

Journal of Statistical Software

September 2026, Volume 117, Issue 6. doi: 10.18637/jss.v117.i06

LorenzRegression: An Implementation of the Lorenz and Penalized Lorenz Regressions in R

Alexandre Jacquemain
UCLouvain

Cédric Heuchenne
University of Liège

Abstract

Lorenz regressions are statistical tools for measuring the extent of inequality in a response variable that is attributable to a set of covariates. In the context of a single-index model, these regression techniques estimate the explained Gini coefficient, a measure of inequality in the conditional expectation of the response given the covariates. In this paper, we describe the **LorenzRegression** package, which implements the non-penalized and penalized Lorenz regressions. The non-penalized procedure is implemented via a genetic algorithm, making use of the **GA** package. For the penalized case, the user can choose between using a lasso or SCAD penalty. In the former case, the estimation procedure is performed with the FABS algorithm, while the latter is implemented with the SCAD-FABS algorithm. Computationally intensive parts of the code are written in C++. A mathematical description of the procedure is provided. The main functions and S3 methods are carefully described and their use is illustrated on income data.

Keywords: Gini coefficient, single-index models, penalized regression, economic inequality, R.

1. Introduction

Lorenz regressions (Heuchenne and Jacquemain 2022) and penalized Lorenz regressions (Jacquemain, Heuchenne, and Pircalabelu 2024) are used in economics to measure the extent of inequality in a response variable Y that can be attributed to a set of covariates X. This is formalized by maximizing the concentration index of the response with respect to a linear combination of the covariates.

In the context of a single-index model, the method provides inference for the explained Gini coefficient, a measure of the inequality in the conditional expectation of Y given X. Due to the presence of a nonparametric link function, the single-index model is more flexible than the parametric models often used in inequality measurement. Furthermore, an estimator for

the explained Gini coefficient can be obtained without having to estimate the link function. The penalized Lorenz regression automatically selects the relevant covariates and avoids overestimation of the explained Gini coefficient. Both the non-penalized and penalized methods come with a proper inferential procedure.

The **LorenzRegression** package is the first and only implementation of the Lorenz regression methodology across all statistical software platforms. This R package (R Core Team 2026) not only provides a novel approach for inequality measurement but also offers comprehensive tools for visualizing inequality through Lorenz and concentration curves and computing summary indices via the Gini coefficient and concentration index. We refer the reader to Section 2.1 for a definition of these objects. All computations may include sample weights. Inequality measurement does not benefit from a widespread implementation in R, even though a few packages are publicly available from the Comprehensive R Archive Network (CRAN). The **laeken** package (Alfons and Templ 2013) estimates indicators of social exclusion from complex surveys and can be used to compute Gini coefficients. The **ineq** package (Zeileis 2014) estimates several inequality measures, including the Gini coefficient and the Lorenz curve. The **wINEQ** package (Wójcik, Giemza, Machowska, and Napora 2024) estimates inequality measures for weighted data, including the Gini coefficient. While we are not aware of any package available on CRAN, an open source implementation of the concentration index is proposed in the **rineq** package (Devleesschauwer, Willimès, Van Malderen, Konings, Speybroeck, and Meili 2025). It also comes with a parametric decomposition method, following the procedure proposed by Speybroeck *et al.* (2010). Beyond R, other statistical softwares also provide tools for inequality measurement. For instance, Stata (StataCorp 2025) offers a comprehensive suite of commands, including `ineqdec0` for inequality indices and their decomposition by subgroups (Jenkins 1999), `lorenz` for Lorenz and concentration curves (Jann 2016), `glcurve` for generalized Lorenz curves (Jenkins and Van Kerm 2004), and `iop` for the measurement of inequality of opportunity (Chávez Juárez and Soloaga 2014). We refer the reader to the Stata manual for more information. While these are robust tools for inequality analysis, none of them currently implement the Lorenz regression methodology, reinforcing the uniqueness of the **LorenzRegression** package.

Estimation in the Lorenz regression is based on maximizing a discrete objective function. Due to lack of differentiability, the numerical solution is obtained through a genetic algorithm, implemented via the R package **GA** (Scrucca 2013). In the penalized Lorenz regression, the issue is circumvented by replacing the discrete objective function with a smooth approximation. The numerical solution is obtained via the FABS algorithm (Shi, Huang, Huang, and Ma 2018) in the case of a lasso penalty and via the SCAD-FABS algorithm (Jacquemain *et al.* 2024) if the SCAD penalty is chosen instead. In the **LorenzRegression** package, the main functions are implemented in C++ and integrated in R, via the **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel and Balamuta 2018) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014) packages. To further reduce computation time, parallel computing is implemented with the **foreach** package (Kane, Emerson, and Weston 2013; Microsoft and Weston 2022). Finally, all plots are produced with **ggplot2** (Wickham 2016).

The paper is organized as follows. In Section 2, we introduce the model settings, the explained Gini coefficient and the Lorenz regression procedure. Section 3 focuses on the penalized case. Section 4 presents the main function of the package and the associated S3 methods. Section 5 illustrates the use of the package on a real-data example. Finally, Section 6 concludes.

2. The Lorenz regression

2.1. Model settings and the explained Gini coefficient

We consider an economic outcome $Y \in \mathbb{R}$ with $0 < \mathsf{E}[Y] < \infty$, where $\mathsf{E}[\cdot]$ is the expected value, and a vector of p covariates $X = (X_1, \ldots, X_p)^\top \in \mathbb{R}^p$. We start by introducing measures of inequality. The Lorenz curve of Y evaluated at p is defined as

$$\mathrm{LC}_Y(p) = \frac{\mathsf{E}[Y \mathbb{1}\{F_Y(Y) \leq p\}]}{\mathsf{E}[Y]},$$

where $F_Y(\cdot)$ is the cumulative distribution function (CDF) of Y and $\mathbb{1}(\cdot)$ is the indicator function. It measures the share of total outcome held by the proportion p of the individuals with the smallest values of Y. In case of perfect equality, the Lorenz curve coincides with the 45° line. The Gini coefficient is twice the area between the 45° line and the Lorenz curve. It is also equivalent to the following definition

$$\mathrm{Gi}_Y = \frac{2\mathsf{COV}[Y, F_Y(Y)]}{\mathsf{E}[Y]},$$

where $\mathsf{COV}[\cdot, \cdot]$ is the covariance operator. When the ordering is produced by another variable, say X_k, the inequality measure thus obtained is called the concentration index, defined as

$$\mathrm{CI}_{Y,X_k} = \frac{2\mathsf{COV}[Y, F_{X_k}(X_k)]}{\mathsf{E}[Y]}.$$

Similarly, the Lorenz curve becomes the concentration curve.

The Lorenz regression aims to maximize the concentration index of the outcome with respect to a linear combination of the covariates. Formally, it solves the optimization problem

$$\max_\theta \frac{2\mathsf{COV}[Y, F_\theta(X^\top\theta)]}{\mathsf{E}[Y]}, \tag{1}$$

where $F_\theta(\cdot)$ is the CDF of $X^\top\theta$, and θ is constrained to satisfy $||\theta|| = 1$, with $||\cdot||$ representing the L2-norm. In other words, the Lorenz regression finds the linear index of covariates that produces a ranking of observations most strongly associated with variation in outcome values. This method has applications in various fields. In health economics, inequity is often assessed by computing the concentration index of health with respect to living standards. However, direct measures of standards of living, such as consumption, are prone to measurement error (O'Donnell, Van Doorslaer, Wagstaff, and Lindelow 2008). A common alternative consists in using an asset index as a proxy. In this context, the Lorenz Regression provides a principled way to construct a linear asset index that maximizes the concentration index of health so that the predicted ranking of living standards best reflects disparities in health outcomes.

In the right inferential framework, the Lorenz regression gains an extra level of interpretation. Assume the single-index model

$$\mathsf{E}[Y|X = x] = H(x^\top\theta_0), \tag{2}$$

where $\mathsf{E}[\cdot|\cdot]$ is the conditional expectation, $H(\cdot)$ is a strictly increasing link function and θ_0 is a vector of unknown parameters that satisfies $||\theta_0|| = 1$, where this constraint ensures model identifiability. In this model, the ranking of $\mathsf{E}[Y|X]$ is entirely determined by $X^\top\theta_0$.

This makes the index interpretable as a ranking score, i.e., a scalar measure that summarizes how the covariates position an individual within the distribution of expected outcomes. The explained Gini coefficient (Heuchenne and Jacquemain 2022) is defined as

$$\mathrm{Gi}_{Y,X} = \mathrm{Gi}_{\mathsf{E}[Y|X]}, \tag{3}$$

and satisfies $\mathrm{Gi}_{Y,X} \leq \mathrm{Gi}_Y$. While Gi_Y measures the inequality of the observed response variable, $\mathrm{Gi}_{Y,X}$ focuses on the inequality of the response as predicted by the covariates. In a similar way, we define the explained Lorenz curve as the Lorenz curve of $\mathsf{E}[Y|X]$. In the single-index model, and as detailed in Heuchenne and Jacquemain (2022), $\mathrm{Gi}_{Y,X}$ is the solution to Equation 1. This allows the Lorenz Regression to identify the explained Gini coefficient and, more generally, to estimate the underlying ranking of conditional expectations without estimating the link function $H(\cdot)$, which is typically a complex nonparametric object. This is because the Lorenz Regression relies only on the ordering of the linear index, which is sufficient to fully estimate the explained Gini coefficient. Viewing the Lorenz regression within the context of the single-index model provides further applications to the method. In inequality of opportunity (IOP), the focus is on quantifying how much income is determined by circumstances beyond the individual's control. One way to formalize this consists in estimating the Gini coefficient of the conditional expectation of income with respect to circumstances. This is often done by regressing income (or log-income) on circumstances and measuring IOP with the Gini coefficient of predicted incomes (Bourguignon, Ferreira, and Menéndez 2007). The Lorenz regression aligns naturally with this task by estimating this quantity under a single-index model and without having to estimate the nonparametric part of the model. The linear index directly represents the advantage ranking, i.e., the ranking of predicted income given the circumstances. This provides a direct interpretation for the parameter vector θ_0.

Beyond the inequality of opportunity framework, Mélard, Jacquemain, Perelman, SILNE-R Consortium, and Lorant (2025) uses the Lorenz regression to study disparities in school-level tobacco control policies across European cities. In that context, the response variable is a policy-need ratio, i.e., a measure of whether policies are implemented in proportion to school-specific needs. Its Gini coefficient provides an indicator of inequity in policy implementation. Covariates include characteristics of both schools and students. The explained Gini coefficient is used to quantify the share of inequity that is attributable to these observed characteristics.

