Package: coresynth (via r-universe)

October 11, 2026

Title Fast and Unified Synthetic Control Methods

Version 1.0.1

Description A unified 'Formula' interface to the Synthetic Control Method (SCM) and related panel-data causal inference estimators: Synthetic Difference-in-Differences (SDID), Generalized Synthetic Control (GSC), Matrix Completion (MC), Time-Aware Synthetic Control (TASC), and Synthetic Interventions (SI), together with an experimental-design variant. Computational bottlenecks (quadratic programming, singular value decomposition, and Kalman filtering) are implemented in 'C++' via 'RcppArmadillo'. Methods are described in Abadie, Diamond and Hainmueller (2010) <doi:10.1198/jasa.2009.ap08746>, Arkhangelsky, Athey, Hirshberg, Imbens and Wager (2021) <doi:10.1257/aer.20190159>, Xu (2017) <doi:10.1017/pan.2016.2>, Athey, Bayati, Doudchenko, Imbens and Khosravi (2021) <doi:10.1080/01621459.2021.1891924>, and Agarwal, Shah and Shen (2025) <doi:10.1287/opre.2025.1590>.

License MIT + file LICENSE

URL https://github.com/yo5uke/coresynth, https://yo5uke.com/coresynth/

BugReports https://github.com/yo5uke/coresynth/issues

Encoding UTF-8

Config/roxygen2/markdown TRUE

Depends R (>= 4.1.0)

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, Formula, ggplot2, broom, jsonlite

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Yosuke Abe [aut, cre]

Maintainer Yosuke Abe <yosuke.abe0507@gmail.com>

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

Date/Publication 2026-10-11 15:49:05 UTC

RemoteUrl https://github.com/cran/coresynth

RemoteRef HEAD

RemoteSha 9ad5134206e09588439c2b02c386cce12ffc82e4

Contents

augment_scm . . . 3
conformal_inference . . . 4
export_json . . . 5
glance.coresynth_inference . . . 6
gsc_boot . . . 6
gsc_ife_cpp . . . 7
gsc_inference . . . 8
kalman_smoother_cpp . . . 9
loo_donors . . . 10
mspe_ratio_pval . . . 11
placebo_in_time . . . 13
plot.coresynth . . . 14
plot.scm_design . . . 16
plot.scm_frontier . . . 17
plot.scm_placebo . . . 18
plot_data . . . 20
pred . . . 23
scm_balance_frontier . . . 24
scm_design . . . 25
scm_fit . . . 27
scm_inference . . . 33
scm_inner_weights_cpp . . . 35
scm_placebo_cpp . . . 36
scm_placebo_x_cpp . . . 37
scm_rank_pval . . . 38
scm_weights_cpp . . . 39
sdid_estimate_cpp . . . 41
sdid_inference . . . 42
sdid_placebo_cpp . . . 43
sdid_time_weights_cpp . . . 44
sdid_unit_weights_cpp . . . 44
si_inference . . . 45
si_pcr_cpp . . . 46
soft_impute_cpp . . . 46
soft_impute_fe_cpp . . . 47
tensor_unfold_cpp . . . 48

tidy.coresynth_inference 49
treated_outcomes 49

Index **52**

`augment_scm` *Augmented Synthetic Control Method (Ridge ASCM)*

Description

Applies a ridge-regression-based bias correction to a fitted SCM object, following Ben-Michael, Feller & Rothstein (2021, JASA). The corrected estimator is:

Usage

```
augment_scm(fit, lambda_ridge = NULL)
```

Arguments

`fit`	A `coresynth` object from `scm_fit()` with `method = "scm"`.
`lambda_ridge`	Ridge penalty (non-negative). `NULL` (default) selects the penalty by leave-one-out cross-validation on the control units.

Details

tau_aug = tau_SCM + (m_tr_post - sum_j W_j * m_j_post)

where m_i_post = Y_pre_i' beta_hat is the ridge outcome model prediction for unit i's mean post-treatment outcome, and beta_hat is estimated by ridge regression across control units.

Value

A list with:

- `att_aug`: Augmented ATT estimate
- `delta`: Bias correction term (m_tr_post - sum_j W_j m_j_post)
- `att_scm`: Original SCM ATT for comparison
- `lambda_ridge`: Ridge penalty used
- `beta_hat`: Ridge regression coefficients (length T_pre)

`conformal_inference` *Conformal Inference for Synthetic Control Estimators*

Description

Implements the permutation-based conformal inference procedure of Chernozhukov, Wuthrich & Zhu (2021, JASA). The test inverts a sharp null $H_0 : \tau = \tau_0$ by imputing the treated post-treatment counterfactual as $Y_{1t} - \tau_0$, re-estimating the counterfactual proxy on **all** T periods (imposing the null), and computing a moving-block permutation p-value from the estimated residuals. A confidence interval is obtained by test inversion over a grid of candidate τ_0.

Usage

```
conformal_inference(
  fit,
  tau0 = 0,
  q = 1,
  alternative = c("two.sided", "greater", "less"),
  ci = TRUE,
  level = 0.95,
  grid = NULL,
  n_grid = 200L,
  grid_mult = 4,
  ...
)
```

Arguments

`fit`	A `coresynth` object from `scm_fit()`.
`tau0`	Null value of the ATT for the reported p-value (default 0).
`q`	Exponent of the S_q test statistic (`S_q = (T_post^{-1} \sum |u_t|^q)^{1/q}`). Default 1, robust to heavy-tailed data (CWZ 2021). Used only for `alternative = "two.sided"`; one-sided tests use the signed mean post-treatment residual.
`alternative`	`"two.sided"` (default), `"greater"`, or `"less"`.
`ci`	Logical; construct a confidence interval by test inversion (default `TRUE`).
`level`	Confidence level for the interval (default 0.95).
`grid`	Optional numeric vector of candidate τ_0 values for test inversion. When `NULL` (default), a grid of `n_grid` points is centred on the point estimate with half-width `grid_mult` times the pre-treatment residual standard deviation.
`n_grid`	Number of grid points when `grid = NULL` (default 200).
`grid_mult`	Half-width multiplier when `grid = NULL` (default 4).
`...`	Unused.

Details

Supported for **sharp** (single-cohort) fits with `method` in `c("scm", "sdid", "gsc", "mc", "si")`. Staggered, multi-arm, and `tasc` fits are not supported (use `sdid_inference()`, `gsc_inference()`, or `si_inference()` instead).

Value

A list of class `c("conformal_inference", "coresynth_inference")` with `estimate`, `se` (`NA`; conformal has no SE), `p_value` (at `tau0`), `ci_lower`, `ci_upper`, `method` (`"conformal"`), `n_controls`, `alternative`, `staggered` (`FALSE`), plus `tau0`, `q`, `grid`, and `p_grid` (p-values along the grid). Compatible with `tidy()` / `glance()`.

References

Chernozhukov, V., Wuthrich, K., & Zhu, Y. (2021). An Exact and Robust Conformal Inference Method for Counterfactual and Synthetic Controls. *Journal of the American Statistical Association*, 116(536), 1849-1864.

`export_json` *Export coresynth Results to JSON*

Description

Generates a comprehensive, standardized JSON record covering all six estimators. Suitable for reproducibility workflows (Xu & Yang 2026) and downstream tooling. Pass the result of `mspe_ratio_pval()` or `gsc_boot()` via the `inference` argument to include inference results.

Usage

```
export_json(x, file = "coresynth_results.json", inference = NULL, digits = 6L)
```

Arguments

`x`	A `coresynth` object from `scm_fit()`.
`file`	Output file path. Default `"coresynth_results.json"`. Pass `NULL` to skip writing and return the R list invisibly.
`inference`	Optional list from `mspe_ratio_pval()` or `gsc_boot()`. When provided, populates the `inference` section and updates `estimate` with `p_value`, `se`, `ci_lower`, `ci_upper`.
`digits`	Number of significant digits applied to numeric values (default 6L).

Value

Invisibly, the R list that was (or would be) serialized.

glance.coresynth_inference

Glance at an inference result

Description

One-row summary of a `coresynth_inference` (or `sdid_inference`) object.

Usage

```
## S3 method for class 'coresynth_inference'
glance(x, ...)
```

Arguments

`x`	An inference object.
`...`	Unused.

Value

A one-row `data.frame` with columns `method`, `n_controls`, `staggered`, `estimate`, `std.error`, `p.value`, `conf.low`, `conf.high`, `alternative`, `n_boot_valid`.

gsc_boot

Parametric Bootstrap Inference for GSC (Xu 2017 S.3)

Description

Generates the null distribution of the ATT under H0 (no treatment effect) by parametric resampling from the estimated IFE factor model. Under H0, both the control panel and treated unit are generated from the fitted factor model with homoskedastic noise. When the fit includes covariate adjustment (beta), the covariate contribution is included in the simulated DGP and re-estimated in each bootstrap replicate.

Usage

```
gsc_boot(fit, B = 499L, alpha = 0.05, seed = NULL)
```

Arguments

`fit`	A `coresynth` object from `scm_fit()` with `method = "gsc"`.
`B`	Bootstrap replications (default 499L).
`alpha`	Significance level for the confidence interval (default 0.05).
`seed`	RNG seed for reproducibility (default NULL).

Value

A list with:

- p_value: Two-sided p-value: mean(|ATT*| >= |ATT_obs|)
- ci_lower: Lower bound of the (1-alpha)*100% CI for the ATT, ATT_obs - q(1 - alpha/2) where q are quantiles of boot_dist
- ci_upper: Upper bound, ATT_obs - q(alpha/2)
- se: Bootstrap standard error
- boot_dist: Numeric vector of length B (bootstrap ATT* values)
- att_obs: Observed ATT from the original fit

---

gsc_ife_cpp *Fast Interactive Fixed Effects (IFE) for Generalized Synthetic Control*

Description

Implements Xu (2017) IFE model with optional covariate adjustment. When X_co has p > 0 slices, runs an EM loop alternating between: E-step: truncated SVD of Y_tilde = Y_co - X_co * beta M-step: panel OLS to update beta given current factors When X_co has 0 slices (default), falls back to the plain 3-step estimator.

Usage

```
gsc_ife_cpp(Y_co, Y_tr_pre, r, X_co, X_tr_pre, max_iter = 50L, tol = 1e-06)
```

Arguments

Y_co	Control units outcome matrix (T x N_co)
Y_tr_pre	Treated units pre-treatment outcomes (T_pre x N_tr)
r	Number of latent factors (must be <= min(T, N_co))
X_co	Time-varying covariate cube (T x N_co x p). Pass an empty cube (0 slices) for the covariate-free estimator.
X_tr_pre	Time-varying covariate cube for treated units in the pre-treatment window (T_pre x N_tr x p). Required for correct Step 2 loading estimation per Xu (2017): lambda_hat is estimated from Y_tr_pre - X_tr_pre * beta (covariate- demeaned). Pass an empty cube (0 slices) to skip demeaning (backward-compatible, but biased when beta != 0).
max_iter	Maximum EM iterations (default 50)
tol	Convergence tolerance on relative beta change (default 1e-6)

Value

A list with components:

- `F`: estimated time factors (T x r).
- `L_co`: control-unit factor loadings (N_co x r).
- `L_tr`: treated-unit factor loadings (N_tr x r).
- `Y_tr_hat`: estimated treated-unit counterfactual outcomes (T x N_tr).
- `singular_values`: singular values from the final truncated SVD.
- `beta`: estimated covariate coefficients (p x 1), empty when no covariates are supplied.

`gsc_inference` *Non-parametric Inference for GSC (Xu 2017)*

Description

Estimates SE and confidence intervals for the ATT via non-parametric cluster bootstrap or jackknife over control units. Works for both sharp and staggered GSC fits. For staggered fits, bootstrap resamples each cohort's control pool independently, and jackknife uses a per-cohort LOO with delta-method variance aggregation.

