Package: cfcompare (via r-universe)

November 7, 2026

Title Compare DID, SDID, Matrix Completion and the Triply Robust Panel Estimator

Version 0.2.1

Description A comparison toolkit for binary-treatment panel causal inference. Runs difference-in-differences (DID; two-way fixed effects), synthetic difference-in-differences (SDID), synthetic control (SC), matrix completion (MC), and the Triply RObust Panel (TROP) estimator of Athey, Imbens, Qu and Viviano (2026) <doi:10.1002/jae.70061> on the same data, and returns their average treatment effects on the treated (ATT) on a single tidy schema with shared plots. TROP, DID and matrix completion are implemented natively; SDID and SC are obtained through the 'synthdid' package, and an alternative matrix-completion / interactive fixed-effects estimator through 'gsynth'. The suggested packages 'synthdid' and 'augsynth' are not on CRAN; they can be installed from GitHub at <https://github.com/synth-inference/synthdid> and <https://github.com/ebenmichael/augsynth>, and every feature that needs them is skipped with a message when they are absent. This is an unofficial, independent implementation and is not affiliated with or endorsed by the authors of the TROP estimator.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1)

Imports ggplot2, stats, utils

Suggests RSpectra, future, future.apply, patchwork, synthdid, gsynth, augsynth, did, fixest, testthat (>= 3.0.0), knitr, rmarkdown

URL https://github.com/takuma1102/cfcompare, https://takuma1102.github.io/cfcompare/

BugReports https://github.com/takuma1102/cfcompare/issues

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Takuma Iwasaki [aut, cre, cph]

Maintainer Takuma Iwasaki <iwasakit@stanford.edu>

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

Date/Publication 2026-11-07 20:59:47 UTC

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

RemoteRef HEAD

RemoteSha fcec6959c7df3c72954fcbbea62877bcd1b1fb7d

Contents

as_att . . . 3
autoplot.cf_comparison . . . 3
autoplot.cf_rmse_curve . . . 4
autoplot.cf_rmse_curves . . . 5
autoplot.cf_rmse_tbl . . . 5
autoplot.cf_se_comparison . . . 6
autoplot.cf_trop_grid . . . 6
autoplot.trop . . . 7
autoplot.trop_event_study . . . 8
autoplot.trop_placebo_test . . . 9
bind_att . . . 9
compare_se_modes . . . 10
counterfactual_matrix . . . 12
format.trop_ablation . . . 13
panel_compare . . . 14
panel_matrices . . . 15
panel_rmse . . . 16
plot.cf_rmse_curve . . . 18
plot_counterfactual . . . 19
plot_trop_surfaces . . . 19
rmse_curve . . . 20
rmse_curves . . . 23
selected_lambda . . . 24
sim_panel . . . 24
sim_semisynthetic . . . 26
surface_matrix . . . 27
trop . . . 28
trop_ablation . . . 30
trop_control . . . 33
trop_event_study . . . 35

trop_matrix 37
trop_placebo_test 39
trop_sensitivity 40
true_att 42

Index **43**

`as_att` *Coerce estimator output to the cfcompare ATT schema*

Description

Brings the output of `trop()`, `panel_compare()`, a **synthdid** estimate, or any object with an `estimate` into the common `cf_att_tbl` schema, so results computed elsewhere can be slotted into the same comparison and plots.

Usage

```
as_att(x, ...)
```

Arguments

`x`	An object to tidy.
`...`	Passed to methods (e.g. `method`, `outcome`, `conf_level`).

Value

A `cf_att_tbl` (a `data.frame`).

`autoplot.cf_comparison`
Forest plot of ATT estimates across methods

Description

Point estimate and (where available) confidence interval for each method, on a single axis – the quick visual comparison this package is built for.

Usage

```
## S3 method for class 'cf_comparison'
autoplot(object, ...)

## S3 method for class 'cf_att_tbl'
autoplot(object, ...)
```

Arguments

object	A cf_comparison (from panel_compare()) or a cf_att_tbl.
...	Unused.

Value

A **ggplot2** object.

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, seed = 1)
cmp <- panel_compare(df, "y", "w", "id", "t",
                     methods = c("DID", "MC", "TROP"), se = "none",
                     control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                            tol = 1e-3, seed = 1L))
autoplot(cmp)
```

---

autoplot.cf_rmse_curve

Line plot of estimation RMSE versus a design dimension

Description

One line per estimator; the y axis is on a log scale (as in the paper). Lower is better.

Usage

```
## S3 method for class 'cf_rmse_curve'
autoplot(object, log_y = TRUE, ...)
```

Arguments

object	A cf_rmse_curve from rmse_curve().
log_y	Logical; log10 y axis (default TRUE).
...	Unused.

Value

A ggplot object.

autoplot.cf_rmse_curves *Autoplot for paired RMSE curves*

Description

Autoplot for paired RMSE curves

Usage

```
## S3 method for class 'cf_rmse_curves'
autoplot(object, combined = FALSE, log_y = TRUE, ...)
```

Arguments

object	A cf_rmse_curves.
combined	FALSE (default) returns a named list of two ggplots; TRUE returns a single side-by-side ggplot (needs the patchwork package).
log_y	Logical; log10 y axis.
...	Unused.

Value

A list of two ggplots, or one combined ggplot.

autoplot.cf_rmse_tbl *Bar chart of out-of-sample RMSE across methods*

Description

Visualises a panel_rmse() result as a ranked bar chart (lowest RMSE first), with +/- 1 standard-error whiskers across placebo runs. This is the cross-model RMSE comparison from the paper.

Usage

```
## S3 method for class 'cf_rmse_tbl'
autoplot(object, ...)
```

Arguments

object	A cf_rmse_tbl from panel_rmse().
...	Unused.

Value

A ggplot object.

autoplot.cf_se_comparison *Plot a TROP standard-error comparison*

Description

Forest plot of a `compare_se_modes()` result: every row carries the same TROP point estimate and only the confidence interval differs between resampling schemes. Zero is always kept in view and marked with a dashed reference line, since whether the interval covers zero is the quantity of interest.

Usage

```
## S3 method for class 'cf_se_comparison'
autoplot(object, ...)
```

Arguments

`object`	A `cf_se_comparison` from `compare_se_modes()`.
`...`	Unused.

Value

A **ggplot2** object.

autoplot.cf_trop_grid *Heatmap of the TROP penalty-sensitivity grid*

Description

Cells are coloured by cross-validation loss (darker = better out-of-sample fit) and annotated with the ATT estimate at that penalty pair; the CV-selected cell is outlined. The two swept penalties (see the `axes` argument of `trop_sensitivity()`) form the x and y axes, and the fixed penalty is reported in the subtitle.

Usage

```
## S3 method for class 'cf_trop_grid'
autoplot(object, ...)
```

Arguments

`object`	A `cf_trop_grid` from `trop_sensitivity()`.
`...`	Unused.

Value

A ggplot object.

`autoplot.trop` *Synthetic-control-style trajectory plot for a single TROP fit*

Description

Draws the treated-unit average observed path against the estimated untreated (counterfactual) path, in the style of the **synthdid** plot: a dashed line marks the last pre-treatment period, the post-treatment gap between the two lines is the estimated effect, and – since TROP carries explicit time weights – the time weights $\theta_s = \exp(-\lambda_{time}|t-s|)$ are drawn as a filled band along the bottom (as in **synthdid**'s time-weight strip) to show which periods the counterfactual leans on. The band shows weights, not observation counts; heights are scaled to the largest weight, and with $\lambda_{time} = 0$ the weights are uniform and the band is flat.

Usage

```
## S3 method for class 'trop'
autoplot(object, show_weights = TRUE, show_se = FALSE, ...)
```

Arguments

`object`	A `trop` fit from `trop()`.
`show_weights`	Logical; draw the time-weight band along the bottom.
`show_se`	Logical; append the fit's standard-error method to the subtitle. Defaults to `FALSE` because this plot does not display standard errors – see `trop_event_study()` for the uncertainty visualisation.
`...`	Unused.

Details

This is a *point-estimate* trajectory: it shows no standard errors or confidence intervals (the post-treatment gap is left unshaded so it is not mistaken for a confidence band). To visualise the fit's uncertainty, use `trop_event_study()` and call `autoplot()` on the result, which draws per-period estimates with pointwise confidence bars. Accordingly the subtitle reports only the selected penalties by default; set `show_se = TRUE` to also note which standard-error method the fit used.

Value

A ggplot object.

See Also

`trop_event_study()` and its `autoplot()` method for per-period effects with confidence bars.

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 3, seed = 1)
autoplot(trop(df, "y", "w", "id", "t", se = "none",
              control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                     tol = 1e-3, seed = 1L)))
```

`autoplot.trop_event_study` *Event-study plot for a TROP fit*

Description

Plots the per-period treatment effects from `trop_event_study()` against event time, with pointwise confidence-interval bars, a solid line at zero, and a dotted line marking treatment onset. When pre-treatment periods are present they are drawn as distinct placebo / pre-trend points (open markers) so the pre-period profile can be read at a glance. The SE method is named at the end of the subtitle.

