Package 'SCMs'

November 4, 2026

Title Specification Curve Analysis for Synthetic Control Methods

Version 0.1.1

Author Zubin Jelveh [aut, cre]

Maintainer Zubin Jelveh <zjelveh@umd.edu>

License GPL-3

URL https://github.com/zjelveh/SCMs

BugReports https://github.com/zjelveh/SCMs/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports CVXR (>= 1.9.0), ECOSolveR, dplyr (>= 1.0.7), foreach (>= 1.5.1), ggplot2 (>= 3.3.3), gtable, kernlab, cowplot, optimx, data.table, methods, parallel, purrr (>= 0.3.4), R6, stats, stringr (>= 1.4.0), tibble (>= 3.1.2), tidyr (>= 1.1.3), utils, doParallel, glmnet, gridtext, gridExtra, generics

Description Estimates synthetic control models while exposing the implementation choices that define the synthetic counterfactual: how matching variables are constructed, how predictor weights are set, which constraints the donor weights satisfy (simplex, lasso, ridge, penalized), whether an intercept is included, and which outcome model corrects for bias (OLS, ridge, lasso, augmented synthetic control). Runs specification-curve analysis over a grid of these choices (Simonsohn, Simmons and Nelson (2020) <doi:10.1038/s41562-020-0912-z>), with placebo-in-space and bootstrap inference for single specifications and placebo inference for the curve as a whole, and uses 'xgboost' models with SHAP values (Lundberg and Lee (2017) <doi:10.48550/arXiv.1705.07874>) to show which choices drive the estimates. Methods follow Abadie, Diamond and Hainmueller (2010) <doi:10.1198/jasa.2009.ap08746>, Abadie (2021) <doi:10.1257/jel.20191450>, Ben-Michael, Feller and Rothstein (2021) <doi:10.1080/01621459.2021.1929245> and Chen and Guestrin (2016) <doi:10.1145/2939672.2939785>.

Suggests testthat (>= 3.0.0), clarabel, broom, xgboost, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2026-11-04 21:40:13 UTC

Contents

augment.scest . . . 3
b.est.clarabel.ols . . . 4
b.est.clarabel.simplex . . . 5
b.est.clarabel.unified . . . 5
b.est.cvxr.fallback . . . 6
bootstrap_null_inference . . . 7
calculate_abadie_significance . . . 8
calculate_bootstrap_pvalues . . . 9
calculate_spec_curve_pvalues_filtered . . . 9
coef.scest . . . 11
create_analysis_config . . . 12
create_bootstrap_dataset . . . 13
create_scm_dataset . . . 13
create_xgboost_config . . . 15
enforce_null_hypothesis . . . 16
estimate_sc . . . 17
formula-interface . . . 18
glance.scest . . . 19
inference_placebo . . . 20
inference_sc . . . 21
perform_bootstrap_inference . . . 22
plot.spec_curve . . . 23
plot_spec_curve . . . 24
print.scdata . . . 29
print.scest . . . 30
print.scest_formula . . . 31
run_spec_curve_analysis . . . 31
run_xgboost_shap_analysis . . . 33
scdata . . . 34
scest . . . 37
scplot . . . 39
solve_qp_clarabel . . . 41
SpecResultsManager . . . 42
spec_curve . . . 45
spec_curve-methods . . . 48
summary.scdata . . . 50
summary.scest . . . 50
summary.scest_formula . . . 51
summary.spec_curve . . . 52
synth . . . 53

tidy-methods 55
tidy.scest 55
tidy.spec_curve 57

Index **59**

`augment.scest` *Augment Synthetic Control Data with Fitted Values and Residuals*

Description

Add fitted values, residuals, and other observation-level statistics to the original data used in synthetic control estimation.

