

Package: balancing (via r-universe)

October 4, 2026

Title Optimization-Based Balancing Weights for Causal Inference

Version 0.0.0.9000

Description Calculates optimization-based balancing weights for causal inference, including entropy balancing, inverse probability tilting, the covariate balancing propensity score, energy balancing, characteristic function distance balancing, and stable balancing weights. Weights target covariate balance directly by solving a convex optimization problem, with support for binary, categorical, and continuous exposures across a range of causal estimands. A Rust core provides the numerical solvers.

License MIT + file LICENSE

URL <https://r-causal.github.io/balancing/>,
<https://github.com/r-causal/balancing>

BugReports <https://github.com/r-causal/balancing/issues>

Depends R (>= 4.3)

Imports causalgenerics (>= 0.1.0.9000), cli, deli, parallel, rlang,
S7, stats, tidyselect (>= 1.2.0), vctrs

Suggests halfmoon, knitr, MASS, mice, optweight, osqp, pkgload,
propensity (>= 0.1.0.9000), quadprog, rmarkdown, testthat (>= 3.0.0), WeightIt, withr

VignetteBuilder knitr

Remotes r-causal/causalgenerics, r-causal/deli, r-causal/propensity

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

SystemRequirements Cargo (Rust's package manager), rustc (>= 1.88),
GNU make, xz, cmake

Config/pak/sysreqs cmake make xz-utils libclang-dev
Repository <https://r-causal.r-universe.dev>
Date/Publication 2026-10-04 21:34:28 UTC
RemoteUrl <https://github.com/r-causal/balancing>
RemoteRef HEAD
RemoteSha 1b40fcb2779daa83d18ad45fb316c46a6f1ff444

Contents

balance	2
balance_method	4
balance_terms	5
balancing	7
balancing_options	8
bw	9
bw_cbps	11
bw_cfd	13
bw_energy	15
bw_entropy	18
bw_ipt	20
bw_sbw	22
estimating_equations	24
ipw.balancing	25
weights,balancing::balancing-method	36

Index	38
--------------	-----------

balance	<i>Estimate balancing weights</i>
---------	-----------------------------------

Description

balance() fits a balancing method to a data frame, returning weights that target covariate balance directly. The exposure and covariates are chosen with tidyselect, the method is one of the method specifications such as bw_entropy(), and the estimand and constraints control what balance the weights achieve.

Usage

```
balance(  
  .data,  
  .exposure,  
  .covariates,  
  method = bw_entropy(),  
  estimand = c("ate", "att", "atc", "ato"),  
  ...,
```

```

    constraints = NULL,
    exposure_type = c("auto", "binary", "categorical", "continuous"),
    .focal_level = NULL,
    sampling_weights = NULL
  )

```

Arguments

<code>.data</code>	A data frame.
<code>.exposure</code>	The exposure column, selected with data-masking. Exactly one column.
<code>.covariates</code>	The covariate columns, selected with tidyselect. At least one column, with no default.
<code>method</code>	A balance_method specification from one of the method constructors, such as bw_entropy() .
<code>estimand</code>	The target estimand: "ate", "att", "atc" (stored as "atu"), or "ato". Defaults to "ate".
<code>...</code>	Reserved; must be empty.
<code>constraints</code>	A balance_terms() specification, or NULL for the method default.
<code>exposure_type</code>	One of "auto" (the default), "binary", "categorical", or "continuous".
<code>.focal_level</code>	The focal exposure level for "att" and "atc". Inferred for a binary exposure; required for a categorical exposure.
<code>sampling_weights</code>	Sampling weights, given as a bare column name or an external numeric vector, or NULL.

Details

The exposure type is detected automatically and announced through an informational message, which `options(balancing.quiet = TRUE)` suppresses. The estimand vocabulary matches propensity: "atc" is accepted as a synonym for the untreated target and stored as "atu". "att" and "atc" reweight toward a focal exposure level, inferred for a binary exposure and required through `.focal_level` for a categorical exposure. Continuous exposures permit only "ate".

Constraints default to first-moment balance. Pass a [balance_terms\(\)](#) specification to balance higher moments, interactions, or quantiles, or to relax exact balance to a tolerance.

A factor covariate expands to one indicator per level, and those indicators sum to the constant every balancing method carries. One indicator per factor is therefore redundant with that constant and is dropped, with an informational alert naming the term. The dropped level is the last one, and balancing the levels that remain balances it too. The factor stays in `@covariates`, and `@balance_table` reports the surviving levels rather than the full set.

A fit can be interrupted between solver iterations, so a long solve stops at the next iteration rather than at the end of the fit. On Unix the poll reads R's interrupt flag directly and does not service R's event loop, so a [setTimeLimit\(\)](#) set around the call fires when the call returns rather than partway through the solve.

A `difftime` covariate balances as the number it stores, in the unit its own column declares. Nothing rescales it and nothing reinterprets the unit, so its constraints, its recipe, and its balance table

match those of the same durations supplied as bare numbers. A Date or POSIXt covariate balances the same way, as the number `as.numeric()` gives it: days since 1970-01-01 for a date, seconds since then for a date-time. Both date-time representations are read that way, so a POSIXlt column balances exactly as the POSIXct column holding the same instants does.

Value

A [balancing](#) object.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_entropy())
fit
weights(fit)
```

balance_method	<i>The balancing method specification classes</i>
----------------	---

Description

The method constructors (`bw_entropy()` and its siblings) return small S7 objects that carry tuning parameters and never touch data. They share an abstract hierarchy so that `balance()` can query a method's capabilities uniformly. `balance_method` is the abstract root. The estimating-equation family (entropy balancing, inverse probability tilting, the covariate balancing propensity score) subclasses `estimating_equation_method`; the quadratic-program family (energy, characteristic function distance, and stable balancing weights) subclasses `quadratic_program_method`. None of the abstract classes can be constructed directly.

Usage

```
balance_method(convergence_tolerance = NULL, max_iterations = NULL)

estimating_equation_method(convergence_tolerance = NULL, max_iterations = NULL)

quadratic_program_method(
  convergence_tolerance = NULL,
  max_iterations = NULL,
  weight_penalty = numeric(0),
  min_weight = numeric(0)
)
```

Arguments

convergence_tolerance	The solver convergence tolerance, or NULL to leave it to the solver. The value resolved for NULL differs by family: 1e-10 on the gradient for the estimating-equation methods, and 1e-8 as both the absolute and the relative tolerance for the quadratic-program methods.
max_iterations	The maximum solver iterations, or NULL to leave the cap to the solver. The value resolved for NULL is 1000 for the estimating-equation methods and 200000 for the quadratic-program methods.
weight_penalty	The L2 penalty on the weights.
min_weight	The smallest permitted weight.

Value

An abstract class object. Constructing a concrete subclass returns a balance_method.

balance_terms	<i>Balance constraints</i>
---------------	----------------------------

Description

balance_terms() records the set of covariate functions a balancing method should equate across exposure groups. It is passed to [balance\(\)](#) through constraints. The specification is data-free: the covariate expansion it describes is applied at fit time.

Usage

```
balance_terms(  
  moments = NULL,  
  interactions = FALSE,  
  quantiles = NULL,  
  tolerance = 0,  
  ...  
)
```

Arguments

moments	The highest covariate power to balance. A single whole number or a named integer vector; NULL (the default) resolves to first moments. For a continuous exposure each power is held at zero weighted correlation with the exposure instead.
interactions	Whether to add pairwise interactions of the base columns. These expand the constraint set the weights must balance, adding the pairwise products of the base columns to the covariate functions a fit constrains: equated across the exposure groups of a discrete exposure, and driven to zero correlation with a continuous one. They say nothing about causal interaction between two exposures, which is an effect rather than a constraint and is reported for a joint exposure by ipw() .

quantiles	Quantile probabilities in $(0, 1)$: a numeric vector applied to every continuous covariate, a named list of probabilities per covariate, or NULL for none.
tolerance	The per-constraint tolerance: a single non-negative number, or a named vector giving the tolerance per source covariate.
...	Reserved for future extensions; must be empty.

Details

The constraint set is built from four ingredients:

- **moments**: the highest power of each numeric covariate to balance. A scalar applies to every covariate; a named integer vector sets powers per covariate, with unnamed covariates defaulting to 1. Powers above 1 are ignored for binary indicator columns.
- **interactions**: when TRUE, all pairwise products of distinct base columns are added, excluding products of two indicators of the same factor.
- **quantiles**: probabilities in $(0, 1)$. Each probability adds an indicator column so that mean balance on the indicator is quantile balance on the covariate. A single probability vector applies to every continuous covariate; a named list sets probabilities per covariate. Quantile constraints apply to discrete exposures only.
- **tolerance**: the largest absolute standardized mean difference (discrete exposures) or exposure-covariate correlation (continuous exposures) permitted per constraint. A scalar applies to every covariate; a named vector sets tolerances per source covariate, and derived columns inherit their source covariate's tolerance. 0 requests exact balance. A positive value selects the inexact problem for entropy balancing and is the central tuning parameter for stable balancing weights.

For a continuous exposure there are no groups to equate, so a constraint column is instead held within tolerance of zero weighted correlation with the exposure. On the continuous energy path this is what moments and interactions request, and there the correlation is held exactly whatever tolerance says, for the reason `bw_energy()` records. The marginal distribution of the exposure and of the covariates is a separate matter, held by `distribution_moments` in `bw_energy()` and `bw_entropy()`; asking for correlation constraints does not add marginal rows, and raising `distribution_moments` adds no correlation constraint.

A factor covariate contributes one indicator per level rather than the reference coding a model formula would use. Those indicators sum to the constant every balancing method carries, so one of them is redundant and the expansion drops it, naming the term in an informational alert. The level dropped is the last one, and the constraints that remain balance it as well: with the other level proportions equated across exposure groups, the omitted one follows. The balance table reports the surviving levels.

Value

A `balance_terms` object.

Examples