Usage

```
## S3 method for class 'trop_event_study'
autoplot(object, ...)
```

Arguments

`object`	A `trop_event_study` object from `trop_event_study()`.
`...`	Unused.

Value

A `ggplot` object.

Examples

```
df  <- sim_panel(N = 16, T = 9, n_treated = 4, t0 = 6, att = 3, seed = 1)
# fixed penalties keep the example fast; omit `lambda` to cross-validate
fit <- trop(df, "y", "w", "id", "t", se = "none",
            lambda = list(time = 0.1, unit = 0.5, nn = 2))
autoplot(trop_event_study(fit, se = "bootstrap",
                          control = trop_control(n_boot = 20L, seed = 1)))
```

autoplot.trop_placebo_test *Plot a TROP placebo test*

Description

Histogram of the placebo-null ATT distribution with the observed ATT drawn as a solid reference line and zero as a dashed line.

Usage

```
## S3 method for class 'trop_placebo_test'
autoplot(object, ...)
```

Arguments

object	A trop_placebo_test from trop_placebo_test().
...	Unused.

Value

A **ggplot2** object.

bind_att *Stack ATT results into one comparison table*

Description

Row-binds several ATT results — trop() fits, panel_compare() results, **synthdid** estimates, existing cf_att_tbls, or anything as_att() understands — into a single cf_att_tbl ready for autoplot() or plot_counterfactual(). Each argument is passed through as_att() first, so multi-row inputs (e.g. a panel_compare() result) keep all their rows, and columns are aligned automatically. This replaces the manual rbind(as.data.frame(...), ...) + class(...) <- "cf_att_tbl" dance.

Usage

```
bind_att(...)
```

Arguments

...	Objects coercible by as_att(), optionally named to set the method label.

Details

Names given to the arguments become `method` labels, overriding the label already on a *single-row* result; a name on a multi-row input is ignored with a warning (it cannot sensibly relabel several methods at once). For per-object control of other fields such as `outcome`, coerce that object with `as_att()` first and pass the result in unnamed.

Value

A `cf_att_tbl` (a `data.frame`) with the inputs stacked row-wise.

See Also

`as_att()`, `panel_compare()`, `compare_se_modes()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
lam <- list(time = 0.1, unit = 0.5, nn = 2)   # fixed penalties (fast example)
f_pooled   <- trop(df, "y", "w", "id", "t", anchor = "pooled",
                   lambda = lam, se = "none")
f_per_cell <- trop(df, "y", "w", "id", "t", anchor = "per_cell",
                   lambda = lam, se = "none")
bind_att(pooled = f_pooled, per_cell = f_per_cell)
```

---

compare_se_modes *Compare TROP standard-error methods on one fit*

Description

Refits the TROP estimator under several inference methods — the unit-level block bootstrap and/or the delete-one-unit jackknife (over every unit) — and returns the estimate, standard error and confidence interval from each on the common `cf_att_tbl` schema. The penalties are chosen once (a single `se = "none"` fit — by cross-validation, or fixed up front with `lambda`) and reused for every inference method, exactly as `trop_ablation()` reuses one fit's penalties, so the ATT is identical across rows and only the uncertainty changes. This is the table to reach for when checking how sensitive the confidence interval is to the choice of resampling scheme — it is an *inference-mode* comparison, not an estimator comparison, so all rows share one point estimate by construction.

Usage

```
compare_se_modes(
  data,
  outcome,
  treatment,
  unit,
  time,
  covariates = NULL,
  lambda = NULL,
```

```
  se = c("bootstrap", "jackknife"),
  anchor = "pooled",
  control = trop_control(),
  labels = NULL
)
```

Arguments

`data`, `outcome`, `treatment`, `unit`, `time`, `covariates`	Passed to `trop()`.
`lambda`	Optional named list `list(time=, unit=, nn=)` to fix the penalties and skip cross-validation, exactly as in `trop()`. When `NULL` (the default) the penalties are chosen once by cross-validation; either way the resolved penalties are shared by every inference method.
`se`	Character vector of inference methods to compare; any of `"bootstrap"`, `"jackknife"`. Defaults to both. (To compare anchors or penalty constraints instead, see `trop()`'s `anchor` argument and `trop_ablation()`.)
`anchor`	Weight anchoring, as in `trop()` (default `"pooled"`).
`control`	A `trop_control()` list, shared by every fit. Set `n_boot` here to control the bootstrap replications.
`labels`	Optional `method` labels, one per `se`; defaults to `"Bootstrap SE"` / `"Jackknife SE"`.

Value

An object of class `cf_se_comparison` (extending `cf_att_tbl`) with one row per inference method, sharing a common point estimate. If a method does not apply to the design (for example the bootstrap with a single control unit) it still contributes a row, carrying the shared estimate with `NA` uncertainty and an explanatory `note`.

See Also

`trop()`, `trop_ablation()`, `bind_att()`

Examples

```
df <- sim_panel(N = 16, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)

compare_se_modes(df, "y", "w", "id", "t",
                 control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                        tol = 1e-3, n_boot = 20L,
                                        seed = 1L))

# Fix the penalties (skip cross-validation), as with trop()'s `lambda`:
compare_se_modes(df, "y", "w", "id", "t",
                 lambda = list(time = 0, unit = 0, nn = Inf),
                 control = trop_control(n_boot = 20L))
```

`counterfactual_matrix` *Estimated counterfactual (untreated potential outcome) matrix*

Description

Returns a method's estimate of the untreated potential outcome $Y(0)$ as an `N x T` matrix on the panel's unit/time grid. This gives every estimator a common interface, so counterfactuals – and the unit-period prediction RMSE built on them (`panel_rmse()` with `metric = "prediction"`) – can be compared on the same footing.

Usage

```
counterfactual_matrix(object, ...)

predict_counterfactual(object, ...)
```

Arguments

`object`	A fitted object: a `trop` fit, a `cf_comparison` from `panel_compare()`, a **synthdid** estimate, or any object carrying a `counterfactual` element.
`...`	Unused; for method extensibility.

Details

Coverage of the returned matrix differs by method. The native engines (`trop()`, and the DID/MC/TROP fits inside `panel_compare()`) return a fitted value for *every* cell. Methods whose model only defines a counterfactual for the treated units – DIFP, and (when the optional package is installed) SDID, SC and gsynth – fill the treated rows and leave control cells `NA` (a control's untreated outcome is simply its observed value).

`predict_counterfactual()` is an alias for `counterfactual_matrix()`.

Value

For a single fit, an `N x T` numeric matrix (with `NA` where the method does not define a counterfactual). For a `cf_comparison`, a named list of such matrices, one per method.

See Also

`panel_rmse()`, `trop()`, `panel_compare()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, seed = 1)
fit <- trop(df, "y", "w", "id", "t",
            lambda = list(time = 0.1, unit = 0.5, nn = 2), se = "none")
M <- counterfactual_matrix(fit)
dim(M)
```

format.trop_ablation *Render a TROP ablation table as paste-ready LaTeX or Markdown*

Description

Produces a publication-quality table from a `trop_ablation()` result. The `"latex"` output uses **booktabs** rules and math-mode penalty headers, matching the style of the paper's tables (load the booktabs LaTeX package).

Usage

```
## S3 method for class 'trop_ablation'
format(
  x,
  output = c("latex", "markdown"),
  digits = 3,
  caption = "TROP penalty ablation",
  label = "tab:trop-ablation",
  ...
)
```

Arguments

`x`	A `trop_ablation` object from `trop_ablation()`.
`output`	`"latex"` (booktabs) or `"markdown"` (GitHub pipe table).
`digits`	Number of decimal places.
`caption, label`	Table caption and cross-reference label (LaTeX only).
`...`	Unused.

Value

A character vector of lines (paste-ready). Combine with `cat(..., sep = "\n")` or `writeLines()`.

See Also

`trop_ablation()`

Examples

```
df <- sim_panel(N = 14, T = 8, n_treated = 3, t0 = 5, att = 2, seed = 1)
ab <- trop_ablation(df, "y", "w", "id", "t",
                    lambda_full = list(time = 0.1, unit = 0.5, nn = 2))
writeLines(format(ab, "markdown"))
```

`panel_compare` *Compare DID, SDID, MC and TROP on the same panel*

Description

Runs a set of panel estimators on a single long panel and returns their average-treatment-effect-on-the-treated (ATT) estimates on a common tidy schema, so applied researchers can compare them at a glance. DID, MC and TROP are computed natively (no external dependencies); SDID and SC are routed through the **synthdid** package when available, and an alternative MC/IFE can be routed through **gsynth**. Methods whose optional package is missing, or that do not apply to the design, are skipped with a message.

