GIR: Gini Index Regression

Estimation, inference and prediction: detailed reference for `gir()`, `bboot.gir()`, and `predict.gir()`

David Díaz-Villarejo

Package version 0.2.1 — October 2026

Contents

1 Getting started **4**

2 Model and interface **4**

3 Usage **5**

4 Mathematical estimator **6**

5 Arguments **6**

6 First-stage dimension allocation in detail **15**

6.1 One continuous variable 15

6.2 Two or more continuous variables 15

7 Discrete cells and basis-size reference **15**

8 Diagnostics **16**

9 Pivoted QR and aliasing **17**

10 Value **18**

11 Methods **18**

12 Warnings and errors **19**

13 Examples **20**

13.1 Basic fit with one continuous and two binary first-stage variables 20

13.2 A numeric discrete variable with four values 20

13.3 A Legendre specification with known support bounds 21

13.4 A cubic spline specification with knots per margin 21

13.5 Target total dimension with two continuous regressors . . . 21
13.6 Directly specify marginal spline knots . . . 22
13.7 Compare cell and full reference sizes . . . 22
13.8 Supply the first-stage estimates . . . 23
13.9 Weights and missing values . . . 23
13.10 Inspect clipping . . . 23
13.11 First-stage singularity policies . . . 24
13.12 Strict second-stage singularity behavior . . . 24
13.13 GIR goodness of fit . . . 25

14 `bboot.gir`: Bayesian bootstrap inference 25
14.1 Name and description . . . 25
14.2 Usage . . . 25
14.3 Bootstrapping a fitted model . . . 26
14.4 Arguments: model, replicates and confidence level . . . 26
14.5 Arguments: transformations and hypotheses . . . 27
14.6 Arguments: random weights and reproducibility . . . 28
14.7 Arguments: first-stage refitting . . . 30
14.8 Arguments: output, failure handling and extra arguments . . . 30
14.9 The two-stage bootstrap calculation . . . 31
14.10 Basic confidence intervals: the paper's Algorithm 2 . . . 32
14.11 Standard errors, covariance and bootstrap bias . . . 32
14.12 Smooth transformations and the bootstrap delta argument . . . 33
14.13 Marginal bootstrap hypothesis tests . . . 33
14.14 Joint bootstrap Wald test . . . 34
14.15 Returned object and extraction methods . . . 34

15 `predict.gir`: linear conditional Gini predictions 35
15.1 Name, target and usage . . . 35
15.2 Arguments . . . 36
15.3 Prediction standard errors and intervals . . . 37
15.4 Pointwise coverage, range and extrapolation . . . 38

16 Complete worked examples 38
16.1 Generate data from the paper's continuous-binary design . . . 38
16.2 Fit and bootstrap the spline specification . . . 38
16.3 The formula convenience form . . . 39
16.4 Contrasts, Gini points and a coefficient ratio . . . 39

16.5 Joint and directional tests . . . 40
16.6 Newdata predictions with a reusable bootstrap . . . 40
16.7 Automatic bootstrap inside prediction . . . 40
16.8 A difference in predicted Gini indices . . . 41
16.9 Supplied multipliers and exact reuse across bases . . . 41
16.10 The oracle first stage . . . 41
16.11 A complete external first-stage callback example . . . 42
16.12 Missing prediction values . . . 42

17 Relation to existing R interfaces **42**

18 Using the package in a Monte Carlo study **43**
18.1 Three different quantities called standard deviation or bias . . . 43
18.2 Warp-speed coverage . . . 43

19 Mathematical validation **44**

20 Installation and development **45**

21 Current limitations **45**

22 References **46**

23 See also **46**

1 Getting started

GIR estimates how the conditional Gini index varies with covariates. Use `gir()` to fit the model, `bboot.gir()` for bootstrap inference, and `predict()` to evaluate the fitted Gini index at chosen covariate values. The method is described in the working paper by Bhardwaj and Díaz-Villarejo [1].

This example fits a separate Gini index for each of two groups. The data include ties, which the conditional mid-distribution handles directly. The code and its output are regenerated when the package is built.

```
> library(GIR)
> d <- data.frame(y = rep(c(1, 2, 2, 4, 5), 8),
+                 group = factor(rep(c("a", "b"), each = 20)))
> fit <- gir(y ~ 0 + group, data = d, discrete = "group")
> coef(fit)
```

```
   groupa    groupb
0.2857143 0.2857143
```

With group indicators in both stages, each coefficient is the empirical Gini index for that group, without a finite-sample correction. Bootstrap inference refits both stages using the same weights in each draw.

```
> bb <- bboot.gir(fit, R = 499L, seed = 42L)
> confint(bb)
```

```
            2.5%     97.5%
groupa 0.2421347 0.3571932
groupb 0.2390379 0.3544644
```

```
> predict(fit, data.frame(group = c("a", "b")),
+         interval = "confidence", bootstrap = bb)
```

```
        fit       lwr       upr
1 0.2857143 0.2421347 0.3571932
2 0.2857143 0.2390379 0.3544644
```

The prediction intervals concern the conditional Gini index at each profile. They are not intervals for individual outcomes or simultaneous confidence bands. For more examples, including continuous covariates, see Section 16. The remaining sections describe the arguments, calculations and diagnostics in detail.

2 Model and interface

`gir()` estimates the point coefficients of the linear Gini Index Regression (GIR) model. Its user interface follows the formula/data conventions of familiar R regression functions: the second-stage model is specified with a two-sided formula, an intercept is included by default, and the fitted object supports standard methods such as `summary()`, `coef()`, `fitted()`, `residuals()`, `model.frame()`, and `model.matrix()`.

The package also provides `bboot.gir()` for two-stage Bayesian-bootstrap inference and `predict.gir()` for linear conditional Gini predictions and pointwise uncertainty. Their detailed references begin in Sections 14 and 15. The GIR-specific goodness-of-fit statistic is an optional output of `gir()`.

The first stage estimates the conditional mid-distribution function

$$M(y \mid x) = P(Y < y \mid X = x) + \frac{1}{2}P(Y = y \mid X = x) = E[\psi_y(Y) \mid X = x],$$

where

$$\psi_y(t) = \mathbf{1}\{t < y\} + \frac{1}{2}\mathbf{1}\{t = y\}.$$

The mid-distribution definition applies to continuous, discrete, and mixed outcomes. The paper's inference theory covers continuous and finite-discrete response regimes, not mixed response laws. For a continuous response, ties occur with probability zero and $M(y \mid x)$ coincides with the ordinary conditional CDF.

3 Usage

```
gir(
  formula,
  data,
  subset,
  weights,
  na.action = stats::na.fail,

  first_stage = NULL,
  continuous = NULL,
  discrete = NULL,
  midcdf = NULL,

  basis = c("legendre", "spline"),
  kappa = NULL,
  basis_constant = 1,
  sieve_size = NULL,
  sieve_size_type = c("auto", "dimension", "knots"),
  sieve_size_reference = c("cell", "full"),
  spline_order = 4L,
  basis_bounds = NULL,

  clip = FALSE,

  first_stage_singular = c("error", "ridge", "drop", "none"),
  first_stage_tol = 1e-12,
  ridge = 1e-10,
  minimum_cell_size = 8L,
  max_basis_columns = 1000L,

  singular.ok = TRUE,
  second_stage_tol = 1e-7,

  GIR = FALSE,

  model = TRUE,
  x = FALSE,
  y = FALSE,
```

```
  contrasts = NULL
)

midcdf(object, ...)
```

4 Mathematical estimator

Let X_i denote the second-stage regressor vector created by the model formula, including a constant unless the formula removes it. Let $w_i \geq 0$ be the optional observation weight, with $w_i = 1$ by default. Once the first stage has produced $\widehat{M}_i = \widehat{M}(Y_i \mid X_i^{(1)})$, where $X_i^{(1)}$ denotes the chosen first-stage conditioning variables, `gir()` computes

$$\widehat{A} = \frac{1}{n}\sum_{i=1}^{n} w_i Y_i X_i X_i', \qquad \widehat{b} = \frac{1}{n}\sum_{i=1}^{n} w_i X_i Y_i \{2\widehat{M}_i - 1\},$$

and solves

$$\widehat{\theta} = \widehat{A}^{-1}\widehat{b}$$

when $\widehat{A}$ is nonsingular. Under rank deficiency and `singular.ok = TRUE`, an estimable subset is obtained by pivoted QR and aliased coefficients are reported as `NA`.

For a discrete cell g, define $\mathcal{I}_g = \{i : D_i = g\}$, and let $p_{K_g}(Z_i)$ be the basis vector for the continuous first-stage variables. The cell-specific first stage used for observation $i \in \mathcal{I}_g$ is

$$\widehat{M}_i = p_{K_g}(Z_i)' \left\{ \sum_{j \in \mathcal{I}_g} w_j p_{K_g}(Z_j) p_{K_g}(Z_j)' \right\}^{-1} \left\{ \sum_{j \in \mathcal{I}_g} w_j p_{K_g}(Z_j) \left[\mathbf{1}\{Y_j < Y_i\} + \frac{1}{2}\mathbf{1}\{Y_j = Y_i\} \right] \right\}.$$

The implementation sorts Y once within each cell and obtains all threshold-specific right-hand sides through cumulative sums. It therefore does not run one separate least-squares regression for every observed threshold.

5 Arguments

`formula`

A two-sided R model formula for the linear GIR second stage. The left-hand side must be one numeric response. The right-hand side is processed by `model.matrix()`, so ordinary formula syntax, factors, interactions, and transformations are available. A constant is included by default. Use `0 +` or `-1` to remove it.

For example,

```
y ~ treatment + age + factor(region)
```

contains an intercept, while

```
y ~ 0 + x1 + x2 + x3
```

does not. The latter convention corresponds to the paper's simulation designs, whose second-stage regressors carry no constant.

Offsets are not implemented and generate an error.

`data`

A data frame, or an object coercible to a data frame, containing all variables used by the second stage, first stage, weights, subset expression, and any supplied mid-CDF column.

`subset`

An optional logical, numeric, or character row index evaluated in `data`. The subset is applied before missing-value processing. Vectors supplied through `weights` or `midcdf` may have length one, `nrow(data)`, or the number of selected rows.

`weights`

Optional nonnegative case or sampling weights. They enter both stages:

- in the cell-specific first-stage Gram matrices and threshold scores; and
- in the second-stage moments $\widehat{A}$ and $\widehat{b}$.

At least one weight in the retained estimation sample must be strictly positive, after subsetting and missing-value handling. Negative or infinite weights produce an error; missing weights are handled by `na.action` together with the other variables. These weights are empirical-moment weights; they should not be interpreted automatically as inverse conditional-variance weights.

`na.action`

The missing-data action. The default, `stats::na.fail`, stops if any variable needed by either stage, the weights, or a supplied mid-CDF contains a missing value. `na.omit` and `na.exclude` remove incomplete rows and produce a warning reporting how many observations were removed. All objects are then aligned to the same estimation sample. Missing weights are treated like any other missing variable: they stop the fit under `na.fail` and their rows are removed under `na.omit` or `na.exclude`.

An initially empty dataset, an empty subset, and an empty sample after missing-value removal each have explicit errors. If all selected observations are removed, the error names the action and selected sample size. If rows remain but all their weights are zero, a separate retained-weight error is issued. These checks precede matrix construction and the routine removal warning, including when R warnings are promoted to errors with `options(warn=2)`. A nonempty sample of zero outcomes is a different problem: the second-stage moments have zero rank.

Custom actions must return a data frame selecting original rows with valid, unique row identifiers. Their removal warning counts incomplete and complete rows separately. An error inside a custom action is reported as a custom-action failure, retaining its original message; it is not automatically attributed to missing values.

`first_stage`

Selects the variables conditioning the nonparametric first-stage estimate. Accepted forms are:

1. `NULL`: use the raw variables appearing on the right-hand side of `formula`;

2. a one-sided formula, such as `~ x1 + x2 + region`; or
3. a character vector of data-column names.

The selection is based on underlying data variables. For example, `first_stage = ~ x + I(x^2)` identifies `x` as the raw continuous variable; the nonlinear series transformations are generated internally.

The first-stage conditioning set may differ from the second-stage regressors, but such a choice changes the conditional mid-distribution being estimated and should be justified econometrically.

`continuous`

A character vector naming the continuous first-stage variables. Every named column must be numeric and must also be included in `first_stage`. Continuous variables are rescaled to $[0, 1]$ using `basis_bounds` before the basis is evaluated.

Variables not classified explicitly are classified automatically. Numeric variables are classified as continuous by default. Therefore, a numeric dummy or a numeric variable taking values $\{1, 2, 3, 4\}$ must be listed explicitly in `discrete` when it is intended to define cells. A message names the numeric columns that were classified by default. An undeclared numeric column with at most two distinct values in the retained estimation sample is an error, because a two-point support cannot carry a continuous basis; declare it in `discrete`, or in `continuous` if that is really intended. This check runs after missing-value handling; values appearing only in discarded observations do not count towards the retained support.

