Applying Generalized Pareto Interpolation with `gpinter`

Thomas Blanchet, Paris School of Economics

April 23, 2018

Generalized Pareto Interpolation is a method for reconstructing complete distributions based on tabulations that only contain information on a few thresholds and bracket shares. It is usually applied in the study of income and wealth, but it can also be used with other types of data.

The method is described in detail in the paper by Thomas Blanchet, Juliette Fournier and Thomas Piketty, "Generalized Pareto curves: Theory and Applications" (2017). This vignette focuses on the use of the R package `gpinter`, which implements generalized Pareto interpolation.

For people with limited needs who do not wish to use R, the method can also be used through the online application at `http://wid.world/gpinter`, which relies on the R package, but does not require any installation or programming skill. Using the R package directly, however, provides more flexibility and more possibilities of automation.

Note for Stata users

There is no native Stata command equivalent to `gpinter`, but if you are a Stata user, you can integrate `gpinter` in your workflow quite easily by calling R from your Stata programs. The example `.do` file at the address `https://github.com/thomasblanchet/gpinter/blob/master/inst/stata/gpinter-stata-example.do` demonstrates how to make this work.

1 Installation

The package is not yet on the CRAN, but it is available on GitHub, so you can install it with the following code:

```
install.packages("devtools")
devtools::install_github("thomasblanchet/gpinter")
```

2 Run the Online Application on your Computer

The code for the online application at `http://wid.world/gpinter` is included in the package, so you can run the application locally on your computer using:

```
library(gpinter)
run_app()
```

To run that code, you will need additional R packages not installed by default with `gpinter`. You can install them all with the following command:

```
devtools::install_github("thomasblanchet/gpinter",
    dependencies = c("Depends", "Imports", "LinkingTo", "Suggests"))
```

3 Using the Package

When tax authorities give information on the distribution of income and wealth, we rarely get a sample of individual observations due to privacy and/or logistical concerns. Instead, we have a table that provides, for a handful of income brackets, the number of people in the bracket and their average income. As an example, we will assume a distribution with average income 37 208 and the following tabulation (it corresponds to the distribution of labor income in the United States in 2010):

percentile	quantile	bracket average
10%	4 310	12 643
50%	23 686	43 908
90%	76 252	108 329
99%	211 861	471 463

This tabulation should be read as follows: the 10% quantile is equal to 4 310, the 50% quantile is equal to 23 686, and the people between those two thresholds have an average income of 12 643. We input that tabulation into R as:

```
average <- 37208 # Average income
p <- c(0.10, 0.50, 0.90, 0.99) # Fractiles between 0 an 1
q <- c(4310, 23686, 76252, 211861) # Matching quantiles
a <- c(12643, 43908, 108329, 471463) # Matching bracket averages
```

3.1 Basic Interpolation

To interpolate the distribution above, we import the package `gpinter` and use the function `tabulation_fit`.

```
library(gpinter)
distribution <- tabulation_fit(p, q, bracketavg = a, average = average)
```

Note that we could have used bracket shares, top shares or top averages instead of bracket averages by changing the name of the argument `bracketavg` (see `?tabulation_fit` for details). They all provide the exact same information on the distribution.

Note also that we did not start the tabulation at $p = 0$. The method always starts the generalized Pareto interpolation at $\min(p)$, and fits a separate parametric model at the bottom of the distribution if necessary. In practice, not starting the tabulation at zero can give better results if the distribution is irregular at the bottom (e.g. because of a mass of people at zero). Indeed, irregular behavior at the very bottom can sometimes perturb the interpolation for the rest of the distribution. The model used from $p = 0$ to $\min(p)$ can be adjusted using the arguments `bottom_model` and `lower_bound`. It will never have any impact on the rest of the distribution, so you can ignore it if you are not interested in the bottom of the distribution.