Usage

```
panel_compare(
  data,
  outcome,
  treatment,
  unit,
  time,
  covariates = NULL,
  methods = c("TROP", "DID", "SC", "MC", "SDID", "DIFP"),
  exclude = NULL,
  anchor = "auto",
  se = c("auto", "jackknife", "bootstrap", "none"),
  control = trop_control(),
  verbose = FALSE
)
```

Arguments

`data`	A long `data.frame` with one row per unit-time.
`outcome`, `treatment`, `unit`, `time`	Column names (strings). `treatment` must be a 0/1 indicator of *active* treatment in that cell (works for block, staggered, and non-absorbing designs).
`covariates`	Optional character vector of time-varying covariate column names. Passed to the native engines (`"DID"`, `"MC"`, `"TROP"`), which share the additive covariate model $L_{js} = X'_{js}\phi + R_{js}$; the synthdid-based methods (`"SDID"`, `"SC"`), `"DIFP"` and the optional engines (`"gsynth"`, `"augsynth"`, `"CS"`) do not receive covariates here and are run without them. Covariates must be fully observed across all unit-time cells.
`methods`	Character vector of methods to run. Any of `"DID"`, `"SDID"`, `"SC"`, `"MC"`, `"DIFP"`, `"TROP"`, `"gsynth"`, `"augsynth"`, `"CS"`. Defaults to the native engines plus SDID and SC: `c("TROP", "DID", "SC", "MC", "SDID", "DIFP")`.
`exclude`	Optional character vector of methods to drop from `methods` (after defaults are applied). Convenient for running "everything except one", e.g. `exclude = "DIFP"`. Unknown names are ignored with a warning.

anchor	Weight anchoring for TROP; see trop().
se	Standard-error method for the native engines; see trop().
control	A list of solver/CV settings from trop_control().
verbose	Logical; print CV progress.

Value

An object of class cf_comparison: a list with the tidy table att (class cf_att_tbl), per-method counterfactual matrices counterfactual, the native fit objects fits, and the reshaped panel.

See Also

trop(), autoplot.cf_comparison(), plot_counterfactual()

Examples

```
# panel_compare() fits TROP with cross-validated penalties, so it is wrapped
# in \donttest; few CV cells and a loose solver tolerance keep it quick
# (use the trop_control() defaults for real analyses)
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, seed = 3)
cmp <- panel_compare(df, "y", "w", "id", "t",
                     methods = c("DID", "MC", "TROP"), se = "none",
                     control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                            tol = 1e-3, seed = 1L))
cmp$att
```

---

panel_matrices *Reshape a long panel into TROP matrices*

Description

Converts a long panel data.frame into the N x T outcome (Y) and treatment (W) matrices used by trop_matrix(), and reports the treated rows and the width of the treated block. It is the data-frame-to-matrix companion to trop_matrix(): rather than building Y, W, treated_units and treated_periods by hand, call panel_matrices() and pass the pieces straight through.

Usage

```
panel_matrices(data, outcome, treatment, unit, time)
```

Arguments

data	A long data.frame/tibble with one row per unit-time cell.
outcome, treatment, unit, time	Column names (strings). treatment must be a 0/1 indicator.

Value

A list with components:

`Y` `N x T` outcome matrix (rows = units, columns = times).

`W` `N x T` 0/1 treatment matrix on the same grid.

`units`,`times` the sorted unit and time labels.

`treated_units` integer row indices of units with any treated cell.

`treated_periods` number of periods in which any treated unit is active, i.e. the width of the post block.

See Also

`trop_matrix()`, `trop()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
pm <- panel_matrices(df, "y", "w", "id", "t")
str(pm)
# Feed straight into the matrix-in entry point (treated rows/periods inferred).
trop_matrix(pm$Y, pm$W, lambda_unit = 0.1, lambda_time = 0.1, lambda_nn = Inf)
```

---

panel_rmse *Out-of-sample RMSE across panel estimators*

Description

Compares estimators by how well each does on held-out control data, following the "random blocks" placebo idea of the doubly/triply robust panel estimator paper. Two scoring rules are available:

Usage

```
panel_rmse(
  data,
  outcome,
  treatment,
  unit,
  time,
  methods = c("DID", "SC", "SDID", "MC", "TROP", "DIFP"),
  exclude = NULL,
  metric = c("placebo", "prediction"),
  horizon = 10L,
  n_pseudo = 10L,
  n_runs = 10L,
  control = trop_control(),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

data	A long data.frame, one row per unit-time.
outcome, treatment, unit, time	Column names (strings).
methods	Methods to compare; subset of "DID", "MC", "TROP", "DIFP", "SDID", "SC", "gsynth", "augsynth", "CS". Defaults to c("DID", "SC", "SDID", "MC", "TROP", "DIFP").
exclude	Optional character vector of methods to drop from methods (e.g. exclude = "DIFP"). Unknown names are ignored with a warning.
metric	"placebo" (placebo-ATT RMSE, all methods) or "prediction" (per-cell held-out counterfactual RMSE for "DID"/"MC"/"TROP"/"DIFP", and "SDID"/"SC" when **synthdid** is installed; "gsynth", "augsynth" and "CS" are NA for this metric).
horizon	Number of final periods held out per placebo cohort.
n_pseudo	Number of placebo (pseudo-treated) control units per run.
n_runs	Number of placebo runs to average over.
control	A list of solver/CV controls from trop_control().
seed	Optional integer seed for reproducible placebo draws.
verbose	Logical; print progress.

Details

- metric = "placebo" (default): in each run a random set of control units is given a *placebo* block treatment in the final horizon periods (true effect zero), every method is estimated on that control-only panel, and the score is sqrt(mean(ATT^2)) across runs. This works for **every** method – native (DID, MC, TROP) and wrapped (SDID/SC via **synthdid**, gsynth, augsynth, CS = Callaway & Sant'Anna via **did**).
- metric = "prediction": per-cell one-step-ahead held-out RMSE (the paper's Table-1 unit-period counterfactual prediction error). A block of control cells is held out and each method predicts the untreated outcome there via its counterfactual matrix (counterfactual_matrix()). Supported for the native methods (DID, MC, TROP), for DIFP, and – when **synthdid** is installed – for SDID and SC. The remaining wrapped methods (gsynth, augsynth, CS) are reported as NA for this metric; use metric = "placebo" for them.

Lower is better. Native methods are tuned once on the real data (DID is parameter-free; MC selects lambda_nn; TROP selects the full triplet by cross-validation) and reused across runs. Wrapped methods are skipped with a note when their package is missing or the design does not apply.

Value

A cf_rmse_tbl (a data.frame) with one row per method and columns method, rmse, rmse_se, n_runs, engine, note.

References

Athey, S., Imbens, G. W., Qu, Z., & Viviano, D. (2025/2026). Triply/Doubly Robust Panel Estimators.

See Also

panel_compare(), autoplot.cf_rmse_tbl()

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
r <- panel_rmse(df, "y", "w", "id", "t",
                methods = c("DID", "TROP"),
                horizon = 2, n_pseudo = 3, n_runs = 2,
                control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                       tol = 1e-3, seed = 1L),
                seed = 1)
r
autoplot(r)
```

---

plot.cf_rmse_curve *Plot one or both estimation-RMSE curves*

Description

Plot one or both estimation-RMSE curves

Usage

```
## S3 method for class 'cf_rmse_curve'
plot(x, log_y = TRUE, file = NULL, width = 8, height = 5, ...)

## S3 method for class 'cf_rmse_curves'
plot(
  x,
  combined = FALSE,
  log_y = TRUE,
  file = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

x	A cf_rmse_curve or cf_rmse_curves.
log_y	Logical; log10 y axis (default TRUE).
file	Optional path. If given, the figure is written to a PNG at width x height inches; for separate panels two files are written with -control/-pre suffixes. If NULL, draws to the current device.

width, height	Figure size in inches. Defaults are deliberately generous (single 8x5; combined 12x5).
...	Unused.
combined	For cf_rmse_curves: FALSE (default) draws the two panels separately; TRUE draws them side by side (paper-style).

Value

The input, invisibly.

---

plot_counterfactual *Plot observed vs predicted counterfactual trajectories*

Description

For each native engine in a comparison (DID, MC, TROP), draws the average outcome over the treated units against the predicted control counterfactual, to show how the methods extrapolate through the post-treatment period.

Usage

```
plot_counterfactual(x, methods = NULL)
```

Arguments

x	A cf_comparison from panel_compare().
methods	Optional subset of methods to draw.

Value

A **ggplot2** object.

---

plot_trop_surfaces *Base-R surface views of a TROP penalty-sensitivity grid*

Description

Draws the cross-validation-loss surface and the ATT surface from a trop_sensitivity() grid as two separate, full-width base-graphics plots, each with a colour key (legend) on the right. Unlike autoplot.cf_trop_grid() — a single ggplot2 heatmap that packs both into one panel — this view keeps the surfaces apart so neither is squeezed, and needs no ggplot2. The surface axes follow the swept penalties recorded on the grid (see the axes argument of trop_sensitivity()).

Usage

```
plot_trop_surfaces(
  grid,
  which = c("both", "cv_loss", "att"),
  ask = interactive(),
  ...
)
```

Arguments

grid	A cf_trop_grid returned by trop_sensitivity().
which	Which surface(s) to draw: "both" (default), "cv_loss", or "att". With "both" the two surfaces are drawn as separate figures; on an interactive screen device you are prompted between them (use a multi-page device such as pdf(), or call once per surface, to keep both).
ask	Logical; when drawing both on an interactive device, prompt before the second figure. Defaults to interactive().
...	Further arguments passed to graphics::filled.contour().

