The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–43

Updates to the ipfraking ecosystem

Stanislav Kolenikov
Abt Associates
stas_kolenikovs@abtassoc.com

Abstract. Kolenikov (2014) introduced package `ipfraking` for weight calibration procedures known as iterative proportional fitting, or raking, of complex survey weights. This article briefly describes the original package, and adds updates to the core program, as well as a host of additional programs that are used to support the process of creating survey weights in the authors' production code.

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 microdata generalizable to the target population.

Earlier, I introduced (Kolenikov 2014) a Stata package called `ipfraking` that implements calibration of survey weights to known control totals to ensure that the resulting weighted data are representative of the population of interest. The process of calibration is aimed at aligning the sample totals of the key variables with those known for the population as a whole.

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

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

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).

Often, survey statisticians have auxiliary information on the units in the frame, and such information can be included 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. 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.). Weight calibration aims at adjusting the margins, or low level interactions, via an iterative optimization aimed at satisfying the *control totals* for the control variables $\mathbf{x} = (x_1, \ldots, x_p)$:

$$\sum_{j \in \mathcal{S}} w_{3j} \mathbf{x}_j = T[\mathbf{X}_j] \tag{4}$$

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 (4) 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.

In package `ipfraking` (Kolenikov 2014), I implemented a popular calibration algorithm, known as *iterated proportional fitting*, or as *raking*, which consists of iterative updating (post-stratification) of each of the margins. (For an in-depth discussion of distinctions between raking and post-stratification, see Kolenikov (2016).) Since 2014, the continuing code development resulted in additional features that this update documents.

2 Package description

Below, I provide full syntax, and list the new features in a dedicated section.

2.1 Syntax of `ipfraking`

`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.

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.

Linear calibration

`linear` requests linear calibration of weights.

Options to control convergence

`tolerance(#)` defines convergence criteria (the change of weights from one iteration to next). The default is 10^{-6}.

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

`nodivergence` overrides the check that the change in weights is greater at the current iteration than in the previous one, i.e., ignores this termination condition. It is generally recommended, especially in calibration with simultaneous trimming.

`ctrltolerance(#)` defines the criterion to assess the accuracy of the control totals. It does not impact iterations or convergence criteria, but rather only triggers alerts in the output. The default value is 10^{-6}.

`trace` requests a trace plot to be added.

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 $w_{3j} \leq u w_{1j}$ 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 $w_{3j} \geq l w_{1j}$ 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.

`sometimes` means that trimming will be performed after a full set of variables has been used for post-stratification. This is the default behavior if any of the numeric trimming options above are specified.

`once` means that trimming will be performed after the raking process is declared to have converged.

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.

Miscellaneous options

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

`meta` puts information taken by `ipfraking` as inputs and produced throughout the process into 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` (e.g., in replicate weight production).

2.2 New features of `ipfraking`

Since the first publication, the following features and options were added.

Reporting of results and errors by `ipfraking` was improved in several directions.

1. The discrepancy for the worst fitting category is now being reported.
2. The number of trimmed observations is reported.
3. If `ipfraking` determines that the categories do not match in the control totals received from `ctotals()` and those found in the data, a full listing of categories is provided, and the categories not found in one or the other are explicitly shown.

Linear calibration (Case 1 of Deville and Särndal (1992)) is provided with `linear` option. The weights are calculated analytically:

$$w_{j,\text{lin}} = w_{1j}(1 + \mathbf{x}_j'\lambda). \quad \lambda = \left(\sum_{j\in\mathcal{S}} w_{1j}\mathbf{x}_j\mathbf{x}_j'\right)^{-1}(T[\mathbf{X}_j] - t_1[y]) \tag{5}$$

This works very fast, but has an undesirable artefact of producing negative weights, as the range of weights is not controlled. (As raking works by multiplying the currents weights by positive factors, if the input weights are all positive, the output weights will be positive as well.) Negative weights are not allowed by the official `svy` commands or commands that work with `[pweights]`. In my experience, running linear weights first, pulling up the negative weights (`replace weight = 1 if weight <= 1`) and re-raking using the iterative proportional fitting "proper" runs faster than raking from scratch.

Option `meta` saves more information in characteristics of the calibrated weight variables.