```
# Balance means and variances of every numeric covariate.
balance_terms(moments = 2)
```

```
# Balance means with a relaxed tolerance.
balance_terms(tolerance = 0.05)
```

balancing	<i>A fitted balancing result</i>
-----------	----------------------------------

Description

`balancing` is the S7 object `balance()` returns. It carries the balancing weights, the resolved method and estimand, the covariate expansion recipe, a balance table, and the solver diagnostics. The raw data are not stored; the recipe carries what is needed to rebuild the constraint matrix.

Arguments

<code>weights</code>	The balancing weights, a <code>bw</code> vector.
<code>method</code>	The fitted <code>balance_method</code> specification.
<code>estimand</code>	The resolved estimand string.
<code>exposure</code>	The exposure column name.
<code>exposure_type</code>	The resolved exposure type.
<code>exposure_levels</code>	The exposure levels the fit weighted, as character strings in the order <code>levels(factor(exposure))</code> gives: a factor's declared order for a factor exposure, and the values sorted in their own type otherwise, so a numeric dose of 9 and 10 orders 9 first. Levels no observation takes are dropped and do not appear. The first element is the reference level every contrast in <code>ipw()</code> is measured against. A continuous exposure carries no levels, so the vector is empty.
<code>covariates</code>	The covariate column names that contributed at least one retained constraint column, in selection order. The constraint expansion drops constant and aliased columns, and a covariate can be selected without contributing any column at all, so this records what the fit constrained rather than what was requested; the request itself stays in call. A fit whose balance comes from its objective rather than from constraints, such as energy or kernel balancing with no moment constraints, therefore records no covariates even though its objective reads every selected one.
<code>focal_level</code>	The focal exposure level for "att" and "atc", or NULL. This is the fitted object's property, set from the <code>.focal_level</code> argument of <code>balance()</code> .
<code>n</code>	The number of observations.
<code>constraints</code>	The resolved <code>balance_terms</code> specification, or NULL.
<code>recipe</code>	The covariate expansion recipe, a list of per-column records.
<code>balance_table</code>	The achieved balance, one row per constraint term.
<code>duals</code>	Solver dual variables for diagnostics, or NULL.

coefficients	The fitted coefficients or dual variables, or NULL.
converged	Whether the solver met its convergence criterion.
iterations	The solver iteration count. An energy fit that could not reach its tolerance re-solves at a reachable one, and when that re-solve converges this sums the original and the fallback solve, so it can exceed the requested <code>max_iterations</code> . When the re-solve does not converge the fit reports the original solve alone, so the count stays within the cap. A continuous energy or stable balancing fit with a positive tolerance refines the bound it hands the solver over several passes, each a solve of its own, and this sums every one of them. See <code>bw_energy()</code> and <code>bw_sbw()</code> for the fuller account.
objective	The solved objective value.
solver_status	The solver that produced the result.
estimating_equations	The <code>balancing_estimating_equations</code> container, or NULL.
vcov	The covariance of the fit's own weight parameters, a p by p matrix named for them, or NULL. A covariance for those parameters comes from the stacked system an <code>ipw()</code> result assembles, so <code>balance()</code> leaves this empty and the result fills it in on the copy of the fit it stores, where <code>stats::vcov()</code> reads it back.
sampling_weights	The sampling weights, or NULL.
call	The originating call.

Value

A balancing object.

balancing_options	<i>Package options</i>
-------------------	------------------------

Description

`balancing` reads a handful of global options, set with `options()`, that tune how a fit runs without changing any function's signature.

Details

- `balancing.quiet`: when TRUE, suppresses the informational alerts `balance()` prints, such as the detected exposure type. Defaults to FALSE.
- `balancing.threads`: the number of worker threads the Rust core may use. When unset, the count is resolved automatically from the physical core count, capped by `OMP_THREAD_LIMIT` and `OMP_NUM_THREADS`, and forced to two under R CMD check. The count is decided in R and handed to the core, which sizes its worker pool from it: `RAYON_NUM_THREADS` is never read, so setting that environment variable changes nothing about how a fit runs.

- `balancing.entropy_solver`: the solver for the exact entropy problem, one of "newton" (the default), "lbfgs", or "lbfgs_then_newton". Newton is the only solver that drives the estimating equations to machine precision; the alternatives trade some precision for speed on large problems. When the option is unset and the Newton solve stops short of its tolerance or returns weights that are not finite, the fit retries once with "lbfgs_then_newton", whose Newton polish restores that precision, and announces the retry. Setting the option pins the solver and disables the retry, so a fit pinned to "newton" reports the failed solve as it stands.
- `balancing.qp_backend`: the quadratic-program backend for the CFD and stable-balancing-weights methods, one of "auto" (the default), "osqp", or "clarabel". Under "auto", the default solver runs first and the fit re-solves with the interior-point backend on a primal-infeasibility certificate; "osqp" disables that fallback and "clarabel" uses the interior-point backend directly. Energy balancing and the CFD energy kernel assemble an indefinite quadratic form, which the interior-point backend refuses, so they always solve through "osqp" and announce a "clarabel" pin as ignored.

Results are deterministic across thread counts: the same inputs produce identical weights at any `balancing.threads` value.

Examples

```
# Silence the exposure-type alert for a single fit.
withr::with_options(list(balancing.quiet = TRUE), {
  n <- 100
  x1 <- rnorm(n)
  df <- data.frame(exposure = rbinom(n, 1, plogis(x1)), x1 = x1)
  balance(df, exposure, x1, method = bw_entropy())
})
```

 bw

Balancing weight vectors

Description

bw objects are numeric vectors carrying the estimand and a set of balancing weights targets. They are the weight vector stored on a [balancing](#) result and returned by its `weights()` method.

bw is a sibling of `propensity::psw`: both inherit the `causal_wts` class, so `causalgenerics::is_causal_wt()` and `causalgenerics::estimand()` work on either.

Usage

```
new_bw(x = double(), estimand = NULL, ...)
```

```
bw(x = double(), estimand = NULL)
```

```
as_bw(x, estimand = NULL)
```

```
is_bw(x)
```

Arguments

x	A numeric vector of weights for <code>bw()</code> and <code>new_bw()</code> , or an object to test or coerce for <code>is_bw()</code> and <code>as_bw()</code> .
estimand	A single string naming the target estimand, or NULL.
...	Additional attributes stored on the object (developer use only).

Details

Constructors:

- `bw()` is the user-facing constructor. It coerces `x` to double before building the object.
- `new_bw()` is the low-level constructor for developers. It assumes `x` is already a double vector.
- `as_bw()` coerces an existing numeric vector to a `bw` object.

Queries:

- `is_bw()` tests whether an object is a `bw` vector.
- `causalgenerics::estimand()` reads the target estimand.

Combining:

Arithmetic preserves the class and estimand, so normalizing weights keeps the metadata. Combining `bw` vectors with matching estimands preserves the class; combining vectors with different estimands, or a `bw` with a `propensity::psw`, warns and falls back to a plain double vector.

Value

- `new_bw()`, `bw()`, `as_bw()`: a `bw` vector.
- `is_bw()`: a single logical value.

Examples

```
w <- bw(c(0.5, 1, 1.5), estimand = "ate")
w
is_bw(w)
estimand(w)

# Arithmetic preserves the class.
w / sum(w)
```

bw_cbps

*Covariate balancing propensity score***Description**

`bw_cbps()` specifies the covariate balancing propensity score for `balance()`. A propensity model is fit so that its parameters satisfy covariate balancing moment conditions rather than the maximum-likelihood score alone. In the just-identified form the moment conditions equal the parameter count, so balance on the requested moments is exact by construction and the weights solve smooth estimating equations. The covariate balancing propensity score supports binary, categorical, and continuous exposures, and is the only method that supports the overlap estimand "ato" for a binary exposure.

Usage

