The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–13

kmr: A Command to Correct Survey Weights for Unit Nonresponse using Group's Response Rates

Ercio Muñoz
Stone Center on Socio-Economic Inequality and CUNY Graduate Center
New York, USA
emunozsaavedra@gc.cuny.edu
Salvatore Morelli
Stone Center on Socio-Economic Inequality and CUNY Graduate Center
New York, USA
smorelli@gc.cuny.edu

Abstract. This article describes **kmr**, a Stata command to estimate a micro compliance function using group's nonresponse rates (2007, Journal of Econometrics 136: 213-235), which can be used to correct survey weights for unit nonresponse. We illustrate the use of **kmr** with an empirical example using the Current Population Survey and state-level nonresponse rates.

Keywords: st0001, kmr, sample surveys, selective unit nonresponse bias, survey reweighting

1 Introduction

Unit nonresponse rates in household socioeconomic surveys have been increasing over the last decades (**?**). Unit nonresponse is problematic for the measurement of inequality and poverty when response is not random, especially when it is related to the variable of interest.

There is evidence that household income systematically affects survey response. Using the Current Population Survey (CPS) of the United States, **?** show that nonresponse increases in the tails of the income distribution. This empirical evidence rejects the ignorability assumption (the fact that nonresponse is random within some arbitrary subgroup of the population). Moreover, they show that approximately one-third to one-half of the difference in inequality measures between the survey and administrative data (tax records) is accounted for by nonresponse.

?? show how the latent income effect on compliance can be consistently estimated with the available data on average response rates by groups (for example, geographic areas) and the measured distribution of income across them. This strategy has been recently used with data of several countries (see **???**). This paper presents `kmr`, a new command in Stata to implement this method, and illustrates its use with an empirical example using the 2018 CPS data and state-level response rates.

The paper is organized as follows. Section 2 describes the methodology. Section 3 describes the `kmr` command. In Section 4 we illustrate the use of the command with the empirical example, and Section 5 concludes.

2 Methodology

As described in **??**, the proposed method has two main advantages: First, it does not assume that within the smallest subgroup the decision to respond is independent of income (ignorability assumption). Second, it relies only on the survey data and does not require any external information.

Here we sketch how the estimator is derived. We start by assuming that the probability of response denoted by $P(D_\epsilon = 1)$, where D_ϵ is an indicator function equal to 1 when the household ϵ responds, depends on a K-vector X_ϵ (i.e., $P(D_\epsilon = 1) = f(X_\epsilon)$). We observe the response rate for J groups together with the values of X for all the respondents, and the respondents can be divided in I groups according to the observed values of X indexed by $i \in I$.
For a given group $j \in J$, the mass of respondents with a given value of X equal to i denoted by m^1_{ij} and can be defined as:

$$m^1_{ij} = \int_0^{m_{ij}} D_{ij\epsilon} d\epsilon \tag{1}$$

where m_{ij} is the total (unobserved) number of households with value of X equal to i in group j. The expected value of m^1_{ij} is given by:

$$E[m^1_{ij}] = m_{ij} P(D_{ij} = 1) = m_{ij} P_i \tag{2}$$

where the last equality comes from the fact that the probability of response for a given value of X is the same across the J groups. Then we can construct a moment condition for group j as follows:

$$E[\sum_i \frac{m^1_{ij}}{P_i}] = \sum_i m_{ij} = m_j \tag{3}$$

where the right-hand side corresponds to the observed total mass of sampled households in group j. To complete the moment condition, we need to assume a functional form for P_i, which we assume to be a logistic function such that:

$$P_i = P(D_{ij\epsilon} = 1 | X_i, \theta) = \frac{e^{X_i'\theta}}{1 + e^{X_i'\theta}} \tag{4}$$

where θ is a K-vector of parameters.
Having set up the population moment condition for group j, we can define its respective sample moment condition as:

$$\psi_j(\theta) = \sum_i \frac{m^1_{ij}}{P_i} - m_j \tag{5}$$

Finally, the estimator is constructed by stacking the J sample moment conditions into $\Psi(\theta)$ to get an estimator for θ of the form:

$$\hat{\theta} = argmin_{\theta}\Psi(\theta)'W^{-1}\Psi(\theta) \tag{6}$$

where W is a positive definite weighting matrix. The $J \times J$ weighting matrix has off-diagonal elements equal to zero because of the assumption of independence of the response decisions of all households between the J groups. It is assumed that the variance of $\psi_j(\theta)$ for each group j is proportional to the mass of the sampled household population m_j, with a factor of proportionality σ^2 that can be ignored for the estimation.