`discrete`

A character vector naming the discrete first-stage variables. Factors, ordered factors, character variables, and logical variables are automatically treated as discrete when they have not been classified explicitly. Numeric discrete variables should be listed explicitly.

The function forms cells from the observed joint combinations of all discrete variables. It estimates the series regression separately within each observed cell. Empty theoretical combinations are not created.

Example: one four-valued discrete regressor. Suppose X is continuous and $D \in \{1, 2, 3, 4\}$. The call

```
fit <- gir(
  y ~ x + factor(d),
  data = dat,
  first_stage = ~ x + d,
  continuous = "x",
  discrete = "d",
  basis = "spline"
)
```

creates the four samples

$$\mathcal{I}_1 = \{i : D_i = 1\}, \quad \mathcal{I}_2 = \{i : D_i = 2\}, \quad \mathcal{I}_3 = \{i : D_i = 3\}, \quad \mathcal{I}_4 = \{i : D_i = 4\},$$

and separately estimates $M(y \mid x, D = 1), \ldots, M(y \mid x, D = 4)$. The fitted values are then returned to their original observation order before the common second stage is run.

With a common basis in every cell, this is algebraically equivalent to one saturated regression using the block design

$$q(X, D) = \begin{pmatrix} \mathbf{1}\{D = 1\} p_K(X) \\ \mathbf{1}\{D = 2\} p_K(X) \\ \mathbf{1}\{D = 3\} p_K(X) \\ \mathbf{1}\{D = 4\} p_K(X) \end{pmatrix},$$

because the Gram matrix is block diagonal. The separate-cell implementation is more memory efficient and permits cell-specific diagnostics and, when requested, cell-specific dimensions.

`midcdf`

An optional numeric vector of conditional mid-CDF estimates, or a character string naming a numeric column in `data`. Supplying it bypasses the entire first-stage series estimation. It is useful for externally estimated nuisance functions, cross-fitting, machine-learning first stages, or oracle exercises.

The vector is checked for finite values and for values outside $[0, 1]$. The `clip` rule is then applied in exactly the same way as for internally estimated values.

`basis`

Two bases are available:

`"legendre"`	An orthonormal Legendre-polynomial series on the normalized interval $[0, 1]$. For each continuous margin, the function uses the standard recurrence and normalizes degree r by $\sqrt{2r + 1}$.
`"spline"`	A polynomial spline sieve represented internally by the numerically stable basis returned by `splines::bs()`. Interior knots are equally spaced on the normalized interval, boundary knots are 0 and 1, and `intercept = TRUE`. Select this basis with `basis = "spline"`.

With two or more continuous variables, the marginal bases are combined by a full row-wise tensor product.

`kappa`

The exponent in the automatic sieve rule

$$T = \lfloor c\, n_{\text{ref}}^{\kappa} \rfloor,$$

where c is `basis_constant`. The defaults are

$$\kappa = 0.245 \quad \text{for Legendre,} \qquad \kappa = 0.300 \quad \text{for splines.}$$

The interpretation of T depends on `sieve_size_type`.

`basis_constant`

The strictly positive multiplier c in the automatic rule. The default is one. For example, `basis_constant = 1.5` replaces $\lfloor n_{\text{ref}}^{\kappa} \rfloor$ by $\lfloor 1.5 n_{\text{ref}}^{\kappa} \rfloor$.

`sieve_size`

An optional fixed sieve size that replaces the automatic power rule.

- Under `sieve_size_type = "dimension"`, it must be one positive scalar and is interpreted as a target total tensor-product dimension.
- Under `sieve_size_type = "knots"`, it may be one nonnegative scalar, repeated over all continuous margins, or a nonnegative vector of length `length(continuous)` giving the marginal interior-knot counts directly.

For Legendre bases, `"knots"` is unavailable.

`sieve_size_type`

Controls how the raw size T is interpreted.

`"auto"`

The default is `"dimension"` for both bases: T is the target total tensor-product dimension, which is the K of the paper, so that $T = \lfloor n_{\text{ref}}^{\kappa} \rfloor$ grows at the rate assumed by the theory. With cubic splines and one continuous regressor, T columns mean $J = T - 4$ interior knots; for example $T = 6$ gives two interior knots. An alternative convention applies $\lfloor n^{0.3} \rfloor$ to the number of interior knots instead, giving ten columns in that example; it is available through `"knots"`.

For $d \geq 2$ continuous regressors, applying a knot rule in every margin would give total tensor dimension

$$K = \prod_{\ell=1}^{d} (J + s_0) = (J + s_0)^d,$$

growing roughly as $n^{0.3d}$, far faster than the rate conditions allow. The default therefore controls the total dimension and allocates it across margins as described next.

`"dimension"`

The raw size T is a target for the complete tensor-product dimension

$$K = \prod_{\ell=1}^{d} m_\ell.$$

For splines of order s_0,

$$m_\ell = s_0 + J_\ell,$$

where J_ℓ is the number of interior knots in margin ℓ.

The function starts at the minimum admissible marginal dimensions and adds one column at a time. Among feasible increments, it favors a currently smaller margin, so the allocation is approximately balanced. It stops when no single-column increment can be made without exceeding the effective target. The dimensions need not be exactly equal, and the achieved product need not be exactly T.

For cubic splines, $s_0 = 4$. Examples are:

d	target T	marginal dimensions (m_ℓ)	knots (J_ℓ)	actual K
2	100	$(10, 10)$	$(6, 6)$	100
2	35	$(7, 5)$	$(3, 1)$	35
2	50	$(7, 7)$	$(3, 3)$	49
3	100	$(5, 5, 4)$	$(1, 1, 0)$	100

When several increments are tied, the order of the variables in `continuous` breaks the tie. For example, with `continuous = c("x1", "x2", "x3")` and target 100, the allocation $(5, 5, 4)$ gives the first two variables one interior knot and the third none.

For a spline of order s_0, the minimum possible full-tensor dimension is

$$K_{\min} = s_0^d.$$

With cubic splines, this equals 4^d. Hence a target below 4^d cannot be attained; the minimum basis is used provided the cell is large enough. For example, with two continuous regressors and target 9, the minimum is $4 \times 4 = 16$.

For Legendre polynomials the minimum is $K_{\min} = 2^d$: every continuous regressor enters at least linearly, so no variable is silently dropped from the first stage. With three continuous regressors and target 6, the allocation is $(2, 2, 2)$, eight columns. Whenever the minimum exceeds the target, the diagnostics record `floored = TRUE`. Because $\lfloor n^{0.3} \rfloor$ reaches 16 only near $n = 10{,}000$ and 64 only near $n = 1{,}000{,}000$, the default cubic spline basis with two or more continuous regressors is the knot-free tensor product at most sample sizes; a lower `spline_order` or a larger `basis_constant` gives more flexibility at moderate n.

`"knots"`

This mode applies only to splines and directly controls marginal interior-knot counts. A scalar is repeated in every margin. A vector permits an intentionally unbalanced allocation. With the automatic rule it applies $\lfloor n_{\text{ref}}^{\kappa} \rfloor$ to the knot count of every margin rather than to the total dimension.

For example,

```
sieve_size_type = "knots",
sieve_size = c(3L, 1L),
spline_order = 4L
```

requests

$$(J_1, J_2) = (3, 1), \qquad (m_1, m_2) = (7, 5), \qquad K = 35.$$

`sieve_size_reference`

Determines which sample size is inserted into the automatic rule. It does not create a holdout sample and is unrelated to cross-fitting.

`"cell"` Use the number n_g of observations in the current joint discrete cell. This is the default. Unequal cell sizes can therefore produce unequal requested dimensions.

`"full"` Use the full estimation-sample size n in every cell. Every cell receives the same nominal requested size, although sparse-cell feasibility caps can still reduce the actual basis.

For example, suppose $n = 1000$, the four cell sizes are $(100, 200, 300, 400)$, $\kappa = 0.3$, and cubic splines under the default total-dimension rule. With `"cell"`, the requested total dimensions are

$$(\lfloor 100^{0.3} \rfloor, \lfloor 200^{0.3} \rfloor, \lfloor 300^{0.3} \rfloor, \lfloor 400^{0.3} \rfloor) = (3, 4, 5, 6),$$

so the first two cells use the four-column minimum (no interior knots) and the others five and six columns (one and two interior knots). With `"full"`, every cell requests

$$\lfloor 1000^{0.3} \rfloor = 7$$

columns, three interior knots.

`spline_order`

The spline order s_0, required to be an integer of at least two. Polynomial degree equals $s_0 - 1$. The default `spline_order = 4L` therefore gives a cubic spline basis. With J_ℓ interior knots and `intercept = TRUE`, the marginal dimension is

$$m_\ell = J_\ell + s_0.$$

`basis_bounds`

Bounds used to map each continuous first-stage variable to $[0, 1]$. If `NULL`, the function uses the minimum and maximum in the final estimation sample. A named list is recommended:

```
basis_bounds = list(
  x1 = c(0.1, 1),
  age = c(18, 65)
)
```

All observed values must lie inside the supplied bounds. Scaling is performed using the full estimation sample before it is divided into discrete cells, so every cell uses the same coordinate transformation.

Known support bounds are preferable whenever the support is known, for example in a simulation study. The bootstrap retains those bounds and the corresponding basis map. Linear Gini prediction uses the second-stage design, not a new first-stage CDF evaluation.

`clip`

A logical value controlling whether first-stage fitted values outside $[0, 1]$ are truncated.

- `FALSE` (default): retain the linear-series fitted values and warn;
- `TRUE`: warn and replace values below zero by zero and values above one by one.

Clipping changes $2\widehat{M}_i - 1$ and therefore changes the GIR point estimate. It is not merely a display operation.

`first_stage_singular`

Specifies how a singular or ill-conditioned first-stage cell is handled. The function diagnoses each cell using the reciprocal condition number of the Gram matrix and the QR rank of the weighted basis matrix.

`"error"` Stop and identify the cells that need attention. This default preserves the unregularized first-stage specification.

`"ridge"` Replace the cell Gram matrix by

$$G_g + \lambda_g I, \qquad \lambda_g = \texttt{ridge} \times \operatorname{mean}(\operatorname{diag} G_g).$$

There is no absolute floor on the matrix scale. The applied value is stored in the diagnostics. If the solve fails, a further $100\lambda_g$ increment is attempted. The total is $101\lambda_g$ if the initial penalty was already applied, or $100\lambda_g$ if only the retry was needed. A successful retry is reported as `action = "ridge"` in either case.

Multiplying all weights by $c > 0$ multiplies G_g, its right-hand side, and the penalty by c, preserving the fitted values up to floating-point error within usable numerical ranges. Nonpositive or nonfinite matrix scales, penalties that underflow or overflow, and nonfinite adjusted matrices are rejected with informative errors.

`"drop"` Use pivoted QR to identify a numerically independent subset of basis columns, remove the remaining columns, and report their names. The resulting effective sieve need not remain a complete tensor product.

`"none"` Apply no proactive correction and pass the unmodified Gram matrix to `solve()`. A numerical failure remains an error.

`first_stage_tol`

The tolerance used in first-stage reciprocal-condition and QR-rank diagnostics. The default is 10^{-12}. A cell is marked problematic when its reciprocal condition number is nonfinite or below the tolerance, or when its numerical rank is below the number of basis columns.

`ridge`

The relative ridge constant. It must be strictly positive when `first_stage_singular = "ridge"`; zero is allowed when ridge is not used. The default is 10^{-10}, a small numerical stabilization heuristic, not a universal regression default, a cross-validated choice, or a value prescribed by the paper's theory. A small penalty can still materially affect directions that the data identify poorly. Inspect the diagnostics and assess sensitivity to the ridge constant when using this option. The overall default remains `first_stage_singular = "error"`, which does not apply ridge.

Earlier versions imposed a floor of 1 on the average Gram diagonal. Removing that floor can change both point estimates and bootstrap inference when ridge is active and the average diagonal is below 1. Affected analyses should refit and rerun their bootstrap. Ridge regularization remains a modification of the first-stage estimator requiring its own statistical justification.

`minimum_cell_size`

The minimum number of positive-weight observations required in every observed joint discrete cell. The default is eight. A failure names the cell and reports both its total row count and positive-weight row count.

`max_basis_columns`

A safety cap for the number of columns in a full tensor-product basis. The actual cell-specific upper bound is

$$\min\{\texttt{max_basis_columns},\ n_{+,g} - 2\},$$

where $n_{+,g}$ is the number of positive-weight observations in cell g. The default package cap is 1000 columns.

`singular.ok`

Controls second-stage rank deficiency in the style of `lm()`.

