Package: marbounds (via r-universe)

May 19, 2026

Title Bounds on Causal Effects with Mixed Informative and Non-Informative Missingness

Version 0.1.0

Description Implements influence-function-based estimators for bounds on causal effects (ATE, composite ATE, separable direct effect) when outcomes are missing due to an unknown mixture of informative and non-informative missingness. Supports user-specified estimands, assumptions, sensitivity parameters, tipping point analysis, and multiplier bootstrap over a grid of parameters. Nuisance functions are estimated via SuperLearner with cross-fitting.

License MIT + file LICENSE

Encoding UTF-8

Imports SuperLearner, stats

Suggests testthat, ggplot2

RoxygenNote 7.3.3

Repository https://mrubinst757.r-universe.dev

Date/Publication 2026-03-19 19:42:52 UTC

RemoteUrl https://github.com/mrubinst757/marbounds

RemoteRef HEAD

RemoteSha 8de5a7666f9a41efdf8d45d3185fe713fe0fda66

Contents

`clip_probs` *Clip propensity and probability estimates away from 0 and 1*

Description

Clip propensity and probability estimates away from 0 and 1

Usage

```
clip_probs(p, eps = 1e-06)
```

Arguments

`p`	Numeric vector of probabilities.
`eps`	Small positive number (default 1e-6).

Value

Clipped vector.

`coef_general_lower_ate`
Coefficient vectors b such that bound = mean(b' phi)

Description

theta = (E(mu0), E(mu1), E(pi0), E(pi1), E(mu0*pi0), E(mu1*pi1)) = (m1_0, m1_1, m2_0, m2_1, m3_0, m3_1) Order of phi columns: phi_1_0, phi_1_1, phi_2_0, phi_2_1, phi_3_0, phi_3_1

Usage

```
coef_general_lower_ate()
```

Details

All bounds/estimands are linear: value = b

compute_bounds *Compute all requested bound/point estimates from influence matrix and parameters*

Description

Compute all requested bound/point estimates from influence matrix and parameters

Usage

```
compute_bounds(
  phi,
  estimand = c("ate", "psi1", "psi2"),
 assumption = c("general", "bounded_delta", "monotonicity_pos", "monotonicity_neg",
    "bounded_risk", "bounded_risk_unbounded_tau", "point_ate", "point_psi1",
    "point_psi2"),
  delta_0u = 1,
  delta_1u = 1,
  delta_0l = 0,
  delta_1l = 0,
  delta_0 = NULL,
  delta_1 = NULL,
  tau_0 = NULL,
  tau_1 = NULL,
  tau = NULL,
  mu0 = NULL,
  mu1 = NULL,
  pi0 = NULL,
  pi1 = NULL,
  smooth_approximation = FALSE,
  epsilon = 0.001
)
```

Arguments

phi	n x 6 influence matrix.
estimand	"ate", "psi1" (composite ATE), "psi2" (SDE).
assumption	"general", "bounded_delta", "monotonicity_pos", "monotonicity_neg", "bounded_risk", "point_ate", "point_psi1", "point_psi2".
delta_0u, delta_1u	Upper bounds on proportion informative missingness (for bounded/mono bounds).
delta_0l, delta_1l	Lower bounds (for Psi_1 bounds).
delta_0, delta_1	Point values (for point identification).
tau_0, tau_1	Bounded risk ratio params (for bounded_risk).

tau	Single sensitivity parameter for point_ate (risk ratio in both arms).
mu0, mu1	Outcome nuisance functions (n-vectors).
pi0, pi1	Missingness probability nuisance functions (n-vectors).
smooth_approximation	Logical; for psi2 bounded_delta, use smooth approximation (TRUE) or indicator method (FALSE).
epsilon	Smoothing parameter for psi2 smooth approximation.

Value

List with elements estimate, lower, upper (as appropriate), and optional se_* for asymptotic SEs.

`estimate_nuisance` *Estimate nuisance functions using SuperLearner with cross-fitting*

Description

Estimates e(X)=P(A=1|X), pi_0(X)=P(C=1|X,A=0), pi_1(X)=P(C=1|X,A=1), mu_0(X)=E(Y|X,A=0,C=0), mu_1(X)=E(Y|X,A=1,C=0). Uses V-fold cross-fitting: for each fold, fits nuisances on the training part and predicts on the held-out part.