Usage

```
gsc_inference(
  fit,
  method = c("bootstrap", "jackknife", "jackknife_global"),
  n_boot = 499L,
  level = 0.95,
  alternative = c("two.sided", "greater", "less"),
  seed = NULL
)
```

Arguments

`fit`	A `coresynth` object from `scm_fit()` with `method = "gsc"`.
`method`	`"bootstrap"` (default) or `"jackknife"`.
`n_boot`	Number of bootstrap replications (default 499L; ignored for jackknife).
`level`	Confidence level (default 0.95).
`alternative`	`"two.sided"` (default), `"greater"`, or `"less"`.
`seed`	RNG seed for reproducibility (default NULL).

Details

Note: `gsc_boot()` performs a *parametric* bootstrap under H0 (hypothesis testing). `gsc_inference()` provides *non-parametric* SE and CIs suitable for inference about the ATT magnitude.

Value

A list of class `coresynth_inference`.

`kalman_smoother_cpp` *Kalman Filter and RTS Smoother (TASC)*

Description

Implements the Kalman filter (forward pass) and Rauch-Tung-Striebel smoother (backward pass) for the state-space model in Rho et al. (2026):

Usage

```
kalman_smoother_cpp(Y, W, A, C, Q, R, z0, P0)
```

Arguments

`Y`	Observed data matrix (N x T). Use NA for unobserved entries.
`W`	Observation / loading matrix (N x r)
`A`	State transition matrix (r x r). Pass diag(r) for random-walk dynamics.
`C`	State drift vector (r x 1)
`Q`	State noise covariance (r x r)
`R`	Observation noise covariance (N x N, diagonal in practice)
`z0`	Initial state mean (r x 1)
`P0`	Initial state covariance (r x r)

Details

State: z(t+1) = A z(t) + C + eta(t), eta(t) ~ N(0, Q) Observation: y_t = W z_t + eps_t, eps_t ~ N(0, R)

Observation rows with NA (treated post-intervention) are automatically dropped at each time step so only control-unit rows update the filter.

The P update uses the numerically stable Joseph form: P(t|t) = (I - K W_obs) P(t|t-1) (I - K W_obs)^T + K R_obs K^T

Value

A list with z_smooth, P_smooth, P_cross, z_pred, z_upd. P_cross is an r x r x (T-1) cube. Slice t (C++ 0-indexed, t=0,...,T-2) stores P(t+1, t | T) (0-indexed), i.e. P(t+2, t+1 | T) in 1-indexed Shumway-Stoffer notation. Formula: P(t+1|T) * J_t^T (eq. 6.68-6.69).

loo_donors *Leave-One-Out Donor Robustness for SCM*

Description

Iteratively re-estimates the synthetic control excluding one contributing donor at a time, holding the predictor weights V fixed at their baseline values (Abadie, Diamond & Hainmueller 2015, footnote 20). The spread of the leave-one-out ATT estimates shows how much the result hinges on any single donor.

Usage

```
loo_donors(fit, weight_threshold = 1e-06)
```

Arguments

`fit`	A sharp `coresynth` object from `scm_fit()` with `method = "scm"`.
`weight_threshold`	Only donors whose baseline weight exceeds this value are dropped (removing a zero-weight donor cannot change the fit). Default `1e-6`.

Details

For penalised fits (`lambda_pen` used), the same penalty is re-applied in each leave-one-out QP.

Value

A list with:

- `att_original`: baseline ATT
- `results`: data.frame with one row per excluded donor (`donor`, `weight`, `att_loo`)
- `att_range`: range of the leave-one-out ATTs

See Also

`placebo_in_time()`, `mspe_ratio_pval()`

mspe_ratio_pval *Permutation Inference via MSPE Ratio for SCM*

Description

Computes the Abadie et al. (2010) / Abadie (2021) permutation p-value. For each control unit, a leave-one-out synthetic control is fitted.

Usage

```
mspe_ratio_pval(
  fit,
  mspe_threshold = 0,
  max_iter = 100L,
  tol = 1e-04,
  use_covariates = NULL,
  alternative = c("two.sided", "greater", "less"),
  mspe_prune = Inf,
  statistic = c("ratio", "post_mspe")
)
```

Arguments

fit	A coresynth object from scm_fit() with method = "scm".
mspe_threshold	Minimum pre-treatment MSPE for including a control unit in the two-sided test. Ignored for one-sided tests. Default: 0 (no filtering).
max_iter	Passed to scm_placebo_cpp() (outcomes-only fits) or scm_placebo_x_cpp() (covariate fits). Default 100L.
tol	Passed to scm_placebo_cpp() or scm_placebo_x_cpp(). Default 1e-4.
use_covariates	Controls which predictor specification the placebo refits use. Default NULL (recommended) mirrors the treated fit: if the fit was estimated with a predictors specification, each placebo unit is refit with that same specification (the Abadie et al. 2010 / Synth convention – treated and placebo statistics are computed under one common spec); outcomes-only fits use the fast C++ outcomes-only placebo. Set TRUE/FALSE to force either path; note that FALSE on a covariate fit compares a covariate-based treated statistic against outcomes-only placebo statistics, which breaks the exchangeability logic of the permutation test.
alternative	Direction of the alternative hypothesis: "two.sided" (default) uses the MSPE ratio statistic; "greater" tests whether the treatment increased the outcome; "less" tests whether the treatment decreased the outcome. One-sided tests use the signed ATT as the test statistic.
mspe_prune	Discard placebo runs whose pre-treatment MSPE exceeds this multiple of the treated unit's, following Abadie, Diamond & Hainmueller (2010, Figures 5-7), who use 20, 5 and 2. A run the synthetic control could not fit before treatment carries no information about how rare a large post-treatment gap is, so leaving

it in the permutation distribution only adds noise. `Inf` (default) keeps every run. This is a different control from `mspe_threshold`, which is a floor guarding the ratio's denominator against near-zero pre-treatment MSPE.

`statistic` Which quantity to rank, for two-sided tests. `"ratio"` (default) is the post/pre MSPE ratio of Abadie, Diamond & Hainmueller (2010, Figure 8), whose denominator adjusts for how well each unit was fitted and so needs no cutoff. `"post_mspe"` is the alternative of Abadie (2021, S.3.5): rank the post-treatment MSPE itself, having discarded the ill-fitting runs through `mspe_prune`. Pair it with a finite `mspe_prune`; without one, placebo runs that were never fitted dominate the distribution.

Details

When `alternative = "two.sided"` (default), the test statistic is the post/pre MSPE ratio, following Abadie et al. (2010). When `alternative = "greater"` or `"less"`, the test statistic is the signed average post-treatment gap (ATT), giving a one-sided permutation test as recommended by Abadie (2021) S.3.5 for improved power when the direction of the treatment effect is known.

The placebo refits mirror the treated fit's outer optimiser and evaluation window: a fit estimated with the multi-start outer search (`v_optim = "multistart"`, or the `"auto"` default with predictors) or with a `v_window` runs every placebo unit through the same configuration, keeping the permutation statistic exchangeable across units.

Value

An object of class `scm_placebo` (a list) with:

- `p_value`: Permutation p-value between 0 and 1
- `mspe_ratio_treated`: MSPE_post / MSPE_pre for the treated unit (two.sided only)
- `mspe_ratios_all`: Named numeric vector (treated first, then controls); two.sided only
- `placebo_effects`: Named N_co-vector of placebo ATT estimates
- `treated_effect`: ATT estimate for the treated unit
- `n_placebo_used`: Number of control units used
- `n_placebo_pruned`: Number discarded by `mspe_prune`
- `mspe_prune`, `statistic`: the settings the p-value was computed under
- `gaps`: T x N_co matrix of placebo gap paths (unit minus its synthetic control over all periods), for the Abadie et al. (2010) Figure 4-7 plot
- `treated_gap`: T-vector of the treated unit's gap path
- `mspe_pre_treated`, `mspe_pre_placebo`: Pre-treatment MSPEs used for the relative pruning rule in `plot.scm_placebo()`
- `times`, `T_pre`: Time axis metadata for plotting

See Also

`plot.scm_placebo()` for the placebo gap and MSPE ratio plots.

placebo_in_time *In-Time Placebo (Backdating) Test for SCM*

Description

Re-estimates the synthetic control after artificially backdating the treatment to a pre-treatment period, following Abadie, Diamond & Hainmueller (2015) and Abadie & Vives-i-Bastida (2022, principle 7: "out-of-sample validation is key"). Only pre-treatment data enter the exercise, so the placebo gap after `t0_placebo` is uncontaminated by the actual intervention. A credible design shows no sizable divergence at the backdated treatment time.

Usage

```
placebo_in_time(fit, t0_placebo = NULL)
```

Arguments

`fit`	A sharp `coresynth` object from `scm_fit()` with `method = "scm"`.
`t0_placebo`	Backdated treatment period as a 1-based position in `fit$times`; must satisfy `2 <= t0_placebo < T_pre`. Default `floor(T_pre / 2)`.

Details

The refit uses the outcomes of periods `1..t0_placebo` as predictors (the `predictors = NULL` default), regardless of how the original fit was specified, because user-supplied `pred()` windows cannot be lagged automatically (ADH 2015 lag their predictors by hand).

Value

A list with:

- `t0_placebo`: the backdated treatment period used
- `times`: time values of the pre-treatment window
- `unit_weights`: placebo donor weights
- `Y_treat`, `Y_synth`, `gap`: series over the pre-treatment window
- `placebo_att`: mean placebo gap over `(t0_placebo, T_pre]`
- `fit_rmspe`: RMSPE over the placebo fitting window `1..t0_placebo`
- `eval_rmspe`: RMSPE over the placebo post window `(t0_placebo, T_pre]`

See Also

`mspe_ratio_pval()` for in-space placebos, `loo_donors()` for donor-robustness checks.

`plot.coresynth` *Plot a coresynth model*

Description

Plot a coresynth model

Usage

```
## S3 method for class 'coresynth'
plot(
  x,
  type = c("trend", "gap", "weights", "pred_weights"),
  colors = NULL,
  labels = NULL,
  linetypes = NULL,
  vline = list(),
  vline_offset = .VLINE_OFFSET_DEFAULT,
  hline = list(),
  fill = NULL,
  top_n = Inf,
  align = FALSE,
  show_donors = 0,
  ...
)
```

Arguments

`x`	A `coresynth` object.
`type`	One of `"trend"` (observed vs synthetic), `"gap"` (ATT over time), `"weights"` (donor unit weight bar chart; SDID fits get a second panel with the time weight profile), or `"pred_weights"` (the predictor/variable weight matrix V as a bar chart; sharp SCM only).
`colors`	For `type = "trend"`: a named vector overriding series colors, e.g. `c(treated = "black")` (valid keys: `"treated"`, `"synthetic"`, plus `"donors"` when `show_donors > 0`). For `type = "gap"`: a single color string for the gap line. Ignored for `type = "weights"` (use `fill` instead).
`labels`	For `type = "trend"`: a named vector overriding the legend text of individual series, e.g. `c(treated = "California")` (valid keys: `"treated"`, `"synthetic"`, plus `"donors"` when `show_donors > 0`). Series not mentioned keep their default label; `colors` and `labels` address series by the same keys, independent of the displayed legend text. Ignored for other types (no legend).
`linetypes`	For `type = "trend"`: a named vector overriding the line type of individual series, e.g. `c(synthetic = "solid")` to draw both the treated and the synthetic series solid (valid keys: `"treated"`, `"synthetic"`, plus `"donors"` when `show_donors > 0`; defaults are `"solid"` for the treated and donor series and

