The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–27

Calibrating survey data using iterative proportional fitting (raking)

Stanislav Kolenikov
Abt SRBI
kolenikovs@srbi.com

Abstract. This article introduces package `ipfraking` implementing weight calibration procedures known as iterative proportional fitting, or raking, of complex survey weights. The package is capable of handling a large number of control variables and trimming the weights in various ways. It also provides diagnostic tools for the weights it creates. Examples of its usage are given, and the suggested workflow is discussed.

Keywords: st0001, survey, calibration, weights, raking

1 Introduction and background

Large scale social, behavioral and health data are often collected via complex survey designs that may involve some or all of stratification, multiple stages of selection and unequal probabilities of selection (Korn and Graubard 1995, 1999). In an ideal setting, varying probabilities of selection are accounted for by using the Horvitz-Thompson estimator of the totals (Horvitz and Thompson 1952; Thompson 1997), and the remaining sampling fluctuations can be further ironed out by post-stratification (Holt and Smith 1979). However, on top of the planned differences in probabilities of obtaining a response from a sampled unit, non-response is a practical problem that has been growing more acute over the recent years (Groves et al. 2001; Pew Research Center 2012). The analysis weights that are provided along with the public use microdata by data collecting agencies are designed to account for unequal probabilities of selection, non-response, and other factors affecting imbalance between the population and the sample, thus making the analyses conducted on such micordata generalizable to the target population. In this paper, I shall discuss the specific issue in the process of creating survey weights: calibrating survey weights to known control totals to ensure that the resulting weighted data are representative of the population of interest.

1.1 Population totals

For a given finite population $\mathcal{U}$ of units indexed $i = 1, \ldots, N$, the interests of survey statisticians often lie in estimating the population total of a variable Y

$$T[Y] = \sum_{i \in \mathcal{U}} Y_i \tag{1}$$

(As is customary in sampling texts, the population quantities will be denoted with capital letters, and the sample quantities, with lowercase letters. The finite population is denoted as $\mathcal{U}$, and the sample drawn from it, as $\mathcal{S}$. The indices of units in the population are $i \in \mathcal{U}$, and those of units in the sample, $j \in \mathcal{S}$.) A lot of other analytical problems can be cast in terms of estimating the totals of the existing or auxiliary variables, and then expressing the quantities of substantive interest (means, ratios, regression coefficients) as functions of these totals (Skinner 1989). For instance, the population mean, such as mean income or the average number of hours per week spent watching TV, is the ratio of the totals

$$\bar{Y} = \frac{T[Y]}{T[1]}$$

where the denominator is a somewhat unusual total of a variable identically equal to 1, i.e., the estimator of the population size (if the latter is unknown). The mean for a domain $\mathcal{D}$ (mean income of females; TV hours of teenagers) is also a ratio of totals

$$\bar{Y}_{\mathcal{D}} = \frac{T[YZ]}{T[Z]}$$

where $Z_i = 1$ when the unit is in the domain, and 0 otherwise. Estimation of totals is thus the cornerstone building block of survey statistics.

1.2 Probability weights

Suppose now that a sample $\mathcal{S}$ of n units indexed by $j = 1, \ldots, n$ is taken from $\mathcal{U}$. If the probability to select the i-th unit is known to be π_i, then the *probability weights*, or *design weights*, are given by the inverse probability of selection:

$$w_{1i} = \pi_i^{-1} \tag{2}$$

With these weights, an unbiased (design-based, non-parametric) estimator of the total (1) is (Horvitz and Thompson 1952)

$$t_1[y] = \sum_{j \in \mathcal{S}} \frac{y_j}{\pi_j} \equiv \sum_{j \in \mathcal{S}} w_{1j} y_j, \tag{3}$$

The subindex 1 indicates that the weights w_{1i} were used in obtaining this estimator. Probability weights protect the end user from potentially informative sampling designs, in which the probabilities of selection are correlated with outcomes, and the design-based methods generally ensure that inference can be generalized to the finite population even when the statistical models used by analysts and researchers are not specified correctly (Pfeffermann 1993; Binder and Roberts 2003).

1.3 Calibrated weights

Often, survey statisticians have auxiliary information on the units in the frame. Some of that information can be used at the sampling stage to inform stratification and

clustering. When creating areal probability samples of human populations, the survey designers have geographic information on the sampled units: from strata that can be defined as regions, states or provinces, and the primary sampling units that can be defined as districts, counties or census tracts; down to the address of the sampled household. When creating establishment survey samples, the frame information that survey statisticians may have at their disposal can include the industry classification code(s) and size of establishment (number of employees or the total revenue). When drawing samples from lists of persons, such as members of a professional organization or patients of a hospital, the frame information can include age and gender of an individual. If such additional information is available, it is usually beneficial to include it at the sampling stage to create more efficient designs. Unequal probabilities of selection are then controlled with probability weights, implemented as `[pw=`*exp*`]` in Stata (and can be permanently affixed to the data set with `svyset` command).

In many situations, however, usable information is not available beforehand, and may only appear in the collected data. In the above example with areal samples, the census totals of the age and gender distribution of the population may exist, but age and gender of the sampled units is unknown until the survey measurement is taken on them. It is still possible to capitalize on this additional data by adjusting the weights in such a way that the reweighted data conforms to these known figures. The procedures to perform these reweighting steps are generally known as *weight calibration* (Deville and Särndal 1992; Deville et al. 1993; Kott 2006, 2009; Särndal 2007).

Suppose there are several (categorical) variables, referred to as *control variables*, that are available for both the population and the sample (age groups, race, gender, educational attainment, etc.). The *post-stratification* adjustment consists of breaking down the population into post-stratification cells defined by specific levels of the control variables (i.e., a cell in a multivariate contingency table), and adjusting the weights within each cell so that the weights sum to the known total. Thus for the unit j in the (population-level) post-stratification cells $\mathcal{C}_k$, the post-stratified weight is

$$w_{2j} = w_{1j} \frac{\sum_{i \in \mathcal{U}} \mathbb{1}[i \in \mathcal{C}_k]}{\sum_{l \in \mathcal{S}} w_{1l} \mathbb{1}[l \in \mathcal{C}_k]} \tag{4}$$

where $\mathbb{1}[\cdot]$ is an indicator function taking the value of one when its argument is true, and zero otherwise. While the probability weight w_{1i} is fixed, the post-stratified weight w_{2j} is random, as it depends on the random sample sizes of $\mathcal{C}_k \cap \mathcal{S}$. The existing Stata `svyset` options, `poststrata()` and `postweight()` (see [SVY] **poststratification**), handle this situation and, in particular, provide appropriate standard errors. The total estimator based on w_{2j} will be naturally denoted as $t_2[y]$, and the expression for it coincides with (3) by replacing the probability weights w_{1j} with post-stratified weights w_{2j}.

In more complex situations, using say five calibration variables, such as gender, age groups, race/ethnicity, education, and urbanicity, leads to five-way contingency tables that will likely have zero or very small count cells. Battaglia et al. (2009) suggest collapsing the categories that contain less than 5% of either the sample cases or the population units. For a sample of size $n = 1,000$ typical for social science studies or

public opinion polls, this recommendation translates to cells of size $\sum_{j\in\mathcal{S}} \mathbb{1}[i \in \mathcal{C}_k] \leq 50$. Instead of adjusting every cell of a multi-way table, weight calibration can target adjusting only the margins, or low level interactions, via an iterative optimization aimed at satisfying the *control totals* for the control variables $\mathbf{x} = (x_1, \ldots, x_p)$:

$$\sum_{j\in\mathcal{S}} w_{3j}\mathbf{x}_j = T[\mathbf{X}_j] \tag{5}$$

where the right hand side is assumed to be known from a census or a higher quality survey. Deville and Särndal (1992) framed the problem of finding a suitable set of weights as that of constrained optimization with the control equations (5) serving as constraints, and optimization targeted at making the discrepancy between the design weights w_{1j} and calibrated weights w_{3j} as close as possible, in a suitable sense. Again, the appropriate total estimator can be denoted as $t_3[y]$.

1.4 Raking algorithm

An early algorithm to perform weight calibration is often attributed to Deming and Stephan (1940) who used it to adjust the counts in a contingency table to satisfy the known margins in log-linear analysis. In applications to the survey weights, the algorithm is described below. At a basic level, this algorithm consists of an outer cycle that checks convergence criteria, and an inner cycle that iterates over the control variables. The multi-index notation of the intermediate weights, $w_j^{k,v}$, indicates the weight of unit j computed in the outer cycle k after post-stratifying with respect the v-th variable. Thus k runs from 1 to a predefined maximum number of iterations K, and v runs from zero (indicating the input weight to a given iteration) through 1 (indicating adjustment with respect to the first control variable) to p (indicating adjustment with respect to the last control variable).

Algorithm 1: basic raking

1. Initialize the iteration counter $k \leftarrow 0$ and the weights as $w_j^{0,p} \leftarrow w_{1j}$. (That is, use the base weights to initialize the raked weight; the superscript $0, p$ is only used for consistency with notation used in the next step.)

2. Increment the iteration counter $k \leftarrow k + 1$, update the weights $w_j^{k,0} \leftarrow w_j^{k-1,p}$. (That is, use the end result of the previous outer cycle iteration to initialize the weights for the current outer cycle iteration.)

3. Inner cycle: go over the control variables $v = 1, \ldots, p$ and update the weights

$$w_j^{k,v} = \begin{cases} w_j^{k,v-1} \dfrac{T[X_v]}{\sum_{l\in\mathcal{S}} w_l^{k,v-1} x_{vl}}, & x_{vj} \neq 0 \\ w_j^{k,v-1}, & x_{vj} = 0 \end{cases}$$

   (That is, post-stratify with respect to the v-th control variable.)

4. If discrepancies between the weighted totals $\sum_{j\in\mathcal{S}} w_j^{k,p} x_v$ and the target totals $T[X_v]$ are within prespecified tolerances for all $v = 1, \ldots, p$, declare convergence and exit to step 7.