The Lorenz regression focuses on estimating the index and the explained Gini coefficient under minimal assumptions. In particular, it does not rely on a specific structure for the error term, nor on assumptions about its distribution. The model assumptions are summarized in Jacquemain *et al.* (2024), in the more general setting of the penalized Lorenz regression. The central assumption is that of a single-index structure with strictly increasing link function. This allows consistent estimation of the index parameter θ_0 and of the explained Gini coefficient, without requiring a fully specified model. That said, users may wish to go one step further and obtain predictions for the conditional expectation $\mathbb{E}[Y|X]$. As explained in Section 2.2, this requires estimating the link function $H(\cdot)$ in a second, nonparametric step. Once $H(\cdot)$ is estimated, residuals can be computed and used for diagnostics. For example, plotting residuals against the estimated index or covariates can help assess potential violations of the single-index assumption. These diagnostics, however, evaluate the entire modeling strategy and should not be interpreted as direct tests of the core Lorenz regression itself. Lastly, the package does not provide inference for the predicted outcomes, as its primary goal is to estimate the index and the explained Gini coefficient under weak structural assumptions.

2.2. Estimation procedure and algorithm

The estimation procedure developed hereunder is based on Heuchenne and Jacquemain (2022), and is adapted to incorporate sample weights. Let $(X_i^\top, Y_i)^\top$, with $i = 1, \ldots, n$ be an i.i.d sample with the same distribution as $(X^\top, Y)^\top$. Assume that we have at our disposal a vector of sample weights $\pi = (\pi_1, \ldots, \pi_n)^\top$ satisfying $\pi_i \geq 0 \ \forall i = 1, \ldots, n$ and $\sum_{i=1}^n \pi_i = 1$. We define the relative rank of observation i in the vector $X^\top\theta$ as

$$\hat{F}_i(\theta) = \sum_{j=1}^{n} \pi_j \Big(\mathbb{1}\{X_i^\top\theta \geq X_j^\top\theta\} - \mathbb{1}\{X_i^\top\theta = X_j^\top\theta, U_i > U_j\} \Big), \tag{4}$$

where $(U_1, \ldots, U_n)^\top$ are i.i.d uniform $(0, 1)$ random variables. The second indicator is an adaptation to ties and is discussed in Section 3 of Heuchenne and Jacquemain (2022). An alternative to this random allocation would consist in using the mean rank, where $\mathbb{1}\{X_i^\top\theta = X_j^\top\theta, U_i > U_j\}$ is replaced by $\frac{1}{2}\mathbb{1}\{X_i^\top\theta = X_j^\top\theta\}$.

In the Lorenz regression procedure, the vector θ_0 and the explained Gini coefficient are estimated with

$$\hat{\theta}_{\mathrm{LR}} = \mathrm{argmax}_{\theta, ||\theta||=1} \sum_{i=1}^{n} \pi_i Y_i \hat{F}_i(\theta) \tag{5}$$

$$\widehat{\mathrm{Gi}}_{Y,X}^{\mathrm{LR}} = \frac{2}{\overline{Y}} \sum_{i=1}^{n} \pi_i Y_i \hat{F}_i(\hat{\theta}_{\mathrm{LR}}) - 1, \tag{6}$$

where $\overline{Y} = \sum_{i=1}^n \pi_i Y_i$. The explained Lorenz curve is then estimated with the concentration curve of the response Y with respect to estimated index $X^\top\hat{\theta}_{\mathrm{LR}}$. We define the Lorenz-$R^2$ as

$$\mathrm{LR}^2 = \frac{\widehat{\mathrm{Gi}}_{Y,X}^{\mathrm{LR}}}{\widehat{\mathrm{Gi}}_Y},$$

where $\widehat{\mathrm{Gi}}_Y$ is the estimated Gini coefficient of Y. The Lorenz-R^2 represents the proportion of inequality explained by the model via an index ranging between 0 and 1. The whole procedure does not entail to estimate the link function $H(\cdot)$. Notice however that $\theta \mapsto \hat{F}_i(\theta)$ is discrete. In the **LorenzRegression** package, the optimization programme displayed in Equation 5 is solved via a genetic algorithm, implemented in the function `Lorenz.GA()`. Its usage is summarized as follows.

```
Lorenz.GA(y, x, standardize = TRUE, weights = NULL, suggestions = NULL,
  ties.method = c("random","mean"), ties.Gini = c("random","mean"),
  seed.random = NULL, seed.Gini = NULL, seed.GA = NULL)
```

The only required arguments are the vector of responses `y` and the $(n \times p)$ matrix of covariates `x`. The argument `standardize` determines whether covariates should be standardized prior to estimation. Default value is `TRUE`, in which case covariates are standardized. A vector of sample weights can be provided using the argument `weights`. By default, it is set to `NULL`, in which case all observations are given a weight of $1/n$. The argument `suggestions` allows the user to provide initial guesses to guide the search process of the genetic algorithm. Since this algorithm is a heuristic search method, providing good initial values may reduce computation time or improve the quality of the fit. The argument can be set either to a

numeric matrix of candidate solutions or to the character string `"OLS"`, in which case half of the initial population is generated as random perturbations around the OLS estimator. By default, `suggestions = NULL`, meaning that the initial population is entirely randomly initialized. The arguments `ties.method` and `ties.Gini` indicate how ties are handled in the computation of $\hat{F}_i(\theta)$. By default, both arguments are set to `"random"`, meaning that Equation 4 is used. If `"mean"` is chosen, the mean rank solution is used instead. The existence of two separate arguments stems from the fact that ties influence first the estimation of θ_0 through Equation 5 and, second, the estimation of $\text{Gi}_{Y,X}$ through Equation 6. To ensure the comparability with other methods, one could choose to estimate the explained Gini coefficient with the mean rank solution, even though the random allocation is used to estimate θ_0. In that case, one would set `ties.method = "random"` but `ties.Gini = "mean"`. The argument `seed.random` sets a fixed seed for generating the vector of uniform random variables during the estimation of θ_0 when `ties.method` is set to `"random"`. Similarly, `seed.Gini` sets a seed during the estimation of $\text{Gi}_{Y,X}$ when `ties.Gini` is set to `"random"`. Lastly, `seed.GA` imposes a seed for the genetic algorithm itself. In all cases, the seed setting is local, meaning that the random seed is restored to its previous state after each operation. If any of these arguments is `NULL` (the default), the corresponding seed is not set. We refer the reader to the help file of `Lorenz.GA()` for details on additional arguments.

In a genetic algorithm, the idea is to start from a population of solution candidates and iteratively make this population evolve such that it is composed of solutions with an increasingly higher score. The evolution is governed by the principles of crossover and mutation, while the score is assessed via a fitness function. We refer the reader to Scrucca (2013) for a more complete description and the implementation in R via the **GA** package. Internally, `Lorenz.GA()` calls function `ga()` from the **GA** package, with the following fitness function

$$f(\theta) = \left[1 - \left|||\theta|| - 1\right|\right] \sum_{i=1}^{n} \pi_i Y_i \hat{F}_i(\theta),$$

where the factor multiplying the sum ensures that the final solution satisfies the norm constraint $||\theta|| = 1$. To speed up computations, the function is written in C++ and integrated in R using **RcppArmadillo**.

Once the linear index has been estimated, users may optionally go one step further and estimate the conditional expectation, which requires estimating the link function $H(\cdot)$. In the **LorenzRegression** package, this is handled by the function `Rearrangement.estimation()`, which implements a two-step procedure. First, the response Y is regressed nonparametrically on the estimated index $X^\top \hat{\theta}_{\text{LR}}$. Second, the resulting function is rearranged to enforce strict monotonicity using the method proposed by Chernozhukov, Fernández-Val, and Galichon (2009) and implemented in the function `rearrangement()` from the **Rearrangement** (Graybill, Chen, Chernozhukov, Fernandez-Val, and Galichon 2016) package. The usage of the function is summarized below.

```
Rearrangement.estimation(y, index, t = index, weights = NULL,
  method = "loess", ...)
```

The arguments `y` and `index` correspond to the response and the estimated index, respectively. The points t at which $H(t)$ should be evaluated are specified via `t`; by default, the original data points are used. Optional sample weights can be passed using `weights`. The `method` argument

controls the nonparametric smoothing procedure in the first step. By default, a `loess()` smoother is used, and its behavior can be tuned using the `...` argument. Alternatively, users can fully customize the smoothing method, as illustrated in Section 5.

3. The penalized Lorenz regression

In practice, an economist may want to determine the explained Gini coefficient from a large set of covariates, without having to perform a pre-selection prior to estimation. In that context, the Lorenz regression procedure detailed in the last section suffers from two issues. First, similarly to the R^2 in linear regression, $\widehat{\text{Gi}}^{\text{LR}}_{Y,X}$ never decreases and, in practice, slightly increases as we introduce new covariates, even if they are actually irrelevant. On a large set of covariates, the explained Gini coefficient is therefore prone to be overestimated. Second, the genetic algorithm may fail to converge when the number of covariates becomes large.

To solve these issues, Jacquemain *et al.* (2024) introduce a penalized estimation procedure, where the discrete empirical distribution function displayed in Equation 4 is replaced by a differentiable approximation. We now turn to this method.

3.1. Estimation procedure and algorithm

The penalized Lorenz regression solves

$$\hat{\theta}_{\text{PLR}} = \text{argmax}_{\theta,||\theta||=1}\left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \pi_i \pi_j Y_i K\left(\frac{X_i^\top \theta - X_j^\top \theta}{h}\right) - \sum_{k=1}^{p} p_\lambda(|\theta_k|)\right\}, \tag{7}$$

where $K(\cdot)$ is the integral of a kernel function, h is a bandwidth, $p_\lambda(\cdot)$ is a nonconcave penalty function and $\lambda > 0$ is a penalty parameter. We advocate to use the SCAD penalty, which satisfies

$$p'_\lambda(x) = \begin{cases} \lambda & \text{if } x \leq \lambda \\ \frac{a\lambda - x}{a-1} & \text{if } \lambda < x \leq a\lambda \\ 0 & \text{if } x > a\lambda, \end{cases} \tag{8}$$

where $x > 0$ and $a > 2$ is an arbitrary constant. In what follows, we call "active" covariates those for which $\theta_{0,k} \neq 0$ and "non-active" those satisfying $\theta_{0,k} = 0$, where $\theta_{0,k}$ is the kth element of θ_0. The SCAD penalty ensures that the estimated coefficients of non-active covariates are set to 0 with a probability tending to one. Also, the estimated explained Gini coefficient is asymptotically normal, with a convergence rate unaffected by the selection process. We refer the reader to Theorems 2.1. and 2.3. in Jacquemain *et al.* (2024) for more details. An alternative would consist in using the lasso penalty, where $p_\lambda(x) = \lambda x$. An estimator $\widehat{\text{Gi}}^{\text{PLR}}_{Y,X}$ for the explained Gini coefficient is obtained by plugging $\hat{\theta}_{\text{PLR}}$ in Equation 6. In practice, inference can be performed using bootstrap.

In the **LorenzRegression** package, the penalized Lorenz regression can be implemented using either the lasso or the SCAD penalty. The former is solved using the FABS algorithm proposed by Shi *et al.* (2018), which has been adapted to our objective function and implemented in the function `Lorenz.FABS()`. The latter is solved using the SCAD-FABS algorithm proposed by Jacquemain *et al.* (2024) and implemented in the function `Lorenz.SCADFABS()`. In both

cases, the computationally intensive parts of the function are written in C++ and integrated into R using **RcppArmadillo**.