(*Continued on next page*)

```
. capture drop rakedwgt3
. ipfraking [pw=finalwgt], gen( rakedwgt3 ) ///
>     ctotal( ACS2011_sex_age Census2011_region Census2011_race ) ///
>     trimhiabs(200000) trimloabs(2000) meta
 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
The worst relative discrepancy of  3.0e-08 is observed for race == 3
Target value =   20053682; achieved value =   20053682
Trimmed due to the upper absolute limit: 5 weights.
   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[source]:          finalwgt
  rakedwgt3[objfcn]:          2.25435521346e-07
  rakedwgt3[maxctrl]:         3.00266822363e-08
  rakedwgt3[converged]:       1
  rakedwgt3[worstcat]:        3
  rakedwgt3[worstvar]:        race
  rakedwgt3[command]:         [pw=finalwgt], gen( rakedwgt3 ) ctotal( ACS2011_sex_age Census2011_region ..
  rakedwgt3[trimloabs]:       trimloabs(2000)
  rakedwgt3[trimhiabs]:       trimhiabs(200000)
  rakedwgt3[trimfrequency]:   sometimes
  rakedwgt3[hash1]:           2347674164
  rakedwgt3[mat3]:            Census2011_race
  rakedwgt3[over3]:           race
  rakedwgt3[totalof3]:        _one
  rakedwgt3[Census2011_race]: 7.48567503861e-09
  rakedwgt3[mat2]:            Census2011_region
  rakedwgt3[over2]:           region
  rakedwgt3[totalof2]:        _one
  rakedwgt3[Census2011_region]:
                              3.00266822363e-08
  rakedwgt3[mat1]:            ACS2011_sex_age
  rakedwgt3[over1]:           sex_age
  rakedwgt3[totalof1]:        _one
  rakedwgt3[ACS2011_sex_age]: 4.13778410340e-09
  rakedwgt3[note1]:           Raking controls used: ACS2011_sex_age Census2011_region Census2011_race
  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 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)
`source`	weight variable specified as the `[pw=]` input
`worstvar`	the variable in which the greatest discrepancy between the targets and the achieved weighted totals (`maxctrl`) was observed
`worstcat`	the category of the `worstvar` variable in which the greatest discrepancy was observed

For the control total matrices $\#= 1, 2, \ldots$, the following meta-information is stored.

`mat`$\#$	the name of the control total matrix
`totalof`$\#$	the multiplier variable (matrix' `coleq`
`over`$\#$	the margin associated with the matrix (i.e., the categories represented by the columns)

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**.

2.3 Excel reports on raked weights: `ipfraking_report`

`ipfraking_report using` *`filename`* `, raked_weight(`*varname*`)` [
`matrices(`*`namelist`*`) by(`*varlist*`) xls replace force`]

The utility command `ipfraking_report` produces a detailed report describing the raked weights, and places it into *filename*`.dta` file (or, if `xls` option is specified, both *filename*`.dta` and *filename*`.xls` files).

Along the way, `ipfraking_report` runs a regression of the log raking ratio w_{3j}/w_{1j} on the calibration variables. This regression is expected to have R^2 very close to 1, and the regression coefficients provide insights regarding which categories received greater vs. smaller adjustments.

(*Continued on next page*)

```
. ipfraking_report using rakedwgt3-report, raked_weight(rakedwgt3) replace by(_one)
Margin variable sex_age (total variable: _one; categories: 11 12 13 21 22 23).
Margin variable region (total variable: _one; categories: 1 2 3 4).
Margin variable race (total variable: _one; categories: 1 2 3).
Auxiliary variable _one (categories: 1).
file rakedwgt3-report.dta saved
```

Source	SS	df	MS		
				Number of obs	= 10,351
				F(10, 10340)	> 99999.00
Model	2086.13859	10	208.613859	Prob > F	= 0.0000
Residual	.78315703	10,340	.000075741	R-squared	= 0.9996
				Adj R-squared	= 0.9996
Total	2086.92175	10,350	.201634952	Root MSE	= .0087

__000003	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
sex_age						
11	.0644365	.0002775	232.21	0.000	.0638925	.0649804
12	.4545577	.0003154	1441.25	0.000	.4539395	.455176
13	.6782466	.0002804	2418.71	0.000	.6776969	.6787963
22	.3966406	.0003049	1300.84	0.000	.3960429	.3972383
23	.7304392	.0002726	2679.97	0.000	.7299049	.7309734
region						
NE	-.4455127	.0002536	-1756.49	0.000	-.4460099	-.4450155
MW	-.4428144	.0002335	-1896.53	0.000	-.4432721	-.4423567
W	-.6672675	.0002407	-2772.21	0.000	-.6677393	-.6667957
race						
Black	.3360321	.0002848	1180.08	0.000	.3354739	.3365902
Other	1.613276	.0006303	2559.34	0.000	1.612041	1.614512
_cons	.5864801	.0002455	2388.48	0.000	.5859988	.5869614

