

# Poverty and Inequality with Complex Survey Data

Guilherme Jacob, Anthony Damico, and Djalma Pessoa

2021-04-05



# Contents



# Chapter 1

## Introduction

The R `convey` library estimates measures of poverty, inequality, and wellbeing. There are two other R libraries covering this subject, `vardpoor` (?) and `laeken` (?), however, only `convey` integrates seamlessly with the R survey package (???).

`convey` is free and open-source software that runs inside the R environment for statistical computing. Anyone can review and propose changes to the source code for this software. Readers are welcome to propose changes to this book as well.

### 1.1 Installation

In order to work with the `convey` library, you will need to have R running on your machine. If you have never used R before, you will need to install that software before `convey` can be accessed. Check out `FlowingData` for a concise list of resources for new R users. Once you have R loaded on your machine, you can install..

- the latest released version from CRAN with

```
install.packages("convey")
```

- the latest development version from github with

```
devtools::install_github("djalmapessoa/convey")
```

In order to know how to cite this package, run `citation("convey")`.

## 1.2 Complex surveys and statistical inference

In this book, we demonstrate how to measure poverty and income concentration in a population based on microdata collected from a complex survey sample. Most surveys administered by government agencies or larger research organizations utilize a sampling design that violates the assumption of simple random sampling (SRS), including:

1. Different units selection probabilities;
2. Clustering of units;
3. Stratification of clusters;
4. Reweighting to compensate for missing values and other adjustments.

Therefore, basic unweighted R commands such as `mean()` or `glm()` will not properly account for the weighting nor the measures of uncertainty (such as the confidence intervals) present in the dataset. For some examples of publicly-available complex survey data sets, see <http://asdfree.com>.

Unlike other software, the R `convey` package does not require that the user specify these parameters throughout the analysis. So long as the `svydesign` object or `svrepdesign` object has been constructed properly at the outset of the analysis, the `convey` package will incorporate the survey design automatically and produce statistics and variances that take the complex sample into account.

## 1.3 Usage Examples

In the following example, we've loaded the data set `eusilc` from the R library `laeken` (?).

```
library(laeken)
data(eusilc)
```

Next, we create an object of class `survey.design` using the function `svydesign` of the library `survey`:

```
library(survey)
des_eusilc <- svydesign(ids = ~rb030, strata = ~db040, weights = ~rb050, data = eusilc)
```

Right after the creation of the design object `des_eusilc`, we should use the function `convey_prep` that adds an attribute to the survey design which saves information on the design object based upon the whole sample, needed to work with subset designs.

```
library(convey)
des_eusilc <- convey_prep( des_eusilc )
```

To estimate the at-risk-of-poverty rate, we use the function `svyarpt`:

```
svyarpr(~eqIncome, design=des_eusilc)
```

```
      arpr      SE
eqIncome 0.14444 0.0028
```

To estimate the at-risk-of-poverty rate across domains defined by the variable db040 we use:

```
svyby(~eqIncome, by = ~db040, design = des_eusilc, FUN = svyarpr, deff = FALSE)
```

|               | db040         | eqIncome  | se          |
|---------------|---------------|-----------|-------------|
| Burgenland    | Burgenland    | 0.1953984 | 0.017202852 |
| Carinthia     | Carinthia     | 0.1308627 | 0.010606502 |
| Lower Austria | Lower Austria | 0.1384362 | 0.006513217 |
| Salzburg      | Salzburg      | 0.1378734 | 0.011581408 |
| Styria        | Styria        | 0.1437464 | 0.007453192 |
| Tyrol         | Tyrol         | 0.1530819 | 0.009884094 |
| Upper Austria | Upper Austria | 0.1088977 | 0.005933094 |
| Vienna        | Vienna        | 0.1723468 | 0.007684540 |
| Vorarlberg    | Vorarlberg    | 0.1653731 | 0.013756389 |

Using the same data set, we estimate the quintile share ratio:

```
# for the whole population
svyqsr(~eqIncome, design=des_eusilc, alpha1= .20)
```

```
      qsr      SE
eqIncome 3.97 0.0426
```

```
# for domains
svyby(~eqIncome, by = ~db040, design = des_eusilc,
      FUN = svyqsr, alpha1= .20, deff = FALSE)
```

|               | db040         | eqIncome | se         |
|---------------|---------------|----------|------------|
| Burgenland    | Burgenland    | 5.008486 | 0.32755685 |
| Carinthia     | Carinthia     | 3.562404 | 0.10909726 |
| Lower Austria | Lower Austria | 3.824539 | 0.08783599 |
| Salzburg      | Salzburg      | 3.768393 | 0.17015086 |
| Styria        | Styria        | 3.464305 | 0.09364800 |
| Tyrol         | Tyrol         | 3.586046 | 0.13629739 |
| Upper Austria | Upper Austria | 3.668289 | 0.09310624 |
| Vienna        | Vienna        | 4.654743 | 0.13135731 |
| Vorarlberg    | Vorarlberg    | 4.366511 | 0.20532075 |

These functions can be used as S3 methods for the classes `survey.design` and `svyrep.design`.

Let's create a design object of class `svyrep.design` and run the function `convey_prep` on it:

```
des_eusilc_rep <- as.svrepdesign(des_eusilc, type = "bootstrap")
des_eusilc_rep <- convey_prep(des_eusilc_rep)
```

and then use the function `svyarpr`:

```
svyarpr(~eqIncome, design=des_eusilc_rep)
```

```
      arpr      SE
eqIncome 0.14444 0.0025
```

```
svyby(~eqIncome, by = ~db040, design = des_eusilc_rep, FUN = svyarpr, deff = FALSE)
```

```
      db040 eqIncome se.eqIncome
Burgenland      Burgenland 0.1953984 0.016713791
Carinthia       Carinthia 0.1308627 0.012061625
Lower Austria   Lower Austria 0.1384362 0.007294696
Salzburg        Salzburg 0.1378734 0.010050357
Styria          Styria 0.1437464 0.008558783
Tyrol           Tyrol 0.1530819 0.010328225
Upper Austria   Upper Austria 0.1088977 0.006212301
Vienna          Vienna 0.1723468 0.007259732
Vorarlberg      Vorarlberg 0.1653731 0.012792618
```

The functions of the library `convey` are called in a similar way to the functions in library `survey`.

It is also possible to deal with missing values by using the argument `na.rm`.

```
# survey.design using a variable with missings
svygini( ~ py010n , design = des_eusilc )
```

```
      gini SE
py010n   NA NA
```

```
svygini( ~ py010n , design = des_eusilc , na.rm = TRUE )
```

```
      gini      SE
py010n 0.64606 0.0036
```

```
# svyrep.design using a variable with missings
svygini( ~ py010n , design = des_eusilc_rep )
```

```
      gini SE
py010n   NA NA
```

```
svygini( ~ py010n , design = des_eusilc_rep , na.rm = TRUE )
```

```
      gini      SE
py010n 0.64606 0.0043
```



## 1.4 Underlying Calculations

In what follows, we often use the linearization method as a tool to produce an approximation for the variance of an estimator. From the linearized variable  $z$  of an estimator  $T$ , we get from the expression (??) an estimate of the variance of  $T$

If  $T$  can be expressed as a function of the population totals  $T = g(Y_1, Y_2, \dots, Y_n)$ , and if  $g$  is linear, the estimation of the variance of  $T = g(Y_1, Y_2, \dots, Y_n)$  is straightforward. If  $g$  is not linear but is a ‘smooth’ function, then it is possible to approximate the variance of  $g(Y_1, Y_2, \dots, Y_n)$  by the variance of its first order Taylor expansion. For example, we can use Taylor expansion to linearize the ratio of two totals. However, there are situations where Taylor linearization cannot be immediately possible, either because  $T$  cannot be expressed as functions of the population totals, or because  $g$  is not a **smooth** function. An example is the case where  $T$  is a quantile.

In these cases, it might work an alternative form of linearization of  $T$ , by **Influence Function**, as defined in (??), proposed in ?. Also, it could be used replication methods such as **bootstrap** and **jackknife**.

In the **convey** library, there are some basic functions that produce the linearized variables needed to measure income concentration and poverty. For example, looking at the income variable in some complex survey dataset, the **quantile** of that income variable can be linearized by the function **convey::svyiqalpha** and the sum total below any quantile of the variable is linearized by the function **convey::svyisq**.

From the linearized variables of these basic estimates, it is possible by using rules of composition, valid for influence functions, to derive the influence function of more complex estimates. By definition the influence function is a Gateaux derivative and the rules of composition valid for Gateaux derivatives also hold for Influence Functions.

The following property of Gateaux derivatives was often used in the library **convey**. Let  $g$  be a differentiable function of  $m$  variables. Suppose we want to compute the influence function of the estimator  $g(T_1, T_2, \dots, T_m)$ , knowing the Influence function of the estimators  $T_i, i = 1, \dots, m$ . Then the following holds:

$$I(g(T_1, T_2, \dots, T_m)) = \sum_{i=1}^m \frac{\partial g}{\partial T_i} I(T_i)$$

In the library **convey** this rule is implemented by the function **contrastinf** which uses the R function **deriv** to compute the formal partial derivatives  $\frac{\partial g}{\partial T_i}$ .

For example, suppose we want to linearize the **Relative median poverty gap**(**rmpg**), defined as the difference between the at-risk-of-poverty threshold (**arpt**) and the median of incomes less than the **arpt** relative to the **arprt**:

$$rmpg = \frac{arpt - medpoor}{arpt}$$

where `medpoor` is the median of incomes less than `arpt`.

Suppose we know how to linearize `arpt` and `medpoor`, then by applying the function `contrastinf` with

$$g(T_1, T_2) = \frac{(T_1 - T_2)}{T_1}$$

we linearize the `rmpg`.

## 1.5 The Variance Estimator

Using the notation in [?](#), the variance of the estimator  $T(\hat{M})$  can approximated by:

$$Var [T(\hat{M})] \cong var \left[ \sum_s w_i z_i \right] \quad (1.1)$$

The `linearized` variable  $z$  is given by the derivative of the functional:

$$z_k = \lim_{t \rightarrow 0} \frac{T(M + t\delta_k) - T(M)}{t} = IT_k(M) \quad (1.2)$$

where,  $\delta_k$  is the Dirac measure in  $k$ :  $\delta_k(i) = 1$  if and only if  $i = k$ .

This **derivative** is called **Influence Function** and was introduced in the area of **Robust Statistics**.

## 1.6 Influence Functions

Some measures of poverty and income concentration are defined by non-differentiable functions so that it is not possible to use Taylor linearization to estimate their variances. An alternative is to use **Influence functions** as described in [?](#) and [?](#). The `convey` library implements this methodology to work with `survey.design` objects and also with `svyrep.design` objects.