The variance of the estimator $\hat{\theta}$ can be computed as follows:

$$\hat{Var}(\hat{\theta}) = \hat{\sigma}^2[\frac{\partial\Psi(\theta)'}{\partial\theta}W^{-1}\frac{\partial\Psi(\theta)}{\partial\theta}]^{-1} \tag{7}$$

with

$$\frac{\partial\psi_j(\theta)}{\partial\theta} = -\sum_i \frac{m_{ij}^1}{P_i^2}\frac{\partial P_i}{\partial\theta} = -\sum_i \frac{m_{ij}^1 X_i}{e^{X_i\theta}} \tag{8}$$

Alternatively, the variance can be computed using bootstrap by randomly sampling J groups with replacement and applying the estimator to each sample. After a given number of repetitions, the bootstrapped variance is computed as the average squared deviation of the bootstrapped estimates from the original estimate. This method is computationally intensive because it needs to solve the minimization problem again for each bootstrapped sample. Nevertheless, it can be easily implemented by using the commands `bsample` and `simulate`, as shown in the empirical example.

3 The kmr Command

3.1 Syntax

The syntax of the `kmr` command is

`kmr` [*varlist*] [*if*] [*in*] , `groups(`*varname*`) interview(`*varname*`)`
`nonresponse(`*varname*`)` [`noconstant sweights(`*varname*`)`
`generate(`*newvarname*`) graph(`*varname*`) technique(`*string*`) delta(`#`)`
`start(`#`) difficult maxiter(`#`)`]

where *varlist* includes the determinants of the response rate.

3.2 Options

`groups(`*varname*`)` is required and specifies a categorical variable representing the group identifiers (these are state identifiers in **??**). This variable can be a numerical or string variable.

`interview(`*varname*`)` is required and specifies the number of interviews obtained for each group.

`nonresponse(`*varname*`)` is required and specifies the number of nonresponses obtained for each group.

`nonconstant` suppresses the constant term.

`sweights(`*varname*`)` specifies the survey weights to be corrected and generates a new variable with the subscript "_c". The new variable contains corrected survey weights that are generated by multiplying the weights provided by the user to the inverse of the estimated probability of response. Ideally, the user would use weights before any unit nonresponse correction only. Unfortunately, these are not generally available in the public use files of standard survey data. Hence, users should be aware that the corrected weights will likely overestimate the total population if the weights used in the sweights option already have a form of unit nonresponse correction. To avoid this problem, users can easily construct and use a new set of uncorrected weights, as done in **??** and shown in the empirical example below.

`generate(`*newvarname*`)` specifies the name of a new variable to be created containing the predicted probability of response. In addition, two other variables with the same name plus the subscripts "_upper" and "_lower" are created. They contain, respectively, the upper and lower bounds of a 95% confidence interval for the predicted value.

`graph(`*varname*`)` generates a line graph of the predicted probability of response against *varname*.

`technique(`*string*`)` specifies the algorithm to use in the minimization problem. The default is "nr" (modified Newton-Raphson). The alternatives are "dfp" (Davidon-Fletcher-Powell), "bfgs" (Broyden-Fletcher-Golfarb-Shanno), "bhhh" (Berndt-Hall-Hall-Hausman), and "nm" (Nelder-Mead[1]).

`delta(#)` value of delta to be used for building the simplex required by the technique "nm". The default `delta` is set to 0.1.

`start(#)` row-vector with initial values for the parameters to start the algorithm. The default initial values are set to a vector of zeros.

`difficult` specifies that the criterion function is likely to be difficult to maximize because of nonconcave regions. The option `difficult` specifies that a different stepping algorithm be used in nonconcave regions (a mixture of steepest descent and Newton).

`maxiter(#)` sets the maximum number of iterations to be performed before the maximization is stopped. The default `maxiter` is set to 100.

1. The only non-gradient algorithm among the options. It is the algorithm used by Matlab's fminsearch command that **?** apply in the code made available from the authors.