```
Raking adjustments for sex_age variable:
  the smallest was        1.798 for category 21 (21)
  the greatest was        3.732 for category 23 (23)
Raking adjustments for region variable (1=NE, 2=MW, 3=S, 4=W):
  the smallest was        0.922 for category 4 (W)
  the greatest was        1.798 for category 3 (S)
Raking adjustments for race variable (1=white, 2=black, 3=other):
  the smallest was        1.798 for category 1 (White)
  the greatest was        9.023 for category 3 (Other)
```

Options of `ipfraking_report`

`raked_weight(`*varname*`)` specifies the name of the raked weight variable to create the report for. This is a required option.

`matrices(`*namelist*`)` specifies a list of matrices (formatted as the matrices supplied to `ctotal()` option of `ipfraking`) to produce weighting reports for. In particular, the variables and their categories are picked up from these matrices; and the control totals/proportions are compared to those defined by the weight being reported on.

`by(`*varlist*`)` specifies a list of additional variables for which the weights are to be tabulated in the raking weights report. The difference with the `matrices()` option is that

the control totals for these variables may not be known (or may not be relevant). In particular, `by(_one)`, where `_one` is identically one, will produce the overall report.

`xls` requests exporting the report to an Excel file.

`replace` specifies that the files produced by `ipfraking_report` (i.e., the `.dta` and the .xls file if `xls` option is specified) should be overwritten.

`force` requires that a variable that may be found repeatedly (between the calibration variables supplied originally to `ipfraking`, the variables found in the independent total `matrices()`, and the variables without the control totals provided in `by()` option) is processed every time it is encountered. (Otherwise, it is only processed once.)

Variables in the raking report

The following variables are saved in the raking report.

(*Continued on next page*)

Variable name	Definition
`Weight_Variable`	The name of the weight variable, `generate()`
`C_Total_Margin_Variable_Name`	The name of the control margin, `rowname` of the corresponding `ctotal()` matrix
`C_Total_Margin_Variable_Label`	The label of the control margin variable
`Variable_Class`	The role of the variable in the report: Raking margin: a variable used as a calibration margin (picked up automatically from the `ctotal()` matrix, provided `meta` option was specified) Other known target: supplied with `matrices()` option of `ipfraking_report` Auxiliary variable: additional variable supplied with `by()` option of `ipfraking_report`
`C_Total_Arg_Variable_Name`	The name of the multiplier variable
`C_Total_Arg_Variable_Label`	The label of the multiplier variable
`C_Total_Margin_Category_Number`	Numeric value of the control total category
`C_Total_Margin_Category_Label`	Label of the control total category
`Category_Total_Target`	The control total to be calibrated to (the specific entry in the `ctotal()` matrix)
`Category_Total_Prop`	Control total proportion (the ratio of the specific entry in the `ctotal()` matrix to the matrix total)
`Unweighted_Count`	Number of sample observations in the category
`Unweighted_Prop`	Unweighted proportion
`Unweighted_Prop_Discrep`	Difference `Unweighted_Prop` - `Category_Total_Prop`
`Category_Total_SRCWGT`	Weighted category total, with source weight
`Category_Prop_SRCWGT`	Weighted category proportion, with source weight
`Category_Total_Discrep_SRCWGT`	Difference `Category_Total_SRCWGT` - - `Category_Total_Target`
`Category_Prop_Discrep_SRCWGT`	Difference `Category_Prop_SRCWGT` - - `Category_Total_Prop`
`Category_RelDiff_SRCWGT`	`reldif(Category_Total_SRCWGT, Category_Total_Target)`
`Overall_Total_SRCWGT`	Sum of source weights
`Source`	The name of the matrix from which the totals were obtained
`Comment`	Placeholder for comments, to be entered during manual review

For each of the input weights (`SRCWGT` suffix), raked weights (`RKDWGT` suffix) and raking ratio (the ratio of raked and input weights, `RKDRATIO` suffix), the following summaries are provided.

Variable name	Definition
Min_*WEIGHT*	Min of source weights
P25_*WEIGHT*	25th percentile of source weights
P50_*WEIGHT*	Median of source weights
P75_*WEIGHT*	75th percentile of source weights
Max_*WEIGHT*	Max of source weights
Mean_*WEIGHT*	Mean of source weights
SD_*WEIGHT*	Standard deviation of source weights
DEFF_*WEIGHT*	Apparent UWE DEFF of source weights

Example

```
. use rakedwgt3-report, clear
(Weighting report on rakedwgt3)
. list C_Total_Margin_Variable_Name C_Total_Margin_Category_Label ///
>         Category_Total_Target Category_Total_RKDWGT DEFF_SRCWGT DEFF_RKDWGT , ///
>         sepby( C_Total_Margin_Variable_Name )
```

	C_Tota..	~y_Label	Categor~t	Categor..	DEFF_SR~T	DEFF_RK~T
1.	sex_age	11	41995394	41995394	1.2148059	1.6259899
2.	sex_age	12	42148662	42148662	1.2462168	1.5716613
3.	sex_age	13	26515340	26515340	1.2241095	1.5460785
4.	sex_age	21	41164255	41164255	1.2325105	1.5639529
5.	sex_age	22	43697440	43697440	1.1937826	1.5175312
6.	sex_age	23	32773080	32773080	1.233902	1.664307
7.	region	NE	40679030	40679030	1.3056639	1.3657837
8.	region	MW	49205289	49205289	1.3475551	1.4909581
9.	region	S	85024007	85024006	1.4950056	1.4912995
10.	region	W	53385843	53385844	1.459859	2.3772667
11.	race	White	1.784e+08	1.784e+08	1.4059259	1.4337901
12.	race	Black	29856865	29856865	1.5173846	1.5092533
13.	race	Other	20053682	20053682	1.3179136	1.2264706
14.	_one	1	.	2.283e+08	1.4164382	1.7349278

Functionality of `ipfraking_report` is aimed at the manual review of its reporting of the categories that differ the most in the output, and the resulting report file in Excel, although for some aspects of automated quality control, it will be useful, as well.

2.4 Collapsing weighting cells: `wgtcellcollapse`

An additional new component of `ipfraking` package is a tool to semi-automatically collapse weighting cells, in order to achieve some minimal sample size.

`wgtcellcollapse` *task* [*if*] [*in*] , [`task_options`]

where *task* is one of:

`define` to define collapsing rules explicitly

`sequence` to create collapsing rules for a sequence of categories

`report` to list the currently defined collapsing rules

`candidate` to find rules applicable to a given category

`collapse` to perform cell collapsing

`label` to label collapsed cells using the original labels after `wgtcellcollapse collapse`

2.5 Syntax of `wgtcellcollapse report`

`wgtcellcollapse report , variables(`*varlist*`)` [`break`]

`variables(`*varlist*`)` is the list of variables for which the collapsing rule are to be reported

`break` requires `wgtcellcollapse report` to exit with error when technical inconsistencies are encountered

2.6 Syntax of `wgtcellcollapse define`

`wgtcellcollapse define , variables(`*varlist*`)` [`from(`***numlist***`) to(`#`)`
`label(`*string*`) max(`#`) clear`]

`variables(`*varlist*`)` is the list of variables for which the collapsing rule can be used

`from(`***numlist***`)` is the list of categories that can be collapsed according to this rule

`to(`#`)` is the numeric value of the new, collapsed category

`label(`*string*`)` is the value label to be attached to the new, collapsed category

`max(`#`)` overrides the automatically determined max value of the collapsed variable

`clear` clears all the rules currently defined

Individual collapsing rules can be defined as follows.