Usage

```
estimate_nuisance(
  X,
  A,
  C,
  Y,
  V = 2,
  sl_lib_prop = "SL.glm",
  sl_lib_miss = "SL.glm",
  sl_lib_outcome = "SL.glm",
  stratify_mu = TRUE,
  family_Y = "gaussian",
  seed = NULL
)
```

Arguments

X	Matrix of covariates.
A	Treatment indicator (0/1).
C	Missingness indicator (1=missing, 0=observed).
Y	Outcome (NA or value when C=0); only used when C=0 for outcome models.
V	Number of cross-fitting folds (default 2; the calling function may use V=1 when a simple single-library SuperLearner such as SL.glm is used).

sl_lib_prop	Character vector of SuperLearner library for propensity e (default "SL.glm").
sl_lib_miss	Character vector of SuperLearner library for missingness pi_0, pi_1 (default "SL.glm").
sl_lib_outcome	Character vector of SuperLearner library for outcome mu_0, mu_1 (default "SL.glm").
stratify_mu	Logical; if TRUE (default), estimate separate outcome models mu_0 and mu_1 stratified by treatment A. If FALSE, estimate a single pooled model E(Y|X,C=0) and predict for both A=0 and A=1.
family_Y	Character; family for outcome model ("gaussian" or "binomial"). Default "gaussian".
seed	Optional seed for fold splits.

Value

List with components: e, pi0, pi1, mu0, mu1 (each length n), and fold_id (fold index per row).

`mar_bounds` *Bounds and point estimates for causal effects under mixed missingness*

Description

Implements the methods from the paper: bounds on ATE, composite ATE (Psi_1), and separable direct effect (Psi_2), with user-specified estimand, assumptions, and sensitivity parameters. Nuisance functions are estimated via SuperLearner with cross-fitting.

Usage

```
mar_bounds(
  data,
  Y,
  A,
  C,
  X,
  estimand = c("ate", "psi1", "psi2"),
 assumption = c("general", "bounded_delta", "monotonicity_pos", "monotonicity_neg",
    "bounded_risk", "bounded_risk_unbounded_tau", "point_ate", "point_psi1",
    "point_psi2"),
  delta_0u = 1,
  delta_1u = 1,
  delta_0l = 0,
  delta_1l = 0,
  delta_0 = NULL,
  delta_1 = NULL,
  tau_0 = NULL,
  tau_1 = NULL,
  tau = NULL,
```

```
  V = 2,
  sl_lib = "SL.glm",
  sl_lib_prop = NULL,
  sl_lib_miss = NULL,
  sl_lib_outcome = NULL,
  stratify_mu = TRUE,
  family_Y = NULL,
  smooth_approximation = FALSE,
  epsilon = 0.001,
  seed = NULL,
  param_grid = NULL,
  grid_bounds = c("both", "lower", "upper"),
  alpha = 0.05,
  B = 1000,
  multiplier = c("rademacher", "gaussian")
)
```

Arguments

`data`	A data.frame containing the analysis data.
`Y`	Character; column name of the outcome (numeric; use NA when missing).
`A`	Character; column name of treatment (0/1).
`C`	Character; column name of missingness indicator (1 = missing, 0 = observed).
`X`	Character vector; column names of covariates.
`estimand`	One of "ate" (average treatment effect), "psi1" (composite ATE), "psi2" (separable direct effect).
`assumption`	One of "general", "bounded_delta", "monotonicity_pos", "monotonicity_neg", "bounded_risk", "bounded_risk_unbounded_tau", "point_ate", "point_psi1", "point_psi2".

general No extra assumptions; widest bounds (ATE/Psi_1).

bounded_delta Proportion of informative missingness bounded by delta_0u, delta_1u (and delta_0l, delta_1l for Psi_1).

monotonicity_pos Informative missingness increases outcome risk; use with delta_0u, delta_1u.

monotonicity_neg Informative missingness decreases outcome risk.

bounded_risk Outcome risk ratio in informatively missing bounded by tau_0, tau_1, with an upper bound that uses the minimum of 1-mu and mu*(tau-1) (as in the paper).

bounded_risk_unbounded_tau Original bounded-risk formula without applying the minimum-based tau constraint (provides backward-compatible behavior).

point_ate Point identification under known tau and delta_0, delta_1 (Assumption known-risk + known-missingness).

point_psi1 Point identification of Psi_1 under known delta_0, delta_1.

point_psi2 Point identification of Psi_2 under known delta_0.