5. If the number of iterations k reaches a prespecified limit K, declare non-convergence, issue corresponding warnings, and exit to step 7.

6. Otherwise, return to step 2. (That is, the achieved accuracy of the control targets insufficient, and more work is needed.)

7. Return the weights $w_j^{k,p}$ at the final stage as the calibrated weights.

In practice, control totals are usually expressed as population counts or proportions in categories of discrete variables (such as gender, race/ethnicity or education level groups). The control variables are then 0/1 indicators representing the particular groups or their low-level interactions. Effectively, the algorithm implements post-stratification adjustment (4) treating each control variable as the post-stratification variable, and cycling over these variables within each iteration. In terms of multivariate optimization, this algorithm proceeds by optimizing over a each margin in sequence. While it is very simple and very explicit in terms of the algebra involved, it is also much slower compared to Newton-Raphson-based methods.

Deming and Stephan (1940) stated that the algorithm minimizes the quadratic discrepancy

$$\sum_{j\in\mathcal{S}} \frac{(w_{1j} - w_{3j})^2}{w_{1j}}$$

under the calibration constraints (5). However, the quadratic problem can be solved explicitly to produce linear calibrated weights which lead to estimates identical to the generalized regression (GREG) estimates (Deville and Särndal 1992, Case 1). The raking algorithm instead solves the optimization problem with objective function that can be expressed as (Deville and Särndal 1992, Case 2)

$$\sum_{j\in\mathcal{S}} w_{3j} \ln(w_{3j}/w_{1j}) - w_{3j} + w_{1j} \tag{6}$$

1.5 Variance estimation

Besides the primary challenge of finding a good set of weights (which is generally solved through iterative optimization), an additional methodological challenge with calibrated estimators is variance estimation. If variables $x_1, \ldots, x_p$ were used for weight calibration, then the asymptotic variance of the calibrated estimator of the survey variable y is

$$\mathbb{V}\big\{t_m[y]\big\} = \sum_{k,l\in\mathcal{U}} (\pi_{kl} - \pi_k\pi_l) \frac{Y_k - \mathbf{X}_k'\mathbf{B}}{\pi_k} \frac{Y_l - \mathbf{X}_l'\mathbf{B}}{\pi_l}, m = 2, 3, \tag{7}$$

where $\mathbf{B}$ is the vector of coefficients from the census regression,

$$\mathbf{B} = \big(\sum_{i\in\mathcal{U}} \mathbf{X}_i\mathbf{X}_i'\big)^{-1} \sum_{i\in\mathcal{U}} \mathbf{X}_i Y_i, \tag{8}$$

This variance can be estimated with

$$v\big\{t_m[y]\big\} = \sum_{k,l\in\mathcal{S}} \frac{\pi_{kl}-\pi_k\pi_l}{\pi_{kl}} \frac{y_k-\mathbf{x}_k'\mathbf{b}}{\pi_k} \frac{y_l-\mathbf{x}_l'\mathbf{b}}{\pi_l}, m = 2,3, \tag{9}$$

where the regression coefficients now solve the sample regression problem:

$$\mathbf{b} = \big(\sum_{j\in\mathcal{S}} w_j\mathbf{x}_j\mathbf{x}_j'\big)^{-1} \sum_{j\in\mathcal{S}} w_j\mathbf{x}_j y_j \tag{10}$$

In regression (10), either the probability weights w_{1j} or the calibrated weights w_{2j}, w_{3j} can be used. The estimator (9) is difficult to use in practice, especially with the publicly released versions of the data. First, this estimator utilizes the original design weights $w_{1i} = \pi_i^{-1}$. Hence, the publicly released data set must include both the calibrated weights and the design weights, which may create confusion. Second, the end user of the data must be given the set of the control variables, which may not be possible if confidential variables were used in calibration. Third, this estimator is not necessarily implemented in survey packages (a third party package `calibest` implements (9) in Stata). Finally, the estimator requires second order selection probabilities, which are rarely computed in practice. The latter is a very general issue with the Horvitz-Thompson estimator (3), as well. Its variance is

$$\mathbb{V}\big\{t_1[y]\big\} = \sum_{k,l\in\mathcal{U}} (\pi_{kl}-\pi_k\pi_l)\frac{Y_k}{\pi_k}\frac{Y_l}{\pi_l} \tag{11}$$

While the second order selection probabilities are nominally required for this estimator, in practice simplifications are taken, e.g., to approximate the actual design as the stratified two-stage sample in which the primary sampling units are drawn with replacement (as is done, for example, in `nhanes2` [SVY] manual example dataset). Due to these complications, variance estimation with calibrated data usually proceeds along the lines of replicate variance estimation methods (Shao 1996; Kolenikov 2010).

When the control totals are obtained from another survey, the sampling variability of the latter should be taken into account (Dever and Valliant 2010). For instance, to calibrate population surveys conducted in the USA, the American Community Survey (U.S. Census Bureau 2009) is often used for demographic variables, and the National Health Interview Survey (Botman et al. 2000) for phone usage. These very large scale surveys have sample sizes in the hundreds of thousands. For typical surveys with sample sizes in hundreds to low thousands, the impact on the standard errors is in the second or the third decimal point, and is usually ignored.

1.6 Pros and cons of weight calibration

By comparing expressions (7) and (11), we can identify the source of efficiency gains associated with weight calibration. If the survey variable y is associated with calibration variables $x_1, \ldots, x_p$, in the sense of having a non-trivial R^2 in the census regression (8), then the calibrated estimator is (asymptotically) more efficient than the direct Horvitz-Thompson estimator by a factor of $1 - R^2$.

Weight calibration can also reduce non-response and coverage errors (Chang and Kott 2008; Kott 2006; Lundström and Särndal 1999), which feature prominently as some of the most important issues that survey community currently faces (Groves 2006). However, for weight calibration to be successful in reducing the non-response bias, the control variables need to correlated with the response propensity and/or the outcome variables (Bethlehem 2002; Judkins et al. 2007).

Weight calibration comes with some costs, too. From an analytic perspective, manipulating the weights almost inevitably leads to increase in their variation, which in turn leads to increases in the design effects. For the unequal probability sample without stratification or clustering, Korn and Graubard (1999) show that the design effect is

$$\mathrm{DEFF}_w = \frac{\sum_{j\in\mathcal{S}} w_j^2}{\left(\sum_{j\in\mathcal{S}} w_j\right)^2} = 1 + \mathrm{CV}_w^2 \tag{12}$$

where CV_w is the coefficient of variation of the weights (a simple standard deviation divided by the simple mean). In practice, I have encountered increases of this coefficient of variation between 20% and 100% on the relative scale, or between 0.2 and 1.5 on the absolute scale, for design effects varying between 1 and 2 in the typical public opinion surveys. From a practical perspective, weight calibration requires additional time expenses by statisticians preparing the data, which increases the cost of the survey and the time elapsing between the end of the data collection period and delivery of the final data set. Additionally, as noted in section 1.5, variance estimation with calibrated data tends to get complicated.

1.7 Weight trimming

As expression (12) shows, it is undesirable for a survey to have a large spread of weights (Théberge 2000). Otherwise, many survey estimates are unduly affected by the observations with large weights, while those with small weights make but minimal contributions. The impact of the observations with high weights will be exacerbated in the analysis of domains, where these observations will stand apart even more given the smaller sizes of domains. For these reasons, weights are often *trimmed*: the largest weights are reduced (say, all the weights greater than the largest allowable number are reduced to that number), and the smallest weights are increased, so that for all j, $L \le w_{3j} \le U$ for some absolute limits L and U. Alternatively, the relative change in weights can be constrained: for all j, $l \le w_{3j}/w_{1j} \le u$ for some ratio limits l and u. Weight trimming may introduce bias, so the amount of trimming needs to be seen as a trade-off between

an apparent efficiency improvement and latent bias (Elliott 2008).

With trimming, the modified algorithm implemented in `ipfraking` proceeds as follows. (See Section 2.1 for the syntax diagram, and in particular Section 2.2 for specification of the trimming options. If not specified otherwise, the default values are $U = u = +\infty$, $L = l = 0$.)

Algorithm 2: raking with simultaneous trimming

1. Initialize the outer cycle iteration counter $k \leftarrow 0$. Initialize the weights $w_j^{0,p} \leftarrow w_{1j}$. Set $D_0 = \infty$ (This is a notation introduced for consistency of notation in step 9.)

2. Increment the outer cycle iteration counter $k \leftarrow k+1$, update the weights $w_j^{k,0} \leftarrow w_j^{k-1,p}$.

3. Initialize the inner cycle over control variables: $v \leftarrow 1$

4. Update the weights using the v-th variable as the post-stratification variable:

$$w_j^{k,v} = \begin{cases} w_j^{k,v-1} \dfrac{T[X_v]}{\sum_{l \in \mathcal{S}} w_l^{k,v-1} x_{vl}}, & x_{vj} \neq 0 \\ w_j^{k,v-1}, & x_{vj} = 0 \end{cases}$$

5. If `trimfrequency` option is specified as `often`, perform weight trimming:

$$w_j^{k,v} \leftarrow \min\bigl(w_j^{k,v}, U, uw_j^{0,p}\bigr),$$

$$w_j^{k,v} \leftarrow \max\bigl(w_j^{k,v}, L, lw_j^{0,p}\bigr)$$

   That is, trim the weights that are greater than U in absolute terms and/or have increased by more than a factor of u from the initial weight; reduce such weights to the largest allowed value. Likewise, trim the weights that are less than L in absolute terms and/or have dropped by more than a factor of l from the initial weight; increase such weights to the smallest allowed value.

6. Increment the internal cycle counter $v \leftarrow v+1$

7. If $v \leq p$, cycle back to step 4. Otherwise the inner cycle over the control variables is completed; proceed to the next step.