3.3 Returned values

`kmr` saves the following in e():

Scalars

`e(aic)`	Akaike information criteria
`e(schwarz)`	Schwarz information criteria
`e(value)`	value of the function
`e(sigmavalue)`	value of sigma
`e(n)`	number of observations
`e(ngroups)`	number of groups

Macros

`e(cmdline)`	command line
`e(title)`	command title
`e(cmd)`	command name
`e(algorithm)`	algorithm used
`e(properties)`	properties

Matrices

`e(b)`	coefficient vector
`e(V)`	variance matrix of the estimates

Functions

`e(sample)`	marks estimation sample

In addition, the command optionally generates four new variables. The predicted probability of compliance, the upper and lower values of its 95% confidence interval, and corrected survey weights.

3.4 Dependency of kmr

`kmr` depends on the Mata function mm_collapse(), which is part of the moremata package (**?**). If not already installed, you can install it by typing ssc install moremata.

4 Empirical Example

To illustrate the use of the command, we use from the 2018 CPS data downloaded from IPUMS (**?**) merged to the number of interviews and type A nonresponses (interviewer finds the household's address but obtains no interviews) obtained from the NBER CPS Supplements website.[2] We estimate the compliance function using the following specification:

$$P_i = \frac{e^{\theta_0+\theta_1 log(y_i)}}{1+e^{\theta_0+\theta_1 log(y_i)}} \tag{9}$$

where y_i corresponds to log of total household gross income per capita in current dollars.[3]

We begin by loading the data set and looking at the state-level geographical variation in nonresponse rates in the United States. We can use the user-written command `maptile` to show these rates in a map:

2. https://www.nber.org/data/current-population-survey-data.html
3. Gross income is factor income plus all public and private monetary transfers.

```
. use cps2018.dta, clear
. preserve
. gen nonresponse = 100*typea/(typea+interview)
. collapse nonresponse, by(statefip)
. ren statefip statefips
. maptile nonresponse, geo(state) geoid(statefips) fcolor(Greys2) ///
>  legdecimals(1) nquantiles(10)
. restore
```

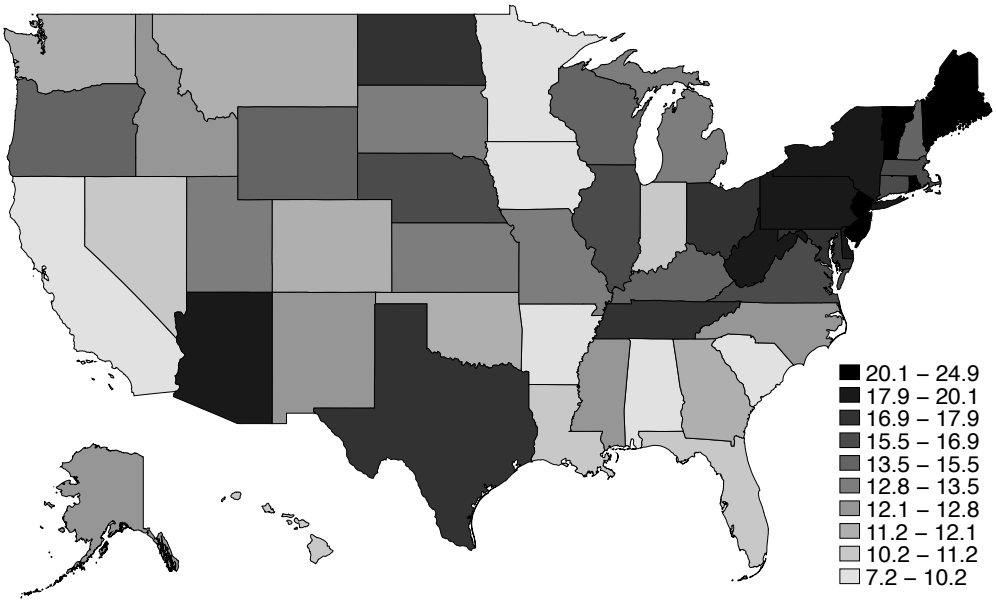

Figure 1: Nonresponse rates

Now we can create the regressors and two sets of weights that have no correction for nonresponse. The first one corresponds to using the raw data (in other words, no weights or weights equal 1). The second one assumes equal weights within states ("grossed-up" weights by state), in which weights are constructed by dividing the population of each state (as derived by summing the official CPS weights) by the number of respondents:[4]

```
. gen ly = log(hhincome_pc)
. gen ly2 = ly^2
. by statefip: egen state_population = sum(asecwth)
. gen weights_1 = 1
. gen weights_2 = state_population/interview
```