In what follows, we focus on the `Lorenz.SCADFABS()` function. We first provide a brief explanation of the algorithm, then turn to the usage of the function. For further details, we refer the reader to Section 3.2 of Jacquemain *et al.* (2024). The SCAD-FABS solves

$$\min_{\theta} Q(\theta) = L(\theta) + \sum_{k=1}^{p} p_\lambda(|\theta_k|)$$

where $Q(\cdot)$ is called the objective function, $L(\cdot)$ is called the loss function and $p_\lambda(\cdot)$ is the SCAD penalty. At the start of the algorithm, all coefficients are set to zero. Then, at each iteration, only one coefficient is updated by a fixed amount of $+$ or $-$ ϵ, where $\epsilon > 0$ is the step size of the algorithm. This operation either reduces the size of the coefficient (backward step) or increases it (forward step). Let θ^t and θ^{t+1} denote the vectors of coefficients at iterations t and $t+1$. The update rule is of the form

$$\begin{aligned} \theta^{t+1} &= \theta^t - \text{sign}(\theta_k^t)\mathbf{1}_k\epsilon && \text{(backward step)} \\ \theta^{t+1} &= \theta^t - \text{sign}\left(\nabla_k L(\theta^t)\right)\mathbf{1}_k\epsilon && \text{(forward step)} \end{aligned}$$

where $\mathbf{1}_k$ denotes the vector of size p taking value 1 at the kth component and 0 everywhere else, θ_k^t is the kth element of θ^t, ∇ is the gradient vector and ∇_k its kth element. The fact that a forward step necessarily increases the amplitude of the coefficient is settled by Lemma 3.1. in Jacquemain *et al.* (2024). The index k corresponding to that of the updated coefficient is determined by

$$\begin{aligned} k &= \arg\min_{l \in \mathcal{A}^t} \left\{ -\nabla_l Q(\theta^t)\text{sign}(\theta_l^t) \right\} && \text{(backward step)} \\ k &= \text{argmax}_{l=1,\ldots,p} \left\{ |\nabla_l L(\theta^t)| - p'_\lambda(|\theta_l^t|) \right\} && \text{(forward step)} \end{aligned}$$

where $\mathcal{A}^t = \{k \in \{1,\ldots,p\} : \theta_k^t \neq 0\}$ and $p'_\lambda(|\theta_k^t|)$ is defined according to Equation 8 for $|\theta_k^t| > 0$ and $p'_\lambda(|\theta_k^t|)$ is set to λ if $\theta_k^t = 0$. The SCAD-FABS solves the minimization problem for a path of λ values, which may be imposed by the user or determined within the algorithm, as implemented in the FABS algorithm proposed by Shi *et al.* (2018). In the penalized Lorenz regression, we incorporate the constraint $||\theta|| = 1$ by considering the Lagrangian function related to our problem. Accordingly, the loss function is

$$L(\theta) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \pi_i \pi_j Y_i K\left(\frac{X_i^\top \theta - X_j^\top \theta}{h}\right) + \gamma \sum_{k=1}^{p} \theta_k^2, \tag{9}$$

where $\gamma > 0$. The main arguments of the `Lorenz.SCADFABS()` function are described in the following code excerpt.

```
Lorenz.SCADFABS(y, x, standardize = TRUE, weights = NULL, kernel = 1,
  h = length(y)^(-1/5.5), gamma = 0.05, a = 3.7, lambda = "Shi",
  eps = 0.005, SCAD.nfwd = NULL)
```

As in the function `Lorenz.GA()`, `y`, `x` and `weights` provide the data and optional sample weights. We strongly recommend to let `standardize = TRUE` as, without standardization, covariates are not treated on equal footing by the penalized procedure.

The smoothness of the objective function is determined by the kernel and the bandwidth. Concerning the former, we propose two choices: either an Epanechnikov kernel (set by `kernel = 1`, the default) or a biweight kernel (set by `kernel = 2`). These kernels satisfy the assumptions required by the theory, as highlighted in Jacquemain *et al.* (2024). The bandwidth h is determined via the argument `h`. To ensure the $\sqrt{n}$-consistency of the estimated explained Gini coefficient, it should satisfy $nh^6 \to 0$ and $nh^5(\log(h^{-1}))^{-1} \to \infty$, where n is the sample size. Therefore, we suggest to set it to $h = Cn^{-1/5.5}$, where $C > 0$ is a constant. The default value is $h = n^{-1/5.5}$. The argument `gamma` corresponds to the Lagrange multiplier of Equation 9, with a default value of 0.05.

The SCAD penalty depends on two parameters, a and λ. The former is ruled by the argument `a`, with default value of 3.7, as advised in Fan and Li (2001). The argument `lambda` takes three possible values. By default, it is set to `"Shi"`, where the path of λ values is determined within the algorithm, as in the FABS procedure of Shi *et al.* (2018). If set to `"grid"`, the path is defined by a grid, equidistant in the log scale. Otherwise, the user may provide a vector of values.

The step size ϵ of the SCAD-FABS algorithm is set by the argument `eps`. A smaller value implies a finer path but a larger computation time. The default value is 0.005. Section 3.2 in Jacquemain *et al.* (2024) highlights an interesting relationship between the FABS and SCAD-FABS algorithms. Both algorithms start from an empty solution, where all coefficients are set to zero, and iteratively increase the size of the coefficients. Besides, the lasso and SCAD penalties are identical for small coefficients. This means that up to a certain point, both algorithms provide identical paths. Jacquemain *et al.* (2024) argue that this divergence point is influenced by the tuning parameters of the SCAD-FABS algorithm. This is formalized by an optional parameter, n_{fwd}. If this parameter is set to a small value, the paths of the two algorithms will quickly diverge. Conversely, if n_{fwd} is large enough, the paths will coincide until the algorithm stops. This parameter is encoded in the `SCAD.nfwd` argument.

3.2. Selection methods

The FABS and SCAD-FABS algorithms return vectors of estimated coefficients for a path of λ values, ranging from high to low penalization. Choosing the appropriate λ is crucial. Additionally, these algorithms depend on several tuning parameters. For example, users may want to fit the regression over a grid of bandwidth values and automatically select the optimal one. Denote by ρ the vector of parameters affecting the fit of the regression. In the **LorenzRegression** package, three methods are available to select ρ: BIC, bootstrap and cross-validation.

Denote by $\widehat{\text{Gi}}(\rho)$ the estimated explained Gini coefficient obtained for a given value of ρ. A first option consists in maximizing the following BIC criterion

$$\text{BIC-score}_\rho = \log(\widehat{\text{Gi}}(\rho)) - k_\rho \frac{\log(n)}{2n},$$

where k_ρ is the number of covariates selected using ρ. This criterion is an adaptation of the BIC score proposed by Lin and Peng (2013). Another possibility benefits from the bootstrap procedure used to perform inference on the explained Gini coefficient. Let R denote the number of bootstrap resamples and $r = 1, \ldots, R$. At each iteration r, two samples are produced. First, a training sample (X_r^*, Y_r^*) is obtained by drawing with replacement from

the original data. Second, a validation sample $(\tilde{X}_r^*, \tilde{Y}_r^*)$ is obtained as the out-of-bags, i.e., the data unused in the training sample. The SCAD-FABS algorithm is run on (X_r^*, Y_r^*) and yields an estimated vector of coefficients $\hat{\theta}_r^*(\rho)$. An out-of-bag score for iteration r, $\widetilde{\text{Gi}}_r^*(\rho)$, is obtained by plugging $\hat{\theta}_r^*(\rho)$ and $(\tilde{X}_r^*, \tilde{Y}_r^*)$ in Equation 6. The bootstrap selection procedure chooses the value ρ that maximizes the out-of-bag score

$$\text{OOB-score}_\rho = \frac{1}{R}\sum_{r=1}^{R} \widetilde{\text{Gi}}_r^*(\rho).$$

A final option uses k-fold cross-validation, where the data are split into k folds of roughly equivalent sizes. For each iteration, $(k-1)$ folds are used as training samples and the remaining one is used as validation sample. The procedure is then similar to the bootstrap selection. Call the score of the jth iteration $\widetilde{\text{Gi}}_{-j}(\rho)$. The cross-validation selection procedure chooses the value ρ that maximizes the cross-validation score

$$\text{CV-score}_\rho = \frac{1}{k}\sum_{j=1}^{k} \widetilde{\text{Gi}}_{-j}(\rho).$$

4. The LorenzRegression package

4.1. The `Gini.coef()`, `Lorenz.curve()`, and `Lorenz.graphs()` functions

The **LorenzRegression** package offers several functions to analyze the inequality pattern characterizing the input data. We first describe the function `Gini.coef()`, which computes a Gini coefficient or a concentration index. Its arguments are

- `y`: Vector gathering the variable of interest for each observation.
- `x`: Vector gathering the variable to use for the ranking of each observation. By default, it is set to `y` and the function computes the Gini coefficient of `y`. If the user supplies a vector `x`, the output becomes the concentration index of `y` with respect to `x`.
- `na.rm`: Should missing values be deleted? Default value is `TRUE`. If `FALSE` is selected, missing values generate an error message.
- `ties.method`: Determines how ties are handled. By default, the argument is set to `"mean"` and the average rank is used. If set to `random`, the random allocation is used instead.
- `seed`: Fixes what seed is used if the random allocation is chosen. Default is `NULL`, in which case no seed is imposed. As in the case of `Lorenz.GA()`, the seed setting is only local in the sense that it is reverted to its previous state after the operation.
- `weights`: Optional vector of sample weights. Default is `NULL`, in which case each observation is given a weight of $1/n$, where n is the number of observations.

With similar arguments, the function `Lorenz.curve()` can either produce the Lorenz curve of `y` or the concentration curve of `y` with respect to `x`. The output is a function, whose unique argument is a proportion between 0 and 1.

Finally, the function `Lorenz.graphs()` allows the user to plot simultaneously the Lorenz curve of a variable, as well as several concentration curves. It takes the following main arguments:

- `formula`: An object of class '`formula`' specifying the variable of interest and the covariates to be used for ranking.
- `data`: A data frame containing the variables displayed in the formula.
- `difference`: A logical value determining whether the vertical axis should represent the deviation from perfect equality. The default is `FALSE`. Setting this argument to `TRUE` can improve readability when the data exhibit little inequality or when the concentration curves are close to each other.

The output of the function is a graph displaying the Lorenz curve of `Y` and the concentration curves of `Y` with respect to the covariates specified in the formula. The formula `Y ~ 1` is used to display the Lorenz curve of `Y` only.

4.2. The `Lorenz.Reg()` function

This section discusses the core function of the **LorenzRegression** package. The function `Lorenz.Reg()` allows the user to fit a Lorenz or penalized Lorenz regression. It acts as a wrapper for the fitting functions `Lorenz.GA()`, `Lorenz.FABS()` and `Lorenz.SCADFABS()`. Its main arguments are described in the following code excerpt.

```
Lorenz.Reg(formula, data, weights, na.action,
  penalty = c("none", "SCAD", "LASSO"),
  grid.arg = c("h", "SCAD.nfwd", "eps", "kernel", "a", "gamma"),
  grid.value = NULL, ...)
```

The usage is similar to standard regression functions. The only required argument is `formula` that specifies the univariate response variable and the covariates, via an object of class '`formula`'. The variables are either taken from the environment from which the function is called, or specified by `data`. Optional sample weights can be specified by the `weights` argument. Finally, the management of missing observations is dealt with `na.action`.