```
bw_cbps(
  ...,
  over_identified = FALSE,
  two_step = TRUE,
  link = c("logit", "probit", "cloglog"),
  convergence_tolerance = 1e-10,
  max_iterations = NULL
)
```

Arguments

<code>...</code>	Reserved for future extensions; must be empty. Tuning parameters must be passed by name.
<code>over_identified</code>	Whether to add the response-residual moments and minimize the generalized-method-of-moments criterion. FALSE fits the just-identified form, whose balance is exact. Binary exposures only; ignored, with a warning, for a categorical or continuous exposure.
<code>two_step</code>	Whether to use the two-step weighting matrix for the over-identified criterion, rather than the continuously updating criterion. Ignored, with a warning, whenever the fit is not over-identified.
<code>link</code>	The propensity link, one of "logit", "probit", or "cloglog". Binary and categorical exposures both fit a propensity model and consume it. A continuous exposure fits none, so the setting is ignored, with a warning, there.
<code>convergence_tolerance</code>	The solver convergence tolerance. 1e-10 is both this argument's default and the value the solver resolves for NULL.
<code>max_iterations</code>	The maximum solver iterations, or NULL for the resolved default of 1000.

Details

For a binary exposure the just-identified fit weights a unit by a function of its modeled propensity determined by the estimand: the inverse propensity for the average treatment effect, the inverse odds for the average treatment effect on the treated, and the overlap factor for the overlap estimand. With mean balance and the logit link the just-identified average-treatment- effect-on-the-treated fit solves the same treated-target moment conditions as entropy balancing and inverse probability tilting, so the three methods produce the same weights.

Setting `over_identified = TRUE` stacks the response-residual moments $\sum(s * (t - p) * x)$ onto the balancing conditions and minimizes a generalized-method-of-moments criterion. Those moments are the propensity model's own score only under the canonical logit link. Under a probit or complementary log-log link they stay a valid moment condition, since the response residual has mean zero at the true parameters whatever the link, but they are not that model's score and the fit is not a likelihood-augmented one. Balance is then approximate, the fit records the criterion value on its objective, and it supplies no estimating equations. The recorded criterion carries no units from the sampling weights: the mean moment and the moment covariance the weighting matrix inverts are each divided by the average sampling weight, so the same design expressed in survey-expansion units reports the same criterion and meets `convergence_tolerance` at the same fit. A converged verdict on that criterion does not always mean the gradient reached `convergence_tolerance`: the solver also certifies convergence when a full Newton step's predicted decrease falls at or below the objective's own floating-point resolution, which is the numerical minimum whatever the gradient reads. A squared criterion reaches that floor with a gradient near the square root of the arithmetic's precision, so this is the ordinary outcome rather than an exception. That criterion is defined for a binary exposure alone: a categorical or continuous exposure has no over-identified form, so the request is warned and ignored and the fit balances its moment conditions exactly. `two_step` selects the two-step weighting matrix for that criterion; it has no effect on a fit that is not over-identified and is warned and ignored there. Every just-identified discrete fit, the overlap estimand included, supplies estimating equations; only the binary over-identified form and a continuous exposure do not.

For a continuous exposure the covariate balancing conditions require the weighted exposure mean to match the sample mean and the weighted covariance between the exposure and every covariate to vanish. The weights that meet these conditions with the least departure from uniformity are the minimum-divergence exponential tilt, the same reweighting the continuous form of entropy balancing uses. This is the nonparametric reading of covariate balancing for a continuous exposure and departs from the parametric generalized propensity score of the Fong, Hazlett, and Imai reference, which derives the weights from a Gaussian density ratio and can be unstable; the two share the balancing conditions but not the weight family.

Value

A `bw_cbps` specification, a [balance_method](#).

References

- Imai, K. and Ratkovic, M. (2014). Covariate balancing propensity score. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1), 243-263.
- Fong, C., Hazlett, C., and Imai, K. (2018). Covariate balancing propensity score for a continuous treatment: Application to the efficacy of political advertisements. *The Annals of Applied Statistics*, 12(1), 156-177.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_cbps())
fit
```

bw_cfd

Characteristic function distance balancing

Description

`bw_cfd()` specifies characteristic function distance balancing, also called kernel balancing, for `balance()`. The weights minimize a kernel measure of the distance between the reweighted exposure groups and a target sample, following Wong and Chan. Every kernel but "energy" is positive semidefinite, so the objective is a convex quadratic program; the energy kernel is conditionally positive semidefinite alone, so its quadratic form is indefinite and the solve routes to the ADMM backend. Either way the reweighting improves multivariate covariate balance without positing a propensity model. Characteristic function distance balancing supports binary and categorical exposures.

Usage

```
bw_cfd(
  ...,
  kernel = c("gaussian", "matern", "laplace", "t", "energy"),
  smoothness = 1.5,
  degrees_of_freedom = 5,
  simulation_draws = 5000,
  improved = TRUE,
  weight_penalty = 1e-04,
  min_weight = 1e-08,
  convergence_tolerance = NULL,
  max_iterations = NULL
)
```

Arguments

...	Reserved for future extensions; must be empty. Tuning parameters must be passed by name.
kernel	The covariate kernel the objective is built on, one of "gaussian", "matern", "laplace", "t", or "energy".

smoothness	The Matern smoothness order, one of 0.5, 1.5, or 2.5. Used only by the Matern kernel.
degrees_of_freedom	The degrees of freedom of the t kernel's frequency distribution, a finite number greater than two. Used only by the "t" kernel. The t kernel approaches the Gaussian kernel as the degrees of freedom grow, so that limit is requested as kernel = "gaussian" rather than as an infinite degrees of freedom, which names no frequency distribution to draw from.
simulation_draws	The number of Monte Carlo frequency projections for the "t" kernel.
improved	Whether to add the between-group term of the improved variant for the average treatment effect with a discrete exposure.
weight_penalty	The L2 penalty on the weights, which stabilizes the quadratic program.
min_weight	The smallest permitted weight.
convergence_tolerance	The quadratic-program solver tolerance, or NULL for the resolved default of 1e-8, which the solver applies as both its absolute and its relative tolerance. Under the alternating-direction backend, which the other kernels take by default and which the "energy" kernel always takes because its quadratic form is indefinite, a tolerance below what the problem can reach spends the full iteration cap and then warns.
max_iterations	The maximum solver iterations, or NULL for the resolved default of 200000.

Details

The objective is the sum of each group's kernel distance to the target sample, built from a covariate kernel matrix rather than a pairwise distance matrix. kernel selects the kernel. The "gaussian", "laplace", and "matern" kernels are distance based: their bandwidth is the median of the pairwise covariate distances, so the fit is invariant to a uniform rescaling of the covariates. smoothness sets the Matern order, one of 0.5, 1.5, or 2.5. The "t" kernel approximates a heavy-tailed spectral kernel by Monte Carlo: simulation_draws frequency vectors are drawn from a multivariate t distribution with degrees_of_freedom degrees of freedom, on the R side under R's random number generator, so a fixed seed reproduces the weights. Raising the degrees of freedom takes the t kernel toward the Gaussian kernel, which is available exactly as kernel = "gaussian" and needs no Monte Carlo draws. The improved variant for the average treatment effect adds the between-group term of the kernel mean embedding, balancing the groups against one another as well as against the sample.

Setting kernel = "energy" uses the negative pairwise distance, which reproduces [bw_energy\(\)](#) with its "scaled_euclidean" distance on the same data and constraints. That equivalence is the reference point for the method, since the energy kernel has an external implementation while the other kernels do not.

Characteristic function distance balancing belongs to the quadratic-program family, which has no estimating equations, so a fit produces no estimating-equations container and the guarantee is objective-level rather than exact moment balance: without moment constraints the kernel objective drives balance, and with them the constraint rows hold within tolerance. The tolerance in [balance_terms\(\)](#) relaxes any added moment constraints rather than selecting an inexact solver. A tolerance supplied without moment constraints has nothing to relax, so it is warned and ignored.

What the optimization pins is the objective and the balance it achieves, not the individual weights. A kernel matrix built on a smooth kernel is ill-conditioned by construction, since nearby units contribute nearly proportional rows, and a quadratic program whose objective is flat along such directions has a set of solutions rather than one. Two quadratic-program backends given the same ill-conditioned problem agree on the objective to seven significant figures while individual unit weights differ materially; a Gaussian kernel at its median bandwidth is the case this was measured on. Treat the per-unit weights of a `bw_cfd()` fit as one member of an equivalence set rather than as a uniquely determined quantity: what the method determines is the reweighted distribution, so estimands and balance statistics computed from the weights are stable where a claim about a particular unit's weight is not.

Value

A `bw_cfd` specification, a [balance_method](#).

References

- Wong, R. K. W. and Chan, K. C. G. (2018). Kernel-based covariate functional balancing for observational studies. *Biometrika*, 105(1), 199-213.
- Huling, J. D. and Mak, S. (2024). Energy balancing of covariate distributions. *Journal of Causal Inference*, 12(1), 20220029.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_cfd())
fit
```

bw_energy

Energy balancing

Description

`bw_energy()` specifies energy balancing for [balance\(\)](#). The weights minimize the energy distance between the reweighted exposure groups and a target sample, subject to a simplex-type constraint set, so the reweighting improves multivariate covariate balance without positing a propensity model. Energy balancing supports binary, categorical, and continuous exposures.

Usage

