107</td>
</tr>
<tr>
<td>VALTJ5</td>
<td>control</td>
<td>age</td>
<td>17</td>
</tr>
<tr>
<td>STHHZ</td>
<td>auxiliary</td>
<td>position in household</td>
<td>3</td>
</tr>
<tr>
<td>HABG</td>
<td>auxiliary</td>
<td>education level</td>
<td>13</td>
</tr>
<tr>
<td>KAMS</td>
<td>auxiliary</td>
<td>work status</td>
<td>4</td>
</tr>
<tr>
<td>BGRAD</td>
<td>auxiliary</td>
<td>work percentage</td>
<td>10</td>
</tr>
<tr>
<td>SOPK</td>
<td>auxiliary</td>
<td>socio-professional category</td>
<td>12</td>
</tr>
<tr>
<td>ANOGA</td>
<td>auxiliary</td>
<td>NOGA code of the enterprise</td>
<td>11</td>
</tr>
</tbody>
</table>

6.1 Description of data

The qualities of methods for population synthesis are best assessed by comparing the results with the ground truth. For Switzerland, the individual data for the year 2000 population census come closest to this notion. (In 2010, the full population census has been replaced with a combination of a register and a structural survey (Swiss Federal Statistical Office (BFS), 2010a,b). These data are used in the case study described in Chapter 7.)

The individual data for the Swiss population census comprises of 7,452,075 observations at individual level. Each observation corresponds to an official residence location of an inhabitant of Switzerland, with household data attached. Thus, for some persons, the census contains more than one observation. For this analysis, only persons in private households are used, collective households and group quarters are excluded. Observations that correspond to a secondary or tertiary residence were also excluded. In total, 7,044,339 observations remain (94.5% of the original data).

Tables 6.1 and 6.2 describe the attributes available in the data at individual or group level. The following attribute kinds are distinguished:

**ID** Each person and household has a unique ID, persons from the same household have the same household ID,
Table 6.2: Description of group-level attributes in the Swiss census data

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Kind</th>
<th>Description</th>
<th>Categories/Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHNR</td>
<td>ID</td>
<td>ID of the household</td>
<td>5–4996799</td>
</tr>
<tr>
<td>HHGR</td>
<td>control</td>
<td>household size</td>
<td>1–17</td>
</tr>
<tr>
<td>HHGR1</td>
<td>control</td>
<td>household size</td>
<td>6</td>
</tr>
<tr>
<td>HHTPZ</td>
<td>auxiliary</td>
<td>household type</td>
<td>7</td>
</tr>
<tr>
<td>ZGDE</td>
<td>zone</td>
<td>municipality</td>
<td>2896</td>
</tr>
<tr>
<td>district</td>
<td>zone</td>
<td>district</td>
<td>184</td>
</tr>
<tr>
<td>canton</td>
<td>zone</td>
<td>canton</td>
<td>26</td>
</tr>
<tr>
<td>large_reg</td>
<td>zone</td>
<td>large region</td>
<td>7</td>
</tr>
<tr>
<td>ms_reg</td>
<td>zone</td>
<td>MS region</td>
<td>106</td>
</tr>
<tr>
<td>lang_reg</td>
<td>zone</td>
<td>language region</td>
<td>4</td>
</tr>
<tr>
<td>urban_rural</td>
<td>zone</td>
<td>urban/rural</td>
<td>4</td>
</tr>
<tr>
<td>urban_rural_3</td>
<td>zone</td>
<td>urban/rural</td>
<td>3</td>
</tr>
<tr>
<td>mun_type_9</td>
<td>zone</td>
<td>municipality type</td>
<td>9</td>
</tr>
<tr>
<td>mun_type_22</td>
<td>zone</td>
<td>municipality type</td>
<td>22</td>
</tr>
</tbody>
</table>

control Sociodemographic attributes such as age, gender, nationality, marital status, and household size, for which controls at zone level (municipality, district, ...) are usually available from statistical offices,

auxiliary Sociodemographic attributes such as position in the household, education level, and occupation specifics, for which aggregate controls are not usually available,

zone Location information at various levels of aggregation.

Attributes at group level are identical for all members of the household, whereas attributes at individual level may vary between persons of the same household. By definition, all zone attributes are at group level. The rightmost column contains the number of categories for categorical, and the range for continuous and ID variables.

All attribute names are inherited from the original dataset, except the zoning attributes in lowercase letters, which are derived from the location at municipality level (ZGDE). Some attributes with a number in the name are binned versions of continuous variables, or lumped categorical variables:

- Age, VALTJ5: Bins of 5 years, a single category for persons aged 80 years and older,
Chapter 6. Large-scale application of fitting algorithms

Figure 6.1: Histograms of category frequencies for control attributes

- Household size, \textit{HHGR1}: Bins for each household size up to 5, one category for households with 6 persons or more,
- Urban/rural, \textit{urban\_rural\_3}: Same as \textit{urban\_rural}, with “isolated city” mapped to “city”.

Figures 6.1 and 6.2: show log-scaled histograms for control and auxiliary attributes, respectively. The categories for auxiliary attributes are unlabeled but arranged by their frequency, because only the frequency of occurrence of each category is relevant for this analysis.
The dataset contains location data at the hectare level, but for this analysis the municipality (ZGDE) is used as the finest aggregation level. The other location attributes are derived using the regional and functional municipality classification provided by the Swiss Federal Statistical Office.

Municipalities, districts, cantons, and large regions define a hierarchy. Districts consist of one or more municipalities. Each canton comprises one or more districts and is part of exactly one large region. MS regions (ms_reg) define a spatial subdivision based more on functional connectivity.
than on administrative boundaries. For each of the four official languages in Switzerland, the corresponding language region (lang_reg) is the area where that language is more prominent than the other three. While each municipality is mapped to exactly one MS region or language region, both MS regions and language regions may contain municipalities from one or more districts or cantons. In addition to these regional subdivisions, municipalities can be classified in four, nine, or even 22 different functional classes (urban_rural, mun_type_9, or mun_type_22).

These regional and functional classifications vary greatly in size, as shown in Figure 6.3: up to four orders of magnitude for municipalities, and still as much as two orders as magnitude for most other classifications. Only large regions and the urban_rural_3 classifications are relatively homogeneous with respect to population size.
6.2 Zero-cell problem

In order to determine the impact of the zero-cell problem discussed in Sections 2.2.4 and 3.4.2, 100 rearrangements of the households in the census data were computed. A rearrangement corresponds to the order in which households are selected, assuming uniform selection probability. For each sampling fraction, each rearrangement yields a reference sample that is as likely as any other reference sample for that particular sampling fraction. This allows to compute, for each rearrangement, the minimum sampling fraction needed to obtain a reference sample where each nonzero target value has at least one corresponding representative. Repeating this for all rearrangements gives a distribution of minimum sampling fractions from which a conservative estimate for the expected minimum sampling fraction can be derived.

While absence of the zero-cell problem is a prerequisite for a perfect fitting result, too few representatives for a control may lead to large errors with uncontrolled variables. For better results, the sampling fraction should be considerably larger than the minimum.

Table 6.3 shows the minimum sampling fractions for single- and higher-dimensional controls. Unsurprisingly, controlling for attributes with infrequent categories requires larger reference samples to avoid the zero-cell problem. Most controls would not suffer from the zero-cell problem even for

---

Table 6.3: Sampling fraction needed to avoid the zero-cell problem

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Full Coverage at</th>
</tr>
</thead>
<tbody>
<tr>
<td>GESL</td>
<td>0.0000% (0.00003%–0.000%)</td>
</tr>
<tr>
<td>HMAT</td>
<td>0.0002% (0.00003%–0.001%)</td>
</tr>
<tr>
<td>GESL:HMAT</td>
<td>0.0003% (0.00006%–0.001%)</td>
</tr>
<tr>
<td>ZIVL</td>
<td>0.0004% (0.00006%–0.002%)</td>
</tr>
<tr>
<td>VALTJ5</td>
<td>0.0011% (0.00055%–0.003%)</td>
</tr>
<tr>
<td>GESL:ZIVL</td>
<td>0.0017% (0.00032%–0.016%)</td>
</tr>
<tr>
<td>HHGR1</td>
<td>0.0020% (0.00023%–0.010%)</td>
</tr>
<tr>
<td>GESL:VALTJ5</td>
<td>0.0024% (0.00101%–0.010%)</td>
</tr>
<tr>
<td>GESL:HMAT:HHGR1</td>
<td>0.0079% (0.00159%–0.023%)</td>
</tr>
<tr>
<td>GESL:HMAT:ZIVL</td>
<td>0.0159% (0.00178%–0.059%)</td>
</tr>
<tr>
<td>GESL:HMAT:VALTJ5</td>
<td>0.0387% (0.00901%–0.135%)</td>
</tr>
</tbody>
</table>
Figure 6.4: Sampling fraction needed to avoid the zero-cell problem when a spatial component is included

ad-hoc samples of a few hundred observations, provided uniform selection probability. Only three-way or finer cross-classifications seem to warrant larger samples.

Figure 6.4 shows the same results if a spatial attribute is controlled simultaneously. Of all the zoning systems available, only the large regions and the language regions, perhaps combined with the derived three-level municipality classification, allow for reasonably sized sampling fractions. The finer and the more unevenly sized the zoning system, the fewer (if any) attributes can be controlled simultaneously.

### 6.3 Sampling error

The rearrangements described in the previous section were used to derive samples with different sampling fractions between 0.25\% and 8\%. Figure 6.5 shows the error introduced by the sampling in a selection of the auxiliary
Figure 6.5: Sampling error with different sampling fractions for the auxiliary variables. The observed total in each category is compared with the total in the sample rescaled uniformly by the inverse of the sampling fraction. This comparison occurs across all attributes, and for all samples. The categories are arranged in the same order as in Figure 6.2. The sampling error for each category is well below 0.5% of the overall population, even for the smallest samples tested. This means that, in each category, less than 0.5% of the overall population are misplaced or missing. The magnitude of the sampling error tends to be larger for categories that occur more frequently in the population.
The auxiliary attributes are used to compute the standardized root mean squared error (SRMSE). Compared to Knudsen and Fotheringham (1986), this analysis uses a variant where the error is computed across all attributes, and not for each cell in a matrix:

\[
SRMSE = \sqrt{\frac{I}{\frac{1}{I} \sum_i N_i} \left( \hat{N}_i - N_i \right)^2}.
\]

Here, \( I \) is the number of categories (across all attributes), and \( \hat{N}_i \) and \( N_i \) are simulated and observed totals, respectively, for category \( i \in \{1, \ldots, k\} \). This metric captures the differences in the frequency of each category of all auxiliary attributes. The results are shown in Figure 6.6: The SRMSE tends to decrease for larger sampling fractions, but a larger sample may still result in a larger SRMSE than a “good” smaller sample.

### 6.4 Comparing methods

Each of the samples analyzed in the previous section was provided as reference sample for the HIPF, IPU, and GR algorithms (see Section 5.1). The Ent algorithm was excluded, because the underlying derivative-free optimization method was unable to achieve convergence in a reasonable
number of iterations. The reasons for convergence failure were not explored further, because GR with the multiplicative method offers a more robust and otherwise identical alternative, as shown in Section 4.3.

The following different controls were applied, with target values derived from the full population:

1. All categorical control variables,
2. Age and gender jointly, and other categorical control variables,
3. All categorical control variables, and large region,
4. All categorical control variables, and large region by three-level municipality type,
5. All categorical control variables, each by large region.

The target values for all controls were derived from the full census sample. Each algorithm was run on each combination of reference sample, and controls, with tolerance $10^{-4}$ but default settings otherwise. (The tolerance specifies the maximum allowed relative error for all categories across all attributes before convergence is assumed.) A total of 9,000 sets of weights has been computed. For successful runs, the SRMSE over all auxiliary attributes also has been computed.

Table 6.4 shows some statistics for the experiments for each algorithm. A successful run is one that completes without error and where convergence has been achieved (according to the prespecified tolerance). For this experiment setup, GR almost always succeeds, while HIPF and IPU fail more frequently. The “Iterations” column contains the mean number of iterations for successful experiments only. Notably, GR seems to require only a handful iterations, whereas HIPF and IPU seem to require considerably more work to achieve convergence in these scenarios. In the “Residuals” column, the mean absolute difference for the category that deviates most (across all attributes) is given, again for successful experiments only. All algorithms achieve a nearly perfect fit. GR satisfies

---

**Table 6.4: Experiment results (total: 9,000)**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Successes</th>
<th>Failures</th>
<th>Iterations</th>
<th>Residuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIPF</td>
<td>1528</td>
<td>1472</td>
<td>49.70</td>
<td>385.46</td>
</tr>
<tr>
<td>IPU</td>
<td>1454</td>
<td>1546</td>
<td>442.90</td>
<td>151.46</td>
</tr>
<tr>
<td>GR</td>
<td>2981</td>
<td>19</td>
<td>3.20</td>
<td>12.94</td>
</tr>
</tbody>
</table>

---

75
the control totals more accurately, this is in part due to a slightly different interpretation of the tolerance for the GR algorithm. However, HIPF and IPU would need even more iterations to improve the fit, if at all possible. In terms of performance (both success rate and number of iterations) and goodness of fit, GR clearly outperforms the two other algorithms.

Figure 6.7 shows the resulting SRMSE values, aggregated over all controls. Both HIPF and GR achieve a reduction of the SRMSE compared to rescaling as described in the previous section, only IPU tends to considerably increase the SRMSE in this experiment. This means that the fitted reference sample is more representative of the true population with respect to both control and auxiliary variables than the rescaled reference sample alone, for all algorithms except IPU.