The `penalty` argument determines the type of Lorenz regression to be fitted. By default, the value `"none"` is chosen and a non-penalized regression is fitted using `Lorenz.GA()`. Otherwise a penalized regression is computed, either using `Lorenz.FABS()` if `penalty = "LASSO"`, or using `Lorenz.SCADFABS()` if `penalty = "SCAD"`.

As observed in Section 3, the fit of the penalized regression depends on several tuning parameters. The user can specify a grid for one tuning parameter via the arguments `grid.arg` and `grid.value`. The `grid.arg` argument is a character string that specifies the tuning parameter for which the grid is to be constructed. The `grid.value` argument is a numeric vector that defines the grid values. Internally, the fitting function is executed separately for each value in the specified grid. The final fit therefore depends on two parameters: the penalization parameter and the chosen tuning parameter.

The arguments specific to the fitting functions were described in Sections 2 and 3 and can be passed through the `...` argument.

The function `Lorenz.Reg()` returns an object of class '`LR`' when fitting a non-penalized Lorenz regression, or an object of class '`PLR`' when fitting a penalized Lorenz regression. The object of class '`LR`' is a list that contains the components:

- `theta`: The estimated vector of parameters of the single-index model, $\hat{\theta}_{\text{LR}}$.
- `Gi.expl`: The estimated explained Gini coefficient, $\widehat{\text{Gi}}^{\text{LR}}_{Y,X}$.
- `LR2`: The Lorenz-R^2 of the regression.

In the case of the penalized Lorenz regression, the fit depends on the penalization and tuning parameters. The object of class '`PLR`' contains the components:

- `path`: A list of the same length as `grid.value`. Each element is a matrix containing information about the solution path for a specific value of the tuning parameter. The first row displays the vector of λ values. The second and third rows provide the Lorenz-R^2 and estimated explained Gini coefficient for each λ value. The fourth row shows the evolution of the BIC score. The remaining rows display the evolution of the estimated coefficients of the single-index model.
- `grid.idx`: The index of the optimal value of the tuning parameter in terms of BIC-score
- `lambda.idx`: The index of the optimal λ value in terms of BIC-score.

Contrary to the BIC method, bootstrap and cross-validation are not directly available as they involve additional computations. These methods are implemented in the functions `Lorenz.boot()` and `PLR.CV()`, see Sections 4.3 and 4.4 respectively.

For both classes, the list also includes more technical components such as the original call made by the user. The following S3 methods are available to provide detailed information about the model fit: `print()`, `summary()`, `ineqExplained()`, `coef()`, `fitted()`, `predict()`, `residuals()`, `autoplot()` (returns a '`ggplot`' object) and `plot()` (displays the plot). These methods are illustrated in Section 5. Notice, however, that inference is handled separately through bootstrap methods, as discussed in the next section.

4.3. The `Lorenz.boot()` function

The function `Lorenz.boot()` fits a Lorenz regression or a penalized Lorenz regression on bootstrap replicates. In the penalized case, it is also used to select the penalization and tuning parameters, via the method described in Section 3.2. The bootstrap replicates are generated by the function `boot()` from the R package **boot** (Canty and Ripley 2024). In this way, the user can adapt the bootstrap procedure to their needs. However, it is important to notice that sample weights should not be incorporated in the resampling because these are already taken into account in the regression procedure. The main arguments of the function are described in the following code excerpt.

```
Lorenz.boot(object, R, store_LC = FALSE, show_progress = TRUE, ...)
```

The argument `object` is an object inheriting either from the S3 class '`LR`' or from the S3 class '`PLR`'. This object contains the model fitted on the original data. The number of bootstrap resamples is specified by `R`. The argument `store_LC` determines whether ordinates of the explained Lorenz curve should be stored for each bootstrap resample. This is useful for plotting bootstrap-based confidence bands, as discussed in Section 5. However, storing these ordinates may significantly increase the size of the object, particularly in the penalized case. Hence, the default is `FALSE`. The argument `show_progress` controls whether a progress bar is displayed during the bootstrap procedure. It is set to `TRUE` by default but is automatically disabled when parallel computing is used. The argument `...` is used to pass arguments to the function `boot()`. For example, and as illustrated in Section 5, the arguments `parallel`, `ncpus`, and `cl` can be specified to enable parallel computing and thereby reduce the computational time required for the bootstrap procedure.

The function returns an updated version of `object`, with an additional class: '`LR_boot`' in the non-penalized case, or '`PLR_boot`' in the penalized case. The returned list now includes a `boot_out` component. This component is an object of class '`boot`' that contains the output of the bootstrap calculation. In the penalized case, further updates are performed due to the availability of a new selection method. The `lambda.idx` and `grid.idx` components are vectors indicating the positions of the optimal parameters for each selection method. Additionally, the `path` matrix has a new row added, representing the out-of-bag score.

As illustrated in Section 5, the role of the bootstrap procedure differs depending on the type of regression. In the non-penalized Lorenz regression, bootstrap allows for inference about both the explained Gini coefficient and the parameter vector θ_0. In the penalized case, inference about θ_0 is not provided due to the sparsity induced across bootstrap replicates. However, inference about the explained Gini coefficient remains valid. Additionally, in the penalized setting, the bootstrap can be used as a selection method via an out-of-bag scoring rule, as discussed in Section 3.2.

4.4. The `PLR.CV()` function

In the penalized case, cross-validation can be used to select the penalization and tuning parameters, see Section 3.2. The function `PLR.CV()` implements this method. First, it splits the data into folds using the function `vfold_cv()` from the R package **rsample** (Frick, Chow, Kuhn, Mahoney, Silge, and Wickham 2025). The user can therefore benefit from the flexibility of `vfold_cv()` to perform more specific sampling. Second, a Lorenz regression or a penalized Lorenz regression is fitted on each training sample and the cross-validation score is computed. This score is then used as a selection method. The main arguments of the function are described in the following code excerpt.

```
PLR.CV(object, k, ...)
```

The structure is similar to that of the function `Lorenz.boot()`. However, `object` must inherit from the S3 class '`PLR`'. The argument `k` specifies the number of folds. Finally, the argument `...` is used to pass arguments to the function `vfold_cv()`.

The function returns an updated version of `object`, with the additional class '`PLR_cv`'. Similar to the function `Lorenz.boot`, the positions of the optimal parameters based on the CV-score are stored in the vectors `lambda.idx` and `grid.idx`. Additionally, a row is added to the `path` matrix with the CV-score.

5. Illustration

We start by loading the packages required in this example.

```
R> library("LorenzRegression")
R> library("ineq")
```

To ensure accessibility, we use the colorblind-friendly Okabe-Ito palette.

```
R> cbf_palette <- palette.colors()
```

Throughout this illustration, we use the dataset `Ilocos` from the R package **ineq** (Zeileis 2014), which contains economic data on 632 households in the Philippines. We use total household income (`income`) as response variable and the following covariates:

- `sex`: A factor with two levels, `"male"` (518 observations) and `"female"` (114 observations), indicating the biological sex of the household head.
- `family.size`: A numerical variable indicating the family size. Results range from 1 to 13, with a mean of 5.19.
- `urbanity`: A factor with two levels, `"rural"` (301 observations) and `"urban"` (331 observations).
- `province`: A factor with four levels: `"Ilocos Norte"` (65 observations), `"Ilocos Sur"` (68 observations), `"La Union"` (116 observations) and `"Pangasinan"` (383 observations).

The data are loaded with

```
R> data("Ilocos", package = "ineq")
```

Descriptive analysis

The Gini coefficient of `income` and the concentration index of `income` with respect to `family.size` are computed using `Gini.coef()`.

```
R> Gi <- Gini.coef(Ilocos$income)
R> Gi

[1] 0.427

R> CI.mean <- Gini.coef(Ilocos$income, Ilocos$family.size)
R> CI.mean

[1] 0.0893
```

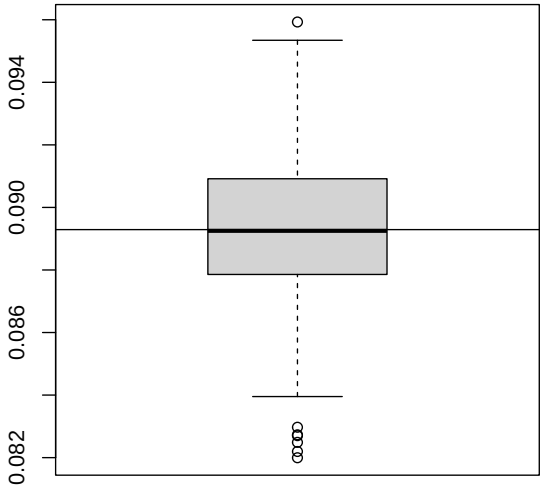


Figure 1: Concentration index of `income` with respect to `family.size` – Random allocation.

This concentration index measures the income inequality observed when households are ranked by family size. A positive value indicates that higher incomes are more prevalent among larger families, demonstrating a positive association between income and family size. Conversely, if larger families were associated with lower incomes, the concentration index would be negative. Since `family.size` is a discrete covariate, ties appear in the ordering generated by this variable. In the computation of the concentration index, the mean rank is used by default. In what follows, we use the random allocation instead, repeating the operation 1000 times with a different seed.

```
R> CI.random <- sapply(1:1000, function(i) Gini.coef(Ilocos$income,
+    Ilocos$family.size, ties.method = "random", seed = i))
R> boxplot(CI.random)
R> abline(h = CI.mean)
```

Figure 1 displays a boxplot of the concentration indices thus obtained. The distribution is centered on the solution obtained with the mean rank, indicated by the horizontal line. The size of the boxplot indicates how the concentration index varies as we change the rule determining how ties are broken. We now compute the Lorenz curve of `income` and evaluate it on a small grid of values.

```
R> LC <- Lorenz.curve(Ilocos$income)
R> p <- seq(0, 1, length.out = 5)
R> LC(p)

[1] 0.0000 0.0791 0.2142 0.4455 1.0000
```

The `Lorenz.graphs()` function is used to produce visual representations of Lorenz and concentration curves. The plot is stored as a '`ggplot`' object for easy customization. The main arguments are `formula` and `data`. The function draws the Lorenz curve of the variable on the

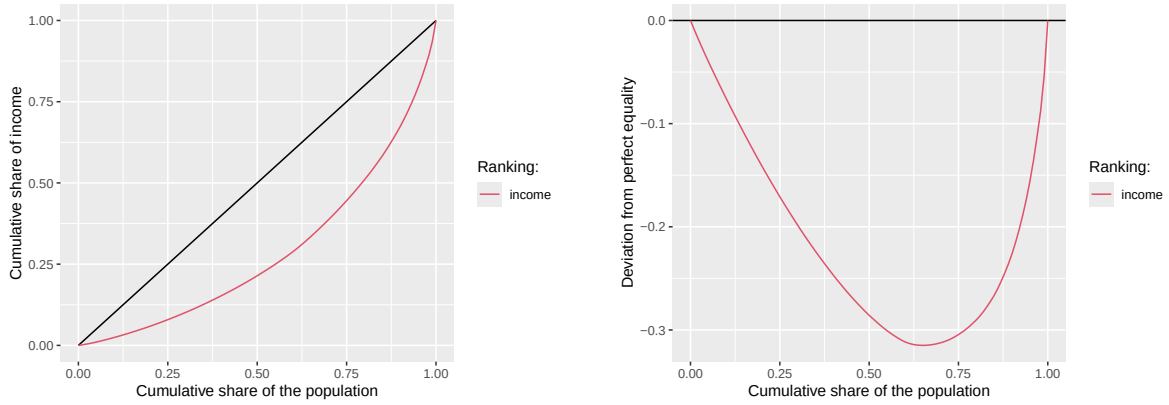


Figure 2: Lorenz curve of `income`.

left-hand side of the formula, along with its concentration curves with respect to the variables on the right-hand side. To plot only the Lorenz curve of `income`, the right-hand side of the formula is set to a constant, as in `income ~ 1`. This ensures that no concentration curves are displayed.

```
R> Lorenz.graphs(income ~ 1, data = Ilocos)
R> Lorenz.graphs(income ~ 1, data = Ilocos, difference = TRUE)
```

The result is represented in Figure 2. To enhance readability, the vertical axis can be expressed as a deviation from perfect equality. This is achieved by setting `difference = TRUE`.