8. If `trimfrequency` option is specified as `sometimes`, perform weight trimming:

$$w_j^{k,p} \leftarrow \min\bigl(w_j^{k,p}, U, uw_j^{0,p}\bigr),$$

$$w_j^{k,p} \leftarrow \max\bigl(w_j^{k,p}, L, lw_j^{0,p}\bigr)$$

9. If the largest change in weights

$$D_k = \max_{j\in\mathcal{S}} \Big| \frac{w_j^{k,p}}{w_j^{k-1,p}} - 1 \Big|$$

is less than or equal to tolerance δ_D (given in `tolerance` option), declare convergence of weights and go to step 11. If $D_k > D_{k-1}$, $k > 1$, the algorithm may be diverging; stop the outer cycle iterations, issue a non-convergence message and go to step 11. Otherwise (i.e., if $\delta_D < D_k < D_{k-1}$), move to the next step: there is some room for weight improvement.

10. If the number of the outer cycle iterations k reaches a prespecified limit K, stop the outer cycle iterations and issue a non-convergence message. Otherwise, cycle back to step 2.

11. If `trimfrequency` option is specified as `once`, perform weight trimming:

$$w_j^{k,p} \leftarrow \min\big(w_j^{k,p}, U, uw_j^{0,p}\big),$$

$$w_j^{k,p} \leftarrow \max\big(w_j^{k,p}, L, lw_j^{0,p}\big)$$

12. If discrepancies between the weighted totals $\sum_{j\in\mathcal{S}} w_j^{k,p} x_v$ and the target totals $T[X_v]$ are greater than prespecified tolerances δ_T (`ctrltolerance` option),

$$\Big| \frac{T[X_v] - \sum_{j\in\mathcal{S}} w_j^{k,p} x_v}{T[X_v] + 1} \Big| > \delta_T$$

for at least one $v = 1, \ldots, p$, issue a warning message (see Section 4).

13. Return the weights $w_j^{k,p}$ as calibrated weights and exit.

The algorithm may be exited for three possible reasons: reaching the maximum number of iterations (indicative of lack of convergence), finding that the changes in weights started diverging, or by reaching the state where the weights do not change from one iteration of post-stratification and possibly trimming to the next. Even in the latter case, convergence of the weights does not imply convergence of the weighted totals to their targets. Hence, there are qualitatively three possible outcomes of running `ipfraking`:

1. The weights have converged as checked in step 9, and the weighted control totals are within tolerances from their targets, as checked in step 12. The raked weights are most likely safe to use, although additional quality control checks, including computation of the DEFF (12), histograms and tabulations with the main variables of interest would be recommended.

2. The weights have converged as checked in step 9, but the weighted control totals are not sufficiently close to their targets, as checked in step 12. The raked weights should be reviewed, and may not be safe to use. This often happens when the trimming options are too aggressive, when the data and the control totals are incompatible, or when the control totals themselves are poor (e.g., the matrices sum to different values, of which `ipfraking` will issue an error message).

3. The weights have not converged after the pre-specified number of iterations, or started diverging. Again, the resulting weights are likely to be unsatisfactory. The number of iterations should be increased, the tolerances should be decreased, or `nodivergence` option can be specified if optimization aborted because the weight convergence criteria went up.

1.8 Other weight calibration programs

There has been a number of packages with similar functionality that are circulating in Stata community. Nick Winter's `survwgt` (Winter 2002) is the most robust and versatile of these, and its `survwgt rake` subcommand implements the same raking algorithm as the basic algorithm of `ipfraking`. The functionality of `survwgt` also includes valuable capabilities to create the balanced repeated replication (BRR) and jackknife replicate weights (Kolenikov 2010), as well as non-response cell adjustments. One feature that `survwgt` does not have is trimming.

A more recent raking package is `ipfweight` (Bergmann 2011). It implements the basic raking, and provides relative trimming similar to `trimfrequency(often)`.

Another user-contributed Stata package, `maxentropy` (Wittenberg 2010), implements Case 4 of Deville and Särndal (1992) using Newton-Raphson optimization with analytical second derivatives, and is much faster than `ipfraking` described here.

Compared to these packages, `ipfraking` was developed to work in the weight production environment of a survey company. To be an effective tool, the weight calibration procedure should not only produce the correct figures, but also provide extensive diagnostics and robustness checks that can potentially be analyzed later in semi-automated fashion, and be robust and fail softly with incorrectly specified inputs. For instance, all of the above packages rely on the user to match the variables and their targets, and some are relatively fragile numerically when the initial weights generate totals that are far off their targets. In `ipfraking`, as you will shortly see, the match between variables and their targets is implemented internally through metadata (variable names and values) stored in Stata target matrices, as the necessary variables and their categories are picked up by `ipfraking` from the targets. Thus the number of necessary inputs, and hence the likelihood of the user error (through incorrect ordering of variables and their categories) is reduced. The targets, in turn, can be easily obtained from the calibration data sets such as American Community Survey. Also, `ipfraking` defines convergence in terms of values of weights rather than the target discrepancies as done in other packages. It thus allows the possibility of the raking procedure converging in computational sense (weights stop changing from one iteration to the next), and then diagnoses the

statistical convergence, i.e., whether the targets are being satisfied.

Besides the internal convergence diagnostics, the weights produced by `ipfraking` were compared to those produced by `survwgt` and `ipfweight` as a certification step (Gould 2001), and were found to be identical within numerical accuracy.

2 Package description

2.1 Syntax

`ipfraking` [*if*] [*in*] [*weight*] `, ctotal(`*matname* [*matname ...*]`)` [
`generate(`*newvarname*`) replace double iterate(#) tolerance(#)`
`ctrltolerance(#) trace nodivergence trimhiabs(#) trimhirel(#)`
`trimloabs(#) trimlorel(#) trimfrequency(once|sometimes|often) double`
`meta nograph`]

Note that the weight statement `[pw=`*varname*`]` is required, and must contain the initial weights.

2.2 Options

Required options

`ctotal(`*matname* [*matname ...*]`)` supplies the names of the matrices that contain the control totals, as well as meta-data about the variables to be used in calibration.

❑ **Technical note**

The row and column names of the control total matrices (see [P] **matrix rownames**) should be formatted as follows.

- `rownames`: the name of the control variable
- `colnames`: the values the control variables takes
- `coleq`: the name of the variable for which total is computed; typically it is identically equal to 1.

See examples in Section 3.

❑

`generate(`*newvarname*`)` contains the name of the new variable to contain the raked weights.

`replace` indicates that the weight variable supplied in the `[pw=`*varname*`]` expression should be overwritten with the new weights.

One and only one of `generate()` or `replace` must be specified.

Options to control convergence

❑ **Technical note**

Convergence in `ipfraking` is defined in terms of the maximum relative change in weights:

$$D_k = \max_{j \in S} \frac{|w_j^{k,p} - w_j^{k-1,p}|}{w_j^{k-1,p}} \tag{13}$$

When D_k is small, $D_k < \delta_D$, it means that the weights stop changing between iterations, i.e., the algorithm came to its steady state. On the other hand, if $D_k > D_{k-1}$, it means that the algorithm may start diverging, at which point it might be reasonable to terminate it. See step 9 of Algorithm 2 in Section 1.7.

Once the algorithm terminates, it also checks whether the control totals are satisfied. Specifically, for each of the control total matrices $M_1, \ldots, M_p$, the relative difference vs. the corresponding weighted sample totals $\hat{M}_1, \ldots, \hat{M}_p$ is computed:

$$m_c = \text{mreldif}(\hat{M}_c, M_c) \tag{14}$$

where the maximum relative difference of two matrices

$$\text{mreldif}(A, B) = \max_{ij} \frac{|a_{ij} - b_{ij}|}{1 + |b_{ij}|}$$

as defined in [D] **functions**. A control relation is satisfied if $m_c < \delta_T$; otherwise, a warning is issued. See step 12 of Algorithm 2 in Section 1.7.

Iterations continue until either $k = K$, a specified number of iterations; $D_k < \delta_D$; or $D_k > D_{k-1}$.

❑

`tolerance(#)` defines the D_k-convergence criterion, i.e., δ_D. The default is $\delta_D = 10^{-6}$.

`iterate(#)` specifies the maximum number of iterations K. The default is $K = 2000$.

`nodivergence` overrides the check that $D_k > D_{k-1}$, i.e., ignores this termination condition.

`ctrltolerance(#)` defines the criterion δ_T to assess the accuracy of the control totals. It does not impact iterations or convergence criteria; it only serves as the final quality control check after the algorithm terminates as defined above. The default value is $\delta_T = 10^{-6}$.

`trace` requests a trace plot to be added. See Section 3.4.

Trimming options

`trimhiabs(#)` specifies the upper bound U on the greatest value of the raked weights. The weights that exceed this value will be trimmed down, so that $w_{3j} \leq U$ for every $j \in \mathcal{S}$.

`trimhirel(#)` specifies the upper bound u on the adjustment factor over the baseline weight. The weights that exceed the baseline times this value will be trimmed down, so that $w_{3j} \leq uw_{1j}$ for every $j \in \mathcal{S}$.

`trimloabs(#)` specifies the lower bound L on the smallest value of the raked weights. The weights that are smaller than this value will be increased, so that $w_{3j} \geq L$ for every $j \in \mathcal{S}$.

`trimlorel(#)` specifies the lower bound l on the adjustment factor over the baseline weight. The weights that are smaller than the baseline times this value will be increased, so that $w_{3j} \geq lw_{1j}$ for every $j \in \mathcal{S}$.

`trimfreqency(`*keyword*`)` specifies when the trimming operations are to be performed. The following keywords are recognized:

`often` means that trimming will be performed after each marginal adjustment, i.e., within each iteration of the inner cycle inside Step 5 of Algorithm 2.

`sometimes` means that trimming will be performed after a full set of variables has been used for post-stratification, i.e., at the end of each outer cycle iteration at step 8 of Algorithm 2. This is the default behavior if any of the numeric trimming options above are specified.

`once` means that trimming will be performed after the outer loop converges at step 11 of Algorithm 2.