Some examples of these measures are:

- At-risk-of-poverty threshold:  $arpt = .60q_{.50}$  where  $q_{.50}$  is the income median;
- At-risk-of-poverty rate  $arpr = \frac{\sum_U 1(y_i \leq arpt)}{N} .100$
- Quintile share ratio

$$qsr = \frac{\sum_U 1(y_i > q_{.80})}{\sum_U 1(y_i \leq q_{.20})}$$

- Gini coefficient  $1 + G = \frac{2 \sum_U (r_i - 1) y_i}{N \sum_U y_i}$  where  $r_i$  is the rank of  $y_i$ .

Note that it is not possible to use Taylor linearization for these measures because they depend on quantiles and the Gini is defined as a function of ranks. This could be done using the approach proposed by Deville (1999) based upon influence functions.

Let  $U$  be a population of size  $N$  and  $M$  be a measure that allocates mass one to the set composed by one unit, that is  $M(i) = M_i = 1$  if  $i \in U$  and  $M(i) = 0$  if  $i \notin U$

Now, a population parameter  $\theta$  can be expressed as a functional of  $M$   $\theta = T(M)$

Examples of such parameters are:

- Total:  $Y = \sum_U y_i = \sum_U y_i M_i = \int y dM = T(M)$
- Ratio of two totals:  $R = \frac{Y}{X} = \frac{\int y dM}{\int x dM} = T(M)$
- Cumulative distribution function:  $F(x) = \frac{\sum_U 1(y_i \leq x)}{N} = \frac{\int 1(y \leq x) dM}{\int dM} = T(M)$

To estimate these parameters from the sample, we replace the measure  $M$  by the estimated measure  $\hat{M}$  defined by:  $\hat{M}(i) = \hat{M}_i = w_i$  if  $i \in s$  and  $\hat{M}(i) = 0$  if  $i \notin s$ .

The estimators of the population parameters can then be expressed as functional of the measure  $\hat{M}$ .

- Total:  $\hat{Y} = T(\hat{M}) = \int y d\hat{M} = \sum_s w_i y_i$
- Ratio of totals:  $\hat{R} = T(\hat{M}) = \frac{\int y d\hat{M}}{\int x d\hat{M}} = \frac{\sum_s w_i y_i}{\sum_s w_i x_i}$
- Cumulative distribution function:  $\hat{F}(x) = T(\hat{M}) = \frac{\int 1(y \leq x) d\hat{M}}{\int d\hat{M}} = \frac{\sum_s w_i 1(y_i \leq x)}{\sum_s w_i}$

## 1.7 Influence Function Examples

- Total:

$$\begin{aligned} IT_k(M) &= \lim_{t \rightarrow 0} \frac{T(M + t\delta_k) - T(M)}{t} \\ &= \lim_{t \rightarrow 0} \frac{\int y.d(M + t\delta_k) - \int y.dM}{t} \\ &= \lim_{t \rightarrow 0} \frac{\int yd(t\delta_k)}{t} = y_k \end{aligned}$$

- Ratio of two totals:

$$\begin{aligned} IR_k(M) &= I\left(\frac{U}{V}\right)_k(M) = \frac{V(M) \times IU_k(M) - U(M) \times IV_k(M)}{V(M)^2} \\ &= \frac{Xy_k - Yx_k}{X^2} = \frac{1}{X}(y_k - Rx_k) \end{aligned}$$

## 1.8 Examples of Linearization Using the Influence Function

- At-risk-of-poverty threshold:

$$arpt = 0.6 \times m$$

where  $m$  is the median income.

$$z_k = -\frac{0.6}{f(m)} \times \frac{1}{N} \times [I(y_k \leq m - 0.5)]$$

- At-risk-of-poverty rate:

$$arpr = \frac{\sum_U I(y_i \leq t)}{\sum_U w_i} .100$$

$$z_k = \frac{1}{N} [I(y_k \leq t) - t] - \frac{0.6}{N} \times \frac{f(t)}{f(m)} [I(y_k \leq m) - 0.5]$$

where:

$N$  - population size;

$t$  - at-risk-of-poverty threshold;

$y_k$  - income of person  $k$ ;

$m$  - median income;

$f$  - income density function;

## 1.9 Replication Designs

All major functions in the library `convey` have S3 methods for the classes: `survey.design`, `svyrep.design` and `DBIdesign`. When the argument `design` is

a survey design with replicate weights created by the library `survey`, `convey` uses the method `svrepdesign`.

Considering the remarks in (?), p. 163, concerning the deficiency of the **Jackknife** method in estimating the variance of **quantiles**, we adopted the type bootstrap instead.

The function **bootVar** from the library **laeken** , (?), also uses the bootstrap method to estimate variances.

## 1.10 Decomposition

Some inequality and multidimensional poverty measures can be decomposed. As of December 2016, the decomposition methods in **convey** are limited to group decomposition.

For instance, the generalized entropy index can be decomposed into between and within group components. This sheds light on a very simple question: of the overall inequality, how much can be explained by inequalities between groups and within groups? Since this measure is additive decomposable, one can get estimates of the coefficients, SEs and covariance between components. For a more practical approach, see (?).

The Alkire-Foster class of multidimensional poverty indices can be decomposed by dimension and groups. This shows how much each group (or dimension) contribute to the overall poverty.

This technique can help understand where and who is more affected by inequality and poverty, contributing to more specific policy and economic analysis.



## Chapter 2

# Poverty Indices

Poverty is an issue discussed since long time ago. As ? points out, Aristotle and Confucius discussed ideas about poverty. In fact, Aristotle’s ideas influenced Thomas Aquinas, one of the pillars of Western philosophy. Since then, societies changed, modifying the theories of justice underlying the idea of poverty.

As the concept and the ethics towards poverty change, so does its measurement. From basic measures like the headcount rate to more complex metrics, such as the FGT index, poverty measurement evolved. Nowadays, poverty measures estimates are calculated using household surveys and censuses (?). Yet, only recently the aspects of statistical inference combining such measures and survey designs were explored<sup>1</sup>. These advances become even more important given the recent efforts in poverty mapping, an analytical method that combined poverty analysis and small area estimation, like ? and ?.

The following subsections shows how poverty estimates and their sampling errors can be estimated using simple commands from the `convey` package.

### 2.1 At Risk of Poverty Threshold (svyarpt)

The at-risk-of-poverty threshold (ARPT) is a measure used to define the people whose incomes imply a low standard of living in comparison to the general living standards. I.e., even though some people are not below the effective poverty line, those below the ARPT can be considered “almost deprived”.

This measure is defined as 0.6 times the median income for the entire population:

$$arpt = 0.6 \times median(y),$$

---

<sup>1</sup>For instance, see ?, ?, ?, and ?.

where,  $y$  is the income variable and `median` is estimated for the whole population. The details of the linearization of the `arpt` are discussed by ? and ?.

---

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a `arpt` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the laeken library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the arpt coefficient
# using the R vardpoor library
varpoord_arpt_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",
```



```

        # row number variable
        ID_level2 = "IDd",

        # strata variable
        H = "db040",

        N_h = NULL ,

        # clustering variable
        PSU = "rb030",

        # data.table
        dataset = dati,

        # arpt coefficient function
        type = "linarpt",

        # poverty threshold range
        order_quant = 50L ,

        # get linearized variable
        outp_lin = TRUE
    )

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_arpt_calculation$all_result$value

## [1] 10859.24
coef( svyarpt( ~ eqincome , des_eusilc ) )

## eqincome

```

```
## 10859.24
# linearized variables do match
# vardpoor
lin_arpt_varpoord<- varpoord_arpt_calculation$lin_out$lin_arpt
# convey
lin_arpt_convey <- attr(svyarpt( ~ eqincome , des_eusilc ), "lin")

# check equality
all.equal(lin_arpt_varpoord, lin_arpt_convey )

## [1] TRUE
# variances do not match exactly
attr( svyarpt( ~ eqincome , des_eusilc ) , 'var' )

##          eqincome
## eqincome 2564.027
varpoord_arpt_calculation$all_result$var

## [1] 2559.442
# standard errors do not match exactly
varpoord_arpt_calculation$all_result$se

## [1] 50.59093
SE( svyarpt( ~ eqincome , des_eusilc ) )

##          eqincome
## eqincome 50.63622
```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svyarpt` and `vardpoor::linarpt` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```
# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )
```

```

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svyarpt( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' )

##           eqincome
## eqincome 2559.442
varpoord_arpt_calculation$all_result$var

## [1] 2559.442

# matches
varpoord_arpt_calculation$all_result$se

## [1] 50.59093
SE( svyarpt( ~ eqincome , des_eusilc_ultimate_cluster ) )

##           eqincome
## eqincome 50.59093

```

For additional usage examples of `svyarpt`, type `?convey::svyarpt` in the R console.

## 2.2 At Risk of Poverty Ratio (svyarpr)

The at-risk-of-poverty rate (ARPR) is the share of persons with an income below the at-risk-of-poverty threshold (`arpt`). The logic behind this measure is that although most people below the ARPT cannot be considered “poor”, they are the ones most vulnerable to becoming poor in the event of a negative economic phenomenon.

The ARPR is a composite estimate, taking into account both the sampling error in the proportion itself and that in the ARPT estimate. The details of the linearization of the `arpr` and are discussed by ? and ?.

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a ARPR coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the vardpoor library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the arpr coefficient
# using the R vardpoor library
varpoord_arpr_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
    ID_level2 = "IDd",
```

```

      # strata variable
      H = "db040",

      N_h = NULL ,

      # clustering variable
      PSU = "rb030",

      # data.table
      dataset = dati,

      # arpr coefficient function
      type = "linarpr",

      # poverty threshold range
      order_quant = 50L ,

      # get linearized variable
      outp_lin = TRUE
    )

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_arpr_calculation$all_result$value

## [1] 14.44422
coef( svyarpr( ~ eqincome , des_eusilc ) ) * 100

## eqincome
## 14.44422

```

```
# linearized variables do match
# vardpoor
lin_arpr_varpoord<- varpoord_arpr_calculation$lin_out$lin_arpr
# convey
lin_arpr_convey <- attr(svyarpr( ~ eqincome , des_eusilc ), "lin")

# check equality
all.equal(lin_arpr_varpoord, 100*lin_arpr_convey )
```

```
## [1] "Mean relative difference: 0.2264738"
```

```
# variances do not match exactly
attr( svyarpr( ~ eqincome , des_eusilc ) , 'var' ) * 10000
```

```
##           eqincome
## eqincome 0.07599778
varpoord_arpr_calculation$all_result$var
```

```
## [1] 0.08718569
```

```
# standard errors do not match exactly
varpoord_arpr_calculation$all_result$se
```

```
## [1] 0.2952722
SE( svyarpr( ~ eqincome , des_eusilc ) ) * 100
```

```
##           eqincome
## eqincome 0.2756769
```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svyarpr` and `vardpoor::linarpr` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```
# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
```

```

# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svyarpr( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' ) * 10000

##           eqincome
## eqincome 0.07586194
varpoord_arpr_calculation$all_result$var

## [1] 0.08718569
# matches
varpoord_arpr_calculation$all_result$se

## [1] 0.2952722
SE( svyarpr( ~ eqincome , des_eusilc_ultimate_cluster ) ) * 100

##           eqincome
## eqincome 0.2754305

```

For additional usage examples of `svyarpr`, type `?convey::svyarpr` in the R console.