To display concentration curves, one or more covariates can be added to the right-hand side of the formula. Here, we display the concentration curve of `income` with respect to `family.size`, see Figure 3. The `palette` argument allows specifying the color scheme. By default, it uses the base R palette, but it can be set to any vector of colors. The first color in the vector is used to represent the line of equality, while the remaining colors are used to distinguish the Lorenz and concentration curves.

```
R> Lorenz.graphs(income ~ family.size, data = Ilocos, palette = cbf_palette)
```

The concentration curve displayed in blue represents the cumulative share of income held by households when they are ordered by family size. Because income is positively associated with family size, the curve lies below the line of equality. In case of a negative association, the curve would lie above the line of equality.

The Lorenz regression

We fit the Lorenz regression to the `Ilocos` data, with `income` as response variable. We start by including `family.size` as only covariate.

```
LR0 <- Lorenz.Reg(income ~ family.size, data = Ilocos, ties.Gini = "mean")
```

Fitting this model is trivial because of the norm constraint imposed on the coefficient vector in Equation 2. Equivalently, from Equation 1, we see that the covariates only matter through

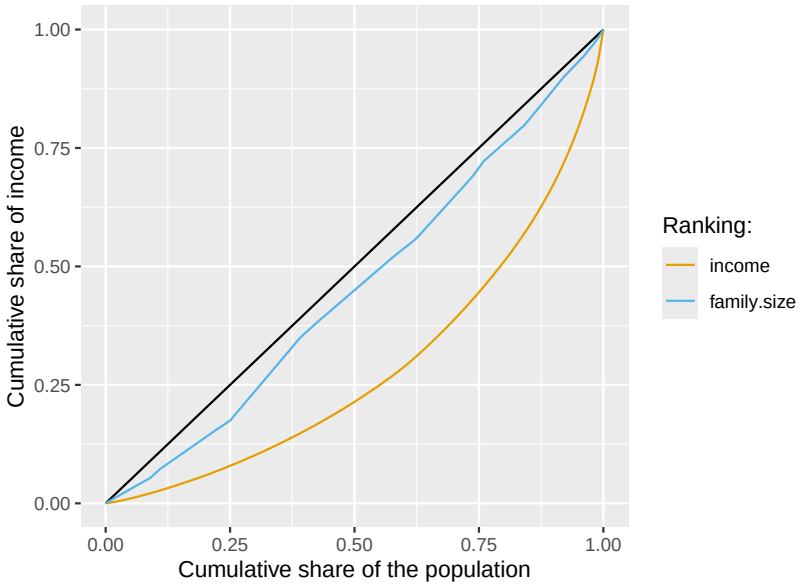


Figure 3: Lorenz curve of `income` (orange) and concentration curve of `income` with respect to `family.size` (blue).

the ranking they generate. Therefore, in a univariate setup, only the sign of the coefficient matters, not its amplitude. The coefficient is either 1 (positive association) or -1 (negative association). We can retrieve it with the method `coef()`. Here, the value of the coefficient is 1 since family size is positively associated with income.

```
R> coef(LR0)

family.size
          1
```

In this univariate setup, the explained Gini coefficient is the absolute value of the concentration index and is retrieved with the method `ineqExplained()`.

```
R> ineqExplained(LR0)

[1] 0.0893
```

We now introduce further covariates: `sex`, `urbanity` and `province`. We also include interactions between `sex` and `family.size`, and between `sex` and `urbanity`. We use the mean rank in the estimation of the explained Gini coefficient, by setting `ties.Gini = "mean"`. The results of the genetic algorithm displayed in this section were obtained on macOS Sonoma 14.7 using vecLib BLAS and LAPACK version 3.12.0. Minor variations in results may occur depending on the specific linear algebra library used and the operating system.

```
LR <- Lorenz.Reg(income ~ sex * family.size + sex * urbanity + province,
  data = Ilocos, ties.Gini = "mean", seed.GA = 123, seed.random = 456)
```

The vector of estimated coefficients is again retrieved with the method `coef()`.

```
R> coef(LR)
```

```
              sexmale           family.size         urbanityurban
              0.23452               0.08316               0.87628
   provinceIlocos Sur      provinceLa Union    provincePangasinan
              0.12147              -0.13083              -0.21207
  sexmale:family.size sexmale:urbanityurban
              0.00814              -0.30546
```

Measures of explained inequality are obtained with the method `ineqExplained()`. We retrieve the explained Gini coefficient by setting `type = "Gini.explained"` (default option) and the Lorenz-R^2 by setting `type = "Lorenz-R2"`.

```
R> ineqExplained(LR, type = "Gini.explained")
```

```
[1] 0.155
```

```
R> ineqExplained(LR, type = "Lorenz-R2")
```

```
[1] 0.362
```

A summary of the model is provided with the method `summary()`.

```
R> summary(LR)
```

```
Call:
Lorenz.Reg(formula = income ~ sex * family.size + sex * urbanity +
    province, data = Ilocos, ties.Gini = "mean", seed.GA = 123,
    seed.random = 456)

Explained Inequality Table

 Explained Total Lorenz-R2
     0.155 0.427     0.362

 (Explained inequality is measured by the explained Gini coefficient.
  Total inequality is the Gini coefficient of the response.
  The Lorenz-R2 is the proportion of inequality explained by the model.)

Coefficients Table for the Single-Index Model
                     Estimate
sexmale                  0.23
family.size              0.08
urbanityurban            0.88
provinceIlocos Sur       0.12
```

```
provinceLa Union         -0.13
provincePangasinan       -0.21
sexmale:family.size       0.01
sexmale:urbanityurban    -0.31
```

As we already observed, the estimated explained Gini coefficient is of 15.5%, which corresponds to 36.2% of the observed inequality. The summary also displays a table with the estimated coefficients. Compared to regression `LR0`, we see that the coefficient attached to `family.size` is greater than zero. This means that, conditional on the other covariates, its impact is still positive. Also, including additional covariates increased the explained inequality from 8.9% to 15.5%, as reflected in the explained Gini coefficient. Finally, since the model has not yet been bootstrapped, no inference is displayed in the coefficients table.

As discussed in Section 1, the Lorenz regression estimates a linear index of covariates whose ranking is most strongly associated with the outcome. Under the single-index model assumption, this linear index also identifies the ranking of the expected outcome. Predictions from the Lorenz regression are obtained using the `predict()` method, either on the original data or on new observations provided via the argument `newdata`. An additional argument `type` specifies whether the predictions concern the linear index (`type = "index"`, the default) or the expected outcome (`type = "response"`). We first illustrate the use of `predict` to estimate the index on a small set of hypothetical profiles.

```
R> pred_data <- data.frame(
+    "sex" = rep(c("male", "female"), 4),
+    "family.size" = rep(c(2, 6, 2, 6), each = 2),
+    "urbanity" = rep(c("urban", "rural"), each = 4),
+    "province" = rep("Pangasinan", 8)
+  )
R> pred_data$rank_pred <- rank(predict(LR, newdata = pred_data))
R> pred_data <- pred_data[order(pred_data$rank_pred), ]
R> pred_data

     sex family.size urbanity   province rank_pred
6 female           2    rural Pangasinan         1
5   male           2    rural Pangasinan         2
8 female           6    rural Pangasinan         3
7   male           6    rural Pangasinan         4
1   male           2    urban Pangasinan         5
2 female           2    urban Pangasinan         6
3   male           6    urban Pangasinan         7
4 female           6    urban Pangasinan         8
```

The `rank_pred` column gives the predicted rank of the index for each observation. Under the assumption of the single-index model, these ranks can be interpreted as a ranking of predicted income. In this example, women with small families living in rural areas rank the lowest, while women with larger families in urban areas rank the highest.

Predictions for the expected outcome are obtained by setting `type = "response"` in the `predict()` method. In this case, the link function $H(\cdot)$ is estimated using the function

`Rearrangement.estimation()` described in Section 2.2. Its behavior can be adjusted, as in the following example, where the span parameter is modified.

```
R> pred_data$pred_loess <- predict(LR, newdata = pred_data,
+    type = "response", span = 0.25)
```

Alternatively, users can fully customize the smoothing procedure by specifying `method` as a list with

- `fit_fun`, a function used to fit the model. It must accept `y`, `x`, and optionally `weights`.
- `predict_fun`, a function to generate predictions from the fitted object. It must accept the fitted object and a `newdata` argument.