Finally, Figure 6.8 shows the resulting SRMSE values for individual controls for the GR and HIPF algorithms. The SRMSE values for GR are comparable to those for HIPF, if convergence has been achieved.
Furthermore, the SRMSE values within each algorithm seem to be very close to each other. This suggests that the choice of control variables does not have an immediate effect on the goodness-of-fit for auxiliary variables. In particular, controlling for a spatial component does not seem to adversely affect the goodness-of-fit if enough observations are available in each region for each controlled category, while at the same time satisfying per-region target values.
6.5 Conclusion

In this chapter, three multi-level fitting algorithms were applied to reconstruct a known ground truth from a reference sample and a corresponding set of control totals, for different sampling fractions and samples. Using the location information from the reference sample is elusive for all but the coarsest classifications due to the notorious zero cell problem. Including location information allows controlling for location totals but does not seem to improve goodness-of-fit for auxiliary variables. The results suggest that GR is able to obtain a better fit faster, and with a larger success rate, than the other algorithms. The next chapter shows an application of GR which also uses continuous control attributes.
Chapter 7

Adapting activity chains to future scenarios

This chapter presents the processes employed to generate a $10^6$ member agent-population for a long-range study of 2030 travel in Switzerland, using methods well established in statistics—survey calibration (see Chapter 4) and statistical matching (described in this chapter). Survey calibration allows exogenous specification of shares of activity types while maintaining representativeness of the population, and statistical matching allows joining data sets with common variables. Consistency with known approaches in transportation planning is shown.

The discussion of the results for Switzerland focuses on the quality metrics available and highlights the links between the activity schedules and the total shares of the activity types. Furthermore, the error introduced by the calibration and matching stages is analyzed and quantified, with special emphasis put on uncontrolled sociodemographic and travel variables. The specified activity shares can be replicated almost perfectly; the resulting mean error in the uncontrolled variables is within the range of a few percent. Hence, this approach is a viable alternative to a complicated estimation of an activity schedule model that would be necessary otherwise. Finally, two practical issues—data sets describing different populations, and biased surveys—and their effect on the outcome are tested by suitably adjusting the input data.

---

1 This chapter is published as a journal paper (Müller and Axhausen, 2014).
7.1 Introduction

The agent-based models in the tradition of Bowman et al. (2006) employ multi-level nested logit models to predict the likelihood of different types of daily activity schedules (Bowman, 2009a,b; Axhausen, 2006; Vovsha et al., 2008; Shiftan et al., 2007; Pendyala and Bhat, 2006; Bhat and Koppelman, 2003; Bhat and Pendyala, 2012). To keep the logit model estimable, the number of allowed patterns is generally restricted to a reasonable subset of all observed daily schedules. Decision tree-based models, such as ALBATROSS (Arentze and Timmermans, 2000), equally restrict the structures available. Still, these models capture some of the trade-offs made by travelers and can therefore assess the impact of certain transport policies that affect the generalized costs of travel, as captured by the log-sum terms of the different nests feeding into the main scheduling choice models at the top of the overall model structure. The estimated model parameters also allow, to a certain extent, an understanding of the mechanics that drive the choice of activities and their order.

This ability comes at the cost of careful and time-consuming model estimation, which is not always available to a project. In addition, the models are by definition constrained to transport investment- and policy-related processes. In some cases, the analyst is asked to assess the impact of exogenous targets on the activity schedules or policies or both that a transport model cannot capture, for example, the arrival of reliable and affordable Internet-based daily and nondaily shopping, or work-from-home initiatives.

Balmer (2007) describes the extraction of observed activity schedules from the Swiss transportation microcensus. The extracted activity schedules are stochastically attached to synthetic agents to create the initial demand for a MATSim model (Meister et al., 2009). The distribution of activity schedules can be influenced only by using some of the extracted activity schedules more often than others. Hettinger (2007) proposes a custom weighting procedure, the results of which could be used to affect the selection of activity schedules during distribution.

A novel approach is presented here to estimate weights for a sample of persons with observed activity schedules to reproduce given targets for the future shares or total durations of different activity types, while the sociodemographic distribution is held fixed. In contrast to (Balmer,
the activity schedule remains connected to the sociodemographic data. While the approach cannot give insights into the individual trade-offs made by travelers, it produces a distribution consistent with the target values at low computational costs and without the model estimation effort. Special emphasis is placed on using methods that are well established in the statistics community but perhaps not so widely recognized in the transportation community:

Survey calibration to reweight data sets to satisfy exogenous constraints (Deville and Särndal, 1992; Deville et al., 1993), see also Chapter 4, and

Statistical matching to combine multiple data sets (d’Orazio et al., 2006).

These methods are presented in detail, along with results for an application for all of Switzerland in which a synthetic population of about 808,000 agents (10% of the population) with corresponding activity schedules is created. The case study is part of an effort to assess the effects of electric vehicles on energy production and the stability of the electric supply network.

The remainder of this chapter is organized as follows. After the data are outlined, the methodology is presented in detail. Subsequently, simulation results are shown and discussed; the chapter then concludes with a summary.

7.2 Data

Two major data sets and a classification of communes have been used for generating the synthetic population. This section presents the data sources and shows which parts of the final population are derived from which data source.

7.2.1 Register survey

As of 2010, the full population census of Switzerland has been replaced by the combination of a full register survey and a detailed 2.5% population survey. The data are collected every year, as opposed to the census that was collected every 10 years. The register survey describes the full population of Switzerland on a certain day of 2010. In the present study, detailed data for all persons have been used (8.08 million observations), however,
Chapter 7. Adapting activity chains to future scenarios

An aggregated version that lists only person counts per hectare is freely available and could be used as well.

The data set contains the *de jure* spatial location at the hectare level, in addition to basic sociodemographics available from the civil registry. A 10% random sample without replacement has been drawn, as this was the target sampling fraction for the transport model. Only the age, sex, and location attributes were used. The age was binned in three classes: younger than 25 years, 25 to 64 years, and 65 years or older.

The population survey has not been used for this study, as the transportation microcensus (see below) already contains all necessary information.

7.2.2 Transportation microcensus

A nationwide representative survey on mobility behavior, the transportation microcensus, is collected every 5 years in Switzerland, the last time in 2010. It contains, among other information, extended sociodemographics and information on mobility behavior (activities and detailed trips) for 1 day for about 62,900 persons (0.78%). Only persons 6 years old and above are included in the sample.

The survey days are distributed uniformly over the year for the entire sample. To generate a population that is representative of a “typical working day”, only roughly 24,300 (0.3%) midweek observations (Tuesday till Thursday) were used. Weights are provided to make the sample representative of the full population and of all weekdays. The data set also contains location information at the hectare level, but that information cannot be used directly because of the relative sparsity of the sample.

The microcensus contains a very detailed description of the activities. Each activity type has been mapped to one of Home, Work, Education, Leisure, or Shopping. (For the present study, daily and long-term shopping is not distinguished.) The activity schedule consisting of the five activity types above, as well as age, sex, work status, and location, have been used from this data set. In addition, activity durations have been used for validation (see Section 7.4).
7.2.3 Commune classification

The communes of Switzerland are classified into 22 types according to commuter movement, occupation, housing conditions, wealth, tourism, population, and role in the Central Place Theory as defined by Christaller (1933). A coarser version of this classification with nine levels is provided. This nine-level classification has been applied to the transportation microcensus and the register survey data to keep the reweighted sample representative of the population (cf. Section 7.3.1) and to determine matching classes (see Section 7.3.2). The commune classification serves as a proxy for the accessibility of the agents’ home locations.

7.2.4 Target population

The transport model requires the following attributes for each agent:

- Age,
- Sex,
- Precise home location,
- Precise workplace location,
- Education level,
- Income, and
- Activity schedule with durations for each activity

The first three attributes are provided by the register survey. For imputing the workplace location, a calibrated commuter matrix and detailed data on businesses are matched (Bösch et al., 2016). Education level and activity schedule are imputed to the register data by statistically matching them with the data taken from the transportation microcensus. The implementation of the population synthesis procedure is detailed in the following section.

7.3 Methodology

This section shows the process of generating a synthetic population as described above as applications of survey calibration and statistical matching. The process consists of two stages. In the first stage, survey calibration is used to reweight a person sample with activity schedules to reflect postulated changes in the frequency of certain activity types. Survey calibration has been treated extensively in Chapter 4; therefore only the
application of survey calibration is described in the next subsection. The second stage combines this calibrated data set with the register survey data by means of statistical matching, the method and its application is described in Section 7.3.2.

7.3.1 Survey calibration

Survey calibration can be used in a straightforward fashion for reweighting a sample of persons with activity schedules as described in Section 7.2.2. Activity schedules are a nested structure—different activities grouped together. Therefore, for each observation, the auxiliary variables $x_k$ contain four columns, one for each activity type (except Home), and each element contains the number of activities of the respective type in the given activity schedule. (Controlling Home activities would unnecessarily restrict the optimization problem and lead to more extreme weights; this issue is discussed further in Section 7.4.1.) The $t_x$ vector contains the new total number of trips for each type. Four scenarios are analyzed here, reflecting the following assumptions based on totals derived from the microcensus data (percentages correspond to those totals derived from the microcensus data):

**Baseline** Original frequency of activities, nothing changes.

**Home office** Working from home is encouraged, the number of work trips decreases by 20%.

**Delivery** Home delivery of goods plays a larger role, people replace 20% of their shopping trips by leisure trips.

**Combined** The combination of the former two scenarios.

In addition, the sample is stratified by age, sex, and commune classification, and the population in each stratum is kept unchanged to ensure that the reweighted sample remains representative of the person population. The three age, two sex, and nine commune classes yield 54 additional elements in $x_k$. This is essentially a post-stratification applied on top of the actual update of activity schedule frequencies—weights are estimated to simultaneously satisfy both constraints. The weights supplied with the transportation microcensus are used naturally as design weights, which are not changed by the Baseline scenario.

Apart from selecting only weekday observations, no filtering has been applied to the microcensus data. Persons with exotic chains are present in
the synthetic population at their expected frequency, but these chains are not mutated or otherwise edited in the process described here.

In the following, a small numerical example is shown. The matrix

\[
\begin{pmatrix}
  x_{k1} \\
  x_{k2} \\
  x_{k3} \\
  x_{k4}
\end{pmatrix} =
\begin{pmatrix}
  0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
  1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

(7.1)

corresponds to the following hypothetical set of persons with activity schedules:

- A young male, HEHLH
- A middle-aged female, HSLH
- A middle-aged male, HWLH
- An elderly female, HLHLH

The first four columns of the matrix represent the frequencies of the Work, Education, Leisure, and Shopping activities, and the remaining six columns denote the sociodemographic attributes of the person. (For simplification, the commune classification is not considered here.) For the Delivery scenario, \( t = (1, 1, 6, 0.8, 1, 0, 1, 1, 0, 1) \) would be used as the calibration totals. (For this small example, no solution exists; this is not a problem when the microcensus data are used, even when constraining on commune classification.)

All scenarios potentially change the total number of activities in the population compared with the Baseline scenario. In particular, the Home office scenario leads to an unrealistic decrease in the total number of activities, contrary to evidence by, e.g., Pendyala et al. (1991). This can be alleviated by choosing appropriate relative changes for other activity types; however, for the present case study it is assumed that the suppressed work trips are not substituted.

### 7.3.2 Statistical matching

The term **statistical matching** (or **data fusion**) refers to a stochastic procedure for integration of nonoverlapping data sets \((X, Y)\) and \((X, Z)\) with a common variable \(X\). Here, \(X, Y,\) and \(Z\) can be multivariate. For this procedure, two approaches are distinguished:
Macro  The joint distribution \((X, Y, Z)\) or its key characteristics is estimated directly from the input data sets.

Micro  A complete synthetic data set \((X, Y, Z)\) is constructed. Especially the micro approach is of particular interest for generating a synthetic population. In the simplest case, conditional independence is assumed between \(Y\) and \(Z\) given \(X\). The matching can be performed by choosing one data set as recipient and the other as donor and then drawing, for each recipient record, a compatible record from the donor data set. For categorical \(X\), two records \((x_1, y)\) and \((x_2, z)\) can be treated as compatible if \(x_1 = x_2\) or if they are “sufficiently close” with respect to some distance measure. Weights in the donor data set can be used during the drawing of a compatible record. This particular kind of matching is also referred to as hot deck imputation, and it seems to be a natural and plausible approach to the problem. A practice-ready R package (d’Orazio, 2012) offers an implementation of this algorithm.

Many more procedures and tools are available in the statistical matching framework, such as parametric methods, Bayesian approaches, replacement of the assumption of conditional independence by the assumption of pairwise independence or by auxiliary information, and evaluation of matching uncertainty by using multiple imputation and expectation maximization. These procedures and tools can be seen as extensions of the hot deck approach. See (d’Orazio et al., 2006) for a detailed treatment of the subject.

For the case study presented here, a 10% sample of the register survey is combined with the calibrated transportation microcensus by using statistical matching with hot deck imputation. The first data set is a complete data set with detailed location information, and the second data set contains important attributes such as extended sociodemographics and activity schedules. As the interest here is in generating a complete population, the register survey is used as the recipient data set. The calibration weights derived in Section 7.3.1 are used as matching weights. Common variables in both data sets are age, sex, and commune classification; records with identical values for all variables are considered compatible. As the transportation microcensus contains only persons that are 6 years old or more, small children aged 5 or less were excluded from the register survey before matching. (The effect of ignoring this is analyzed in Section 7.4.2.3.)