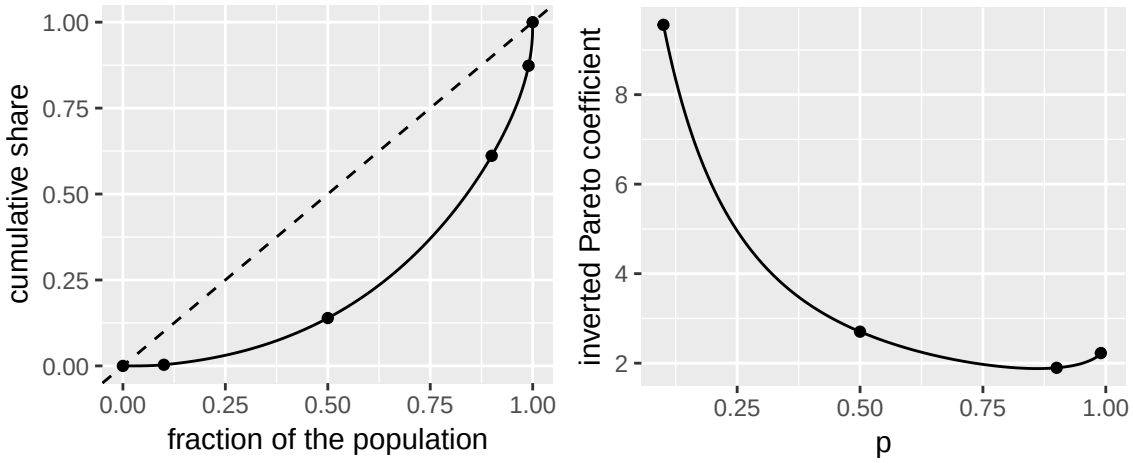
The function `tabulation_fit` returns an R object that is a self-contained, complete internal representation of the interpolated distribution. You should not read the attributes of that object directly, but instead use the auxiliary functions as follows:

```
# Top 5% and 0.1% share
top_share(distribution, c(0.95, 0.999))
#> [1] 0.27191705 0.04792791

# Bracket 95%-99% share
bracket_share(distribution, 0.95, 0.99)
```

```r
#> [1] 0.1452069

# Top 5% and 0.1% averages
top_average(distribution, c(0.95, 0.999))
#> [1]  202349.8 1783301.5

# Bracket 95%-99% average
bracket_average(distribution, 0.95, 0.99)
#> [1] 135071.5

# 95% and 99.9% quantiles
fitted_quantile(distribution, c(0.95, 0.999))
#> [1] 101688.1 720856.3

# Cumulative distribution function at $100,000 and $1,000,000
fitted_cdf(distribution, c(1e5, 1e6))
#> [1] 0.9479556 0.9994325

# Density at $100,000 and $1,000,000
fitted_density(distribution, c(1e5, 1e6))
#> [1] 1.247901e-06 9.732845e-10

# Gini index
gini(distribution)
#> [1] 0.5431319
```

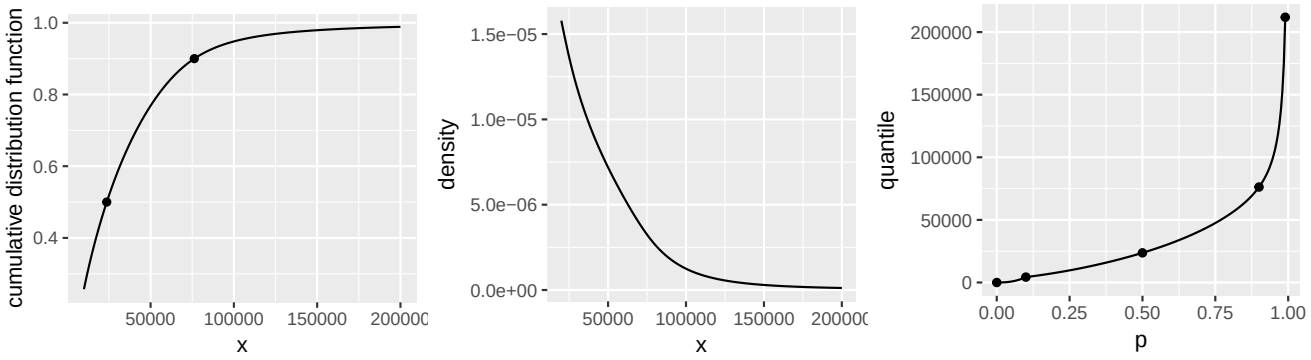
You can also plot the distribution directly. Here are the Lorenz curve and the generalized Pareto curve:

```r
plot_lorenz(distribution, xlim = c(0, 1))
plot_gpc(distribution, xlim = c(0.1, 1))
```



The black dots in these plots correspond to the points included in the tabulation, and therefore known with certainty. The other common plots available are:

```r
plot_cdf(distribution, xlim = c(1e4, 2e5)) # Cumulative distribution function
plot_density(distribution, xlim = c(2e4, 2e5)) # Density
plot_quantile(distribution, xlim = c(0, 0.99)) # Quantile
```



3.2 Other Interpolation Options

Sometimes, a tabulation may include the bracket shares but not the bracket thresholds, or the bracket thresholds but not the bracket shares. In such cases, you can use the functions `shares_fit` and `thresholds_fit`:

```
distribution_shares_only <- shares_fit(p, bracketavg = a, average = average)
# You can specify the "average" argument in thresholds_fit if you have it
distribution_thresholds_only <- thresholds_fit(p, q)
```

Both functions employ methods derived from generalized Pareto interpolation, but because they use less information they can be less precise. They return objects to be manipulated exactly like the output of `tabulation_fit`. As you can see, the results from the three functions are quite similar:

```
top_share(distribution_shares_only, 0.95)
#> [1] 0.2689886
top_share(distribution_thresholds_only, 0.95)
#> [1] 0.2667667
# Both top 5% shares are similar to what we get with the complete tabulation
top_share(distribution, 0.95)
#> [1] 0.271917
```

3.3 Transformation of Distributions

After interpolation, the package can perform three types of transformations of the distributions: merging, individualization and adding up.