Value

Invisibly, a list with the cv_loss and att surface matrices (rows = y-axis penalty, columns = x-axis penalty).

See Also

trop_sensitivity(), autoplot.cf_trop_grid()

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
g <- trop_sensitivity(df, "y", "w", "id", "t",
                      lambda_time = c(0, 0.25, 1),
                      control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                             tol = 1e-3, seed = 1L))
plot_trop_surfaces(g, which = "cv_loss")
```

---

rmse_curve *Estimation-RMSE curve over one design dimension*

Description

Sweeps one design dimension – the number of control units (`"n_control"`), pre-treatment periods (`"n_pre"`), treated units (`"n_treated"`) or post-treatment periods (`"n_post"`) – and, at each value, runs a semi-synthetic Monte Carlo: panels are drawn from a latent factor model in which treatment is selected on the factor loadings (so plain DID/TWFE is biased), a known constant effect `att` is imposed, every estimator is run, and the estimation RMSE against the known truth is recorded. The result is the data behind the paper-style "RMSE vs. N_control / T_pre" line plot, one line per estimator. With a dense `values` grid and enough `n_runs` the curves are smooth, as in the paper.

Usage

```
rmse_curve(
  vary = c("n_control", "n_pre", "n_treated", "n_post"),
  values = NULL,
  n_runs = 500L,
  methods = c("DID", "SDID", "SC", "MC", "DIFP", "TROP"),
  exclude = NULL,
  n_control = 60L,
  n_treated = 8L,
  n_pre = 16L,
  n_post = 6L,
  rank = 4L,
  att = 2,
  noise = 1,
  ar = 0.4,
  trend_sd = 0.05,
  anchor = "pooled",
  control = trop_control(),
  seed = 1L,
  parallel = FALSE,
  progress = interactive(),
  verbose = FALSE
)
```

Arguments

`vary`	Which dimension to sweep: `"n_control"`, `"n_pre"`, `"n_treated"` or `"n_post"`.
`values`	Integer vector of values for the swept dimension. Defaults to `seq(20, 45, by = 5)` for control units, `seq(5, 38, by = 2)` for pre-periods, and `seq(2, 12, by = 2)` for treated units or post-periods. Each value is an actual measured point (its own Monte Carlo); pass a denser/wider grid for smoother or longer curves.
`n_runs`	Monte Carlo replications per value (higher = smoother; the paper uses ~1000). Default 500. **Note:** with the dense default grid and six estimators this is a large simulation (thousands of fits per panel); reduce `n_runs` and/or `values` for a quick look, or parallelise the outer loop.
`methods`	Estimators to include; any subset of the `panel_compare()` methods. Default `c("DID", "SDID", "SC", "MC", "DIFP", "TROP")`, the six estimators compared

in the paper. gsynth, augsynth and CS can be added if those packages are installed.

exclude — Optional character vector of methods to drop from methods (e.g. exclude = "DIFP"). Unknown names are ignored with a warning.

n_control, n_treated, n_pre, n_post — Base design; the dimension named in vary is overridden by values, the rest are held fixed. Defaults give a sizeable panel (60 controls, 8 treated, 16 pre- and 6 post-periods).

rank, att, noise — Number of latent factors, the imposed (true) ATT, and the idiosyncratic-noise scale.

ar — AR(1) coefficient of the idiosyncratic errors (serial correlation).

trend_sd — SD of the heterogeneous unit-specific linear trend slopes (non-parallel trends); treatment is also selected on this slope.

anchor — Estimation anchor for the native ATT ("pooled" by default for speed across the many fits).

control — Solver/CV controls from trop_control().

seed — Base integer seed (each replication uses a distinct offset).

parallel — Logical; if TRUE and the future.apply package is installed, the Monte Carlo tasks are run in parallel honouring the active future plan (e.g. future::plan(future::multisession, workers = 6)). Results are identical to the sequential default because each task is self-seeded. Default FALSE.

progress — Logical; print a rough progress indicator (completed fits, a percentage, elapsed time and an ETA) periodically while the curve runs. Helpful for the heavy large-n_runs jobs. Defaults to interactive(), so it shows in an interactive session and stays silent in scripts and tests. Messages go to stderr(); wrap in suppressMessages() to mute.

verbose — Logical; additionally print a line as each swept value finishes. Independent of progress.

Details

This is a simulation diagnostic on synthetic data; it does not take a user panel. To run a curve on your own data, draw panels with sim_semisynthetic() and call panel_compare() in a loop.

Value

A cf_rmse_curve (a data.frame) with columns method, x (the swept value), rmse, bias, n_runs. The swept-dimension label is in attr(., "vary").

See Also

rmse_curves(), panel_rmse(), autoplot.cf_rmse_curve()

Examples

```
# quick look (small grid + few reps); raise n_runs/values for paper quality
cc <- rmse_curve("n_control", values = c(12, 16), n_runs = 1, n_pre = 6, n_post = 3,
                 n_treated = 2, methods = c("DID", "TROP"),
                 control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                        tol = 1e-3, seed = 1L))
autoplot(cc)
```

`rmse_curves` *Estimation-RMSE curves over both design dimensions*

Description

Convenience wrapper that runs `rmse_curve()` for both the number of control units and the number of pre-treatment periods and bundles the two curves, so they can be plotted individually (default) or side by side (`combined = TRUE`), as in the paper's two-panel figure.

Usage

```
rmse_curves(values_control = NULL, values_pre = NULL, ...)
```

Arguments

`values_control, values_pre`	Optional grids for each sweep (see `rmse_curve()` defaults).
`...`	Passed to `rmse_curve()` (`n_runs`, `methods`, base design, `att`, `noise`, `control`, `seed`, `parallel`, `verbose`, ...). Set `parallel = TRUE` (plus a `future::plan`) to run both sweeps in parallel.

Value

A `cf_rmse_curves` object: a list with `$n_control` and `$n_pre`, each a `cf_rmse_curve`.

See Also

`rmse_curve()`, `autoplot.cf_rmse_curves()`

Examples

```
# defaults are a large simulation; use a small grid + few reps for a quick look
g <- rmse_curves(values_control = c(12, 16), values_pre = c(5, 8), n_post = 3,
                 n_treated = 2, n_runs = 1, methods = c("DID", "TROP"),
                 control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                        tol = 1e-3, seed = 1L))
plot(g)                  # two separate figures (default)
# combined = TRUE needs the optional 'patchwork' package:
if (requireNamespace("patchwork", quietly = TRUE))
  plot(g, combined = TRUE) # side-by-side, paper-style
```

selected_lambda *CV-selected penalties from a TROP penalty-sensitivity grid*

Description

Returns the cross-validation-selected penalties of a `trop_sensitivity()` grid — the grid cell minimising the CV loss — as a `list(time=, unit=, nn=)`, the same shape `trop()` accepts for its `lambda` argument. So the data-driven choice on a grid can be fed straight back into a fit, e.g. `trop(..., lambda = selected_lambda(g))`, without digging the values out of `attr(g, "selected")` by hand.

Usage

```
selected_lambda(grid)
```

Arguments

`grid`	A `cf_trop_grid` from `trop_sensitivity()`.

Value

A named list with the selected `time`, `unit` and `nn` penalties. The full selected row (including the ATT and CV loss there) remains available in `attr(grid, "selected")`.

See Also

`trop_sensitivity()`, `surface_matrix()`, `trop()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
g <- trop_sensitivity(df, "y", "w", "id", "t",
                      lambda_time = c(0, 0.25, 1),
                      control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                             tol = 1e-3, seed = 1L))
selected_lambda(g)
```

sim_panel *Simulate a panel from a low-rank factor model*

Description

Generates a long panel whose untreated potential outcomes follow an interactive-fixed-effects (factor) model on top of two-way fixed effects, in the spirit of the data-generating processes in Athey, Imbens, Qu & Viviano (2026). A block treatment is applied to `n_treated` units from period `t0` onward, with a constant additive effect `att`.

Usage

```
sim_panel(
  N = 30,
  T = 20,
  n_treated = 5,
  t0 = NULL,
  rank = 3L,
  att = 1,
  noise = 0.5,
  seed = NULL,
  n_cov = 0L,
  phi = NULL,
  confounding = 0
)
```

Arguments

N	Number of units.
T	Number of periods.
n_treated	Number of treated units.
t0	First treated period (block design).
rank	Number of latent factors.
att	True treatment effect added to treated cells.
noise	Standard deviation of idiosyncratic noise.
seed	Optional RNG seed.
n_cov	Number of time-varying covariates to generate (default 0, none). Generated covariates are cell-level standard-normal draws whose linear index is part of the untreated potential outcome Y(0), matching the additive covariate model of trop() (L = X.phi + R); pass their names to covariates in trop() / panel_compare() to recover the coefficients. Defaults to length(phi) when phi is supplied.
phi	Optional numeric coefficient vector for the covariates (length n_cov). When NULL and n_cov > 0, coefficients are drawn from a standard normal. The coefficients actually used are returned on the "phi" attribute of the result.
confounding	Non-negative strength of selection-on-confounders in the treatment assignment. With confounding = 0 (default) the first n_treated units are treated independently of the latent factors and covariates, so treatment is ignorable and DID/TWFE is unbiased. With confounding > 0 treated units are selected on their latent factor loadings and, when covariates are present (n_cov > 0), on their post-period covariate index $X'\phi$ as well. The factor part biases estimators without a low-rank term (DID, SDID, SC); the covariate part biases estimators that ignore covariates (SDID, SC, uncontrolled DID). A fully robust estimator (e.g. TROP with covariates=) removes both. Larger values give stronger selection.