- `FALSE`: a singular second-stage moment matrix is an error;
- `TRUE`: pivoted QR retains an estimable subset, aliased coefficients are returned as `NA`, and a warning is issued.

The default is `TRUE`. A full-rank but ill-conditioned matrix is not regularized; the function returns the unregularized estimate and warns.

`second_stage_tol`

The QR-rank and reciprocal-condition tolerance for the second stage. The default is 10^{-7}.

`GIR`

A logical value. When `TRUE`, compute the GIR-specific goodness-of-fit measure based on the estimated conditional centered mid-rank. Let $z_i = 2\widehat{M}_i - 1$, let $\widehat{z}_i = X_i'\widehat{\theta}$, and let

$$\widehat{a}_0 = \frac{\sum_i w_i Y_i z_i}{\sum_i w_i Y_i}.$$

Then

$$R^2_G = 1 - \frac{\sum_i w_i Y_i (z_i - \widehat{z}_i)^2}{\sum_i w_i Y_i (z_i - \widehat{a}_0)^2}.$$

When the unrestricted formula contains an intercept and the weighted GIR loss is nonnegative, the constant-only model is nested. Without an intercept, the function warns that the usual $[0, 1]$ interpretation need not hold.

`model`, `x`, and `y`

Logical retention controls:

- `model = TRUE`: retain the second-stage model frame;
- `x = TRUE`: retain the second-stage design matrix;
- `y = TRUE`: retain the response vector.

The flags govern these public slots. To support reproducible bootstrap refits, the fitted object also retains an `estimation` snapshot containing the distinct design rows with the index that maps observations to them, the response, weights, first-stage data and controls, irrespective of these three flags. Setting them to `FALSE` does not remove all underlying data from the object or make it anonymous. Observations with the same regressor values share a design row, so with cell-level regressors (industry and year effects in firm data, for example) the snapshot holds one row per cell rather than one per observation; `x = TRUE` expands it to the full design.

`contrasts`

An optional contrast specification passed to `model.matrix()` for the second-stage formula.

6 First-stage dimension allocation in detail

6.1 One continuous variable

When $d = 1$, there is no nontrivial tensor-product multiplication. The marginal dimension equals the total dimension.

Under the default total-dimension rule with $n_{\text{ref}} = 500$,

$$K = \lfloor 500^{0.3} \rfloor = 6, \qquad J_1 = K - 4 = 2,$$

a cubic spline with two interior knots and six columns. The alternative `"knots"` convention applies the same number to the knot count instead,

$$J_1 = 6, \qquad m_1 = J_1 + 4 = 10, \qquad K = m_1 = 10.$$

6.2 Two or more continuous variables

For $d \geq 2$, the total basis is the row-wise tensor product. If the marginal dimensions are $(m_1, \ldots, m_d)$, then

$$K = \prod_{\ell=1}^{d} m_\ell.$$

Under `"dimension"`, the allocation algorithm chooses the $(m_1, \ldots, m_d)$ vector. Under `"knots"`, the user or automatic rule chooses $(J_1, \ldots, J_d)$, after which $m_\ell = J_\ell + s_0$.

The balanced allocation is a deterministic implementation convention; it does not infer variable importance. When a tie is resolved in favor of an earlier variable, changing the order of `continuous` can change which margin gets the final additional column.

7 Discrete cells and basis-size reference

With both continuous and discrete first-stage variables, the default `sieve_size_reference = "cell"` generally does *not* force the same basis dimension in every cell. If cell sizes differ, the raw rules

$$\left\lfloor c n_g^{\kappa} \right\rfloor$$

can differ. Feasibility capping or singularity handling can create further differences.

To request a common nominal automatic size, use `sieve_size_reference = "full"`. To request a common fixed size, provide `sieve_size`. Even then, the actual basis can be reduced in a sparse

cell, and `first_stage_singular = "drop"` can lower the effective dimension. Always inspect the diagnostics rather than inferring the realized dimensions from the call alone.

8 Diagnostics

For an internally estimated first stage, use

```
d <- fit$first_stage$diagnostics
```

The data frame has one row per observed joint discrete cell. It contains the following fields.

Field	Meaning
`cell`	Human-readable joint discrete-cell label.
`observations`	Number of rows in the cell.
`positive_weight_observations`	Number of rows with strictly positive observation weight.
`requested_size`	Raw requested scalar or knot vector before cell feasibility adjustments.
`size_type`	Realized interpretation: `"dimension"`, `"knots"`, or `"constant"`.
`marginal_dimensions`	A list-column. Each row contains a named integer vector $(m_1, \ldots, m_d)$, with names equal to `continuous`. These are the marginal basis dimensions actually constructed after size allocation and capping, but before optional QR column dropping.
`marginal_knot_counts`	A list-column. For `basis = "spline"`, each row contains the named integer vector $(J_1, \ldots, J_d)$. For Legendre bases, it contains a named vector of `NA_integer_` values because knot counts are not applicable.
`basis_dimension`	Total number of constructed tensor-product columns. With continuous variables, this equals $\prod_\ell m_\ell$; with none, it equals one.
`effective_dimension`	Number of columns used after singularity handling. It differs from `basis_dimension` when `first_stage_singular = "drop"`.
`rank`	Numerical rank reported for the cell fit.
`rcond`	Reciprocal condition number of the original cell Gram matrix. Smaller values indicate worse conditioning.
`singular`	Whether the original cell design was diagnosed as singular or ill-conditioned.
`action`	`"none"`, `"ridge"`, or `"drop"`.
`ridge_lambda`	Actual ridge amount added to the Gram diagonal.
`dropped_columns`	Semicolon-separated names of columns removed by pivoted QR.
`capped`	Whether the requested size was reduced by a cell-specific feasibility limit.
`floored`	Whether the minimum tensor dimension (2^d for Legendre, s_0^d for splines) exceeded the requested size, so the minimum was used.

The list-columns preserve named integer vectors rather than flattening them to text. Typical extraction is:

```
# Complete vector in the first cell
d$marginal_dimensions[[1L]]
d$marginal_knot_counts[[1L]]

# Stack all cells into ordinary matrices
marginal_dimensions_by_cell <- do.call(
  rbind,
  d$marginal_dimensions
)

marginal_knots_by_cell <- do.call(
  rbind,
  d$marginal_knot_counts
)
```

For example, a cubic two-dimensional spline cell with

```
d$marginal_dimensions[[1L]]
# x1 x2
#  7  5

d$marginal_knot_counts[[1L]]
# x1 x2
#  3  1
```

has

$$K = 7 \times 5 = 35.$$

If pivoted QR subsequently drops three interaction columns, the diagnostics can show

$$\texttt{basis_dimension} = 35, \qquad \texttt{effective_dimension} = 32.$$

There need not exist a meaningful effective marginal vector whose product is 32, because QR may remove individual tensor interactions rather than an entire marginal function. The reported marginal vectors therefore describe the constructed tensor basis before column dropping.

9 Pivoted QR and aliasing

For a matrix P, an ordinary QR decomposition writes

$$P = QR,$$

where Q has orthonormal columns and R is upper triangular. A column-pivoted QR decomposition writes

$$P\Pi = QR,$$

where Π permutes the columns. The algorithm attempts to move informative, linearly independent columns first and redundant columns last.

For example, the design

$$P = \begin{bmatrix} \mathbf{1} & x & 2x \end{bmatrix}$$

has three columns but rank two because its third column is twice its second. Pivoted QR retains two independent columns and identifies one aliased column.

In the first stage, `"drop"` applies pivoted QR to the weighted design $\mathrm{diag}(\sqrt{w})P$. In the second stage, `singular.ok = TRUE` uses the same principle to report estimable coefficients and return `NA` for aliased ones.

Rank deficiency and ill-conditioning are not identical. A matrix may be full rank but nearly singular. The GIR package therefore records both QR rank and reciprocal condition number.

10 Value

An object of class

```
c("gir", "gir.object")
```

with principal components:

Component	Contents
`coefficients`	Named point estimates, with aliased entries set to `NA`.
`coefficients.complete`	A complete coefficient vector used for fitted values, with zero in aliased positions.
`fitted.values`	Fitted linear conditional Gini index $X_i'\widehat{\theta}$; equivalently the fitted index in the generated mid-rank regression.
`residuals`	$(2\widehat{M}_i - 1) - X_i'\widehat{\theta}$.
`rank`, `df.residual`, `aliased`	Second-stage rank information.
`midcdf`	Named vector $(\widehat{M}_1, \ldots, \widehat{M}_n)$.
`centered.midrank`	Named vector $2\widehat{M}_i - 1$.
`GIR.R2`	`NULL` unless `GIR = TRUE`; otherwise a list containing the value, restricted coefficient, unrestricted loss, and restricted loss.
`first_stage`	First-stage specification, bounds, cell factor, clipping information, singularity settings, and diagnostics.
`second_stage`	Singularity indicators, reciprocal condition number, tolerance, moment matrix, and moment vector.
`call`, `formula`, `terms`, `contrasts`, `xlevels`, `assign`	Formula and design metadata.
`weights`	Final estimation-sample weights.
`na.action`	Omission information, when applicable.
`nobs`, `original.nobs`, `subset.nobs`, `row.index`	Sample-size and row-alignment information.
`model`, `x`, `y`	Retained model frame, design, and response according to the retention arguments.
`estimation`	Resolved estimation snapshot for refitting and prediction, retained independently of the public-slot flags: the distinct design rows, the index mapping observations to them, response, weights, first-stage data and controls.

11 Methods

The current object supports:

```
print(fit)
summary(fit)
```

```
coef(fit)
fitted(fit)
residuals(fit)
weights(fit)
nobs(fit)
formula(fit)
terms(fit)
model.frame(fit)
model.matrix(fit)
midcdf(fit)
summary(midcdf(fit))
```

`summary(fit)` remains the point-estimator summary and prints a one-column coefficient table. Use `summary(bboot.gir(fit, ...))` for inference; the point object is not silently refitted by its summary method. `summary(midcdf(fit))` uses the ordinary numeric vector summary and reports the minimum, quartiles, median, mean, and maximum.

12 Warnings and errors

Errors

The function stops for, among other cases:

- missing data under the default `na.fail`;
- empty input, an empty subset, or no observations after missing-value handling;
- nonnumeric or nonfinite response values;
- negative or infinite weights, or no positive weight in the retained sample;
- first-stage columns not found in `data`;
- overlap between `continuous` and `discrete`;
- an undeclared numeric first-stage column with at most two distinct values;
- observed continuous values outside `basis_bounds`;
- a sparse joint discrete cell;
- a cell too small for the minimum tensor basis (2^d Legendre or s_0^d spline columns);
- first-stage singularity under `"error"`, including a marginal basis with more columns than its variable has distinct values in a cell;
- a failed unregularized solve under `"none"`;
- second-stage singularity when `singular.ok = FALSE`.

Warnings

The function warns when:

- one or more response values are strictly negative;

- observations are removed by `na.omit` or `na.exclude`;
- a custom action removes rows (with separate counts for missing and complete rows);
- first-stage fitted values fall outside $[0, 1]$, whether or not clipped;
- a requested first-stage basis is capped;
- first-stage singularity is handled by ridge or column dropping, including the case of a marginal basis larger than its variable's support, which is identified in the warning;
- the second-stage design is rank deficient or ill-conditioned;
- R^2_G is undefined or requested without an intercept.

A negative response does not mechanically prevent calculation of the moment estimator, but it undermines the usual Gini interpretation and the ordinary nonnegative-loss interpretation of R^2_G.

13 Examples

The examples below assume that the package has been installed and attached with `library(GIR)`.

13.1 Basic fit with one continuous and two binary first-stage variables

```
fit <- gir(
  y ~ x1 + x2 + x3,
  data = dat,
  first_stage = ~ x1 + x2 + x3,
  continuous = "x1",
  discrete = c("x2", "x3"),
  basis = "legendre",
  GIR = TRUE
)

summary(fit)
summary(midcdf(fit))
fit$first_stage$diagnostics
```

The two binary variables generate up to four observed cells. The Legendre basis is estimated separately in each cell.

13.2 A numeric discrete variable with four values

```
fit4 <- gir(
  y ~ x + factor(d),
  data = dat4,
  first_stage = ~ x + d,
  continuous = "x",
  discrete = "d",          # important: d is numeric
  basis = "spline",
  spline_order = 4L
)

levels(fit4$first_stage$cells)
fit4$first_stage$diagnostics
```

If `d` takes $1, 2, 3, 4$, the first-stage diagnostic table has one row per observed value. The second-stage formula uses `factor(d)` to avoid imposing a linear effect in the numerical labels.