D’Orazio et al. (2006) recommend using the larger data set as donor and the smaller data set as recipient. Violating this recommendation
obviously leads to donor records used more than once and, therefore, to a modification of the variability of the imputed variables (in this case, extended sociodemographics and activity schedules). Potential sampling errors and spurious correlations in the transportation microcensus will be amplified, which renders the generated data a poor choice for statistical inference. However, it is usable as input to a transport microsimulation, because both data sets can be considered reliable, and the generated activity chains will be modified anyway when searching for the user equilibrium.

7.4 Results

This section presents experimental results from generating 100 synthetic populations with different random seeds, with a fixed 10% sample of the register survey. Section 7.4.1 presents the results after the calibration of the transportation microcensus to the four scenarios defined in Section 7.3.1. Because this method is entirely deterministic, only one run per scenario is considered here. Matching the calibrated microcensus to the register survey sample yields the final synthetic population; this result is discussed in Section 7.4.2.

The experiments were conducted on the R platform for statistical computing, using the packages survey (Lumley, 2012) and StatMatch (d’Orazio, 2012), among others. The calibration took only a few seconds per run; creating the 10% population requires just less than 15 minutes on a current compute server, including data input and output.

7.4.1 Deterministic results

First, the weights that result from calibrating the transportation microcensus are considered. After that, average activity counts and durations and activity chain frequencies are analyzed.

7.4.1.1 Weights

The original data are also weighted; these weights are used for the Baseline scenario. The larger the relative difference of a weight from the average weight, the more (or less) emphasis is put on the corresponding observation. Extreme weights might be necessary to ensure representativeness (or, in this
Figure 7.1: Distribution of weights after survey calibration

All graphs show the distribution of weights against the rank by decreasing weight. The top graph shows absolute values on a logarithmic scale (rotated CDF). All curves are very similar, most weights are between 0.3 and 3, the median weight is below 1 for all scenarios. The middle graph shows the cumulative sum, the deviation from the straight line is an indicator of non-uniformness (Fourie and Müller; 2011). In turn, the graph at the bottom is a transvected version of the middle plot: Uniform weights now correspond to a horizontal line.
Figure 7.2: Individual weights after survey calibration

This plot shows individual weights on a logarithmic scale for a sample of 50 observations relative to the weight of the same observation in the Baseline scenario.

case, satisfaction of external controls); however, observations with extreme weights substantially contribute to the variance of population estimators (Deville and Särndal, 1992).

Figure 7.1 shows three graphs of the distribution of the weights to visualize the deviation from uniform weights. As expected, the Combined scenario deviates most, followed by the Delivery and the Home office scenarios. However, it appears that the bias in the Baseline scenario, which uses the original microcensus weights, is much larger than the additional bias introduced by each of the scenarios. While the weights do not seem to differ much on average between the scenarios, they do differ significantly for individual observations, as Fig. 7.2 suggests. For individual observations, when comparing the scenarios to the Baseline scenario, it seems that the deviation of the Combined scenario equals the added deviation of the Home office and Delivery scenarios (on the logarithmic scale); this equality is true only in approximation.
7.4.1.2 Activity count and duration

Table 7.1 shows the relative difference compared with the Baseline scenario of mean activity count and duration for the different activity types and scenarios. In the upper part of the table the change in activity counts from the Baseline scenario is shown: The frequency is exactly as specified by the configuration of the scenario, except for Home activities, which have not been controlled for and for which the frequency is between $-1.6\%$ and $1.9\%$. The deviation of the average activity duration per activity type, shown in the lower part of the table, is in an acceptable range between $-2.4\%$ and $8.1\%$. Notably, among all scenarios, the activity duration is altered most for the Home office scenario (third column from the right of Table 7.1): Activities of all types, especially Work activities, tend to take longer on average. An interpretation is attempted in the following subsection.

7.4.1.3 Activity schedules

The distribution of the 20 most frequent types of activity schedules in the different scenarios, sorted according to their frequency in the Baseline scenario, is shown in Table 7.2. The table shows, for each activity schedule type, the relative frequency, the deviation from the Baseline scenario and the rank compared with the Baseline. Perhaps the biggest surprise is the first row, activity schedule type HWH. For the Home office scenario, the frequency of this activity schedule type increases, while it decreases for the other scenarios. A possible reason for this counterintuitive result might be a correlation between this activity schedule type and the person classification (age, sex, and type of residential commune, cf. Section 7.3.1). This result is even worse when the analyst is also controlling for the average frequency of Home activities during calibration. In the Home office scenario the share of the HLH and HSH schedules increases to compensate for the unexpected increase of the share of HWH schedules, at the expense of schedules with two or more work activities such as HWHWH and HWLWH—the latter drop by six positions each in the overall frequency ranking. That result might explain the increase in the average duration of Work activities: The share of HWH schedules (with long work activities) increases slightly while the share of schedules with two (shorter) work activities decreases.

For the Combined scenario, the absolute increase of share is largest for
Table 7.1: Frequency and duration by activity type, after calibration

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Value</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>Frequency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>abs</td>
<td>2.44</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Work</td>
<td>abs</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Education</td>
<td>abs</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Leisure</td>
<td>abs</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Shopping</td>
<td>abs</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Duration [h]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>abs</td>
<td>2.08</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Work</td>
<td>abs</td>
<td>5.79</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Education</td>
<td>abs</td>
<td>3.91</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Leisure</td>
<td>abs</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
<tr>
<td>Shopping</td>
<td>abs</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>rel [%]</td>
<td>0.00</td>
</tr>
</tbody>
</table>

This table shows a cross tabulation of activity type frequency and duration for each scenario, after survey calibration. For each scenario, and there for each activity type, the absolute and relative differences to the corresponding Baseline values are given.

The HLH and HLHLH schedules. The relative differences for the Home office and Delivery scenarios sum up very closely to that of the Combined scenario.
Table 7.2: Most frequent activity schedule types after calibration

<table>
<thead>
<tr>
<th>Schedule</th>
<th>B</th>
<th>H</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f$</td>
<td>$rk$</td>
<td>$f$</td>
<td>$rk$</td>
</tr>
<tr>
<td>HWH</td>
<td>10.8</td>
<td>1</td>
<td>10.9</td>
<td>2</td>
</tr>
<tr>
<td>HLH</td>
<td>9.9</td>
<td>2</td>
<td>11.6</td>
<td>1</td>
</tr>
<tr>
<td>HSH</td>
<td>6.7</td>
<td>3</td>
<td>7.4</td>
<td>3</td>
</tr>
<tr>
<td>HWHWH</td>
<td>3.4</td>
<td>4</td>
<td>2.5</td>
<td>10</td>
</tr>
<tr>
<td>HWHLH</td>
<td>3.3</td>
<td>5</td>
<td>3.3</td>
<td>6</td>
</tr>
<tr>
<td>HLHLH</td>
<td>3.1</td>
<td>6</td>
<td>3.6</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>3.0</td>
<td>7</td>
<td>3.4</td>
<td>5</td>
</tr>
<tr>
<td>HSHLH</td>
<td>2.5</td>
<td>8</td>
<td>2.8</td>
<td>7</td>
</tr>
<tr>
<td>HWLWH</td>
<td>2.5</td>
<td>9</td>
<td>1.8</td>
<td>15</td>
</tr>
<tr>
<td>HEH</td>
<td>2.5</td>
<td>10</td>
<td>2.7</td>
<td>8</td>
</tr>
<tr>
<td>HLSH</td>
<td>2.4</td>
<td>11</td>
<td>2.7</td>
<td>9</td>
</tr>
<tr>
<td>HWLH</td>
<td>2.3</td>
<td>12</td>
<td>2.3</td>
<td>11</td>
</tr>
<tr>
<td>HEHEH</td>
<td>1.9</td>
<td>13</td>
<td>1.9</td>
<td>14</td>
</tr>
<tr>
<td>HSLH</td>
<td>1.7</td>
<td>14</td>
<td>1.9</td>
<td>12</td>
</tr>
<tr>
<td>HEHLH</td>
<td>1.7</td>
<td>15</td>
<td>1.9</td>
<td>13</td>
</tr>
<tr>
<td>HWSH</td>
<td>1.7</td>
<td>16</td>
<td>1.7</td>
<td>16</td>
</tr>
<tr>
<td>HEHEHLH</td>
<td>1.1</td>
<td>17</td>
<td>1.1</td>
<td>19</td>
</tr>
<tr>
<td>HLSLHL</td>
<td>1.0</td>
<td>18</td>
<td>1.1</td>
<td>18</td>
</tr>
<tr>
<td>HWHWHLH</td>
<td>1.0</td>
<td>19</td>
<td>0.7</td>
<td>23</td>
</tr>
<tr>
<td>HLHSH</td>
<td>0.9</td>
<td>20</td>
<td>1.0</td>
<td>20</td>
</tr>
<tr>
<td>(Rest)</td>
<td>36.6</td>
<td>33.8</td>
<td>38.5</td>
<td>35.9</td>
</tr>
</tbody>
</table>

This table shows the frequency of the 20 most frequent activity schedules in the Baseline scenario and compares it with the frequency ($f$, in percent) and rank ($rk$) in the other scenarios.

### 7.4.2 Stochastic results

In the following, the results after matching are presented. In contrast to the previous section, the results are of a stochastic nature, and therefore the analysis will also cover mostly the distribution along with point estimates. First, the frequency and duration of activity types under the four scenarios are examined. After the effect on uncontrolled variables is analyzed, a
sensitivity analysis is performed to assess the impact of data sets describing different populations and biased survey data.

### 7.4.2.1 Activity count and duration

The results after matching are summarized in Fig. 7.3. There is some slight bias, mostly well within the range of ±1 %, and about −3 % for frequency of Education activities (third facet on the left). This is the result of the statistical matching, as this bias is present even for the Baseline scenario where the calibration does not alter the original weights. A possible reason might be that the data sets used for the matching describe slightly different populations. The most notable difference, the absence of persons under 6 years in the microcensus, has been accounted for; however, the microcensus weights are based on the de facto population while the register survey reports the de jure population. Fortunately, the relative deviation of the other scenarios from the Baseline scenario is, on average, much smaller than the bias in the Baseline scenario. The external specification of activity type frequency, on average, is not expected to contribute substantial bias to the resultant synthetic population.

Table 7.3 shows the relative error and the coefficient of variation of activity frequency and duration, compared to the Baseline scenario. The relative error is very close to that shown in Table 7.1; an indication that little additional error is introduced with the statistical matching. The CVs can be considered negligible.

### 7.4.2.2 Uncontrolled variables

Figures 7.4 and 7.5 show the effect of the statistical matching on uncontrolled sociodemographic and travel variables, respectively. According to Fig. 7.4, the error is mostly well within ±6 % for sociodemographic variables. For the marital status, separated is over- and single is underrepresented even in the Baseline scenario. Education remains mostly unchanged; a tendency towards higher education can be seen in the Delivery scenario. Generally, the resulting populations are biased towards lower income levels, fewer cars, less driving licenses and less access to a car. This bias is reduced in the Delivery scenario, aggravated in the Home office scenario and balanced again in the Combined scenario. A conceivable correlation between leisure activities and an expensive lifestyle could explain the preference for the
Chapter 7. Adapting activity chains to future scenarios

Figure 7.3: Relative error of frequency and duration per activity type

This boxplot compares the distribution of the frequency and duration for the supported activity types in the final population to the corresponding value that has resulted from calibration. The comparison is carried out for all scenarios; each scenario has been run 100 times to obtain a sample of these error distributions.
### Table 7.3: Frequency and duration by activity type, after matching

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Value</th>
<th>Scenario</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Mean [%]</td>
<td>B</td>
<td>-1.62</td>
<td>1.91</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>CV [%]</td>
<td>D</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
</tr>
<tr>
<td>Work</td>
<td>Mean [%]</td>
<td>C</td>
<td>-20.04</td>
<td>0.03</td>
<td>-20.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.15</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td>Education</td>
<td>Mean [%]</td>
<td></td>
<td>-0.01</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.22</td>
<td>0.23</td>
<td>0.24</td>
</tr>
<tr>
<td>Leisure</td>
<td>Mean [%]</td>
<td></td>
<td>20.01</td>
<td>19.98</td>
<td>19.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.11</td>
<td>0.12</td>
<td>0.13</td>
</tr>
<tr>
<td>Shoppi ng</td>
<td>Mean [%]</td>
<td></td>
<td>-19.99</td>
<td>-19.95</td>
<td>-19.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
</tr>
<tr>
<td>Home</td>
<td>Mean [%]</td>
<td></td>
<td>5.72</td>
<td>-0.67</td>
<td>4.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.15</td>
<td>0.16</td>
<td>0.14</td>
</tr>
<tr>
<td>Work</td>
<td>Mean [%]</td>
<td></td>
<td>8.15</td>
<td>-1.88</td>
<td>5.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.10</td>
<td>0.10</td>
<td>0.11</td>
</tr>
<tr>
<td>Education</td>
<td>Mean [%]</td>
<td></td>
<td>1.90</td>
<td>-0.75</td>
<td>1.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.15</td>
<td>0.14</td>
<td>0.15</td>
</tr>
<tr>
<td>Leisure</td>
<td>Mean [%]</td>
<td></td>
<td>5.20</td>
<td>-2.26</td>
<td>2.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.15</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>Shopping</td>
<td>Mean [%]</td>
<td></td>
<td>5.12</td>
<td>0.00</td>
<td>4.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
<td>0.26</td>
<td>0.25</td>
</tr>
</tbody>
</table>

This table shows a cross tabulation of activity type frequency and duration for each scenario, after statistical matching. For each scenario, and for each activity type, the relative differences to the corresponding Baseline values and the coefficients of variation are given.