"dashed" for the synthetic one). For type = "gap": a single line type for the gap line (default "solid"). Values are ggplot2 line type names ("solid", "dashed", "dotted", "dotdash", "longdash", "twodash", "blank") or the equivalent integer codes 0:6. Series are addressed by the same keys as colors and labels, so the three compose independently. Ignored for type = "weights" and "pred_weights" (bar charts).

vline — Aesthetic overrides for the vertical treatment-time line, as a list passed to ggplot2::geom_vline() (e.g. list(color = "red")). NULL or FALSE hides the line entirely. The list may also carry an xintercept element giving one or more absolute positions on the time axis (e.g. list(xintercept = 1988), or a date string such as "1989-01-01" on a Date axis), replacing the default treatment-time position; for placement relative to the treatment period use vline_offset instead. Applies to "trend" and "gap".

vline_offset — For type = "trend" and "gap": where to draw the vertical treatment line, in periods relative to the first post-treatment period. The default -1 puts the line on the last pre-treatment period, so everything to its left is pre-treatment fit and the divergence begins to its right; 0 moves it onto the first post-treatment period, where the treated unit and its counterfactual have usually separated already. Fractional values interpolate between adjacent observed times (-0.5 is midway between the last pre- and first post-treatment period), which works on numeric, Date, and POSIXct axes alike. Cannot be combined with an xintercept element in vline.

hline — Aesthetic overrides for the horizontal zero line in type = "gap", as a list passed to ggplot2::geom_hline(). NULL or FALSE hides the line. Ignored for other types.

fill — For type = "weights" and type = "pred_weights": a single color string overriding the bar fill. Ignored for other types.

top_n — For type = "weights": show only the top_n donors with the largest weights (default Inf keeps every donor with a non-negligible weight). For type = "pred_weights": show only the top_n predictors with the largest V weights (default Inf shows every predictor). Ignored for other types (and for the SDID time weight panel, which is always shown in full).

align — For type = "trend" and "gap": when TRUE, shift the synthetic series by its pre-treatment level gap to the treated series so that both are drawn on the same level. SDID matches trends only up to a free intercept, so its raw trend plot shows the synthetic control at a different level; with align = TRUE the offset is the time-weighted (λ) pre-period gap, which makes the average post-period gap in the plot equal the SDID estimate exactly. Other methods use the plain pre-period mean gap (usually a negligible shift, since they already match levels). Default FALSE (raw series).

show_donors — For type = "trend": also draw the outcome paths of the show_donors donor units with the largest weights as thin background lines (Inf shows every donor). Requires a fit that stores donor unit weights and outcomes (sharp SCM/SDID/SI). Default 0 (no donor paths).

... — Ignored.

Value

A ggplot2 plot object.

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:10, year = 1:20)
panel$treated <- as.integer(panel$unit == 5 & panel$year > 15)
panel$gdp <- panel$unit + 0.5 * panel$year +
  rnorm(nrow(panel)) + 3 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")


plot(fit, type = "trend")
plot(fit, type = "gap")
plot(fit, type = "weights")
plot(fit, type = "weights", top_n = 5)

# Predictor (V) weights: which predictors the fit leans on
plot(fit, type = "pred_weights")

# Overlay the five largest donors behind the treated/synthetic series
plot(fit, type = "trend", show_donors = 5)

# SDID: align the synthetic series on the lambda-weighted pre-period level
fit_sdid <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "sdid")
plot(fit_sdid, type = "trend", align = TRUE)
plot(fit_sdid, type = "gap",   align = TRUE)

# Customize series colors, legend text, line types, and reference lines
plot(fit, type = "trend",
     colors = c(treated = "black"),
     labels = c(treated = "Unit 5"),
     vline  = list(color = "red", linetype = "dashed"))

# Draw both series solid, or restyle the gap line
plot(fit, type = "trend", linetypes = c(synthetic = "solid"))
plot(fit, type = "gap",   linetypes = "dashed")

# Move the treatment line onto the first post-treatment period,
# pin it to an absolute time, or drop it entirely
plot(fit, type = "trend", vline_offset = 0)
plot(fit, type = "trend", vline = list(xintercept = 15.5))
plot(fit, type = "trend", vline = FALSE)
```

---

plot.scm_design *Plot an scm_design object*

Description

Plot an scm_design object

Usage

```
## S3 method for class 'scm_design'
plot(x, type = c("outcome", "gap"), ...)
```

Arguments

x	An scm_design object.
type	"outcome" (default): synthetic treated vs synthetic control outcome series over all periods. "gap": estimated treatment effect in the experimental periods, with split-conformal confidence intervals.
...	Currently ignored.

Value

A ggplot object: for type = "outcome", the synthetic treated and synthetic control outcome series; for type = "gap", the estimated treatment effect over the experimental periods with split-conformal confidence intervals. The object is returned for printing or further customisation.

---

plot.scm_frontier *Plot a Balance Possibility Frontier*

Description

Draws the trade-off traced by scm_balance_frontier(): the imbalance of the treated average, which the aggregate ATT rests on, against the root mean square of the per-block imbalances, which the block-level effects rest on. Separate SCM sits at one end of the curve and fully pooled SCM at the other, with the heuristic the fit used marked along it.

Usage

```
## S3 method for class 'scm_frontier'
plot(x, relative = TRUE, label_nu = TRUE, colors = NULL, ...)
```

Arguments

x	An scm_frontier object from scm_balance_frontier().
relative	Plot the imbalances normalised by the separate solution (default), which puts both axes on comparable scales, rather than in the outcome's own units.
label_nu	Annotate the drawn points with their nu.
colors	Optional named overrides for the "frontier" line and the "heuristic" marker.
...	Ignored.

Details

The curve is normally strongly convex, which is the point of looking at it: a small move away from either end usually buys a large reduction in the other imbalance for very little, so the interesting values of nu are the ones near the bend.

Value

A ggplot2 plot object.

See Also

scm_balance_frontier()

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:15, year = 1:20)
adopt <- c(`1` = 12, `2` = 15, `3` = 17)
panel$treated <- as.integer(!is.na(adopt[as.character(panel$unit)]) &
  panel$year >= adopt[as.character(panel$unit)])
panel$treated[is.na(panel$treated)] <- 0L
panel$gdp <- panel$unit + 0.4 * panel$year +
  rnorm(nrow(panel)) + 2 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")
plot(scm_balance_frontier(fit, nu_grid = c(0, 0.25, 0.5, 0.75, 1)))
```

---

plot.scm_placebo *Plot SCM In-Space Placebo Results*

Description

Visualizes the placebo study returned by mspe_ratio_pval(), following Abadie, Diamond & Hainmueller (2010, Section 3.4).

Usage

```
## S3 method for class 'scm_placebo'
plot(
  x,
  type = c("gaps", "ratios"),
  mspe_prune = NULL,
  colors = NULL,
  labels = NULL,
  linetypes = NULL,
  vline = list(),
  vline_offset = .VLINE_OFFSET_DEFAULT,
  hline = list(),
  ...
)
```

Arguments

x	A scm_placebo object from mspe_ratio_pval().
type	One of "gaps" (ADH 2010, Figures 4-7) or "ratios" (ADH 2010, Figure 8).
mspe_prune	Exclude placebo units whose pre-treatment MSPE exceeds mspe_prune times the treated unit's. NULL (default) reuses the cutoff mspe_ratio_pval() computed the p-value under, so the picture and the test describe the same set of placebo runs; Inf keeps every run whatever the test did. A rule stated on the RMSPE scale, such as tidysynth's "2 times the treated unit's pre-period RMSPE", corresponds to the squared multiple (mspe_prune = 4).
colors	A named vector overriding series colors, e.g. c(treated = "black"). Valid keys: "treated", "placebo".
labels	A named vector overriding the legend text of individual series, e.g. c(treated = "California"). Valid keys: "treated", "placebo". Series not mentioned keep their default label; colors and labels address series by the same keys, independent of the displayed legend text.
linetypes	Only for type = "gaps": a named vector overriding the line type of individual series, e.g. c(placebo = "dotted"). Valid keys: "treated", "placebo" (both default to "solid"). Values are ggplot2 line type names ("solid", "dashed", "dotted", "dotdash", "longdash", "twodash", "blank") or the equivalent integer codes 0:6. Series are addressed by the same keys as colors and labels. Ignored for type = "ratios" (points, not lines).
vline	Only for type = "gaps": aesthetic overrides for the vertical treatment-time line, as a list passed to ggplot2::geom_vline(). NULL or FALSE hides the line entirely. The list may also carry an xintercept element giving one or more absolute positions on the time axis, replacing the default treatment-time position.
vline_offset	Only for type = "gaps": where to draw the vertical treatment line, in periods relative to the first post-treatment period. The default -1 puts the line on the last pre-treatment period; 0 moves it onto the first post-treatment period, and fractional values interpolate between adjacent observed times. Cannot be combined with an xintercept element in vline.
hline	Only for type = "gaps": aesthetic overrides for the horizontal zero line, as a list passed to ggplot2::geom_hline(). NULL or FALSE hides the line entirely.
...	Ignored.

Details

type = "gaps" overlays the treated unit's gap path (treated minus synthetic control) on the placebo gap paths obtained by reassigning the intervention to each donor unit (ADH 2010, Figure 4). Placebo units whose synthetic control fits poorly before treatment carry no information about the rarity of a large post-treatment gap, so ADH exclude units whose pre-treatment MSPE exceeds a multiple of the treated unit's: 20, 5, and 2 in their Figures 5-7 (mspe_prune).

type = "ratios" shows the statistic behind the two-sided permutation p-value, one point per unit (ADH 2010, Figure 8): the post/pre-treatment MSPE ratio, or the post-treatment MSPE where the test was run with statistic = "post_mspe". The ratio needs no pruning cutoff by construction, which is why mspe_ratio_pval() leaves mspe_prune at Inf; where the test did prune, the same

runs are dropped here so that the points and the p-value in the subtitle describe one set of placebo runs.

Value

A ggplot2 plot object.

See Also

mspe_ratio_pval()

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:10, year = 1:20)
panel$treated <- as.integer(panel$unit == 5 & panel$year > 15)
panel$gdp <- panel$unit + 0.5 * panel$year +
  rnorm(nrow(panel)) + 3 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")
placebo <- mspe_ratio_pval(fit)


# Treated gap overlaid on the donor-pool placebo gaps (ADH 2010, Fig. 4)
plot(placebo, type = "gaps")

# Prune poorly fitting placebos and relabel the legend
plot(placebo, type = "gaps", mspe_prune = 5,
     labels = c(treated = "Unit 5"))

# Set the line type of the placebo paths off against the treated one
plot(placebo, type = "gaps", linetypes = c(placebo = "dotted"))

# Move the treatment line onto the first post-treatment period
plot(placebo, type = "gaps", vline_offset = 0)

# Post/pre-treatment MSPE ratios (ADH 2010, Fig. 8)
plot(placebo, type = "ratios")
```

---

plot_data *Extract the tidy data behind a coresynth plot*

Description

Returns the tidy data.frame that plot() draws for a given type, so the underlying series, weights, or placebo paths can be inspected, joined into a table, or re-plotted directly. plot() stays the quick path; plot_data() is the handle for anyone who wants to relabel simplified series names, feed the numbers into their own figure, or postprocess them further.

Usage

```
plot_data(x, ...)