We can estimate the probability of response as a function of the log of total household

4. Alternatively, we can construct uncorrected weights by dividing the sum of respondents and nonrespondents by the number of respondents, but this would make little difference.

gross income per capita, produce a line graph of it together with its 95% confidence interval, and generate a set of corrected weights called "weights_1_c":[5]

```
. mat init = -.9,12
. kmr ly, groups(statefip) i(interview) n(typea) gen(P) sweights(weights_1) start(init)
Iteration 0:   f(p) =  636.90409  (not concave)
Iteration 1:   f(p) =  242.52283  (not concave)
Iteration 2:   f(p) =  155.56547  (not concave)
Iteration 3:   f(p) =  151.69864
Iteration 4:   f(p) =  151.41533
Iteration 5:   f(p) =  151.23525  (not concave)
Iteration 6:   f(p) =  151.23214  (not concave)
Iteration 7:   f(p) =  151.23182
Iteration 8:   f(p) =  151.23105
Iteration 9:   f(p) =  151.23105

Compliance function                             Number of obs      =    66899
                                                AIC                =    59.44
Number of groups   =   51                       Schwarz            =  56.8224
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          ly |  -.9866157   .2845149    -3.47   0.001    -1.544255   -.4289768
       _cons |   12.16841    3.11691     3.90   0.000      6.05938    18.27744
------------------------------------------------------------------------------

. ereturn list
scalars:
               e(value) =  151.2310513467116
          e(sigmavalue) =  4.8230144407221
                 e(aic) =  59.43614197528939
            e(schwarz) =  56.82243633640928
                  e(n) =  66899
            e(ngroups) =  51
macros:
            e(cmdline) : "kmr ly, groups(statefip) i(interview) n(typea) gen(P) "
              e(title) : "Compliance function estimate using group´s response rates"
                e(cmd) : "kmr"
          e(technique) : "nr"
         e(properties) : "b V"
matrices:
                  e(b) :  1 x 2
                  e(V) :  2 x 2
functions:
             e(sample)
. sort ly
. line P P_upper P_lower ly, ytitle("Probability of response") ///
>  xtitle("Log(income per capita)") lpattern("1" "-" "-") ///
>  lcolor("black" "black" "black") ///
>  legend(order(1 "Point estimate" 2 "95% CI")) scheme(s1color)
```

Now we try a different specification that adds the squared log of income per capita as a second regressor. This will help to capture the fact that high nonresponse rates

5. Note that we correct the unitary weights. To compute total population or total income, we should multiply the corrected weights by the ratio between the country population and the sampled households (interviews + nonresponses).