The following example uses a smoothing spline in place of the default `loess` method.

```
R> smooth_spline_method <- list(
+    fit_fun = function(y, x, ...) {
+      smooth.spline(x = x, y = y, ...)
+    },
+    predict_fun = function(fit, newdata) {
+      predict(fit, x = newdata[,1])$y
+    }
+  )
R> pred_data$pred_spline <- predict(LR, newdata = pred_data,
+    type = "response", method = smooth_spline_method)
```

The resulting predictions using both methods are shown below.

```
R> pred_data
```

```
     sex family.size urbanity   province rank_pred pred_loess pred_spline
6 female           2    rural Pangasinan         1      46092       37879
5   male           2    rural Pangasinan         2      52551       58599
8 female           6    rural Pangasinan         3      55415       65360
7   male           6    rural Pangasinan         4      88190       88771
1   male           2    urban Pangasinan         5     101851      105770
2 female           2    urban Pangasinan         6     104419      110291
3   male           6    urban Pangasinan         7     151599      135984
4 female           6    urban Pangasinan         8     159864      137817
```

Alternatively, predictions on the existing data can be obtained using the method `fitted()`. As for `predict()`, the function has a `type` argument. When the argument is set to `"index"` (the default), the fitted index is returned, while when the argument is set to `"response"`, the whole conditional expectation is estimated.

```
R> Ilocos$index_LR <- fitted(LR)
R> Lorenz.graphs(income ~ family.size + index_LR, Ilocos,
+    palette = cbf_palette)
```

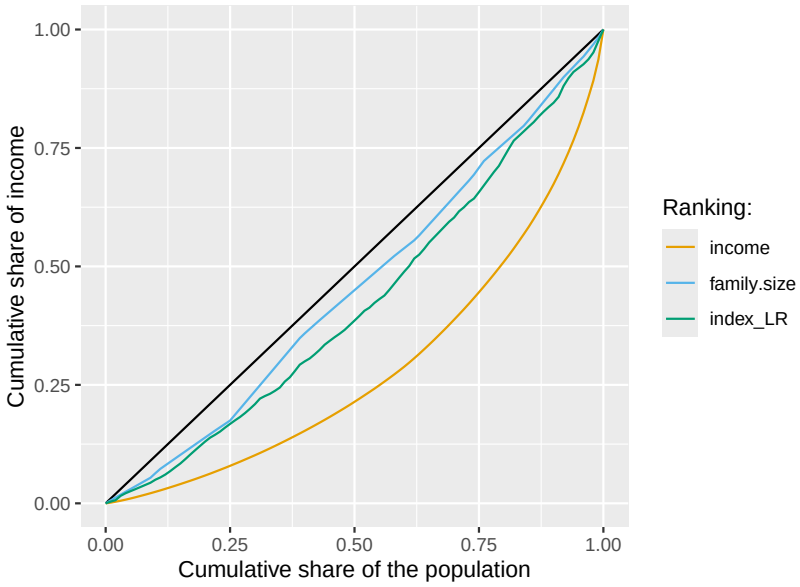


Figure 4: Lorenz curve of `income` (orange), explained Lorenz curve using `family.size` only (blue) and explained Lorenz curve from the full non-penalized Lorenz regression (green).

We can display explained inequality via concentration and Lorenz curves. In the code excerpt above, we make use of the function `Lorenz.graphs()` to produce Figure 4.[1] The orange curve is the Lorenz curve of `income`. The blue curve is the concentration curve of `income` with respect to `family.size`. Finally, the green curve is the explained Lorenz curve obtained from the Lorenz regression. The closer the green curve to the orange curve, the greater the Lorenz-R^2, i.e., the more observed inequality is explained by the regression model.

Residuals can be computed as the difference between the observed and fitted values. These residuals are extracted using the method `residuals()`. Additionally, a diagnostic plot displaying residuals against the estimated index can be produced by setting `type = "residuals"` in the method `plot()`. This diagnostic aims to assess the goodness-of-fit of the full model, which combines the estimation of the index (via the Lorenz regression) and the subsequent estimation of the link function.

```
R> plot(LR, type = "residuals", method = smooth_spline_method)
```

Inference about the model parameters can be obtained after performing bootstrap calculations with the function `Lorenz.boot()`. In this example, we set the number of bootstrap replicates to 500 and enable parallel computing on 6 CPUs to decrease computation time. We also set `store_LC = TRUE` to store the ordinates of the explained Lorenz curve for each bootstrap sample. This feature will be used later for constructing confidence bands.

```
R> set.seed(123)
R> LR_boot <- Lorenz.boot(LR, R = 500, store_LC = TRUE,
+    parallel = "multicore", ncpus = 6)
```

[1]The method `plot()` with `type = "explained"` would produce the same graph without the blue curve.

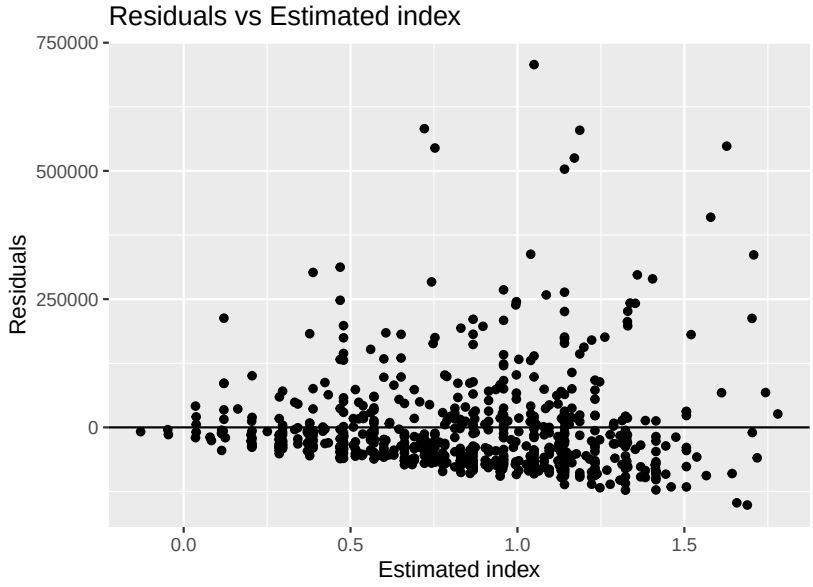


Figure 5: Residual plot of the non-penalized Lorenz regression.

Using `summary()`, the coefficients table is updated to include bootstrap standard errors, z-values and p-values.

```
Coefficients Table for the Single-Index Model
                     Estimate Std. Error z value Pr(>|z|)
sexmale               0.23452    0.17882    1.31    0.190
family.size           0.08316    0.04507    1.84    0.065 .
urbanityurban         0.87628    0.08275   10.59   <2e-16 ***
provinceIlocos Sur    0.12147    0.17631    0.69    0.491
provinceLa Union     -0.13083    0.16524   -0.79    0.429
provincePangasinan   -0.21207    0.11809   -1.80    0.073 .
sexmale:family.size   0.00814    0.02914    0.28    0.780
sexmale:urbanityurban -0.30546    0.19170   -1.59    0.111
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

At a 5%-level of significance, we see that only `urbanityurban` is significant. A confidence interval for the explained Gini coefficient is obtained with the method `confint()`.

```
R> confint(LR_boot)

 2.5 % 97.5 %
 0.109  0.183
```

By default, 95% confidence intervals are constructed, but the level can be changed using the argument `level`. Three bootstrap methods are available and are chosen via the argument `type`. Parametric boostrap (`"norm"`, the default) uses the asymptotic normality of

the estimated explained Gini coefficient and estimates the variance by bootstrap. Percentile bootstrap (`"perc"`) directly plugs in the quantiles of the estimated explained Gini coefficient obtained on the bootstrap resamples. Finally, basic bootstrap (`"basic"`) is based on bootstrapping the whole distribution of the estimator.

Confidence intervals for each individual coefficient can be obtained by changing the `parm` argument to `"theta"`. We also change the significance level and the bootstrap method to illustrate the use of arguments `level` and `type`.

```
R> confint(LR_boot, parm = "theta", level = 0.9, type = "perc")
```

```
                           5 %   95 %
sexmale                -0.1282 0.4838
family.size             0.0215 0.1587
urbanityurban           0.6684 0.9099
provinceIlocos Sur     -0.1674 0.4113
provinceLa Union       -0.3212 0.1922
provincePangasinan     -0.3458 0.0559
sexmale:family.size    -0.0452 0.0479
sexmale:urbanityurban -0.5323 0.0669
```

Thanks to the `store_LC = TRUE` setting in the bootstrap procedure, the ordinates of the explained Lorenz curve were retained for each resample. This allows the construction of pointwise confidence bands around the estimated curve. These bands are automatically displayed by the `plot()` method when applied to the bootstrapped object. The `palette` argument works as in `Lorenz.graphs()`.

```
R> plot(LR_boot, palette = cbf_palette)
```

The penalized Lorenz regression

We fit a penalized Lorenz regression to the `Ilocos` data, with the same response variable and covariates as before. The penalty is the SCAD and we construct a grid for the bandwidth.

```
R> PLR <- Lorenz.Reg(income ~ sex * family.size + sex * urbanity + province,
+    data = Ilocos, penalty = "SCAD", grid.arg = "h",
+    grid.value = c(0.5, 1, 2) * nrow(Ilocos)^(-1/5.5))
```

At this stage, BIC is the only selection method available and is therefore used to select the pair (h, λ). The estimated coefficients obtained with the non-penalized and penalized procedures are similar. With BIC as a selection method, the SCAD penalty does not induce sparsity as no coefficient is set to zero.

```
R> coef.mat <- data.frame("NPLR" = coef(LR), "PLR_BIC" = coef(PLR))
R> coef.mat
```

```
                          NPLR PLR_BIC
sexmale                0.23452  0.3523
```

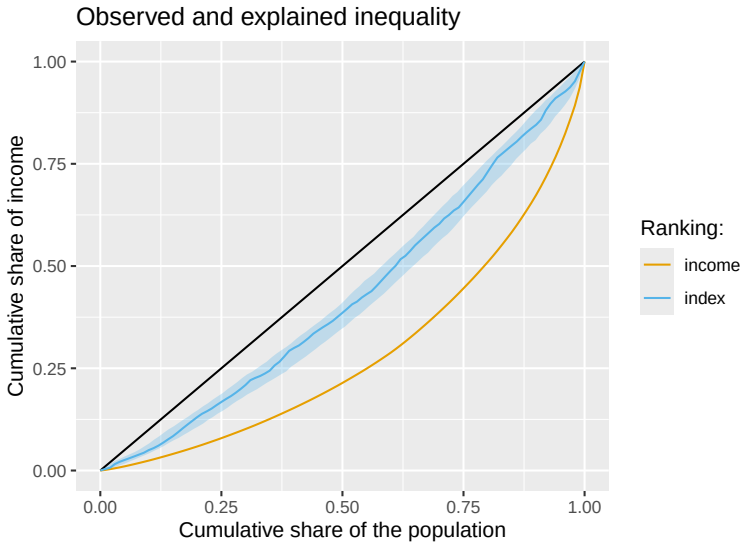


Figure 6: Explained Lorenz curve from the full non-penalized Lorenz regression with pointwise bootstrap confidence bands.

```
family.size             0.08316  0.0898
urbanityurban           0.87628  0.8339
provinceIlocos Sur      0.12147  0.0748
provinceLa Union       -0.13083 -0.0779
provincePangasinan     -0.21207 -0.1909
sexmale:family.size     0.00814 -0.0157
sexmale:urbanityurban -0.30546 -0.3523
```

In terms of explained inequality, the loss in the explained Gini coefficient due to the penalization is negligible.

```
R> ineqExplained(LR) - ineqExplained(PLR)
```

```
[1] 0.000608
```

In the penalized Lorenz regression, the bootstrap serves two purposes: it enables the selection of the pair (h, λ) via out-of-bag scoring, and it provides inference about the explained Gini coefficient.

```
R> set.seed(123)
R> PLR_boot <- Lorenz.boot(PLR, R = 500, parallel = "multicore", ncpus = 6)
```

When displaying the results of the fitted model, the choice of the selection method among those available is determined via the argument `pars.idx`, whose default value is `"BIC"`. Here, we set it to `"Boot"` in order to obtain the estimated coefficients using bootstrap selection.