Usage

```
## S3 method for class 'scest'
augment(x, data = NULL, ...)
```

Arguments

`x`	An object of class "scest" from `scest()`
`data`	Optional data.frame/data.table to augment. If NULL, attempts to reconstruct
`...`	Additional arguments (not currently used)

Value

A data.table with original data plus:

- `.fitted` - Numeric. Fitted values from synthetic control
- `.resid` - Numeric. Residuals (actual - fitted)
- `.period_type` - Character. "pre" or "post" treatment
- `.is_treated` - Logical. Whether observation is from treated unit

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
sc_obj <- scdata(
  df = toy,
  id.var = "unit",
  time.var = "year",
  outcome.var = "y",
```

```
  period.pre = 1:4,
  period.post = 5:6,
  unit.tr = "treated",
  unit.co = c("donor1", "donor2"),
  covagg = list(list(var = "outcome_var", partition_periods = list(type = "by_period")))
)
fit <- scest(sc_obj, w.constr = list(name = "simplex"))
generics::augment(fit)
```

`b.est.clarabel.ols` *Clarabel W Optimization - OLS (Unconstrained)*

Description

Solve unconstrained quadratic programming problem using clarabel

Usage

```
b.est.clarabel.ols(A, Z, C = NULL, J, KM, w.constr, V)
```

Arguments

`A`	Treated unit features matrix
`Z`	Donor features matrix
`C`	Optional constant-term matrix.
`J`	Number of donor units
`KM`	Number of additional variables
`w.constr`	Constraint specification
`V`	Feature weighting matrix

Value

Numeric vector of weights

b.est.clarabel.simplex

Clarabel W Optimization - Simplex

Description

Solve simplex-constrained QP using clarabel (sum to 1, non-negative)

Usage

```
b.est.clarabel.simplex(A, Z, C = NULL, J, KM, w.constr, V)
```

Arguments

A	Treated unit features matrix
Z	Donor features matrix
C	Optional constant-term matrix.
J	Number of donor units
KM	Number of additional variables
w.constr	Constraint specification
V	Feature weighting matrix

Value

Numeric vector of weights

b.est.clarabel.unified

Unified Clarabel W Optimization Dispatcher

Description

Dispatch to appropriate clarabel constraint-specific function

Usage

```
b.est.clarabel.unified(A, Z, C = NULL, J, KM, w.constr, V)
```

Arguments

A	Treated unit features matrix
Z	Donor features matrix
C	Optional constant-term matrix.
J	Number of donor units
KM	Number of additional variables
w.constr	Constraint specification
V	Feature weighting matrix

Value

Numeric vector of weights

---

b.est.cvxr.fallback *CVXR fallback function for W optimization*

Description

Solve W optimization using CVXR (original implementation)

Usage

```
b.est.cvxr.fallback(A, Z, C = NULL, J, KM, w.constr, V, CVXR.solver)
```

Arguments

A	Treated unit features matrix
Z	Donor features matrix
C	Optional constant-term matrix.
J	Number of donor units
KM	Number of additional variables
w.constr	Constraint specification
V	Feature weighting matrix
CVXR.solver	CVXR solver to use

Value

Numeric vector of weights

`bootstrap_null_inference` *Bootstrap-based significance testing under null hypothesis of no treatment effect*

Description

Implements bootstrap inference by: 1) Enforcing null hypothesis by subtracting actual treatment effect from post-treatment values, 2) Bootstrap resampling at unit level, 3) Re-estimating SCM on bootstrapped samples, 4) Computing p-values from bootstrap distribution

Usage

```
bootstrap_null_inference(
  sc.pred,
  dataset,
  n_bootstrap = 1000,
  cores = 1,
  verbose = FALSE,
  seed = NULL
)
```

Arguments

`sc.pred`	Result from estimate_sc function
`dataset`	Original dataset used for estimation
`n_bootstrap`	Number of bootstrap replications (default: 1000)
`cores`	Number of cores for parallel processing (default: 1)
`verbose`	Logical for verbose output (default: FALSE)
`seed`	Random seed for reproducibility (default: NULL)

Details

Bootstrap Null Hypothesis Inference for Synthetic Control

Value

List containing bootstrap results and p-values

`calculate_abadie_significance` *Calculate Multiple Test Statistics and Rank-based P-values*

Description

Calculates significance measures following Abadie et al. (2010) German reunification paper. Computes post/pre RMSPE ratios, treatment effects, normalized treatment effects, and their rank-based p-values: one-sided in `expected_direction`, or two-sided (ranked by |statistic|) when `expected_direction = "two_sided"`. The RMSPE ratio is always one-sided.

Usage