13.3 A Legendre specification with known support bounds

```
fit_legendre <- gir(
  y ~ 0 + x1 + x2 + x3,
  data = dat,
  first_stage = ~ x1 + x2 + x3,
  continuous = "x1",
  discrete = c("x2", "x3"),
  basis = "legendre",
  kappa = 0.245,
  basis_constant = 1,
  sieve_size_type = "dimension",
  sieve_size_reference = "cell",
  basis_bounds = list(x1 = c(0.1, 1)),
  clip = FALSE,
  first_stage_singular = "ridge",
  first_stage_tol = 1e-12,
  ridge = 1e-10,
  singular.ok = FALSE
)
```

13.4 A cubic spline specification with knots per margin

```
fit_spline <- gir(
  y ~ 0 + x1 + x2 + x3,
  data = dat,
  first_stage = ~ x1 + x2 + x3,
  continuous = "x1",
  discrete = c("x2", "x3"),
  basis = "spline",
  kappa = 0.300,
  basis_constant = 1,
  sieve_size_type = "knots",
  sieve_size_reference = "cell",
  spline_order = 4L,
  basis_bounds = list(x1 = c(0.1, 1)),
  clip = FALSE,
  first_stage_singular = "ridge",
  first_stage_tol = 1e-12,
  ridge = 1e-10,
  singular.ok = FALSE
)
```

Here `sieve_size_type = "knots"` makes $\lfloor n_g^{0.3} \rfloor$ the number of interior knots in cell g rather than the total dimension. With one continuous regressor, the cell's marginal and total dimensions equal the knot count plus four.

13.5 Target total dimension with two continuous regressors

```
fit_35 <- gir(
```

```
  y ~ x1 + x2,
  data = dat2,
  first_stage = ~ x1 + x2,
  continuous = c("x1", "x2"),
  basis = "spline",
  spline_order = 4L,
  sieve_size_type = "dimension",
  sieve_size = 35L,
  first_stage_singular = "ridge"
)

fit_35$first_stage$diagnostics$marginal_dimensions[[1L]]
# x1 x2
#  7  5

fit_35$first_stage$diagnostics$marginal_knot_counts[[1L]]
# x1 x2
#  3  1
```

Provided the cell-size cap does not bind, the tensor dimension is $7 \times 5 = 35$. With target 50, the balance-oriented path returns $(7, 7)$, hence 49 columns, rather than searching globally for the unbalanced factorization $10 \times 5 = 50$.

13.6 Directly specify marginal spline knots

```
fit_knots <- gir(
  y ~ x1 + x2,
  data = dat2,
  first_stage = ~ x1 + x2,
  continuous = c("x1", "x2"),
  basis = "spline",
  spline_order = 4L,
  sieve_size_type = "knots",
  sieve_size = c(3L, 1L)
)
```

This explicitly requests marginal dimensions $(7, 5)$ and total dimension 35. A scalar `sieve_size = 3L` would request three knots in both margins and total dimension $7^2 = 49$.

13.7 Compare cell and full reference sizes

```
fit_cell <- gir(
  y ~ x + factor(d), data = dat4,
  first_stage = ~ x + d,
  continuous = "x", discrete = "d",
  basis = "spline",
  sieve_size_reference = "cell"
)

fit_full <- gir(
  y ~ x + factor(d), data = dat4,
  first_stage = ~ x + d,
  continuous = "x", discrete = "d",
  basis = "spline",
  sieve_size_reference = "full"
```

```
)

fit_cell$first_stage$diagnostics[
  c("cell", "requested_size", "marginal_dimensions")
]

fit_full$first_stage$diagnostics[
  c("cell", "requested_size", "marginal_dimensions")
]
```

13.8 Supply the first-stage estimates

```
fit_external <- gir(
  y ~ treatment + x1 + x2,
  data = dat,
  midcdf = external_midcdf,
  clip = FALSE,
  singular.ok = FALSE
)

# A data-column name is also accepted.
fit_external_2 <- gir(
  y ~ treatment + x1 + x2,
  data = dat,
  midcdf = "external_midcdf"
)
```

13.9 Weights and missing values

```
fit_weighted <- gir(
  y ~ treatment + x,
  data = dat,
  weights = survey_weight,
  na.action = stats::na.omit,
  first_stage = ~ treatment + x,
  continuous = "x",
  discrete = "treatment",
  basis = "spline"
)
```

Rows incomplete in any required object are removed together, and the same weights enter both stages.

13.10 Inspect clipping

```
fit_unclipped <- gir(
  y ~ x, data = dat,
  continuous = "x",
  basis = "legendre",
  clip = FALSE
)
```

```
fit_clipped <- gir(
  y ~ x, data = dat,
  continuous = "x",
  basis = "legendre",
  clip = TRUE
)

range(midcdf(fit_unclipped))
range(midcdf(fit_clipped))
```

13.11 First-stage singularity policies

```
fit_strict <- gir(
  y ~ x, data = dat,
  continuous = "x",
  basis = "spline",
  first_stage_singular = "error"
)

fit_ridge <- gir(
  y ~ x, data = dat,
  continuous = "x",
  basis = "spline",
  first_stage_singular = "ridge",
  ridge = 1e-10
)

fit_drop <- gir(
  y ~ x, data = dat,
  continuous = "x",
  basis = "spline",
  first_stage_singular = "drop"
)
```

After a ridge or drop fit, inspect

```
fit_ridge$first_stage$diagnostics[
  c("cell", "singular", "action", "ridge_lambda")
]

fit_drop$first_stage$diagnostics[
  c("cell", "basis_dimension", "effective_dimension",
    "action", "dropped_columns")
]
```

13.12 Strict second-stage singularity behavior

```
fit <- gir(
  y ~ x1 + x2,
  data = dat,
  singular.ok = FALSE
)
```

This stops rather than returning aliased coefficients when the GIR moment matrix is singular.

13.13 GIR goodness of fit

```
fit_r2 <- gir(
  y ~ treatment + x1 + x2,
  data = dat,
  first_stage = ~ treatment + x1 + x2,
  continuous = c("x1", "x2"),
  discrete = "treatment",
  basis = "spline",
  GIR = TRUE
)

summary(fit_r2)
fit_r2$GIR.R2
```

14 `bboot.gir`: Bayesian bootstrap inference

14.1 Name and description

`bboot.gir()` obtains Bayesian-bootstrap coefficient draws from a fitted linear GIR model. In its default mode it re-estimates both the conditional mid-distribution first stage and the GIR second stage using the same mean-one exponential multipliers. It reports the basic confidence intervals in Algorithm 2 of the paper, bootstrap standard errors and covariance, bootstrap bias estimates, and centered-bootstrap hypothesis tests.

The interval construction and two-stage resampling scheme are taken from Algorithm 2 of the paper [1]. Exact ties use the mid-distribution response of the paper's first stage. The standard errors, smooth transformations, marginal tests, joint Wald test and prediction standard errors are package extensions; their assumptions and calculations are described below.

14.2 Usage

```
bboot.gir(
  object,
  R = 999L,
  level = 0.95,
  ...,
  transform = NULL,
  null = 0,
  alternative = c("two.sided", "less", "greater"),
  joint = FALSE,
  seed = NULL,
  cluster = NULL,
  cluster_order = c("auto", "original", "estimation"),
  bootstrap_weights = NULL,
  normalize_weights = TRUE,
  first_stage_boot = c("refit", "fixed"),
  midcdf_boot = NULL,
  inference = TRUE,
  keep_weights = FALSE,
  keep_midcdf = FALSE,
  failure = c("error", "omit")
)
```

The preferred workflow separates estimation from inference:

```
fit <- gir(y ~ x + factor(d), data = dat,
           first_stage = ~ x + d,
           continuous = "x", discrete = "d")
bb <- bboot.gir(fit, R = 999L, level = 0.95, seed = 42L)
summary(bb)
confint(bb)
vcov(bb)
```

Here and below, examples using an unspecified `dat` illustrate syntax; Section 16 supplies a complete runnable data-generating example.

14.3 Bootstrapping a fitted model

The fitted object identifies the actual estimation sample after `subset` and missing-value processing, the second-stage columns and contrasts, the continuous/discrete first-stage classification, and all selected cell-specific basis dimensions. Repeating a call against a subsequently modified data frame could otherwise change the estimator being bootstrapped.

A `gir` fit retains the data and settings used for estimation. `bboot.gir(fit)` uses that snapshot, not the possibly changed data named in `fit$call`. It does not allow extra `gir()` arguments when a fitted object is supplied. To change the basis or formula, fit the new point estimator first, then bootstrap that fit.

A formula convenience form is also available:

```
bb <- bboot.gir(
  y ~ x + factor(d),
  data = dat,
  first_stage = ~ x + d,
  continuous = "x", discrete = "d",
  basis = "spline", spline_order = 4L,
  singular.ok = FALSE,
  R = 999L, level = 0.95, seed = 42L
)
fit <- bb$fit
```

In this form, the first argument is the formula, not a fitted model. The additional, fully named arguments are passed to `gir()` once. Expressions for `subset`, `weights`, and a supplied `midcdf` are evaluated through the same data-based machinery as in a direct `gir()` call. `R`, `level`, and the other explicit bootstrap arguments remain bootstrap options, not point-estimator arguments.

14.4 Arguments: model, replicates and confidence level

`object`. Supply a fit created by `gir()`. A two-sided formula is accepted as the convenience form above. An older saved fit without the estimation snapshot must be refitted with the current function. A fitted second-stage model with aliased coefficients is not accepted for bootstrap coefficient inference: use an identified, full-rank specification.

`R`. The positive integer number of requested bootstrap replicates. Its default is 999. This is denoted by B in the mathematical formulas below. If a `bootstrap_weights` matrix is supplied and `R` is omitted, its number of columns determines `R`; an explicitly supplied inconsistent value is an error. With `inference=TRUE`, at least two successful draws are required. The choice `R=1`,

`inference=FALSE` provides a single draw, as used in a warp-speed Monte Carlo study; it does not provide a usable sample-specific standard error or confidence interval.

`level`. A scalar strictly between zero and one; the default is 0.95. The confidence intervals are two-sided, equal-tailed basic bootstrap intervals at that level. Changing the test's `alternative` does not change these intervals into one-sided bounds. A different interval level can be obtained from stored draws with `confint(bb, level=0.90)` or `summary(bb, level=0.90)`.

For a two-sided level $1-\alpha$, only about $B\alpha/2$ draws inform each tail. The summary warns when this number is below ten. This is a numerical precision diagnostic, not a theorem about a universally adequate value of B. The small B values used in unit tests and algebraic illustrations are not recommended for reported empirical tail inference.

14.5 Arguments: transformations and hypotheses

`transform`. Defines $h(\theta)$, the statistic or vector of statistics to test. It does not replace the raw coefficients stored in `t0` and `t`. The accepted forms are:

`NULL` Use all coefficients, in their original order. With `null=0`, this gives separate coefficient-zero tests by default.

Numeric vector
An unnamed vector must have one entry per coefficient and defines a single linear contrast. A named vector can select a subset of coefficient names; all unmentioned coefficients receive weight zero. For example, `c(x2=1, x3=-1)` defines $h(\theta) = \theta_{x2} - \theta_{x3}$. The resulting statistic is named `contrast`.

Numeric matrix
Each row defines a linear contrast. An unnamed matrix must have one column per coefficient. Named columns may be supplied in any order and may select a subset; omitted coefficient columns are filled with zero. Row names label the statistics; absent row names are generated as `contrast1`, and so on.

Function
A deterministic function of the named coefficient vector. It must return a finite numeric vector of the same length and with the same names for the point estimate and every successful draw. An unnamed result is assigned names `statistic1`, and so on. Undefined ratios, logs, or other nonfinite values cause an error rather than selective deletion of transformed draws.

The intended theoretical class is smooth transformations with a nondegenerate first derivative at the true parameter. The numerical interface does not verify this mathematical condition. In particular, $h(\theta) = \theta_j^2$ when $\theta_{0j} = 0$, absolute values at zero, maxima at ties, and ratios with nearly zero denominators may be nonregular; obtaining numeric output does not establish bootstrap validity for them. For joint inference, the derivative matrix of the restrictions must have full row rank at the null. A necessary check, enforced in the code, is that the number of restrictions does not exceed the number of coefficients; that check is not sufficient on its own.

`null`. The hypothesized value r in $H_0 : h(\theta_0) = r$. An unnamed scalar is repeated for all transformed components. An unnamed vector must have exactly one entry per component. A named vector must match all transformed statistic names exactly and is reordered accordingly. The default is zero. For a coefficient ratio equal to one, supply `null=1`; for a log-ratio equal to zero, supply `null=0`.

alternative. `"two.sided"` uses twice the smaller centered-bootstrap tail probability, capped at one. `"less"` tests against $h_j(\theta_0) < r_j$, and `"greater"` against $h_j(\theta_0) > r_j$. The same alternative applies to each component. Marginal p-values are not adjusted for multiple testing.

joint. If `TRUE`, also compute the centered, common-covariance bootstrap Wald test of all supplied restrictions jointly. It requires `alternative="two.sided"`. This is not the ordinary OLS F test, and it does not use a Student or F reference distribution. A singular or numerically ill-conditioned bootstrap restriction covariance yields an explicit warning and an undefined joint statistic/p-value, not an arbitrary generalized-inverse test. Remove redundant restrictions or obtain more draws as appropriate.