## 2.3 Relative Median Income Ratio (svyrmir)

The relative median income ratio (`rmir`) is the ratio of the median income of people aged above a value (65) to the median of people aged below the same value. In mathematical terms,

$$rmir = \frac{\text{median}\{y_i; \text{age}_i > 65\}}{\text{median}\{y_i; \text{age}_i \leq 65\}}.$$

The details of the linearization of the `rmir` and are discussed by ? and ?.

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a `rmir` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the vardpoor library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the rmir coefficient
# using the R vardpoor library
varpoord_rmir_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
    ID_level2 = "IDd",

    # strata variable
```



```

        H = "db040",

        N_h = NULL ,

        # clustering variable
        PSU = "rb030",

        # data.table
        dataset = dati,

        # age variable
        age = "age",

        # rmir coefficient function
        type = "linrmir",

        # poverty threshold range
        order_quant = 50L ,

        # get linearized variable
        outp_lin = TRUE

    )

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_rmir_calculation$all_result$value

## [1] 0.9330361

```

```

coef( svymir( ~ eqincome , des_eusilc, age = ~age ) )

## eqincome
## 0.9330361
# linearized variables do match
# vardpoor
lin_rmir_varpoord<- varpoord_rmir_calculation$lin_out$lin_rmir
# convey
lin_rmir_convey <- attr(svymir( ~ eqincome , des_eusilc, age = ~age ), "lin")

# check equality
all.equal(lin_rmir_varpoord, lin_rmir_convey[,1] )

## [1] TRUE
# variances do not match exactly
attr( svymir( ~ eqincome , des_eusilc, age = ~age ) , 'var' )

## eqincome
## eqincome 0.000127444
varpoord_rmir_calculation$all_result$var

## [1] 0.0001272137
# standard errors do not match exactly
varpoord_rmir_calculation$all_result$se

## [1] 0.0112789
SE( svymir( ~ eqincome , des_eusilc , age = ~age ) )

## eqincome
## eqincome 0.01128911

```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svymir` and `vardpoor::linrmir` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```

# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

```

```

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svymir( ~ eqincome , des_eusilc_ultimate_cluster , age = ~age ) , 'var' )

##              eqincome
## eqincome 0.0001272137
varpoord_rmir_calculation$all_result$var

## [1] 0.0001272137
# matches
varpoord_rmir_calculation$all_result$se

## [1] 0.0112789
SE( svymir( ~ eqincome , des_eusilc_ultimate_cluster , age = ~age ) )

##              eqincome
## eqincome 0.0112789

```

For additional usage examples of `svymir`, type `?convey::svymir` in the R console.

## 2.4 Relative Median Poverty Gap (svyrmprg)

The relative median poverty gap (`rmprg`) is the relative difference between the median income of people having income below the `arpt` and the `arpt` itself:

$$rmprg = \frac{\text{median}\{y_i, y_i < arpt\} - arpt}{arpt}$$

The details of the linearization of the `rmpg` are discussed by ? and ?.

---

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a `rmpg` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the vardpoor library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the rmpg coefficient
# using the R vardpoor library
varpoord_rmpg_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
```

```

        ID_level2 = "IDd",

        # strata variable
        H = "db040",

        N_h = NULL ,

        # clustering variable
        PSU = "rb030",

        # data.table
        dataset = dati,

        # rmpg coefficient function
        type = "linrmpg",

        # poverty threshold range
        order_quant = 50L ,

        # get linearized variable
        outp_lin = TRUE

    )

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_rmpg_calculation$all_result$value

## [1] 18.9286

```

```

coef( svyrmpg( ~ eqincome , des_eusilc ) ) * 100

## eqincome
## 18.9286

# linearized variables do match
# vardpoor
lin_rmpg_varpoord<- varpoord_rmpg_calculation$lin_out$lin_rmpg
# convey
lin_rmpg_convey <- attr(svyrmpg( ~ eqincome , des_eusilc ), "lin")

# check equality
all.equal(lin_rmpg_varpoord, 100*lin_rmpg_convey[,1] )

## [1] TRUE

# variances do not match exactly
attr( svyrmpg( ~ eqincome , des_eusilc ) , 'var' ) * 10000

##          eqincome
## eqincome 0.332234
varpoord_rmpg_calculation$all_result$var

## [1] 0.3316454

# standard errors do not match exactly
varpoord_rmpg_calculation$all_result$se

## [1] 0.5758866

SE( svyrmpg( ~ eqincome , des_eusilc ) ) * 100

##          eqincome
## eqincome 0.5763974

```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svyrmpg` and `vardpoor::linrmpg` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```

# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

```

```

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svyrmprg( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' ) * 10000

##          eqincome
## eqincome 0.3316454
varpoord_rmpg_calculation$all_result$var

## [1] 0.3316454
# matches
varpoord_rmpg_calculation$all_result$se

## [1] 0.5758866
SE( svyrmprg( ~ eqincome , des_eusilc_ultimate_cluster ) ) * 100

##          eqincome
## eqincome 0.5758866

```

For additional usage examples of `svyrmprg`, type `?convey::svyrmprg` in the R console.

## 2.5 Median Income Below the At Risk of Poverty Threshold (svypoormed)

Median income below the at-risk-of-poverty- threshold (poormed) is median of incomes of people having the income below the `arpt`:

$$poormed = median\{y_i; y_i < arpt\}$$

The details of the linearization of the `poormed` are discussed by ? and ?.

---

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a `poormed` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the vardpoor library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the poormed coefficient
# using the R vardpoor library
varpoord_poormed_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
```



## 2.5. MEDIAN INCOME BELOW THE AT RISK OF POVERTY THRESHOLD (SVYPOORMED)33

```
ID_level2 = "IDd",

# strata variable
H = "db040",

N_h = NULL ,

# clustering variable
PSU = "rb030",

# data.table
dataset = dati,

# poormed coefficient function
type = "linpoormed",

# poverty threshold range
order_quant = 50L ,

# get linearized variable
outp_lin = TRUE

)

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_poormed_calculation$all_result$value

## [1] 8803.735
```

```

coef( svypoormed( ~ eqincome , des_eusilc ) )

## eqincome
## 8803.735
# linearized variables do match
# vardpoor
lin_poormed_varpoord<- varpoord_poormed_calculation$lin_out$lin_poormed
# convey
lin_poormed_convey <- attr(svypoormed( ~ eqincome , des_eusilc ), "lin")

# check equality
all.equal(lin_poormed_varpoord, lin_poormed_convey )

## [1] TRUE
# variances do not match exactly
attr( svypoormed( ~ eqincome , des_eusilc ) , 'var' )

##          eqincome
## eqincome  5311.47
varpoord_poormed_calculation$all_result$var

## [1] 5302.086
# standard errors do not match exactly
varpoord_poormed_calculation$all_result$se

## [1] 72.81542
SE( svypoormed( ~ eqincome , des_eusilc ) )

##          eqincome
## eqincome 72.87983

```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svypoormed` and `vardpoor::linpoormed` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```

# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

```

```

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svypoormed( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' )

##           eqincome
## eqincome 5302.086
varpoord_poormed_calculation$all_result$var

## [1] 5302.086
# matches
varpoord_poormed_calculation$all_result$se

## [1] 72.81542
SE( svypoormed( ~ eqincome , des_eusilc_ultimate_cluster ) )

##           eqincome
## eqincome 72.81542

```

For additional usage examples of `svypoormed`, type `?convey::svypoormed` in the R console.

## 2.6 Foster-Greer-Thorbecke class (svyfgt, svyfgtdec)

? proposed a family of indicators to measure poverty. This class of *FGT* measures, can be defined as

$$p = \frac{1}{N} \sum_{k \in U} h(y_k, \theta),$$

where

$$h(y_k, \theta) = \left[ \frac{(\theta - y_k)}{\theta} \right]^\gamma \delta \{y_k \leq \theta\},$$

where:  $\theta$  is the poverty threshold;  $\delta$  the indicator function that assigns value 1 if the condition  $\{y_k \leq \theta\}$  is satisfied and 0 otherwise, and  $\gamma$  is a non-negative constant.

If  $\gamma = 0$ , the FGT(0) equals the poverty headcount ratio, which accounts for the spread of poverty. If  $\gamma = 1$ , FGT(1) is the mean of the normalized income shortfall of the poor. By doing so, the measure takes into account both the spread and the intensity of poverty. When  $\gamma = 2$ , the relative weight of larger shortfalls increases even more, which yields a measure that accounts for poverty severity, i.e., the inequality among the poor. This way, a transfer from a poor person to an even poorer person would reduce the FGT(2).

Although ? already presented a decomposition for the FGT(2) index, ? provided a general formula that decomposes the FGT( $\gamma$ ) for any  $\gamma \geq 2$ . Put simply, any such FGT( $\gamma$ ) index can be seen as function of the headcount ratio, the average normalized income gap among the poor and a generalized entropy index of the normalized income gaps among poor. In mathematical terms,

$$FGT_\gamma = FGT_0 \cdot I^\gamma \cdot [1 + (\gamma^2 - \gamma)GEI_\gamma^*], \gamma \geq 2$$

where  $I$  is the average normalized income gap among the poor and  $GEI_\gamma^*$  is a generalized entropy index of such income gaps among the poor.

This result is particularly useful, as one can attribute cross-sectional differences of a FGT index to differences in the spread, depth and inequality of poverty.