Latter in the Delivery scenario. However, the reduction of work trips in the Home office scenario seems to put more weight on the nonworking part of the population.

For travel variables (Fig. 7.5), almost unchanged distributions in the
Figure 7.4: Error of sociodemographic indicators, after matching

This two-page plot shows, for each classification from a list of sociodemographic variables, its absolute share in the survey data and the distribution of the error under each scenario when comparing to the survey data. Figure 7.5 is a similar plot for travel variables.

Baseline scenario are observed; the error varies between $-6\%$ and $10\%$ for the other scenarios. For daily travel times, the Home office scenario slightly prefers shorter times and distances, while the Delivery and Combined scenarios put a strong preference on longer trips—again, the result of
the increased share of leisure trips. The situation is similar for the travel distance by car; however, the preference for longer distances is not as strong as with the daily travel time, and the Combined scenario is almost unbiased with respect to these indicators. In all scenarios a reduction of transit trips can be observed, perhaps caused implicitly by a reduction of work trips and by the fact that leisure trips are mostly undertaken by car.
Figure 7.5: Error of travel indicators, after matching
Figure 7.5: Error of travel indicators, after matching (cont.)
7.4.2.3 Population mismatch

In an earlier version of the simulation an attempt was made to match the calibrated microcensus with the full register survey, not accounting for the population mismatch resulting from the absence of small children younger than 6 years of age in the microcensus. That procedure has resulted in an extreme overrepresentation of Education activities and a slight underrepresentation of other activity types. All records in the register survey corresponding to small children are matched to microcensus records that very likely include an Education activity and prefer out-of-home leisure time to time at home.

The results presented earlier in Section 7.4.2.1 were obtained by ignoring small children for the statistical matching. In this subsection simulations in which the age threshold has been varied between zero (i.e., using all records) and 10 in steps of two are analyzed. For each scenario and each age threshold, only one simulation has been run. Figure 7.6 shows the relative error of the frequency and duration, respectively, of Education activities. The error in frequency introduced by the population mismatch (left-hand side plot) is much larger than the difference between the scenarios. However, although the error in duration shows a negative trend with increasing age threshold (right-hand side plot), the stochastic variation reverses the trend in some cases.

7.4.2.4 Biased survey data

For the simulation of a highly biased survey, the weights of the microcensus are transformed through substitution of each observation’s weight by its $k$-th power with $k \in \{0, 1, 2\}$. This process leads to three scenarios: uniform weights, unaltered weights, and exaggerated weights. For each scenario and each weight transformation, 10 synthetic populations were generated; all of them satisfied the external constraints. Figure 7.7 shows the distribution of the errors compared with the original microcensus data. The differences between the scenarios are often negligible compared with the differences between different initial weights. A large additional error introduced by the matching process might be an indicator for potential distortion of the weights.
7.5 Conclusion

This chapter presents a novel approach to weighting a sample of persons with activity schedules so that given targets for shares of different activity types are satisfied. The weighting is then used to randomly distribute the persons over the study area. An application is presented in which a 10% synthetic population for all of Switzerland is created.

The methodology used for reweighting the synthetic population has been successfully applied in the field of survey statistics for more than 20 years. Stochastic distribution is considered as a special case of statistical matching, a relatively new area of research focusing on combining data sets according to common attributes. Both methods can be applied in a straightforward fashion to the problem at hand, with very little modeling and computational effort. The result is a synthetic population that matches the targets almost perfectly and introduces very little bias, provided that the matched data sets refer to the same population. Only a little programming effort was required thanks to the availability of free implementations for survey calibration and statistical matching for the R platform for statistical computing.
Figure 7.7: Relative error of frequency and duration for distorted weights

- Home, frequency
- Home, duration
- Work, frequency
- Work, duration
- Education, frequency
- Education, duration
- Leisure, frequency
- Leisure, duration
- Shopping, frequency
- Shopping, duration

Exponent applied to original weights

Relative error [%]
An expectation-maximization approach, as shown by Han and Polak (2002) for the very similar case of imputation of missing values, can be used during the matching stage to update the weighting of the donor dataset. The weight update would offer better control of the distribution of the imputed values and allow for minor differences between recipient and donor populations. Further potential enhancements include the control for total duration per activity type and support for a finer categorization of activities (e.g., the distinction between daily and long-term shopping). Finally, the weighting approach could be compared with a model-based approach by using actual simulation results of a transportation model.
Chapter 8

Accelerating weighted random sampling without replacement

Random sampling from discrete populations is one of the basic primitives in statistical computing, and also frequently applied when generating synthetic populations. This chapter starts by briefly introducing weighted and unweighted sampling with and without replacement. The case of weighted sampling without replacement appears to be most difficult to implement efficiently, which might be one reason why the implementation available in the statistical software package R performs slowly for large problem sizes. Four alternative implementations for the case of weighted sampling without replacement are presented, with an analysis of their run time and correctness.

8.1 Introduction

Random sampling from discrete populations is one of the basic primitives in statistical computing. This chapter focuses on a specific variant: sampling without replacement from a finite population with a non-uniform weight distribution. Applications of weighted sampling without replacement include market surveys, quality control in manufacturing, and on-line advertising (Efraimidis, 2010). For generating synthetic populations, weighted sampling without replacement can be applied to stochastically convert fractional survey weights (see Chapters 3 and 4) to integer weights.
The “Truncate-Replicate-Sample” method (Lovelace and Ballas, 2013) uses the fractional component of the survey weights $w_i$ as weights in sampling without replacement. The resulting sample decides if $\lfloor w_i \rfloor$ or $\lfloor w_i \rfloor + 1$ copies of an observation are included in the synthetic population.

Throughout this chapter, the term weight refers to the relative probability that an item is sampled. A related problem, sampling from a population with given inclusion probabilities (without specifying an order), is beyond the scope of this chapter.

A surprisingly elegant algorithm for the seemingly difficult problem of weighted random sampling without replacement has been devised by Efraimidis and Spirakis (2006), compared to which other solutions (e.g., the heap-based algorithm by Wong and Easton (1980)) seem unnecessarily difficult. While the authors provide a prototype implementation in the Java language, no formal assessment of its correctness has been presented so far, and some rework seems necessary to reuse this implementation. This chapter fills this gap by providing a thoroughly validated, optimized and ready-to-use implementation for the statistical software package R (R Core Team, 2017).

First, different techniques for sampling from discrete populations are reviewed. Several implementations for sampling without replacement are discussed, this includes evaluation of run time performance and correctness. The chapter concludes with suggestions for incorporating the findings into R.

### 8.2 Sampling from discrete populations

Algorithm 8.1 is offered as a definition of sampling from discrete populations with or without replacement from arbitrary, including uniform, weight distributions. (A pair of dice $\heartsuit \diamondsuit$ indicates random draws.)

From this definition, it can be observed that sampling with replacement appears to be a simpler problem than sampling without replacement, as the lines 5 to 8 in Algorithm 8.1 are not required. Furthermore, if all weights $p_i$ are equal, the problem is simpler as well: The selection probability $P(i = k)$ of the sampled items in line 4 always equals $\frac{1}{n}$ and does not have to be computed explicitly.

In the framework of Algorithm 8.1; sampling without replacement with
Algorithm 8.1 sample\((n, s, \text{replace}, p_i)\)

**Require:** \(n\): Size of the population

**Require:** \(s\): Number of items to sample

**Require:** replace: true to request sampling with replacement

**Require:** \(p_i\): Weight of each item for \(i \in \{1, \ldots, n\}\)

**Ensure:** Returns a vector \(k_j \in \{1, \ldots, n\}\) for \(j \in \{1, \ldots, s\}\) that contains the indexes of the items sampled

1: if \(s = 0\) then
2: return vector of length 0
3: end if
4: Randomly select \(k\) so that, for all \(i\), \(P(k = i) = \frac{p_i}{\sum_j p_j}\)
5: if not replace then
6: \(n \leftarrow n - 1\)
7: remove item \(k\) from \(p_i\)
8: end if
9: return \(k \oplus \text{sample}(n, s - 1, \text{replace}, p_i)\)

non-uniform weights seems to be the hardest problem. This intuition carries over to the more specialized algorithms for sampling with and without replacement, and with uniform or arbitrary weights, which are presented in the remainder of this section.

### 8.2.1 Sampling with replacement

The with replacement case corresponds to repeated selection of \(k\) from a fixed discrete weight distribution. The uniform case can be implemented easily by transforming the output of a random number generator that returns uniformly distributed floating-point numbers in \([0, 1)\). (Implementing such a random number generator is nontrivial in itself but outside the scope of this chapter.)

More work is needed in the non-uniform case: Here, Walker’s alias method \(\text{\cite{Walker}}\), which is also used in R, is an option. Assuming
Chapter 8. Accelerating weighted random sampling without replacement

w.l.o.g. \[ \sum_j p_j = n, \] it is possible to construct a subdivision \((l_i, r_i, s_i)\) with 
\(i, l_i, r_i \in \{1, \ldots, n\}\) and \(0 < s_i \leq 1\) so that

\[
p_i = \sum_{j : l_j = i} s_j + \sum_{j : r_j = i} (1 - s_j).
\]

Sampling an item requires sampling from \(\{1, \ldots, n\}\) (to choose \(i\)) and then
sampling from \([0, 1)\) (to choose \(l_i\) or \(r_i\)): If the random number is less than
\(s_i\), item \(l_i\) is chosen, otherwise item \(r_i\). (Figuratively, the probability mass
given by \(p_i\) is distributed over \(n\) “boxes” so that the space in each box \(i\)
is assigned to at most two items \(l_i\) and \(r_i\). The share occupied by item \(l_i\)
in box \(i\) is given by \(s_i\). Some items may be distributed over several boxes.
Sampling an item means selecting a box and choosing between the two
items in this box.)

As an example, assume \(p_1 = 1.2\) and \(p_2 = 0.8\). A possible split is
\(l = [1, 1], r = [1, 2]\) and \(s = [1, 0.2]\). Drawing \(i \in 1, 2\) yields both results
with probability 0.5. Only for \(i = 2\) element 2 is chosen with probability 0.8,
which amounts to a joint probability \(P(2) = 0.5 \cdot 0.8 = 0.4 = p_2/(p_1 + p_2)\).

Walker’s alias method is optimal, requiring only \(O(n)\) preprocessing
time (in a modification suggested by Vose (1991)). Hence, for non-uniform
weights, the run time is at least \(O(n + s)\), and the input size \(n\) will dominate
unless \(s \gg n\). More recently, Marsaglia et al. (2004) have suggested
a table-based method that seems to perform much faster in practice but
expresses the weights as rationals with a fixed base and is therefore not
usable directly for distributions with a large range. Shmerling (2013)
presents a comprehensive review and suggests a general method suitable
even for quasi-infinite ranges.

### 8.2.2 Sampling without replacement

In the **without replacement** case, each selected item is removed from the
collection of candidate items. Again, the uniform case is much simpler. An
array of size \(n\), initialized with the natural sequence, can be used for storing
the candidate items. The selection of the item corresponds to choosing an
index at random in this array. Removal of an item with known index can be
done in \(O(1)\) time by simply replacing it with the last item in the array and
truncating the array by one.
For the non-uniform case, lines 4 and 7 in Algorithm 8.1 can be implemented with a data structure that maintains a subdivision of an interval into $n$ subintervals and allows lookups and updates. Walker’s alias method seems to be ill-suited for this purpose, as each item potentially spreads over several “boxes”, and an efficient update algorithm seems elusive. Wong and Easton (1980) propose a data structure similar to a heap that can be initialized in $O(n)$ time and supports simultaneous lookup and update in $O(\log n)$ time, the reader is referred to the original paper for details.

8.2.3 Sampling according to selection probabilities

Tillé (2006) defines a more rigorous framework for sampling algorithms from the perspective of the likelihood that a sample is selected based on a given sampling design. In the context of that framework, Algorithm 8.1 belongs to the class of “draw by draw” algorithms. For the application of sampling theory, the order of the selected elements is not important and usually ignored; in contrast, Algorithm 8.1 returns an ordered sequence of sampled elements.

8.3 Implementation

R offers reasonably efficient implementations for all cases except non-uniform sampling without replacement. The stock implementation for weighted random sampling without replacement requires $O(n \cdot s)$ run time, which is equivalent to $O(n^2)$ if $s = O(n)$. This chapter explores alternative approaches: rejection sampling, one-pass sampling and reservoir sampling. Only the first can be described formally within the framework of Algorithm 8.1; however an actual implementation would use sampling with replacement as a subroutine. The last two are based on an arithmetic transformation of a weight distribution.

8.3.1 Rejection sampling

In the framework of Algorithm 8.1, rejection sampling corresponds to flagging sampled items as “invalid” (instead of removing them) in line 7, and repeating the sampling in line 4 until hitting a valid item. Note that the distribution of the result is not modified if invalid items are purged.
occasionally. This corresponds to the class of “rejective algorithms” in the framework of Tillé (2006).

Therefore, sampling without replacement can be emulated by repeated sampling with replacement, as shown in Algorithm 8.2. The general idea is to sample slightly more items than necessary (with replacement), and then to throw away the duplicate items. If the resulting sequence of items is shorter than requested, the result for a much smaller problem is appended. In Algorithm 8.2, duplicate items in the result of a sampling with replacement (line 1) correspond to invalid items in the rejection sampling, and the recursive call in line 7 corresponds to purging the invalid items.