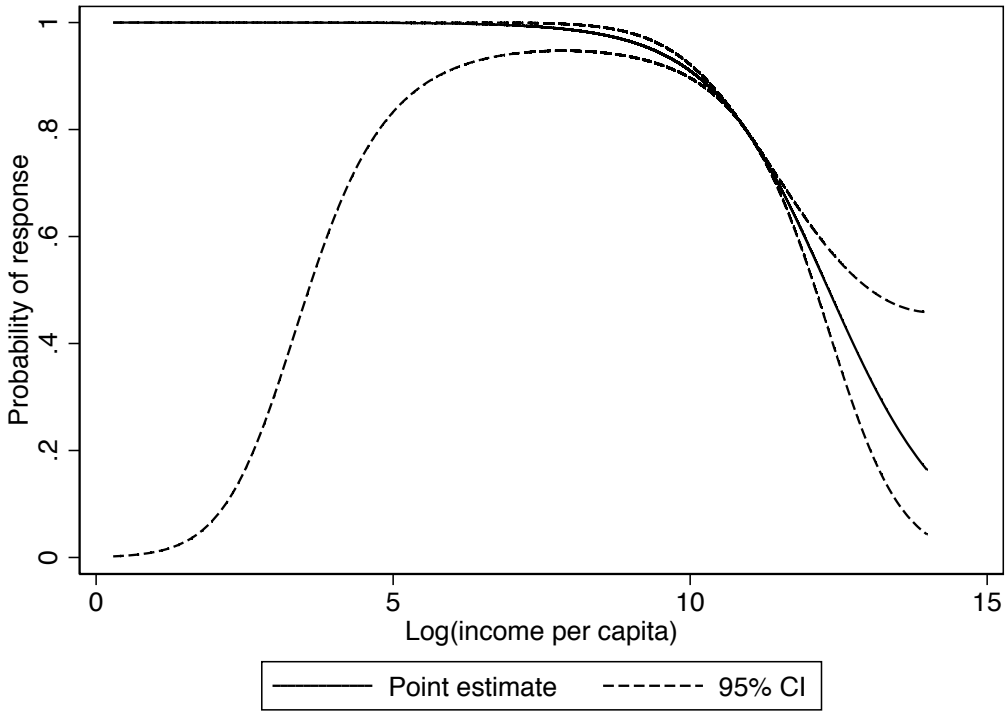

Figure 2: Compliance function

may occur in both tails of the income distribution, not just among rich households, as documented in **?**.

The inclusion of the log of income squared does capture some non-linearity of compliance with respect to income. However, the estimates appear to be less precisely estimated (two of the coefficient's p-values are below 5% confidence level). Moreover, the Akaike criterion suggests that the linear specification is preferable.

```
. kmr ly ly2, groups(statefip) i(interview) n(typea) gen(P2) difficult
Iteration 0:   f(p) =  39738.763
Iteration 1:   f(p) =  386.13037  (not concave)
Iteration 2:   f(p) =  196.55339  (not concave)
Iteration 3:   f(p) =  191.69224  (not concave)
Iteration 4:   f(p) =  189.12821  (not concave)
Iteration 5:   f(p) =  180.71912
Iteration 6:   f(p) =  180.15309
Iteration 7:   f(p) =  171.19227  (not concave)
Iteration 8:   f(p) =  168.25725
Iteration 9:   f(p) =  167.49508  (not concave)
Iteration 10:  f(p) =  157.66439
Iteration 11:  f(p) =  155.69955  (not concave)
```

```
Iteration 12:  f(p) =  151.74549  (not concave)
Iteration 13:  f(p) =  150.88544
Iteration 14:  f(p) =  150.76561  (not concave)
Iteration 15:  f(p) =  150.31748
Iteration 16:  f(p) =  150.26733  (not concave)
Iteration 17:  f(p) =   149.6072  (not concave)
Iteration 18:  f(p) =  149.55878  (not concave)
Iteration 19:  f(p) =  149.31096
Iteration 20:  f(p) =  149.25587
Iteration 21:  f(p) =  149.24739  (not concave)
Iteration 22:  f(p) =  149.24666
Iteration 23:  f(p) =  149.24614
Iteration 24:  f(p) =  149.24614

Compliance function                               Number of obs      =    66899
                                                  AIC                =    60.76
Number of groups   =   51                         Schwarz            =  58.0582
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          ly |    1.653704   .9867161     1.68   0.094    -.2802243    3.587632
         ly2 |     -.11918   .0467946    -2.55   0.011    -.2108959   -.0274642
       _cons |   -2.319921   5.306909    -0.44   0.662    -12.72127    8.081429
------------------------------------------------------------------------------

. gen weights_1_c2 = weights_1/P2

. ereturn list

scalars:
              e(value) =  149.2461430077821
         e(sigmavalue) =  4.847966701583557
                e(aic) =  60.76233511274758
            e(schwarz) =  58.05817197875191
                  e(n) =  66899
            e(ngroups) =  51

macros:
             e(cmdline) : "kmr ly ly2, groups(statefip) i(interview) n(typea) gen(P2) "
               e(title) : "Compliance function estimate using group´s response rates"
                 e(cmd) : "kmr"
           e(technique) : "nr"
          e(properties) : "b V"

matrices:
                   e(b) :  1 x 3
                   e(V) :  3 x 3

functions:
              e(sample)

. line P2 P2_upper P2_lower ly, ytitle("Probability of response") ///
>  xtitle("Log(income per capita)") lpattern("1" "-" "-") ///
>  lcolor("black" "black" "black") ///
>  legend(order(1 "Point estimate" 2 "95% CI")) scheme(s1color)
```