The FGT poverty class and its decomposition is implemented in the library convey by the function `svyfgt` and `svyfgtdec`, respectively. The argument `thresh_type` of this function defines the type of poverty threshold adopted. There are three possible choices:

1. **abs** – fixed and given by the argument `thresh_value`
2. **relq** – a proportion of a quantile fixed by the argument `proportion` and the quantile is defined by the argument `order`.
3. **relm** – a proportion of the mean fixed the argument `proportion`

The quantile and the mean involved in the definition of the threshold are estimated for the whole population. When  $\gamma = 0$  and  $\theta = .6 * MED$  the measure is

## 2.6. FOSTER-GREER-THORBECKE CLASS (SVYFGT, SVYFGTDEC) 37

equal to the indicator `arpr` computed by the function `svyarpr`. The linearization of the FGT(0) is presented in ?.

Next, we give some examples of the function `svyfgt` to estimate the values of the FGT poverty index.

Consider first the poverty threshold fixed ( $\gamma = 0$ ) in the value 10000. The headcount ratio (FGT0) is

```
svyfgt(~eqincome, des_eusilc, g=0, abs_thresh=10000)
```

```
          fgt0      SE
eqincome 0.11444 0.0027
```

The poverty gap ratio (FGT(1)) ( $\gamma = 1$ ) index for the poverty threshold fixed at the same value is

```
svyfgt(~eqincome, des_eusilc, g=1, abs_thresh=10000)
```

```
          fgt1      SE
eqincome 0.032085 0.0011
```

To estimate the FGT(0) with the poverty threshold fixed at  $0.6 * MED$  we fix the argument `type_thresh="relq"` and use the default values for `percent` and `order`:

```
svyfgt(~eqincome, des_eusilc, g=0, type_thresh= "relq")
```

```
          fgt0      SE
eqincome 0.14444 0.0028
```

that matches the estimate obtained by

```
svyarpr(~eqincome, design=des_eusilc, .5, .6)
```

```
          arpr      SE
eqincome 0.14444 0.0028
```

To estimate the poverty gap ratio with the poverty threshold equal to  $0.6 * MEAN$ , we use:

```
svyfgt(~eqincome, des_eusilc, g=1, type_thresh= "relm")
```

```
          fgt1      SE
eqincome 0.051187 0.0011
```

---

### A replication example

In July 2006, ? presented at the North American Stata Users' Group Meetings on the stata Atkinson Index command. The example below reproduces those statistics.

In order to match the presentation's results using the `svyfgt` function from the `convey` library, the poverty threshold was considered absolute despite being directly estimated from the survey sample. This effectively treats the variance of the estimated poverty threshold as zero; `svyfgt` does not account for the uncertainty of the poverty threshold when the level has been stated as absolute with the `abs_thresh=` parameter. In general, we would instead recommend using either `relq` or `reln` in the `type_thresh=` parameter in order to account for the added uncertainty of the poverty threshold calculation. This example serves only to show that `svyfgt` behaves properly as compared to other software.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the foreign library
library(foreign)

# create a temporary file on the local disk
tf <- tempfile()

# store the location of the presentation file
presentation_zip <- "http://repec.org/nasug2006/nasug2006_jenkins.zip"

# download jenkins' presentation to the temporary file
download.file( presentation_zip , tf , mode = 'wb' )

# unzip the contents of the archive
presentation_files <- unzip( tf , exdir = tempdir() )

# load the institute for fiscal studies' 1981, 1985, and 1991 data.frame objects
x81 <- read.dta( grep( "ifs81" , presentation_files , value = TRUE ) )
x85 <- read.dta( grep( "ifs85" , presentation_files , value = TRUE ) )
x91 <- read.dta( grep( "ifs91" , presentation_files , value = TRUE ) )

# NOTE: we recommend using ?convey::svyarpt rather than this unweighted calculation #

# calculate 60% of the unweighted median income in 1981
unwtd_arpt81 <- quantile( x81$eybhc0 , 0.5 ) * .6

# calculate 60% of the unweighted median income in 1985
unwtd_arpt85 <- quantile( x85$eybhc0 , 0.5 ) * .6
```

## 2.6. FOSTER-GREER-THORBECKE CLASS (SVYFGT, SVYFGTDEC) 39

```
# calculate 60% of the unweighted median income in 1991
unwtd_arpt91 <- quantile( x91$eybhc0 , 0.5 ) * .6

# stack each of these three years of data into a single data.frame
x <- rbind( x81 , x85 , x91 )
```

Replicate the author's survey design statement from stata code..

```
. ge poor = (year==1981)*(x < $z_81) + (year==1985)*(x < $z_85) + (year==1991)*(x < $z_91)
. * account for clustering within HHs
. svyset hrn [pweight = wgt]
```

.. into R code:

```
# initiate a linearized survey design object
y <- svydesign( ~ hrn , data = x , weights = ~ wgt )

# immediately run the `convey_prep` function on the survey design
z <- convey_prep( y )
```

Replicate the author's headcount ratio results with stata..

```
. svy: mean poor if year == 1981
(running mean on estimation sample)
```

Survey: Mean estimation

|                    |      |                   |         |
|--------------------|------|-------------------|---------|
| Number of strata = | 1    | Number of obs =   | 9772    |
| Number of PSUs =   | 7476 | Population size = | 5.5e+07 |
|                    |      | Design df =       | 7475    |

|      | Linearized |           |                      |
|------|------------|-----------|----------------------|
|      | Mean       | Std. Err. | [95% Conf. Interval] |
| poor | .1410125   | .0044859  | .132219 .149806      |

```
. svy: mean poor if year == 1985
(running mean on estimation sample)
```

Survey: Mean estimation

|                    |      |                   |         |
|--------------------|------|-------------------|---------|
| Number of strata = | 1    | Number of obs =   | 8991    |
| Number of PSUs =   | 6972 | Population size = | 5.5e+07 |
|                    |      | Design df =       | 6971    |

|      |         | Linearized |                      |          |
|------|---------|------------|----------------------|----------|
|      | Mean    | Std. Err.  | [95% Conf. Interval] |          |
| poor | .137645 | .0046531   | .1285235             | .1467665 |

```
. svy: mean poor if year == 1991
(running mean on estimation sample)
```

Survey: Mean estimation

```
Number of strata =      1      Number of obs   =    6468
Number of PSUs   =   5254      Population size = 5.6e+07
                                   Design df      =    5253
```

|      |          | Linearized |                      |          |
|------|----------|------------|----------------------|----------|
|      | Mean     | Std. Err.  | [95% Conf. Interval] |          |
| poor | .2021312 | .0062077   | .1899615             | .2143009 |

..using R code:

```
headcount_81 <-
  svyfgt(
    ~ eybhc0 ,
    subset( z , year == 1981 ) ,
    g = 0 ,
    abs_thresh = unwt_d_arpt81
  )

headcount_81

##          fgt0      SE
## eybhc0 0.14101 0.0045
confint( headcount_81 , df = degf( subset( z , year == 1981 ) ) )

##          2.5 %    97.5 %
## eybhc0 0.1322193 0.1498057

headcount_85 <-
  svyfgt(
    ~ eybhc0 ,
    subset( z , year == 1985 ) ,
    g = 0 ,
    abs_thresh = unwt_d_arpt85
  )
```



2.6. FOSTER-GREER-THORBECKE CLASS (SVYFGT, SVYFGTDEC) 41

```
)

headcount_85

##          fgt0      SE
## eybhc0 0.13764 0.0047
confint( headcount_85 , df = degf( subset( z , year == 1985 ) ) )

##          2.5 %    97.5 %
## eybhc0 0.1285239 0.1467661
headcount_91 <-
  svyfgt(
    ~ eybhc0 ,
    subset( z , year == 1991 ) ,
    g = 0 ,
    abs_thresh = unwtd_arpt91
  )

headcount_91

##          fgt0      SE
## eybhc0 0.20213 0.0062
confint( headcount_91 , df = degf( subset( z , year == 1991 ) ) )

##          2.5 % 97.5 %
## eybhc0 0.1899624 0.2143
```

Confirm this replication applies for the normalized poverty gap as well, comparing stata code..

```
. ge ngap = poor*($z_81- x)/$z_81 if year == 1981

. svy: mean ngap if year == 1981
(running mean on estimation sample)
```

Survey: Mean estimation

|                    |   |               |                 |           |
|--------------------|---|---------------|-----------------|-----------|
| Number of strata = | 1 | Number of obs | =               | 9772      |
| Number of PSUs     | = | 7476          | Population size | = 5.5e+07 |
|                    |   |               | Design df       | = 7475    |

| -----       |  |            |           |                      |
|-------------|--|------------|-----------|----------------------|
|             |  | Linearized |           |                      |
|             |  | Mean       | Std. Err. | [95% Conf. Interval] |
| -----+----- |  |            |           |                      |
| ngap        |  | .0271577   | .0013502  | .0245109 .0298044    |

-----

..to R code:

```
norm_pov_81 <-
  svyfgt(
    ~ eybhc0 ,
    subset( z , year == 1981 ) ,
    g = 1 ,
    abs_thresh = unwt_d_arpt81
  )