For example, testing both $\theta_2 - \theta_3 = 0$ and $\theta_2/\theta_3 = 1$ jointly is redundant when the denominator is nonzero. Use one of these equivalent restrictions, not both as independent restrictions.

14.6 Arguments: random weights and reproducibility

seed. `NULL` uses and advances the caller's random-number stream. A nonnegative integer initializes a local seeded computation and restores both the caller's RNG kind and its previous `.Random.seed` on exit, including error exits. A supplied seed and supplied bootstrap multipliers cannot be used together.

cluster. Optional clustering of the multipliers. A one-sided formula such as `~ industry` names retained model-frame or first-stage columns, whose combination defines the clusters (`~ industry + region` clusters on the intersection); a vector supplies one cluster value per estimation row, or one per row of the original data. The `cluster_order` argument below determines how a vector is aligned. For each replicate one Exp(1) value is drawn per cluster and shared by all its observations in both stages, after which the vector is normalized by its observation-level mean as usual. This is the Bayesian cluster bootstrap and is the appropriate choice when observations are dependent within groups, for example firms within industries when the treatment varies at the industry level. It cannot be combined with `bootstrap_weights`; a supplied multiplier matrix already fixes the dependence structure.

Each distinct combination of values defines a cluster. Group membership uses the values in each column, so labels such as `("a.b", "c")` and `("a", "b.c")` remain distinct. Numeric identifiers are distinguished by their stored values rather than their printed formatting. A formula such as `~ industry + region` specifies one-way clustering at industry-region intersections, not two-way clustering by industry and region separately. Clusters are numbered in order of first occurrence in the estimation rows; changing labels while preserving group membership preserves seeded multipliers.

When every first-stage cell lies inside a single cluster, the cell fits are invariant to the common multiplier of that cluster, so the first stage does not vary across draws and the bootstrap distribution reflects between-cluster variation only. The estimated first stage then enters the inference through the residual variation across clusters, exactly as in a clustered regression of cell Gini indices; the returned `cluster$first_stage_invariant` reports whether this is the case. The paper's validity theorem covers independent observations; clustered inference rests on the usual many-clusters argument and is the researcher's responsibility.

cluster_order. Specifies the row order of a cluster vector:

- `"original"` requires one label per original-data row, in the order at the time `gir()` was called. The vector is aligned to the final estimation sample using `fit$row.index`, including

subsetting, reordering, repeated selections, and removal of incomplete observations.

- `"estimation"` requires one label per final estimation row and uses the vector unchanged. For example, a vector already constructed as `dat$id[fit$row.index]` has this order.
- `"auto"` (the default) infers the order when the vector length matches only one of the original-data and estimation-sample sizes. If both sizes match, it accepts the vector only when the row mapping is unchanged. Otherwise it stops and asks for an explicit order, since length alone cannot distinguish original-order labels from already-aligned labels.

Missing cluster labels are checked after alignment. A missing label on a discarded original row is harmless; a missing label on a used row is rejected. Non-default values require a cluster vector. Leave `"auto"` for a cluster formula, whose retained columns are already aligned, or for `cluster = NULL`.

For example, keeping all six observations in a different order requires an explicit choice for a vector. The following computes draws only to illustrate alignment:

```
cluster_dat <- data.frame(y = c(1, 4, 2, 7, 3, 9),
                          id = c("A", "A", "B", "B", "C", "C"))
cluster_fit <- gir(y ~ 1, data = cluster_dat,
                   subset = c(1, 3, 5, 2, 4, 6), first_stage = ~ id,
                   discrete = "id", minimum_cell_size = 1L)
cluster_boot <- bboot.gir(cluster_fit, cluster = cluster_dat$id,
                          cluster_order = "original", R = 3L,
                          seed = 17L, inference = FALSE)
cluster_boot$cluster$index  # 1 2 3 1 2 3
```

An equivalent call supplies `cluster_dat$id[cluster_fit$row.index]` with `cluster_order = "estimation"`. A retained-column formula `cluster = ~ id` also gives the same grouping without a vector-order choice. Earlier analyses that supplied misaligned vectors should rerun their bootstrap: inference can change even when the cluster count is unchanged. Point estimates are unaffected.

`bootstrap_weights`. Optional positive finite multipliers, supplied as an $n_{\text{used}} \times B$ matrix, with observations in rows and replicates in columns. A numeric vector gives one replicate. The order must be the final estimation-row order, not the original unfiltered data order. These are multipliers, not original observation weights and not products of the two. The code multiplies them by `fit$weights` once.

This argument allows paired comparisons: the same multiplier vector can be passed to Legendre, spline, and oracle fits. Supplying it consumes no random numbers in the internal GIR implementation. An external `midcdf_boot` callback must also avoid extra randomness for this property to apply to the entire computation.

`normalize_weights`. With `TRUE`, divide each supplied multiplier column by its own mean. With `FALSE`, use and store the supplied entries unchanged. For the internal estimator, including relative ridge handling, a common rescaling preserves the draws up to floating-point error within usable numerical ranges. External first-stage callbacks must respect weight scaling themselves. Generated exponential multipliers are always normalized as in Algorithm 2, irrespective of this option. Normalization is over the full estimation sample, not separately inside each discrete cell.

Zero or negative supplied multipliers are rejected because they are not the positive exponential-weight scheme implemented here. Original case weights may be zero, as in `gir()`; their effective bootstrap weights remain zero. Underflow or overflow in the combined weights is an error, not a reason to draw a replacement multiplier vector.

14.7 Arguments: first-stage refitting

`first_stage_boot`. The default `"refit"` re-estimates the first stage for every multiplier draw. It uses the original sample, cells, support transformation, knot locations, constructed marginal dimensions, and tensor-product specification. The original numerical singularity policy and clipping rule are applied again under the bootstrap weights. The numerical rank and any dropped columns can therefore change with the weights if the selected policy allows it; those changes are recorded in the diagnostics.

The explicit alternative `"fixed"` holds the original mid-CDF vector fixed and reweights only the second stage. It is suitable for an oracle first stage, where the true conditional distribution is treated as known, or another truly known nuisance. It is not the default for an estimated first stage and produces a warning when inferential summaries are requested. Holding an estimated first-stage vector fixed generally omits its estimation effect.

`midcdf_boot`. An optional callback for a point fit with an externally supplied first stage:

```
function(object, weights, replicate) {
  # Re-estimate the external method under these combined weights.
  # Return one mid-CDF value per retained estimation row.
}
```

The `weights` argument contains the product of the original observation weights and that replicate's multiplier vector, which is left unnormalized when supplied with `normalize_weights = FALSE`. The callback must return a finite numeric vector of length `nobs(object)` in the same row order. The original clipping rule is subsequently applied.

A stored external mid-CDF vector alone is not enough to reconstruct how it was estimated. Thus `first_stage_boot="refit"` errors for such a fit unless this callback is supplied. A callback is not accepted for an internally estimated series first stage or together with `first_stage_boot="fixed"`. The callback's own statistical validity remains the researcher's responsibility.

14.8 Arguments: output, failure handling and extra arguments

`inference`. `TRUE` computes intervals, standard errors, bias summaries, and tests. `FALSE` returns raw draws and diagnostics without those summaries. With at least two successful stored draws, `summary(bb)` can compute summaries later, even if the original call was draw-only. One draw alone is insufficient.

`keep_weights`. Retain the multiplier matrix as used in `bb$bootstrap_weights`; supplied values remain unnormalized when `normalize_weights = FALSE`. The default `FALSE` avoids storing an additional $n \times B$ matrix. These saved values are the multipliers, not the combined case weights.

`keep_midcdf`. Retain the first-stage fitted values for every draw in `bb$midcdf`, an $n \times B$ matrix. This is useful for nuisance diagnostics and reproducibility checks, but can use substantial memory. For example, $n = 100000$ and $B = 999$ require roughly 800 MB for the numeric entries of one such matrix, before object overhead. The default is `FALSE`.

`failure`. `"error"` stops at the first failed replicate with its number and reason. This is the default. `"omit"` keeps the failed replicate's entire coefficient row as `NA`, records the error, and computes summaries from successful rows only, with a warning. It never redraws until a successful fit appears.

Omitting failures conditions on numerical success and can distort inference; a substantial failure rate should be investigated, not accepted as routine. An undefined coefficient transformation is an error even with this option, because selectively removing transformed draws would change the statistic's bootstrap distribution.

`....` Only the formula convenience form passes these fully named options to `gir()`. With a fitted object, any extra argument is an error. Thus a misspelling or an attempted change of basis cannot silently leave the original point estimator and bootstrap estimator different.

14.9 The two-stage bootstrap calculation

For clarity, write a_i for the original case weights (equal to one in the paper's baseline sampling setting), and $w_i^{*(b)}$ for the bootstrap multipliers. For each replicate b, draw

$$e_i^{(b)} \overset{\text{iid}}{\sim} \text{Exp}(1), \qquad \bar{e}^{(b)} = \frac{1}{n}\sum_{i=1}^{n} e_i^{(b)}, \qquad w_i^{*(b)} = e_i^{(b)}/\bar{e}^{(b)}.$$

Let $a_i^{*(b)} = a_i w_i^{*(b)}$. Within a discrete cell g, form

$$\begin{aligned} Q_g^{*(b)} &= \sum_{j \in \mathcal{I}_g} a_j^{*(b)} p_j p_j', \\ S_g^{*(b)}(Y_i) &= \sum_{j \in \mathcal{I}_g} a_j^{*(b)} p_j \left\{ \mathbf{1}(Y_j < Y_i) + \tfrac{1}{2}\mathbf{1}(Y_j = Y_i) \right\}, \\ \widehat{M}_i^{*(b)} &= p_i' \{Q_g^{*(b)}\}^{-1} S_g^{*(b)}(Y_i). \end{aligned}$$

The displayed inverse describes the unregularized full-rank calculation; the explicitly selected singularity policy applies when necessary. With the same weights, compute

$$\begin{aligned} A^{*(b)} &= \frac{1}{n}\sum_i a_i^{*(b)} Y_i X_i X_i', \\ b^{*(b)} &= \frac{1}{n}\sum_i a_i^{*(b)} Y_i X_i \{2\widehat{M}_i^{*(b)} - 1\}, \\ \widehat{\theta}^{*(b)} &= \{A^{*(b)}\}^{-1} b^{*(b)}. \end{aligned}$$

Using different independent multiplier vectors in the two stages would discard their joint resampling dependence and would not implement Algorithm 2.

Basis construction, sorting, and exact tie positions are cached once per `bboot.gir()` call. Gram matrices and cumulative weighted scores must still be recomputed for each draw. This retains the threshold-sorting advantage without reconstructing every formula and basis B times. It does not approximate thresholds or replace refitting by a fixed-first-stage shortcut.

The second-stage moments depend on X_i only through its value. Both `gir()` and every bootstrap draw therefore first sum $a_i^{*(b)} Y_i$ and $a_i^{*(b)} Y_i \{2\widehat{M}_i^{*(b)} - 1\}$ over the observations that share a regressor row, and then form $A^{*(b)}$ and $b^{*(b)}$ from the distinct rows. The result is identical to the observation-level formulas up to the order of floating-point summation. With cell-level regressors the cost of a draw's second stage is proportional to the number of distinct rows rather than to the sample size.

With `cluster`, the exponential values are drawn per cluster, $e_c^{(b)} \overset{\text{iid}}{\sim} \text{Exp}(1)$ for $c = 1, \ldots, C$, and every observation i in cluster $c(i)$ receives $w_i^{*(b)} = e_{c(i)}^{(b)}/\bar{e}^{(b)}$, with $\bar{e}^{(b)}$ the mean over the n observations. The two stages are computed exactly as above with these multipliers.

Multiplying nonuniform case weights by iid multipliers is implemented as a weighted extension. The paper's iid, unweighted validity theorem does not alone justify design-based inference for complex surveys, clustered multipliers, panel dependence, or time series. The function warns for nonuniform/zero original weights when inference is requested. It supplies the cluster option described above but no survey-design bootstrap.