Algorithm 8.2 sample.rej\((n, s, p_i)\)

Require: \(n\): Size of the population
Require: \(s\): Number of items to sample
Require: \(p_i\): Weight of each item for \(i \in \{1, \ldots, n\}\)
Ensure: Returns a vector \(k_j \in \{1, \ldots, n\}\) for \(j \in \{1, \ldots, s\}\) that contains the indexes of the items sampled

1: \(k_i \leftarrow \text{unique(sample}(n, \text{expected.items}(n, s), \text{true}, p_i))\)
2: \(l \leftarrow \text{length}(k_i)\)
3: if \(l \geq s\) then
4: return the first \(s\) items of \(k_i\)
5: end if
6: remove items \(k_i\) from \(p_i\)
7: return \(k_i \oplus \text{sample.rej}(n - l, s - l, p_i)\)

Here, expected.items\((n, s)\) is an estimate for the number of items that need to be drawn with replacement, so that the result can be expected to contain at least \(s\) unique items. (An incorrect estimate only affects the run time, not the correctness of the algorithm.) Note that, with expected.items\((n, s) = 1\) everywhere, Algorithms 8.1 and 8.2 are in fact identical. For a uniform distribution, it can be shown that the result has approximately \(s\) unique items in expectation with expected.items\((n, s) = n(H_n - H_{n-s}) = n \sum_{i=n-s+1}^{n} \frac{1}{i}\). This is an underestimate for non-uniform distributions. Nevertheless, the implementation in this package uses this estimate, capped at \(2n\). As shown in Section 8.5, this algorithm performs worse than the alternatives shown in the following section in the majority of cases, but still better than the
stock implementation for large values of $n$ and $s$. Therefore, no tuning of the estimation of the number of expected items has been carried out.

### 8.3.2 One-pass sampling

A particularly interesting algorithm has been devised only recently by Efraimidis and Spirakis (2006). In the simplest version (here referred to as one-pass sampling), it is sufficient to draw $n$ random numbers, combine them arithmetically with the weight distribution $p_i$, and perform a partial sort to find the indexes of the $s$ smallest items. Algorithm 8.3 is a modified version of Algorithm A in the original paper that operates on the logarithmic scale for increased numerical stability.

#### Algorithm 8.3 sample.rank($n, s, p_i$)

<table>
<thead>
<tr>
<th>Require:</th>
<th>$n$: Size of the population</th>
<th>Require:</th>
<th>$s$: Number of items to sample</th>
<th>Require:</th>
<th>$p_i$: Weight of each item for $i \in {1, \ldots, n}$</th>
<th>Ensure:</th>
<th>Returns a vector $k_j \in {1, \ldots, n}$ for $j \in {1, \ldots, s}$ that contains the indexes of the items sampled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1: $r_i \leftarrow \text{Exp}(1)/p_i$ for all $i \in {1, \ldots, n}$</td>
<td>2: <strong>return</strong> the positions of the $s$ smallest elements in $r_i$</td>
</tr>
</tbody>
</table>

The arithmetic transformation of the weight distribution is carried out in line 1. A sequence of i.i.d. samples from the exponential distribution with rate 1 is divided by the weights, the order of the results defines the sampling order. Intuitively, an item with a large weight has a larger probability of appearing earlier in this sorting order. Efraimidis and Spirakis (2006) prove that Algorithms 8.1 and 8.3 are equivalent.

The algorithm amazes with its elegance and simplicity. This also allows for almost trivial parallelization, provided that independent random number generators are available to each thread. Computational complexity is dominated by the partial sort (which can be implemented in $O(n + s \log n)$, or even in $O(n)$ for floating-point numbers (Terdiman, 2000)). However, the cost of generating $n$ random variates may outweigh the cost for sorting even for moderately large values of $s$. The next subsection describes an extension to overcome this issue.
Chapter 8. Accelerating weighted random sampling without replacement

Algorithm 8.4 sample.expj(n, s, p_i)

Require: \( n \): Size of the population
Require: \( s \): Number of items to sample
Require: \( p_i \): Weight of each item for \( i \in \{1, \ldots, n\} \)
Ensure: Returns a vector \( k_j \in \{1, \ldots, n\} \) for \( j \in \{1, \ldots, s\} \) that contains the indexes of the items sampled

1: Initialize reservoir with the first \( s \) elements
2: Set keys for these elements based on their weight and one random number per item
3: while not all items processed do
4: Choose item with lowest key in the reservoir
5: Determine number of items to skip, based on this key and a random number
6: Find and remove item with the lowest key in the reservoir
7: Add current item to the reservoir
8: Set the key of the new item based on its weight and a random number
9: end while
10: return Items in reservoir sorted by their key

8.3.3 Reservoir sampling

Reservoir sampling with exponential jumps is a modified version of one-pass sampling. A reservoir of “active” items is maintained. Each generated random number decides how many input items are skipped until the current “least likely” item is removed from the reservoir. Algorithm 8.4 shows a verbal description, further details and formal proofs of correctness are beyond the scope of this chapter and can be found in (Efraimidis and Spirakis, 2006). Only \( O(s \log \frac{n}{s}) \) random numbers (in expectation) are needed with this extension, whereas the simple version always requires \( n \) random numbers. The exponential jumps method requires fewer updates of the reservoir (and therefore fewer random numbers and less run time) if the weights are arranged in descending order. In addition to drawing random numbers, the extraction of the smallest item from a priority queue (line 4) is the most expensive operation.
8.4 Implementation

The wrswoR package (Müller, 2017a) contains implementations for the algorithms presented in the previous section: One R implementation of rejection sampling (Algorithm 8.2, denoted by \textit{rej}), two implementations (R and C++) of one-pass sampling (Algorithm 8.3, \textit{rank} and \textit{crank}), and one C++ implementation of reservoir sampling with exponential jumps (\textit{expj}, Algorithm 8.4). The Rcpp package (Eddelbuettel and François, 2011; Eddelbuettel, 2013) is used to generate the glue between R and C++.

In the package, the functions are prefixed with \texttt{sample_int\_}. All functions share the same interface, the function arguments correspond to those of Algorithms 8.1 to 8.4: size is the \textit{s} argument, and prob is the \textit{p} argument. See also the documentation in Appendix D for details.

For testing the new routines against the R implementation, a wrapper function \texttt{sample_int\_R()} is provided, which calls the base R function \texttt{sample.int()} with \texttt{replace = FALSE}.

The R implementations are very similar to the pseudocode: As an example, the \textit{rank} implementation is shown below.

```r
function (n, size, prob) {
  .check_args(n, size, prob)
  head(order(rexp(n)/prob), size)
}
```

The \textit{crank} implementation has been somewhat optimized for cache efficiency. Due to its relative complexity, the \textit{expj} implementation is kept very close to the pseudocode in the original paper, still this function also operates on the logarithmic scale for numerical stability. The transformation works in a fashion very similar to that of Algorithm 8.3:

The remainder of this chapter presents performance characteristics and a validation of the new implementations.

8.5 Performance

This section presents run time tests for various combinations of input parameters, attempts to provide guidance when to choose which implementation,
and discusses the correctness of the implementation. All test results shown in this section are based on data available in the \texttt{wrswoR.benchmark} package (Müller, 2017b).

### 8.5.1 Input parameters

The run time tests used different values for the function arguments \( n \), \( s \) and \( \text{prob} \). Instead of directly specifying \( s \), it is given as a proportion of \( n \), denoted by \( r = \frac{s}{n} \). The following weight distributions (used for \( p_i \)) were tested:
- \textbf{uniform} \( p_i = 1 \) everywhere
- \textbf{linear} Sequence from 1 to \( n \) (\( \{p_i\} = \{1, \ldots n\} \)), ascending (↗), descending (↘) and shuffled (←→)
- \textbf{geometric} Starting at 1, the weight is multiplied with a constant \( \alpha \) for each step (\( p_{i+1} = \alpha p_i \), ascending, descending, and shuffled); the constant is chosen so that both minimal and maximal weights and the sum of weights is still representable as a floating-point number.

The geometric case is very extreme and unlikely to occur in practice, it is included here to test potential limitations of the implementations.

### 8.5.2 Run time

The run time was measured using the \texttt{microbenchmark} package (Mersmann, 2015) in block order with a warmup of 10 iterations using the default 100 iterations. The tests ran on a single core of an Intel Xeon CPU X5680 clocked at 3.33 GHz with 12 MB cache, running Red Hat Enterprise Linux Server release 7.2, R version 3.2.3, and version 0.4 of the \texttt{wrswoR} package.

Figure 8.1 presents an overview of the median run time for different input sizes, output size ratios, weight distributions and implementations. The R implementation is outperformed by all other implementations for \( n \approx 10\,000 \), in many cases even for much smaller inputs. In the log-log scale used here, the slope of the curves translates to computational complexity; the steeper slope for the R implementation corresponds to its quadratic complexity compared to the only slightly superlinear complexity of the other algorithms. No data were obtained for the R and \texttt{rej} implementations if the computation would have taken too long, this is reflected by a premature ending of the corresponding curves in Fig. 8.1.
Figure 8.1: Median run times

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>R</th>
<th>rej</th>
<th>rank</th>
<th>crank</th>
<th>expj</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>n</strong></td>
<td>102</td>
<td>104</td>
<td>106</td>
<td>102</td>
<td>104</td>
</tr>
<tr>
<td><strong>Run time (s)</strong></td>
<td>10^{-4}</td>
<td>10^{-2}</td>
<td>10^{-1}</td>
<td>10^{-4}</td>
<td>10^{-2}</td>
</tr>
</tbody>
</table>

- **r = 0.01**
- **r = 0.1**
- **r = 1**
As expected, the \textit{expj} implementation is among the fastest, especially for \( r \ll 1 \). In the case \( r = 0.01 \) for the \textit{geometric ascending} distribution, the new implementations win only by a margin; in particular, the run time of \textit{expj} depends on the ordering of the weights which is unfavorable here.

The \textit{rej} and \textit{rank} implementations exhibit initial costs on the sub-millisecond scale even for small input sizes, probably due to the fact that both are implemented purely in R. In addition, the \textit{rej} code is by far the slowest (but still faster than the stock implementation) for \textit{geometric} distributions, because in each step only a tiny fraction of items have a non-negligible weight, and hence most sampled items are rejected as duplicates (line 1 of Algorithm 8.2).
Figure 8.3: Comparison of R and expj run times for linear ascending weights

Figure 8.2 compares run times for crank and expj for the different weight distributions, values above 1 mean that expj is faster. The expj implementation seems to perform better than crank if $r$ is small or $n$ is large. For the pathological geometric cases, the run time differences between ascending and descending weights are substantial for small $r$. The advantage of the expj code for $r = 1$ and large $n$ is surprising and can only be explained with differences in run time between partial sort (which is used for crank) and priority queue (for expj).

For the break-even analysis, expj is compared to the stock implementation in Fig. 8.3 for linear ascending weights. The expj implementation can be up to about 2 times slower than the stock implementation, for absolute run times of around 10 microseconds for $n = 100$. It is remarkable that the relative performance of expj is worst with $r = 0.1$ in this case. The relative slowness of the stock implementation for the case $r = 0.01$ is due to a mandatory pre-sorting of weights using heap sort even for $s = 1$, which is not required for expj.
Figure 8.4: Break-even point of R and \( \exp j \) run times

Figure 8.4 shows a more detailed break-even point analysis for a larger choice for \( r \) and for all weight distributions tested. Compared to \( \exp j \), the stock implementation performs best with a uniform weight distribution, offsetting the break-even point to just below 500 for the best choice of \( r \approx 0.1 \). In other words, for \( n < 500 \) and \( s = \lceil 0.1n \rceil \), the stock implementation is still the best choice in the case of a uniform or near-uniform distribution, with a speedup of at most 2.12.

Because the order of the weights has an effect on the run time, it is worthwhile to evaluate if a prior sorting step may lead to better performance. The run time required for sorting has been measured on the same computing environment for the different weights distributions and problem sizes tested. For large samples, radix sort (Terdiman, 2000) is almost always the fastest alternative. The median of the overheads of sampling from a shuffled distribution of weights (w.r.t. the same distribution sorted in decreasing order) is compared with the corresponding cost of sorting in Fig. 8.5. It seems that prior sorting cannot reduce the overall run time.
Figure 8.5: Overhead of unsorted data vs. sorting costs
8.6 Correctness

This section aims at validating the new implementations. A correct implementation should satisfy the following criteria:

1. All output items are between 1 and $n$.
2. Each item occurs at most once in the output.
3. For given parameters $n$, $s$, and $p_i$, the probability that item $i$ is at position $j$ in the output (with $1 \leq i \leq n$ and $1 \leq j \leq s$) is identical for the implementation under test and the stock implementation.

Verifying these criteria seems to be challenging due to the stochasticity of the algorithms. The first two can be simply checked by observing the output. The following subsection describes a procedure for checking the third criterion.

8.6.1 Methodology

For fixed $i$ and $j$ and for fixed parameters $n$, $s$, and $p_i$, each call to the sampling routine is a Bernoulli trial with fixed success probability $\pi_{i,j}$. Repeated sampling leads to an i.i.d. sequence of Bernoulli trials. In general, computing the exact value of $\pi_{i,j}$ for large $j$ seems to require considerable computational resources. Therefore, the value of $\pi_{i,j}$ is assumed unknown, and only the equality of the proportions is tested for the different implementations using a two-sided test for equal proportions (essentially a $\chi^2$ test, implemented by the `prop.test()` function). The correctness check is performed as follows:

- The parameters $n$, $s$, and $p_i$, and the implementation under test, are fixed.
- For both the tested and the stock implementation, $N$ random samples without replacement are drawn and recorded.
- For all $i$ and $j$, the number of samples where item $i$ is in position $j$ (denoted by $f_{i,j}$) is computed.
- The counts are tested for equality of proportions, yielding a $p$-value for each tuple $(i, j)$.