```
calculate_abadie_significance(
  taus_result,
  rmse_result,
  rmspe_threshold = 0,
  expected_direction = "two_sided"
)
```

Arguments

`taus_result`	Data.table containing treatment effects for all units
`rmse_result`	Data.table containing RMSE values for all units
`rmspe_threshold`	Numeric. Threshold for filtering poor pre-treatment fits (default: 0)
`expected_direction`	Character. Expected direction of treatment effect: "negative", "positive", or "two_sided" (default: "two_sided")

Details

Calculate Abadie-style Significance Measures

Value

List containing Abadie significance measures for all test statistics

calculate_bootstrap_pvalues

Calculate Bootstrap P-values Using Rank-Based Method

Description

Calculates empirical bootstrap p-values by determining where the actual treatment effect ranks within the bootstrap distribution. This is the standard approach for bootstrap hypothesis testing.

Usage

```
calculate_bootstrap_pvalues(bootstrap_effects, actual_effects, verbose = FALSE)
```

Arguments

`bootstrap_effects`	List of bootstrap effect distributions
`actual_effects`	List of actual treatment effects
`verbose`	Logical for verbose output

Details

The p-values are calculated as follows:

- One-tailed positive: P(bootstrap >= actual) = (n - rank) / n
- One-tailed negative: P(bootstrap <= actual) = (rank + 1) / n
- Two-tailed: 2 * min(positive, negative), capped at 1.0

Value

Data.table with p-values and rank information

calculate_spec_curve_pvalues_filtered

Calculate Placebo-in-Space Curve-Level Inference

Description

Calculates curve-level statistics and p-values on filtered specification-curve results using placebo-in-space comparison.

Usage

```
calculate_spec_curve_pvalues_filtered(
  filtered_results,
  curve_stat = c("abs_median_tau", "median_abs_tau", "consistency_ratio"),
  weighting = c("none", "pre_rmspe_percentile"),
  two_sided = TRUE,
  expected_direction = "two_sided",
  grid_policy = "strict",
  min_placebos = 1L,
  min_specs = 2L
)
```

Arguments

`filtered_results`	Data.table. Filtered results data (post-filtering by outcomes, RMSE, etc.).
`curve_stat`	Character vector. Curve-level statistics to compute: `"median_tau"`, `"abs_median_tau"`, `"median_abs_tau"`, `"consistency_ratio"`.
`weighting`	Character vector. Weighting mode(s) applied to each requested curve-level statistic: `"none"` or `"pre_rmspe_percentile"`.
`two_sided`	Logical. Whether to compute two-sided p-values using absolute-value tails.
`expected_direction`	Character. Expected sign direction for one-sided tests (`"negative"` or `"positive"`). Ignored when `two_sided = TRUE`.
`grid_policy`	Character. Grid mismatch policy: `"strict"`, `"drop_incomplete_units"`, `"intersect_specs"`.
`min_placebos`	Integer. Minimum number of placebo units required after applying `grid_policy`. Must be >= 1.
`min_specs`	Integer. Minimum number of specifications required after applying `grid_policy`. Must be >= 2.

Details

Calculate Specification Curve P-values on Filtered Data

For each unit and specification, this function uses the signed per-spec effect summary `tau_s`. Two-sided p-values always compare absolute tails against placebo-unit curve statistics:

$$p = (1 + \#\{|T_j| \geq |T_{treated}|\})/(J + 1).$$

One-sided p-values (`two_sided = FALSE`) require a signed statistic and count placebos at least as extreme in `expected_direction`, e.g. for `"negative"`

$$p = (1 + \#\{T_j \leq T_{treated}\})/(J + 1).$$

The supported curve statistics are:

- `median_tau`: $\mathrm{median}_s(\tau_s)$ (signed)
- `abs_median_tau`: $|\mathrm{median}_s(\tau_s)|$

- `median_abs_tau`: $\text{median}_s(|\tau_s|)$
- `consistency_ratio`: $|\sum_s \tau_s| / \sum_s |\tau_s|$

For weighted variants, pre-RMSPE-percentile weights are defined within unit as $w_s = 1 - rank(\text{RMSPE}_s^{pre})/(S+1)$ and applied directly to the statistic computation.

Value

List with:

- `treated_summary`: Treated-unit curve statistics and p-values.
- `stats_by_unit`: Curve statistics for treated and placebo units.
- `attrition_report`: Unit/spec retention summary under `grid_policy`.

`coef.scest` *Coef Method for Synthetic Control Methods*

Description

The coef method for synthetic control prediction fitted objects.

Usage