Value

A long `data.frame` with columns `id`, `t`, `y`, `w`, and the noiseless counterfactual `y0` (useful for evaluating estimators). When `n_cov > 0` it additionally carries covariate columns `x1`, ..., `x{n_cov}`, and the true coefficients are stored on `attr(., "phi")`.

Examples

```
df <- sim_panel(N = 16, T = 9, n_treated = 3, t0 = 6, seed = 42)
head(df)

# Two covariates with known coefficients, recoverable by trop(covariates=):
dc <- sim_panel(N = 16, T = 9, n_treated = 3, t0 = 6,
                n_cov = 2, phi = c(1.2, -0.7), seed = 1)
attr(dc, "phi")
```

---

sim_semisynthetic *Build a semi-synthetic panel from real data*

Description

Takes a real long panel, uses its outcomes (optionally smoothed through a low-rank-plus-two-way-fixed-effects fit) as the untreated potential outcomes `Y(0)`, and imposes a *known* treatment effect on a chosen block of units and periods. Because the baseline is real data but the effect is known, the result is a ground-truth benchmark that "closely matches" the real setting – the style of semi-synthetic experiment used to evaluate panel estimators in Athey, Imbens, Qu & Viviano (2026). Pair it with `panel_compare()` or `panel_rmse()` to score estimators against the truth.

Usage

```
sim_semisynthetic(
  data,
  outcome,
  unit,
  time,
  n_treated,
  t0 = NULL,
  att = 1,
  effect = NULL,
  baseline = c("observed", "lowrank"),
  lambda_nn = NULL,
  noise = 0,
  seed = NULL
)
```

Arguments

`data`	A real long `data.frame`, one row per unit-time.
`outcome`, `unit`, `time`	Column names (strings).
`n_treated`	Number of units to assign to the placebo treated group (sampled at random from all units).
`t0`	First treated period (block design). Defaults to about three quarters of the way through the panel.
`att`	Constant additive treatment effect imposed on treated cells. Ignored if `effect` is supplied.
`effect`	Optional per-treated-period effect: a single number, or a numeric vector of length `T - t0 + 1` giving a dynamic effect path.
`baseline`	`"observed"` uses the real outcomes directly as `Y(0)`; `"lowrank"` replaces them with a low-rank + two-way-FE fit (optionally plus resampled residuals scaled by `noise`) for a smoother synthetic baseline.
`lambda_nn`	Nuclear-norm penalty for the `"lowrank"` baseline fit.
`noise`	For `"lowrank"`, standard-deviation multiplier on resampled residuals added back to the fit (0 = noiseless baseline).
`seed`	Optional RNG seed.

Value

A long `data.frame` with columns `id`, `t`, `y`, `w`, `y0` (the imposed untreated potential outcome) and `tau` (the true effect, 0 off treatment).

See Also

`sim_panel()`, `panel_compare()`, `panel_rmse()`

Examples

```
real <- sim_panel(N = 16, T = 10, n_treated = 0L, att = 0, seed = 1)
ss <- sim_semisynthetic(real, "y", "id", "t",
                        n_treated = 6, t0 = 8, att = 3, seed = 2)
mean(ss$tau[ss$w == 1])   # true ATT = 3
```

---

surface_matrix *Surface matrix from a TROP penalty-sensitivity grid*

Description

Reshapes a long `trop_sensitivity()` grid into a wide matrix of one quantity — the cross-validation loss or the ATT — laid out over the two swept penalties (see the `axes` argument of `trop_sensitivity()`). Rows are the y-axis penalty, columns the x-axis penalty; the matrix `dimnames` are named after the two penalty columns so the layout is self-describing. This is the same reshape `plot_trop_surfaces()` performs internally, exposed so the surfaces can be inspected or post-processed without re-deriving them with `xtabs()`.

Usage

```
surface_matrix(grid, value = c("cv_loss", "att"))
```

Arguments

grid	A cf_trop_grid from trop_sensitivity().
value	Which quantity to lay out: "cv_loss" (default) or "att".

Value

A numeric matrix (y-axis penalty in rows, x-axis penalty in columns).

See Also

trop_sensitivity(), selected_lambda(), plot_trop_surfaces()

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
g <- trop_sensitivity(df, "y", "w", "id", "t",
                      lambda_time = c(0, 0.25, 1),
                      control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                             tol = 1e-3, seed = 1L))
surface_matrix(g, "cv_loss")
surface_matrix(g, "att")
```

---

trop *Triply RObust Panel (TROP) estimator*

Description

Fits the TROP estimator of Athey, Imbens, Qu & Viviano (2026) on a long panel. TROP combines a low-rank-plus-two-way-fixed-effects outcome model with exponential-decay unit weights (upweighting controls similar to the treated) and time weights (upweighting periods near the treated periods). Penalties are chosen by leave-one-out cross-validation on the control cells. The estimator nests DID/TWFE, matrix completion and synthetic-control-type weighting as special cases.

Usage

```
trop(
  data,
  outcome,
  treatment,
  unit,
  time,
  covariates = NULL,
  lambda = NULL,
```

```
  standardize = FALSE,
  anchor = c("auto", "per_cell", "pooled"),
  se = c("bootstrap", "auto", "jackknife", "none"),
  grids = NULL,
  control = trop_control(),
  verbose = FALSE
)
```

Arguments

`data`	A long `data.frame` with one row per unit-time.
`outcome`, `treatment`, `unit`, `time`	Column names (strings). `treatment` must be a 0/1 indicator of *active* treatment in that cell (works for block, staggered, and non-absorbing designs).
`covariates`	Optional character vector of column names for time-varying covariates. When supplied, the low-rank term is augmented additively as $L_{js} = X'_{js}\phi + R_{js}$ (Athey, Imbens, Qu & Viviano 2025, Section 6.2): the covariate-linear part is unpenalised and the nuclear norm applies to the residual R. Penalty cross-validation is performed on the covariate-residualised model. Covariates must be fully observed (including in treated cells, where they enter the counterfactual). The fitted coefficients are returned in `$phi`.
`lambda`	Optional named list `list(time=, unit=, nn=)` to fix the penalties and skip cross-validation. When `standardize = TRUE`, supplied penalties are interpreted on the standardized-outcome scale. Whether `nn` is supplied here or chosen by cross-validation, the solver adds the small scale-relative stabilising floor `nn_floor` (see `trop_control()`) to every *finite* `nn` (the reported `$lambda$nn` stays nominal and the resolved floor is stored in `$lambda$nn_floor`); set `control = trop_control(nn_floor = 0)` to apply `nn` exactly as given.
`standardize`	Logical; if `TRUE`, the outcome is standardized internally as `(Y - mean(Y)) / sd(Y)` (over all observed cells) before fitting, and the ATT, standard error, confidence interval, per-cell effects and counterfactual are mapped back to the raw outcome scale. This matches the convention of the official Stata command (v0.2.1+); as there, the selected `lambda` values are reported on the *standardized* scale, which makes them comparable across outcomes and across implementations. Default `FALSE` (raw outcome, the previous behaviour). The returned `panel$Y` is on the fitting (standardized) scale, with the centre/scale stored in `$scaling`.
`anchor`	How weights are anchored to treated cells: `"per_cell"` re-solves eq. (2) with cell-specific weights for every treated cell (faithful to the paper); `"pooled"` solves once per *treatment spell group* – treated cells with the same start and end period, pooled across units, the `group(time)` estimands of the official Stata command – with weights anchored to that group (unit distances to the average trajectory of the group's units over the periods in which none of them is treated, time distances to the midpoint of the spell; see `trop_control()`'s `time_center`). Simultaneous permanent adoption is one group, staggered adoption gives one group per cohort, and a unit that switches in and out of treatment contributes one group per spell; the ATT is the treated-cell-weighted average of

	the group effects, which are returned in `$tau_groups`. `"auto"` (default) uses `per_cell` when there are at most `max_cells` treated cells and `pooled` otherwise. `trop_matrix()` has no `anchor` argument; it is always pooled over a single trailing block.
`se`	Standard-error method: `"bootstrap"` (default; unit-level block bootstrap, stratified by treatment status unless `trop_control(boot_resample = "controls")`), `"jackknife"` (delete-one-unit over *every* unit, treated and control, as the Stata command's `vce(jackknife)`; replicates that leave no treated unit are skipped), `"auto"` (jackknife when there are at least 2 treated units, else bootstrap), or `"none"`.
`grids`	Optional list of penalty grids; see `trop_control()`.
`control`	A list of solver/CV settings from `trop_control()`.
`verbose`	Logical; print CV progress.

Value

An object of class `trop`: a list with the ATT `estimate`, `std.error`, `conf.low`/`conf.high`, the selected penalties `lambda`, any covariate coefficients `phi`, per-cell effects (`tau_cells`), the per-spell-group effects `tau_groups` (pooled anchor only: one row per group with its start/end period, unit and cell counts and `estimate`; the ATT is their `n_cells`-weighted average), the estimated counterfactual matrix, the reshaped panel, and `scaling` (the outcome centre/scale used internally; the identity unless `standardize = TRUE`). All effect and outcome-level components are on the raw outcome scale; `panel$Y` and `lambda` are on the fitting scale.

References

Athey, S., Imbens, G. W., Qu, Z., & Viviano, D. (2026). Triply Robust Panel Estimators. *Journal of Applied Econometrics*. doi:10.1002/jae.70061.

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, seed = 1)
# fixed penalties keep this example fast and deterministic; omit `lambda`
# to choose (time, unit, nn) by cross-validation
fit <- trop(df, "y", "w", "id", "t",
            lambda = list(time = 0.1, unit = 0.5, nn = 2), se = "none")
fit
```