```
bw_energy(
  ...,
  distance = c("scaled_euclidean", "mahalanobis", "euclidean"),
  improved = TRUE,
  weight_penalty = 1e-04,
  min_weight = 1e-08,
  distribution_moments = NULL,
  dimension_adjustment = TRUE,
  convergence_tolerance = 1e-06,
  max_iterations = NULL
)
```

Arguments

...	Reserved for future extensions; must be empty. Tuning parameters must be passed by name.
distance	The covariate distance definition the energy objective is built on, one of "scaled_euclidean" (each covariate centered at its weighted mean and divided by its weighted standard deviation), "mahalanobis", or "euclidean".
improved	Whether to add the between-group energy distance of the improved variant for the average treatment effect with a discrete exposure.
weight_penalty	The L2 penalty on the weights, which stabilizes the quadratic program. For a continuous exposure it is also what sets the residual exposure-covariate correlation, as the details section explains.
min_weight	The smallest permitted weight. The reported weights average one within each exposure group, so a floor approaching one leaves almost no room above it: the weight spread shrinks in proportion to the headroom $1 - \text{min_weight}$, and the fit degenerates smoothly into uniform weights and reports the balance uniform weights achieve. Nothing warns at that boundary, because the problem stays feasible and the solution is a real one. bw_sbw() , whose tolerances are hard constraints rather than an objective, refuses the same floor as infeasible instead.
distribution_moments	For a continuous exposure, the number of exposure and covariate marginal moments held equal to the sample under the base measure, or NULL for the first moments. This is the only route to those rows: the moments in balance_terms() asks for exposure-covariate correlation constraints instead and leaves the marginals here. Energy balancing carries no base weights, so the base measure is the sampling weights, and without them the marginals are held equal to the unweighted sample.
dimension_adjustment	For a continuous exposure, whether to weight the covariate energy distance by the covariate dimensionality adjustment.
convergence_tolerance	The quadratic-program solver tolerance, which the solver applies as both its absolute and its relative tolerance, or NULL for the family default of $1e-8$. Energy

balancing defaults to $1e-6$ rather than to that family default because its quadratic form is indefinite: on a small sample the alternating-direction residual floors above $1e-8$, and a run that keeps going past that floor walks away from the optimum instead of stalling at it. The energy objective always solves through the alternating-direction backend, whatever `balancing.qp_backend` names, so there is no backend to choose here: a tolerance below what the problem can reach spends the full iteration cap, then warns and reports the iterate of a re-solve at a tolerance the problem does reach, provided that re-solve converges within the same `max_iterations`. When it does not, the fit reports the iterate of the original solve.

`max_iterations` The maximum solver iterations, or NULL for the resolved default of 200000. The re-solve above is given the same cap, and when it converges the reported `@iterations` sums the two solves, so an energy fit that could not reach its tolerance can report more iterations than this. The refinement passes of a continuous fit with a positive tolerance are summed the same way, each pass being a solve of its own.

Details

For a binary or categorical exposure the objective is the sum of each group's energy distance to the target sample. The improved variant for the average treatment effect adds the between-group energy distance, which balances the groups against one another as well as against the sample. A focal estimand reweights the non-focal groups toward the focal group, whose units keep their base weight. The energy distance is built from a pairwise covariate distance matrix; `distance` selects how that matrix is formed.

For a continuous exposure the objective is the weighted distance covariance between the exposure and the covariates, following Huling, Greifer, and Chen, plus the marginal energy distances of the weighted exposure and covariate distributions. `distribution_moments` sets how many exposure and covariate marginal moments are held equal to the sample under the base measure, and `dimension_adjustment` reweights the covariate energy distance by the covariate dimensionality.

The two knobs a continuous fit carries are separate. `moments` in `balance_terms()` adds a constraint that holds the weighted correlation of the exposure with each covariate power within its tolerance, which defaults to zero, and `distribution_moments` pins the marginal moments of the exposure and of the covariates. Neither sets the other: a fit that wants both asks for both. The correlation rows are held within the tolerance `balance_terms()` carries, and reaching that band takes more than one solve. The quadratic program bounds a linearized correlation whose exposure and covariate scales are fixed at the sample, and the spread of energy weights shrinks both weighted standard deviations, so a single solve at the requested band overshoots it: a band of 0.05 lands between 0.070 and 0.086 at 200 to 1000 observations. The fit therefore tightens the bound it hands the program and re-solves, up to eight passes, until the reported correlation sits inside the band. A band of 0.05 took two passes at 350 and at 1000 observations, so it costs about two solves against the one the same fit at exact balance takes, exact balance having nothing to tighten. A band the passes cannot reach is reported at its last iterate, and the balance warning judges it as it judges any other fit.

Without those rows the continuous objective targets distributional independence between the exposure and the covariates rather than zero correlations, and it does not drive the correlations to zero. A residual weighted correlation of roughly 0.1 to 0.3 is ordinary at a few hundred to a few thousand observations. What holds it up is `weight_penalty`, which trades that residual against effective sample size: at its default of $1e-4$ the penalty term is about three quarters of the objective at 1000

observations, leaving a largest correlation near 0.22 to 0.25 at an effective sample size near 71 percent, while a penalty of zero brings the correlation down to 0.05 to 0.07 and the effective sample size down to about 20 percent. Ask for `balance_terms(moments = 1)` to remove the correlation outright, at a cost in effective sample size of its own.

Energy balancing belongs to the quadratic-program family, which has no estimating equations, so a fit produces no estimating-equations container and the tolerance in `balance_terms()` relaxes the constraints a fit added rather than selecting an inexact solver. A tolerance supplied without those constraints has nothing to relax, so it is warned and ignored, and the balance table reports the tolerance the fit enforced, which is zero.

Value

An `bw_energy` specification, a [balance_method](#).

References

Huling, J. D. and Mak, S. (2024). Energy balancing of covariate distributions. *Journal of Causal Inference*, 12(1), 20220029.

Huling, J. D., Greifer, N., and Chen, G. (2024). Independence weights for causal inference with continuous treatments. *Journal of the American Statistical Association*, 119(546), 1657-1670.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_energy())
fit
```

bw_entropy

Entropy balancing

Description

`bw_entropy()` specifies entropy balancing for [balance\(\)](#). The weights minimize the Kullback-Leibler divergence from a set of base weights subject to the covariate constraints, so among all reweightings that achieve balance the solution stays as close as possible to the base weights. Entropy balancing supports binary, categorical, and continuous exposures.

Usage

```
bw_entropy(
  ...,
  base_weights = NULL,
  distribution_moments = NULL,
  convergence_tolerance = 1e-10,
  max_iterations = NULL
)
```

Arguments

...	Reserved for future extensions; must be empty. Tuning parameters must be passed by name.
base_weights	A numeric vector of base weights, one per observation, or NULL for uniform base weights. The estimated weights minimize $\sum(w * \log(w / \text{base_weights}))$.
distribution_moments	For continuous exposures, the number of exposure and covariate marginal moments held equal to the sample under the base measure, or NULL for the constraint moments. Raised automatically when smaller than the constraint moments. The base measure is the product of the sampling weights and base_weights, so without either the marginals are held equal to the unweighted sample.
convergence_tolerance	The solver convergence tolerance. What it measures depends on which problem is solved. The exact problem, chosen when every tolerance in <code>balance_terms()</code> is zero, measures the gradient sup norm and responds to this value across its range. A positive tolerance selects the inexact problem, solved by FISTA against the relative change in the loss; that criterion is the weaker of the two, so the value is tightened to at most $1e-14$ to hold the achieved balance inside the requested box, and anything above $1e-14$ is inert there. $1e-10$ is both this argument's default and the value the solver resolves for NULL.
max_iterations	The maximum solver iterations, or NULL for the resolved default of 1000. When the L-BFGS then Newton hybrid runs, either as the automatic retry of a Newton solve that came back short or because <code>options(balancing.entropy_solver = "lbfgs_then_newton")</code> asked for it, the cap applies to each phase separately and the reported iteration count is the sum of the two, so such a fit can report more iterations than the cap.

Details

For a binary exposure the average treatment effect reweights each exposure group to the pooled covariate means, and the average treatment effect on the treated reweights the control group to the treated covariate means while the treated group is left unweighted. Its reported weights are its base weights carried to the group's sampling-weighted total rather than the base weights themselves, so they are proportional to the base weights and constant base weights come back as ones whatever level they were set at. When every requested tolerance is zero the constraints hold exactly and the weights solve smooth estimating equations, which `balance()` records for the M-estimation variance in `ipw()`. A positive tolerance in `balance_terms()` selects the inexact problem, which balances each constraint to within the tolerance and does not produce estimating equations.

The exact problem is solved by Newton's method, which starts from the base measure and is the only solver that drives the estimating equations to machine precision. A flat or badly scaled constraint set can leave that cold start short of its tolerance, so a failed Newton solve is retried once with the L-BFGS-then-Newton hybrid, which reaches a neighborhood with L-BFGS before polishing it with Newton and so ends at the same precision. The retry announces itself, and `@solver_status` records the solver the returned fit came from. Pinning the `balancing.entropy_solver` option, described in [balancing_options](#), selects one solver and disables the retry.

Value

An `bw_entropy` specification, a [balance_method](#).

References

Hainmueller, J. (2012). Entropy balancing for causal effects: A multivariate reweighting method to produce balanced samples in observational studies. *Political Analysis*, 20(1), 25-46.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_entropy())
fit
```

bw_ipt

Inverse probability tilting

Description

`bw_ipt()` specifies inverse probability tilting for [balance\(\)](#). A propensity model is fit not by maximum likelihood but by a tilted moment condition that forces each treatment group's weighted covariate means to their estimand targets, so balance on the requested moments is exact by construction. Inverse probability tilting supports binary and categorical exposures.

Usage

```
bw_ipt(
  ...,
  link = c("logit", "probit", "cloglog"),
  convergence_tolerance = 1e-10,
  max_iterations = NULL
)
```

Arguments

... Reserved for future extensions; must be empty. Tuning parameters must be passed by name.

link The propensity link, one of "logit", "probit", or "cloglog".

convergence_tolerance The solver convergence tolerance on the tilting moment. $1e-10$ is both this argument's default and the value the solver resolves for NULL.

max_iterations The maximum solver iterations, or NULL for the resolved default of 1000.

Details

For each treatment level the propensity $p_i = G(x_i' \beta)$ is estimated so that the weighted covariate total matches the target population total. The average treatment effect tilts every level to the whole sample, weighting a unit by the inverse of its modeled propensity. A focal estimand tilts each non-focal level to the focal level, weighting a unit by $(1 - p_i) / p_i$, and leaves the focal units at weight one. With mean balance and the logit link the tilt solves the same treated-target problem as entropy balancing, so the two methods produce the same average-treatment-effect-on-the-treated weights for a binary exposure.

The weights solve smooth estimating equations regardless of the requested tolerance, which [balance\(\)](#) records for the M-estimation variance in [ipw\(\)](#).

Value

An bw_ipt specification, a [balance_method](#).

References

Graham, B. S., Pinto, C. C. de X., and Egel, D. (2012). Inverse probability tilting for moment condition models with missing data. *The Review of Economic Studies*, 79(3), 1053-1079.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(df, exposure, c(x1, x2), method = bw_ipt())
fit
```

bw_sbw

*Stable balancing weights***Description**

`bw_sbw()` specifies stable balancing weights for `balance()`. Among all weightings that hold each reweighted exposure group's covariate means inside a tolerance band, stable balancing weights select the one of least dispersion, following Zubizarreta. The default "l2" norm minimizes the sum of squared weights, so for a fixed per-group total it minimizes the weight variance. Stable balancing weights support binary, categorical, and continuous exposures.

Usage

```
bw_sbw(
  ...,
  norm = c("l2", "l1", "linf"),
  min_weight = 1e-08,
  convergence_tolerance = NULL,
  max_iterations = NULL
)
```

Arguments

<code>...</code>	Reserved for future extensions; must be empty. Tuning parameters must be passed by name.
<code>norm</code>	The weight-dispersion norm to minimize, one of "l2" (the sum of squared weights, minimum variance), "l1" (the sum of absolute deviations from one), or "linf" (the largest absolute deviation from one).
<code>min_weight</code>	The smallest permitted weight.
<code>convergence_tolerance</code>	The quadratic-program solver tolerance, or NULL for the resolved default of 1e-8, which the solver applies as both its absolute and its relative tolerance. Under the default backend, a tolerance below what the problem can reach spends the full iteration cap and then warns.
<code>max_iterations</code>	The maximum solver iterations, or NULL for the resolved default of 200000. A continuous fit with a positive tolerance refines the bound it hands the solver over several passes, each a solve given this same cap, and the reported <code>@iterations</code> sums them, so such a fit can report more iterations than this.