norm_pov_81

##           fgt1      SE
## eybhc0 0.027158 0.0014
confint( norm_pov_81 , df = degf( subset( z , year == 1981 ) ) )

##           2.5 %      97.5 %
## eybhc0 0.02451106 0.02980428
```

For additional usage examples of `svyfgt`, type `?convey::svyfgt` in the R console.

## 2.7 Watts poverty measure (`svywatts`, `svywatts-dec`)

The measure proposed in ? satisfies a number of desirable poverty measurement axioms and is known to be one of the first distribution-sensitive poverty measures, as noted by ?. It is defined as

$$Watts = \frac{1}{N} \sum_{i \in U} \log \left( \frac{y_i}{\theta} \right) \delta(y_i \leq \theta).$$

? points out that the Watts poverty index can provide an estimate of the expected time to exit poverty. Given the expected growth rate of income per capita among the poor,  $g$ , the expected time taken to exit poverty  $T_\theta$  would be

$$T_\theta = \frac{Watts}{g}.$$

The Watts poverty index also has interesting decomposition properties. ? proposed a decomposition for the Watts poverty index, rewriting it in terms of the

headcount ratio, the Watts poverty gap ratio and the mean log deviation of poor incomes<sup>2</sup>. Mathematically,

$$Watts = FGT_0(I_w + L_*)$$

where  $I_w = \log(\theta/\mu_*)$  is the Watts poverty gap ratio<sup>3</sup> and  $L_*$  is the mean log deviation of incomes among the poor. This can be estimated using the `svywattsdec` function.

This result can also be interpreted as a decomposition of the time taken to exit poverty, since

$$\begin{aligned} T_\theta &= \frac{Watts}{g} \\ &= \frac{FGT_0}{g}(I_w + L_*) \end{aligned}$$

As ? points out, if the income among the poor is equally distributed (i.e.,  $L_* = 0$ ), the time taken to exit poverty is simply  $FGT_0 I_w / g$ . Therefore,  $FGT_0 L_* / g$  can be seen as the additional time needed to exit poverty as a result of the inequality among the poor.

## 2.8 Clark-Hemming-Ulph class of poverty measures (svychu)

? proposes two classes of distribution-sensitive poverty measures. Yet, the poverty measurement literature focuses on the second class<sup>4</sup>, expressed as

$$CHU_\alpha = \begin{cases} \frac{1}{\alpha N} \sum_{i \in U} [1 - (y_i/\theta)^\alpha] \cdot \delta(y_i \leq \theta), & \alpha \leq 1, \alpha \neq 0 \\ 1 - \left( \prod_{i \in U} y_i^{\delta(y_i \leq \theta)} \right)^{1/N} / \theta, & \alpha = 0 \end{cases}$$

As an special case,  $CHU_0 = 1 - \exp(-Watts)$ . The  $\alpha$  parameter defines the sensivity towards regressive income transfers among the poor, such that the lower its value, larger is the regressive transfer impact on the index. When  $\alpha \rightarrow 1$ ,  $CHU_1 = FGT_0 \cdot I$ , a measure insensitive to regressive income transfers among the poor.

---

<sup>2</sup>The mean log deviation (also known as Theil-L or Bourguignon-Theil index) is an inequality measure of the generalized entropy class. The family of generalized entropy indices is discussed in the next chapter.

<sup>3</sup> $\mu_*$  stands for the average income among the poor.

<sup>4</sup>See ? and ?, for instance.



## Chapter 3

# Inequality Measurement

Another problem faced by societies is inequality. Economic inequality can have several different meanings: income, education, resources, opportunities, wellbeing, etc. Usually, studies on economic inequality focus on income distribution.

Most inequality data comes from censuses and household surveys. Therefore, in order to produce reliable estimates from this samples, appropriate procedures are necessary.

This chapter presents brief presentations on inequality measures, also providing replication examples if possible. It starts with an initial attempt to measure the inequality between two groups of a population; then, it presents ideas of overall inequality indices, moving from the quintile share ratio to the Lorenz curve and measures derived from it; then, it discusses the concept of entropy and presents inequality measures based on it. Finally, it ends with a discussion regarding which inequality measure should be used.

### 3.1 The Gender Pay Gap (svygpg)

Although the *GPG* is not an inequality measure in the usual sense, it can still be an useful instrument to evaluate the discrimination among men and women. Put simply, it expresses the relative difference between the average hourly earnings of men and women, presenting it as a percentage of the average of hourly earnings of men.

In mathematical terms, this index can be described as,

$$GPG = \frac{\bar{y}_{male} - \bar{y}_{female}}{\bar{y}_{male}}$$

,

which is precisely the estimator used in the package. As we can see from the formula, if there is no difference among classes,  $GPG = 0$ . Else, if  $GPG > 0$ , it means that the average hourly income received by women are  $GPG$  percent smaller than men's. For negative  $GPG$ , it means that women's hourly earnings are  $GPG$  percent larger than men's. In other words, the larger the  $GPG$ , larger is the shortfall of women's hourly earnings.

We can also develop a more straightforward idea: for every \$1 raise in men's hourly earnings, women's hourly earnings are expected to increase  $$(1 - GPG)$ . For instance, assuming  $GPG = 0.8$ , for every \$1.00 increase in men's average hourly earnings, women's hourly earnings would increase only \$0.20.

The details of the linearization of the GPG are discussed by ? and ?.

---

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a `gpg` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the laeken library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# coerce the gender variable to numeric 1 or 2
eusilc$one_two <- as.numeric( eusilc$rb090 == "female" ) + 1

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the gpg coefficient
```

```

# using the R varpoor library
varpoord_gpg_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
    ID_level2 = "IDd",

    # strata variable
    H = "db040",

    N_h = NULL ,

    # clustering variable
    PSU = "rb030",

    # data.table
    dataset = dati,

    # gpg coefficient function
    type = "lingpg" ,

    # gender variable
    gender = "one_two",

    # poverty threshold range
    order_quant = 50L ,

    # get linearized variable
    outp_lin = TRUE
  )

# construct a survey.design
# using our recommended setup
des_eusilc <-

```

```

    svydesign(
      ids = ~ rb030 ,
      strata = ~ db040 ,
      weights = ~ rb050 ,
      data = eusilc
    )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_gpg_calculation$all_result$value

## [1] 7.645389

coef( svygpg( ~ eqincome , des_eusilc , sex = ~ rb090 ) ) * 100

## eqincome
## 7.645389

# linearized variables do match
# vardpoor
lin_gpg_varpoord<- varpoord_gpg_calculation$lin_out$lin_gpg
# convey
lin_gpg_convey <- attr(svygpg( ~ eqincome , des_eusilc, sex = ~ rb090 ),"lin")

# check equality
all.equal(lin_gpg_varpoord,100*lin_gpg_convey[,1] )

## [1] TRUE

# variances do not match exactly
attr( svygpg( ~ eqincome , des_eusilc , sex = ~ rb090 ) , 'var' ) * 10000

##          eqincome
## eqincome 0.6493911
varpoord_gpg_calculation$all_result$var

## [1] 0.6482346

# standard errors do not match exactly
varpoord_gpg_calculation$all_result$se

## [1] 0.8051301
SE( svygpg( ~ eqincome , des_eusilc , sex = ~ rb090 ) ) * 100

##          eqincome
## eqincome 0.8058481

```



The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svygp` and `vardpoor::lingpg` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```
# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svygp( ~ eqincome , des_eusilc_ultimate_cluster , sex = ~ rb090 ) , 'var' ) * 10000

##          eqincome
## eqincome 0.6482346
varpoord_gpg_calculation$all_result$var

## [1] 0.6482346

# matches
varpoord_gpg_calculation$all_result$se

## [1] 0.8051301

SE( svygp( ~ eqincome , des_eusilc_ultimate_cluster , sex = ~ rb090 ) ) * 100
```

```
##           eqincome
## eqincome 0.8051301
```

For additional usage examples of `svygpg`, type `?convey::svygpg` in the R console.

## 3.2 Quintile Share Ratio (`svyqsr`)

Unlike the previous measure, the quintile share ratio is an inequality measure in itself, depending only of the income distribution to evaluate the degree of inequality. By definition, it can be described as the ratio between the income share held by the richest 20% and the poorest 20% of the population.

In plain terms, it expresses how many times the wealthier part of the population are richer than the poorest part. For instance, a  $QSR = 4$  implies that the upper class owns 4 times as much of the total income as the poor.

The quintile share ratio can be modified to a more general function of fractile share ratios. For instance, `?` presents interesting arguments for using the Palma index, defined as the ratio between the share of the 10% richest over the share held by the poorest 40%.

The details of the linearization of the QSR are discussed by `?` and `?`.

---

### A replication example

The R `vardpoor` package (`?`), created by researchers at the Central Statistical Bureau of Latvia, includes a `qsr` coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the laeken library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)
```

```

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the qsr coefficient
# using the R varpoord library
varpoord_qsr_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",

    # row number variable
    ID_level1 = "IDd",

    # row number variable
    ID_level2 = "IDd",

    # strata variable
    H = "db040",

    N_h = NULL ,

    # clustering variable
    PSU = "rb030",

    # data.table
    dataset = dati,

    # qsr coefficient function
    type = "linqsr",

    # poverty threshold range
    order_quant = 50L ,

    # get linearized variable
    outp_lin = TRUE

  )

```

```

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_qsr_calculation$all_result$value

## [1] 3.970004
coef( svyqsr( ~ eqincome , des_eusilc ) )

## eqincome
## 3.970004

# linearized variables do match
# vardpoor
lin_qsr_varpoord<- varpoord_qsr_calculation$lin_out$lin_qsr
# convey
lin_qsr_convey <- attr(svyqsr( ~ eqincome , des_eusilc ),"lin")

# check equality
all.equal(lin_qsr_varpoord, lin_qsr_convey )

## [1] TRUE

# variances do not match exactly
attr( svyqsr( ~ eqincome , des_eusilc ) , 'var' )

##
## eqincome
## eqincome 0.001810537
varpoord_qsr_calculation$all_result$var

## [1] 0.001807323

# standard errors do not match exactly
varpoord_qsr_calculation$all_result$se

## [1] 0.04251263

```

```
SE( svyqsr( ~ eqincome , des_eusilc ) )
```

```
##                eqincome
## eqincome 0.04255041
```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svygp` and `vardpoor::lingpg` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```
# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svyqsr( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' )