In this setting, for fixed $(i, j)$, the $p$-value is itself a random variable that is distributed uniformly over $(0, 1]$ under the null hypothesis of equal proportions (i.e., if the tested implementation is correct). On the other hand, if the implementation is faulty, the rejection rate for the null hypothesis
will be large, and a substantial share of the p-values will be very close to 0. While this procedure does not constitute a proof of correctness, it offers a means to automatically test the implementations for nontrivial errors. A similar procedure (using a visual representation with violin plots) caught an implementation error in the `expj` code that occurred only in the case $1 < s < n$.

To assert the sensitivity of the testing procedure, a faulty implementation was simulated by passing altered weights to R’s implementation. The modification consists of updating $p'_i := p_i \cdot \left(1 + \text{skew} \cdot \frac{i - 1}{n - 1}\right)$, where a skew of zero means no change, and a skew of 1% corresponds to relative differences increasing between 0% and 1%.

The test for equal proportions can be substituted by Fisher’s exact test, which tends to produce lower p-values and therefore is usually more powerful than the test for equal proportions. However, Fisher’s exact test has $O(N)$ complexity, because it evaluates the density of the hypergeometric distribution on a support of the order of $N$. Using this test would have been prohibitive in the setting described here.

### 8.6.2 Example

Figure 8.6 shows a Schweder plot (Schweder and Spjøtvoll, 1982) of the p-values resulting from an experiment that draws $N = 2^{22}$ samples for $n = 7$, $s = 4$, and a geometric weight distribution with $\alpha = 1.08$, using all five implementations. Different values of $i$ and $j$ are denoted with different colors and shapes. The theoretical distribution is shown as a dotted line, and aligns very well with the observed p-values. Fisher’s combined probability test is a meta-analysis method that combines multiple p-values (from different but related studies) into one; it is implemented in the `metap` package (Dewey, 2016). For this particular run of the experiment, Fisher’s method cannot reject the null hypothesis of uniformity ($p = 0.896$).

As an example for a positive test, Fig. 8.7 shows results for the same experiment, now substituting the stock implementation with a faulty one with skew = 0.25%. Despite the relative similarity of the weight distributions, the distribution of the p-values deviates substantially from the uniform
distribution, with more p-values close to zero than expected. Here, Fisher’s method detects significant, although not overwhelming, evidence against the null hypothesis ($p = 0.0183$).

### 8.6.3 Results

A fairly comprehensive test also has been carried out, covering all $n \in \{2, \ldots, 80\}$, a subset of $s \in \{1, \ldots, n\}$, and all $(i, j)$. For each combination,
Figure 8.7: Schweder plot for p-values resulting from comparing the R implementation with a skewed version of itself

the cell frequencies $f_{i,j}$ were collected for all new implementations, and for the stock implementation with and without altered weights (using skew values between 0.25% and 16%), for $N$ ranging from $2^{10}$ to $2^{24}$ (only powers of 2). Each cell frequency was compared to that of the stock implementation. This resulted in around $5 \times 10^8$ p-values, which were again combined using Fisher’s method.
Figure 8.8: Combining p-values for a comprehensive test

Figure 8.8 shows the results of the meta-analysis separately for each $N$ and for each (supposedly correct or faulty) implementation. Comparing the stock implementation to itself (using different random seeds) resulted in a p-value of almost 1 for all $N$, the same holds for all new codes. On the other hand, all skews tested led to strong rejection of the correctness hypothesis (p-value effectively 0) sooner or later; as expected, the smaller the skew, the larger the $N$ that is required for rejection.

This comparison is less sensitive to implementation errors that occur only for specific arguments (e.g., if an implementation behaves as expected except if $n$ is a power of 2). To catch such deficiencies, it is helpful to analyze finer aggregates of the p-values. Figure 8.9 shows combined p-values separately for all pairs of $n$ and $N$ when comparing each new code to the stock implementation. Some p-values are in the range of $(0.01, 0.1]$ or even $(10^{-4}, 0.01]$, but this can be expected due to the uniform distribution of the p-values under the null hypothesis. The plot in Fig. 8.10 is similar, but shows the p-values that result from comparing the stock implementation with a skewed version of itself, for different skews. Here, for all skews except 0, the combined p-value approaches zero sooner or later as $N$ increases.
8.6. Correctness

Figure 8.9: Combined p-values for different values of $n$ and $N$, resulting from comparing each new code to the stock implementation

Figure 8.10: Combined p-values for different values of $n$ and $N$, resulting from comparing the stock implementation to a skewed version of itself
8.7 Conclusion

This chapter describes four new implementations for weighted random sampling without replacement in R: Rejection sampling, two implementations of one-pass sampling, and reservoir sampling with exponential jumps. The new implementations, even those written in pure R, clearly outperform the one provided by the base package if the number of items to choose from is just above 10,000, this threshold is below 500 for reservoir sampling with exponential jumps. Each of the algorithms presented here has its advantages:

- Rejection sampling is a simple and straightforward method that builds upon weighted sampling with replacement.
- One-pass sampling can be parallelized easily.
- Reservoir sampling with exponential jumps is fast even for degenerate weight distributions, and economical in its use of random numbers.

In particular, reservoir sampling with exponential jumps (Efraimidis and Spirakis, 2006) requires just about double the time of the stock implementation in the worst case, code optimization (such as using a cache-efficient heap structure for the priority queue) might help further reduce this threshold or even remove it entirely. Reservoir sampling performs best if large weights tend to occur before small weights, but a prior sorting step does not seem to improve run time.

For validation, the new implementations have been compared with the stock implementation by counting the number of occurrences for each item and each possible position in a large number of runs, and testing the null hypothesis of equal proportions. This yields a massive number of p-values, which can be combined using Fisher’s method, a meta-analysis technique. The validation methodology is able to clearly detect an emulated implementation error, which consisted of skewing the input frequency distribution in a predefined fashion, whereas no difference between the new and the stock implementations could be measured. So far, the detection of non-systematic errors or other failure modes have not been tested.

In order to include a faster sampling algorithm into base R, an implementation in C seems necessary. Other platforms for scientific computing, such as Python or Julia, would also benefit if this implementation was provided in an open-source library with a documented interface.

For the current implementation in R, a user might not expect a natural
operation such as random sampling to take excessive time, without the ability to interrupt it. Allowing to interrupt execution in the current implementation (via `R_CheckUserInterrupt()`) would at least save the unaware user the frustration of a lost workspace.

The algorithms presented here generate an ordered sample of items based on relative weights. If the relative importance is instead given as inclusion probabilities, and the order of the items is irrelevant, e.g., as in the application of survey sampling, the `UPxxx()` functions in the `sampling` package (Tillé and Matei, 2016) offer a viable alternative.
Chapter 9

Discussion

Assuming availability of a sample of households compatible with the study area, a synthetic population can be generated by reweighting this sample to control totals for the study area, and then replicating the households in the sample according to the estimated weights. The problem appears challenging if the control totals are given at both household and person levels, several algorithms (presented in Chapter 3) are available from the transportation planning literature. The relative complexity of these algorithms can be attributed to the nested data structure on which they operate. Once the problem is formulated on matrices and vectors of real numbers, it appears to be much easier to handle conceptually and analytically, and a long existing statistical method with a solid foundation, generalized raking (Chapter 4), can be applied in a relatively straightforward fashion. A reusable, documented and packaged software implementation tailored to the application is provided, both for domain-specific existing methods and for generalized raking (Chapter 5). The performance of the new method has been shown superior to existing approaches (Chapter 6). Thanks to the simple data structure, it can also be applied to the seemingly unrelated problem of adapting agent behavior to assumptions about the future (Chapter 7). The case of weighted sampling without replacement (Chapter 8) is another example where data transformation converts a computationally intensive problem—repeated summation—to a simple one—sorting.

As with any modeling exercise, the generation of synthetic populations is mostly driven by the available data and their cleanliness. Because of inconsistent data formats and semantics, the task of data preparation and validation of the results still remains the responsibility of the modeler. Here the statistical software environment R with its immensely rich collection
Chapter 9. Discussion

of add-on packages has proven invaluable. It is only natural to extend this software environment by contributing implementations for the domain-specific algorithms (Chapter 5 and Appendices B to F). This allows the entire process from raw data input to result validation to run automatically without leaving system boundaries in a fully reproducible fashion. The codes are not bound to a specific data schema, thus the original promise of a generalized approach to population synthesis is fulfilled to a large extent.

Based on the case study shown in Chapter 7, a set of MATSim scenarios for Switzerland is currently created (Bösch et al., 2016). The new IVT baseline scenarios will allow simulating transport and transport-related phenomena such as energy consumption or emission of air pollutants, including interactions between household members, for present average weekday, present average week-end day, and various points of time in the future with varying assumptions about behavior change. The implementation again follows the reproducibility maxim, so that updated scenarios based on more recent input data or behavioral assumptions can be created easily.

When implementing this case study, another case of simplification of a problem through data transformation has been observed. Here, hot-deck statistical matching (see Section 7.3.2) is used again to pair observations from two heterogeneous multivariate datasets. In addition to the matching variables, two matching observations are now required to be “close” to each other in terms of additional auxiliary variables, with respect to Gower’s distance. It is possible to embed arbitrary combinations of nominal, ordered, and continuous attributes into $\mathbb{R}^n$, so that Gower’s distance in the original space maps to the $L_1$-distance in the transformed space (Appendix E). After that, readily available solutions for nearest-neighbor matching (Appendix F) can be applied, which avoids computing all-pairs distances and results in a substantial reduction of the problem’s computational complexity.

All methods discussed here assume the availability of a household sample with all persons available. Unfortunately, this cannot be taken for granted. If the household structure is synthesized beforehand, a reweighting method may be applied to ensure consistency with marginal totals, but an analysis of the overall quality of the result is called for. Also, the effect of the accuracy of the synthetic population on the quality of the resulting transportation model has not been assessed so far.

As outlined in Section 2.5, reweighting is not the only option for generating synthetic populations. In a broader scope, population synthesis
with reweighting can be viewed as estimating a model of the study area’s population, and then simulating it. As such, the reweighting can be replaced by any multivariate model capable of capturing the structure inherent to the data to be reproduced. An intermediary modeling step resolves all data confidentiality issues that may arise with sharing a population derived from a protected reference sample.

The experience gained here suggests that the modeling process becomes substantially easier once a suitable data transformation to a simpler data structure is found. This transformation should be reversible and capture the data generation process: collapsing person attributes to counts (Section 4.3.1) does not fulfill these criteria. Rather than inventing new algorithms, future research should focus on finding such data transformations so that standard statistical and machine learning methods can be applied.
Bibliography


Ireland, C. T. and S. Kullback (1968) Contingency tables with given marginals, Biometrika, 55, 179–188.


Tillé, Y. and A. Matei (2016) *sampling: Survey Sampling*, [https://CRAN.R-project.org/package=sampling](https://CRAN.R-project.org/package=sampling); R package version 2.8.


given distribution, *IEEE Transactions on Software Engineering*, 17 (9)

activity-based microsimulation models, *Transportation Research Record*,
2054, 102–109.

variables with general distributions, *ACM Transactions on Mathematical
Software (TOMS)*, 3 (3) 253–256.

Verlag New York.

and ‘gather()’ Functions*, [https://CRAN.R-project.org/package=tidyr](https://CRAN.R-project.org/package=tidyr). R package version 0.6.1.

Wickham, H. and R. Francois (2016) *dplyr: A Grammar of Data Manip-
ulation*, [https://CRAN.R-project.org/package=dplyr](https://CRAN.R-project.org/package=dplyr). R package version 0.5.0.

microdata by using data from small area statistics and samples of

Wilson, A. G. (1970) *Entropy in Urban and Regional Modelling*, Pion,
London.

Sampling without Replacement, *SIAM Journal on Computing*, 9 (1)

Xie, Y. (2014) *knitr: A comprehensive tool for reproducible research
in R*, in V. Stodden, F. Leisch and R. D. Peng (eds.) *Implementing
978-1466561595.