## Default S3 method:
plot_data(x, ...)

## S3 method for class 'coresynth'
plot_data(
  x,
  type = c("trend", "gap", "weights", "pred_weights"),
  align = FALSE,
  top_n = Inf,
  show_donors = 0,
  ...
)

## S3 method for class 'scm_placebo'
plot_data(x, type = c("gaps", "ratios"), mspe_prune = NULL, ...)
```

Arguments

x	A coresynth fit (from scm_fit()) or a scm_placebo object (from mspe_ratio_pval()).
...	Passed to methods (unused by the current methods).
type	For a coresynth fit: one of "trend", "gap", "weights", or "pred_weights", matching plot.coresynth(). For a scm_placebo object: "gaps" or "ratios", matching plot.scm_placebo().
align	For type = "trend"/"gap": when TRUE, shift the synthetic series by its pre-treatment level gap to the treated series, exactly as in plot.coresynth(). Default FALSE (raw series).
top_n	For type = "weights"/"pred_weights": keep only the top_n largest weights (default Inf, every row).
show_donors	For type = "trend": also return the outcome paths of the show_donors largest-weight donors as series = "Donors" rows, adding a unit column that identifies each donor (NA for the treated and synthetic series). Default 0 (treated and synthetic series only).
mspe_prune	For a scm_placebo object: drop placebo units whose pre-treatment MSPE exceeds mspe_prune times the treated unit's, as in plot.scm_placebo(). NULL (default) reuses the cutoff mspe_ratio_pval() computed the p-value under; Inf keeps every run.

Details

The frame mirrors what the matching plot(x, type = ...) call shows, with two deliberate departures that make it a better data source:

- Plain column names (time, value, series, weight, ...) are used instead of the dotted convention of augment(), since this is data to manipulate rather than model-augmented observations.

- The cosmetic "drop donors with weight below 1e-4" filter that `plot(type = "weights")` applies is *not* used here: every donor is returned (use `top_n` to subset), so the frame is the complete set of weights.

Only the arguments that change *which rows or values* appear are accepted (`align`, `top_n`, `show_donors`, `mspe_prune`); purely cosmetic arguments of `plot()` (`colors`, `labels`, `linetypes`, `vline`, `fill`, ...) have no data counterpart and are not part of this interface.

Value

A tidy `data.frame`. Columns by `type`:

`"trend"` `time`, `value`, `series` (`"Treated"` / `"Synthetic Control"`); with `show_donors > 0`, also `"Donors"` rows and a `unit` column.

`"gap"` `time`, `gap` (treated minus synthetic control).

`"weights"` `unit`, `weight`; SDID fits add a `panel` column (`"omega"` unit weights, `"lambda"` time weights), with `unit` holding the pre-period label for `"lambda"` rows.

`"pred_weights"` `predictor`, `weight` (sharp SCM only).

`"gaps"` `time`, `gap`, `unit` (`NA` for the treated series), `series` (`"Treated"` / `"Placebo (donor pool)"`).

`"ratios"` `unit`, `ratio`, `post_mspe`, `series` – both candidate statistics, so the frame answers either `mspe_ratio_pval()` `statistic`.

See Also

`plot.coresynth()`, `plot.scm_placebo()`

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:10, year = 1:20)
panel$treated <- as.integer(panel$unit == 5 & panel$year > 15)
panel$gdp <- panel$unit + 0.5 * panel$year +
  rnorm(nrow(panel)) + 3 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")

head(plot_data(fit, type = "trend"))
plot_data(fit, type = "gap")
plot_data(fit, type = "weights")

# Relabel the simplified series names, then plot it yourself
df <- plot_data(fit, type = "trend")
df$series <- sub("Synthetic Control", "Synthetic Unit 5", df$series)

ggplot2::ggplot(df, ggplot2::aes(time, value, color = series)) +
  ggplot2::geom_line()
```

pred *Predictor Specification for SCM*

Description

Creates a single predictor specification for use in scm_fit() with method = "scm". Pass a list() of pred() calls as the predictors argument to define the full covariate matrix.

Usage

```
pred(vars, times, op = "mean")
```

Arguments

vars	Character vector of variable names. All variables share the same times window and op operator. Use separate pred() calls for variables with different time windows.
times	Numeric/integer vector of time values to aggregate over. Values are matched against the time index of the panel passed to scm_fit(), and are expected to be pre-treatment: a window naming times the panel does not carry silently aggregates the overlap only, and one reaching past the treatment date aggregates values the treatment has already moved, so scm_fit() warns about both.
op	Aggregation operator applied to each variable over times. One of "mean" (default), "median", or "sum".

Value

A pred_spec object (a named list with class "pred_spec").

See Also

scm_fit() for the predictors argument that consumes a list() of pred_spec objects.

Examples

```
# Three variables averaged over the same window
pred(c("lnincome", "retprice", "age15to24"), 1980:1988)

# Single variable at a specific year
pred("cigsale", 1975)

# Single variable averaged over a range
pred("beer", 1984:1988)

# Abadie, Diamond & Hainmueller (2010) California Prop 99 style: combine
# several covariates aggregated over different windows plus three outcome
# lags at specific years. The resulting list is passed to
# scm_fit(..., predictors = predictors).
```

```
predictors <- list(
  pred(c("lnincome", "retprice", "age15to24"), 1980:1988),
  pred("beer",    1984:1988),
  pred("cigsale", 1988),
  pred("cigsale", 1980),
  pred("cigsale", 1975)
)
predictors
```

`scm_balance_frontier` *Balance Possibility Frontier for Staggered SCM*

Description

Traces how the two pre-treatment imbalances of partially pooled SCM trade off as the pooling parameter `nu` varies: `q_pool`, the imbalance of the average of the treated units, which the aggregate ATT rests on, and `q_sep`, the root mean square of the per-block imbalances, which the block-level effects rest on. Both are the cohort-level measures of Ben-Michael, Feller & Rothstein (2022, Appendix A.2), in which a block enters through the sum over its treated units, so a block of n_g units counts n_g times towards `q_pool` and n_g^2 times towards `q_sep`. Supplying `weights` to `scm_fit()` replaces the unit count with the mass it gives them, and the frontier is traced on that same objective.

Usage

```
scm_balance_frontier(fit, nu_grid = seq(0, 1, length.out = 11L))
```

Arguments

`fit`	A staggered `coresynth` object from `scm_fit()` with `method = "scm"`.
`nu_grid`	Values of `nu` to trace.

Details

Ben-Michael, Feller & Rothstein (2022) call this plot "an important tool for using partially pooled SCM in practice" (S.4.2): the curve is strongly convex, so a little pooling usually buys a large reduction in one imbalance for a small rise in the other, and where that bargain stops being worth it is a judgement the data can inform but not settle. Their `nu` heuristic, which `scm_fit()` uses by default, is one point on this curve.

A fit made with `cov_method = "balance"` is traced on the objective that includes its covariate term, and the frontier reports that term's two measures in `q_cov_sep` and `q_cov_pool`. The plot still draws the outcome imbalances, which are then only part of what the fit trades off.

Value

A `data.frame` of class `scm_frontier`, one row per `nu`, with `nu`, `q_sep`, `q_pool`, their values normalised by the separate solution (`q_sep_rel`, `q_pool_rel`), `estimate`, and `heuristic` marking the row closest to the fit's own `nu`. A fit made with `cov_method = "balance"` trades off a covariate imbalance as well, so its frontier carries `q_cov_sep` and `q_cov_pool` too, and the two normalisation constants are then the *combined* measure of the paper's footnote 7 rather than the outcome imbalance alone – which is why `q_sep_rel` and `q_pool_rel` are not 1 at `nu = 0` there. Plot it with `plot()`.

References

Ben-Michael, E., Feller, A., & Rothstein, J. (2022). Synthetic controls with staggered adoption. *JRSS-B*, 84(2), 351-381.

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:15, year = 1:20)
adopt <- c(`1` = 12, `2` = 15, `3` = 17)
panel$treated <- as.integer(!is.na(adopt[as.character(panel$unit)]) &
  panel$year >= adopt[as.character(panel$unit)])
panel$treated[is.na(panel$treated)] <- 0L
panel$gdp <- panel$unit + 0.4 * panel$year +
  rnorm(nrow(panel)) + 2 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")
fr <- scm_balance_frontier(fit, nu_grid = c(0, 0.5, 1))
fr
```

`scm_design` *Experimental Synthetic Control Design*

Description

Selects which units to assign to the treatment arm (and which to the control arm) in a planned experiment, following Abadie and Zhao (2026). Both sets of units are chosen by minimising the distance between their weighted-average predictor vectors and the population-average predictor vector $\bar{X}$, so the resulting estimates are less susceptible to post-randomisation bias than pure random assignment.

Usage

```
scm_design(
  data,
  outcome,
  unit,
  time,
  T0,
  T_fit = NULL,
```

```
  m_min = 1L,
  m_max = 1L,
  f = NULL,
  predictors = NULL,
  design = c("base", "weakly_targeted", "unit_level"),
  beta = 1,
  xi = 1,
  alpha = 0.05,
  normalize = TRUE,
  max_subsets = 100000L
)
```

Arguments

`data`	Long-format data frame (one row per unit–time).
`outcome`	Name of the outcome column.
`unit`	Name of the unit identifier column.
`time`	Name of the time identifier column.
`T0`	Last pre-experimental period (a value present in the time column). Periods after `T0` are the experimental periods.
`T_fit`	Number of fitting periods, counted from the **start** of the pre-experimental phase. Defaults to `NULL`, which uses all pre-experimental periods for fitting (no blank periods; inference disabled). When `T_fit` is smaller than the total number of pre-experimental periods, the remaining periods become blank periods used for inference.
`m_min`	Minimum number of units assigned to treatment (default 1).
`m_max`	Maximum number of units assigned to treatment (default 1).
`f`	Named numeric vector of population weights f_j. Defaults to uniform weights $1/J$. Will be normalised to sum to 1.
`predictors`	A `list()` of `pred()` specifications that define the predictor matrix X_j. Defaults to `NULL`, which uses all fitting-period outcome values as predictors.
`design`	Design formulation: `"base"` (default), `"weakly_targeted"`, or `"unit_level"`.
`beta`	Trade-off parameter $\beta > 0$ for the Weakly targeted design (default 1).
`xi`	Trade-off parameter $\xi > 0$ for the Unit-level design (default 1).
`alpha`	Significance level for confidence intervals (default 0.05).
`normalize`	If `TRUE` (default), each row of the predictor matrix is divided by its cross-unit standard deviation before optimisation, so predictors measured on different scales contribute equally.
`max_subsets`	Maximum number of treatment-set candidates to evaluate before switching to random sampling (default 100 000).

Details

Three design formulations are available:

- `"base"` (eq. 7): both the synthetic treated and the synthetic control independently target the population average $\bar{X}$.
- `"weakly_targeted"` (eq. 9): the treated and control weights jointly minimise the distance of the synthetic treated to $\bar{X}$ plus `beta` times the distance between the synthetic control and the synthetic treated.
- `"unit_level"` (eq. 10): each treated unit gets its own synthetic control; the treated weights trade off matching $\bar{X}$ against `xi` times each treated unit's own synthetic-control fit, and the aggregate control weight is the `w`-weighted combination of the per-unit controls (eq. 11).

Inference uses "blank periods" — pre-experimental periods whose outcomes were *not* used to estimate the weights. Set `T_fit` strictly smaller than the number of pre-experimental periods to enable the permutation test and split- conformal confidence intervals from Section 3 of Abadie and Zhao (2026).

Value

An object of class `"scm_design"` with components:

- `treated_units`: unit identifiers selected for treatment
- `control_units`: unit identifiers in the control pool
- `w`: J-length weight vector for the synthetic treated unit (sums to 1)
- `v`: J-length weight vector for the synthetic control unit (sums to 1)
- `tau_hat`: estimated treatment effects for each experimental period
- `p_value`: permutation p-value (NA when blank periods are unavailable)
- `ci_lower`, `ci_upper`: per-period split-conformal confidence interval
- `Y_synth_tr`, `Y_synth_co`: synthetic treated/control series (all periods)
- `estimate`: ATT (mean of `tau_hat`)

References

Abadie, A. and Zhao, J. (2026). "Synthetic Controls for Experimental Design." MIT Working Paper.

`scm_fit` *Fit a Synthetic Control Method Model*

Description

Unified formula interface for Synthetic Control and related causal inference methods. The formula syntax is:

Usage