##                eqincome
## eqincome 0.001807323
vardpoord_qsr_calculation$all_result$var

## [1] 0.001807323
```

```
# matches
varpoord_qsr_calculation$all_result$se

## [1] 0.04251263
SE( svyqsr( ~ eqincome , des_eusilc_ultimate_cluster ) )

##               eqincome
## eqincome 0.04251263
```

For additional usage examples of `svyqsr`, type `?convey::svyqsr` in the R console.

### 3.3 Lorenz Curve (`svylorenz`)

Though not an inequality measure in itself, the Lorenz curve is a classic instrument of distribution analysis. Basically, it is a function that associates a cumulative share of the population to the share of the total income it owns. In mathematical terms,

$$L(p) = \frac{\int_{-\infty}^{Q_p} yf(y)dy}{\int_{-\infty}^{+\infty} yf(y)dy}$$

where  $Q_p$  is the quantile  $p$  of the population.

The two extreme distributive cases are

- Perfect equality:
  - Every individual has the same income;
  - Every share of the population has the same share of the income;
  - Therefore, the reference curve is

$$L(p) = p \quad \forall p \in [0, 1].$$

- Perfect inequality:
  - One individual concentrates all of society's income, while the other individuals have zero income;
  - Therefore, the reference curve is

$$L(p) = \begin{cases} 0, & \forall p < 1 \\ 1, & \text{if } p = 1. \end{cases}$$

In order to evaluate the degree of inequality in a society, the analyst looks at the distance between the real curve and those two reference curves.

The estimator of this function was derived by ?:

$$L(p) = \frac{\sum_{i \in S} w_i \cdot y_i \cdot \delta\{y_i \leq \widehat{Q}_p\}}{\widehat{Y}}, \quad 0 \leq p \leq 1.$$

Yet, this formula is used to calculate specific points of the curve and their respective SEs. The formula to plot an approximation of the continuous empirical curve comes from ?.

---

### A replication example

In October 2016, (?) released a pre-publication working paper to estimate lorenz and concentration curves using stata. The example below reproduces the statistics presented in his section 4.1.

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the stata-style webuse library
library(webuse)

# load the NLSW 1988 data
webuse("nlsw88")

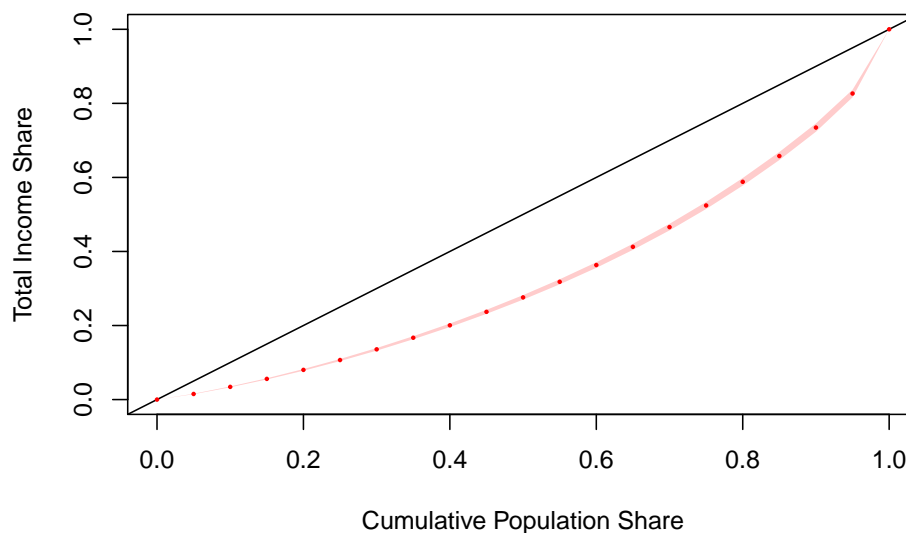
# coerce that `tbl_df` to a standard R `data.frame`
nlsw88 <- data.frame( nlsw88 )

# initiate a linearized survey design object
des_nlsw88 <- svydesign( ids = ~1 , data = nlsw88 )

## Warning in svydesign.default(ids = ~1, data = nlsw88): No weights or
## probabilities supplied, assuming equal probability

# immediately run the `convey_prep` function on the survey design
des_nlsw88 <- convey_prep(des_nlsw88)

# estimates lorenz curve
result.lin <- svylorenz( ~wage, des_nlsw88, quantiles = seq( 0, 1, .05 ), na.rm = TRUE )
```



*# note: most survey commands in R use Inf degrees of freedom by default  
 # stata generally uses the degrees of freedom of the survey design.  
 # therefore, while this extended syntax serves to prove a precise replication of stata  
 # it is generally not necessary.*

```
section_four_one <-
  data.frame(
    estimate = coef( result.lin ) ,
    standard_error = SE( result.lin ) ,
    ci_lower_bound =
      coef( result.lin ) +
      SE( result.lin ) *
      qt( 0.025 , degf( subset( des_nlsw88 , !is.na( wage ) ) ) ) ,
    ci_upper_bound =
      coef( result.lin ) +
      SE( result.lin ) *
      qt( 0.975 , degf( subset( des_nlsw88 , !is.na( wage ) ) ) )
  )
```



|      | estimate  | standard_error | ci_lower_bound | ci_upper_bound |
|------|-----------|----------------|----------------|----------------|
| 0    | 0.0000000 | 0.0000000      | 0.0000000      | 0.0000000      |
| 0.05 | 0.0151060 | 0.0004159      | 0.0142904      | 0.0159216      |
| 0.1  | 0.0342651 | 0.0007021      | 0.0328882      | 0.0356420      |
| 0.15 | 0.0558635 | 0.0010096      | 0.0538836      | 0.0578434      |
| 0.2  | 0.0801846 | 0.0014032      | 0.0774329      | 0.0829363      |
| 0.25 | 0.1067687 | 0.0017315      | 0.1033732      | 0.1101642      |
| 0.3  | 0.1356307 | 0.0021301      | 0.1314535      | 0.1398078      |
| 0.35 | 0.1670287 | 0.0025182      | 0.1620903      | 0.1719670      |
| 0.4  | 0.2005501 | 0.0029161      | 0.1948315      | 0.2062687      |
| 0.45 | 0.2369209 | 0.0033267      | 0.2303971      | 0.2434447      |
| 0.5  | 0.2759734 | 0.0037423      | 0.2686347      | 0.2833121      |
| 0.55 | 0.3180215 | 0.0041626      | 0.3098585      | 0.3261844      |
| 0.6  | 0.3633071 | 0.0045833      | 0.3543192      | 0.3722950      |
| 0.65 | 0.4125183 | 0.0050056      | 0.4027021      | 0.4223345      |
| 0.7  | 0.4657641 | 0.0054137      | 0.4551478      | 0.4763804      |
| 0.75 | 0.5241784 | 0.0058003      | 0.5128039      | 0.5355529      |
| 0.8  | 0.5880894 | 0.0062464      | 0.5758401      | 0.6003388      |
| 0.85 | 0.6577051 | 0.0066148      | 0.6447333      | 0.6706769      |
| 0.9  | 0.7346412 | 0.0068289      | 0.7212497      | 0.7480328      |
| 0.95 | 0.8265786 | 0.0062686      | 0.8142857      | 0.8388715      |
| 1    | 1.0000000 | 0.0000000      | 1.0000000      | 1.0000000      |

For additional usage examples of `svylorenz`, type `?convey::svylorenz` in the R console.

### 3.4 Gini index (svygini)

The Gini index is an attempt to express the inequality presented in the Lorenz curve as a single number. In essence, it is twice the area between the equality curve and the real Lorenz curve. Put simply:

$$G = 2 \left( \int_0^1 p dp - \int_0^1 L(p) dp \right)$$

$$\therefore G = 1 - 2 \int_0^1 L(p) dp$$

where  $G = 0$  in case of perfect equality and  $G = 1$  in the case of perfect inequality.

The estimator proposed by ? is defined as:

$$\widehat{G} = \frac{2 \sum_{i \in S} w_i r_i y_i - \sum_{i \in S} w_i y_i}{\widehat{Y}}$$

The linearized formula of  $\widehat{G}$  is used to calculate the SE.

---

### A replication example

The R `vardpoor` package (?), created by researchers at the Central Statistical Bureau of Latvia, includes a gini coefficient calculation using the ultimate cluster method. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the vardpoor library
library(vardpoor)

# load the laeken library
library(laeken)

# load the synthetic EU statistics on income & living conditions
data(eusilc)

# make all column names lowercase
names( eusilc ) <- tolower( names( eusilc ) )

# add a column with the row number
dati <- data.table::data.table(IDd = 1 : nrow(eusilc), eusilc)

# calculate the gini coefficient
# using the R vardpoor library
varpoord_gini_calculation <-
  varpoord(

    # analysis variable
    Y = "eqincome",

    # weights variable
    w_final = "rb050",
```

```

        # row number variable
        ID_level1 = "IDd",

        # row number variable
        ID_level2 = "IDd",

        # strata variable
        H = "db040",

        N_h = NULL ,

        # clustering variable
        PSU = "rb030",

        # data.table
        dataset = dati,

        # gini coefficient function
        type = "lingini",

        # poverty threshold range
        order_quant = 50L ,

        # get linearized variable
        outp_lin = TRUE

    )

# construct a survey.design
# using our recommended setup
des_eusilc <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc
  )

# immediately run the convey_prep function on it
des_eusilc <- convey_prep( des_eusilc )

# coefficients do match
varpoord_gini_calculation$all_result$value

```

```
## [1] 26.49652
coef( svygini( ~ eqincome , des_eusilc ) ) * 100

## eqincome
## 26.49652
# linearized variables do match
# varpoord
lin_gini_varpoord<- varpoord_gini_calculation$lin_out$lin_gini
# convey
lin_gini_convey <- attr(svygini( ~ eqincome , des_eusilc ),"lin")

# check equality
all.equal(lin_gini_varpoord,100*lin_gini_convey )

## [1] TRUE
# variances do not match exactly
attr( svygini( ~ eqincome , des_eusilc ) , 'var' ) * 10000

##
## eqincome
## eqincome 0.03790739
varpoord_gini_calculation$all_result$var

## [1] 0.03783931
# standard errors do not match exactly
varpoord_gini_calculation$all_result$se

## [1] 0.1945233
SE( svygini( ~ eqincome , des_eusilc ) ) * 100

##
## eqincome
## eqincome 0.1946982
```

The variance estimate is computed by using the approximation defined in (??), where the linearized variable  $z$  is defined by (??). The functions `convey::svygini` and `vardpoor::lingini` produce the same linearized variable  $z$ .

However, the measures of uncertainty do not line up, because `library(vardpoor)` defaults to an ultimate cluster method that can be replicated with an alternative setup of the `survey.design` object.

```
# within each strata, sum up the weights
cluster_sums <- aggregate( eusilc$rb050 , list( eusilc$db040 ) , sum )

# name the within-strata sums of weights the `cluster_sum`
names( cluster_sums ) <- c( "db040" , "cluster_sum" )
```

```

# merge this column back onto the data.frame
eusilc <- merge( eusilc , cluster_sums )

# construct a survey.design
# with the fpc using the cluster sum
des_eusilc_ultimate_cluster <-
  svydesign(
    ids = ~ rb030 ,
    strata = ~ db040 ,
    weights = ~ rb050 ,
    data = eusilc ,
    fpc = ~ cluster_sum
  )

# again, immediately run the convey_prep function on the `survey.design`
des_eusilc_ultimate_cluster <- convey_prep( des_eusilc_ultimate_cluster )