Appendices
Appendix A

Adjusting the ratio of persons per household in Hierarchical IPF

This appendix analyzes the optimization problem for adjusting the persons-per-household ratio respecting the Principle of Minimum Discrimination Information (cf. Section 3.2.3). The problem is defined as follows: Given household expansion factors $f_h \in \mathbb{R}_+$ and the number of persons $p_h \in \mathbb{N}$ for each household $h \in H$, and grand totals $N, \nu \in \mathbb{N}$, compute updated expansion factors $f'_h \in \mathbb{R}_+$ to minimize the relative entropy

$$D(f'_h||f_h) = \sum_h f'_h \ln \frac{f'_h}{f_h}.$$ 

subject to the constraints

$$\sum_h f'_h = N \quad \text{(A.1)}$$

$$\sum_h p_h \cdot f'_h = \nu. \quad \text{(A.2)}$$

The factors $p_h$, and the totals $N$ and $\nu$ are positive, with $1 \leq p_h$. Therefore, $N < \nu < N \cdot \max_h p_h$ is required for the constraints to be satisfiable with $f_h > 0$. 
Applying the method of Lagrange multipliers to the objective function and the constraints yields the following auxiliary function:

$$\Lambda = \sum_h f'_h \ln \frac{f'_h}{f_h} + \lambda_1 \left( \sum_h f'_h - N \right) + \lambda_2 \left( \sum_h p_h \cdot f'_h - \nu \right). \quad (A.3)$$

The necessary condition $\nabla \Lambda = 0$ for the optimum of the original problem leads to the following precondition, valid for all households $h$:

$$\frac{\partial}{\partial f'_h} \Lambda = \ln \frac{f'_h}{f_h} + f'_h \frac{1}{f'_h} + \lambda_1 + \lambda_2 \cdot p_h$$

$$= \ln \frac{f'_h}{f_h} + (1 + \lambda_1 + \lambda_2 \cdot p_h)$$

$$= 0$$

$$\frac{f'_h}{f_h} = e^{-\lambda_1} \cdot (e^{-\lambda_2})^{p_h}$$

$$= c \cdot d^{p_h} \quad \text{(with} \; c := e^{-\lambda_1} \text{and} \; d := e^{-\lambda_2}). \quad (A.4)$$

Equation $(A.4)$ means that the ratio of new vs. old expansion factors is determined by the household size only, and that it follows a geometric progression with respect to the household size. Thus, the constraints can be rewritten as:

\begin{align*}
(A.1) & \iff \sum_p \sum_{h: p_h=p} f'_h = N \\
& \iff \sum_p \sum_{h: p_h=p} f_h \cdot c \cdot d^{p_h} = N \\
& \iff \sum_p \left( c \cdot d^p \cdot \sum_{h: p_h=p} f_h \right) = N \\
& \iff c \cdot \sum_p F_p \cdot d^p = N \quad \text{(A.5)}
\end{align*}

\begin{align*}
(A.2) & \iff c \cdot \sum_p p \cdot F_p \cdot d^p = \nu. \quad (A.6)
\end{align*}
The quantity \( F_p \geq 0 \) is the number of households with exactly \( p \) persons. Canceling \( c \) yields a necessary and sufficient condition for \( d \):

\[
\begin{align*}
\text{(A.5)} \land \text{(A.6)} \iff & \quad \frac{1}{N} \cdot \sum_p F_p \cdot d^p = \frac{1}{v} \cdot \sum_p p \cdot F_p \cdot d^p \\
\iff & \quad \sum_p F_p \cdot d^p = \sum_p F_p \cdot \left( N \cdot v^{-1} \cdot p \right) \cdot d^p \\
\iff & \quad 0 = \sum_p F_p \cdot \left( N \cdot v^{-1} \cdot p - 1 \right) \cdot d^p.
\end{align*}
\]

The coefficients \( a_p \) of this polynomial are less than or equal zero for \( p \leq v \cdot n^{-1} \), and greater than or equal zero for \( p > v \cdot n^{-1} \). By Descartes’ rule of signs, this polynomial has exactly one positive real root, and \( d \) (and also \( c \)) can be determined uniquely. In turn, this means that the \( f' \) derived from this solution constitute the only critical point for the Lagrange function Eq. (A.3).
Appendix B

The MultiLevelIPF package

Type  Package
Package  MultiLevelIPF
Title  Implementation of algorithms that extend IPF to nested structures
Version  0.3-6
Date  2017-03-28
Authors@R  aut, cre  Kirill Müller <krlmlr+r@mailbox.org>
Description  The IPF algorithm operates on count data. This package offers implementations for several algorithms that extend this to nested structures: “parent” and “child” items for both of which constraints can be provided.
License  GPL-3
URL  http://krlmlr.github.io/MultiLevelIPF/
BugReports  https://github.com/krlmlr/MultiLevelIPF/issues
Depends  methods
Imports  plyr, dplyr, BB, Matrix, grake (>= 0.1-1), hms, kimisc
Suggests  testthat, XML
Remotes  grake/grake
Roxygen  list(markdown = TRUE)
MultiLevelIPF-package

*Implementation of algorithms that extend IPF to nested structures*

### Description

The IPF algorithm operates on count data. This package offers implementations for several algorithms that extend this to nested structures: "parent" and "child" items for both of which constraints can be provided.

### computeMargins

*Compute margins for a weighting of a multi-level fitting problem*

#### Description

These functions allows checking a fit in terms of the original input data.

#### Usage

```r
compute_margins(fitting_problem, weights, verbose = FALSE)
margin_to_df(controls, count = NULL, verbose = FALSE)
```
fitting_problem

Arguments

fitting_problem
A fitting problem created by the fitting_problem() function.

weights
A vector with one entry per row of the original reference sample

verbose
If TRUE, print diagnostic output.

controls
Margins as returned by computeMargins or as passed to the controls parameter of fitting_problem().

count
Name of control total column, autodetected by default.

Details

computeMargins() computes margins in the format used for the input controls (i.e., as expected by the controls parameter of the fitting_problem() function), based on a reference sample and a weights vector.
margins_to_df() converts margins to a data frame for easier comparison.

See Also

ml_fit()

fitting_problem
Create an instance of a fitting problem

Description

Create an instance of a fitting problem
fitting_problem

Usage

fitting_problem(ref_sample, controls =
  list(individual = individual_controls,
        group = group_controls), field_names,
        individual_controls, group_controls,
        prior_weights = NULL)

is.fitting_problem(x)

## S3 method for class 'fitting_problem'
format(x, ...)

## S3 method for class 'fitting_problem'
print(x, ...)

special_field_names(groupId, individualId,
indiviualsPerGroup = NULL, count = NULL)

Arguments

ref_sample  The reference sample
controls     Control totals, by default initialized from the individual_controls and group_controls arguments
field_names  Names of special fields, can be constructed using the special_field_names() function
individual_controls, group_controls
Control totals at individual and group level, given as a list of data frames where each data frame defines a control
prior_weights Prior (or design) weights at group level; by default a vector of ones will be used, which corresponds to random sampling of groups
x            An object
Ignored.

groupId, individualId
   Name of the column that defines the ID of the
group or the individual

individualsPerGroup
   Obsolete.

count
   Name of control total column in control tables (use
first numeric column in each control by default).

Value

An object of class fitting_problem, essentially a named list with
the following components:

refSample  The reference sample, a data.frame.
controls  A named list with two components, individual and
group. Each contains a list of controls as data.frames.
fieldNames  A named list with the names of special fields.

Description

This function transforms a multi-level fitting problem to a representation
more suitable for applying the algorithms: A matrix with one row per
controlled attribute and one column per household, a weight vector
with one weight per household, and a control vector.

Usage
flatten_ml_fit_problem

```r
c(# flatten_ml_fit_problem(fitting_problem, 
  model_matrix_type = c("combined", 
    "separate"), verbose = FALSE)

c(as.flat_ml_fit_problem(x, model_matrix_type 
  = c("combined", "separate"), ...)
```

### Arguments

- **fitting_problem**
  A fitting problem created by the `fitting_problem()` function.

- **model_matrix_type**
  Which model matrix building strategy to use? See details.

- **verbose**
  If TRUE, print diagnostic output.

- **x**
  An object

- **...**
  Further parameters passed to the algorithm

### Details

The standard way to build a model matrix (`model_matrix = "combined"`) is to include intercepts and avoid repeating redundant attributes. A simpler model matrix specification, available via `model_matrix = "separate"`, is used by Ye et al. (2009): Here, simply one column per target value is used, which results in a larger model matrix if more than one control is given.

### Value

An object of classes `flat_ml_fit_problem`, essentially a named list.

### See Also

`ml_fit()`
Examples

```r
path <- toy_example("Tiny")
flatten_ml_fit_problem(fitting_problem = readRDS(path))
```

---

**ml_fit**

Estimate weights for a fitting problem

---

**Description**

These functions reweight a reference sample to match constraints given by aggregate controls.

- `ml_fit()` accepts an algorithm as argument and calls the corresponding function. This is useful if the result of multiple algorithms are compared to each other.
- `ml_fit_dss()` uses generalized raking and calls `grake::dss` internally.
- `ml_fit_entropy_o()` optimizes entropy using the `BB::dfsane` function.
- `ml_fit_hipf()` implements Hierarchical Iterative Proportional Fitting.
- `ml_fit_ipu()` implements Iterative Proportional Updating.

**Usage**

```r
ml_fit(algorithm = c("entropy_o", "dss", "ipu", "hipf"), fitting_problem, verbose = FALSE, ..., tol = 1e-06)
```

```r
ml_fit_dss(fitting_problem, method = c("raking", "linear", "logit"), ginv = grake::gginv(), tol = 1e-06, verbose = FALSE)
```
ml_fit_entropy_o(fitting_problem, verbose = FALSE, tol = 1e-06, dfsane_args = list())

ml_fit_hipf(fitting_problem, diff_tol = 16 * .Machine$double.eps, tol = 1e-06, maxiter = 2000, verbose = FALSE)

ml_fit_ipu(fitting_problem, diff_tol = 16 * .Machine$double.eps, tol = 1e-06, maxiter = 2000, verbose = FALSE)

**Arguments**

- **algorithm**  
  Algorithm to use

- **fitting_problem**  
  A fitting problem created by the `fitting_problem()` function.

- **verbose**  
  If TRUE, print diagnostic output.

- **...**  
  Further parameters passed to the algorithm

- **tol**  
  Tolerance, the algorithm has succeeded when all target values are reached within this tolerance.

- **method**  
  Calibration method, one of "raking" (default), "linear", or "logit"

- **ginv**  
  Function that computes the Moore-Penrose pseudoinverse.

- **dfsane_args**  
  Additional arguments (as a named list) passed to the `BB::dfsane()` function used internally for the optimization.

- **diff_tol**  
  Tolerance, the algorithm stops when relative difference of control values between iterations drops below this value.

- **maxiter**  
  Maximum number of iterations.
Value

All functions return an object of class \texttt{ml_fit}, which is a named list under the hood. The class matches the function called, e.g., the return value of the \texttt{ml_fit_ipu} function also is of class \texttt{ml_fit_ipu}.

References


See Also

\texttt{grake::dss()}, \texttt{grake::gginv()}

\texttt{BB::dfsane()}

Examples

```r
path <- toy_example("Tiny")
ml_fit(algorithm = "entropy_o", fitting_problem = readRDS(path))
ml_fit_dss(fitting_problem = readRDS(path))
ml_fit_dss(fitting_problem = readRDS(path), ginv = solve)
ml_fit_entropy_o(fitting_problem = readRDS(path))
ml_fit_hipf(fitting_problem = readRDS(path))
ml_fit_ipu(fitting_problem = readRDS(path))
```
toy_example

Access to toy examples bundled in this package

Description

Returns the paths to all available toy examples, or to a specific toy example. Load via `readRDS()`.

Usage

```r
toy_example(name = NULL)
```

Arguments

- `name` Name of the example, default: return all

Value

A named vector of file system paths.
Appendix C

The grake package

Package  grake
Title    Generalized Raking as Described by Deville et al. (1993)
Version  0.1-2
Authors@R
  ctb, cre  Kirill Müller <krlmlr+r@mailbox.org>
  aut  Andreas Alfons
  aut  Matthias Templ
  ctb  Brian Ripley (gginv() has been derived from MASS::ginv())
Description  This package implements functions for generalized raking as
described in Deville et al. (1993): “Generalized raking procedures
in survey sampling”. The code strives to provide performance and
ease of use.
License  GPL-3
URL  http://krlmlr.github.io/grake/
BugReports  https://github.com/krlmlr/grake/issues
Imports  Matrix
Suggests  testthat, sampling
Description

This package implements functions for generalized raking as described in Deville et al. (1993): "Generalized raking procedures in survey sampling". The code strives to provide performance and ease of use.

See Also

dss, gginv

dss

Calibrate sample weights

Description

Calibrate sample weights according to known marginal population totals. Based on initial sample weights, the so-called $g$-weights are computed by generalized raking procedures. The final sample weights need to be computed by multiplying the resulting $g$-weights with the initial sample weights.

Usage

dss(X, d, totals, q = NULL, method = c("raking", "linear", "logit"), bounds = NULL, maxit = 500, ginv = gginv(), tol = 1e-06, attributes = FALSE)
Arguments

X  a matrix of calibration variables.

d  a numeric vector giving the initial sample (or design) weights.

totals  a numeric vector of population totals corresponding to the calibration variables in X.

q  a numeric vector of positive values accounting for heteroscedasticity. Small values reduce the variation of the g-weights.

method  a character string specifying the calibration method to be used. Possible values are "linear" for the linear method, "raking" for the multiplicative method known as raking and "logit" for the logit method.

bounds  a numeric vector of length two giving bounds for the g-weights to be used in the logit method. The first value gives the lower bound (which must be smaller than or equal to 1) and the second value gives the upper bound (which must be larger than or equal to 1). If NULL, the bounds are set to c(0, 10).

maxit  a numeric value giving the maximum number of iterations.

ginv  a function that computes the Moore-Penrose generalized inverse (default: an optimized version of ginv). In some cases it is possible to speed up the process by using a function that computes a "regular" matrix inverse such as {solve.default}.

tol  relative tolerance; convergence is achieved if the difference of all residuals (relative to the corresponding total) is smaller than this tolerance.

attributes  should additional attributes (currently success, iterations, method and bounds) be added to the result? If FALSE (default), a warning is given
if convergence within the given relative tolerance could not be achieved.

Value

A numeric vector containing the $g$-weights.

Note

This is a faster implementation of parts of calib from package sampling. Note that the default calibration method is raking and that the truncated linear method is not yet implemented.