The numeric trimming options `trimhiabs(#)`, `trimhirel(#)`, `trimloabs(#)`, `trimlorel(#)` can be specified in any combination, or entirely omitted to produce untrimmed weights. By default, there is no trimming. See Section 3.3 for examples.

Miscellaneous options

`double` specifies that the new variable named in `generate()` option should be generated as double type. See [D] **data types**.

`meta` puts the name(s) of the control vectors and the achieved control accuracies m_c as characteristics stored with the variable specified in `generate()` option. See Section 3.5.

`nograph` omits the histogram of the calibrated weights, which can be used to speed up `ipfraking` once the diagnostics on the weights are completed (e.g., in replicate weight production).

2.3 Utility programs

Besides the main weight calibration program, `ipfraking` package provides two additional utility programs to create and manipulate `ipfraking`-compatible matrices.

`mat2do` *matrix_name* `using` *do_file_name* `,` [`replace append list type`]

`mat2do` stores the values and the attributes (row and column names) of a Stata matrix as a do-file. By running this do-file, the matrix can be fully reproduced. The names of the matrix and the do-file are required.

`replace` overwrites the existing file.

`append` adds the code the the existing do-file.

`list` adds `matrix list` command to the end of the do-file, so that when the *do_file_name* is executed, the listing is provided for verification.

`type` lists the matrix and the resulting do-file.

`xls2row` *matrix_name* `using` *filename* `, cellrange([`*start*`]:[`*end*`]) sheet(`*name*`)` `over(`*varname*`)` [`scale(#)`]

The utility program `xls2row` reads the calibration totals from the specified Excel file and stores them in the matrix *matrix_name*. The name of the Excel file, the range of cells and the name of the sheet to take the values from are required, and specified in the same way as in `import excel` (see [D] **import excel**). Mathematically speaking, `xls2row` performs a vec-transformation of the matrix read from an Excel sheet, i.e., stores the result by columns. The `matrix coleq` of the resulting matrix is the convention name `_one`.

To optimize the performance, note that `xls2row` relies on `preserve` as an intermediate step. It is thus advisable to run `xls2row` upfront before loading potentially large data sets that would otherwise be written to disk and restored back a number of times.

`cellrange()` specifies the range of cells in an Excel file, e.g., B2:D15.

`sheet()` specifies the name of the sheet in Excel file to take values from.

`over(`*varname*`)` is the variable corresponding to the control total being imported from Excel. The columns of the resulting row vector *matrix_name* will the labeled with the values of *varname*(i.e., as the `matrix colname` of the matrix *matrix_name*), and *varname* itself will appear as the `matrix rowname` of the matrix *matrix_name*. If the number of categories of *varname* does not match the number of non-missing imported values, an error message will be issued, and the target matrix will not be created.

`scale(#)` optionally scales the entries of the resulting row vector so that they sum to the specified value.

3 Examples

3.1 Basic syntax and input requirements

In this very simple example, I shall demonstrate the basic mechanics of `ipfraking`, its input requirements and output. These examples are intended to only demonstrate the syntax and the output of `ipfraking`, and may or may not provide substantively meaningful results.

▷ Example 1

We shall work with the standard example of `svy` data, an excerpt from the NHANES II data set available from Stata Corp. website. We shall introduce some small changes to the data so that `ipfraking` will have some work to do.

```
. webuse nhanes2, clear
. generate byte _one = 1
.
. svy : total _one , over( sex, nolabel )
(running total on estimation sample)
Survey: Total estimation
Number of strata =      31       Number of obs    =      10351
Number of PSUs   =      62       Population size  =  117157513
                                  Design df        =         31
            1: sex = 1
            2: sex = 2
```

Over	Total	Linearized Std. Err.	[95% Conf.	Interval]
_one				
1	5.62e+07	1377465	5.34e+07	5.90e+07
2	6.10e+07	1396159	5.82e+07	6.38e+07

```
. matrix NHANES2_sex = e(b)
. matrix rownames NHANES2_sex = sex
.
. svy : total _one , over( race, nolabel )
(running total on estimation sample)
Survey: Total estimation
Number of strata =      31       Number of obs    =      10351
Number of PSUs   =      62       Population size  =  117157513
                                  Design df        =         31
            1: race = 1
            2: race = 2
            3: race = 3
```

(*Continued on next page*)

Over	Total	Linearized Std. Err.	[95% Conf.	Interval]
_one				
1	1.03e+08	2912042	9.71e+07	1.09e+08
2	1.12e+07	1458814	8213964	1.42e+07
3	2968728	1252160	414930.1	5522526

```
. matrix NHANES2_race = e(b)
. matrix rownames NHANES2_race = race
.
. matrix NHANES2_sex[1,1] = NHANES2_sex[1,1]*1.25
. matrix NHANES2_race[1,1] = NHANES2_race[1,1]*1.4
.
```

Let us now look at the matrices that will serve as an input to the raking procedure.

```
. matrix list NHANES2_sex, f(%12.0g)
NHANES2_sex[1,2]
         _one:     _one:
            1         2
sex  70199350  60998033
. matrix list NHANES2_race, f(%12.0g)
NHANES2_race[1,3]
             _one:        _one:        _one:
                1            2            3
race  144199368.6     11189236      2968728
```

These input matrices are organized as follows. Input matrices always have a single row, just as estimation results `e(b)` do. The column names follow the naming conventions of `e(b)`, namely, the name of the variable for which the total is being computed (here, `_one`) and the numeric categories of the variable that was used in the `over` option (here, `sex`, with values 1 for males and 2 for females; and `race`, with values 1 for whites, 2 for blacks, and 3 for other). These values must be in an increasing order. Since that variable is not stored in the e(b) per se, it needs to be added to this matrix, which is done in the form of the row name. The entries of the matrix are the totals that the weights in the categories of the control variables need to sum up to. In this example, they are scaled to be the population totals. Alternatively, these can be made to sum up to the sample size, as is done sometimes in public opinion research, or to 1, which is what `proportion` estimation command would produce.

The input requirements in terms of control totals are thus made as simple as possible. If a higher quality survey is available, all the survey statistician needs to do is to obtain the totals for the categories of the control variables using `svy: total ···, over( ···, nolabel )` and save the name of that variable along with the matrix. Note that the `total` is computed with `over( ..., nolabel)` suboption to suppress the otherwise informative labeling of the categories; `ipfraking` expects the numeric values of the categories as column names (see [P] **matrix rownames**). The name of the matrix itself is immaterial, but it is a good programming practice to have informative names

(McConnell 2004). Thus the names of the matrices in the examples generally follow the convention *data_source_variable*.

We are now ready to run `ipfraking` and see what it produces.

```
. ipfraking [pw=finalwgt], ctotal( NHANES2_sex NHANES2_race ) gen( rakedwgt1 )
Warning: the totals of the control matrices are different:
   Target 1 (NHANES2_sex) total              =            131197383
   Target 2 (NHANES2_race) total             =          158357332.6
 Iteration 1, max rel difference of raked weights = .56227988
 Iteration 2, max rel difference of raked weights = .00073288
 Iteration 3, max rel difference of raked weights = 2.356e-07
Warning: the controls NHANES2_sex did not match
   Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	15299	10274	1914	90831	.6716
Adjust factor	1.3490		0.8846	1.5614	

In this simple case with just two control variables and the control totals that are not very different from the existing sample totals, the procedure converged very quickly in three iterations. A diagnostic message was produced upfront by `ipfraking` informing about apparent differences in total population counts as obtained from the different control total matrices. As a result, the control totals for the variable that was adjusted first (`sex`) could not match the required control totals even after the weights converged in the sense of differing little between iterations. Both of these warnings are only produced when problems are encountered.

The summary table is always produced, and shows some relevant characteristics of the original weights w_{1j}, the raked weights w_{3j}, and the raking ratios w_{3j}/w_{1j}. As expected, the coefficient of variation went up from 0.645 to 0.672.

The graphic output produced by `ipfraking` is shown on Figure 1. Generally, we would want to inspect these graphs to see if there any unexpected patterns, such as highly outlying values, gaps in the distribution (here, there are only six distinct values of the adjustment factor corresponding to the 2×3 combinations of the control variables) or concentration near the limits of the weight range (as is typical for trimmed weights, see below in section 3.3). Also, these graphs may inform later trimming decisions: the trimming limits can be chosen to conform to the breaks in the distributions of the untrimmed raked weights.

◁

3.2 Preparing control matrices from scratch

In many situations, the control totals will be obtained from outside of Stata, and need to be prepared to work with `ipfraking`.

▷ **Example 2**

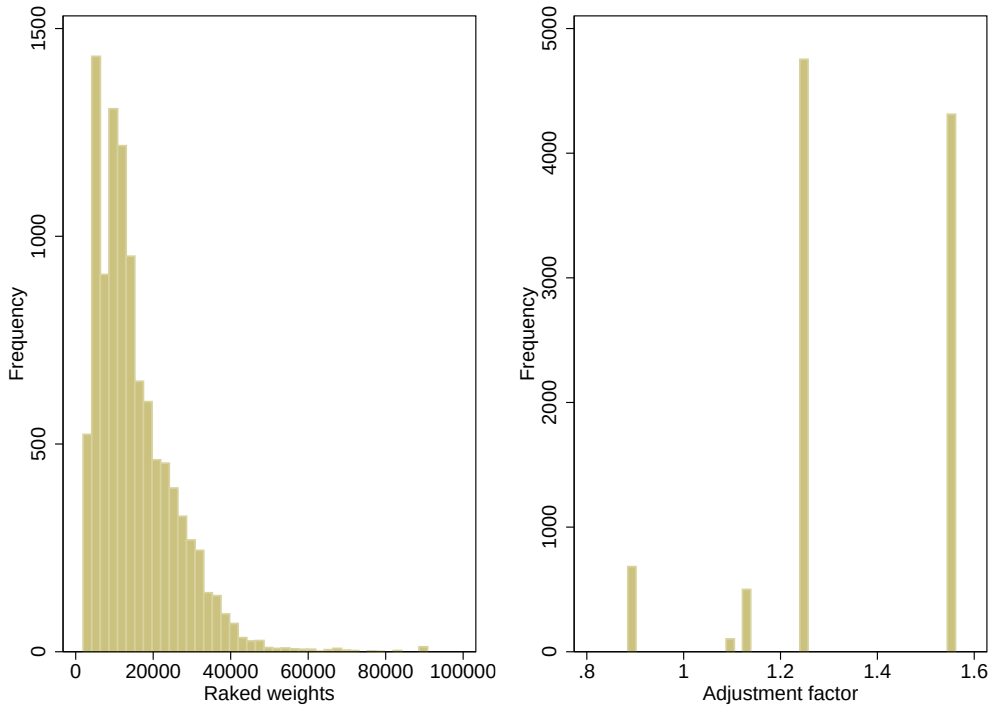

Figure 1: Histograms of the raked weights and calibration ratios, Example 1.

Suppose I wanted to calibrate the NHANES II data set to the latest control totals available from the US Census Bureau website. Using the tables S0101 from the 2011 American Community Survey 1-year estimates and NST-EST2011 from the US Census Bureau population projections, the latest available at the time of writing this paper, the figures displayed in Table 1 can be obtained.

Thus, we have information in the two-way age by sex table, as well as two additional margins. We shall need an additional sex-by-age group variable, and we shall try to make its values somewhat informative (e.g., the value `12` of the variable `sex_age` means the first group of sex and the second group of age):