```
.
. clear
.
. set obs 4
number of observations (_N) was 0, now 4
.
. gen byte x = _n
.
. label define x_lbl 1 "One" 2 "Two" 3 "Three" 4 "Four"
.
. label values x x_lbl
```

```
.
. wgtcellcollapse define, var(x) from(1 2 3) to(123)
.
. wgtcellcollapse report, var(x)
Rule (1): collapse together
  x == 1 (One)
  x == 2 (Two)
  x == 3 (Three)
  into x == 123 (123)
  WARNING: unlabeled value x == 123

.
```

Note how `break` option of `wgtcellcollapse` can be used to abort the execution when technical deficiencies in the rules or in the data are encountered. In this case, the label of the new category 123 was not defined, and this is considered a serious enough deficiency to stop.

```
.
. wgtcellcollapse report, var(x) break
Rule (1): collapse together
  x == 1 (One)
  x == 2 (Two)
  x == 3 (Three)
  into x == 123 (123)
  ERROR: unlabeled value x == 123
assertion is false
r(9);

.
. wgtcellcollapse define, var(x) clear
.
. wgtcellcollapse define, var(x) from(1 2 3) to(123) label("One through three")
.
. wgtcellcollapse report, var(x) break
Rule (1): collapse together
  x == 1 (One)
  x == 2 (Two)
  x == 3 (Three)
  into x == 123 (One through three)

.
```

2.7 Syntax of `wgtcellcollapse sequence`

`wgtcellcollapse sequence , variables(`*varlist*`) from(`***numlist***`) depth(#)`

`variables(`*varlist*`)` is the list of variables for which the collapsing rule can be used

`from(`***numlist***`)` is the sequence of values from which the plausible subsequences can be constructed

`depth(#)` is the maximum number of the original categories that can be collapsed

Moderate length sequences of collapsing categories can be defined as follows.

```
.
. clear
.
. set obs 4
number of observations (_N) was 0, now 4
.
. gen byte x = _n
.
. label define x_lbl 1 "One" 2 "Two" 3 "Three" 4 "Four"
.
. label values x x_lbl
.
. wgtcellcollapse sequence, var(x) from(1 2 3 4) depth(3)
.
. wgtcellcollapse report, var(x)
Rule (1): collapse together
  x == 1 (One)
  x == 2 (Two)
  into x == 212 (One to Two)
Rule (2): collapse together
  x == 2 (Two)
  x == 3 (Three)
  into x == 223 (Two to Three)
Rule (3): collapse together
  x == 3 (Three)
  x == 4 (Four)
  into x == 234 (Three to Four)
Rule (4): collapse together
  x == 1 (One)
  x == 2 (Two)
  x == 3 (Three)
  into x == 313 (One to Three)
Rule (5): collapse together
  x == 1 (One)
  x == 223 (Two to Three)
  into x == 313 (One to Three)
Rule (6): collapse together
  x == 3 (Three)
  x == 212 (One to Two)
  into x == 313 (One to Three)
Rule (7): collapse together
  x == 2 (Two)
  x == 3 (Three)
  x == 4 (Four)
  into x == 324 (Two to Four)
Rule (8): collapse together
  x == 2 (Two)
  x == 234 (Three to Four)
  into x == 324 (Two to Four)
Rule (9): collapse together
  x == 4 (Four)
  x == 223 (Two to Three)
```

```
  into x == 324 (Two to Four)
.
```

When creating sequential collapses, `wgtcellcollapse sequence` uses the following mnemonics in creating the new labels:

- First comes the length of the collapsed subsequence (up to `depth(#)`).
- Then comes the starting value of the category in the subsequence (padded by zeroes as needed).
- Then comes the ending value of the category in the subsequence (padded by zeroes as needed).

In the example above, rules 7 through 9 lead to collapsing into the new category 324. This should be interpreted as "the subsequence of length 3 that starts with category 2 and ends with category 4". A numeric value of the collapsed category that reads like 50412 means "the subsequence of length 5 that starts with category 4 and ends with category 12".

Note that `wgtcellcollapse sequence` respects the order in which the categories are supplied in the `from()` option, and does not sort them.

2.8 Syntax of `wgtcellcollapse candidate`

`wgtcellcollapse candidate , variable(`*varname*`) category(#)` [`max #`]

`variable(`*varname*`)` is the variable whose collapsing rules are to be searched

`category(#)` is the category for which the candidate rules are to be identified

`max(#)` is the maximum value of the categories in the candidate rules to be returned

The rules found are quietly returned through the mechanism of `sreturn`, see [P] **return**, as they are intended to be stay in memory sufficiently long for `wgtcellcollapse collapse` to evaluate each rule.