3.3.1 Merge distributions

You can merge several distributions using `merge_dist`. The most common use case is if you have a distribution for several countries, and you want to know the distribution for the region defined by these countries. You may also apply it for regions within a country, or two different populations (e.g. migrants and natives).

Let's interpolate two distributions for total income in France and the United States in 2010:

```
distribution_us <- tabulation_fit(
    p = c(0.1, 0.5, 0.9, 0.99),
    threshold = c(5665, 31829, 96480, 351366),
    bracketavg = c(18030.88, 54936.66, 151099.1, 1068912),
```

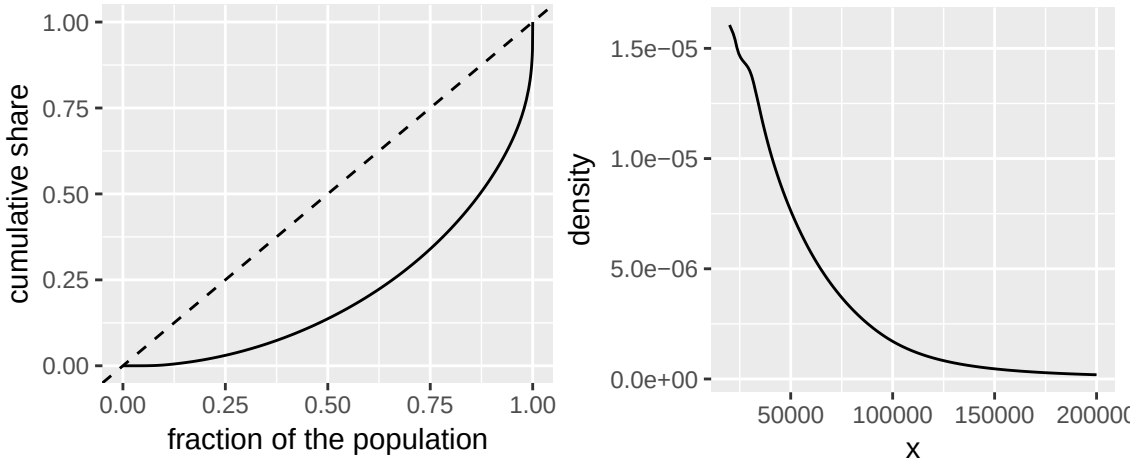
```
    average = 53587.59
)
distribution_fr <- tabulation_fit(
    p = c(0.264, 0.329, 0.432, 0.589, 0.761, 0.915, 0.984),
    average = 27727.804,
    threshold = c(11682.243, 14018.692, 17523.364, 23364.486,
        35046.729, 58411.215, 116822.43),
    bracketavg = c(12867.991, 15882.009, 20246.495, 28734.813,
        44450.935, 76572.43, 226254.673)
)
# We can check that inequality is higher in France than in the US
top_share(distribution_us, 0.99)
#> [1] 0.1994701
top_share(distribution_fr, 0.99)
#> [1] 0.1026875
```

To merge distributions, we apply `merge_dist` to the two previously interpolated distribution, alongside the population size for each of them:

```
distribution_us_fr <- merge_dist(
    dist = list(distribution_us, distribution_fr),
    popsize = c(2.257e+08, 36962517)
)
```

It returns an object that can be used like the output of `tabulation_fit`:

```
# Top 1% share of the region US + FR
top_share(distribution_us_fr, 0.99)
#> [1] 0.1951543
# Lorenz curve and density
plot_lorenz(distribution_us_fr, xlim = c(0, 1))
plot_density(distribution_us_fr, xlim = c(2e4, 2e5))
```



3.3.2 Individualization

Income tax tabulations refer to *tax units*, which in many cases correspond more or less to *households*. Using households as the statistical unit can make result overly dependent on the demographic structure of

the population, so it is preferable to "individualize" the distribution. The package can do so under the convention that income is split equally between spouses, with the function `individualize_dist`.