14.10 Basic confidence intervals: the paper's Algorithm 2

Write B for the number of successful draws, $\alpha = 1 - \texttt{level}$, and

$$\Delta_{bj} = \widehat{\theta}_j^{*(b)} - \widehat{\theta}_j.$$

For each coordinate, let $q^*_{u,j}$ be the empirical quantile of these centered *unscaled* draws, computed by `stats::quantile(type=7)`. The interval is

$$\boxed{\left[\widehat{\theta}_j - q^*_{1-\alpha/2,j},\ \widehat{\theta}_j - q^*_{\alpha/2,j}\right].}$$

This is exactly the centered/basic construction in Algorithm 2. It is not $[q_{\alpha/2}(\widehat{\theta}^*_j), q_{1-\alpha/2}(\widehat{\theta}^*_j)]$, which would be a percentile interval. Despite the Bayesian-bootstrap weight scheme, these reported intervals are frequentist basic confidence intervals, not equal-tail posterior credible intervals.

The factor $\sqrt{n}$ from the asymptotic theorem is unnecessary here. Were one to take quantiles of $\sqrt{n}\Delta_{bj}$, one would divide those quantiles by $\sqrt{n}$ when constructing the interval, obtaining the same endpoints. Type-7 quantiles are R's default. A different quantile interpolation rule can cause a finite-B discrepancy even with exactly the same draws.

Numerical example. Suppose $\widehat{\theta}_j = 0.20$, $B = 5$, and the centered draws are

$$(-0.06, -0.03, -0.01, 0.02, 0.05).$$

For an illustrative 80% interval, type-7 interpolation gives $q^*_{0.1} = -0.048$ and $q^*_{0.9} = 0.038$. Thus the basic interval is

$$[0.20 - 0.038,\ 0.20 - (-0.048)] = [0.162, 0.248].$$

The corresponding raw draws are $(0.14, 0.17, 0.19, 0.22, 0.25)$; their percentile interval would instead be $[0.152, 0.238]$. The methods differ. Five replicates are used only to expose the arithmetic.

14.11 Standard errors, covariance and bootstrap bias

Let $\bar{\theta}^* = B^{-1}\sum_{b=1}^{B}\widehat{\theta}^{*(b)}$. The implemented coefficient covariance estimate is

$$\widehat{V}_{\text{boot}} = \frac{1}{B-1}\sum_{b=1}^{B}(\widehat{\theta}^{*(b)} - \bar{\theta}^*)(\widehat{\theta}^{*(b)} - \bar{\theta}^*)', \qquad \widehat{\text{se}}_j = \sqrt{(\widehat{V}_{\text{boot}})_{jj}}.$$

These are `cov(bb$t[bb$success, ])` and the corresponding column standard deviations. The covariance concerns $\widehat{\theta}$, approximately Σ/n, not the limiting covariance Σ of $\sqrt{n}(\widehat{\theta} - \theta_0)$. No additional division by n, $\sqrt{n}$, or $\sqrt{B}$ is made.

Dividing the bootstrap standard deviation by $\sqrt{B}$ estimates the Monte Carlo error of a bootstrap *average*, a different quantity. Nor is an OLS residual degrees-of-freedom factor $n - p$ used. The first-stage estimation effect is captured by jointly refitting the two stages.

The reported bootstrap bias estimate is

$$\widehat{\text{bias}}_{\text{boot}} = \bar{\theta}^* - \widehat{\theta}.$$

It is descriptive and does not replace the point estimate by a bias-corrected one. In the five-draw example, $\bar{\theta}^* = 0.194$, the bias estimate is -0.006, and the sample standard deviation is approximately 0.0427785.

What the theorem does and does not establish. Conditional weak convergence of the centered bootstrap law supports the stated quantile construction at continuity points of the limiting distribution. By itself it does *not* imply convergence of its second moments. Interpreting the conventional bootstrap covariance as a consistent variance estimator also requires appropriate moment convergence, for example conditional uniform integrability of squared scaled errors. The paper's proof does not separately establish that additional result. This distinction, studied explicitly for weighted semiparametric bootstraps by Cheng [8], also applies to covariance-based prediction SEs and the common-covariance joint test. The code computes the usual empirical covariance; it is not a substitute for checking those additional regularity conditions.

14.12 Smooth transformations and the bootstrap delta argument

For $h : \mathbb{R}^p \to \mathbb{R}^q$, set

$$\widehat{\eta} = h(\widehat{\theta}), \qquad \widehat{\eta}^{*(b)} = h(\widehat{\theta}^{*(b)}), \qquad D_b = \widehat{\eta}^{*(b)} - \widehat{\eta}.$$

Compute transformed basic intervals and standard deviations directly from these transformed draws, not by transforming the already computed raw coefficient interval endpoints. The latter operation is generally not equivalent for nonlinear transformations or contrasts of correlated coefficients.

If h is continuously differentiable near θ_0 and its relevant limiting derivative is nondegenerate, the first-order expansions are

$$\sqrt{n}\{h(\widehat{\theta}) - h(\theta_0)\} = Dh(\theta_0)\sqrt{n}(\widehat{\theta} - \theta_0) + o_p(1),$$
$$\sqrt{n}\{h(\widehat{\theta}^*) - h(\widehat{\theta})\} = Dh(\theta_0)\sqrt{n}(\widehat{\theta}^* - \widehat{\theta}) + o_{P_w}(1).$$

Thus, under the bootstrap theorem and the additional transformation regularity, the second law approximates the first. This is the justification for the smooth extensions, not a claim that every function accepted by R is regular.

For a fixed contrast $L\theta$, the covariance is $L\widehat{V}_{\text{boot}}L'$. In particular,

$$\widehat{\text{se}}(\widehat{\theta}_2 - \widehat{\theta}_3)^2 = \widehat{V}_{22} + \widehat{V}_{33} - 2\widehat{V}_{23}.$$

The off-diagonal covariance cannot generally be discarded.

14.13 Marginal bootstrap hypothesis tests

For $H_0 : \eta_{0j} = r_j$, let $d_j = \widehat{\eta}_j - r_j$. The reference draws are the *centered errors* D_{bj}, not the uncentered $\widehat{\eta}_j^{*(b)}$. The implemented tail probabilities are

$$\widehat{p}_{j,\text{less}} = \frac{1 + \sum_{b=1}^{B} \mathbf{1}\{D_{bj} \le d_j\}}{B+1},$$
$$\widehat{p}_{j,\text{greater}} = \frac{1 + \sum_{b=1}^{B} \mathbf{1}\{D_{bj} \ge d_j\}}{B+1},$$
$$\widehat{p}_{j,\text{two.sided}} = \min\{1,\ 2\min(\widehat{p}_{j,\text{less}}, \widehat{p}_{j,\text{greater}})\}.$$

Under the null, both the observed deviation and the centered bootstrap error have the same first-order limiting law, subject to the conditions just stated. The equal-tail convention is chosen to match the tail logic of Algorithm 2.

The plus-one numerator/denominator is an explicit finite-simulation safeguard against reporting a zero tail probability merely because no finite set of draws exceeded the observed value. It vanishes as B increases. It does not make this approximate bootstrap test an exact finite-sample randomization test. The smallest possible reported two-sided p-value is $2/(B+1)$, up to the cap at one. Quantile interpolation and the plus-one rule mean that a finite-B p-value decision and a type-7 interval exclusion can differ near a threshold; no exact finite-B inversion equivalence is asserted.

If all transformed draws are identical, the numerical SE is zero but the marginal p-value is set to `NA` with a warning. A degenerate computed interval is not evidence that the parameter is known without uncertainty.

14.14 Joint bootstrap Wald test

Let $\widehat{V}_h$ be the sample covariance of the transformed draws. For $H_0 : h(\theta_0) = r$, the common-covariance statistics are

$$W_{\text{obs}} = (\widehat{\eta} - r)' \widehat{V}_h^{-1} (\widehat{\eta} - r), \qquad W_b^* = D_b' \widehat{V}_h^{-1} D_b.$$

The p-value is

$$\widehat{p}_{\text{joint}} = \frac{1 + \sum_{b=1}^{B} \mathbf{1}\{W_b^* \geq W_{\text{obs}}\}}{B+1}.$$

The same covariance matrix is used in all these statistics. There is no nested bootstrap and no replicate-specific standard-error estimate. A numerical correlation-matrix scaling improves the rank check's invariance to measurement units without changing this quadratic form.

Regular joint inference requires a full-row-rank restriction derivative at the null and a positive-definite limiting restriction covariance, in addition to the relevant bootstrap moment consistency. More draws cannot repair mathematically redundant restrictions. The component named `df` records the number of restrictions; the p-value above is bootstrap calibrated, not a chi-squared or F tail computed from that number.

14.15 Returned object and extraction methods

The result has class `"gir_bootstrap"`. Its components `t0`, `t`, and `R` follow the naming convention of `boot`, but use the GIR extraction methods listed below to work with this object.

Component	Meaning
`fit`	The point-estimation object used, including its resolved data and basis specification.
`t0`	Named raw coefficient vector $\widehat{\theta}$.
`t`	$R \times p$ raw coefficient draws; failed rows are entirely `NA`.
`R`, `R.success`, `success`	Requested count, successful count, and logical success indicator.
`std.error`, `covariance`	Raw coefficient bootstrap SEs and covariance.
`conf.int`, `bias`	Raw basic intervals and bootstrap mean-minus-estimate bias.
`coefficient.table`	Raw Estimate, Boot. SE, Boot. bias, lwr and upr table.
`test`	Transformed estimate/null/deviation, SEs, covariance, basic intervals, marginal p-values, and optional joint test.
`level`, `quantile.type`	Requested level and quantile type 7.
`transform`, `null`, `alternative`, `joint`	Inferential specification.
`first_stage_boot`	Refitted, fixed, or external-refit mode.

Component	Meaning
`cluster`	`NULL`, or the number of clusters, the cluster index over estimation rows, and whether the first-stage cells are nested in the clusters.
`seed`, `rng.kind`, `rng.start`	Seed option, RNG kind, and captured starting RNG state.
`bootstrap_weights`	Optional saved multipliers as used, $n \times R$.
`midcdf`	Optional saved first-stage draw matrix, $n \times R$.
`diagnostics`	One row per attempted replicate, including success/error, conditioning and warning information.

With `inference=FALSE`, the summary fields are `NULL`; draw matrices and diagnostics remain available. Extraction and reanalysis use:

```
coef(bb)                          # raw point coefficients, not bootstrap means
vcov(bb)                          # raw coefficient covariance
confint(bb, level = 0.90)          # raw coefficient basic intervals
confint(bb, parm = c("x2", "x3"))
summary(bb, transform = c(x2 = 1, x3 = -1), null = 0)
bb$test$conf.int                  # transformed intervals from the original call
bb$test$p.value
bb$test$joint
bb$diagnostics
```

`summary(bb)` can change the confidence level, transformation, null, alternative, and joint-test choice without refitting or generating draws. Its coefficient table concerns the raw coefficients; its `test` component concerns the selected transformations. The original `bb` is not mutated by obtaining a new summary.

The bootstrap diagnostics include the minimum first-stage reciprocal condition number across cells, labels of cells receiving a numerical adjustment, the number of out-of-range mid-CDF values before clipping, second-stage reciprocal condition number, error messages, and a list of captured warning messages. Original cellwise marginal dimension/knot vectors remain at `bb$fit$first_stage$diagnostics`. The constructed basis allocation is fixed throughout that bootstrap run.

15 `predict.gir`: linear conditional Gini predictions

15.1 Name, target and usage

`predict.gir()` evaluates the fitted linear conditional Gini index at specified covariates. It is normally invoked through the standard generic `predict(fit, ...)`. The target is

$$G(x) = x'\theta_0 \quad \text{under correct specification,} \qquad \widehat{G}(x) = x'\widehat{\theta}.$$

Under misspecification, $x'\theta_0$ is the fitted linear-approximation target identified in the paper, not necessarily the true CGIF at each individual x. Confidence statements concern that approximation target when the model is misspecified.

The method does not predict Y, $E[Y \mid X]$, or $M(y \mid x)$. A model for the conditional Gini alone does not specify the conditional law needed for individual-outcome prediction intervals.

```
predict.gir(
  object,
  newdata = NULL,
```

```
  se.fit = FALSE,
  interval = c("none", "confidence"),
  level = 0.95,
  bootstrap = NULL,
  R = 999L,
  seed = NULL,
  na.action = stats::na.pass,
  ...
)
```

15.2 Arguments

`object`. A current `gir` fit. New uncertainty calculations require an identified, full-rank coefficient specification. Point predictions from aliased fits are allowed only where the requested linear function is estimable; see below.

`newdata`. A data frame containing the raw predictor variables used by the second-stage formula. The response and variables used only in the first stage are not needed. `NULL` uses the original retained second-stage design and produces values in the same order as `fitted(fit)`. That order contains the estimation sample only; it does not reinsert observations removed by the original fit, even when the original fitting action was `na.exclude`. Use `fit$row.index` to map to original data rows.

For new data, the method uses the original model terms, contrasts, and factor levels. Training parameters stored by safe formula transformations such as `poly(x, 2)` or `splines::bs(x, ...)` are reused. New factor levels are errors; missing required predictors are errors rather than an intentional fallback to an unrelated vector in the calling environment. Custom formula functions must themselves support meaningful evaluation on new observations.