```
.
. wgtcellcollapse candidate, var(x) cat(2)
.
. sreturn list
macros:
           s(goodrule) : "1 2 4 7 8"
              s(rule8) : "2:234=324"
              s(rule7) : "2:3:4=324"
              s(rule4) : "1:2:3=313"
              s(rule2) : "2:3=223"
              s(rule1) : "1:2=212"
                s(cat) : "2"
                  s(x) : "x"
```

```
.
. wgtcellcollapse candidate, var(x) cat(2) max(9)
.
. sreturn list
macros:
           s(goodrule) : "1 2 4 7"
              s(rule7) : "2:3:4=324"
              s(rule4) : "1:2:3=313"
              s(rule2) : "2:3=223"
              s(rule1) : "1:2=212"
                s(cat) : "2"
                  s(x) : "x"

.
. wgtcellcollapse candidate, var(x) cat(212)
.
. sreturn list
macros:
           s(goodrule) : "6"
              s(rule6) : "3:212=313"
                s(cat) : "212"
                  s(x) : "x"

.
. wgtcellcollapse candidate, var(x) cat(55)
.
. sreturn list
macros:
                s(cat) : "55"
                  s(x) : "x"

.
```

In the second call to the option `max(9)` was used to restrict the returned rules to the rules that deal with the original categories only. In the third call, a list of rules that involve a collapsed category `cat(212)` was requested. Requests for nonexisting categories are not considered errors, but simply produce empty lists of "good rules"

2.9 Syntax of `wgtcellcollapse collapse`

`wgtcellcollapse collapse` [*if*] [*in*], variables(*varlist*) `mincellsize(`#`)`
saving(*dofile_name*) [generate(*newvarname*) `replace append`
`feed(`*varname*`) strict sort(`*varlist*`) run maxpass(`#`)` maxcategory(#)
zeroes(*numlist*) `greedy`]

variables(*varlist*) provides the list of variables whose cells are to be collapsed. When more than one variable is specified, `wgtcellcollapse collapse` proceeds from right to left, i.e., first attempts to collapse the rightmost variable.

`mincellsize(`#`)` specifies the minimum cell size for the collapsed cells. For most weighting purposes, values of 30 to 50 can be recommended.

`gen`erate(*newvarname*) specifies the name of the collapsed variable to be created.

`feed(`*varname*`)` provides the name of an already existing collapsed variable.

`strict` modifies the behavior of `wgtcellcollapse collapse` so that only collapsing rules for which all participating categories have nonzero counts are utilized.

`sort(`*varlist*`)` sorts the data set before proceeding to collapse the cell. The default sort order is in terms of the values of the collapsed variable. A different sort order may produce a different set of collapsed cell when cells are tied on size.

`maxpass(#)` specifies the maximum number of passes through the data set. The default value is 10000.

`maxcat`egory(#) is the maximum category value of the variable being collapsed. It is passed to the internal calls to `wgtcellcollapse candidate`, see above.

`zer`oes(*numlist*) provides a list of the categories of the collapsed variable that may have zero counts in the data.

`greedy` modifies the behavior `wgtcellcollapse collapse` to prefer the rules that collapse the maximum number of categories.

Options to deal with the do-file to write the collapsing code to:

`sav`ing(*dofile_name*) specifies the name of the do-file that will contain the cell collapsing code.

`replace` overwrites the do-file if one exists.

`append` appends the code to the existing do-file.

`run` specifies that the do-file created is run upon completion. This option is typically specified with most runs.

The primary intent of `wgtcellcollapse collapse` is to create the code that can be utilized for both the survey data file and the population targets data file that are assumed to have identically named variables. Thus it does not only manipulate the data in the memory and collapses the cells, but also produces the do-file code that can be recycled. To that effect, when a do-file is created with the `replace` and `saving()` options, the user needs to specify `generate()` option to provide the name of the collapsed variable; and when the said do-file is appended with the the `replace` and `saving()` options, the name of that variable is provided with the `feed()` option.

The algorithm `wgtcellcollapse collapse` uses to identify the cells to be collapsed is a variation of greedy search. It first identifies the cells with the lowest (positive) counts; finds the candidate rules for the variable(s) to be collapsed; and uses the rule that has produces the smallest size of the collapsed cell across all applicable rules. So when it finds several rules that are applicable to the cell being currently processed that has a size of 5, and the candidate rules produces cells of sizes 7, 10 and 15, `wgtcellcollapse collapse` will use the rule that produces the cell of size 7. The algorithm runs until all cells have sizes of at least `mincellsize(#)` or until `maxpass(#)` passes through the

data are executed. It is a pretty dumb algorithm, actually, and it fails quite often. For that reason, a number of hooks are provided to modify its behavior.

Hint 1. Since `wgtcellcollapse collapse` works with the sample data, it will not be able to identify categories that are not observed in the sample (e.g., rare categories), but may be present in the population. This will lead to errors at the raking stage, when the control total matrices have more categories than the data, forcing `ipfraking` to stop. To help with that, the option `zeroes()` allows the user to pass the categories of the variables that are known to exist in the population but not in the sample.

Hint 2. The behavior of `wgtcellcollapse collapse, zeroes()` may still not be satisfactory. As it evaluates the sample sizes of the collapsed cells across a number of candidate rules that involve zero cells, it will probably pick up the rule with lowest number, and that rule may as well leave some other candidate rules with zero cells untouched. This may create problems when `wgtcellcollapse collapse` returns to those untouched cells, and looks for the existing cells to collapse them with, creating collapsing rules with breaks in the sequences. To improve upon that behavior, option `greedy` makes `wgtcellcollapse collapse` look for a rule that has many categories as possible, thus collapsing as many categories with zero counts in one swipe as it can.

Hint 3. Other than for dealing with zero cells, the option `strict` should be specified most of the times. It effectively makes sure that the candidate rules correspond to the actual data.

Hint 4. Sometimes, you see some combinations in the data that seem like a nobrainer to collapse. Well, they are nobrainers to you, but `wgtcellcollapse collapse` is not that smart. If you want to guarantee some specific combination of cells to be collapsed by `wgtcellcollapse collapse`, your best bet may be to explicitly identify them with the *if* condition, and specify some ridiculously large cell size like `mincellsize(10000)` so that `wgtcellcollapse collapse` makes every possible effort to collapse those cells. It will exit with a complaint that this size could not be achieved, but hopefully the cells will be collapsed as needed.

2.10 Syntax of `wgtcellcollapse label`

`wgtcellcollapse label , variable(`*varname*`) category(`#`)` [`verbose force`]

`variable(`*varname*`)` is the collapsed variable to be labeled.

`verbose` outputs the labeling results. There may be a lot of output.

`force` instructs `wgtcellcollapse label` to only use categories present in the data.

Motivating example

Development of `wgtcellcollapse` was to address the need to collapse cells of the margin variables so that each cell has a minimum sample size; and to do so in a way that can

be easily made consistent between the sample data and the population targets data. The problem arises when some of the target variables have dozens of categories, most of which have small counts. While the primary motivation comes from transportation surveys, the ideas are also applicable to other domains, e.g., continuous age variables or highly detailed race/ethnicity or region of origin categories in health or economic surveys.

The workflow of `wgtcellcollapse` is demonstrated with the following simulated data set of trips along a metro line composed of 21 stations:

```
. use stations, clear
. list station_id, sep(0)
```

	station_id
1.	3. Alewife
2.	6. Brookline
3.	10. Carmenton
4.	13. Dogville
5.	17. East End
6.	21. Framington
7.	25. Grand Junction
8.	28. High Point
9.	32. Irvingtown
10.	36. Johnsville
11.	39. King Street
12.	42. Limerick
13.	46. Moscow City
14.	49. Ninth Street
15.	52. Ontario Lake
16.	56. Picadilly Square
17.	59. Queens Zoo
18.	63. Redline Circle
19.	66. Silver Spring
20.	70. Toledo Town
21.	73. Union Station

Turnstile counts were collected at entrances and exits of the stations, producing the following population figures.

```
. use trip_population, clear
. table board_id daypart if alight_id == 73, c(mean geton) cellwidth(10)
note: cellwidth too small, variable name truncated;
      to increase cellwidth, specify cellwidth(#)
```

	daypart				
board_id	AM Peak	Midday	PM Reverse	Night	Weekend
3. Alewife	1423	34	219	113	44
6. Brookline	7198	298	773	169	144
10. Carmenton	19254	181	3739	872	422
13. Dogville	12626	872	3476	769	1270
17. East End	2470	143	1263	145	114
21. Framington	634	50	1296	133	60

25. Grand Junction	2208	233	439	88	166
28. High Point	4319	424	3740	482	115
32. Irvingtown	1221	34	444	30	167
36. Johnsville	93	4	64	2	6
39. King Street	398	46	76	11	13
42. Limerick	1021	19	129	53	34
46. Moscow City	3300	776	984	140	301
49. Ninth Street	38	22	191	5	5
52. Ontario Lake	606	22	80	18	23
56. Picadilly Square	642	71	622	153	69
59. Queens Zoo	331	23	174	15	19
63. Redline Circle	270	4	63	13	3
66. Silver Spring	3402	240	950	206	445
70. Toledo Town	5085	61	744	272	112

```
. table alight_id daypart if board_id==3, c(mean getoff) cellwidth(10)
note: cellwidth too small, variable name truncated;
      to increase cellwidth, specify cellwidth(#)
```

	daypart				
alight_id	AM Peak	Midday	PM Reverse	Night	Weekend
6. Brookline	19	0	3	2	0
10. Carmenton	492	18	56	23	15
13. Dogville	2475	42	423	153	80
17. East End	929	31	193	67	68
21. Framington	404	13	91	28	27
25. Grand Junction	576	20	147	42	41
28. High Point	2189	89	560	165	167
32. Irvingtown	288	10	91	21	18
36. Johnsville	41	0	11	2	1
39. King Street	131	3	38	8	6
42. Limerick	277	9	87	20	18
46. Moscow City	1746	78	556	142	128
49. Ninth Street	88	2	25	3	4
52. Ontario Lake	232	11	70	14	14
56. Picadilly Square	633	33	198	47	47
59. Queens Zoo	230	10	71	13	14
63. Redline Circle	90	2	26	3	4
66. Silver Spring	1134	67	369	91	85
70. Toledo Town	1372	81	444	112	118
73. Union Station	53193	3038	16007	2733	2677

A survey was administered to a sample of the metro line users, with the following counts of cases collected.