```
. generate byte age_grp = 1 + (age>=40) + (age>=60) if !mi(age)
. generate sex_age = sex*10 + age_grp
```

With that, the matrices will have to be defined explicitly, and their labels need to be hand-coded, too (see [P] **matrix rownames**). Note that the US Census Bureau 2011 projections relate to the total population, while the target population of the study is the population age 20+. Assuming that the age structure is the same across regions and

Table 1: Control totals for the 2011 US population.

Group	Population
ACS 2011 1-year estimates, Table S0101	
Male, total	153,267,860
Ages 20–39	27.4%
Ages 40–59	27.5%
Ages 60+	17.3%
Female, total	158,324,057
Ages 20–39	26.0%
Ages 40–59	27.6%
Ages 60+	20.7%
US Census Bureau 2011 projections, Table NST-EST2011-01	
Northeast	55,521,598
Midwest	67,158,835
South	116,046,736
West	72,864,748
US Census Bureau 2011 projections, Table NC-EST2011-03	
White	243,470,497
Black	40,750,746
Other	27,370,674
Total	311,591,917

races, the control totals for region and race need to be rescaled to the adult population to avoid the warning messages. (More accurate figures can be obtained from ACS microdata which can be downloaded from the U.S. Census Bureau website.)

```
. matrix ACS2011_sex_age = ( ///
>     153267860*0.274, 153267860*0.275, 153267860*0.173, /// males
>     158324057*0.260, 158324057*0.276, 158324057*0.207  /// females
> )
. matrix colnames ACS2011_sex_age = 11 12 13 21 22 23
. matrix coleq    ACS2011_sex_age = _one
. matrix rownames ACS2011_sex_age = sex_age
. scalar ACS2011_total_pop = 311591917
. matrix ACS2011_adult_pop = ACS2011_sex_age * J(colsof(ACS2011_sex_age),1,1)
. matrix Census2011_region = ///
>     (55521598, 67158835, 116046736, 72864748 )
. matrix Census2011_region = Census2011_region * ACS2011_adult_pop / ACS2011_to
> tal_pop
. matrix colnames Census2011_region = 1 2 3 4
. matrix coleq    Census2011_region = _one
. matrix rownames Census2011_region = region
. matrix Census2011_race = ///
>     (243470497, 40750746, 27370674 )
```

(*Continued on next page*)

```
. matrix Census2011_race = Census2011_race * ACS2011_adult_pop / ACS2011_total_
> pop
. matrix colnames Census2011_race = 1 2 3
. matrix coleq    Census2011_race = _one
. matrix rownames Census2011_race = race
```

Let us check the matrix entries and labels once again before producing the weights. Note that the values of the control variable categories are given in an increasing order.

```
. matrix list ACS2011_sex_age, f(%10.0g)
ACS2011_sex_age[1,6]
             _one:     _one:     _one:     _one:     _one:     _one:
               11        12        13        21        22        23
sex_age  41995394  42148662  26515340  41164255  43697440  32773080
. matrix list Census2011_region, f(%10.0g)
Census2011_region[1,4]
            _one:     _one:     _one:     _one:
              1         2         3         4
region  40679030  49205289  85024007  53385843
. matrix list Census2011_race, f(%11.0g)
Census2011_race[1,3]
            _one:       _one:       _one:
              1           2           3
race   178383622  29856864.7  20053682.2
```

As the labels appear to be in place, let us run `ipfraking`:

```
. ipfraking [pw=finalwgt], gen( rakedwgt2 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race )
 Iteration 1, max rel difference of raked weights = 14.95826
 Iteration 2, max rel difference of raked weights = .19495004
 Iteration 3, max rel difference of raked weights = .02204455
 Iteration 4, max rel difference of raked weights = .00315355
 Iteration 5, max rel difference of raked weights = .00043857
 Iteration 6, max rel difference of raked weights = .00006061
 Iteration 7, max rel difference of raked weights = 8.365e-06
 Iteration 8, max rel difference of raked weights = 1.154e-06
 Iteration 9, max rel difference of raked weights = 1.593e-07
   Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	22055	19227	4050	338675	.8717
Adjust factor	2.1464		0.9264	18.3694	

The diagnostic plots for these weights are given in Figure 2. They do appear to have some outlying cases (which are not very clearly seen on these plots as they are single count observations with outlying weights), and we shall address them in the next section with trimming.

◁

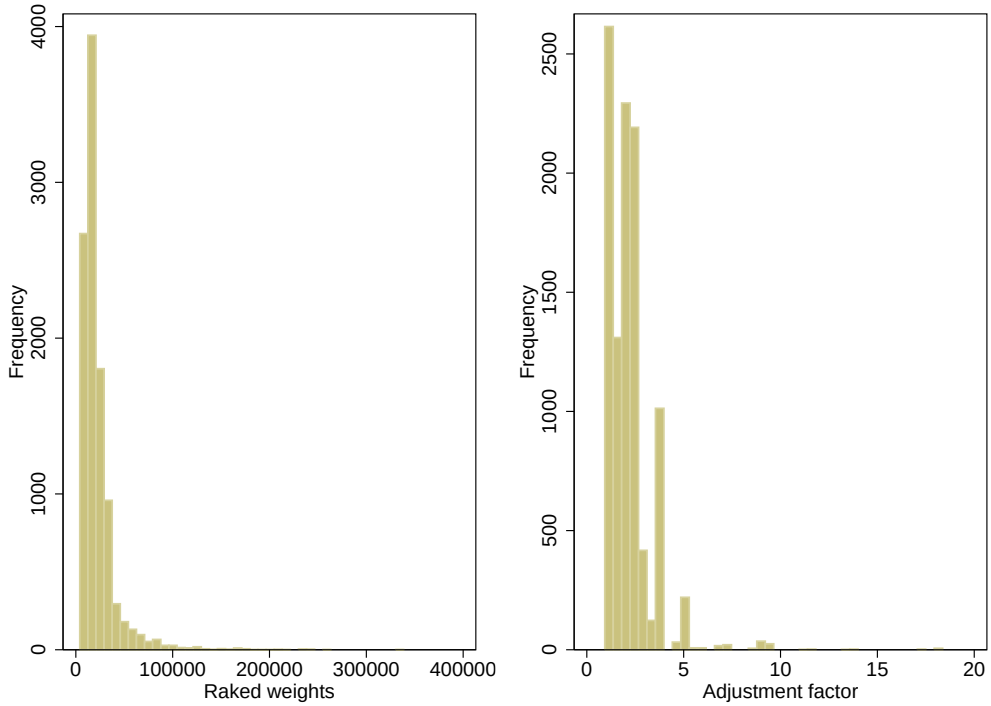

Figure 2: Histograms of the raked weights and calibration ratios, Example 2.

3.3 Trimming options

As discussed in Section 1.7 above, if variability of the weights becomes excessive, the weights can be trimmed by restricting the extremes. Using `ipfraking` options, upper and/or lower limits can be defined for either the absolute values of the weights or the relative changes from the base weights. The frequency of the trimming operations can also be controlled. Trimming can be applied once to the final data (`trimfreq(once)`) at step 11 of Algorithm 2. Alternatively, trimming can be applied after every full cycle over variables at step 8 of Algorithm 2. Finally, trimming can be applied after each sub-iteration at step 5 of the algorithm.

▷ Example 3

Inspecting the histograms on Figure 2, it appears reasonable to restrict the upper tail of the raked weights. A more detailed investigation of the histogram reveals a somewhat greater concentration of the raked weights around the value of 160,000, and sparse bars beyond 200,000. This latter number will be used as the top cut-off point for trimming, and is provided as an input to `ipfraking` via option `trimhiabs`. Also,

I specified the absolute lower bound of 2,000, which is the minimum of the original weights, but, as the output in the previous example suggested, the calibrated weights tend to run above 4,000, so specifying the lower limit as `trimloabs(2000)` may not really affect the calibration procedure.

```
. ipfraking [pw=finalwgt], gen( rakedwgt3 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs( 200000 ) trimloabs( 2000 )

 Iteration 1, max rel difference of raked weights = 14.95826
 Iteration 2, max rel difference of raked weights = .21474256
 Iteration 3, max rel difference of raked weights = .02754514
 Iteration 4, max rel difference of raked weights = .00511347
 Iteration 5, max rel difference of raked weights = .00095888
 Iteration 6, max rel difference of raked weights = .00018036
 Iteration 7, max rel difference of raked weights = .00003391
 Iteration 8, max rel difference of raked weights = 6.377e-06
 Iteration 9, max rel difference of raked weights = 1.199e-06
 Iteration 10, max rel difference of raked weights = 2.254e-07

   Summary of the weight changes

               |      Mean    Std. dev.       Min        Max        CV
---------------+-----------------------------------------------------
Orig weights   |     11318        7304        2000      79634    .6453
Raked weights  |     22055       18908        4033     200000    .8573
Adjust factor  |    2.1486                  0.9220    18.9828
```

The resulting coefficient of variation of weights, 0.857, is slightly better than that with unrestricted range of weights, 0.872. The summary also shows that the weights were capped at 200,000, as requested.