```
R> coef.mat$PLR_Boot <- coef(PLR_boot, pars.idx = "Boot")
R> coef.mat
```

```
                           NPLR PLR_BIC PLR_Boot
sexmale                 0.23452  0.3523   0.0000
family.size             0.08316  0.0898   0.0989
urbanityurban           0.87628  0.8339   0.9951
provinceIlocos Sur      0.12147  0.0748   0.0000
provinceLa Union       -0.13083 -0.0779   0.0000
provincePangasinan     -0.21207 -0.1909   0.0000
sexmale:family.size     0.00814 -0.0157   0.0000
sexmale:urbanityurban -0.30546 -0.3523   0.0000
```

As a selection method, the bootstrap induces a very sparse model, including only two covariates (`family.size` and `urbanityurban`). However, adopting this simpler model results in only a 1 percentage point reduction in the explained Gini coefficient.

```
R> ineqExplained(PLR_boot, pars.idx = "Boot") - ineqExplained(PLR)

[1] -0.0104
```

We can retrieve confidence intervals for the explained Gini coefficient using the method `confint()`. In what follows, we first provide the confidence intervals obtained when (h, λ) is selected using BIC. Then, we provide the confidence intervals obtained with the bootstrap selection method.

```
R> confint(PLR_boot)

 2.5 % 97.5 %
 0.109  0.183

R> confint(PLR_boot, pars.idx = "Boot")

 2.5 % 97.5 %
0.0968 0.1751
```

The confidence level and the bootstrap method are again adjusted using the arguments `level` and `type`. Recall however that, in contrast with the non-penalized procedure, inference about the parameter vector is not available due to the sparsity induced in the bootstrap replicates.

We apply cross-validation as a third selection method using the function `PLR.CV()`, with the number of folds set to 5. The method `summary()`, provides an overview of the fitted model obtained using all three selection methods. The models selected by bootstrap and cross-validation are extremely similar.

```
R> PLR_boot_CV <- PLR.CV(PLR_boot, k = 5, seed.CV = 123)
R> summary(PLR_boot_CV)

Call:
Lorenz.Reg(formula = income ~ sex * family.size + sex * urbanity +
```

```
    province, data = Ilocos, penalty = "SCAD", grid.arg = "h",
    grid.value = c(0.5, 1, 2) * nrow(Ilocos)^(-1/5.5))

Explained Inequality Table

     Explained Total Lorenz-R2
BIC      0.154 0.427     0.361
Boot     0.144 0.427     0.336
CV       0.153 0.427     0.359

 (Explained inequality is measured by the explained Gini coefficient.
  Total inequality is the Gini coefficient of the response.
  The Lorenz-R2 is the proportion of inequality explained by the model.)

Estimated Coefficients Table for the Single-Index Model
                          BIC   Boot      CV
sexmale                0.3523 0.0000  0.2429
family.size            0.0898 0.0989  0.0955
urbanityurban          0.8339 0.9951  0.8967
provinceIlocos Sur     0.0748 0.0000  0.0845
provinceLa Union      -0.0779 0.0000 -0.0656
provincePangasinan    -0.1909 0.0000 -0.1881
sexmale:family.size   -0.0157 0.0000 -0.0033
sexmale:urbanityurban -0.3523 0.0000 -0.2845
```

We can visualize the explained inequality for each of the three selection procedures using the method `plot()`. Figure 7 displays the explained Lorenz curve obtained with the bootstrap selection procedure.

```
R> plot(PLR_boot_CV, pars.idx = "Boot", palette = cbf_palette)
```

We can also use the `Lorenz.graphs` function to add the explained Lorenz curves obtained with each selection procedure to the graph previously shown in Figure 4. To enhance readability, the vertical axis is expressed as a deviation from perfect equality by setting `difference = TRUE`. The resulting curves are displayed in Figure 8.

```
R> Ilocos$index_PLR_BIC <- fitted(PLR_boot_CV, pars.idx = "BIC")
R> Ilocos$index_PLR_Boot <- fitted(PLR_boot_CV, pars.idx = "Boot")
R> Ilocos$index_PLR_CV <- fitted(PLR_boot_CV, pars.idx = "CV")
R> Lorenz.graphs(income ~ family.size + index_LR + index_PLR_BIC +
+    index_PLR_Boot + index_PLR_CV, Ilocos, difference = TRUE,
+    palette = cbf_palette)
```

As with the non-penalized procedure, the penalized Lorenz regression allows predictions for both the index and the expected outcome. Below, we compare outcome predictions across methods.

```
R> pred_data2 <- pred_data[,1:4]
R> pred_data2$LR <- predict(LR, newdata = pred_data2, type = "response")
```

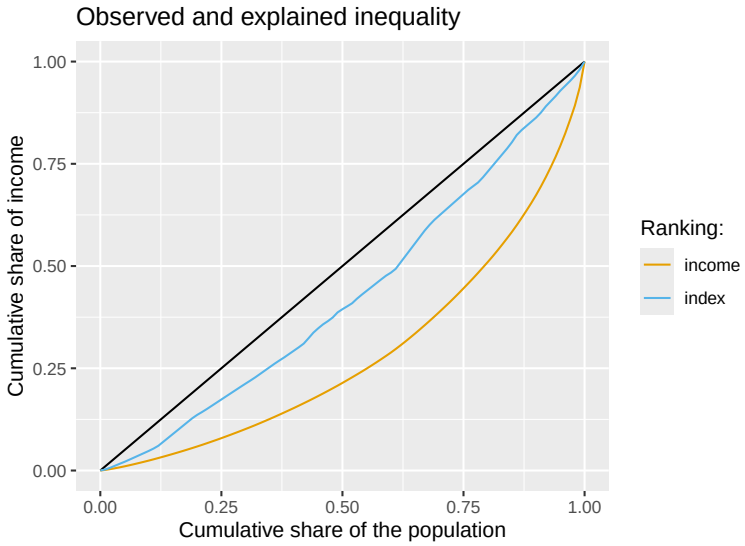


Figure 7: Lorenz curve of `income` (orange) and explained Lorenz curve from the penalized Lorenz regression using bootstrap selection (blue).

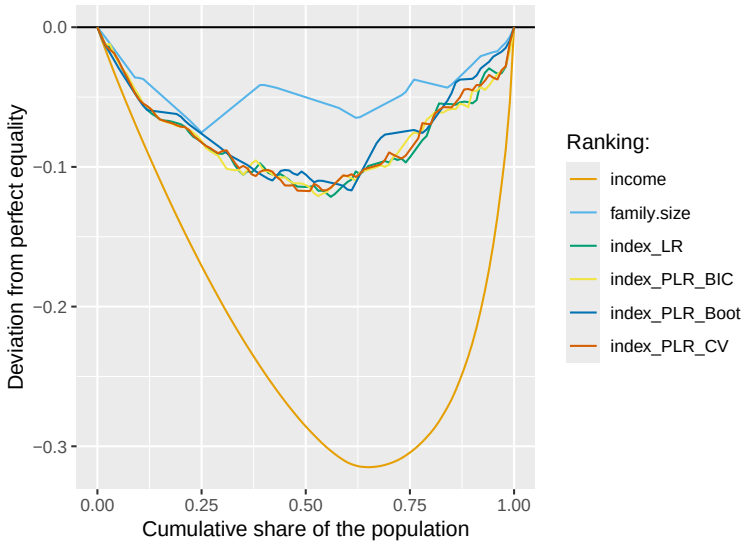


Figure 8: Lorenz curve of `income` (orange), explained Lorenz curve using `family.size` only (blue), explained Lorenz curve from the non-penalized Lorenz regression (green) and explained Lorenz curves from the penalized Lorenz regression using BIC, CV, and bootstrap selection methods (other colors).

```
R> pred_data2$PLR_BIC <- predict(PLR_boot_CV, newdata = pred_data2,
+    pars.idx = "BIC", type = "response")
R> pred_data2$PLR_Boot <- predict(PLR_boot_CV, newdata = pred_data2,
+    pars.idx = "Boot", type = "response")
R> pred_data2$PLR_CV <- predict(PLR_boot_CV, newdata = pred_data2,
+    pars.idx = "CV", type = "response")
```

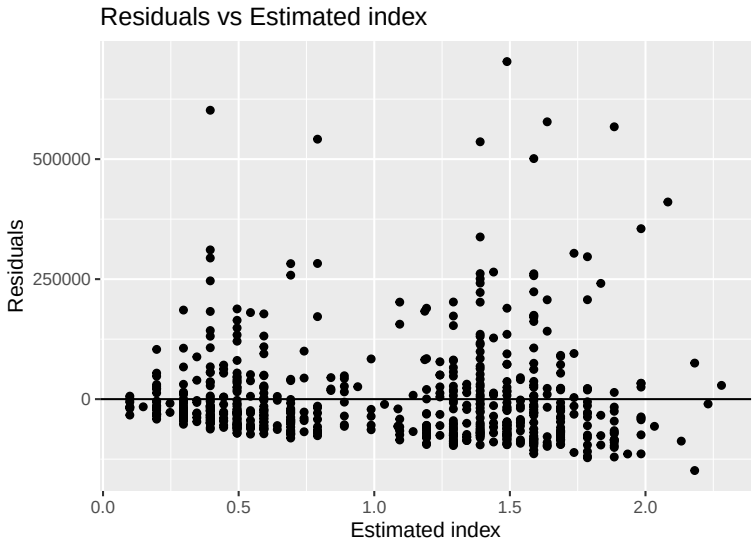


Figure 9: Residual plot of the penalized Lorenz regression with bootstrap selection.

```
R> pred_data2
```

```
     sex family.size urbanity   province     LR PLR_BIC PLR_Boot PLR_CV
6 female           2    rural Pangasinan  38686   42034    55646  34989
5   male           2    rural Pangasinan  61869   64647    55646  59673
8 female           6    rural Pangasinan  68804   67568    99206  72800
7   male           6    rural Pangasinan  90854   89872    99206  89717
1   male           2    urban Pangasinan 100500  101219   114326 102679
2 female           2    urban Pangasinan 105054  104360   114326 106834
3   male           6    urban Pangasinan 136290  134092   138294 135836
4 female           6    urban Pangasinan 137429  139633   138294 138927
```

Finally, we display a residual plot for the model selected via the bootstrap criterion.