`trop_ablation` *Penalty ablation for the TROP estimator*

Description

Refits TROP under a sequence of penalty constraints to show how the ATT moves as the estimator is stripped back towards matrix completion and difference-in-differences. This is the robustness exercise behind Table 5 of Athey, Imbens, Qu & Viviano (2026): constraining `lambda_unit` and/or `lambda_time` to zero (dropping unit/time weights) and/or `lambda_nn` to infinity (dropping the low-rank regression adjustment).

Usage

```
trop_ablation(
  data,
  outcome,
  treatment,
  unit,
  time,
  covariates = NULL,
  control = trop_control(),
  anchor = "pooled",
  se = "none",
  lambda_full = NULL,
  rmse = FALSE,
  horizon = NULL,
  n_pseudo = NULL,
  n_runs = 200L
)

## S3 method for class 'trop_ablation'
print(x, digits = 3, ...)

## S3 method for class 'trop_ablation'
plot(x, file = NULL, width = NULL, height = NULL, res = 200, digits = 3, ...)
```

Arguments

data, outcome, treatment, unit, time, covariates	Passed to trop().
control	A trop_control() list (used for every fit).
anchor	Weight anchoring, as in trop() (default "pooled").
se	Standard-error method for every specification (default "none"; any trop() method is allowed, e.g. "jackknife"). When not "none", the standard error and confidence interval appear in the table.
lambda_full	Optional named list list(time=, unit=, nn=) to fix the full specification's penalties and skip cross-validation.
rmse	Logical; if TRUE, also score each specification by a placebo RMSE. A pseudo block-treatment (true effect zero) is assigned to control units, every specification is refit at its penalties, and the RMSE is sqrt(mean(estimate^2)) over n_runs draws. The same draws are scored under every specification, so the relative RMSE with TROP (full) = 1 is the quantity reported in Table 5 of the paper. Adds rmse and rmse_rel columns, and makes plot() show the (relative) RMSE instead of the ATT.
horizon, n_pseudo	Placebo design used when rmse = TRUE: the number of final periods to pseudo-treat and the number of pseudo-treated control units. Default to the real design's post length and treated-unit count.
n_runs	Number of placebo draws when rmse = TRUE.

x	A trop_ablation object.
digits	Number of decimal places for the penalties and estimates.
...	Unused.
file	Optional output path. When supplied, the table is rendered to that file instead of the active graphics device; the format is taken from the extension (.png, the default, or .pdf for vector output). For LaTeX or Markdown source use format() instead.
width, height	Figure size in inches. Defaults adapt to the number of rows and columns.
res	Resolution in PPI for the .png device (ignored for .pdf).

Details

The full specification's penalties are taken from a single cross-validated trop() fit; every constrained specification then reuses those values where they are not constrained, so the rows are directly comparable. The result is a table object: print it for a clean console table, or format() it to paste-ready LaTeX (booktabs) or Markdown. It is also an ordinary data.frame, so it can be passed to knitr::kable(), gt::gt(), etc.

Value

An object of class trop_ablation (a data.frame) with one row per specification: spec, the three penalties lambda_time/lambda_unit/ lambda_nn, estimate, std.error, conf.low/conf.high, and rank. When rmse = TRUE it also has rmse and rmse_rel (RMSE relative to the full TROP).

References

Athey, S., Imbens, G. W., Qu, Z., & Viviano, D. (2026). Triply Robust Panel Estimators. *Journal of Applied Econometrics*. doi:10.1002/jae.70061.

See Also

trop(); format() for paste-ready LaTeX/Markdown output

Examples

```
df <- sim_panel(N = 14, T = 8, n_treated = 3, t0 = 5, att = 2, seed = 1)
# fixed base penalties keep the example fast; omit `lambda_full` to select
# them by cross-validation
ab <- trop_ablation(df, "y", "w", "id", "t",
                    lambda_full = list(time = 0.1, unit = 0.5, nn = 2))
ab                                  # clean console table

cat(format(ab, "latex"), sep = "\n") # paste-ready LaTeX (booktabs)
```

trop_control *Control settings for* trop()

Description

Control settings for trop()

Usage

```
trop_control(
  max_iter = 2000L,
  tol = 1e-06,
  n_cv_cells = 120L,
  cv_cycles = 2L,
  max_cells = 60L,
  time_center = c("midpoint", "python"),
  pooled_treat_distance = NULL,
  conf_level = 0.95,
  n_boot = 200L,
  boot_resample = c("stratified", "controls"),
  boot_ci = c("percentile", "normal"),
  svd = c("truncated", "full"),
  cv_method = c("loocv", "placebo"),
  n_placebo = 30L,
  nn_floor = "auto",
  nn_floor_scale = 0.001,
  workers = 1L,
  seed = NULL
)
```

Arguments

max_iter	Maximum solver iterations. Higher by default because small lambda_nn converges slowly under the soft-impute iteration.
tol	Reference solver convergence tolerance. The applied tolerance is tightened automatically as lambda_nn weakens (small penalties converge slowly), keeping accuracy roughly constant across the penalty grid; a fixed tolerance otherwise under-converges at small lambda_nn. See trop().
n_cv_cells	Number of control cells sampled for the CV criterion. Set to Inf (or 0) to use *every* control cell, i.e. the paper's full eq. (5) criterion (the Stata command's cells(0)); slower but exact.
cv_cycles	Number of coordinate-descent cycles in penalty selection.
max_cells	Threshold for anchor = "auto" to switch to pooled weights.
time_center	Where the pooled time weights theta_s = exp(-lambda_time |s - c|) are centred within a treated spell (anchor = "pooled", the placebo CV criterion and

the pooled counterfactual behind the plots / event study). "midpoint" (default) uses the true midpoint c = (first + last) / 2 of the spell, the convention of the official Stata command (v0.2.4+); for a single treated period it reduces to the paper's per-cell distance |s - t|, so a single treated cell gets the same estimate under both anchors. "python" places the centre half a period later ((first + last + 1) / 2), the T - tp/2 convention of the official Python package, which trop_matrix() always uses; choose it for like-for-like comparisons with that package.

pooled_treat_distance

Optional overrides for the distances of a pooled group to *itself*, as a list with elements unit and/or time (single non-negative numbers; NULL, the default, keeps the usual distances). unit replaces the unit distance of every member of the group – which is otherwise its distance to the group average, hence generally non-zero with several members – by the given value, so list(unit = 0) gives the treated units the full unit weight; time replaces the time distance of the periods inside the spell by the given value, so list(time = 0) gives the treated periods the full time weight. This is the pooled_treat_distance(unit(#) time(#)) option of the official Stata command (v0.2.6+); it applies to the pooled anchor, the placebo-CV criterion and the pooled counterfactual behind the plots / event study, and is ignored by the per-cell anchor.

conf_level

Confidence level for intervals.

n_boot

Number of replications for the bootstrap standard error (se = "bootstrap").

boot_resample

Which units the bootstrap resamples: "stratified" (default) draws the treated and the control units separately, each with replacement (the stratified block bootstrap of the paper and of the Stata command); "controls" keeps the treated units fixed and resamples only the control units (the default resample = "controls" of the official Python package's trop.bootstrap, v0.1.11+, suited to designs with a single or very few treated units).

boot_ci

Bootstrap confidence-interval type: "percentile" or "normal".

svd

Singular-value decomposition used by the soft-impute solver: "truncated" (default) computes only the leading singular triplets with RSpectra when it is installed and the matrix is large enough to benefit, falling back to the full SVD otherwise; "full" always uses the exact base R svd(). The two agree to numerical tolerance; "truncated" is faster on large panels, "full" is used for exact numerical-agreement checks.

cv_method

Penalty cross-validation criterion. "loocv" (default) scores held-out control-cell prediction error (the paper's eq. (4)-(5)); "placebo" instead stamps the *actual* treated-unit adoption patterns onto randomly resampled control units and minimises the mean squared placebo ATT (the placebo-RMSE / pattern-resampling criterion of the official Python and Stata packages). Under block adoption the two placebo variants coincide; under staggered adoption each placebo draw reproduces the real cohort structure, so "placebo" is the recommended choice there (a warning is emitted if "loocv" is used on a staggered design). Ignored when the penalties are supplied via lambda (CV is then skipped). The placebo criterion always uses the pooled (per-cohort block-centre, cell-weighted) ATT, even when the final fit uses anchor = "per_cell".