Setting the absolute limits on the range of the raked weights is often very subjective. A somewhat better plan might be to set limits in terms of the range of the adjustment factors, as shown in the next example. The relative change in the weights can be bounded with `trimlorel()` and `trimhirel()` options. I also demonstrate here how to use the results of `summarize` to feed into `ipfraking`. While ensuring that accurate numbers are being carried over in the context of the code, the approach is fragile for interactive work: simply running the single line with the sole `ipfraking` command that refers to the `r()` return values may break down if `summarize` was not the immediately preceding command.

```
. sum finalwgt

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    finalwgt |     10351    11318.47     7304.04       2000      79634

. ipfraking [pw=finalwgt], gen( rakedwgt4 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs(`=2.5*r(max)´) trimloabs(`=r(min)´) trimhirel(6)
```

(*Continued on next page*)

```
 Iteration 1, max rel difference of raked weights = 5
 Iteration 2, max rel difference of raked weights = .25592859
 Iteration 3, max rel difference of raked weights = .0626759
 Iteration 4, max rel difference of raked weights = .0158786
 Iteration 5, max rel difference of raked weights = .00299304
 Iteration 6, max rel difference of raked weights = .00070812
 Iteration 7, max rel difference of raked weights = .00016401
 Iteration 8, max rel difference of raked weights = .00003734
 Iteration 9, max rel difference of raked weights = 8.434e-06
 Iteration 10, max rel difference of raked weights = 1.898e-06
 Iteration 11, max rel difference of raked weights = 4.265e-07
Warning: the controls ACS2011_sex_age did not match
Warning: the controls Census2011_region did not match
Warning: the controls Census2011_race did not match
   Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	21830	18115	4113	199085	.8298
Adjust factor	2.1323		0.8973	6.0000	

◁

Setting the trimming options too aggressively may lead to adverse consequences. First, it may bias the estimates, as discussed in Section 1.6. Second, as this example demonstrates, it can impede (statistical) convergence: the output contains multiple warnings about targets not being achieved within desired accuracy, while no problems were encountered without trimming.

3.4 Tracking convergence

Let us now look in more detail into the issue of trimming frequency, and demonstrate another diagnostic plot that can be produced by `ipfraking`.

▷ Example 4

We return to the first set of options of Example 3, and re-run the raking procedure.

```
. capture drop rakedwgt3
. ipfraking [pw=finalwgt], gen( rakedwgt3 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs(200000) trimloabs(2000) trimfreq(sometimes) trace
 Iteration 1, max rel difference of raked weights = 14.95826
  (output omitted)
 Iteration 10, max rel difference of raked weights = 2.254e-07
   Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	22055	18908	4033	200000	.8573
Adjust factor	2.1486		0.9220	18.9828	

The option `trace` requests that trace plots be added to the diagnostic plots, as shown on Figure 3. The trace plots are presented on the absolute scale and on the log scale. The exponentially declining discrepancy appears to be a general phenomenon. In other words, after the first few iterations, discrepancy between the currently weighted totals to the control totals roughly follows the rate of const $\times \alpha^k$ for some $\alpha < 1$, where k is the (outer cycle) iteration number. When convergence is very slow or the sample size is very large, this rule may be helpful in determining the number of iterations necessary to achieve the required accuracy, and hence the expected computing time. Zero cross-cells and collinearity between the control variables may make the convergence factor α close to 1 thus hampering convergence. This happens when the control variables have very similar meaning, such as age and grade of children: it is impossible to have children of age 8 in grade 10. Also, sets of interactions of categorical variables, such as interactions of age group and education along with age group and race, are guaranteed to produce zero cells in the cross-tabulation: it is impossible to have any observations in the cells defined say by (age under 40 interacted with higher education) on one margin against (age above 60 interacted with white race) on the other.

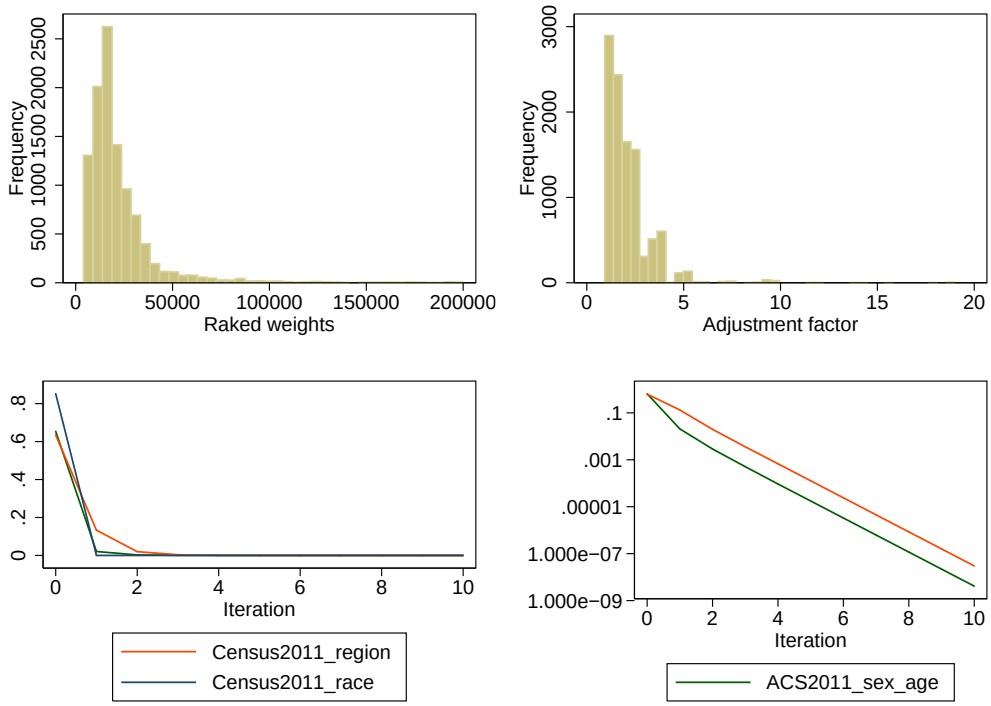

Figure 3: Diagnostic plots for Example 4.

While `trimfreq(sometimes)` is the default in presence of other trimming options,

the behavior can be changed with explicit specification of trimming frequency. Note that slightly different weights will be produced that way.

```
. ipfraking [pw=finalwgt], gen( rakedwgt5 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs(200000) trimloabs(2000) trimfreq(often) trace
 Iteration 1, max rel difference of raked weights = 14.95826
 Iteration 2, max rel difference of raked weights = .21613885
 Iteration 3, max rel difference of raked weights = .02673316
 Iteration 4, max rel difference of raked weights = .00480164
 Iteration 5, max rel difference of raked weights = .00086195
 Iteration 6, max rel difference of raked weights = .00015444
 Iteration 7, max rel difference of raked weights = .00002762
 Iteration 8, max rel difference of raked weights = 4.940e-06
 Iteration 9, max rel difference of raked weights = 8.832e-07
   Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	22055	18905	4033	200000	.8572
Adjust factor	2.1487		0.9220	18.9844	

```
. compare rakedwgt3 rakedwgt5
```

		difference		
	count	minimum	average	maximum
rakedwgt3<rakedwgt5	3638	-15.27963	-1.226753	-.0128687
rakedwgt3=rakedwgt5	4			
rakedwgt3>rakedwgt5	6709	.0011514	.6652557	2471.578
jointly defined	10351	-15.27963	.0000264	2471.578
total	10351			

In this example, trimming the weights after adjusting each of the margins led to fewer iterations. This may or may not translate to lower overall computing times as more computing is performed within each iteration.

◁

3.5 Metadata

The results of raking operations can be stored with the newly created weight variables for later review and reproduction of the results. Let us reproduce the example in the previous section adding all the metadata available:

▷ Example 5

```
. capture drop rakedwgt3
. ipfraking [pw=finalwgt], gen( rakedwgt3 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs(200000) trimloabs(2000) meta
```

(*Continued on next page*)

```
Iteration 1, max rel difference of raked weights = 14.95826
(output omitted)
Iteration 10, max rel difference of raked weights = 2.254e-07
  Summary of the weight changes
```

	Mean	Std. dev.	Min	Max	CV
Orig weights	11318	7304	2000	79634	.6453
Raked weights	22055	18908	4033	200000	.8573
Adjust factor	2.1486		0.9220	18.9828	

```
. char li rakedwgt3[]
  rakedwgt3[command]:          [pw=finalwgt], gen( rakedwgt3 ) ctotal( ACS2011_s
> ex_age Ce..
  rakedwgt3[trimloabs]:        trimloabs(2000)
  rakedwgt3[trimhiabs]:        trimhiabs(200000)
  rakedwgt3[trimfrequency]:    sometimes
  rakedwgt3[objfcn]:           2.25435521346e-07
  rakedwgt3[maxctrl]:          3.00266822363e-08
  rakedwgt3[converged]:        1
  rakedwgt3[Census2011_race]: 7.48567503861e-09
  rakedwgt3[Census2011_region]:
                               3.00266822363e-08
  rakedwgt3[ACS2011_sex_age]: 4.13778410340e-09
  rakedwgt3[note1]:            Raking controls used: ACS2011_sex_age Census2011_
> region Ce..
  rakedwgt3[note0]:            1
```

The following characteristics are stored with the newly created weight variable (see [P] **char**).

`command`	The full command as typed by the user
matrix name	The relative matrix difference from the corresponding control total, see [D] **functions**
`trimhiabs`, `trimloabs`, `trimhirel`, `trimlorel`, `trimfrequency`	Corresponding trimming options, if specified
`maxctrl`	the greatest `mreldif` between the targets and the achieved weighted totals
`objfcn`	the value of the relative weight change D_k (13) at exit
`converged`	whether `ipfraking` exited due to convergence (1) vs. due to an increase in the objective function or reaching the limit on the number of iterations (0)

Also, `ipfraking` stores the notes regarding the control matrices used, and which of the margins did not match the control totals, if any. See [D] **notes**.

3.6 Replicate weights

As discussed in Section 1.5, one of the greater challenges of weight calibration is ensuring that variance estimates take into account the greater precision achieved by adjusting the sample towards the fixed population quantities. As estimating the variances using

linearization is cumbersome, replicate variance estimation may be more attractive.

▷ Example 6

The simplest code for calibrated replicate weights is obtained by calling `ipfraking` from within `bsweights` (Kolenikov 2010) which can pass the name of a replicate weight variable to an arbitrary calibration routine. In this example, we shall use the same settings as in Section 3.2 and thus we shall have the calibrated weight `rakedwgt2` which was produced in that example as the main weight for which the bootstrap weights provide the measure of sampling variability.