```
## S3 method for class 'scest'
coef(object, ...)
```

Arguments

`object`	Class "scest" object, obtained by calling `scest`.
`...`	Other arguments (eg. `ncols`).

Value

No return value, called to show `scest` constructed weights.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

See Also

`scest` for synthetic control prediction.

Supported methods: `print.scest`, `summary.scest`, `coef.scest`.

create_analysis_config *Create Configuration for Multiple Dataset Analysis*

Description

Creates standardized configuration objects for running specification curve analysis across multiple datasets.

Usage

```
create_analysis_config(
  analysis_type = c("homicide", "ois"),
  custom_params = list()
)
```

Arguments

analysis_type	Character. Type of analysis ("homicide" or "ois").
custom_params	List. Custom parameters to override defaults.

Details

Create Analysis Configuration

Value

List containing configuration parameters for the specified analysis type.

Examples

```
# Create homicide analysis configuration
homicide_config <- create_analysis_config("homicide")

# Create OIS analysis configuration with custom parameters
ois_config <- create_analysis_config("ois",
  custom_params = list(treated_period = 2021))
```

create_bootstrap_dataset *Create Bootstrap Dataset by Unit-level Resampling*

Description

Create Bootstrap Dataset by Unit-level Resampling

Usage

```
create_bootstrap_dataset(
  dataset_null,
  bootstrap_units,
  col_name_unit,
  treated_unit
)
```

Arguments

dataset_null	Original dataset with null enforced
bootstrap_units	Bootstrapped unit names
col_name_unit	Column name for unit identifier
treated_unit	Name of treated unit

Value

Bootstrap dataset

create_scm_dataset *Create Dataset for Synthetic Control Method*

Description

This function prepares a dataset for use with the Synthetic Control Method (SCM) by formatting and standardizing the input data.

Usage

```
create_scm_dataset(
  dataset,
  outcome,
  covagg,
  col_name_unit_name,
  name_treated_unit,
```

```
  col_name_period,
  treated_period,
  min_period,
  end_period,
  constant = FALSE
)
```

Arguments

dataset	Data frame or data.table containing the panel data.
outcome	Character. Name of the outcome variable column.
covagg	List of covariate-aggregation operations passed through to scdata.
col_name_unit_name	Character. Name of the column containing unit names.
name_treated_unit	Character. Name of the treated unit.
col_name_period	Character. Name of the column containing time periods.
treated_period	Numeric. Time period when treatment starts.
min_period	Numeric or NULL. Earliest time period to include. If NULL, uses the minimum in the dataset.
end_period	Numeric or NULL. Latest time period to include. If NULL, uses the maximum in the dataset.
constant	Logical. Whether to include a constant term. Default is FALSE.

Value

A list containing the prepared data for SCM analysis.

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
scm_data <- create_scm_dataset(
  dataset = toy,
  outcome = "y",
  covagg = list(
    list(var = "outcome_var", partition_periods = list(type = "by_period"))
  ),
  col_name_unit_name = "unit",
  name_treated_unit = "treated",
  col_name_period = "year",
  treated_period = 5,
```

```
  min_period = 1,
  end_period = 6
)
inherits(scm_data, "scdata")
```

`create_xgboost_config` *Create Configuration for Direct Long Format Analysis*

Description

Creates configuration for XGBoost analysis that works directly with long format data from spec_curve().