`se.fit`. If `FALSE`, return a prediction vector or interval matrix. If `TRUE`, return a list with components `fit`, `se.fit`, `level`, `method`, and `R`. SEs concern the estimated linear index, not observation noise. Setting this option requests a bootstrap when none is supplied.

`interval`. `"none"` returns point values. `"confidence"` returns a matrix with columns `fit`, `lwr`, and `upr`, using pointwise basic bootstrap intervals. There is no `"prediction"` option for individual outcomes and no simultaneous confidence-band option in this version.

`level`. Confidence level strictly between zero and one. Its default is 0.95, even if an existing bootstrap object was originally summarized at a different level. The stored draws can be reused at the level requested here.

`bootstrap`. An optional matching result of `bboot.gir(fit, ...)`. Reusing it is the preferred method when several predictions or transformed tests are needed. Compatibility is checked against the raw coefficient vector, estimation snapshot, first-stage specification, and point mid-CDF vector. A bootstrap of a different sample or modified estimator is rejected. Raw coefficient draws are used even if the supplied bootstrap result also contains tests of nonlinear transformations.

`R` and `seed`. Used only for an automatic `bboot.gir()` call when uncertainty is requested and `bootstrap=NULL`. They have the same meaning as in that function. Do not supply these controls

together with an existing bootstrap result. Explicit bootstrap controls without `se.fit=TRUE` or `interval="confidence"` are errors. A pure point-prediction call draws no random numbers.

`na.action`. Applies to `newdata`, not to the already fitted sample. The default `na.pass` keeps positions corresponding to missing predictors and reports `NA` predictions/SEs there. `na.omit` removes incomplete rows. `na.exclude` removes them for matrix construction and reinserts their missing positions in the returned prediction and SE objects. `na.fail` stops. Infinite transformed design values are rejected. Empty `newdata`, or omission of all prediction rows, returns a length-zero point vector or a zero-row confidence-interval matrix with columns `fit`, `lwr` and `upr`. With `se.fit=TRUE`, the standard-error vector has matching length. `na.exclude` restores `NA` positions, including when all prediction rows were missing.

`...`. Extra `bboot.gir()` controls for an automatic bootstrap only, for example `cluster` and `cluster_order`, or setting `first_stage_boot` to `"fixed"` for a known nuisance. Cluster vectors describe the original fitting data or estimation sample, never `newdata`. These controls are not `gir()` options and cannot change the point fit. With an existing bootstrap, extra controls are rejected. A simple prediction needs none of them.

15.3 Prediction standard errors and intervals

For a row vector x', define the bootstrap predictions

$$\widehat{G}^{*(b)}(x) = x'\widehat{\theta}^{*(b)}.$$

Their sample standard deviation is, up to floating-point arithmetic,

$$\boxed{\widehat{\mathrm{se}}\{\widehat{G}(x)\} = \sqrt{x'\widehat{V}_{\mathrm{boot}}x}.}$$

The confidence interval follows the same basic construction as Algorithm 2, now for a fixed linear functional:

$$\boxed{\left[\widehat{G}(x) - q^*_{1-\alpha/2}(x),\ \widehat{G}(x) - q^*_{\alpha/2}(x)\right],}$$

where $q^*_u(x)$ is the type-7 quantile of $\{x'(\widehat{\theta}^{*(b)} - \widehat{\theta}) : 1 \le b \le B\}$. Coefficient estimation uncertainty from both stages is already represented in these joint draws. No OLS residual variance is added.

For an intercept and one covariate,

$$\widehat{\mathrm{se}}\{\widehat{G}(x)\}^2 = \widehat{V}_{00} + 2x\widehat{V}_{01} + x^2\widehat{V}_{11}.$$

For example, suppose $\widehat{\theta} = (0.10, 0.20)'$ and

$$\widehat{V} = \begin{pmatrix} 0.0004 & -0.0001 \\ -0.0001 & 0.0009 \end{pmatrix}.$$

At $x = 0.5$, the fitted Gini is 0.20 and its variance estimate is $0.0004 - 0.0001 + 0.000225 = 0.000525$, giving an SE of approximately 0.022913. The basic interval still requires the projected draws; this variance number alone does not determine its possibly asymmetric endpoints.

Adding the marginal lower coefficient bounds to obtain a lower prediction bound is generally incorrect. Predictions use the same joint coefficient draw for all terms, preserving covariance. Computation is divided into blocks of prediction rows to avoid storing the entire newdata-by-bootstrap matrix at once.

15.4 Pointwise coverage, range and extrapolation

Each interval applies to the specified fixed covariate row. A collection of pointwise 95% intervals is not a 95% simultaneous band over all rows or over a continuum of x. No claim of uniform coverage is made for a growing or post-selected prediction grid.

Neither predictions nor confidence endpoints are clipped to $[0, 1]$. Out-of-range point predictions cause a warning and remain visible. This is separate from `gir(clip=TRUE)`, which clips first-stage mid-CDF values. A constrained nonlinear Gini model or interval transformation would be a separate specification, not a display change.

The linear index can be evaluated outside the observed covariate range, but this is extrapolation. The method does not guarantee overlap for an unobserved joint categorical profile or correctness of the linear model outside the support used for estimation.

For an aliased second-stage fit, a new linear functional must lie in the estimable row space of the weighted training design. Non-estimable rows return `NA` with a warning. Bootstrap prediction inference nevertheless requires a full-rank coefficient specification in this implementation. Refitting the same identified model without redundant columns is preferable to obtaining coefficient uncertainty for an arbitrary alias convention.

16 Complete worked examples

16.1 Generate data from the paper's continuous-binary design

The following is a self-contained setup using base R. It draws one dataset from the paper's simulation design with one continuous and two binary regressors.

```
library(GIR)

set.seed(150L)
n <- 2000L
x1 <- runif(n, 0.1, 1)
x2 <- rbinom(n, 1L, 0.5)
x3 <- rbinom(n, 1L, 0.5)
m <- 0.075*x1 + 0.100*x2 + 0.125*x3
alpha <- (1 - m) / (2*m)
y <- rbeta(n, shape1 = alpha, shape2 = 1)
dat <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3)
```

The true coefficient vector is $(0.075, 0.100, 0.125)'$, with no second-stage intercept. The true conditional CDF and mid-CDF coincide because the response law is continuous: $M(y \mid x) = y^{\alpha(x)}$ on $[0, 1]$.

16.2 Fit and bootstrap the spline specification

```
fit <- gir(
  y ~ 0 + x1 + x2 + x3, data = dat,
  first_stage = ~ x1 + x2 + x3,
  continuous = "x1", discrete = c("x2", "x3"),
  basis = "spline", spline_order = 4L,
  kappa = 0.300, basis_constant = 1,
  sieve_size_type = "knots", sieve_size_reference = "cell",
  basis_bounds = list(x1 = c(0.1, 1)), clip = FALSE,
```

```
  first_stage_singular = "ridge", first_stage_tol = 1e-12,
  ridge = 1e-10, singular.ok = FALSE
)

bb <- bboot.gir(fit, R = 999L, level = 0.95, seed = 42L)
summary(fit)                  # point-estimator summary
summary(bb)                   # bootstrap inference summary
coef(bb)                      # same point coefficients as coef(fit)
confint(bb)                   # Algorithm 2 intervals
bb$std.error
vcov(bb)
bb$bias
bb$diagnostics
fit$first_stage$diagnostics$marginal_dimensions
fit$first_stage$diagnostics$marginal_knot_counts
```

This call selects `sieve_size_type = "knots"` explicitly, so that $\lfloor n_g^{0.3} \rfloor$ is the number of interior knots; the package default applies the rule to the total dimension instead. With this seed, some series mid-CDF estimates fall outside $[0, 1]$, so the fit and bootstrap report warnings. Inspect `midcdf(fit)` and `bb$diagnostics`. The example retains those estimates; choosing `clip = TRUE` would change the estimator.

16.3 The formula convenience form

```
bb_formula <- bboot.gir(
  y ~ 0 + x1 + x2 + x3,
  data = dat,
  first_stage = ~ x1 + x2 + x3,
  continuous = "x1", discrete = c("x2", "x3"),
  basis = "spline", spline_order = 4L,
  sieve_size_type = "knots", sieve_size_reference = "cell",
  basis_bounds = list(x1 = c(0.1, 1)),
  first_stage_singular = "ridge", singular.ok = FALSE,
  R = 999L, level = 0.95, seed = 42L
)
# Same model, defaults and local seed as the preceding example.
all.equal(bb$t, bb_formula$t, tolerance = 1e-12)
```

The fitted-object form is shorter and more resistant to accidentally changing an option between estimation and inference.

16.4 Contrasts, Gini points and a coefficient ratio

```
# Difference between two coefficients: H0 theta_x2 - theta_x3 = 0.
s_diff <- summary(bb, transform = c(x2 = 1, x3 = -1), null = 0)
s_diff$test$table
s_diff$test$conf.int

# Express the x2 coefficient in Gini points (100 times the index).
s_points <- summary(bb, transform = c(x2 = 100), null = 0)
s_points$test$table

# Nonlinear example; appropriate only for a well-separated nonzero denominator.
ratio <- function(b) c(ratio = unname(b["x2"] / b["x3"]))
s_ratio <- summary(bb, transform = ratio, null = 1)
```

```
s_ratio$test$table
```

These calls reuse the same raw draws. They do not require an analytic derivative from the user, but smoothness and nondegeneracy are still assumptions. A ratio interval can be unreliable near a zero denominator even when no numeric error is raised.

16.5 Joint and directional tests

```
# Nonredundant joint restrictions: H0 theta_x2 = theta_x3 = 0.
selected <- function(b) b[c("x2", "x3")]
s_joint <- summary(bb, transform = selected, null = c(x2 = 0, x3 = 0),
                   joint = TRUE)
s_joint$test$joint

# Directional marginal alternative: theta_x2 > 0.
s_greater <- summary(bb, transform = c(x2 = 1), null = 0,
                     alternative = "greater")
s_greater$test$p.value
```

The reported confidence intervals remain two-sided; only the test changes with `alternative`. The joint test above tests the vector of two restrictions and is not equivalent to requiring that both marginal p-values fall below the same significance level.

16.6 Newdata predictions with a reusable bootstrap

```
newdat <- data.frame(
  x1 = c(0.2, 0.5, 0.8),
  x2 = c(0, 1, 1),
  x3 = c(0, 0, 1)
)
predict(fit, newdata = newdat)
predict(fit)                          # identical to fitted(fit)

pred <- predict(fit, newdata = newdat,
                interval = "confidence", se.fit = TRUE,
                level = 0.95, bootstrap = bb)
pred$fit
pred$se.fit

# Reuse the same coefficient draws at another level.
predict(fit, newdata = newdat, interval = "confidence",
        level = 0.90, bootstrap = bb)
```

This gives conditional Gini predictions for the listed profiles, not predicted wages or incomes. No outcome column is required in `newdat`.

16.7 Automatic bootstrap inside prediction

```
predict(fit, newdata = newdat, se.fit = TRUE,
        interval = "confidence", R = 999L, seed = 42L)
```

This is convenient for one prediction request. For multiple requests, obtaining `bb <-`

`bboot.gir(fit, ...)` once and reusing it avoids repeated fitting and makes comparisons share the same bootstrap randomness.

16.8 A difference in predicted Gini indices

For two fixed profiles x_A, x_B, the difference is $(x_A - x_B)'\theta$. In the no-intercept worked model:

```
contrast_AB <- c(x1 = 0.8 - 0.2, x2 = 1 - 0, x3 = 1 - 0)
s_AB <- summary(bb, transform = contrast_AB, null = 0)
s_AB$test$table
```

Do not subtract the two separately computed prediction interval endpoints. The contrast calculation retains their dependence through the common coefficient draws. For a model with interactions, construct $x_A - x_B$ from its full second-stage model-matrix rows, not merely differences of raw data columns.

16.9 Supplied multipliers and exact reuse across bases

```
# Draw a single vector once, after generating one simulated dataset.
e <- rexp(nobs(fit), rate = 1)
w_star <- e / mean(e)

one_spline <- bboot.gir(
  fit, R = 1L, inference = FALSE,
  bootstrap_weights = w_star, normalize_weights = FALSE
)
one_spline$t[1L, ]

fit_legendre <- gir(
  y ~ 0 + x1 + x2 + x3, data = dat,
  continuous = "x1", discrete = c("x2", "x3"),
  basis = "legendre", kappa = 0.245,
  sieve_size_type = "dimension", sieve_size_reference = "cell",
  basis_bounds = list(x1 = c(0.1, 1)),
  first_stage_singular = "ridge", first_stage_tol = 1e-12,
  ridge = 1e-10, singular.ok = FALSE
)
one_legendre <- bboot.gir(
  fit_legendre, R = 1L, inference = FALSE,
  bootstrap_weights = w_star, normalize_weights = FALSE
)
```

Calling each basis variant with independently generated weights would not give paired bootstrap draws.