As we mention at the end of section 2, we can also compute the standard errors using bootstrap. We do so by defining a small program called kmrboot that resamples states with replacement, and estimates the compliance function for each new sample. This program is then called one thousand times by the command `simulate`, which stores the estimated coefficients in each repetition.

```
. * Estimating the compliance function and storing the coefficients
```

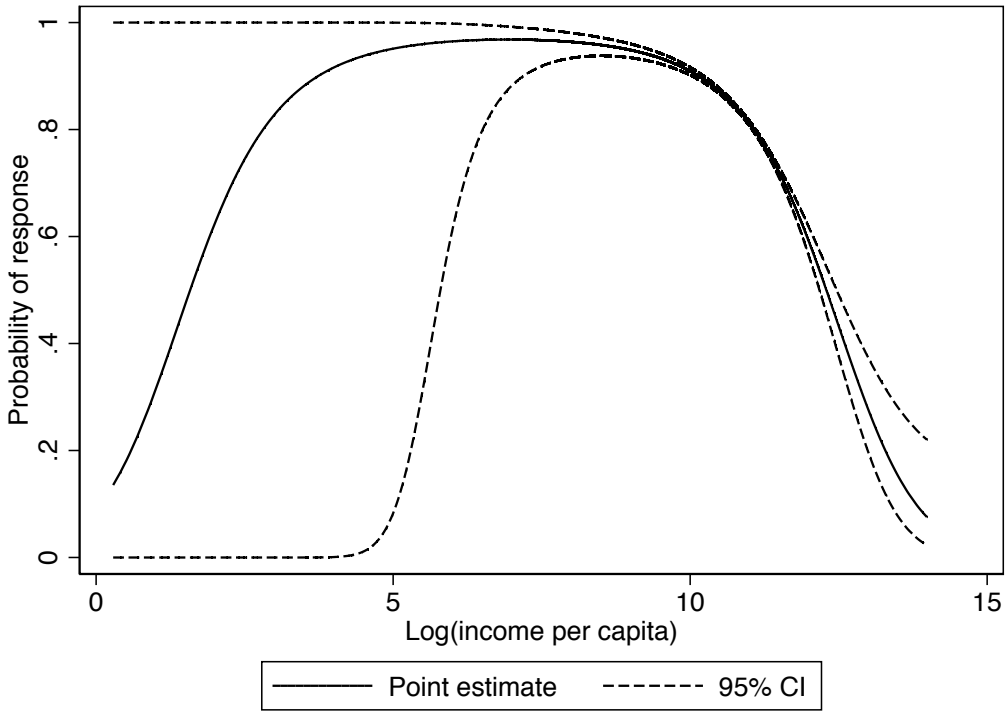

Figure 3: Compliance function quadratic on log(income)

```
. mat init = -.9,12
. quietly kmr ly, groups(statefip) i(interview) n(typea) start(init)
. matrix b = e(b)
.
. * Re-sampling clusters with replacement
. capture program drop kmrboot
. program define kmrboot, rclass
  1.  preserve
  2.   bsample, cluster(statefip) idcluster(newstatefip)
  3.   quietly kmr ly, groups(newstatefip) i(interview) n(typea) start(b)
  4.   return scalar ly = e(b)[1,1]
  5.   return scalar _cons = e(b)[1,2]
  6.  restore
  7. end
.
. * Repeat the re-sampling a thousand times
. preserve
. simulate ly = r(ly) _cons = r(_cons), reps(1000) seed(1) nodots: kmrboot
      command:  kmrboot
           ly:  r(ly)
```

```
       _cons:  r(_cons)

.
. * Analyze the results
. bstat, stat(b) n(1000)
Bootstrap results                                 Number of obs     =      1,000
                                                  Replications      =        999

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          ly |   -.986617   .3232507    -3.05   0.002    -1.620177   -.3530572
       _cons |   12.16843   3.450679     3.53   0.000     5.405219    18.93163
------------------------------------------------------------------------------
Note: One or more parameters could not be estimated in some of the bootstrap
      replicates; standard-error estimates include only complete replications.

.
. * Store confidence intervals
. mat lb = r(table)[5,1..2]
. mat ub = r(table)[6,1..2]
. restore
```

From the results of the bootstrap exercise, we find a level of uncertainty surrounding the parameter estimates that is similar to the standard errors previously reported, although the confidence interval is slightly wider.