```
. use trip_sample, clear
. tab board_id daypart
```

	daypart					
board_id	AM Peak	Midday	PM Revers	Night	Weekend	Total
3. Alewife	46	4	11	7	3	71
6. Brookline	235	4	35	6	7	287
10. Carmenton	652	4	185	46	24	911
13. Dogville	411	41	166	35	56	709
17. East End	86	5	64	4	4	163

21. Framington	30	3	74	3	1	111
25. Grand Junction	73	13	23	6	6	121
28. High Point	160	20	188	25	12	405
32. Irvingtown	35	2	25	1	15	78
36. Johnsville	5	1	1	0	0	7
39. King Street	17	1	2	0	1	21
42. Limerick	28	0	9	1	3	41
46. Moscow City	95	31	50	7	13	196
49. Ninth Street	0	0	9	0	0	9
52. Ontario Lake	13	1	4	1	1	20
56. Picadilly Square	23	4	35	7	5	74
59. Queens Zoo	10	1	14	0	2	27
63. Redline Circle	13	0	5	0	0	18
66. Silver Spring	106	18	38	12	17	191
70. Toledo Town	154	6	33	11	3	207
Total	2,192	159	971	172	173	3,667

```
. tab alight_id daypart
```

	daypart					
alight_id	AM Peak	Midday	PM Revers	Night	Weekend	Total
6. Brookline	1	0	0	0	0	1
10. Carmenton	11	1	1	0	1	14
13. Dogville	85	1	14	6	5	111
17. East End	35	1	18	1	4	59
21. Framington	14	1	2	2	2	21
25. Grand Junction	14	2	8	1	1	26
28. High Point	72	4	22	10	8	116
32. Irvingtown	9	0	4	2	2	17
36. Johnsville	3	0	1	0	0	4
39. King Street	0	0	3	0	0	3
42. Limerick	13	0	2	0	2	17
46. Moscow City	80	6	22	6	6	120
49. Ninth Street	3	1	1	0	0	5
52. Ontario Lake	2	0	1	2	1	6
56. Picadilly Square	23	1	8	3	2	37
59. Queens Zoo	6	0	5	1	0	12
63. Redline Circle	5	0	0	0	0	5
66. Silver Spring	49	0	19	3	9	80
70. Toledo Town	43	3	24	6	7	83
73. Union Station	1,724	138	816	129	123	2,930
Total	2,192	159	971	172	173	3,667

As only 3667 surveys were collected from a total of 96783 riders, we would reasonably expect that things do not align quite well. We expect weighting to correct for at least a portion of that nonresponse. The data available for calibration includes the population turnstile counts listed above, and we will produce interactions of daypart and station that will serve as two weighting margins (one for the stations where the metro users boarded, and one for the stations where they got off).

First, we need to define the weighting rules. In this case, the stations are numbered sequentially, with the northernmost, say, station Alewife being number 3, and the southernmost station, Union Station, where everybody gets off to rush to their city jobs or attractions, being number 73. Below, we create a list of stations and provide it to

`wgtcellcollapse sequence`. We would be collapsing stations along the line, with the expectation that travelers boarding or leaving at adjacent stations within the same day part are more similar to one another than the travelers boarding or leaving a particular station at different times of the day. Still, some collapsing rules can be defined for the `daypart` variable as well — mostly because `wgtcellcollapse collapse` expects all variables to have collapsing rules defined.

```
. use trip_sample, clear
. wgtcellcollapse sequence , var(daypart) from(2 3 4) depth(3)
. levelsof board_id, local(stations_on)
3 6 10 13 17 21 25 28 32 36 39 42 46 49 52 56 59 63 66 70
. levelsof alight_id, local(stations_off)
6 10 13 17 21 25 28 32 36 39 42 46 49 52 56 59 63 66 70 73
. local all_stations : list stations_on | stations_off
. wgtcellcollapse sequence , var(board_id alight_id) from(`all_stations´) depth(20)
```

The number of collapsing rules for variables `board_id` and `alight_id` is 2961 each.

Let us say that we want to define weighting cells with at least 20 cases in each. We will thus start with weighting cells defined as station-by-daypart interaction, and collapsing stations within daypart to achieve the cell sizes of at least 20 cases. Here is what a simple run of `wgtcellcollapse collapse` might look like.

2.11 Utility programs

The original package `ipfraking` provided two additional utility programs, `mat2do` and `xls2row`. An additional utility program was added to compute the design effects and margins of error, common tasks associated with describing survey weights. Specifically, the Transparency Initiative of the American Association for Public Opinion Research (AAPOR 2014) requires that

> For probability samples, the estimates of sampling error will be reported, and the discussion will state whether or not the reported margins of sampling error or statistical analyses have been adjusted for the design effect due to weighting, clustering, or other factors.

`whatsdeff` *weight_variable* [*if*] [*in*] , [`by(`*varlist*`)`]

The utility program `whatsdeff` calculates the apparent design effect due to unequal weighting, $\text{DEFF}_{\text{UWE}} = 1 + CV_w^2 =$ `1 + r(Var)/(r(mean))^2` from `summarize` *weight_variable*. Additionally, it reports the effective sample size, $n/\text{DEFF}_{\text{UWE}}$, and also returns the margins of error for the sample proportions that estimate the population proportions of 10% and 50%.