Details

The balance tolerance is the method's central tuning parameter. It is set through `constraints = balance_terms(tolerance = ...)` in `balance()`, not on the method spec, and it must be positive: with an exact (zero) tolerance the problem reduces to exact moment balance, which abandons the minimum-variance rationale and is prone to infeasibility, so a fit without a positive tolerance is refused. Given a positive tolerance the weights hold each group's weighted covariate means within

the band while minimizing the weight dispersion. For a discrete exposure no feasible reweighting has smaller dispersion than the fit returns.

A single scalar tolerance applies to every covariate; a named vector sets a tolerance per covariate. A covariate left at zero in a named vector demands exact balance on that covariate while the others are relaxed, the infeasibility-prone case, so a mixed specification is an explicit choice rather than a convenience.

For a discrete exposure the constraints default to first-moment balance; pass `balance_terms()` to balance higher moments, interactions, or quantiles, each inside its tolerance band. For a focal estimand the focal group keeps unit weight and the other groups are pulled to the focal group's covariate means.

For a continuous exposure the weighted exposure-covariate correlations are held within the tolerance. The quadratic program bounds a linearized correlation whose scales are fixed at the sample, so the fit tightens that internal bound over a few passes until the reported weighted correlation sits inside the requested band. The returned weights therefore minimize dispersion over the tightened internal band rather than over every weighting that meets the reported band, so the strict minimum-dispersion guarantee is stated for discrete exposures only.

Stable balancing weights belong to the quadratic-program family, which has no estimating equations, so a fit produces no estimating-equations container.

The norm argument selects how the weight dispersion is measured, always against the uniform baseline of one within each reweighted group. "l2" minimizes the sum of squared weights, so for a fixed per-group total it minimizes the weight variance. "l1" minimizes the sum of absolute deviations from one, which tends to leave many weights untouched and concentrate the reweighting on a few units. "linf" minimizes the single largest absolute deviation from one, which spreads the reweighting as evenly as the balance constraints allow. The "l1" and "linf" problems are linear programs solved through the same quadratic-program backends as "l2"; their solutions can be non-unique, so a fit reports the achieved dispersion rather than promising a unique weighting.

Value

An `bw_sbw` specification, a `balance_method`.

References

Zubizarreta, J. R. (2015). Stable weights that balance covariates for estimation with incomplete outcome data. *Journal of the American Statistical Association*, 110(511), 910-922.

Examples

```
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
df <- data.frame(
  exposure = rbinom(n, 1, plogis(0.5 * x1 - 0.5 * x2)),
  x1 = x1,
  x2 = x2
)
fit <- balance(
  df,
```

```

    exposure,
    c(x1, x2),
    method = bw_sbw(),
    constraints = balance_terms(tolerance = 0.05)
  )
  fit

```

estimating_equations *Extract the estimating-equations container*

Description

estimating_equations() returns the [balancing_estimating_equations](#) a fit produced, the pieces a stacked sandwich variance needs after balancing. It is available only for fits whose weights solve smooth estimating equations: the estimating-equation family (entropy balancing, inverse probability tilting, just-identified covariate balancing propensity score) with exact balance. Fits without estimating equations, such as any tolerance-relaxed or quadratic-program fit, raise `balancing_ipw_unsupported_error`.

Usage

```
estimating_equations(x, ...)
```

Arguments

<code>x</code>	A balancing result.
<code>...</code>	Ignored.

Value

A [balancing_estimating_equations](#) object.

Examples

```

n <- 200
x1 <- rnorm(n)
df <- data.frame(exposure = rbinom(n, 1, plogis(0.5 * x1)), x1 = x1)
fit <- balance(df, exposure, x1, method = bw_entropy())
estimating_equations(fit)

```

ipw.balancing

*Inverse probability weighting for a balancing fit***Description**

A **balancing** fit registers a method on `causalgenerics::ipw()`, so a set of balancing weights drives the same bring-your-own-model workflow as a propensity score fit. You supply the fit and a weighted outcome model, and `ipw()` returns causal effect estimates with standard errors that account for having estimated the weights.

Arguments

<code>wt_mod</code>	A balancing fit that produced the weights.
<code>outcome_mod</code>	A weighted outcome model of class <code>stats::glm()</code> or <code>stats::lm()</code> , fitted with the balancing weights and carrying the exposure among its predictors. It may adjust for covariates alongside the exposure. For a continuous exposure it is a marginal structural model whose every exposure-reading term reads the exposure alone.
<code>.data</code>	The data frame holding the exposure and outcome. If NULL, the values are taken from the outcome model frame. It carries the exposure column the fixed-exposure predictions are built from, so it has nothing to supply for a continuous exposure, which makes no such predictions, and is ignored there. It must hold the rows both models were fitted on, in the same order. Half of the stacked system reads it and half reads the fit: the counterfactual predictions, the subgroup indicators, and a focal estimand's standardization come from <code>.data</code>, while the weight equations and the outcome-model score stay in the fit's order. A frame holding the right rows in another order therefore leaves every effect estimate unchanged, since a weighted mean does not care in which order it is summed, and makes every standard error wrong. Each column <code>.data</code> and the outcome model frame name in common is compared value by value, and a disagreement raises <code>balancing_ipw_input_error</code> naming the column and the first row it disagrees at. Columns the outcome model never saw, the modifier <code>.by</code> names among them, are free.
<code>estimand</code>	The causal estimand. If NULL, the fit's estimand is used. As in <code>balance()</code> , <code>"atc"</code> is accepted as a synonym for <code>"atu"</code> . Supplying an estimand that disagrees with the fit raises <code>balancing_estimand_error</code> .
<code>conf_level</code>	The confidence level for the intervals. Default 0.95.
<code>effects</code>	The presentation mode the result records, either <code>"marginal"</code> (the default) or <code>"conditional"</code> . The marginal reading reports the population-averaged causal contrasts described above; the conditional reading reports the outcome model's coefficient surface. Where a result has both surfaces, both are computed whichever mode is named, since the stacked system is solved either way, so the argument settles which one the result presents and nothing else. <code>causalgenerics::as_marginal()</code> and <code>causalgenerics::as_conditional()</code> move such a result between the two

readings afterwards, a pooled result as much as an unpooled one, and the accessors take an effects argument of their own for a single call.

One shape of result has a single reading. When a continuous exposure enters `outcome_mod` through several design columns, as a polynomial or a spline basis does, no coefficient of that model is a causal effect: a curve has a different slope at every dose. Such a result therefore records the conditional reading, supports no other, and announces that it has done so. Naming effects = "conditional" builds the same result without the announcement, and naming effects = "marginal" raises `balancing_ipw_input_error`, since it asks for a reading the model has none of. Naming every reading is naming none of them, so a call that supplies more than one reading, as a wrapper forwarding an effects default of its own does, is announced rather than refused. How many readings arrived is all that is read, not which ones, so a default vector given in another order counts as naming nothing in the same way. Every later door into the marginal reading is shut as well: the accessors and `causalgenerics::as_marginal()` refuse it, and a pooled set of such results carries the refusal forward. Marginalizing a dose response over the observed doses is left to the `margineffects` package, applied to the conditional result: `avg_slopes()` for slopes, `avg_comparisons()` for contrasts, and `avg_predictions()` for causal dose-response functions; see <https://marginaleffects.com/chapters/interactions.html>.

The conditional reading reports the coefficients of the stored `outcome_mod` against the outcome block of the stacked sandwich, which is the block the wrapper around that model carries, so its standard errors account for having estimated the weights rather than treating them as fixed. The stored `wt_mod` is the balancing fit itself rather than a wrapper of one, since the weight block of the same stack is already its own: the stored copy carries that block, the covariance of the fit's weight parameters, and `stats::vcov()` on it reads the block back. The fit that went into the call is untouched, and reports no covariance of its own.

`.by` A modifier to report the effects within the levels of, given unquoted and selected with `tidyselect` out of `.data` where one was supplied and out of the outcome model frame otherwise. The default, `NULL`, is the absence of a request: the result then reports the whole-sample effects alone and names no subgroups at all. A selection reaching any number of columns other than one is refused, since the effects are reported within the levels of a single variable. The effect modification section below describes the rows a request adds and the configurations it refuses, and the joint exposure section describes why a declared crossing takes none.

`...` Ignored, for compatibility with the generic.

Details

The point estimates are the g-computation marginal means: the outcome model is predicted with the exposure fixed to each level and averaged over the estimand's target population. Those means are reported in their own right, one row per exposure level under the effect label "mean", and the effect measures follow as contrasts of them. For a binary outcome the method returns the risk difference (`rd`), the log risk ratio (`log(rr)`), and the log odds ratio (`log(or)`); for a continuous outcome it returns the difference in means (`diff`).

A binary and a categorical exposure are read by one rule. The reference level is the first of the fit's own levels: a factor's declared level order for a factor exposure, and the sorted values otherwise.

The estimates table carries a contrast column, placed after effect, naming the level a mean row belongs to and naming each contrast row as "<level> vs <reference>", so a binary result names its single comparison "1 vs 0" rather than leaving it unnamed. A K-level exposure therefore leads with K mean rows, in the fit's own level order with the reference level first, and follows them with one block of measures per non-reference level. A categorical exposure declared as a crossing of two treatments by `causalgenerics::joint_exposure()` replaces those level-against-reference rows with the surface described under Joint exposures below, which reports its cell means the same way.