```
scm_fit(
  formula,
  data,
  method = c("scm", "sdid", "gsc", "mc", "tasc", "si"),
  ...,
  predictors = NULL,
  covariates = NULL,
  v_selection = c("insample", "oos"),
  donor_mspe_threshold = Inf,
  lambda_pen = NULL,
  v_optim = c("auto", "coord_descent", "bfgs", "multistart"),
  qp_solver = c("auto", "active_set", "wolfe"),
  v_window = NULL,
  nu = NULL,
  fixedeff = NULL,
  pool_cohorts = TRUE,
  weights = NULL
)
```

Arguments

formula	A Formula object, e.g. y ~ D | unit + time.
data	A data.frame in **long** format (one row per unit-time).
method	One of "scm", "sdid", "gsc", "mc", "tasc", "si".
...	Estimator-specific arguments forwarded to the chosen method: lambda plus cv_folds, cv_nlambda and cv_seed ("mc"), r ("gsc", "tasc"), em_iter and fix_A ("tasc"), zeta2 and zeta_t ("sdid"), k ("si"), control_group ("scm", "sdid", "gsc", "si"), and scale_predictors, cov_method and xi ("scm"). A name the chosen estimator does not take is an error rather than a silently ignored argument. Every argument after ... must be named in full: this is what lets lambda reach method = "mc" instead of partially matching lambda_pen. control_group chooses each cohort's donors in a **staggered** fit of "scm", "sdid", "gsc" or "si". "never_treated" (default) uses the units never treated in the panel, the pure controls of Clarke et al. (2024, Algorithm 1); at least two are required. "clean" is deprecated: it adds the units adopting after the cohort, but each cohort's effect runs to the end of the panel, so those units' post-adoption outcomes enter its counterfactual and bias the effect towards zero. "mc" and "tasc" impute from every untreated cell and need no never-treated units. cov_method chooses how covariates enter a **staggered** "scm" fit. "partial_out" (default) residualises the outcome on them first, the device of Clarke et al. (2024, S.2.2). "balance" is the formulation of Ben-Michael, Feller & Rothstein (2022, S.5.2, Equation 11): each covariate's pre-treatment average enters the partially pooled objective as a second imbalance measure, weighted against the lagged outcomes by xi. It requires the partially pooled solver, so it cannot be combined with nu = "separate", donor_mspe_threshold, lambda_pen or v_selection

= "oos". xi defaults to the paper's rule, the sample variance of the donor units' outcomes before the last adoption date.

Three tuning parameters that govern model complexity are selected from the data when left at their NULL default, each by the rule its source paper specifies: r for "gsc" by the leave-one-period-out cross-validation of Xu (2017, Algorithm 1), lambda for "mc" by the K-fold cross-validation of Athey et al. (2021), and k for "si" by the optimal hard threshold of Gavish & Donoho (2014). Supplying a value skips the search. The r of "tasc" is a different quantity – the dimension of the state vector its Kalman smoother carries, not a factor count estimated from a donor block – and keeps its fixed default of 2.

predictors A list() of pred() specifications that define the predictor matrix for SCM (see Abadie et al. 2010, S.2.3). Each pred() entry aggregates one or more variables over a time window. Pass NULL (default) to use all pre-treatment outcome periods as predictors; a one-line message states this default when it applies. A specification consisting solely of the outcome variable at each single pre-treatment period (one pred() per period, jointly covering the full pre-treatment window) defines the same predictor matrix and is therefore fitted through the same outcomes-only path as NULL, returning an identical fit; supply v_optim = "multistart" explicitly to force the predictor-path optimiser instead. That window is whatever data carries before the treatment date, not the span of the pred() list: adding earlier periods to data (to widen a plot, say) leaves the same list short of full coverage and moves the fit onto the predictor path, so a message reports the mismatch when it happens. Restricting the periods the fit is judged on is a separate choice from predictors – subset data, or keep data whole and pass v_window. pred() windows are expected to lie inside the pre-treatment periods present in data: times the panel does not carry, and times at or after the treatment date, each raise a warning because the row then aggregates something other than what the window reads as. Applies to method = "scm" only. Predictor rows are scaled by their standard deviation across all units before optimisation, matching the Synth reference implementation (ADH 2011, JSS); pass scale_predictors = FALSE to disable. When several units adopt at the same date they are pooled into one cohort, and each predictor row of the treated vector is their average under the same weights the outcome rows are pooled with, so both sides of the fit describe one target series.

covariates An optional named list of additional time-varying covariates to partial out before estimation. Each element is a character string naming a column in data. Supported for method = "sdid", "gsc", and **staggered** method = "scm". Partialling out is the device of Clarke et al. (2024, S.2.2), which SDID needs because it has no way to balance a covariate. SCM does: Abadie, Diamond & Hainmueller (2010, S.2.3) put covariates into the V-weighted predictor distance, and Ben-Michael, Feller & Rothstein (2022, S.5.2) add a separate balance term for them. A sharp SCM fit therefore rejects covariates and asks for predictors = list(pred(...)) instead; the staggered path borrows the partial-out only because predictors is unavailable there.

v_selection V matrix selection method for method = "scm". "insample" (default) follows Abadie et al. (2010): V is chosen by minimising in-sample pre-treatment MSPE. "oos" follows Abadie (2021) S.3.2 / ADH (2015): the pre-treatment window is split into a training half and a validation half. In the default outcomes-as-

predictors case, candidate W(V) are fitted on training-half outcomes only, V* minimises the validation-half MSPE, and the final W* is refit with V* on the outcomes of the last `floor(T_pre/2)` pre-treatment periods (so `v_weights` has `floor(T_pre/2)` entries). With user-supplied `predictors`, the predictor matrix is fixed and only the MSPE evaluation window is restricted to the validation half; lag your `pred()` windows to the training period for a fully out-of-sample exercise.

`donor_mspe_threshold`

Donor pool filtering threshold (Abadie 2021 S.4). For `method = "scm"` only. Each donor's individual pre-treatment MSPE (using that donor alone as the counterfactual) is divided by the minimum such MSPE across all donors. Donors whose ratio exceeds this threshold are excluded from estimation. `Inf` (default) disables filtering.

`lambda_pen` Penalised SCM parameter (Abadie & L'Hour 2021, JASA). For `method = "scm"` only. `NULL` (default) runs standard unpenalised SCM. `"auto"` selects the penalty via out-of-sample pre-treatment MSPE on the same validation window as `v_selection = "oos"`. A non-negative number uses that value directly.

`v_optim` Outer V-optimisation method for `method = "scm"`. The outer problem (choose V so that the implied donor weights W(V) minimise pre-treatment outcome MSPE) is non-convex, and a single local search can settle in a poor basin when `predictors` are supplied. `"auto"` (default) therefore selects `"multistart"` for predictor-based fits and `"coord_descent"` for outcomes-only fits (where the single start is empirically reliable). `"multistart"` runs a deterministic multi-start search: a fixed start set (uniform V, one-hot V per predictor, and 100 fixed-seed random draws) is screened at one inner QP each, the leaders are polished by coordinate descent, and the winner is refined by a Nelder-Mead pass. Its solution is never worse (in pre-treatment loss) than `"coord_descent"`, at roughly the cost of a handful of single-start fits, and is fully reproducible (no RNG state is consumed). `"coord_descent"` is the classic single-start coordinate descent with an 11-point grid, and is also the outcomes-only and staggered engine and the multi-start never-worse reference. `"bfgs"` (a single-start L-BFGS-B) is **deprecated** and will be removed in a future release: it has no advantage over `"multistart"` (which dominates it on predictor fits) or `"coord_descent"`. `mspe_ratio_pval()` mirrors a multi-start fit in its placebo refits so the permutation test stays symmetric.

`qp_solver` Inner-QP solver for `method = "scm"`. `"wolfe"` is the Wolfe (1976) min-norm-point method, which exploits the fact that the inner QP is a projection onto the convex hull of the donor columns in the k-dimensional predictor space: by Caratheodory's theorem an optimum supported on at most `k + 1` donors always exists, and Wolfe returns such a solution. `"active_set"` is the warm-started active-set method. Both return an exact optimum; they differ in *which* optimum when the QP is degenerate (fewer predictors than donors in the support), where the optimal set is a face rather than a point. `"wolfe"` is preferred there because its solution is sparse and reproducible, while `"active_set"` breaks those ties on round-off and can return different weights for the same fit under a different build. `"auto"` (default) is `"wolfe"` where that degeneracy can arise – sharp fits with `predictors`, and outcomes-only fits with fewer than half as many pre-treatment periods as donors – and `"active_set"` for the other outcomes-only fits, whose

QP has a unique optimum in practice, so both solvers return the same weights and the active set gets there several times faster. It is also `"active_set"` on the paths that run their own solver: `v_selection = "oos"`, `lambda_pen`, `v_optim = "bfgs"`, and staggered adoption. Naming `"wolfe"` explicitly on one of those paths is an error rather than a silent downgrade. The resolved value is recorded in `fit$qp_solver`.

`v_window` Optional vector of pre-treatment time values (matching the time index in `data`) over which the outer V optimisation evaluates the pre-treatment fit, for `method = "scm"` (sharp fits only). `NULL` (default) evaluates on all pre-treatment periods. The window restricts only the outer evaluation loss: predictor matrices (or the full pre-treatment outcome rows in the outcomes-only case) still enter the inner QP unchanged, and the reported `loss` and `mspe_ratio_pval()` MSPE components always cover the full pre-treatment window. Cannot be combined with `v_selection = "oos"`, which manages its own train/validation split. Use it to judge the fit on part of the pre-treatment period while keeping the rest of `data` available for plotting and post-estimation.

`nu` How strongly the adoption cohorts are pooled in **staggered** SCM fits (Ben-Michael, Feller & Rothstein 2022, JRSS-B). Cohort weight vectors are chosen jointly to minimise `nu * (normalised pooled pre-treatment imbalance)^2 + (1 - nu) * (normali` trading the fit of each cohort against the fit of their average, which is what the aggregate ATT rests on. `"auto"` (the default) is the paper's heuristic, the ratio of the pooled to the average per-cohort imbalance of the separate solution; it is also what `augsynth`, the authors' implementation, uses when `nu` is unset. A number in `[0, 1]` sets it directly, with `0` separate and `1` fully pooled – the paper advises against both extremes. `"separate"` is `0` reached through the older per-cohort path, which keeps the V optimisation and donor filtering that the pooled solver has no room for; it is also what the default falls back to when `donor_mspe_threshold`, `lambda_pen`, or `v_selection = "oos"` rules the pooled solver out. Balance diagnostics are stored in `fit$pooling`.

Staggered panels only. Under a single adoption date there are no cohorts to pool across, and several units sharing one date are already fully pooled into that cohort: S.4.1 of the paper shows the across-unit variation that would justify doing otherwise is zero when the adoption date is shared.

`fixedeff` Unpenalised additive level terms. For `method = "mc"` it estimates unit and time fixed effects outside the nuclear norm, as in Athey et al. (2021, S.8.1), and defaults to `TRUE`; `FALSE` recovers the S.4 baseline where the level is absorbed into the penalised matrix. For `method = "scm"` it defaults to `TRUE` on staggered panels, matching `augsynth` and the S.6 finding that the intercept-shifted estimator "dominates either of the alternatives"; it does not apply to sharp fits. If `TRUE`, staggered SCM demeans every unit by its own pre-treatment mean within each cohort before fitting (intercept shift; Ben-Michael, Feller & Rothstein 2022, Section 5.1), which turns the estimator into a weighted difference-in-differences and typically improves fit when outcome levels differ across units. The reported `Y_synth` is shifted back to the raw outcome scale. Works with both the default and the partially pooled path. For `method = "scm"` on staggered panels only.

`pool_cohorts` For **staggered** `method = "scm"`: whether treated units sharing an adoption date are averaged into one block before the weights are fitted. `TRUE` (default) is

the Appendix A.2 hierarchy of Ben-Michael, Feller & Rothstein (2022), whose S.4.1 shows there is no across-unit variation to exploit within a cohort. `FALSE` is their S.4 formulation, one weight vector per treated unit, which is what the rank-based permutation test in `scm_rank_pval()` needs. `augsynth` makes the unit-level form its default; the paper's own appendix does not. It applies to both staggered paths – the partially pooled solver and `nu = "separate"` – so it still takes effect where `donor_mspe_threshold`, `lambda_pen` or `v_selection = "oos"` have sent the fit down the latter. Sharp fits have a single cohort and reject it.

`weights` Optional named numeric vector over the treated units giving the mass each carries in the aggregate ATT, for `method = "scm"`. `NULL` (default) weights them equally, which for a shared adoption date is the `N_treated * T_post` weighting used throughout. Abadie (2021, S.8) notes that with regions or countries a population- or GDP-weighted average effect is often the quantity of interest; supply those figures here. The mass is used consistently: a block's target series is the correspondingly weighted average of its treated units, that same mass is what the block counts for in the partially pooled objective (in place of the cohort size of Ben-Michael, Feller & Rothstein 2022, Appendix A.2) and in its `T_post`-multiplied aggregation weight, and `scm_inference()` resamples each unit at its own mass. Only the ratios matter – the vector is rescaled internally – so equal weights reproduce the unweighted fit exactly.

Details

`outcome ~ treatment | unit_id + time_id`

Value

An object of classes `c("coresynth_<method>", "coresynth")`. Fits with staggered adoption additionally inherit from `"coresynth_staggered"`, and multi-arm SI fits from `"coresynth_multiarm"`; S3 methods such as `tidy()` and `augment()` dispatch on these subclasses. All methods return at minimum:

- `method`: estimator name
- `estimate`: average treatment effect (ATT)
- `times`: time index vector
- `T_pre`: number of pre-treatment periods
- `Y_treat`: treated unit outcome series
- `gap`: treatment effect series (Y_treat - counterfactual)

Staggered `method = "scm"` fits add `cohort_estimates`, `cohort_fits`, and the settings the fit resolved: `nu` (the number the partially pooled solver used, or `"separate"` for the per-cohort path), `fixedeff`, `pool_cohorts` and `qp_solver`.

Examples