# matches
attr( svygini( ~ eqincome , des_eusilc_ultimate_cluster ) , 'var' ) * 10000

##              eqincome
## eqincome 0.03783931
varpoord_gini_calculation$all_result$var

## [1] 0.03783931
# matches
varpoord_gini_calculation$all_result$se

## [1] 0.1945233
SE( svygini( ~ eqincome , des_eusilc_ultimate_cluster ) ) * 100

##              eqincome
## eqincome 0.1945233

```

### 3.5 Entropy-based Measures

Entropy is a concept derived from information theory, meaning the expected amount of information given the occurrence of an event. Following (?), given an event  $y$  with probability density function  $f(\cdot)$ , the information content given the occurrence of  $y$  can be defined as  $g(f(y))$ :  $= -\log f(y)$ . Therefore, the expected information or, put simply, the *entropy* is

$$H(f) \doteq -E[\log f(y)] = -\int_{-\infty}^{\infty} f(y) \log f(y) dy$$

Assuming a discrete distribution, with  $p_k$  as the probability of occurring event  $k \in K$ , the entropy formula takes the form:

$$H = -\sum_{k \in K} p_k \log p_k.$$

The main idea behind it is that the expected amount of information of an event is inversely proportional to the probability of its occurrence. In other words, the information derived from the observation of a rare event is higher than of the information of more probable events.

Using ideas presented in ?, substituting the density function by the income share of an individual  $s(q) = F^{-1}(q)/\int_0^1 F^{-1}(t)dt = y/\mu$ , the entropy function becomes the Theil<sup>1</sup> inequality index

$$I_{Theil} = \int_0^{\infty} \frac{y}{\mu} \log \left( \frac{y}{\mu} \right) dF(y) = -H(s)$$

Therefore, the entropy-based inequality measure increases as a person's income  $y$  deviates from the mean  $\mu$ . This is the basic idea behind entropy-based inequality measures.

### 3.6 Generalized Entropy and Decomposition (svygei, svygeidec)

Using a generalization of the information function, now defined as  $g(f) = \frac{1}{\alpha-1}[1 - f^{\alpha-1}]$ , the  $\alpha$ -class entropy is

$$H_{\alpha}(f) = \frac{1}{\alpha-1} \left[ 1 - \int_{-\infty}^{\infty} f(y)^{\alpha-1} f(y) dy \right].$$

This relates to a class of inequality measures, the Generalized entropy indices, defined as:

$$GE_{\alpha} = \frac{1}{\alpha^2 - \alpha} \int_0^{\infty} \left[ \left( \frac{y}{\mu} \right)^{\alpha} - 1 \right] dF(x) = -\frac{H_{\alpha}(s)}{\alpha}.$$

---

<sup>1</sup>Also known as Theil-T index.

### 3.6. GENERALIZED ENTROPY AND DECOMPOSITION (SVYGEI, SVYGEIDEC) 63

The parameter  $\alpha$  also has an economic interpretation: as  $\alpha$  increases, the influence of top incomes upon the index increases. In some cases, this measure takes special forms, such as mean log deviation and the aforementioned Theil index.

In order to estimate it, ? proposed the following:

$$GE_{\alpha} = \begin{cases} (\alpha^2 - \alpha)^{-1} [U_0^{\alpha-1} U_1^{-\alpha} U_{\alpha} - 1], & \text{if } \alpha \in \mathbb{R} \setminus \{0, 1\} \\ -T_0 U_0^{-1} + \log(U_1/U_0), & \text{if } \alpha \rightarrow 0 \\ T_1 U_1^{-1} - \log(U_1/U_0), & \text{if } \alpha \rightarrow 1 \end{cases}$$

where  $U_{\gamma} = \sum_{i \in S} w_i \cdot y_i^{\gamma}$  and  $T_{\gamma} = \sum_{i \in S} w_i \cdot y_i^{\gamma} \cdot \log y_i$ . Since those are all functions of totals, the linearization of the indices are easily achieved using the theorems described in ?.

This class also has several desirable properties, such as additive decomposition. The additive decomposition allows to compare the effects of inequality within and between population groups on the population inequality. Put simply, an additive decomposable index allows for:

$$I_{Total} = I_{Between} + I_{Within}.$$

---

#### A replication example

In July 2006, ? presented at the North American Stata Users' Group Meetings on the stata Generalized Entropy Index command. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the foreign library
library(foreign)

# create a temporary file on the local disk
tf <- tempfile()

# store the location of the presentation file
presentation_zip <- "http://repec.org/nasug2006/nasug2006_jenkins.zip"

# download jenkins' presentation to the temporary file
download.file( presentation_zip , tf , mode = 'wb' )
```

```

# unzip the contents of the archive
presentation_files <- unzip( tf , exdir = tempdir() )

# load the institute for fiscal studies' 1981, 1985, and 1991 data.frame objects
x81 <- read.dta( grep( "ifs81" , presentation_files , value = TRUE ) )
x85 <- read.dta( grep( "ifs85" , presentation_files , value = TRUE ) )
x91 <- read.dta( grep( "ifs91" , presentation_files , value = TRUE ) )

# stack each of these three years of data into a single data.frame
x <- rbind( x81 , x85 , x91 )

```

Replicate the author's survey design statement from stata code..

```

. * account for clustering within HHs
. version 8: svyset [pweight = wgt], psu(hrn)
pweight is wgt
psu is hrn
construct an

```

.. into R code:

```

# initiate a linearized survey design object
y <- svydesign( ~ hrn , data = x , weights = ~ wgt )

# immediately run the `convey_prep` function on the survey design
z <- convey_prep( y )

```

Replicate the author's subset statement and each of his svygei results..

```
. svygei x if year == 1981
```

Warning: x has 20 values = 0. Not used in calculations

Complex survey estimates of Generalized Entropy inequality indices

```

pweight: wgt                      Number of obs    = 9752
Strata: <one>                      Number of strata = 1
PSU: hrn                          Number of PSUs   = 7459
Population size = 54766261

```

| Index  | Estimate | Std. Err. | z     | P> z  | [95% Conf. Interval] |          |
|--------|----------|-----------|-------|-------|----------------------|----------|
| GE(-1) | .1902062 | .02474921 | 7.69  | 0.000 | .1416987             | .2387138 |
| MLD    | .1142851 | .00275138 | 41.54 | 0.000 | .1088925             | .1196777 |
| Theil  | .1116923 | .00226489 | 49.31 | 0.000 | .1072532             | .1161314 |
| GE(2)  | .128793  | .00330774 | 38.94 | 0.000 | .1223099             | .135276  |
| GE(3)  | .1739994 | .00662015 | 26.28 | 0.000 | .1610242             | .1869747 |



### 3.6. GENERALIZED ENTROPY AND DECOMPOSITION (SVYGEI, SVYGEIDEC)65

-----  
 ..using R code:

```
z81 <- subset( z , year == 1981 )

svygei( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = -1 )

##              gei              SE epsilon
## eybhc0 0.190206 0.024748          -1

svygei( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 0 )

##              gei              SE epsilon
## eybhc0 0.1142851 0.0027513          0

svygei( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) )

##              gei              SE epsilon
## eybhc0 0.1116923 0.0022648          1

svygei( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 2 )

##              gei              SE epsilon
## eybhc0 0.1287930 0.0033076          2

svygei( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 3 )

##              gei              SE epsilon
## eybhc0 0.1739994 0.0066199          3
```

Confirm this replication applies for subsetted objects as well. Compare stata output..

```
. svygei x if year == 1985 & x >= 1
```

Complex survey estimates of Generalized Entropy inequality indices

```
pweight: wgt                      Number of obs    = 8969
Strata: <one>                      Number of strata = 1
PSU: hrn                          Number of PSUs   = 6950
      Population size = 55042871
```

| Index  | Estimate | Std. Err. | z     | P> z  | [95% Conf. Interval] |          |
|--------|----------|-----------|-------|-------|----------------------|----------|
| GE(-1) | .1602358 | .00936931 | 17.10 | 0.000 | .1418723             | .1785993 |
| MLD    | .127616  | .00332187 | 38.42 | 0.000 | .1211052             | .1341267 |
| Theil  | .1337177 | .00406302 | 32.91 | 0.000 | .1257543             | .141681  |
| GE(2)  | .1676393 | .00730057 | 22.96 | 0.000 | .1533304             | .1819481 |
| GE(3)  | .2609507 | .01850689 | 14.10 | 0.000 | .2246779             | .2972235 |

..to R code:

```
z85 <- subset( z , year == 1985 )

svygei( ~ eybhc0 , subset( z85 , eybhc0 > 1 ) , epsilon = -1 )
```

```
##              gei          SE epsilon
## eybhc0 0.1602358 0.0093689      -1
```

```
svygei( ~ eybhc0 , subset( z85 , eybhc0 > 1 ) , epsilon = 0 )
```

```
##              gei          SE epsilon
## eybhc0 0.1276160 0.0033217        0
```

```
svygei( ~ eybhc0 , subset( z85 , eybhc0 > 1 ) )
```

```
##              gei          SE epsilon
## eybhc0 0.1337177 0.0040628        1
```

```
svygei( ~ eybhc0 , subset( z85 , eybhc0 > 1 ) , epsilon = 2 )
```

```
##              gei          SE epsilon
## eybhc0 0.1676393 0.0073002        2
```

```
svygei( ~ eybhc0 , subset( z85 , eybhc0 > 1 ) , epsilon = 3 )
```

```
##              gei          SE epsilon
## eybhc0 0.260951 0.018506          3
```

Replicate the author's decomposition by population subgroup (work status) shown on PDF page 57..

```
# define work status (PDF page 22)
```

```
z <- update( z , wkstatus = c( 1 , 1 , 1 , 1 , 2 , 3 , 2 , 2 ) [ as.numeric( esbu ) ] )
```

```
z <- update( z , factor( wkstatus , labels = c( "1+ ft working" , "no ft working" , "e
```

```
# subset to 1991 and remove records with zero income
```

```
z91 <- subset( z , year == 1991 & eybhc0 > 0 )
```

```
# population share
```

```
svymean( ~wkstatus, z91 )
```

```
##              mean      SE
## wkstatus 1.5594 0.0099
```

```
# mean
```

```
svyby( ~eybhc0, ~wkstatus, z91, svymean )
```

```
##   wkstatus   eybhc0      se
## 1         1 278.8040 3.703790
## 2         2 151.6317 3.153968
```

### 3.6. GENERALIZED ENTROPY AND DECOMPOSITION (SVYGEI, SVYGEIDEC)67

```
## 3          3 176.6045 4.661740
```

```
# subgroup indices: ge_k
```

```
svyby( ~ eybhc0 , ~wkstatus , z91 , svygei , epsilon = -1 )
```

```
##   wkstatus   eybhc0       se
## 1         1  0.2300708 0.02853959
## 2         2 10.9231761 10.65482557
## 3         3  0.1932164  0.02571991
```

```
svyby( ~ eybhc0 , ~wkstatus , z91 , svygei , epsilon = 0 )
```

```
##   wkstatus   eybhc0       se
## 1         1 0.1536921 0.006955506
## 2         2 0.1836835 0.014740510
## 3         3 0.1653658 0.016409770
```

```
svyby( ~ eybhc0 , ~wkstatus , z91 , svygei , epsilon = 1 )
```

```
##   wkstatus   eybhc0       se
## 1         1 0.1598558 0.008327994
## 2         2 0.1889909 0.016766120
## 3         3 0.2023862 0.027787224
```

```
svyby( ~ eybhc0 , ~wkstatus , z91 , svygei , epsilon = 2 )
```

```
##   wkstatus   eybhc0       se
## 1         1 0.2130664 0.01546521
## 2         2 0.2846345 0.06016394
## 3         3 0.3465088 0.07362898
```

```
# GE decomposition
```

```
svygeidec( ~eybhc0, ~wkstatus, z91, epsilon = -1 )
```

```
##           total within between
## coef 3.6829 3.6466  0.0363
## SE   3.3999 3.3993  0.0541
```

```
svygeidec( ~eybhc0, ~wkstatus, z91, epsilon = 0 )
```

```
##           total   within between
## coef 0.1952363 0.1619352  0.0333
## SE   0.0064615 0.0062209  0.0027
```

```
svygeidec( ~eybhc0, ~wkstatus, z91, epsilon = 1 )
```

```
##           total   within between
## coef 0.2003897 0.1693958  0.0310
## SE   0.0079299 0.0082236  0.0027
```

```
svygeidec( ~eybhc0, ~wkstatus, z91, epsilon = 2 )
```

```
##          total   within between
## coef 0.274325 0.245067 0.0293
## SE   0.016694 0.017831 0.0038
```

For additional usage examples of `svygei` or `svygeidec`, type `?convey::svygei` or `?convey::svygeidec` in the R console.