`n_placebo` Number of placebo assignments drawn when `cv_method = "placebo"` (larger is more stable but slower).

`nn_floor` Small non-negative stabilising constant *added* to every finite `lambda_nn` inside the solver (`lambda_nn = Inf`, the DID/TWFE special case, is never affected). Rationale: the nuclear-norm proximal step soft-thresholds singular values by `lambda_nn / (2 max w)`, so at `lambda_nn = 0` the step degenerates to the identity, the completion of the treated cells is no longer pinned down by the program, and soft-impute converges slowly; a strictly positive floor keeps the weak-regularisation limit well posed. The default `"auto"` resolves to `nn_floor_scale * s1`, where `s1` is the operator norm (largest singular value) of the two-way-demeaned control outcome – the same data-scale reference the default `lambda_nn` grid is built from. The floor is therefore *relative to the outcome scale*: it is resolved on the fitting scale, so it adapts automatically when `standardize = TRUE` (both modes get the same floor in relative terms). Supply a single non-negative number to use that exact value instead (interpreted on the fitting scale, like `lambda`), or `0` to disable the adjustment entirely, in which case penalties are applied exactly as given – use `nn_floor = 0` for like-for-like numerical comparisons against other implementations at identical penalty values. The resolved value is stored in the fit as `$lambda$nn_floor` and printed when positive.

`nn_floor_scale` Relative constant used when `nn_floor = "auto"`: the resolved floor is `nn_floor_scale * s1` (default `1e-3`, i.e. 0.1\ data's operator norm – an epsilon next to the default `lambda_nn` grid, whose smallest finite point is `0.04 * s1`). Ignored when `nn_floor` is numeric.

`workers` Number of parallel workers for the embarrassingly parallel loops (cross-validation cells, and the bootstrap / jackknife replicates). `1` (default) runs serially. Values `> 1` use `future.apply`/`future` when installed (a transient `multisession` plan is set up and restored automatically); if those packages are missing it falls back to serial with no error. Results are reproducible given `seed`.

`seed` Optional seed for CV cell sampling (reproducibility).

Value

A list of control parameters.

`trop_event_study` *Per-period (event-study) effects from a TROP fit*

Description

Decomposes a fitted `trop()` ATT into per-period treatment effects indexed by event time (periods relative to each unit's first treated period) and attaches pointwise standard errors and confidence intervals. The same resampling used for the overall SE is reused: each resample's per-cell effects are aggregated by event time rather than into a single mean, so no extra modelling assumptions are introduced. With `pre_periods = TRUE` the pre-treatment gaps are returned as placebo / pre-trend points (a flat, near-zero pre-period profile supports the design).

Usage

```
trop_event_study(
  object,
  se = c("bootstrap", "jackknife", "none"),
  pre_periods = TRUE,
  control = NULL,
  ...
)
```

Arguments

object	A trop fit from trop().
se	Standard-error method for the per-period effects: "bootstrap" (default; unit-level block bootstrap, stratified unless trop_control(boot_resample = "controls")), "jackknife" (delete-one-unit over every unit, as for the overall ATT), or "none".
pre_periods	Logical; include pre-treatment event times as placebo / pre-trend points (default TRUE).
control	A list of solver/CV/bootstrap settings from trop_control(); defaults to trop_control(). Use it to set n_boot, boot_ci, seed, and workers for the resampling.
...	Unused.

Details

Because the treated composition varies across bootstrap resamples and each event time draws on relatively few cells, the per-period intervals are wider than the overall ATT interval and cover each point individually (they are not simultaneous bands).

Value

An object of class trop_event_study: a list whose estimates is a data.frame with one row per event time (event_time, estimate, std.error, conf.low, conf.high, n_cells, period), plus the overall att, the se.method, and metadata.

References

Athey, S., Imbens, G. W., Qu, Z., & Viviano, D. (2026). Triply Robust Panel Estimators. *Journal of Applied Econometrics*. doi:10.1002/jae.70061.

See Also

autoplot.trop_event_study() to plot the result.

Examples

```
df  <- sim_panel(N = 16, T = 9, n_treated = 4, t0 = 6, att = 3, seed = 1)
# fixed penalties keep the example fast; omit `lambda` to cross-validate
fit <- trop(df, "y", "w", "id", "t", se = "none",
            lambda = list(time = 0.1, unit = 0.5, nn = 2))
```

```
es  <- trop_event_study(fit, se = "bootstrap", pre_periods = TRUE,
                        control = trop_control(n_boot = 20L, seed = 1))
es
```

`trop_matrix` *TROP estimate on matrix input*

Description

A thin, matrix-in wrapper around the TROP working model, written independently from the paper. The matrix-in form is convenient for numerically comparing `trop()` against other matrix-based implementations on identical inputs. For data-frame input, cross-validated penalty selection, inference and the multi-estimator comparison, use `trop()` and `panel_compare()` instead.

Usage

```
trop_matrix(
  Y,
  W,
  treated_units = NULL,
  lambda_unit,
  lambda_time,
  lambda_nn,
  treated_periods = NULL,
  control = trop_control()
)
```

Arguments

`Y`	N x T outcome matrix.
`W`	N x T 0/1 treatment matrix (1 = actively treated cell).
`treated_units`	Integer row indices of the treated units, used to anchor the unit weights. Defaults to `NULL`, in which case the treated rows are inferred from `W` as the rows with at least one treated cell.
`lambda_unit, lambda_time`	Non-negative decay parameters for the unit and time weights.
`lambda_nn`	Nuclear-norm penalty; use `Inf` to drop the low-rank term. As in `trop()`, the solver adds the small scale-relative stabilising floor `nn_floor` (see `trop_control()`) to a finite `lambda_nn`; `Inf` is never affected, so the weighted-TWFE special case used for the exact numerical-agreement checks is unchanged. For like-for-like comparisons against other implementations at identical finite penalty values, pass `control = trop_control(nn_floor = 0)` so `lambda_nn` is applied exactly as supplied.

`treated_periods`
: Number of final columns treated as the post block, used to build the pre-period mask and the time-distance centre. Defaults to `NULL`, in which case it is inferred from `W` as the number of periods in which any treated unit is active.

`control`
: A list of solver controls from `trop_control()`.

Details

The untreated outcome model is fitted on the control cells with the supplied weights, and the returned effect is the average over treated cells of `Y - alpha - beta - L`, exactly as in the paper's eq. (2). With `lambda_nn = Inf` the low-rank term is dropped and the fit is weighted two-way fixed effects, matching the reference to numerical tolerance. With a finite `lambda_nn` the nuclear-norm term is solved by this package's proximal-gradient / soft-impute routine, which targets the same convex program as eq. (2) (loss `sum_js w (Y - alpha - beta - L)^2`, with no 1/2 factor); `lambda_nn` is therefore on the same scale as the official Python and Stata packages, and finite-penalty results agree with them to solver tolerance.

`trop_matrix()` always uses the pooled weighting scheme anchored to the centre of the treated block: it takes no `anchor` argument and has no per-cell mode. It therefore computes the same pooled, block-centre estimator as the official Python package's `TROP_TWFE_average` (implemented independently), and exists as a like-for-like matrix-in cross-check against that pooled reference. `trop(..., anchor = "pooled")` uses the same convention (unit distances to the *average* treated trajectory, time distances to the centre of the treated block), so on a trailing-block design the two agree to solver tolerance for the same penalties. For the per-cell estimator (the paper's eq. (12) / Algorithm 2), use `trop(..., anchor = "per_cell")`, which shares the data-frame core `.trop_att()`.

Value

A single numeric ATT estimate.

References

Athey, S., Imbens, G. W., Qu, Z., & Viviano, D. (2026). *Triply Robust Panel Estimators.* Journal of Applied Econometrics. doi:10.1002/jae.70061.