```
# Synthetic balanced panel: 10 units over 20 periods, unit 1 treated
# after period 15.
```

```
set.seed(1)
panel <- expand.grid(unit = 1:10, year = 1:20)
panel$treated <- as.integer(panel$unit == 1 & panel$year > 15)
panel$gdp <- panel$unit + 0.5 * panel$year +
  rnorm(nrow(panel)) + 3 * panel$treated

fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "sdid")
summary(fit)


# Visualise the estimated gap (requires ggplot2)
plot(fit, type = "gap")
```

`scm_inference` *Inference for Staggered SCM*

Description

Standard errors, confidence intervals and p-values for the aggregate ATT of a staggered SCM fit, by the two procedures of Ben-Michael, Feller & Rothstein (2022).

Usage

```
scm_inference(
  fit,
  method = c("jackknife", "wild_bootstrap"),
  n_boot = 1000L,
  level = 0.95,
  alternative = c("two.sided", "greater", "less"),
  seed = NULL
)
```

Arguments

`fit`	A staggered SCM fit from `scm_fit()` (`method = "scm"` on a panel with multiple adoption times). Sharp fits are rejected: use `mspe_ratio_pval()` or `conformal_inference()` instead.
`method`	`"jackknife"` (default) or `"wild_bootstrap"`.
`n_boot`	Number of bootstrap draws (`"wild_bootstrap"` only). Default 1000.
`level`	Confidence level. Default 0.95.
`alternative`	Direction of the alternative hypothesis for the p-value: `"two.sided"` (default), `"greater"`, or `"less"`.
`seed`	Optional RNG seed (`"wild_bootstrap"` only).

Details

`method = "jackknife"` (default) is the leave-one-unit-out jackknife of their Appendix A.1: every unit in the panel – treated units and donors alike – is dropped in turn, the weights (and intercepts, for `fixedeff` fits) are refitted with the pooling parameter `nu` held at the fit's value, and the aggregate ATT is recomputed. The variance is $(n-1)/n \sum_i (\widehat{ATT}^{(-i)} - \overline{ATT})^2$, and the interval and p-value use the normal approximation. Because the donors are resampled too, the interval reflects the noise in the synthetic controls, which every treated unit of a cohort shares.

`method = "wild_bootstrap"` is the multiplier bootstrap of their Section 5.3, which adapts Otsu & Rai (2017). The ATT is written as a sum over every unit of the panel, and each unit's term is perturbed by its own golden-ratio two-point multiplier while the weights stay fixed: a treated unit's term is its effect's deviation from the ATT plus its residual, a donor's is its residual times the weight the synthetic controls put on it. Residuals are against a two-way (unit and period) model of the untreated outcome. Following Otsu & Rai's Equation (9), only the treated units' heterogeneity term is centred at the ATT; centring every unit, as the paper's Equation (13) reads, would make the interval grow with the size of the effect. The draws cost no refitting, so this is the choice for panels too large for the jackknife. It is conservative where the outcome has structure the two-way model does not absorb (unit-specific trends): in simulation a nominal 95% interval covered 96-99.5% across donor pools of 8 to 40 and 3 to 24 treated units, and 93% with 2 treated units.

The jackknife refits the model once per unit, which grows quickly with the panel: about 0.2 s at 50 units, 0.7 s at 100 and 3 s at 200, but over a minute at 400. Donors that carry no weight in any block are not refitted (dropping one leaves the solution unchanged), and each refit starts from the full fit's weights.

Both work with both staggered SCM paths (the partially pooled default and `nu = "separate"`) and honour the aggregation weights `N_treated x T_post`, or the `weights` given to `scm_fit()`.

The jackknife treats the outcome as fixed: a fit with `covariates` partialled out keeps the partialled-out outcome, and one with `cov_method = "balance"` keeps its scaled covariate matrix, rather than re-estimating either without the dropped unit. A dropped unit that was the only member of its cohort removes that cohort from the estimate, as in the paper's $J^{(-i)}$.

Value

A `coresynth_inference` object with the standard fields (`estimate`, `se`, `p_value`, `ci_lower`, `ci_upper`, `method`, `staggered`, `n_controls`, `alternative`), compatible with `tidy.coresynth_inference()` and `glance.coresynth_inference()`. `n_treated` records the number of treated units. The wild bootstrap adds the draws in `boot_ests`, centred on the estimate; the jackknife adds the leave-one-out estimates in `jack_ests` (named by the dropped unit, `NA` where a refit was not possible).

References

Ben-Michael, E., Feller, A., & Rothstein, J. (2022). Synthetic controls with staggered adoption. *JRSS-B*, 84(2), 351-381.

Otsu, T., & Rai, Y. (2017). Bootstrap inference of matching estimators for average treatment effects. *JASA*, 112(520), 1720-1732.

Examples

```
set.seed(1)
```

```
dat <- expand.grid(time = 1:20, id = paste0("u", 1:12))
dat$y <- rnorm(nrow(dat)) + as.numeric(factor(dat$id))
dat$d <- as.integer(
  (dat$id %in% c("u1", "u2") & dat$time > 10) |
  (dat$id %in% c("u3", "u4") & dat$time > 14)
)
fit <- scm_fit(y ~ d | id + time, data = dat, method = "scm")
scm_inference(fit)
```

`scm_inner_weights_cpp` *SCM Inner Weights (QP Given V)*

Description

Solves the inner-loop QP for SCM: given a fixed diagonal metric matrix V, finds donor weights W on the simplex minimising the V-weighted covariate loss. The returned weights are a KKT-verified exact optimum whenever the active-set solver converges, with accelerated projected gradient as a fallback.

Usage

```
scm_inner_weights_cpp(X0, X1, V_diag, wolfe = FALSE)
```

Arguments

`X0`	Covariate matrix for control units (k x N_co)
`X1`	Covariate vector for the treated unit (k x 1)
`V_diag`	Diagonal of the metric matrix V (k x 1, non-negative, need not sum to 1)
`wolfe`	If `TRUE`, solve with the Wolfe min-norm-point method, which returns a Caratheodory-sparse optimum (at most k+1 donors carry weight) instead of one arbitrary point of a degenerate optimal face. `FALSE` (default) uses the warm-started active-set solver.

Value

Donor weight vector W (N_co x 1) on the unit simplex

scm_placebo_cpp *Fast Leave-One-Out Placebo Test for SCM (Abadie et al. 2010)*

Description

For each control unit, treats it as pseudo-treated and fits SCM weights from the remaining N_co-1 donors. Returns MSPE components for constructing MSPE-ratio permutation p-values in R.

Usage

```
scm_placebo_cpp(
  Y_pre,
  Y_post,
  max_iter = 100L,
  tol = 1e-04,
  z_rows = NULL,
  wolfe = FALSE
)
```

Arguments

Y_pre	Control pre-treatment outcomes (T_pre x N_co)
Y_post	Control post-treatment outcomes (T_post x N_co)
max_iter	Outer coordinate-descent iterations (default 100)
tol	Convergence tolerance for V updates (default 1e-4)
z_rows	Optional 1-based pre-period row indices of the outer evaluation window (the v_window of the treated fit), so each placebo refit optimises V on the same window. NULL (default) uses all rows. MSPE components are always computed on the full pre/post windows.
wolfe	If TRUE, each placebo refit uses the Wolfe min-norm-point inner solver, matching a treated fit made with qp_solver = "wolfe" so the permutation stays symmetric.

Value

A list with:

- mspe_pre: N_co-vector of pre-treatment MSPE per placebo unit
- mspe_post: N_co-vector of post-treatment MSPE per placebo unit
- effects: N_co-vector of mean post-period gap per placebo unit
- gaps: (T_pre + T_post) x N_co matrix of placebo gap paths

scm_placebo_x_cpp *Fast Leave-One-Out Placebo Test for SCM with a Predictor Specification*

Description

Covariate-spec counterpart of scm_placebo_cpp(): for each control unit, treats it as pseudo-treated with its own predictor column X0[, i] and fits the nested V/W optimisation against the remaining donors' predictors X0[, -i], evaluating the prediction loss on pre-treatment outcomes. Each leave-one-out problem is identical to a scm_weights_cpp() call on the same submatrices; iterations are independent and run in parallel under OpenMP.

Usage