```
R> plot(PLR_boot_CV, type = "residuals", pars.idx = "Boot")
```

Diagnostics specific to the penalized Lorenz regression

As noted earlier, the three selection methods can produce significantly different models. We present several functions to help users make a definitive choice. First, the method `plot()` can be used to track the evolution of the scores for each selection procedure by setting the argument `type` to `"diagnostic"`. Figure 10 illustrates this, with each facet corresponding to a specific value of the tuning parameter, previously defined by the arguments `grid.arg` and `grid.value`. The horizontal axis represents $-\log(\lambda)$, and therefore aligns with model complexity. To ensure comparability, the scores on the vertical axis are normalized so that the maximum score is equal to 1 for each method. Typically, the scores increase initially as relevant covariates are introduced into the model and then decrease when the additional complexity fails to provide sufficient information. However, the scores may reach a plateau,

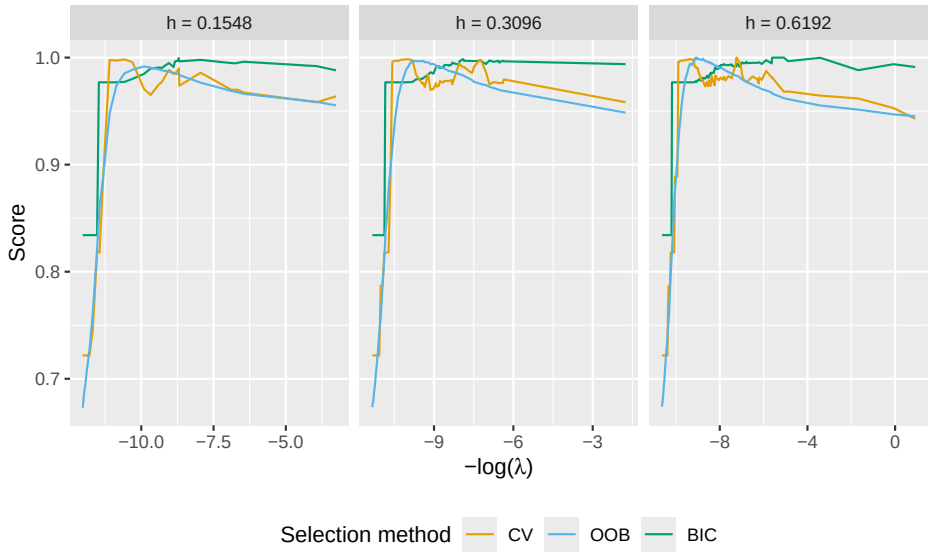


Figure 10: Evolution of the scores for the penalized Lorenz regression

which is observed with the BIC score in this context. This plateau explains the contrasting result between BIC and the other two methods.

```
R> plot(PLR_boot_CV, type = "diagnostic", palette = cbf_palette)
```

In light of the score evolution for BIC, one might prefer to trade a small loss in the score for a simpler model. The function `diagnostic.PLR()` identifies the "best" pair of grid and penalty parameters that is "close enough" to the optimum. The closeness is specified by the argument `tol`. In this case, we consider all pairs for which the normalized BIC score is greater than 0.99. The best pair among this set is determined as the one that results in the sparsest model.[2]

```
R> diagnostic.PLR(PLR_boot_CV, tol = 0.99, method = "BIC")$best
```

```
   grid.idx lambda.idx minloglambda nnzeroes score.BIC score.OOB score.CV
68        2         36        -8.93        5     0.991     0.993    0.976
```

Alternatively, the user might decide to select the pair that offers the best trade-off among the available methods, ensuring that all normalized scores exceed a specified value. This can be achieved by setting `method = "intersect"`.

```
R> diagnostic.PLR(PLR_boot_CV, tol = 0.95, method = "intersect")$best
```

```
   grid.idx lambda.idx minloglambda nnzeroes score.BIC score.OOB score.CV
15        1         15        -10.3        2      0.98     0.989    0.996
```

[2]If multiple pairs yield the same level of sparsity, we select the pair that maximizes the minimum score across all available selection methods.

As we already observed, most methods applicable to the penalized Lorenz regression include an argument `pars.idx`, allowing the user to specify the pair of grid and penalty parameters. This argument can take the values `"BIC"`, `"Boot"` or `"CV"` to specify a selection method. Otherwise, it can be a numeric vector of length two, indicating the indices of the chosen tuning and penalty parameters. As an illustration, we construct a confidence interval for the explained Gini coefficient using the pair of parameters returned in the previous code excerpt.

```
R> confint(PLR_boot_CV, pars.idx = c(3, 61))

 2.5 % 97.5 %
0.0968 0.1729
```

6. Summary and discussion

In this paper, we demonstrate the use of the **LorenzRegression** package, which implements the non-penalized and penalized Lorenz regressions. These methods are appropriate whenever one is interested in explaining the inequality in a response variable with a set of covariates. In particular, they provide inference for the explained Gini coefficient, a measure of explained inequality under a single-index model. The penalized procedure allows for an automatic selection of the relevant covariates.

The illustration in Section 5 demonstrates that the fitting of the model is similar to standard regression techniques. It is performed with the function `Lorenz.Reg()`, which benefits from several S3 methods. Estimated coefficients of the single-index model are obtained with `coef()`, while `ineqExplained()` provides measures of explained inequality. The methods `fitted()` and `predict()` are used to predict either the index or the expected outcome, while `residuals()` is used to extract the model residuals. Summaries of the fitted model are obtained numerically with `summary()`, or graphically with `autoplot()` and `plot()`. Bootstrap inference, enabling the construction of confidence intervals, is provided by the function `Lorenz.boot()` and the method `confint()`. Diagnostic tools are available for the penalized regression. Additionally, the package enables the computation and graphical representation of Lorenz and concentration curves, as well as the computation of Gini coefficients or concentration indices. All functions allow the user to specify sample weights.

Computational details

Computations in this paper were performed on macOS Sonoma 14.7 using vecLib BLAS and LAPACK version 3.12.0. The analysis was carried out in R 4.5.0 with the **LorenzRegression** 2.3.2 package, both of which can be obtained from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

References

Alfons A, Templ M (2013). "Estimation of Social Exclusion Indicators from Complex Surveys: The R Package **laeken**." *Journal of Statistical Software*, **54**(15), 1–25. doi:10.18637/jss.v054.i15.

Bourguignon F, Ferreira FHG, Menéndez M (2007). "Inequality of Opportunity in Brazil." *Review of Income and Wealth*, **53**(4), 585–618. doi:10.1111/j.1475-4991.2007.00247.x.

Canty A, Ripley BD (2024). ***boot**: Bootstrap R (S-PLUS) Functions*. doi:10.32614/CRAN.package.boot. R package version 1.3-31.

Chávez Juárez F, Soloaga I (2014). "**iop**: Estimating Ex-Ante Inequality of Opportunity." *The Stata Journal*, **14**(4), 830–846. doi:10.1177/1536867x1401400408.

Chernozhukov V, Fernández-Val I, Galichon A (2009). "Improving Point and Interval Estimators of Monotone Functions by Rearrangement." *Biometrika*, **96**(3), 559–575. doi:10.1093/biomet/asp030.

Devleesschauwer B, Willimès S, Van Malderen C, Konings P, Speybroeck N, Meili K (2025). ***rineq**: Concentration Index and Decomposition for Health Inequalities*. doi:10.32614/CRAN.package.rineq. R package version 0.3.0.

Eddelbuettel D, Balamuta JJ (2018). "Extending R with C++: A Brief Introduction to **Rcpp**." *The American Statistician*, **72**(1), 28–36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. doi:10.1198/016214501753382273.

Frick H, Chow F, Kuhn M, Mahoney M, Silge J, Wickham H (2025). ***rsample**: General Resampling Infrastructure*. doi:10.32614/CRAN.package.rsample. R package version 1.3.0.

Graybill W, Chen M, Chernozhukov V, Fernandez-Val I, Galichon A (2016). ***Rearrangement**: Monotonize Point and Interval Functional Estimates by Rearrangement*. doi:10.32614/CRAN.package.rearrangement. R package version 2.1.

Heuchenne C, Jacquemain A (2022). "Inference for Monotone Single-Index Conditional Means: A Lorenz Regression Approach." *Computational Statistics & Data Analysis*, **167**(C). doi:10.1016/j.csda.2021.107347.

Jacquemain A, Heuchenne C, Pircalabelu E (2024). "A Penalised Bootstrap Estimation Procedure for the Explained Gini Coefficient." *Electronic Journal of Statistics*, **18**(1), 247 – 300. doi:10.1214/23-ejs2200.

Jann B (2016). "Estimating Lorenz and Concentration Curves." *The Stata Journal*, **16**(4), 837–866. doi:10.1177/1536867x1601600403.

Jenkins S, Van Kerm P (2004). "**GLCURVE**: Stata Module to Derive Generalised Lorenz Curve Ordinates." Statistical Software Components, Boston College Department of Economics. URL https://ideas.repec.org/c/boc/bocode/s366302.html.

Jenkins SP (1999). "**INEQDEC0**: Stata Module to Calculate Inequality Indices with Decomposition by Subgroup." Statistical Software Components, Boston College Department of Economics. URL https://ideas.repec.org/c/boc/bocode/s366007.html.

Kane MJ, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(14), 1–19. doi:10.18637/jss.v055.i14.

Lin H, Peng H (2013). "Smoothed Rank Correlation of the Linear Transformation Regression Model." *Computational Statistics & Data Analysis*, **57**, 615–630. doi:10.1016/j.csda.2012.07.012.

Mélard N, Jacquemain A, Perelman J, SILNE-R Consortium, Lorant V (2025). "Are Policies Implemented Where Most Needed? Exploring Equity in School-Based Smoking Prevention in 2013 and 2016 in Six European Cities." *Journal of School Health*. doi:10.1111/josh.70031.

Microsoft, Weston S (2022). ***foreach***: *Provides Foreach Looping Construct*. doi:10.32614/CRAN.package.foreach. R package version 1.5.2.

O'Donnell O, Van Doorslaer E, Wagstaff A, Lindelow M (2008). *Analyzing Health Equity Using Household Survey Data: A Guide to Techniques and Their Implementation*. The World Bank, Washington, DC. doi:10.1596/978-0-8213-6933-3.

R Core Team (2026). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. doi:10.32614/R.manuals. URL https://www.R-project.org/.

Scrucca L (2013). "**GA**: A Package for Genetic Algorithms in R." *Journal of Statistical Software*, **53**(4), 1–37. doi:10.18637/jss.v053.i04.

Shi X, Huang Y, Huang J, Ma S (2018). "A Forward and Backward Stagewise Algorithm for Nonconvex Loss Functions with Adaptive Lasso." *Computational Statistics & Data Analysis*, **124**, 235–251. doi:10.1016/j.csda.2018.03.006.

Speybroeck N, Konings P, Lynch J, Harper S, Berkvens D, Lorant V, Geckova A, Hosseinpoor AR (2010). "Decomposing Socioeconomic Health Inequalities." *International Journal of Public Health*, **55**(4), 347–351. doi:10.1007/s00038-009-0105-z.

StataCorp (2025). *Stata Statistical Software: Release 19*. StataCorp LLC, College Station. Https://www.stata.com.

Wickham H (2016). ***ggplot2***: *Elegant Graphics for Data Analysis*. 2nd edition. Springer-Verlag. doi:10.1007/978-3-319-24277-4.

Wójcik S, Giemza A, Machowska K, Napora J (2024). ***wINEQ***: *Inequality Measures for Weighted Data*. doi:10.32614/CRAN.package.wINEQ. R package version 1.2.1.

Zeileis A (2014). ***ineq***: *Measuring Inequality, Concentration, and Poverty*. doi:10.32614/CRAN.package.ineq. R package version 0.2-13.

Affiliation:

Alexandre Jacquemain, Cédric Heuchenne
CAPE
UCLouvain Saint-Louis - Bruxelles
1000 Brussels, Belgium
and
ISBA
UCLouvain
1357 Louvain-la-Neuve, Belgium
E-mail: alexandre.jacquemain@uclouvain.be, cedric.heuchenne@uclouvain.be

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
September 2026, Volume 117, Issue 6 *Submitted:* 2024-10-14
doi:10.18637/jss.v117.i06 *Accepted:* 2025-09-01