Finally, we can use the user-written command `fastgini` to compute the Gini coefficients using the following alternatives: CPS sample weights, raw data, "grossed-up" by state, and the *kmr* corrected weights according to the specification that uses log of income:

```
. mat ginis = J(5,3,.)
. qui fastgini hhincome_pc [w = asecwth]
. mat ginis[1,1] = r(gini)
. qui fastgini hhincome_pc [w = weights_1]
. mat ginis[2,1] = r(gini)
. qui fastgini hhincome_pc [w = weights_2]
. mat ginis[3,1] = r(gini)
. qui fastgini hhincome_pc [w = weights_1_c]
. mat ginis[4,1] = r(gini)
. qui fastgini hhincome_pc [w = 1/P_lower]
. mat ginis[4,2] = r(gini)
. qui fastgini hhincome_pc [w = 1/P_upper]
. mat ginis[4,3] = r(gini)
.
. * Create probability of response according to bootstrap results
. g p_boot_upper = invlogit(ly*ub[1,1]+ub[1,2])
. g p_boot_lower = invlogit(ly*lb[1,1]+lb[1,2])
.
. mat ginis[5,1] = ginis[4,1]
. qui fastgini hhincome_pc [w = 1/p_boot_lower]
```

```
. mat ginis[5,2] = r(gini)
. qui fastgini hhincome_pc [w = 1/p_boot_upper]
. mat ginis[5,3] = r(gini)
. mat colnames ginis = "Point" "Lower" "Upper"
. mat rownames ginis = "ASEC" "Raw" "Grossed-up" "kmr" "kmrboot"
. matlist ginis
```

	Point	Lower	Upper
ASEC	.4652877	.	.
Raw	.4652385	.	.
Grossed-up	.4645361	.	.
kmr	.5051071	.5988177	.4799021
kmrboot	.5051071	.577905	.4652385

In our exercise we derive a range of values for our corrected Gini coefficient using the point estimates of the compliance function and its upper and lower bounds derived from the 95% confidence interval (row "kmr", columns "Point", "Lower", and "Upper" respectively). We then do the same for the range of values of the compliance function derived from the bootstrap exercise (row "kmrboot"). We prefer this "range of values" approach to the use of standard errors obtained via the `fastgini` command alone, on the grounds that the latter would leave out a substantial source of uncertainty originating from the adjustments of the original weights.[6]

We can summarize the results as follows: the use of sample weights does not significantly change the Gini coefficient compared to the use of unweighted data (comparing the two rows, "ASEC" and "Raw"); the proposed method of weight adjustment increases the estimated Gini coefficient by at least 8.6%, going from an uncorrected Gini of 0.465 to 0.505 (comparing the two rows, "ASEC" and "kmr"). The uncertainty associated with the estimation of the compliance function is non-negligible and it does not change significantly when the standard errors are computed using the bootstrap method (comparing the two columns "Lower" and "Upper").

5 Concluding remarks

Unit nonresponse in household surveys could lead to biases in inequality and poverty measurement. The typical methods to correct survey weights for unit nonresponse assume ignorability within some arbitrary subgroup of the population, which recent empirical evidence suggests may not hold in the case of household survey data.

In this article, we presented the command `kmr`, which is designed to implement the econometric method introduced by **?** to estimate a survey compliance function using group level nonresponse rates, allowing us to relax the ignorability assumption.

6. For instance, using jackknife procedure without taking into consideration the uncertainty associated with the estimated weights, we get a much narrower confidence interval for the Gini coefficient (0.496;0.514).

6 Acknowledgments

We thank Carolyn Fisher for comments on the draft and Anton Korinek for providing a Matlab code with the data set used in their paper, which greatly facilitated this project. We are also grateful for the insightful and critical comments received by two anonymous referees which pushed us to improve the structure of the paper.

About the authors

Ercio Munoz is Ph.D. Candidate in Economics and Research Associate at the Stone Center on Socio-Economic Inequality at the Graduate Center in the City University of New York.

Salvatore Morelli is Core Faculty at Stone Center on Socio-Economic Inequality at the Graduate Center in the City University of New York.