16.10 The oracle first stage

```
dat$oracle_midcdf <- dat$y^alpha
fit_oracle <- gir(y ~ 0 + x1 + x2 + x3, data = dat,
                  midcdf = oracle_midcdf, singular.ok = FALSE)
one_oracle <- bboot.gir(
  fit_oracle, R = 1L, inference = FALSE,
  first_stage_boot = "fixed",
  bootstrap_weights = w_star, normalize_weights = FALSE
```

```
)
```

This is the oracle calculation of the paper's simulations: the true conditional CDF is held fixed. It does not justify holding a data-estimated mid-CDF fixed in an empirical analysis.

16.11 A complete external first-stage callback example

Here a single unconditional mid-distribution is intentionally supplied to an intercept-only model. The callback re-estimates that same object under the combined weights, using an explicit loop only for clarity:

```
small <- data.frame(y = c(0, 1, 1, 2, 2, 3, 4, 5))
weighted_mid <- function(y, a) {
  vapply(y, function(yi) {
    sum(a * ((y < yi) + 0.5*(y == yi))) / sum(a)
  }, numeric(1))
}
small$m <- weighted_mid(small$y, rep(1, nrow(small)))
external_fit <- gir(y ~ 1, data = small, midcdf = m)
refit_external <- function(object, weights, replicate) {
  weighted_mid(object$estimation$y, weights)
}
external_bb <- bboot.gir(external_fit, R = 999L, seed = 8L,
                         midcdf_boot = refit_external)
summary(external_bb)
```

The callback uses the actual retained sample, not a separately cleaned data vector. A general external conditional estimator needs its own weighted refitting implementation; the simple loop above is not a replacement for it.

16.12 Missing prediction values

```
nd_na <- newdat
nd_na$x1[2L] <- NA_real_
predict(fit, nd_na)                          # middle entry is NA
predict(fit, nd_na, na.action = na.omit)     # two entries
predict(fit, nd_na, na.action = na.exclude) # three entries, middle NA
# predict(fit, nd_na, na.action = na.fail)  # intentional error
```

For uncertainty, the same row policy applies to the interval matrix and SEs.

17 Relation to existing R interfaces

`quantreg::boot.rq`. The documented function is a lower-level regression bootstrap engine taking `x`, `y`, `tau`, and `R`, with a matrix of coefficient draws returned for subsequent analysis. Its `bsmethod="wxy"` uses unit exponential weights [4]. The GIR package retains the regression-specific resampling logic, replicate count, and access to raw draws, but accepts a fitted model to preserve the two-stage specification. It adds the paper's basic intervals and an explicit smooth-transformation/test interface. It does not call a quantile regression engine or import `quantreg` at runtime.

`boot::boot`. The generic `boot()` interface takes data and a statistic and can pass extra arguments through `...`; its familiar `t0`/`t`/`R` layout motivates the output of `bboot.gir()` [6]. However, `stype="w"` only tells `boot()` that a statistic's second argument represents weights. It does not, by itself, request iid exponential Bayesian multipliers. A generic wrapper is possible, but the dedicated engine makes the required weight law, common two-stage weighting, fixed basis, tie handling, and cached calculation explicit. The class returned by `bboot.gir()` is not a `boot` object, and `boot.ci()` is not the supported extraction method.

`predict.rq` and `predict.lm`. The fitted-object/newdata design, `interval`, `level`, and `na.action` choices follow familiar regression conventions; the `se.fit` list form is especially close to `predict.lm` [5, 7]. The statistical target and uncertainty calculation differ. `predict.rq` has several interval approaches, including percentile bootstrap prediction-function intervals, whereas this version of the GIR package implements only Algorithm 2's basic logic. Unlike `predict.lm`, `predict.gir()` does not add an individual-outcome noise term or provide a residual scale and Student degrees of freedom. There is only one prediction scale here: the linear Gini index.

18 Using the package in a Monte Carlo study

18.1 Three different quantities called standard deviation or bias

For a real dataset, `bb$std.error` estimates sampling uncertainty from many bootstrap draws conditional on that dataset. Its `bias` is the bootstrap mean minus the fitted coefficient.

Across Monte Carlo datasets, a simulation study instead computes

$$\widehat{\mathrm{Bias}}_{\mathrm{MC}} = \frac{1}{S}\sum_{r=1}^{S}\widehat{\theta}^{(r)} - \theta_0, \quad \widehat{\mathrm{SD}}_{\mathrm{MC},j} = \left\{\frac{1}{S-1}\sum_{r=1}^{S}(\widehat{\theta}_j^{(r)} - \bar{\theta}_j)^2\right\}^{1/2}.$$

These are not `bb$bias` or a within-dataset bootstrap SE; each quantity should be compared with its own counterpart.

18.2 Warp-speed coverage

The warp-speed method [2] uses one bootstrap draw per Monte Carlo dataset and pools

$$\Delta_{rj} = \widehat{\theta}_j^{*(r)} - \widehat{\theta}_j^{(r)} \quad (r = 1, \ldots, S).$$

Let q_j^- and q_j^+ be the pooled type-7 quantiles at $\alpha/2$ and $1-\alpha/2$. The dataset-specific intervals are

$$[\widehat{\theta}_j^{(r)} - q_j^+,\ \widehat{\theta}_j^{(r)} - q_j^-],$$

and coverage is the fraction containing the known truth. This pooling is distinct from the many-draw conditional bootstrap of Algorithm 2.

In such a study, obtain each dataset's point estimate with `gir()` and its single draw with `bboot.gir()` using `R=1` and `inference=FALSE`. When several estimators are compared on the same dataset, supply the same multiplier vector to each of them through `bootstrap_weights`, so that their draws are paired. Running $B = 999$ draws inside every Monte Carlo dataset is a different experiment.

For already assembled $S \times p$ matrices `theta_hat` and `theta_star`, the aggregation is:

```
alpha_level <- 1 - confidence_level
delta <- theta_star - theta_hat
q_lo <- apply(delta, 2L, quantile, probs = alpha_level/2,
              names = FALSE, type = 7L)
q_hi <- apply(delta, 2L, quantile, probs = 1-alpha_level/2,
              names = FALSE, type = 7L)
ci_lo <- sweep(theta_hat, 2L, q_hi, "-")
ci_hi <- sweep(theta_hat, 2L, q_lo, "-")
truth <- matrix(rep(true_theta, each = nrow(theta_hat)),
                nrow = nrow(theta_hat))
coverage <- colMeans(ci_lo <= truth & truth <= ci_hi)
mc_bias <- colMeans(theta_hat) - true_theta
mc_sd <- apply(theta_hat, 2L, sd)
```

19 Mathematical validation

The source tests include independent finite-sample checks tied to the paper:

- `test-math-algorithms.R` implements Algorithms 1 and 2 directly with QR-based threshold regressions. It checks full-rank Legendre and spline tensor bases, unequal discrete cells, ties and zeros, and paired bootstrap refits using the same multipliers in both stages.
- `test-math-gini-identity.R` checks Numerical Equivalence using pairwise absolute differences. It verifies the empirical group Gini, without the optional finite-sample correction, and its weighted counterpart in saturated bootstrap draws.
- `test-math-invariance.R` checks Scale Invariance for positive outcome rescaling, and row-order invariance when multiplier rows move with the observations. Coefficients, predictions, goodness of fit and paired bootstrap inference are compared.
- `test-math-r-squared.R` checks Algorithm 3, the weighted variation decomposition, and nested second-stage comparisons with the first stage held fixed. Zero benchmark variation is undefined; a model without an intercept can have negative GIR-R^2.

These fixtures use no ridge or clipping and run without network access or the draft paper. The inference arithmetic checks use 40 deterministic multiplier columns and confidence level 0.5; this is a small software test, not a coverage experiment or an applied-inference recommendation. General comparisons use tolerance 10^{-10}, and small pairwise identities use 10^{-12}. The reference calculations do not call GIR's internal basis or fitting helpers. The separate cluster and ridge tests verify implementation extensions without establishing their theoretical validity.

From the source directory, run `testthat::test_local(filter="math-")` for these checks, or `testthat::test_local()` for the full suite. They also run under `R CMD check`. The installed help topic `help("GIR-validation", package="GIR")` provides the detailed mapping and qualifications. Passing the tests does not prove asymptotic results or nominal confidence-interval coverage; the simulation framework above is a separate validation exercise.

Additional checks in `test-empty-samples.R` cover empty samples, retained weights, custom missing-value actions, alignment and omission metadata, and empty prediction/interval shapes. The point-estimator and bootstrap tests check the complete set of expected warnings and require other calls to run without warnings. Run `testthat::test_local(filter="empty-samples")` to isolate the sample-handling tests.

20 Installation and development

Before the first CRAN release, installation from GitHub requires access to the private repository:

```
# install.packages("remotes")
remotes::install_github("uptoaparameter/GIR", build_vignettes = TRUE)
library(GIR)
?gir
?bboot.gir
?predict.gir
```

The help files in `man/` are generated from the roxygen comments in `R/`; after editing those comments, run `roxygen2::roxygenize()` in the package directory. Run `testthat::test_local()` for the tests. Full release checks build the source package and run `R CMD check --as-cran`, including this vignette and the function reference. Maintainer instructions are in `tools/README.md` in the repository.

Open the installed guide with `vignette("GIR-guide", package = "GIR")` and obtain the package and paper citations with `citation("GIR")`.

The fit and bootstrap objects retain resolved estimation data, even when `model=FALSE`, `x=FALSE`, and `y=FALSE` omit corresponding public slots. Those flags do not anonymize an object or remove its internal snapshot. Do not publish fitted objects containing confidential data without a separate review of their contents.

21 Current limitations

- Bootstrap inference uses iid or cluster-level exponential multipliers; no complex-survey or time-series resampling scheme is implemented, and the paper's theorem covers the iid case.
- Prediction intervals are pointwise confidence intervals for the linear Gini index, not individual-outcome intervals or simultaneous bands.
- Bootstrap coefficient and prediction inference require a full-rank second-stage specification. Nonregular transformations require separate methods; a numerical result alone does not establish validity.
- The paper's weak-convergence proof is not itself a proof of bootstrap variance moment consistency. Covariance-based inference requires the additional regularity explained above.
- Numerical clipping, regularization, and external first-stage methods do not automatically inherit the unmodified series estimator's theory.
- Full tensor products can become large when the number of continuous regressors increases.
- Discrete conditioning is fully saturated through joint cells, which can create sparse groups when many discrete variables or levels are used.
- Interior spline knots are currently equally spaced on the normalized support; custom or quantile-spaced knot locations are not yet exposed.
- The first-stage fitted function is not constrained to be monotone in the outcome threshold or to lie in $[0, 1]$, except for optional pointwise clipping at the observed evaluation pairs.

22 References

[1] Bhardwaj, V. and Díaz-Villarejo, D. (2026). *Gini Index Regression*. Working paper, Universidad Carlos III de Madrid. https://ddiazvil.com/assets/papers/JMP_GIR_DiazVillarejo.pdf.

[2] Giacomini, R., Politis, D. N. and White, H. (2013). A warp-speed method for conducting Monte Carlo experiments involving bootstrap estimators. *Econometric Theory*, 29(3), 567–589.

[3] R Core Team. *B-Spline Basis for Polynomial Splines:* `splines::bs`. R documentation for the recommended `splines` package.

[4] Koenker, R. and contributors. *Bootstrapping Quantile Regression:* `boot.rq`. R documentation for `quantreg`. https://CRAN.R-project.org/package=quantreg.

[5] Koenker, R. and contributors. *Quantile Regression Prediction:* `predict.rq`. R documentation for `quantreg`. https://CRAN.R-project.org/package=quantreg.

[6] Canty, A., Ripley, B. and contributors. *Bootstrap Resampling:* `boot`; and *Print a Summary of a Bootstrap Object:* `print.boot`. Official R documentation for the recommended `boot` package. https://stat.ethz.ch/R-manual/R-devel/library/boot/html/boot.html.

[7] R Core Team. *Predict Method for Linear Model Fits:* `predict.lm`. Official R documentation. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/predict.lm.html.

[8] Cheng, G. (2014, revised preprint). *Moment Consistency of the Exchangeably Weighted Bootstrap for Semiparametric M-Estimation*. arXiv:1109.4204v2. https://arxiv.org/abs/1109.4204.

23 See also

Related infrastructure and interfaces: `lm()`, `model.frame()`, `model.matrix()`, `splines::bs()`, `boot::boot()`, `quantreg::boot.rq()`, `quantreg::predict.rq()`, and `stats::predict.lm()`. These comparisons do not add runtime dependencies on `boot` or `quantreg`.