```
scm_placebo_x_cpp(
  X0,
  Y_pre,
  Y_post,
  max_iter = 100L,
  tol = 1e-04,
  z_rows = NULL,
  multistart = FALSE,
  wolfe = FALSE
)
```

Arguments

X0	Predictor matrix for control units (k x N_co), on the same scale as the treated fit (SD-scaled when scale_predictors = TRUE)
Y_pre	Control pre-treatment outcomes (T_pre x N_co)
Y_post	Control post-treatment outcomes (T_post x N_co)
max_iter	Outer coordinate-descent iterations (default 100)
tol	Convergence tolerance for V updates (default 1e-4)
z_rows	Optional 1-based pre-period row indices of the outer evaluation window (the v_window of the treated fit), so each placebo refit optimises V on the same window. NULL (default) uses all rows. MSPE components are always computed on the full pre/post windows.
multistart	If TRUE, each placebo refit uses the same deterministic multi-start outer search as the treated fit, keeping the permutation test symmetric.
wolfe	If TRUE, each placebo refit uses the Wolfe min-norm-point inner solver, matching a treated fit made with qp_solver = "wolfe".

Value

A list with:

- mspe_pre: N_co-vector of pre-treatment MSPE per placebo unit
- mspe_post: N_co-vector of post-treatment MSPE per placebo unit
- effects: N_co-vector of mean post-period gap per placebo unit
- gaps: (T_pre + T_post) x N_co matrix of placebo gap paths A placebo unit whose solver fails yields NaN entries.

---

scm_rank_pval *Rank-Based Permutation Test for Several Treated Units*

Description

Permutation inference on the aggregate effect when more than one unit is treated, following Abadie & L'Hour (2021, S.3.2). Each treated unit contributes its own test statistic; the identities of the treated units are then reassigned at random among all units, and the sum of the ranks of the real treated units' statistics is compared with its permutation distribution.

Usage

```
scm_rank_pval(
  fit,
  B = 500L,
  statistic = c("mspe_ratio", "effect", "abs_effect"),
  alternative = c("two.sided", "greater", "less"),
  seed = 1L
)
```

Arguments

fit	A coresynth object from scm_fit() with method = "scm" and more than one treated unit.
B	Number of treatment reassignments.
statistic	Per-unit statistic to rank. "mspe_ratio" (default) is the post/pre MSPE ratio of Abadie, Diamond & Hainmueller (2010), which adjusts for how well each unit was fitted; "effect" is the mean post-treatment gap and "abs_effect" its absolute value.
alternative	"two.sided" (default), "greater", or "less".
seed	Seed for the reassignment draw, so the p-value is reproducible without consuming the caller's RNG state.

Details

Ranking is what makes the test usable here. `mspe_ratio_pval()` compares one treated unit against placebo runs on single donors, which breaks down as soon as the treated series is an average: the average is less noisy than any single donor, so its pre/post MSPE ratio is inflated and the test rejects far too often. Ranks sidestep that, because every statistic entering the comparison, real or permuted, is computed on a single unit.

The fit must expose per-unit effects, so a staggered fit has to be made with `pool_cohorts = FALSE`. A sharp fit with several treated units pools them by design, and its per-unit statistics are computed here from the stored per-unit series.

Value

An object of class `c("scm_rank_test", "coresynth_inference")` with `p_value`, `statistic` (the observed rank sum), `rank_sums` (its permutation distribution), `unit_statistics`, `n_treated`, `B` and `alternative`.

References

Abadie, A., & L'Hour, J. (2021). A penalized synthetic control estimator for disaggregated data. *JASA*, 116(536), 1817-1834.

See Also

`mspe_ratio_pval()` for the single-treated-unit test.

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:12, year = 1:20)
panel$treated <- as.integer(panel$unit <= 3 & panel$year > 12)
panel$gdp <- panel$unit + 0.4 * panel$year +
  rnorm(nrow(panel)) + 2 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")
scm_rank_pval(fit, B = 50)
```

---

scm_weights_cpp *SCM Outer Weights (Joint Optimization of W and V)*

Description

Jointly optimises donor weights W (on the simplex) and the diagonal metric matrix V via coordinate descent on the pre-treatment prediction MSPE, following Abadie, Diamond & Hainmueller (2010).

Usage

```
scm_weights_cpp(
  X0,
  X1,
  Z0,
  Z1,
  max_iter = 100L,
  tol = 1e-04,
  t_train = -1L,
  z_rows = NULL,
  multistart = FALSE,
  cheap_face = FALSE,
  wolfe = FALSE
)
```

Arguments

X0	Covariate matrix for control units (k x N_co, typically pre-treatment outcomes)
X1	Covariate vector for the treated unit (k x 1)
Z0	Outcome matrix for control units in the pre-treatment window (T_pre x N_co)
Z1	Outcome vector for the treated unit in the pre-treatment window (T_pre x 1)
max_iter	Maximum coordinate-descent iterations (default 100)
tol	Convergence tolerance on MSPE improvement (default 1e-4)
t_train	Validation-window split for V selection. -1 (default): V selected on the full Z window (in-sample). Positive: rows t_train..(T_pre-1) of Z form the validation window used to select V (W is fitted on the full X throughout); after selecting V*, W is refit and the reported loss uses the full Z window.
z_rows	Optional 1-based row indices of Z defining the evaluation window for the outer V optimisation (the v_window argument of scm_fit()). NULL (default) evaluates on the full Z window. Takes precedence over t_train; the reported loss always uses the full Z window.
multistart	If TRUE, the outer V optimisation runs a deterministic multi-start search (screened start set, coordinate-descent polish, Nelder-Mead refinement) instead of a single coordinate-descent pass from the uniform V. The result is never worse (in outer loss) than the single-start path.
cheap_face	If TRUE, the inner simplex QP first tries a cheap bordered-KKT direct solve on each active-set face, falling back to the scale-robust null-space solve when that system is singular. Worth trying only in the outcomes-only regime (no user predictors, no predictor rescaling), where V is dense and the faces that carry the fit are well conditioned; it reproduces the null-space solution to round-off. FALSE (default) goes straight to the null-space solve, which is what predictor fits need for scale invariance and rank-deficient faces.
wolfe	If TRUE, every inner QP is solved with the Wolfe min-norm-point method, which returns a Caratheodory-sparse optimum (at most k+1 donors carry weight) instead of one arbitrary point of a degenerate optimal face. FALSE (default) uses the warm-started active-set solver.

Details

When `t_train > 0`, V is selected by minimising MSPE on a validation window (rows t_train..T_pre-1 of Z) while W is fitted on the full predictor matrix X. This is appropriate when X is a fixed predictor matrix that contains no validation-period outcome information (the user-supplied predictors case). For the outcomes-only case the proper Abadie (2021) S.3.2 train/validation split is implemented in R (`.scm_oos_outcomes()`): candidate W(V) are fitted on training-half outcomes only, by passing the training rows as X and the validation rows as Z to this function with `t_train = -1`.

Value

A list with:

- `W`: Donor weight vector (N_co x 1) on the unit simplex
- `V`: Optimal metric diagonal (k x 1, normalised to sum to 1)
- `loss`: Final pre-treatment prediction loss (full pre-treatment window)

`sdid_estimate_cpp` *Calculate SDID Estimate (tau_sdid)*

Description

Given unit weights omega and time weights lambda, computes the SDID estimator as a weighted two-way difference:

Usage

```
sdid_estimate_cpp(Y_pre_co, Y_post_co, Y_pre_tr, Y_post_tr, omega, lambda)
```

Arguments

`Y_pre_co`	Control pre-treatment outcomes (T_pre x N_co)
`Y_post_co`	Control post-treatment outcomes (T_post x N_co)
`Y_pre_tr`	Treated pre-treatment outcomes (T_pre x 1)
`Y_post_tr`	Treated post-treatment outcomes (T_post x 1)
`omega`	Unit weights (N_co x 1)
`lambda`	Time weights (T_pre x 1)

Details

tau_sdid = (Y_tr_post_mean - Y_tr_pre_wt) - (Y_co_post_wt - Y_co_pre_wt)

Value

A single numeric value: the SDID treatment-effect estimate `tau_sdid`.

sdid_inference *Inference for Synthetic Difference-in-Differences*

Description

Computes standard errors and p-values for a SDID estimate using one of four methods, following Clarke et al. (2024): permutation placebo test (Algorithm 4), cluster bootstrap (Algorithm 2), leave-one-out jackknife (Algorithm 3), or (staggered fits only) a global jackknife across the unique control units of all cohorts.

Usage

```
sdid_inference(
  fit,
  method = c("placebo", "bootstrap", "jackknife", "jackknife_global"),
  n_boot = 200L,
  level = 0.95,
  alternative = c("two.sided", "greater", "less"),
  seed = NULL
)
```

Arguments

fit	A coresynth object with method = "sdid" (sharp or staggered).
method	Inference method: "placebo" (permutation), "bootstrap", "jackknife", or "jackknife_global" (staggered only).
n_boot	Number of bootstrap replications (only for method = "bootstrap").
level	Confidence level for the interval (all methods).
alternative	Direction of the alternative hypothesis: "two.sided", "greater", or "less".
seed	Integer seed for reproducibility (only for method = "bootstrap").

Details

For method = "placebo", the p-value is the permutation p-value, while the standard error is the dispersion of the placebo distribution (Clarke et al. 2024, Algorithm 4) and the confidence interval is the normal approximation around the estimate with that SE. The placebo SE assumes the treated unit's noise is comparable to the control units'; interpret it with caution when the donor pool is small.

Value

A list with:

- estimate: The SDID point estimate.
- se: Standard error (placebo: placebo-distribution SD, Algorithm 4).
- p_value: Permutation or normal-approximation p-value.

- `ci_lower`, `ci_upper`: Confidence interval bounds.
- `method`: The inference method used.
- `n_controls`: Number of control units.
- `alternative`: The alternative hypothesis direction.
- `placebo_effects`: Named vector of LOO placebo effects (placebo only).
- `boot_ests`: Bootstrap estimate distribution (bootstrap only).

`sdid_placebo_cpp` *Fast Placebo Test for SDID*

Description

For each control unit, treats it as the "pseudo-treated" unit and estimates the leave-one-out SDID effect. The distribution of these placebo effects provides a permutation-based null distribution for inference.

Usage

```
sdid_placebo_cpp(Y_pre, Y_post, time_weights, zeta2)
```

Arguments

`Y_pre`	Control units pre-treatment outcomes (T_pre x N_co)
`Y_post`	Control units post-treatment outcomes (T_post x N_co)
`time_weights`	Lambda weights for pre-treatment periods (T_pre x 1)
`zeta2`	Ridge penalty (same as used in the main estimate)

Value

A numeric vector of length `N_co`. Each element is the leave-one-out placebo SDID effect obtained by treating that control unit as the pseudo-treated unit; the vector serves as a permutation-based null distribution for inference.

sdid_time_weights_cpp *Calculate SDID Time Weights (lambda)*

Description

Solves the time-weight QP (with implicit intercept lambda_0 concentrated out):

Usage

```
sdid_time_weights_cpp(Y_pre_co, Y_post_target, zeta_t)
```

Arguments

Y_pre_co	Pre-treatment outcomes for control units, row-demeaned (T_pre x N_co)
Y_post_target	Post-treatment mean per control unit, demeaned (N_co x 1)
zeta_t	Ridge penalty for time weights (paper: 1e-6 * sigma_hat)

Details

min over lambda in Delta_pre: ||Y_post_target - Y_pre_co^T lambda||^2 + zeta_t^2 * N_co * |||lambda||^2

The caller is responsible for pre-demeaning Y_pre_co (row-wise) and Y_post_target (subtract the cross-unit mean) to concentrate out lambda_0, as described in Arkhangelsky et al. (2021) Algorithm 1, Eq. (2.3).

Value

A numeric vector of length T_pre holding the SDID time weights lambda (non-negative and summing to one).

sdid_unit_weights_cpp *Calculate SDID Unit Weights (omega)*

Description

Solves the regularized QP: min over omega in Delta: sum_t (sum_i omega_i Y_it - Y_tr_t)^2 + zeta^2 * T_pre * ||omega||^2

Usage

```
sdid_unit_weights_cpp(Y_pre, Y_tr_pre, zeta2)
```

Arguments

Y_pre	Pre-treatment outcome matrix for control units (T_pre x N_co)
Y_tr_pre	Pre-treatment outcome vector for treated unit (T_pre x 1), averaged if multiple
zeta2	Ridge penalty parameter (zeta^2). The code internally multiplies by T_pre per the paper.

Details

This corresponds to equation (5) in Arkhangelsky et al. (2021).