To do so, we need the fraction of singles (or equivalently couples) in several bracket. For the case of France, we have:

```
p_singles <- c(0, 0.264, 0.329, 0.432, 0.589, 0.761, 0.915, 0.984)
share_singles <- c(0.872, 0.872, 0.872, 0.756, 0.557, 0.261, 0.15, 0.15)
```

The values for `p_singles` do not have to be the same as the one in the original income tabulation. We use the function `individualize_dist` as:

```
distribution_fr_indiv <- individualize_dist(distribution_fr,
    p_singles, singlebracket = share_singles)
# Compare top 1% share before and after individualization: the individualized
# distribution is a little more equal
top_share(distribution_fr, 0.99)
#> [1] 0.1026875
top_share(distribution_fr_indiv, 0.99)
#> [1] 0.09104723
```

The method assumes that within each bracket, the income of singles is the same as the income of couples. This is almost certainly an approximation, but in practice it works well because the income difference between singles and couples is overwhelmingly a between-bracket phenomenon and not a within-bracket phenomenon. However, this assumption can be changed using the argument `ratio` (see `?individualize_dist`).

3.3.3 Adding up components

If you have the distribution of labor income on the one hand, and the distribution of capital income on the other hand, and know the dependence between both components, you can get the distribution of total income using `addup_dist`. Take the distribution of labor and capital income in the United States:

```
distribution_us_labor <- tabulation_fit(
    p = c(0.1, 0.5, 0.9, 0.99),
    threshold = c(4130, 23686, 76252, 211861),
    bracketavg = c(12643.3, 43908.3, 108329.2, 471463.3),
    average = 37208.059
)
distribution_us_capital <- tabulation_fit(
    p = c(0.1, 0.5, 0.9, 0.99),
    threshold = c(-1176, 2780, 28939, 173917),
    bracketavg = c(328.6372, 10657.18, 59412.08, 688689.3),
    average = 16370.471
)
# Capital income is more unequal than labor
top_share(distribution_us_labor, 0.99)
#> [1] 0.12671
top_share(distribution_us_capital, 0.99)
#> [1] 0.42069
```

To add up both distributions, we will assume that their dependency follows a Gumbel copula with parameter θ. The higher θ, the stronger the dependence, with $\theta = 1$ meaning full independence, and $\theta = +\infty$ meaning

perfect correlation of the ranks. The value $\theta = 1.5$ is usually a good fit for the distribution of labor and capital income.

```
distribution_us_added_up <- addup_dist(distribution_us_labor,
    distribution_us_capital, theta = 1.5)
# Compare top 1% share for total income interpolated directly and total income
# calculated via addup_dist: both are very close
top_share(distribution_us, 0.99)
#> [1] 0.1994701
top_share(distribution_us_added_up, 0.99)
#> [1] 0.2023528
```

3.3.4 Combine transformations

The package does not allow you to transform distributions that were already transformed, because the computational costs increase very fast with the number of layers of transformation. So for example we cannot merge directly the previous output of `addup_dist` with `distribution_fr`:

```
distribution_us_fr2 <- merge_dist(
    dist = list(distribution_us_added_up, distribution_fr),
    popsize = c(2.257e+08, 36962517)
)
#> Error in FUN(X[[i]], ...): 'dist' objects must be of class 'gpinter_dist_orig'
```

A very simple and efficient workaround is to export a tabulation from the first transformation, and interpolate it again. You should export a very detailed tabulation to keep as much information as possible from your initial transformation. You can do so quickly using `generate_tabulation`.

```
tabulation_us_added_up <- generate_tabulation(distribution_us_added_up,
    fractiles = c(seq(0.01, 0.99, 0.01), seq(0.991, 0.999, 0.001),
        seq(0.9991, 0.9999, 0.0001), seq(0.99991, 0.99999, 0.00001))
)
```

When re-interpolating from such a detailed tabulation, you should use the option `fast = TRUE`. It will use a simpler but faster interpolation method (mean-split histogram). Indeed, generalized Pareto interpolation can be slower and is unnecessary when you have such a detailed tabulation in input.

```
distribution_us_added_up2 <- tabulation_fit(
    p = tabulation_us_added_up$fractile,
    threshold = tabulation_us_added_up$threshold,
    bracketavg = tabulation_us_added_up$bracket_average,
    average = distribution_us_added_up$average,
    fast = TRUE
)
distribution_us_fr2 <- merge_dist(
    dist = list(distribution_us_added_up2, distribution_fr),
    popsize = c(2.257e+08, 36962517)
)
# Compare the top 1% shares obtained from the US distribution directly
# interpolated, and the US distribution added up from labor and capital income.
# As you can see both are very close.
top_share(distribution_us_fr, 0.99)
```

```
#> [1] 0.1951543
top_share(distribution_us_fr2, 0.99)
#> [1] 0.1980893
```