See Also

`trop()`, `panel_compare()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
Y <- matrix(0, max(df$id), max(df$t)); W <- Y
for (k in seq_len(nrow(df))) { Y[df$id[k], df$t[k]] <- df$y[k]
  W[df$id[k], df$t[k]] <- df$w[k] }
tu <- which(rowSums(W) > 0)
trop_matrix(Y, W, tu, 0.1, 0.1, Inf, treated_periods = 4)

# treated_units and treated_periods are inferred from W when omitted, so the
# matrices produced by panel_matrices() can be fed in directly.
```

```
pm <- panel_matrices(df, "y", "w", "id", "t")
trop_matrix(pm$Y, pm$W, lambda_unit = 0.1, lambda_time = 0.1, lambda_nn = Inf)
```

`trop_placebo_test` *Placebo (randomisation) test for a TROP ATT*

Description

Assesses whether a fitted `trop()` ATT is distinguishable from zero by comparing it to a placebo null distribution. Control units are repeatedly relabelled as pseudo-treated — using the *same number of treated units and the same treatment timing* as the real design, so block, staggered and non-absorbing patterns are all reproduced faithfully — and the estimator is refit at the fit's cross-validated penalties (the true effect is zero by construction). The observed ATT is then placed against that null.

Usage

```
trop_placebo_test(
  object,
  B = 500L,
  alternative = c("two.sided", "greater", "less"),
  control = NULL
)

## S3 method for class 'trop_placebo_test'
print(x, digits = 4, ...)
```

Arguments

`object`	A `trop` fit from `trop()`.
`B`	Number of placebo assignments to draw.
`alternative`	Direction of the test: `"two.sided"` (default), `"greater"` or `"less"`.
`control`	A `trop_control()` list for the placebo refits (penalties are held fixed at the fit's values; this mainly sets `seed` and `workers`). Defaults to `trop_control()`; set `seed` here for reproducible draws.
`x`	A `trop_placebo_test` object.
`digits`	Number of significant digits to print.
`...`	Unused.

Details

This is a randomisation/permutation test, not a standard error: it answers "could a TROP design like mine produce an effect this large when there is none?" For interval estimates use the bootstrap or jackknife (see `trop()` and `compare_se_modes()`).

Value

An object of class `trop_placebo_test` (a list) with the `observed` ATT, the vector of `placebo` ATTs, the `p.value`, the placebo-null `mean` and `sd`, a central null interval `null.low`/`null.high`, `alternative`, `B`, and `n_treated_units`. It has `print()` and `autoplot()` methods.

See Also

`trop()`, `compare_se_modes()`

Examples

```
df  <- sim_panel(N = 16, T = 9, n_treated = 3, t0 = 6, att = 3, seed = 1)
# fixed penalties keep the example fast; omit `lambda` to cross-validate
fit <- trop(df, "y", "w", "id", "t", se = "none",
            lambda = list(time = 0.1, unit = 0.5, nn = 2))
pt  <- trop_placebo_test(fit, B = 30)
pt
```

---

trop_sensitivity *TROP penalty-sensitivity grid (heatmap data)*

Description

Sweeps any two of the three TROP penalties — λ_{time}, λ_{unit} and λ_{nn} — against each other while holding the third fixed, and records at each grid point both the resulting ATT estimate and the leave-one-out cross-validation loss. This is the data behind the diagnostic heatmap: colour the cells by CV loss to see where the data-driven choice lands, and read the ATT off each cell to judge how sensitive the estimate is to the penalties. Use `axes` to choose which two penalties go on the x and y axes; the remaining (fixed) penalty is reported in the heatmap subtitle.

Usage

```
trop_sensitivity(
  data,
  outcome,
  treatment,
  unit,
  time,
  lambda_time = NULL,
  lambda_nn = NULL,
  lambda_unit = NULL,
  axes = c("nn", "time"),
  anchor = "pooled",
  control = trop_control(),
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A long `data.frame`, one row per unit-time.
`outcome`, `treatment`, `unit`, `time`	Column names (strings).
`lambda_time`, `lambda_nn`, `lambda_unit`	Penalty inputs. For a penalty that is on an axis (see `axes`), supply a numeric grid to sweep (or `NULL` for a data-driven default grid; the `lambda_nn` default keeps only the finite part, as the heatmap axis needs finite values). For the penalty held fixed, supply a single value, or `NULL` to choose it once by cross-validation and hold it fixed across the sweep.
`axes`	Length-2 character vector naming the two penalties to sweep, as `c(x, y)`: the first is the x axis, the second the y axis. Each must be one of `"time"`, `"unit"`, `"nn"`. Defaults to `c("nn", "time")` (the historical layout). The third penalty is held fixed.
`anchor`	Estimation anchor passed to the ATT computation (`"pooled"` by default for speed).
`control`	A list of solver/CV controls from `trop_control()`.
`seed`	Optional integer seed for CV-cell sampling.
`verbose`	Logical; print progress.

Value

A `cf_trop_grid` (a `data.frame`) with columns `lambda_time`, `lambda_nn`, `lambda_unit`, `att`, `cv_loss`. The two swept penalties vary across rows; the fixed penalty is constant. The grid point minimising `cv_loss` (the data-driven choice) is stored in `attr(., "selected")`, and the chosen layout in `attr(., "axes")` (named x/y) and `attr(., "fixed")`.

See Also

`trop()`, `autoplot.cf_trop_grid()`

Examples

```
df <- sim_panel(N = 14, T = 9, n_treated = 3, t0 = 6, att = 2, seed = 1)
g <- trop_sensitivity(df, "y", "w", "id", "t",
                      lambda_time = c(0, 0.1, 0.5), lambda_nn = c(2, 5),
                      control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                             tol = 1e-3, seed = 1L))
autoplot(g)

# Sweep the unit penalty against the nuclear-norm penalty instead, holding
# the time penalty fixed (chosen by CV):
g2 <- trop_sensitivity(df, "y", "w", "id", "t", axes = c("nn", "unit"),
                       control = trop_control(n_cv_cells = 5L, cv_cycles = 1L,
                                              tol = 1e-3, seed = 1L))
autoplot(g2)
```

`true_att` *True ATT of a (semi-)synthetic panel*

Description

Convenience accessor for the ground-truth average treatment effect on the treated of a simulated panel. It reads the per-cell true effect — the `tau` column produced by `sim_semisynthetic()`, or `y - y0` for a `sim_panel()` panel — and averages it over the actively treated cells, giving the number estimators are trying to recover. Pair it with `panel_compare()` or `panel_rmse()` to score estimates against the truth without re-deriving it by hand.

Usage

```
true_att(data, treatment = "w", tau = "tau")
```

Arguments

`data`	A long `data.frame` with a treatment indicator and either a `tau` column (true per-cell effect) or both `y` and `y0` columns.
`treatment`	Name of the 0/1 treatment column (default `"w"`).
`tau`	Name of the true-effect column (default `"tau"`). When that column is absent the per-cell effect is taken as `y - y0`.

Value

A single numeric: the true ATT averaged over treated cells.

See Also

`sim_semisynthetic()`, `sim_panel()`, `panel_compare()`

Examples

```
real <- sim_panel(N = 16, T = 10, n_treated = 0L, att = 0, seed = 1)
ss <- sim_semisynthetic(real, "y", "id", "t",
                        n_treated = 6, t0 = 8, att = 3, seed = 2)
true_att(ss)   # 3
```

Index

as_att, 3
as_att(), *9, 10*
autoplot(), *7*, *9*, *40*
autoplot.cf_att_tbl
(autoplot.cf_comparison), 3
autoplot.cf_comparison, 3
autoplot.cf_comparison(), *15*
autoplot.cf_rmse_curve, 4
autoplot.cf_rmse_curve(), *22*
autoplot.cf_rmse_curves, 5
autoplot.cf_rmse_curves(), *23*
autoplot.cf_rmse_tbl, 5
autoplot.cf_rmse_tbl(), *18*
autoplot.cf_se_comparison, 6
autoplot.cf_trop_grid, 6
autoplot.cf_trop_grid(), *19, 20*, *41*
autoplot.trop, 7
autoplot.trop_event_study, 8
autoplot.trop_event_study(), *36*
autoplot.trop_placebo_test, 9

bind_att, 9
bind_att(), *11*

cf_att_tbl, *10*
compare_se_modes, 10
compare_se_modes(), *6*, *10*, *39, 40*
counterfactual_matrix, 12
counterfactual_matrix(), *17*

format(), *32*
format.trop_ablation, 13

graphics::filled.contour(), *20*

panel_compare, 14
panel_compare(), *3, 4*, *9, 10*, *12*, *18, 19*, *21*,
22, *25–27*, *37, 38*, *42*
panel_matrices, 15
panel_rmse, 16
panel_rmse(), *5*, *12*, *22*, *26, 27*, *42*

plot.cf_rmse_curve, 18
plot.cf_rmse_curves
(plot.cf_rmse_curve), 18
plot.trop_ablation (trop_ablation), 30
plot_counterfactual, 19
plot_counterfactual(), *9*, *15*
plot_trop_surfaces, 19
plot_trop_surfaces(), *27, 28*
predict_counterfactual
(counterfactual_matrix), 12
print.trop_ablation (trop_ablation), 30
print.trop_placebo_test
(trop_placebo_test), 39

rmse_curve, 20
rmse_curve(), *4*, *23*
rmse_curves, 23
rmse_curves(), *22*

selected_lambda, 24
selected_lambda(), *28*
sim_panel, 24
sim_panel(), *27*, *42*
sim_semisynthetic, 26
sim_semisynthetic(), *22*, *42*
surface_matrix, 27
surface_matrix(), *24*

trop, 28
trop(), *3*, *7*, *9*, *11, 12*, *15, 16*, *24, 25*, *31–33*,
35–41
trop_ablation, 30
trop_ablation(), *10, 11*
trop_control, 33
trop_control(), *11*, *15*, *17*, *22*, *29–31*,
36–39, *41*
trop_event_study, 35
trop_event_study(), *7, 8*
trop_matrix, 37
trop_matrix(), *15, 16*, *34*

trop_placebo_test, 39
trop_placebo_test(), *9*
trop_sensitivity, 40
trop_sensitivity(), *6*, *19*, *20*, *24*, *27*, *28*
true_att, 42