Usage

```
create_xgboost_config(
  dataset_name,
  treated_unit_name,
  outcome_filter = NULL,
  spec_features = c("outcome_model", "const", "fw", "feat", "data_sample"),
  treated_unit_only = TRUE,
  tune_xgboost = NULL,
  xgboost_grid = NULL,
  xgboost_cv_folds = 5,
  xgboost_params = NULL
)
```

Arguments

`dataset_name`	Character. Name of the dataset for identification.
`treated_unit_name`	Character. Name of the treated unit.
`outcome_filter`	Character. Outcome variable to filter on (optional).
`spec_features`	Character vector. Specification features to use as predictors.
`treated_unit_only`	Logical. If TRUE, only analyze the treated unit.
`tune_xgboost`	Logical or NULL. If TRUE, tune XGBoost hyperparameters via k-fold CV on the treated unit before SHAP/LOO. If NULL, defaults to TRUE when xgboost_params is NULL and FALSE otherwise.
`xgboost_grid`	Data.frame, data.table, or list. Grid of hyperparameters for tuning. Must include max_depth, eta, nrounds, subsample, colsample_bytree. If NULL, uses an internal default grid.
`xgboost_cv_folds`	Integer. Number of CV folds for tuning. Default is 5.
`xgboost_params`	List. Fixed parameters for XGBoost training. If NULL, defaults are used. If tune_xgboost is TRUE, xgboost_params must be NULL.

Value

List containing configuration parameters.

Examples

```
xgb_cfg <- create_xgboost_config(
  dataset_name = "toy",
  treated_unit_name = "treated",
  outcome_filter = "outcome",
  spec_features = c("feat", "outcome_model", "const", "fw", "data_sample")
)
```

---

enforce_null_hypothesis

Enforce Null Hypothesis by Subtracting Treatment Effects

Description

Enforce Null Hypothesis by Subtracting Treatment Effects

Usage

```
enforce_null_hypothesis(dataset, sc.pred, actual_effects, verbose = FALSE)
```

Arguments

`dataset`	Original dataset
`sc.pred`	SC estimation results
`actual_effects`	List of actual treatment effects by outcome model
`verbose`	Logical for verbose output

Value

Dataset with null hypothesis enforced

`estimate_sc` *Estimate Synthetic Control Model*

Description

This function processes and estimates a Synthetic Control Model (SCM) based on the provided dataset and parameters.

Usage

```
estimate_sc(
  dataset,
  outcome,
  covagg,
  col_name_unit_name,
  name_treated_unit,
  col_name_period,
  treated_period,
  min_period,
  end_period,
  outcome_models = c("none", "ols", "ridge", "lasso", "augsynth"),
  feature_weights = "uniform",
  w.constr = NULL,
  V = "separate",
  V.mat = NULL,
  solver = "ECOS",
  constant = FALSE
)
```

Arguments

`dataset`	A data frame containing the panel data.
`outcome`	Character. Name of the outcome variable (column in dataset).
`covagg`	List of covariate-aggregation operations passed to `scdata`. Each operation should define at least `var`, and can optionally define `select_periods`, `partition_periods`, and `compute`.
`col_name_unit_name`	Character. Column name in dataset containing unit names (e.g., state names).
`name_treated_unit`	Character. Name of the treated unit (e.g., the state that received treatment).
`col_name_period`	Character. Column name in dataset containing time periods.
`treated_period`	Numeric. Time period when treatment starts for the treated unit.
`min_period`	Numeric. Earliest time period in the dataset.
`end_period`	Numeric. Latest time period in the dataset.

outcome_models	Character vector. Outcome models to fit. Default is c("none", "ols", "ridge", "lasso", "augsynth").
feature_weights	Character. Method for assigning weights to predictors ("uniform" or "optimize"). Default is "uniform".
w.constr	Optional. Constraints on the weights.
V	Character. Covariance matrix estimation method. Default is "separate".
V.mat	Optional. Pre-computed covariance matrix.
solver	Character. Solver to use for the optimization problem. Default is "ECOS".
constant	Logical. Whether to include a constant term in synthetic control weights. Default is FALSE.