`delta_0u`, `delta_1u`
: Upper bounds on proportion of missingness that is informative (in [0,1]). Default 1.

`delta_0l`, `delta_1l`
: Lower bounds (for Psi_1 bounds). Default 0.

`delta_0`, `delta_1`
: Point values for point identification (when assumption is point_*).

`tau_0`, `tau_1`
: Bounded risk parameters (>= 1) for "bounded_risk". Single `tau` for "point_ate" (same in both arms).

`tau`
: Single sensitivity parameter for "point_ate" (risk ratio in both arms).

`V`
: Number of cross-fitting folds for nuisance estimation (default 2; uses 1 when `SL.glm` is the only library for all nuisance functions).

`sl_lib`
: A single SuperLearner library specification used for all nuisance models by default (propensity, missingness, outcome); defaults to `"SL.glm"`.

`sl_lib_prop`, `sl_lib_miss`, `sl_lib_outcome`
: Optional SuperLearner libraries for propensity, missingness, and outcome models. If `NULL`, they fall back to `sl_lib`.

`stratify_mu`
: Logical; if TRUE (default), estimate separate outcome models for A=0 and A=1. If FALSE, estimate a single pooled model E(Y|X,C=0).

`family_Y`
: Character; family for outcome model ("gaussian" or "binomial"). If NULL (default), automatically detects: "binomial" for binary Y (only 0/1 values), otherwise "gaussian". Set explicitly to override auto-detection.

`smooth_approximation`
: Logical; for Psi_2 bounds under bounded_delta, use smooth approximation (TRUE) or indicator function method (FALSE, default).

`epsilon`
: Smoothing parameter for Psi_2 smooth approximation (default 0.001). Only used when smooth_approximation = TRUE.

`seed`
: Optional seed for cross-fitting splits.

`param_grid`
: Optional list or data.frame describing a grid of sensitivity parameters for which to compute bounds and/or point estimates and confidence bands. For `estimand = "ate"`: `delta_0u`, `delta_1u`, `tau_0`, `tau_1`, `tau`, `delta_0`, `delta_1` (by assumption). For `estimand = "psi1"` (bounded_delta): `delta_0u`, `delta_1u`, `delta_0l`, `delta_1l`; (point_psi1): `delta_0`, `delta_1`. For `estimand = "psi2"` (point_psi2): `delta_0`. Column names may also be provided without underscores (e.g., `delta0u`).

`grid_bounds`
: Which bounds to compute over the grid: `"both"` (default), `"lower"`, or `"upper"`. For point-identified grids (point_ate, point_psi1, point_psi2) this is ignored and a single `estimate_grid` is returned.

`alpha`
: Significance level for pointwise and uniform (multiplier bootstrap) confidence intervals over the grid (default 0.05).

`B`
: Number of multiplier bootstrap replications for uniform bands over the grid (default 1000).

`multiplier`
: Multiplier distribution for the bootstrap: `"rademacher"` (default) or `"gaussian"`.

Value

A list with components: `naive` (naive estimate assuming MAR), `lower`, `upper` (for bounds), or `estimate` (for point ID), with `se_*` or `se` and optional `nuisance`, `phi` for downstream use. `result` contains a convenient dataframe summary: for scalar results, it includes the estimates and confidence intervals; for grid results (`param_grid` provided), it contains the combined grid output in long format with `bound_type` column for bounds. When `param_grid` is provided, the list also contains `lower_grid`, `upper_grid`, and/or `estimate_grid` as separate list objects with pointwise and uniform inference.