A continuous exposure has no levels to contrast, so there is no pair of marginal means to difference. What the method reports instead is the dose response of a weighted marginal structural model: the balancing weights break the exposure-covariate association, and every coefficient of the outcome model that reads the exposure is an effect on the model's own link scale. The estimates table holds one row per such coefficient, read straight off the weighted fit, since nothing is standardized here.

An exposure entering through one design column, whether as a bare term or as a transformation of one, is the whole of the dose response, so its coefficient is that response's slope everywhere. Such a model keeps the single-row table it has always returned, with the eight columns of the shared contract and no contrast column, and the row is named for the link: slope for an identity link, whether the model arrives as a `stats::lm()` or as a gaussian `stats::glm()`; log(or) for a logit; and log(rr) for a log link.

An exposure entering through several columns, as in $y \sim \text{exposure} + I(\text{exposure}^2)$ or $y \sim \text{splines::ns}(\text{exposure}, 3)$, has no such row, since a curve has a different slope at every dose. The table then holds one row per column, gains a contrast column naming each row after the coefficient the fit names, and the scale word steps back to coef at an identity link. A logit still reports log(or) and a log link log(rr), because a coefficient of those models is a log ratio whatever column it multiplies. Another link raises `balancing_ipw_input_error` either way, since its coefficients are none of those things. A continuous fit targets the average treatment effect and nothing else, so an estimand supplied alongside it either agrees or raises `balancing_estimand_error`, as it does for any other fit.

The standard errors come from a stacked M-estimator that the deli package differentiates and sandwiches. The stacked parameter vector holds four blocks: the weight parameters, the outcome-model coefficients, the marginal means, one per exposure level, and the effect contrasts. Their estimating functions are, in the same order, the weight-parameter estimating equations the fit carries, re-evaluated at new weight parameters through the hooks the fit's `balancing_estimating_equations` container supplies; the outcome-model score, from `deli::ee_glm()`, carrying the balancing weights as the fit would have reported them at those parameters, per-group renormalization included; the marginal-mean equations, which predict the outcome model with the exposure fixed to each level and standardize over the estimand's target population; and one deterministic row per contrast, setting the contrast parameter equal to its formula in the two means. `deli::compute_sandwich()` differentiates that system at the fitted values and returns its empirical sandwich covariance.

A continuous exposure stacks the same system with the g-computation half removed. There are no fixed-exposure predictions to standardize and no contrasts to form, so the stack holds the weight parameters and the marginal structural model's coefficients alone, and each reported effect is already one of those coefficients rather than a parameter derived from them. A basis widens that stack by its own columns and changes nothing else.

Nothing is re-solved along the way. Every parameter enters at the value its own fit already found, and the stacked estimating functions are only re-evaluated around that point. Because each contrast is a parameter of the stack rather than a transformation applied afterward, its standard error is already on the diagonal of the joint covariance and no delta-method step stands between the sandwich and the reported effects. The joint covariance propagates the uncertainty from estimating

the weights into the effect standard errors, which a variance that treats the weights as fixed would understate.

The result reads through the accessors `causalgenerics` registers on the class, which every fitting package's results share: `stats::coef()` for the reported effects under their display labels, `stats::vcov()` for their covariance, `stats::confint()` for their intervals, `stats::nobs()` for the number of observations, and `stats::weights()` for the weights the outcome model was fitted with. The covariance those accessors read is the block of the stacked one belonging to the reported effects, which is on the estimates table rather than derived from it: the effect measures are transformations of the same pair of marginal means, so they covary, and the off-diagonals a caller combining two of them needs are not recoverable from the standard errors alone. The stored outcome model carries its own block of the same covariance, so `vcov()` on it reports the variance of its coefficients with the uncertainty from estimating the weights included, where a bare refit of the same weighted model treats the weights as fixed and understates it.

A row's display label is built from the identity columns of the estimates table, in the order the table carries them: the effect measure, then the contrast where the surface names one, then the group where it names one. A contrast names a categorical exposure's pair of levels, a continuous surface's basis coefficient, or a joint exposure's treatment; a group names the subgroup a .by request reports the row within, or the level a joint exposure holds the other treatment at. That one label names the row wherever a caller reads it, so the printed table, the names of `stats::coef()`, the dimnames of `stats::vcov()`, and the rownames of `stats::confint()` agree row for row with the estimates table. The stacked `fit$theta` and `fit$vcov` carry names of their own, which name blocks of the estimating-equation system rather than reported rows and are described under Value.

The sandwich covariance is a large-sample one, and how large a sample it takes differs by exposure. The binary risk-difference standard error is calibrated at a few hundred observations; the continuous slope's is anticonservative there. Over 500 draws its ratio of mean standard error to the standard deviation of the estimates was 0.836 at 300 observations and 0.942 at 1200. Read a continuous interval as the asymptotic statement it is, and prefer the bootstrap of the inference vignette at small sample sizes.

The method is available only for fits whose weights solve smooth estimating equations: the estimating-equation family (entropy balancing, inverse probability tilting, and the just-identified covariate balancing propensity score) with exact balance at a binary or categorical exposure, and entropy balancing with exact balance at a continuous one. Any other fit, including an entropy fit at a positive tolerance, an over-identified or quadratic-program fit, or a continuous fit from a method that solves no estimating equations, raises `balancing_ipw_unsupported_error` and points to the bootstrap workflow described in the inference vignette.

The outcome model must carry the same exposure levels the fit weighted. Fitting it on data that drop a level, or that carry one the fit never saw, raises `balancing_ipw_input_error`: the counterfactual predictions and the weights would then describe different exposures, and the resulting table would name a contrast it had not computed.

The outcome model must carry the exposure among its predictors, and may adjust for covariates alongside it, including in interactions with the exposure: $y \sim \text{exposure}$, $y \sim \text{exposure} + x_1 + x_2$, and $y \sim \text{exposure} * x_1$ are all supported. A categorical exposure enters as a factor, so its fixed-exposure designs come from the model's own contrasts. A model without an exposure term raises `balancing_ipw_input_error`, since its fixed-exposure predictions would all be the same prediction and every contrast it reported would be zero.

A continuous exposure narrows that by variable membership, since what it reports are coefficients of the model rather than contrasts of predictions from it. Every term reading the exposure

must read the exposure alone, however many design columns it expands to, so $y \sim \text{exposure}$, $y \sim \text{exposure} + x1$, $y \sim \text{exposure} + I(\text{exposure}^2)$, $y \sim \sin(\text{exposure})$, $y \sim \text{poly}(\text{exposure}, 2)$ and $y \sim \text{splines::ns}(\text{exposure}, 3)$ are all supported. A term reading a covariate alongside the exposure raises `balancing_ipw_input_error`, so $y \sim \text{exposure} * x1$, $y \sim \text{exposure} + \text{exposure}:x1$ and $y \sim I(\text{exposure} * x1)$ are refused: each contributes a coefficient that is a change in the dose response per unit of that covariate, so there is no one effect for a row to report and no covariate value a row could name it at. The check reads the model's terms rather than the text of its formula, so the mixing is caught however it is written.

An offset reaches the linear predictor without being a term, so it is checked on its own, and for every exposure type. An offset expression naming the exposure, written either into the formula or passed through the model's `offset` argument, raises `balancing_ipw_input_error`. An offset is held at its observed value while the exposure is fixed to each level, so a discrete exposure's marginal means would read one exposure in the design and another in the offset, and a continuous exposure's coefficient would be something other than the effect of a one-unit change. An exposure-free offset stays supported. That check is static, so it accepts any offset whose stored expression does not name the exposure: a precomputed vector under some other symbol, and equally a wrapper that forwards the offset through its dots, which records `. 1` in the fitted call. Keeping the exposure out of such an offset is the caller's to honor, and it matters most for a discrete exposure, where a laundered offset corrupts the fixed-exposure marginal means and can reverse the sign of the reported contrast rather than merely shifting a coefficient.

The exposure may be a factor, a character column, or an integer code, and a term that transforms it counts as carrying it. A character column becomes a factor in a model formula on its own, while an integer code is written `factor(exposure)` for a model with one parameter per level; left untransformed, an integer code enters as a slope in the codes and the marginal means are the g-computation means of that model rather than of a saturated one. A transformed exposure is not a column of the model frame, since the frame stores the transformation, so such a model is passed with `.data`.

Which population the marginal means are averaged over is part of the estimand, and matters as soon as the outcome model adjusts for anything. A marginal model is saturated in the exposure, one free parameter per exposure level, so absent an offset it predicts a single value per level, its marginal means are the weighted group means whatever link the family carries, and no choice of population can change them. An adjusted model predicts a value per unit, so the means are standardized over the estimand's target population: every unit for a pooled estimand, and the focal group's units for "att" or "atc". Sampling weights, where the fit has them, weight that average as well.

The link enters the outcome-model score, and it enters exactly: the bread is differentiated from the estimating functions themselves rather than read off an information-matrix formula, so a non-canonical link such as probit or cloglog is handled exactly rather than approximately, for an adjusted model as much as for a marginal one.

Two further conditions on the outcome model raise the same `balancing_ipw_input_error`. Its family must be binomial, quasibinomial, or gaussian, which includes a plain `stats::lm()`, since the reported effects are the contrasts derived for those families' marginal means. And it must have been fitted with the weights the fit produced, since the stacked variance differentiates the outcome-model score through those weights: the model's weights are compared against the fit's, per unit at a relative tolerance of $1e-6$. Those are the weights `weights(fit)` returns, which already carry the fit's sampling weights if it has any. Sampling weights compose multiplicatively onto the balancing weights and the stack holds them fixed, since they are a design quantity rather than an estimate.