```
. set seed 2013
. set rmsg on
r; t=0.00 14:50:44
. bsweights bsw , reps(310) n(-1) balanced dots ///
>     calibrate( ipfraking [pw=@], replace nograph meta ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) )
Balancing within strata:
.............................
Rescaling weights
..................................................    50
..................................................   100
..................................................   150
..................................................   200
..................................................   250
..................................................   300
..........
r; t=178.79 14:53:43
. forvalues k=1/310 {
  2.     _dots `k´ 0
  3.     assert `: char bsw`k´[converged]´ == 1
  4.     assert `: char bsw`k´[maxctrl]´ < 10*c(epsfloat)
  5. }
..................................................    50
..................................................   100
..................................................   150
..................................................   200
..................................................   250
..................................................   300
..........r; t=0.32 14:53:43
. set rmsg off
. svyset [pw=rakedwgt2], vce(bootstrap) bsrw( bsw* ) dof( 31 )
      pweight: rakedwgt2
          VCE: bootstrap
          MSE: off
    bsrweight: bsw1 bsw2 bsw3 bsw4 bsw5 bsw6 bsw7 bsw8 bsw9 bsw10 bsw11 bsw12
  (output omitted)
               bsw301 bsw302 bsw303 bsw304 bsw305 bsw306 bsw307 bsw308 bsw309
               bsw310
    Design df: 31
  Single unit: missing
     Strata 1: <one>
         SU 1: <observations>
        FPC 1: <zero>
```

The options of `bsweights` request 310 replicate weights (a multiple of 31 strata), resample one less PSU than available in a given stratum, and obtain the first-order balance within a stratum. With the 2 PSU/stratum design and these options, `bsweights` produces random half-samples of data. The at-character `@` is a placeholder for the name of the replicate weight variable. For explanations of these and other options of `bsweights`, see Kolenikov (2010). The procedure took about 3 minutes on a laptop computer, which can be considered moderately computationally intensive beyond interactive. A new option of `ipfraking` in the above code is `nograph` that suppresses the histograms. The additional asserts (Gould 2003) following the bootstrap weight generation demonstrate how the minimal quality assurance can be done on the bootstrap weights in the weight production workflow.

A more compact set of weights can be developed based on the existing BRR weights and a slightly more explicit code cycling over the weight variables:

```
. webuse nhanes2brr, clear
. svy : proportion highbp
(running proportion on estimation sample)
BRR replications (32)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
................................
Survey: Proportion estimation    Number of obs    =      10351
                                 Population size  =  117157513
                                 Replications     =         32
                                 Design df        =         31
```

	Proportion	BRR Std. Err.	[95% Conf.	Interval]
highbp				
0	.8941859	.0067023	.8805165	.9078553
1	.1058141	.0067023	.0921447	.1194835

```
. generate byte _one = 1
. generate byte age_grp = 1 + (age>=40) + (age>=60) if !mi(age)
. generate sex_age = sex*10 + age_grp
. ipfraking [pw=finalwgt], gen( rakedwgt2 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race )
 Iteration 1, max rel difference of raked weights = 14.95826
 Iteration 2, max rel difference of raked weights = .19495004
 Iteration 3, max rel difference of raked weights = .02204455
 Iteration 4, max rel difference of raked weights = .00315355
 Iteration 5, max rel difference of raked weights = .00043857
 Iteration 6, max rel difference of raked weights = .00006061
 Iteration 7, max rel difference of raked weights = 8.365e-06
 Iteration 8, max rel difference of raked weights = 1.154e-06
 Iteration 9, max rel difference of raked weights = 1.593e-07
```

(*Continued on next page*)

```
   Summary of the weight changes
                |     Mean    Std. dev.     Min        Max        CV
----------------+---------------------------------------------------
Orig weights    |    11318       7304      2000      79634    .6453
Raked weights   |    22055      19227      4050     338675    .8717
Adjust factor   |   2.1464               0.9264    18.3694
. forvalues k=1/32 {
  2.     quietly ipfraking [pw=brr_`k´], gen( brrc_`k´ ) nograph ///
>         ctotal( ACS2011_sex_age Census2011_region Census2011_race )
  3.     _dots `k´ 0
  4. }
................................
. svyset [pw=rakedwgt2], vce(brr) brrw( brrc* ) dof( 31 )

      pweight: rakedwgt2
          VCE: brr
          MSE: off
    brrweight: brrc_1 brrc_2 brrc_3 brrc_4 brrc_5 brrc_6 brrc_7 brrc_8 brrc_9
               brrc_10 brrc_11 brrc_12 brrc_13 brrc_14 brrc_15 brrc_16
               brrc_17 brrc_18 brrc_19 brrc_20 brrc_21 brrc_22 brrc_23
               brrc_24 brrc_25 brrc_26 brrc_27 brrc_28 brrc_29 brrc_30
               brrc_31 brrc_32
    Design df: 31
  Single unit: missing
     Strata 1: <one>
         SU 1: <observations>
        FPC 1: <zero>
. svy : proportion highbp
(running proportion on estimation sample)
BRR replications (32)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
................................
Survey: Proportion estimation    Number of obs    =      10351
                                 Population size  =  228294169
                                 Replications     =         32
                                 Design df        =         31