Author(s)

Andreas Alfons, with improvements by Kirill Müller

References


Examples

```r
obs <- 1000
vars <- 100
Xs <- matrix(runif(obs * vars), nrow = obs)
d <- runif(obs) / obs
totals <- rep(1, vars)
g <- dss(Xs, d, totals, method = "linear", ginv = solve)
g2 <- dss(Xs, d, totals, method = "raking")
```
The `gginv` function creates a function that calculates the Moore-Penrose generalized inverse of a matrix X using a fixed tolerance value and a custom implementation for computing the singular value decomposition.

Usage

```r
gginv(tol = sqrt(.Machine$double.eps), svd = base::svd)
```

Arguments

- `tol` A relative tolerance to detect zero singular values.
- `svd` A function that computes the singular value decomposition of a matrix

Details

The `svd` argument is expected to adhere to the interface of `base::svd`. It will be called as `svd(x)` (with the `nu` and `nv` arguments unset) and is expected to return a named list with components `d`, `u` and `v`.

Value

A function that accepts one argument `X` that computes a MP generalized inverse matrix for it.

Author(s)

Adapted implementation from the MASS package.
See Also

ginv, svd
Appendix D

The `wrswoR` package

Type Package
Package `wrswoR`
Title Weighted Random Sampling without Replacement
Version 1.0-10
Date 2017-04-14
Authors@R
  aut, cre Kirill Müller <krlmr+r@mailbox.org>
Description A collection of implementations of classical and novel algorithms for weighted sampling without replacement.
License GPL-3
URL http://krlmlr.github.io/wrswoR
BugReports https://github.com/krlmlr/wrswoR/issues
Depends R (>= 3.0.2)
Imports Rcpp, logging (>= 0.4-13)
Suggests `wrswoR.benchmark` (>= 0.1), import, kimisc (>= 0.2-4), testthat, roxygen2, knitr, rmarkdown, rticles (>= 0.1), knitr,citations, metap, tidyr, microbenchmark, sampling, BatchExperiments, dplyr, ggplot2, tikzDevice (>= 0.9-1)
LinkingTo Rcpp (>= 0.11.5)
VignetteBuilder knitr
Roxygen list(markdown = TRUE)
**wrswoR-package**

---

**Faster weighted sampling without replacement**

**Description**

R’s default sampling without replacement using `base::sample.int()` seems to require quadratic run time, e.g., when using weights drawn from a uniform distribution. For large sample sizes, this is too slow. This package contains several alternative implementations.

**Details**

Implementations adapted from [http://stackoverflow.com/q/15113650/946850](http://stackoverflow.com/q/15113650/946850).

**Author(s)**

Kirill Müller

**References**


**Examples**

```r
sample_int_rej(100, 50, 1:100)
```
Weighted sampling without replacement

Description

These functions implement weighted sampling without replacement using various algorithms, i.e., they take a sample of the specified size from the elements of 1:n without replacement, using the weights defined by prob. The call `sample_int_*(n, size, prob)` is equivalent to `sample.int(n, size, replace = F, prob)`. (The results will most probably be different for the same random seed, but the returned samples are distributed identically for both calls.) Except for `sample_int_R()` (which has quadratic complexity as of this writing), all functions have complexity $O(n \log n)$ or better and often run faster than R’s implementation, especially when n and size are large.

Usage

```
sample_int_R(n, size, prob)
sample_int_ccrank(n, size, prob)
sample_int_crank(n, size, prob)
sample_int_expj(n, size, prob)
sample_int_expjs(n, size, prob)
sample_int_rank(n, size, prob)
sample_int_rej(n, size, prob)
```
**Sample int R**

**Arguments**

- **n**  
a positive number, the number of items to choose from. See ‘Details.’

- **size**  
a non-negative integer giving the number of items to choose.

- **prob**  
A vector of probability weights for obtaining the elements of the vector being sampled.

**Details**

`sampe_int_R()` is a simple wrapper for `base::sample.int()`.

`sampe_int_expj()` and `sampe_int_exps()` implement one-pass random sampling with a reservoir with exponential jumps (Efraimidis and Spirakis, 2006, Algorithm A-ExpJ). Both functions are implemented in Rcpp; *_expj() uses log-transformed keys, *_expjs() implements the algorithm in the paper verbatim (at the cost of numerical stability).

`sampe_int_rank()`, `sampe_int_crank()` and `sampe_int_ccrank()` implement one-pass random sampling (Efraimidis and Spirakis, 2006, Algorithm A). The first function is implemented purely in R, the other two are optimized Rcpp implementations (*_crank() uses R vectors internally, while *_ccrank() uses std::vector; surprisingly, *_crank() seems to be faster on most inputs). It can be shown that the order statistic of $U^{(1/w_i)}$ has the same distribution as random sampling without replacement ($U \sim \text{uniform}(0, 1)$ distribution). To increase numerical stability, $\log(U)/w_i$ is computed instead; the log transform does not change the order statistic.

`sampe_int_rej()` uses repeated weighted sampling with replacement and a variant of rejection sampling. It is implemented purely in R. This function simulates weighted sampling without replacement using somewhat more draws with replacement, and then discarding duplicate values (rejection sampling). If too few items are sampled, the routine calls itself recursively on a (hopefully) much smaller problem. See also http://stats.stackexchange.com/q/20590/6432;
Value

An integer vector of length `size` with elements from `1:n`.

Author(s)

Dinre (for `*_rank()`), Kirill Müller (for all other functions)

References

http://stackoverflow.com/q/15113650/946850


See Also

`base::sample.int()`

Examples

```r
# Base R implementation
s <- sample_int_R(2000, 1000, runif(2000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_R(6, 3, p),
                      n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A, Rcpp version using std::vector
s <- sample_int_ccrank(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_ccrank(6, 3, p),
                      n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A, Rcpp version using R vectors
s <- sample_int_crank(20000, 10000, runif(20000))
```

173
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_crank(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A-ExpJ (with log-transformed keys)
s <- sample_int_expj(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_expj(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A-ExpJ (paper version)
s <- sample_int_expjs(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_expjs(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Algorithm A
s <- sample_int_rank(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_rank(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)

## Rejection sampling
s <- sample_int_rej(20000, 10000, runif(20000))
stopifnot(unique(s) == s)
p <- c(995, rep(1, 5))
n <- 1000
set.seed(42)
tbl <- table(replicate(sample_int_rej(6, 3, p),
    n = n)) / n
stopifnot(abs(tbl - c(1, rep(0.4, 5))) < 0.04)
Appendix E

The mangow package

**Package**  mangow

**Title**  Generate a Manhattan Distance Problem from a Gower’s Distance Problem

**Version**  0.1-1

**Date**  2016-06-28

**Authors@R**

* aut, cre  Kirill Müller <krlmlr+r@mailbox.org>

**Description**  The transformation of a Gower’s distance problem to a Manhattan distance problem allows using algorithms that operate on the Manhattan metric (e.g., nearest-neighbor search in the ‘RANN.L1’ package) on mixed-type datasets.

**License**  GPL-3

**URL**  http://krlmlr.github.io/mangow

**BugReports**  https://github.com/krlmlr/mangow

**Suggests**  testthat, cluster, sets, StatMatch
Convert a Gower distance problem to a Manhattan distance problem

Description

This function converts a data frame to a matrix with the same number of rows. For any two rows of the resulting matrix, the Manhattan distance equals the Gower distance in the input data between the corresponding rows.

Usage

mangow(data)

Arguments

data The input data frame with continuous, categorical (factor) and ordered variables

Value

A numeric matrix

Examples

iris_sub <- iris[c(1:2, 50:51, 100:101), ]
row.names(iris_sub) <- NULL
iris_sub
cluster::daisy(iris_sub, "gower")
mangow_iris_sub <- mangow(iris_sub)
cluster::daisy(mangow_iris_sub, "manhattan")
Appendix F

The **RANN.L1** package

**Package**  RANN.L1  
**Title**  Fast Nearest Neighbour Search (Wraps ANN Library) Using L1 Metric  
**Version**  2.5.0  
**Author**  Sunil Arya and David Mount (for ANN), Samuel E. Kemp, Gregory Jefferis, Kirill Müller  
**Maintainer**  Kirill Müller <krlmlr+r@mailbox.org>  
**Description**  Finds the k nearest neighbours for every point in a given dataset in O(N log N) time using Arya and Mount’s ANN library (v1.1.3). There is support for approximate as well as exact searches, fixed radius searches and ‘bd’ as well as ‘kd’ trees. The distance is computed using the L1 (Manhattan, taxicab) metric. Please see package ‘RANN’ for the same functionality using the L2 (Euclidean) metric.  
**License**  GPL (>=3)  
**URL**  [https://github.com/jefferis/RANN/tree/master-L1](https://github.com/jefferis/RANN/tree/master-L1)  
**Suggests**  testthat  
**Copyright**  ANN library is copyright University of Maryland and Sunil Arya and David Mount. See file COPYRIGHT for details.
RANN.L1-package  
Wrapper for Arya and Mount’s Approximate Nearest Neighbours (ANN) C++ library

Description

Wrapper for Arya and Mount’s Approximate Nearest Neighbours (ANN) C++ library

See Also

nn2

nn2  
Nearest Neighbour Search

Description

Uses a kd-tree to find the p number of near neighbours for each point in an input/output dataset. The advantage of the kd-tree is that it runs in O(M log M) time.

Usage

nn2(data, query = data, k = min(10, nrow(data)), treetype = c("kd", "bd"), searchtype = c("standard", "priority", "radius"), radius = 0, eps = 0)
Arguments

**data**
An \( M \times d \) data.frame or matrix, where each of the \( M \) rows is a point or a (column) vector (where \( d=1 \)).

**query**
A set of \( N \times d \) points that will be queried against data. \( d \), the number of columns, must be the same as data. If missing, defaults to data.

**k**
The maximum number of nearest neighbours to compute. The default value is set to the smaller of the number of columns in data

**treetype**
Character vector specifying the standard 'kd' tree or a 'bd' (box-decomposition, AMNSW98) tree which may perform better for larger point sets

**searchtype**
See details

**radius**
Radius of search for searchtype='radius'

**eps**
Error bound: default of 0.0 implies exact nearest neighbour search

Details

The RANN.L1 package utilizes the Approximate Near Neighbor (ANN) C++ library, which can give the exact near neighbours or (as the name suggests) approximate near neighbours to within a specified error bound. For more information on the ANN library please visit [http://www.cs.umd.edu/~mount/ANN/](http://www.cs.umd.edu/~mount/ANN/).

Search types: **priority** visits cells in increasing order of distance from the query point, and hence, should converge more rapidly on the true nearest neighbour, but standard is usually faster for exact searches. **radius** only searches for neighbours within a specified radius of the point. If there are no neighbours then nn.idx will contain 0 and nn.dists will contain 1.340781e+154 for that point.

Value

A list of length 2 with elements:
nn2

nn.idx  A N x k integer matrix returning the near neighbour indices.
nn.dists  A N x k matrix returning the near neighbour Manhattan distances.

Author(s)

Gregory Jefferis based on earlier code by Samuel E. Kemp (knnFinder package); L1 port by Kirill Müller

References


Examples

x1 <- runif(100, 0, 2*pi)
x2 <- runif(100, 0,3)
DATA <- data.frame(x1, x2)
nearest <- nn2(DATA,DATA)
Kirill Müller
Algorithm engineer and data scientist

PERSONAL INFORMATION

Address: Weidmannstrasse 8
8046 Zurich
Switzerland
Phone: +41 44 633 33 17
E-mail: kirill.mueller@ivt.baug.ethz.ch
Nationality: German
Date of birth: February 26, 1979
Place of birth: Odessa, Ukraine

EDUCATION

Jan 10 – Oct 16 Doctoral studies, ETH Zurich, Switzerland
Advisor: Prof. Dr. Kay W. Axhausen

Oct 02 – Dec 06 Studies of Computer Science, University of Karlsruhe, Germany
Diploma Thesis: Design and Implementation of an Efficient Hierarchical Speed-up Technique for Computation of Exact Shortest Paths in Graphs
Degree: German Diploma (M.Sc. equivalent), Final Grade: A

Oct 98 – Sep 02 Studies of Computer Science, Humboldt-University Berlin, Germany
Aug 91 – Jun 98 Heinrich-Hertz-Oberschule, Berlin, Germany
Degree: German Abitur (High School Diploma equivalent), Final Grade: B
Participation in several mathematic and computer science competitions at national level

WORK EXPERIENCE

Jan 10 – present Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Graduate Research and Teaching Assistant

Jul 98 – Dec 09 orgAnice Software GmbH, Berlin, Germany
Software Engineer
Teaching experience

Mar 11 – Jul 11  Supervision of First Year Project: *Multi-agent simulation in transportation planning*, IVT, ETH Zurich
Mar 14 – Jul 14

Extracurricular activities

Jan 05 – present  Contributions to open-source projects, in particular to the R ecosystem

Skills

Programming  C/C++, R, Java, Python, C#, VB.NET
VBA, .NET interop
\LaTeX{}, metaprogramming, compiler construction

Data processing  PostgreSQL/PostGIS, Microsoft SQL Server, SQLite, MySQL
QGIS
agent-based simulation
survey calibration and data fusion
statistical analysis and modelling
visualization, grammar of graphics

IT  Linux, Windows
docker/vagrant
system administration

Best practices  version control, Git, GitHub
reproducible research and data processing
tidy data
continuous integration
test-driven development
clean code

Language  German *native*
Russian *native*
English *full professional proficiency*
French *basic*
ARTICLES


BOOK CHAPTERS


REFEREED CONFERENCE PAPERS


CONFERENCE AND WORKING PAPERS


MASTER’S THESIS


SUPERVISED STUDENT THESSES


RESEARCH REPORTS


PRESENTATIONS