Of the two binomial families, `stats::quasibinomial()` is the one to fit a binary outcome with

here, and the examples below use it: balancing weights are not counts, so `stats::binomial()` warns about non-integer successes at every fit. The two solve the same estimating equation, since they share the binomial variance function, and the dispersion the quasi family estimates never reaches the sandwich, which is built from the score alone. The results are therefore identical rather than merely close.

An offset is supported, written either as an `offset()` term in the outcome formula or passed through the model's `offset` argument, so long as it does not read the exposure. It is carried through both the outcome-model score and the fixed-exposure linear predictors, so the marginal means are the g-computation means with each unit's offset held at its observed value. That is the right treatment of a quantity the exposure does not move and the wrong treatment of one it does, which is why the exposure-reading case is refused above.

Value

An object of class `ipw`, an implementation of `causalgenerics::ipw()`. Alongside `estimand`, `wt_mod`, `outcome_mod`, the `estimates` table, and the `effects` field recording the presentation mode described above, the result carries two fields describing the variance:

- `se_method`, the string "mestimation", naming how the standard errors were computed.
- `fit`, the fitted variance system, a list of `theta`, the stacked parameter vector, and `vcov`, its sandwich covariance. Both are named by stacked block: `theta_w1` onward for the weight parameters, `beta_` followed by the design column name for the outcome-model coefficients, then the marginal means and one name per contrast. A binary exposure names its means `mu0` and `mu1` and its contrasts by measure alone; a categorical exposure names each mean `mu_` followed by its level and each contrast by measure and level, as `rd_b`. A continuous exposure carries neither block, since its effects are outcome-model coefficients; each of those coefficients is named for the label its estimates row carries, as `slope` or `coef I(exposure^2)`, in place of the `beta_` name the others keep. A `.by` request appends, after all of those, every subgroup's mean block in subgroup order, then every subgroup's contrast block in that same order, then one contrast block per non-reference subgroup against the reference one. Each of their names is the name of the block it repeats, suffixed with its group, as `mu0_sex = female` and `rd_sex = female vs sex = male`. A declared joint exposure keeps the mean block and replaces the contrast block, naming each of its own contrasts for the measure and the row, as `rd_a: 1 vs 0 e = 0`. The standard errors in estimates are `sqrt(diag(fit$vcov))` read at those effect names.

The `estimates` table carries the covariance of the reported effects as its `ipw_vcov` attribute, which is what `stats::vcov()` returns in the marginal reading. Both its `dimnames` are the display labels described above, as "rd b vs a sex = female". The stored `outcome_mod` is wrapped by `causalgenerics::new_ipw_model()`, which carries the outcome-model block of `fit$vcov` under the model's own coefficient names, so `vcov()` on it reports the joint-estimation variance. The stored `wt_mod` carries the leading block of the same covariance under the `theta_w` names above, which name the fit's own parameters on either route, so `vcov()` on it reports the covariance of the weight parameters. `stats::df.residual()` returns `NA_integer_`, since the stacked system is not a fit with residual degrees of freedom of its own.

`stats::nobs()` delegates to the stored outcome model, which counts the rows it was fitted on that carry a nonzero weight. A unit given no sampling weight is pinned at zero rather than dropped, so the weight vector stays the length of the data the fit saw while the outcome model counts one row fewer for each pinned unit, and `nobs()` on the result is then smaller than `length(weights(result))`.

Effect modification

`.by` names a modifier, and a result carrying one reports the rows it reports without a request, then the marginal means within each of the modifier's levels, then the effects within each of them, then each non-reference subgroup against the reference one. The estimates table gains a group column naming the subgroup each row was estimated in, placed after contrast, since a subgroup qualifies the whole comparison rather than one side of it. A block of means is never split by the contrasts written from it, which is what puts every subgroup's means ahead of every subgroup's contrasts rather than beside them. The whole-sample rows come first, under the group "overall"; a subgroup's rows are named "var = value", as "sex = female"; and a contrast of subgroups joins the two, as "sex = female vs sex = male". The reference subgroup is the modifier's first level, which for a factor is the first of its declared levels rather than the first in sorted order. A character modifier declares no levels, so it is read as a factor on the way in and its reference subgroup is its alphabetically first value, whatever order the values appear in. Declare the column a factor to measure the contrasts against some other subgroup.

A subgroup reports its marginal means and then the collapsible measures alone. For a binary outcome those measures are `rd` and `log(rr)`; a continuous outcome reports `diff`, the only measure it has. The log odds ratio stays among the whole-sample rows. An odds ratio is noncollapsible, so the odds ratio over a sample is not an average of the odds ratios within its subgroups and the difference of two of them is not the difference in effect it reads as; nothing in the whole-sample rows averages anything over subgroups, which is why they keep it. A contrast of two subgroups compares two effects and has no mean of its own, so the mean rows repeat over the subgroups themselves alone. A categorical exposure crosses its contrasts with the subgroups, so each subgroup block reports each contrast in the order the whole-sample block reports it.

A subgroup's marginal means are the g-computation means over that subgroup alone, standardized the way the whole-sample means are: over every unit of the subgroup for a pooled estimand, and over its focal units for "att" or "atc". Sampling weights weight that average as well.

Every row a request adds is a parameter of the same stacked system the whole-sample rows are parameters of. The blocks are appended after everything the ungrouped stack carries and nothing earlier reads a parameter of theirs, so the leading blocks are the ones an ungrouped fit produces and the whole-sample rows of a grouped result are the rows it reported. What that buys is the covariance: the subgroups share the weight parameters and the outcome model's coefficients, so their effects covary, and each contrast of subgroups is a parameter of the joint system rather than a difference of two separate fits, whose variance would have to be read as a sum.

A level of the modifier that no unit carries names an empty subgroup and is dropped rather than refused, which is what lets a modifier be subset without being recoded first. Four configurations are refused with `balancing_ipw_input_error`: a selection naming any number of columns other than one, a modifier that is neither a factor nor a character column, a modifier carrying missing values, and a modifier one of whose subgroups holds fewer than all of the exposure levels. That last one has no contrast to report in the subgroup that is short a level, and refitting either model does not supply a comparison the data do not hold. `.by` with a continuous exposure raises `balancing_ipw_unsupported_error`: what such a fit reports is the marginal structural model's own exposure coefficients rather than contrasts of standardized means, so there is no effect within a subgroup for the argument to name, and `.by` with a declared joint exposure raises it for the reason given under Joint exposures below. A modifier that reaches the argument through `.data` carries that frame's row-order requirement with it, described under `.data` above: the subgroups are built from the rows it holds while the weights stay in the fit's order.

An outcome model with no term reading both the exposure and the modifier raises `balancing_ipw_by_interaction_warning` and the result is still built. The subgroup effects are g-computation on the model as it was specified, so the effect differs across subgroups only where a term reads both columns, and that may be the modeling choice a caller meant to make. The warning reports which terms were read rather than announcing that the effect is the same in every subgroup, because a model may carry the modification through a column derived from the modifier instead: `y ~ exposure * sex_female` reported by `.by = sex` names no term reading `sex`, and its subgroup effects differ all the same.

Joint exposures

`causalgenerics::joint_exposure()` crosses two discrete treatments into one categorical exposure and records the crossing on the vector. The result is a factor, so `balance()` weights it as it weights any factor over those cells, and the crossing changes which effects are reported rather than which population is balanced.

Reported as a plain categorical exposure, such a fit gives each cell against the reference cell, under contrasts like "`a = 1, e = 0` vs `a = 0, e = 0`". Those rows are arithmetically right and they answer a question nobody asked. A declared crossing is reported in the two treatments instead, and the cell-against-cell rows are replaced rather than supplemented, so their labels appear nowhere a row is named: in no estimates column, no coefficient name, no covariance dimname, no interval rowname, and no printed row. The surface holds three kinds of row:

- the counterfactual mean of each cell, under the effect label "`mean`", with the cell as its contrast and "`overall`" as its group;
- the simple effects, each treatment's effect within a fixed level of the other, with the treatment as the contrast, written "`a: 1 vs 0`", and the level the other is held at as the group, written "`e = 0`". These include the comparisons cell-against-cell reporting cannot express at all, such as the first treatment's effect with the second set to one;
- the interaction, the difference between two of the first treatment's simple effects, with the two compared levels of the second as its group, written "`e = 1 vs e = 0`". It is reported once. Interaction is symmetric in the two treatments, so the difference between the first's simple effects is the difference between the second's, and reporting both would put one quantity in the table under two names.

A two-by-two crossing therefore reports fourteen rows for a binary outcome: four means, four simple effects on each of two scales, and the interaction on each of them. A continuous outcome reports nine, since it has one scale.

The group column names something different here than it names under `.by`. A group naming a level of the other treatment names the value that treatment is set to in the two cell means the row contrasts, which is a setting of the intervention rather than a subgroup of units. Every row on this surface, the cell means included, standardizes over the whole sample, so the contrasts are differences and double differences of cell means over one population.

No contrast row carries a log odds ratio, for the reason no subgroup row does: an odds ratio is noncollapsible, so neither a simple effect reported beside one nor a difference of two of them says what it appears to. The "`mean`" rows are means and carry no scale of their own.

Every row is a parameter of the same stacked system. The cell means are the block the categorical path already carries, and the contrast block is written over those same means, so a simple effect and the cell-against-cell contrast that happens to equal it report the same estimate and the same standard error. Each interaction row is the difference of two simple-effect parameters, which makes it the

double difference of the four means by construction rather than by two arithmetic routes that have to agree.

The declaration is read off the exposure column of the frame the method resolves, which is `.data` where the caller supplied one and the outcome model's own frame otherwise. The fit records its levels as plain strings and keeps no memory of the crossing, so it is the frame and not the fit that decides which surface is reported, and dropping the declaration with `factor(x)` returns the cell-against-cell rows.

A crossing whose two treatments carry one name, as in `joint_exposure(a = x, a = y)`, raises `balancing_ipw_input_error`. Every row is keyed by the treatment it contrasts and the level the other is held at, both written from the component names, so one name over two treatments names two different effects the same way. Declare each treatment under a name of its own.