                |                   BRR
                |  Proportion   Std. Err.     [95% Conf. Interval]
----------------+---------------------------------------------------
highbp          |
              0 |    .8730544   .0081501      .8564323    .8896766
              1 |    .1269456   .0081501      .1103234    .1435677
```

The data can be analyzed with the standard `svy` prefix, and the standard errors will appropriately capture the efficiency gains from weight calibration. No additional action is required for the analyst or researcher.

◁

CAUTION: the input weights for the replicate weight calibration must be the probability replicate weights. The existing NHANES II weights have been adjusted for non-response and calibrated by the data provider, and are used above for demonstration purposes only.

4 Error messages and troubleshooting

4.1 Critical errors

The following critical errors will stop execution of `ipfraking`.

`pweight is required`

The `[pweight=...]` component of `ipfraking` syntax is required. Probability weights must be specified as inputs to `ipfraking`.

`ctotal() is required`

The `ctotal()` component of `ipfraking` syntax is required. Names of the matrices containing the control totals must be specified.

`one and only one of generate() or replace must be specified`

Either `generate()` option with the name of the new variable must be supplied to `ipfraking`, or `replace` to replace the variable specified in `[pw=...]` statement.

`raking procedure appears diverging`

The maximum relative difference of weights D_k has increased from the previous iteration. This may or may not indicate a problem. Re-run `ipfraking` with `nodivergence` option to override the warning.

`cannot process matrix` *matrix_name*

For whatever reason, `ipfraking` could not process this matrix. The matrix may not have been defined or the variables in this matrix cannot be found.

`variable` *varname* `corresponding to the control matrix` *matrix_name* `not found`

The variables contained in row or column names of this matrix cannot be found.

varname1 `and` *varname2* `variables are not compatible`

When running `total` *varname1*`, over(`*varname2*`)`, an error was encountered. One of the variables may be a string variable or have missing values resulting in empty estimation sample.

`categories of` *varname* `do not match in the control` *matrix_name* `and in the data (nolab option)`

There was a mismatch in the categories of *varname* found in the data and in the control matrix *matrix_name*. This could happen for any of the following reasons: (i) there were more categories in one than in the other; (ii) the entries are in the wrong order in the control matrix; (iii) the labels in the control matrix do not correspond to the category values in the data set; (iv) the control matrix was obtained via `total` *varname2*`, over(`*varname*`)`, but `nolabel` suboption of `over()` was omitted, and the labels of the control matrix may include some unexpected text. Tabulate *varname*

without labels, and compare the results to the matrix listing of the *matrix_name*.

`cannot compute controls for` *matrix_name* `over` *varname* `with the current weights`

This is a generic error message that something bad happened while `ipfraking` was computing the totals for the current set of weights. This error message should generally be very rare, but as computing the totals may be the slowest operation of the iterative optimization process, stopping `ipfraking` with a *Ctrl+Break* combination or the *Break* GUI button may produce this error message.

`trimhiabs|trimloabs|trimhirel|trimlorel must be a positive number`

One or more of the trimming options are given as a non-positive number or a non-number.

`trimhiabs must be greater than trimloabs`

`trimhirel must be greater than trimlorel`

The trimming parameters are illogical (the lower bound is greater than the upper bound). Respecify the values of the trimming parameters.

4.2 Other errors and warnings

The following warning messages may be produced by `ipfraking`. The program will continue running, but you must double-check the results for potential problems.

`the totals of the control matrices are different`

The sum of values of the control matrices are different. These sums will be listed for review. Convergence is still possible, but some of the control total checks are likely to fail.

`trimfrequency() option is specified without numeric settings; will be ignored`

The option `trimfrequency()` was specified without any numeric trimming options. There is no way to interpret this, and `ipfraking` will proceed without trimming.

`trimfrequency() option is specified incorrectly, assume default value (sometimes)`

Something other than `often`, `sometimes` or `once` was supplied in `trimfrequency`, and the default value is being used instead.

`raking procedure did not converge`

The maximum number of iterations was reached, but weights never met the convergence criteria (see step 9 of Algorithm 2 in Section 1.7). The user may want to increase the number of iterations or relax convergence criteria.

`the controls` *matrix_name* `did not match`

After convergence of weights was declared, `ipfraking` checked again the control totals, and found that the results differed from the target for one or more of the control total matrices. Any of the following can cause this: (i) the sum of entries of this particular matrix differs from the others; (ii) the trimming options are too restrictive, and do not allow the weights to adjust enough; (iii) the problem may not have a solution due to incompatible control totals or a bad sample.

`division by zero weighted total encountered with` *matrix_name* `control`

The weights for a category of the control variable summed to zero. `ipfraking` will skip calibration over this variable and proceed to the next one.

`# missing values of` *varname* `encountered; convergence will be impaired`

A control variable has missing values in the calibration sample. There is little way for `ipfraking` to figure out how to deal with the weights for the observations with missing values. The user would need either to restrict the sample to non-missing values of all control variables, to impute the missing values or to create a separate category for the missing values of a given control variable (which may lead to difficulties in defining valid population control totals for it).

Acknowledgements

The author is grateful to Ben Phillips, Andrew Burkey and Brady West, as well as the editor and an anonymous referee, who suggested additional functionality and provided helpful comments to improve the readability of this article. The opinions stated in this paper are of the author only, and do not represent the position of Abt SRBI.

5 References

Battaglia, M. P., D. Izrael, D. C. Hoaglin, and M. R. Frankel. 2009. Practical considerations in raking survey data. *Survey Practice*, http://surveypractice.wordpress.com/2009/06/29/raking-survey-data/.

Bergmann, M. 2011. IPFWEIGHT: Stata module to create adjustment weights for surveys. Statistical Software Components, Boston College Department of Economics. RePEc:boc:bocode:s457353.

Bethlehem, J. 2002. Weighting Nonresponse Adjustments Based on Auxiliary Information. In *Survey Nonresponse*, ed. R. M. Groves, D. A. Dillman, J. L. Eltinge, and R. J. A. Little, 375–288. New York: Wiley.

Binder, D. A., and G. R. Roberts. 2003. Design-based and Model-based Methods for Estimating Model Parameters. In *Analysis of Survey Data*, ed. R. L. Chambers and C. J. Skinner, chap. 3. New York: John Wiley & Sons.

Botman, S., T. Moore, C. Moriarity, and V. Parsons. 2000. Design and estimation for the National Health Interview Survey, 1995–2004. Technical Report 130, National Center for Health Statistics.

Chang, T., and P. S. Kott. 2008. Using calibration weighting to adjust for nonresponse under a plausible model. *Biometrika* 95(3): 555–571.

Deming, E. W., and F. F. Stephan. 1940. On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. *Annals of Mathematical Statistics* 11(4): 427–444.

Dever, J. A., and R. Valliant. 2010. A comparison of variance estimators for poststratification to estimated control totals. *Survey Methodology* 36(1): 45–56.

Deville, J. C., and C. E. Särndal. 1992. Calibration Estimators in Survey Sampling. *Journal of the American Statistical Association* 87(418): 376–382.

Deville, J. C., C. E. Särndal, and O. Sautory. 1993. Generalized Raking Procedures in Survey Sampling. *Journal of the American Statistical Association* 88(423): 1013–1020.

Elliott, M. R. 2008. Model Averaging Methods for Weight Trimming. *Journal of Official Statistics* 24(4): 517–540.

Gould, W. 2001. Statistical software certification. *Stata Journal* 1(1): 29–50.

———. 2003. Stata tip 3: How to be assertive. *Stata Journal* 3(4): 448–449.

Groves, R. M. 2006. Nonresponse Rates and Nonresponse Bias in Household Surveys. *Public Opinion Quarterly* 70(5): 646–675.

Groves, R. M., D. A. Dillman, J. L. Eltinge, and R. J. A. Little. 2001. *Survey Nonresponse*. Wiley Series in Survey Methodology, Wiley-Interscience.

Holt, D., and T. M. F. Smith. 1979. Post Stratification. *Journal of the Royal Statistical Society, Series A* 142(1): 33–46.

Horvitz, D. G., and D. J. Thompson. 1952. A Generalization of Sampling Without Replacement From a Finite Universe. *Journal of the American Statistical Association* 47(260): 663–685.

Judkins, D. R., D. Morganstein, P. Zador, A. Piesse, B. Barrett, and P. Mukhopadhyay. 2007. Variable selection and raking in propensity scoring. *Statistics in Medicine* 26(5): 1022–1033.

Kolenikov, S. 2010. Resampling inference with complex survey data. *The Stata Journal* 10: 165–199.

Korn, E. L., and B. I. Graubard. 1995. Analysis of Large Health Surveys: Accounting for the Sampling Design. *Journal of the Royal Statistical Society, Series A* 158(2): 263–295.

———. 1999. *Analysis of Health Surveys*. John Wiley and Sons.

Kott, P. S. 2006. Using Calibration WeightingtoAdjust for Nonresponse andCoverage Errors. *Survey Methodology* 32(2): 133–142.

———. 2009. Calibration Weighting: Combining Probability Samples and Linear Prediction Models. In *Sample Surveys: Inference and Analysis*, ed. D. Pfeffermann and C. R. Rao, vol. 29B of *Handbook of Statistics*, chap. 25. Oxford, UK: Elsevier.

Lundström, S., and C.-E. Särndal. 1999. Calibration as a Standard Method for Treatment of Nonresponse. *Journal of Official Statistics* 15(2): 305–327.

McConnell, S. 2004. *Code Complete: A Practical Handbook of Software Construction*. 2nd ed. Microsoft Press.

Pew Research Center. 2012. Assessing the Representativeness of Public Opinion Surveys. Technical report, Pew Research Center for People and Press. Available at http://www.people-press.org/files/legacy-pdf/Assessing the Representativeness of Public Opinion Surveys.pdf.

Pfeffermann, D. 1993. The role of sampling weights when modeling survey data. *International Statistical Review* 61: 317–337.

Särndal, C.-E. 2007. The calibration approach in survey theory and practice. *Survey Methodology* 33(2): 99–119.

Shao, J. 1996. Resampling Methods in Sample Surveys (with discussion). *Statistics* 27: 203–254.

Skinner, C. J. 1989. Domain Means, Regression and Multivariate Analysis. In *Analysis of Complex Surveys*, ed. C. J. Skinner, D. Holt, and T. M. Smith, chap. 3, 59–88. New York: Wiley.

Théberge, A. 2000. Calibration and restricted weights. *Survey Methodology* 26(1): 99–107.

Thompson, M. E. 1997. *Theory of Sample Surveys*, vol. 74 of *Monographs on Statistics and Applied Probability*. New York: Chapman & Hall/CRC.

U.S. Census Bureau. 2009. American Community Survey: Design and Methodology. Technical report, U.S. Census Bureau, U.S. Government Printing Office, Washington, DC.

Winter, N. 2002. SURVWGT: Stata module to create and manipulate survey weights. Statistical Software Components, Boston College Department of Economics. RePEc:boc:bocode:s427503.

Wittenberg, M. 2010. An introduction to maximum entropy and minimum cross-entropy estimation using Stata. *Stata Journal* 10(3): 315–330.

Appendix: Common notation

$\mathcal{C}_k$	Sec. 1.3	Calibration cell
D_k	(13)	Maximum relative difference of weights from iteration $k-1$ to iteration k
δ_D	Sec. 1.7	Convergence criteria for D_k
δ_T	Sec. 1.7	Quality control criteria for control totals
i	Sec. 1.1	Subscript i usually applies to units in population
j	Sec. 1.1	Subscript j usually applies to units in sample
k	Sec. 1.7	The outer cycle iteration number
K	Sec. 1.7	The maximum number of the outer cycle iterations
l	Sec. 1.7	Relative limit on weights: all the weights will be made $\geq$ ($l\times$ the input weight)
l		summation index, where I run out of other traditional integer letters
L	Sec. 1.7	Absolute limit on weights: all the weights will be made $\geq L$
n		Sample size; number of sampled units
N		Population size; number of units in population or frame
π_i		Probability of selection of unit i specified by the sampling design
$\mathcal{S}$	Sec. 1.1	Sample; set of sampled units
$T[y]$	(1)	Population-based total of variable $[y]$
$t_m[y]$	(3)	Sample-based weighted estimate of the total $T[y]$; subscript $m = 1, 2, 3$ indicates the type of weights used in computing the total
u	Sec. 1.7	Relative upper limit on weights: all the weights will be made $\leq$ ($u\times$ the input weight)
L	Sec. 1.7	Absolute upper limit on weights: all the weights will be made $\leq U$
$\mathcal{U}$	Sec. 1.1	Universe or population; set of units in population
w_{1i}	(2)	Probability (design) weights; inverse probability of selection
w_{2j}	(4)	Post-stratified weights; random; depend on group sizes in sample; analytically computable
w_{3j}	Sec. 1.3	Calibrated (raked) weights; random; require iterative optimization
x_v	Sec. 1.4	the v-th calibration (control) variable, $v = 1, \ldots, p$. The population total $T[x_v]$ is known.

About the author

Stanislav (Stas) Kolenikov is a Senior Survey Statistician at Abt SRBI. His research interests include applications of statistical methods in public opinion research, such as advanced sampling techniques, survey weighting, calibration, missing data imputation, and variance estimation. Besides survey statistics, Stas has extensive experience developing and applying statistical methods in social sciences, with focus on structural equation modeling and microeconometrics. He has been writing Stata programs since 1998 when Stata was version 5.