Value

A numeric vector of length N_co holding the SDID unit weights omega (non-negative and summing to one).

---

si_inference *Non-parametric Inference for SI (Agarwal et al. 2025)*

Description

Estimates SE and confidence intervals for the ATT via non-parametric cluster bootstrap or jackknife over control units. Works for both sharp and staggered SI fits. For staggered fits, bootstrap resamples each cohort's control pool independently, and jackknife uses a per-cohort LOO with delta-method variance aggregation.

Usage

```
si_inference(
  fit,
  method = c("bootstrap", "jackknife", "jackknife_global"),
  n_boot = 499L,
  level = 0.95,
  alternative = c("two.sided", "greater", "less"),
  seed = NULL
)
```

Arguments

fit	A coresynth object from scm_fit() with method = "si".
method	"bootstrap" (default) or "jackknife".
n_boot	Number of bootstrap replications (default 499L; ignored for jackknife).
level	Confidence level (default 0.95).
alternative	"two.sided" (default), "greater", or "less".
seed	RNG seed for reproducibility (default NULL).

Value

A list of class `coresynth_inference`.

`si_pcr_cpp` *SI-PCR: Synthetic Interventions via Principal Component Regression*

Description

Implements the SI-PCR estimator of Agarwal et al. (2025). Uses the top-k SVD of pre-treatment control outcomes to find donor weights that predict each treated unit's pre-treatment trajectory, then applies those weights to post-treatment control outcomes.

Usage

```
si_pcr_cpp(Y_pre_co, Y_post_co, Y_pre_tr, k)
```

Arguments

`Y_pre_co`	Pre-treatment control outcomes (T_pre x N_co)
`Y_post_co`	Post-treatment control outcomes (T_post x N_co)
`Y_pre_tr`	Pre-treatment treated outcomes (T_pre x N_tr)
`k`	Number of SVD components to retain

Value

A list with:

- `W`: Donor weight matrix (N_co x N_tr)
- `Y_hat`: Counterfactual post-treatment outcomes (T_post x N_tr)

`soft_impute_cpp` *Fast Matrix Completion using Soft-Impute Algorithm*

Description

Solves: min_L (1/2) ||O o (Y - L)||*F*^2 + *lambda* * ||*L*||* via iterative SVD soft-thresholding (Mazumder, Hastie, Tibshirani 2010). Note: lambda is NOT normalized by |O|.

Usage

```
soft_impute_cpp(Y, O, lambda, max_iter = 1000L, tol = 1e-05)
```

Arguments

Y	Observed outcome matrix (T x N). Unobserved entries should be 0.
O	Binary mask matrix (T x N): 1 = observed, 0 = missing (treated post).
lambda	Nuclear norm penalty (soft-threshold on singular values).
max_iter	Maximum iterations.
tol	Convergence tolerance (relative Frobenius norm change).

Details

This is soft_impute_fe_cpp() with the fixed effects switched off.

Value

A numeric matrix of the same dimension as Y (T x N): the completed low-rank matrix L that minimises the soft-thresholded nuclear-norm objective.

---

soft_impute_fe_cpp *Matrix Completion with Optional Two-Way Fixed Effects*

Description

Solves min over L, a, b of (1/2) * squaredFrobenius(O o (Y - L - a1' - 1b')) + lambda * nuclearNorm(L) by alternating a back-fitting update of the fixed effects with one soft-thresholded SVD step for L (Mazumder, Hastie & Tibshirani 2010).

Usage

```
soft_impute_fe_cpp(
  Y,
  O,
  lambda,
  estimate_fe = TRUE,
  max_iter = 1000L,
  tol = 1e-05,
  fe_sweeps = 5L
)
```

Arguments

Y	Observed outcome matrix (T x N). Unobserved entries should be 0.
O	Binary mask matrix (T x N): 1 = observed, 0 = missing (treated post).
lambda	Nuclear norm penalty (soft-threshold on singular values).
estimate_fe	Estimate unpenalised row and column effects.
max_iter	Maximum iterations.
tol	Convergence tolerance (relative Frobenius norm change).
fe_sweeps	Back-fitting sweeps per iteration. The mask makes the two effects non-orthogonal, so one closed-form pass does not solve for both.

Details

The row and column effects are the unpenalised fixed effects of Athey et al. (2021, S.8.1): "we regularize the estimates of L*, but do not wish to regularize the estimates of the fixed effects". Leaving them inside `L` shrinks the panel's level towards zero along with its factor structure, which biases the imputed counterfactual whenever the level is far from zero. `estimate_fe = false` reproduces the S.4 baseline model `Y = L + e`.

Both effects absorb an additive constant, so `a` and `b` are identified only up to a shift between them; their sum, which is what enters the completed matrix, is unaffected.

Value

A list with `L` (the penalised low-rank part), `a` (row effects), `b` (column effects), and `M`, the completed matrix `L + a 1' + 1 b'`.

`tensor_unfold_cpp` *Tensor Unfolding (Matricization) for Synthetic Interventions*

Description

Tensor Unfolding (Matricization) for Synthetic Interventions

Usage

```
tensor_unfold_cpp(T_cube, mode)
```

Arguments

`T_cube`	A 3D array (cube) of dimensions (n1, n2, n3)
`mode`	The mode to unfold along (1, 2, or 3)

Value

A numeric matrix: the mode-`mode` unfolding (matricization) of `T_cube`, with dimensions `n1 x (n2 * n3)`, `n2 x (n1 * n3)`, or `n3 x (n1 * n2)` for `mode` 1, 2, or 3 respectively.

tidy.coresynth_inference *Tidy an inference result*

Description

Coerces a `coresynth_inference` or `sdid_inference` object to a one-row tidy `data.frame` with broom-style column names so it can be combined with regression output for paper tables.

Usage

```
## S3 method for class 'coresynth_inference'
tidy(x, conf.int = TRUE, ...)
```

Arguments

`x`	A `coresynth_inference` (or `sdid_inference`) object returned by `sdid_inference()`, `gsc_inference()`, or `si_inference()`.
`conf.int`	Logical. Include `conf.low`/`conf.high` columns when CI is available (default `TRUE`). Methods without a CI report `NA`.
`...`	Unused.

Value

A one-row `data.frame` with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, `method`, `alternative`, `n_controls`, `staggered`.

treated_outcomes *Extract Outcome Series from a coresynth Fit*

Description

Accessor generics that return the outcome series stored in a fitted `coresynth` object under a uniform interface, regardless of the estimation method:

Usage

```
treated_outcomes(x, ...)

## S3 method for class 'coresynth'
treated_outcomes(x, na.rm = FALSE, ...)

synthetic_outcomes(x, ...)

## S3 method for class 'coresynth'
```

```
synthetic_outcomes(x, na.rm = FALSE, ...)

## S3 method for class 'coresynth_staggered'
synthetic_outcomes(x, na.rm = FALSE, ...)

## S3 method for class 'coresynth_tasc'
synthetic_outcomes(x, na.rm = FALSE, ...)

donor_outcomes(x, ...)

## S3 method for class 'coresynth_scm'
donor_outcomes(x, ...)

## S3 method for class 'coresynth_sdid'
donor_outcomes(x, ...)

## S3 method for class 'coresynth_si'
donor_outcomes(x, ...)

## S3 method for class 'coresynth_gsc'
donor_outcomes(x, ...)

## S3 method for class 'coresynth_mc'
donor_outcomes(x, ...)

## S3 method for class 'coresynth_tasc'
donor_outcomes(x, ...)

## S3 method for class 'coresynth'
donor_outcomes(x, ...)
```

Arguments

x	A coresynth object from scm_fit().
...	Passed to methods.
na.rm	Logical; passed to the per-period averaging over multiple treated units (default FALSE).

Details

- treated_outcomes(): the treated unit's observed outcome series (length T). When several units are treated it is their per-period average – weighted by weights where a sharp fit was given them, and unweighted across cohorts for a staggered fit, whose treated units do not share a treatment date.
- synthetic_outcomes(): the estimated counterfactual series (length T), i.e. the synthetic control or model-fitted outcome.
- donor_outcomes(): the $T \times N_{co}$ matrix of observed donor (control unit) outcomes over all periods.

Each accessor returns NULL when the requested series is not stored in the fit. Staggered-adoption fits keep their counterfactual per cohort (in fit$cohort_fits), because collapsing it to one series would average cohorts whose treatment starts at different times, so synthetic_outcomes() returns NULL for them; use augment() for the per-cohort series.

Value

For treated_outcomes() and synthetic_outcomes(), a numeric vector of length T, or NULL. For donor_outcomes(), a $T \times N_{co}$ numeric matrix (donors in columns, named when unit names are available), or NULL.

Examples

```
set.seed(1)
panel <- expand.grid(unit = 1:10, year = 1:20)
panel$treated <- as.integer(panel$unit == 1 & panel$year > 15)
panel$gdp <- panel$unit + 0.5 * panel$year +
  rnorm(nrow(panel)) + 3 * panel$treated
fit <- scm_fit(gdp ~ treated | unit + year, data = panel, method = "scm")

y1   <- treated_outcomes(fit)    # observed treated series
y1_0 <- synthetic_outcomes(fit)  # synthetic counterfactual
Yco  <- donor_outcomes(fit)      # donor outcome matrix
all.equal(y1 - y1_0, unname(fit$gap))
```

Index

augment(), *21*, *32*, *51*
augment_scm, 3

conformal_inference, 4
conformal_inference(), *33*

donor_outcomes *(*treated_outcomes*)*, 49

export_json, 5

ggplot2::geom_hline(), *15*, *19*
ggplot2::geom_vline(), *15*, *19*
glance(), *5*
glance.coresynth_inference, 6
glance.coresynth_inference(), *34*
gsc_boot, 6
gsc_boot(), *5*
gsc_ife_cpp, 7
gsc_inference, 8
gsc_inference(), *49*

kalman_smoother_cpp, 9

loo_donors, 10
loo_donors(), *13*

mspe_ratio_pval, 11
mspe_ratio_pval(), *5*, *10*, *13*, *18–22*, *30*, *31*, *33*, *39*

placebo_in_time, 13
placebo_in_time(), *10*
plot(), *20*, *22*, *25*
plot.coresynth, 14
plot.coresynth(), *21*, *22*
plot.scm_design, 16
plot.scm_frontier, 17
plot.scm_placebo, 18
plot.scm_placebo(), *12*, *21*, *22*
plot_data, 20
pred, 23
pred(), *26*, *29*, *30*

scm_balance_frontier, 24
scm_balance_frontier(), *17*, *18*
scm_design, 25
scm_fit, 27
scm_fit(), *3–6*, *8*, *10*, *11*, *13*, *21*, *23*, *24*, *33*, *34*, *38*, *40*, *45*, *50*
scm_inference, 33
scm_inference(), *32*
scm_inner_weights_cpp, 35
scm_placebo_cpp, 36
scm_placebo_cpp(), *11*, *37*
scm_placebo_x_cpp, 37
scm_placebo_x_cpp(), *11*
scm_rank_pval, 38
scm_rank_pval(), *32*
scm_weights_cpp, 39
scm_weights_cpp(), *37*
sdid_estimate_cpp, 41
sdid_inference, 42
sdid_inference(), *49*
sdid_placebo_cpp, 43
sdid_time_weights_cpp, 44
sdid_unit_weights_cpp, 44
si_inference, 45
si_inference(), *49*
si_pcr_cpp, 46
soft_impute_cpp, 46
soft_impute_fe_cpp, 47
soft_impute_fe_cpp(), *47*
synthetic_outcomes *(*treated_outcomes*)*, 49

tensor_unfold_cpp, 48
tidy(), *5*, *32*
tidy.coresynth_inference, 49
tidy.coresynth_inference(), *34*
treated_outcomes, 49