### 3.7 J-Divergence and Decomposition (`svyjdiv`, `svyjdivdec`)

The J-divergence measure (?) can be seen as the sum of  $GE_0$  and  $GE_1$ , satisfying axioms that, individually, those two indices do not. Using  $U_\gamma$  and  $T_\gamma$  functions defined in ?, the estimator can be defined as:

$$J = \frac{1}{N} \sum_{i \in S} \left( \frac{y_i - \mu}{\mu} \right) \log \left( \frac{y_i}{\mu} \right)$$

$$\therefore \hat{J} = \frac{\hat{T}_1}{\hat{U}_1} - \frac{\hat{T}_0}{\hat{U}_0}$$

Since it is a sum of two additive decomposable measures,  $J$  itself is decomposable.

For additional usage examples of `svyjdiv` or `svyjdivdec`, type `?convey::svyjdiv` or `?convey::svyjdivdec` in the R console.

### 3.8 Atkinson index (`svyatk`)

Although the original formula was proposed in ?, the estimator used here comes from ?:

$$\hat{A}_\epsilon = \begin{cases} 1 - \hat{U}_0^{-\epsilon/(1-\epsilon)} \hat{U}_1^{-1} \hat{U}_{1-\epsilon}^{1/(1-\epsilon)}, & \text{if } \epsilon \in \mathbb{R}_+ \setminus \{1\} \\ 1 - \hat{U}_0 \hat{U}_0^{-1} \exp(\hat{T}_0 \hat{U}_0^{-1}), & \text{if } \epsilon \rightarrow 1 \end{cases}$$

The  $\epsilon$  is an inequality aversion parameter: as it approaches infinity, more weight is given to incomes in bottom of the distribution.

---

#### A replication example

In July 2006, ? presented at the North American Stata Users' Group Meetings on the stata Atkinson Index command. The example below reproduces those statistics.

Load and prepare the same data set:

```
# load the convey package
library(convey)

# load the survey library
library(survey)

# load the foreign library
library(foreign)

# create a temporary file on the local disk
tf <- tempfile()

# store the location of the presentation file
presentation_zip <- "http://repec.org/nasug2006/nasug2006_jenkins.zip"

# download jenkins' presentation to the temporary file
download.file( presentation_zip , tf , mode = 'wb' )

# unzip the contents of the archive
presentation_files <- unzip( tf , exdir = tempdir() )

# load the institute for fiscal studies' 1981, 1985, and 1991 data.frame objects
x81 <- read.dta( grep( "ifs81" , presentation_files , value = TRUE ) )
x85 <- read.dta( grep( "ifs85" , presentation_files , value = TRUE ) )
x91 <- read.dta( grep( "ifs91" , presentation_files , value = TRUE ) )

# stack each of these three years of data into a single data.frame
x <- rbind( x81 , x85 , x91 )
```

Replicate the author's survey design statement from stata code..

```
. * account for clustering within HHs
. version 8: svyset [pweight = wgt], psu(hrn)
pweight is wgt
psu is hrn
construct an
```

.. into R code:

```
# initiate a linearized survey design object
y <- svydesign( ~ hrn , data = x , weights = ~ wgt )

# immediately run the `convey_prep` function on the survey design
z <- convey_prep( y )
```

Replicate the author's subset statement and each of his svyatk results with

```
stata..
```

```
. svyatk x if year == 1981
```

```
Warning: x has 20 values = 0. Not used in calculations
```

```
Complex survey estimates of Atkinson inequality indices
```

```
pweight: wgt                      Number of obs    = 9752
Strata: <one>                      Number of strata = 1
PSU: hrn                          Number of PSUs   = 7459
                                   Population size  = 54766261
```

| Index  | Estimate | Std. Err. | z     | P> z  | [95% Conf. Interval] |
|--------|----------|-----------|-------|-------|----------------------|
| A(0.5) | .0543239 | .00107583 | 50.49 | 0.000 | .0522153 .0564324    |
| A(1)   | .1079964 | .00245424 | 44.00 | 0.000 | .1031862 .1128066    |
| A(1.5) | .1701794 | .0066943  | 25.42 | 0.000 | .1570588 .1833       |
| A(2)   | .2755788 | .02597608 | 10.61 | 0.000 | .2246666 .326491     |
| A(2.5) | .4992701 | .06754311 | 7.39  | 0.000 | .366888 .6316522     |

```
..using R code:
```

```
z81 <- subset( z , year == 1981 )
```

```
svyatk( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 0.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.054324 0.0011
```

```
svyatk( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) )
```

```
##          atkinson      SE
```

```
## eybhc0   0.108 0.0025
```

```
svyatk( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 1.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.17018 0.0067
```

```
svyatk( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 2 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.27558 0.026
```

```
svyatk( ~ eybhc0 , subset( z81 , eybhc0 > 0 ) , epsilon = 2.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.49927 0.0675
```

Confirm this replication applies for subsetted objects as well, comparing stata code..

```
. svyatk x if year == 1981 & x >= 1
```

Complex survey estimates of Atkinson inequality indices

```
pweight: wgt                      Number of obs   = 9748
Strata: <one>                      Number of strata = 1
PSU: hrn                          Number of PSUs   = 7457
                                   Population size  = 54744234
```

| Index  | Estimate | Std. Err. | z     | P> z  | [95% Conf. Interval] |          |
|--------|----------|-----------|-------|-------|----------------------|----------|
| A(0.5) | .0540059 | .00105011 | 51.43 | 0.000 | .0519477             | .0560641 |
| A(1)   | .1066082 | .00223318 | 47.74 | 0.000 | .1022313             | .1109852 |
| A(1.5) | .1638299 | .00483069 | 33.91 | 0.000 | .154362              | .1732979 |
| A(2)   | .2443206 | .01425258 | 17.14 | 0.000 | .2163861             | .2722552 |
| A(2.5) | .394787  | .04155221 | 9.50  | 0.000 | .3133461             | .4762278 |

..to R code:

```
z81_two <- subset( z , year == 1981 & eybhc0 > 1 )
```

```
svyatk( ~ eybhc0 , z81_two , epsilon = 0.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.054006 0.0011
```

```
svyatk( ~ eybhc0 , z81_two )
```

```
##          atkinson      SE
```

```
## eybhc0 0.10661 0.0022
```

```
svyatk( ~ eybhc0 , z81_two , epsilon = 1.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.16383 0.0048
```

```
svyatk( ~ eybhc0 , z81_two , epsilon = 2 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.24432 0.0143
```

```
svyatk( ~ eybhc0 , z81_two , epsilon = 2.5 )
```

```
##          atkinson      SE
```

```
## eybhc0 0.39479 0.0416
```

For additional usage examples of `svyatk`, type `?convey::svyatk` in the R console.

### 3.9 Which inequality measure should be used?

The variety of inequality measures begs a question: which inequality measure should be used? In fact, this is a very important question. However, the nature of it is not statistical or mathematical, but ethical. This section aims to clarify and, while not proposing a “perfect measure”, to provide the reader with an initial guidance about which measure to use.

The most general way to analyze if one distribution is more equally distributed than another is by the Lorenz curve. When  $L_A(p) \geq L_B(p), \forall p \in [0, 1]$ , it is said that  $A$  is more equally distributed than  $B$ . Technically, we say that  $A$  (Lorenz) *dominates*  $B$ <sup>2</sup>. In this case, all inequality measures that satisfy basic properties<sup>3</sup> will agree that  $A$  is more equally distributed than  $B$ .

When this dominance fails, i.e., when Lorenz curves do cross, Lorenz ordering is impossible. Then, under such circumstances, the choice of which inequality measure to use becomes relevant.

Each inequality measure is a result of a subjective understanding of what is a fair distribution. As ?, p.348 puts it, “[...] the economist is primarily interested, not in the distribution of income as such, but in the effects of the distribution of income upon the distribution and total amount of economic welfare, which may be derived from income.” The importance of how economic welfare is defined is once again expressed by ?, where an inequality measure is directly derived from a class of welfare functions. Even when a welfare function is not explicit, such as in the Gini index, we must agree that an implicit, subjective judgement of the impact of inequality on social welfare is assumed.

The idea of what is a fair distribution is a matter of Ethics, a discipline within the realm of Philosophy. Yet, as ?, Ch.1 proposes, the analyst should match socially supported moral values and theories of justice to the set of technical tools for policy evaluation.

Although this can be a useful principle, a more objective answer is needed. By knowing the nature and properties of inequality measures, the analyst can further reduce the set of applicable inequality measures. For instance, choosing from the properties listed in ?, p.74, if we require group-decomposability, scale invariance, population invariance, and that the estimate in  $[0, 1]$ , we must resort to the Atkinson index.

Even though the discussion can go deep in technical and philosophical aspects,

---

<sup>2</sup>? and ? provide helpful insights to how majorization, Lorenz dominance, and inequality measurement are connected. On the topic of majorization, ? is still the main reference, while ? provide a more modern approach.

<sup>3</sup>Namely, Schur-convexity, population invariance, and scale invariance.



this choice also depends on the public. For example, it would not be surprising if a public official doesn't know the Atkinson index; however, he might know the Gini index. The same goes for publications: journalists have been introduced to the Gini index and can find it easier to compare and, therefore, write about it. Also, we must admit that the Gini index is much more straightforward than any other measure.

In the end, the choice is mostly subjective and there is no consensus of which is the "greatest inequality measure". We must remember that this choice is only problematic if Lorenz curves cross and, in that case, it is not difficult to justify the use of this or that inequality measure.