```
. webuse nhanes2, clear
```

```
. whatsdeff finalwgt
```

Group	Min	Mean	Max	CV	DEFF	N	N eff
Overall	2000.00	11318.47	79634.00	0.6453	1.4164	10351	7307.97

```
. return list
scalars:
                  r(N) =  10351
              r(MOE10) =  .0068792766212984
              r(MOE50) =  .0114654610354974
       r(Neff_Overall) =  7307.97435325364
       r(DEFF_Overall) =  1.416397964696134
. whatsdeff finalwgt, by(sex)
```

Group	Min	Mean	Max	CV	DEFF	N	N eff
sex							
Male	2000.00	11426.14	79634.00	0.6578	1.4326	4915	3430.94
Female	2130.00	11221.12	61534.00	0.6333	1.4010	5436	3880.01
Overall	2000.00	11318.47	79634.00	0.6453	1.4164	10351	7307.97

```
. return list
scalars:
                  r(N) =  10351
              r(MOE10) =  .0068792766212984
              r(MOE50) =  .0114654610354974
       r(Neff_Overall) =  7307.97435325364
       r(DEFF_Overall) =  1.416397964696134
        r(Neff_Female) =  3880.00710397866
        r(DEFF_Female) =  1.40102836266093
          r(Neff_Male) =  3430.938195872213
          r(DEFF_Male) =  1.432552765279559
```

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*)

```
------------------------------------------------------------------------------
             |                Linearized
        Over |      Total   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.

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

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

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

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 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 **??** 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 **??** of Algorithm 2. Alternatively, trimming can be applied after every full cycle over variables at step **??** of Algorithm 2. Finally, trimming can be applied after each sub-iteration at step **??** 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 **??**. 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
 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
The worst relative discrepancy of  3.0e-08 is observed for race == 3
Target value =   20053682; achieved value =   20053682
Trimmed due to the upper absolute limit: 5 weights.
   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[source]:          finalwgt
  rakedwgt3[objfcn]:          2.25435521346e-07
  rakedwgt3[maxctrl]:         3.00266822363e-08
  rakedwgt3[converged]:       1
  rakedwgt3[worstcat]:        3
  rakedwgt3[worstvar]:        race
  rakedwgt3[command]:         [pw=finalwgt], gen( rakedwgt3 ) ctotal( ACS2011_sex_age Census2011_region ..
  rakedwgt3[trimloabs]:       trimloabs(2000)
  rakedwgt3[trimhiabs]:       trimhiabs(200000)
  rakedwgt3[trimfrequency]:   sometimes
  rakedwgt3[hash1]:           2347674164
  rakedwgt3[mat3]:            Census2011_race
  rakedwgt3[over3]:           race
```

```
rakedwgt3[totalof3]:        _one
rakedwgt3[Census2011_race]: 7.48567503861e-09
rakedwgt3[mat2]:            Census2011_region
rakedwgt3[over2]:           region
rakedwgt3[totalof2]:        _one
rakedwgt3[Census2011_region]:
                            3.00266822363e-08
rakedwgt3[mat1]:            ACS2011_sex_age
rakedwgt3[over1]:           sex_age
rakedwgt3[totalof1]:        _one
rakedwgt3[ACS2011_sex_age]: 4.13778410340e-09
rakedwgt3[note1]:           Raking controls used: ACS2011_sex_age Census2011_region Census2011_race
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 (**??**) 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 **??**, 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
```

	Proportion	BRR 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 an 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 **??** of Algorithm 2 in Section **??**). 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 Associates.

5 References

AAPOR. 2014. *AAPOR Terms and Conditions for Transparency Certification*. The American Association for Public Opinion Research. Available at http://www.aapor.org/AAPOR$_Main/media/MainSiteFiles/TI-Terms-and-Conditions-10-4-17.pdf$.

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.

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.

Gould, W. 2003. Stata tip 3: How to be assertive. *Stata Journal* 3(4).

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.

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

———. 2014. Calibrating survey data using iterative proportional fitting. *The Stata Journal* 14(1): 22–59.

———. 2016. Post-stratification or non-response adjustment? *Survey Practice* 9(3). Available at http://www.surveypractice.org/index.php/SurveyPractice/article/view/315.

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.

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.

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

About the author

Stanislav (Stas) Kolenikov is a Senior Scientist at Abt Associates. His work involves applications of statistical methods in data collection for public opinion research, public health, transportation, and other disciplines that utilize collection of survey data. Within survey methodology, his expertise includes advanced sampling techniques, survey weighting, calibration, missing data imputation, variance estimation, nonresponse analysis and adjustment, and small area 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.