Two further configurations raise `balancing_ipw_unsupported_error`. A declared crossing with `.by` is refused: effect modification of a joint intervention is a three-way question, the interaction between two treatments within the levels of a third variable, and this surface reports neither that nor a projection of it. A declared crossing weighted for anything but "ate" is refused as well: every cell mean here standardizes to one population, and a tilted estimand standardizes each of them to a population the simple effects and the interaction are not defined over.

Multiple imputation

Missing covariate data is handled by imputing first and analyzing within each completed dataset: impute, then balance, weight, and fit the outcome model once per imputation, then pool the per-imputation results with `causalgenerics::pool_ipw()`. That function combines them by Rubin's rules with a Barnard-Rubin degrees-of-freedom adjustment, and documents the rules and the agreements the results have to satisfy.

The per-imputation analysis is written as an `lapply()` over the completed datasets rather than as `with(imp, ...)`. `balance()` takes its data as the first argument and selects covariates out of it with `tidyselect`, so it needs that frame as a named object; inside `with()` the completed frame is the evaluation environment instead and has no name to give. Walking the imputations directly also keeps the frame in hand for the weight column and the outcome model, which the same step needs.

Only the estimating-equation methods at exact balance reach this workflow at all, for the reason they are the only ones `ipw()` accepts anywhere: the pooled standard error is built from per-imputation standard errors, and a fit carrying no estimating equations produces no result to pool. The refusal comes at the per-imputation `ipw()` call rather than at the pooling step.

The complete-data degrees of freedom are found without being told. A balancing result reports none of its own, since the stacked system is not a fit with residual degrees of freedom, so `causalgenerics::pool_ipw()` falls through to the outcome models and takes the smallest residual degrees of freedom among them. Pooling the same results with `mice::pool()` instead needs two things this route supplies on its own: the propensity package loaded, since the `tidy()` method it reaches an `ipw` result through lives there rather than in balancing, and that count passed explicitly as its `dfcom` argument, since it reads only what the results themselves report.

A pooled balancing result carries both readings. `ipw()` hands every outcome model over already wrapped with its block of the corrected covariance, so a set of balancing results always has a conditional surface to pool beside the marginal one, and `causalgenerics::pool_ipw()` pools both whichever reading it is asked for. `causalgenerics::as_marginal()` and `causalgenerics::as_conditional()` therefore move the pooled result between the readings after pooling, as they move an unpooled one,

and the pooled `stats::coef()`, `stats::vcov()`, `stats::confint()`, and `base::as.data.frame()` take an `effects` argument naming a reading for one call.

References

Kostouraki A, Hajage D, Rachet B, et al. On variance estimation of the inverse probability-of-treatment weighting estimator: A tutorial for different types of propensity score weights. *Statistics in Medicine*. 2024;43(13):2672-2694. doi:10.1002/sim.10078

Examples

```
n <- 200
x1 <- rnorm(n)
z <- rbinom(n, 1, plogis(0.5 * x1))
y <- rbinom(n, 1, plogis(-0.5 + 0.8 * z + 0.3 * x1))
df <- data.frame(exposure = z, x1 = x1, y = y)

fit <- balance(df, exposure, x1, method = bw_entropy(), estimand = "ate")
df$.wts <- weights(fit)

# quasibinomial() solves the same estimating equation as binomial() and does
# not warn that weights are not counts.
outcome_mod <- glm(
  y ~ exposure,
  data = df,
  family = quasibinomial(),
  weights = .wts
)

ipw(fit, outcome_mod)

# The outcome model may also adjust for covariates, in which case the
# marginal means are standardized over the estimand's target population.
adjusted_mod <- glm(
  y ~ exposure + x1,
  data = df,
  family = quasibinomial(),
  weights = .wts
)

ipw(fit, adjusted_mod)

# `by` reports the effects again within the levels of a modifier, then
# contrasts each level against the first of them.
df$grp <- factor(ifelse(x1 > 0, "high", "low"), levels = c("low", "high"))
by_mod <- glm(
  y ~ exposure * grp,
  data = df,
  family = quasibinomial(),
  weights = .wts
)
```

```

ipw(fit, by_mod, .by = grp)

# A categorical exposure reports each level against the reference level, and
# the estimates table names the contrast.
odds_b <- exp(0.6 * x1)
odds_c <- exp(-0.5 * x1)
denominator <- 1 + odds_b + odds_c
draw <- runif(n)
df$arm <- factor(
  ifelse(
    draw < 1 / denominator,
    "a",
    ifelse(draw < (1 + odds_b) / denominator, "b", "c")
  ),
  levels = c("a", "b", "c")
)
df$relapse <- rbinom(
  n,
  1,
  plogis(-0.4 + 0.5 * (df$arm == "b") + 0.9 * (df$arm == "c") + 0.3 * x1)
)

arm_fit <- balance(df, arm, x1, method = bw_ipt(), estimand = "ate")
df$.arm_wts <- weights(arm_fit)
arm_mod <- glm(
  relapse ~ arm,
  data = df,
  family = quasibinomial(),
  weights = .arm_wts
)

ipw(arm_fit, arm_mod)

# A continuous exposure reports the dose response of a weighted marginal
# structural model. An exposure entering through one design column is that
# response's slope, reported as one row named for the model's link.
df$dose <- 0.7 * x1 + rnorm(n)
df$score <- 2 + 0.5 * df$dose + 0.4 * x1 + rnorm(n)

dose_fit <- balance(df, dose, x1, method = bw_entropy(), estimand = "ate")
df$.dose_wts <- weights(dose_fit)
dose_mod <- lm(score ~ dose, data = df, weights = .dose_wts)

ipw(dose_fit, dose_mod)

# An exposure entering through several columns has no coefficient that is a
# causal effect, so the result records the conditional reading and supports
# no other. Naming that reading builds the same result without the
# announcement that explains the default.
curve_mod <- lm(score ~ poly(dose, 2), data = df, weights = .dose_wts)

ipw(dose_fit, curve_mod, effects = "conditional")

```

```

# With missing covariate data, analyze within each completed dataset and
# pool the results afterward.
set.seed(2024)
n <- 150
mx1 <- rnorm(n)
mx2 <- rnorm(n)
mz <- rbinom(n, 1, plogis(0.6 * mx1 - 0.4 * mx2))
my <- rbinom(n, 1, plogis(-0.3 + 0.5 * mz + 0.4 * mx1))
incomplete <- data.frame(exposure = mz, x1 = mx1, x2 = mx2, y = my)
incomplete$x1[rbinom(n, 1, plogis(-1.2 + 0.5 * mx2)) == 1] <- NA

imp <- mice::mice(incomplete, m = 2, print = FALSE, seed = 4321)

fits <- lapply(mice::complete(imp, "all"), function(completed) {
  completed_fit <- balance(
    completed,
    exposure,
    c(x1, x2),
    method = bw_entropy(),
    estimand = "ate"
  )
  completed$.wts <- weights(completed_fit)
  ipw(
    completed_fit,
    glm(
      y ~ exposure,
      data = completed,
      family = quasibinomial(),
      weights = .wts
    )
  )
})

pool_ipw(fits)

# The pooled result carries both readings, so it moves to the outcome
# models' coefficients after pooling.
as_conditional(pool_ipw(fits))

```

weights, balancing::balancing-method

Extract balancing weights

Description

Returns the weights a [balancing](#) result fitted, as a [bw](#) vector carrying the estimand they target. When sampling weights are present they are composed onto the balancing weights by default.

Arguments

<code>object</code>	A balancing result.
<code>...</code>	Ignored.
<code>include_sampling_weights</code>	Whether to multiply the balancing weights by the sampling weights. Defaults to TRUE.

Value

A [bw](#) vector of the same length as the data the fit was built from, in the same row order. It carries the estimand and no other metadata about the fit.

Index

as_bw (bw), 9

balance, 2

balance(), 4, 5, 7, 8, 11, 13, 15, 18–22, 25, 32

balance_method, 3, 4, 7, 12, 15, 18, 20, 21, 23

balance_terms, 5, 7

balance_terms(), 3, 14, 16–19, 23

balancing, 4, 7, 9, 24, 25, 36, 37

balancing_estimating_equations, 8, 24, 27

balancing_options, 8, 20

base::as.data.frame(), 34

bw, 7, 9, 36, 37

bw_cbps, 11

bw_cfd, 13

bw_energy, 15

bw_energy(), 6, 8, 14

bw_entropy, 18

bw_entropy(), 2–4, 6

bw_ipt, 20

bw_sbw, 22

bw_sbw(), 8, 16

causalgenerics::as_conditional(), 25, 33

causalgenerics::as_marginal(), 25, 26, 33

causalgenerics::estimand(), 9, 10

causalgenerics::ipw(), 25, 30

causalgenerics::is_causal_wt(), 9

causalgenerics::joint_exposure(), 27, 32

causalgenerics::new_ipw_model(), 30

causalgenerics::pool_ipw(), 33

deli::compute_sandwich(), 27

deli::ee_glm(), 27

estimating_equation_method
(balance_method), 4

estimating_equations, 24

ipw(), 5, 7, 8, 19, 21

ipw.balancing, 25

is_bw (bw), 9

mice::pool(), 33

new_bw (bw), 9

options(), 8

propensity::psw, 9, 10

quadratic_program_method
(balance_method), 4

setTimeLimit(), 3

stats::binomial(), 30

stats::coef(), 28, 34

stats::confint(), 28, 34

stats::df.residual(), 30

stats::glm(), 25, 27

stats::lm(), 25, 27, 29

stats::nobs(), 28, 30

stats::quasibinomial(), 29

stats::vcov(), 8, 26, 28, 30, 34

stats::weights(), 28

weights(), 9

weights, balancing::balancing-method, 36