`multiplier_bootstrap_grid`

Multiplier bootstrap for simultaneous inference over a parameter grid

---

Description

For a grid of sensitivity parameters, each defining a linear functional b' theta (e.g. a bound or point estimate), computes point estimates and then uses the multiplier bootstrap to construct simultaneous confidence intervals or bands. Let psi_i = b' phi_i be the influence function for the functional. The bootstrap draws G_1,...,G_n iid (e.g. Rademacher or N(0,1)) and forms T* = sqrt(n) * mean(G_i * psi_i). Repeating B times approximates the distribution of sqrt(n)(thetahat - theta).

Usage

```
multiplier_bootstrap_grid(
  mar_result,
  param_grid,
  bound_spec = c("lower", "upper", "point"),
 assumption = c("bounded_delta", "monotonicity_pos", "monotonicity_neg", "bounded_risk",
    "point_ate"),
  B = 1000,
  alpha = 0.05,
  multiplier = c("rademacher", "gaussian")
)
```

Arguments

`mar_result`	Output from `mar_bounds()` containing `phi` and optionally `nuisance`.
`param_grid`	Data.frame or list; each row (or combination) gives a parameter set. Columns must include those needed for the coefficient vector (e.g. delta_0u, delta_1u, tau, etc.). Alternatively, a list of vectors with names `delta_0u`, `delta_1u`, etc., which will be expanded via `expand.grid`.
`bound_spec`	Character: "lower", "upper", or "point". For "lower"/"upper" the grid rows define the sensitivity parameters for that bound (e.g. delta_0u, delta_1u for monotonicity_pos). For "point" the grid gives (delta_0, delta_1, tau) for point_ate.
`assumption`	Same as in `mar_bounds`: "bounded_delta", "monotonicity_pos", "monotonicity_neg", "bounded_risk", "point_ate".

B	Number of multiplier bootstrap replications (default 1000).
alpha	Significance level for simultaneous CIs (default 0.05).
multiplier	"rademacher" (default) or "gaussian".

Value

List with grid (data.frame with columns from param_grid plus estimate, se, ci_lower, ci_upper, simultaneous_lower, simultaneous_upper), boot_replicates (B x nrow(grid) matrix of bootstrap T* values), and critical_value (estimated 1-alpha quantile of max over grid of |T*|).

---

phi_var *Empirical variance matrix of phi (for asymptotic variance of b' theta)*

Description

Empirical variance matrix of phi (for asymptotic variance of b' theta)

Usage

```
phi_var(phi)
```

Arguments

phi	n x 6 matrix.

Value

6 x 6 variance matrix.

---

psi_naive *Naive estimate of the ATE (Psi_tilde) assuming MAR*

Description

Returns the influence-function-based estimate and variance using the EIF for the naive estimand Psi_tilde = E[mu_1(X) - mu_0(X)], where mu_a(X) = E[Y | X, A = a, C = 0] (outcome mean among observed). Package convention: C = 0 denotes observed (non-missing), C = 1 denotes missing. The EIF is I(A=1,C=0)/(P(C=0|X,A=1)*P(A=1|X)) * (Y - mu_1(X)) - I(A=0,C=0)/(P(C=0|X,A=0)*P(A=0|X)) * (Y - mu_0(X)) + mu_1(X) - mu_0(X), with variance estimated by (1/n) * sample variance of the EIF.

Usage

```
psi_naive(data, Y, A, C, X, nuis = NULL)
```

Arguments

data	Data.frame.
Y, A, C, X	Column names as in mar_bounds.
nuis	Optional pre-computed nuisance list (e, pi0, pi1, mu0, mu1).

Value

List with estimate (EIF-based point estimate), var (variance estimate (1/n)*Var(EIF)), se (sqrt(var)), and optionally eif (length-n vector of EIF values).

---

theta_hat *Compute empirical mean of influence functions (point estimates of theta components)*

Description

Compute empirical mean of influence functions (point estimates of theta components)

Usage

```
theta_hat(phi)
```

Arguments

phi	n x 6 matrix from influence_functions().

Value

Vector of length 6: (m1_0, m1_1, m2_0, m2_1, m3_0, m3_1).

Index