Value

A list of class "scm_estimate" containing estimated SCM results and input parameters.

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
fit <- estimate_sc(
  dataset = toy,
  outcome = "y",
  covagg = list(
    list(var = "outcome_var", partition_periods = list(type = "by_period"))
  ),
  col_name_unit_name = "unit",
  name_treated_unit = "treated",
  col_name_period = "year",
  treated_period = 5,
  min_period = 1,
  end_period = 6,
  outcome_models = "none"
)
names(fit)
```

---

formula-interface *Formula-Based Interface for SCMs*

Description

Provides a formula interface for synthetic control methods, similar to lm() and other standard R modeling functions. This makes the package more accessible to users familiar with standard R model syntax while maintaining compatibility with data.table workflows.

Details

Formula Interface for Synthetic Control Methods

---

glance.scest *Extract Model-Level Summary Statistics*

Description

Extract model-level fit statistics and metadata from synthetic control estimation results in a single-row data.table.

Usage

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

Arguments

x	An object of class "scest" from `scest()`
...	Additional arguments (not currently used)

Value

A single-row data.table with columns:

- `n_donors` - Integer. Number of donor units
- `n_pre_periods` - Integer. Number of pre-treatment periods
- `n_post_periods` - Integer. Number of post-treatment periods
- `rmse_pre` - Numeric. Root mean squared error in pre-treatment fit
- `rmse_post` - Numeric. Root mean squared error in post-treatment prediction
- `constraint_type` - Character. Type of weight constraint used
- `feature_weighting` - Character. Feature weighting method used
- `n_active_donors` - Integer. Number of donors with meaningful weights
- `max_weight` - Numeric. Largest donor weight
- `herfindahl_index` - Numeric. Concentration measure of weights
- `has_constant` - Logical. Whether constant term was included

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
sc_obj <- scdata(
  df = toy,
  id.var = "unit",
  time.var = "year",
  outcome.var = "y",
  period.pre = 1:4,
  period.post = 5:6,
  unit.tr = "treated",
  unit.co = c("donor1", "donor2"),
  covagg = list(list(var = "outcome_var", partition_periods = list(type = "by_period")))
)
fit <- scest(sc_obj, w.constr = list(name = "simplex"))
generics::glance(fit)
```

---

inference_placebo *Perform placebo inference with multiple test statistics*

Description

Performs placebo inference by estimating synthetic control for each control unit and computing three test statistics: post/pre RMSE ratio, treatment effect, and normalized treatment effect. P-values are one-sided in `expected_direction`, or two-sided (ranked by |statistic|) when `expected_direction = "two_sided"`. The RMSPE ratio is always one-sided.

Usage

```
inference_placebo(
  sc.pred,
  dataset,
  cores,
  verbose,
  expected_direction = "two_sided"
)
```

Arguments

`sc.pred`	List. Results from synthetic control estimation
`dataset`	Data.frame. Original dataset used for estimation
`cores`	Numeric. Number of cores for parallel processing

verbose Logical. Whether to print progress information

expected_direction
Character. Expected direction of treatment effect: "negative", "positive", or "two_sided" (default: "two_sided")

Details

Placebo Inference for Synthetic Control Method

Value

List containing:

- taus: Treatment effects for all units
- rmse: RMSE values for all units
- abadie_significance: List with three elements:
  - rmse_ratio: Post/pre RMSE ratio test statistic and p-values
  - treatment_effect: Treatment effect test statistic and p-values
  - normalized_te: Normalized treatment effect test statistic and p-values

---

inference_sc *Perform Placebo Inference for Synthetic Control Results*

Description

Wrapper around `inference_placebo` for rank-based placebo inference.

Usage

```
inference_sc(
  sc.pred,
  dataset,
  cores = 1,
  verbose = TRUE,
  expected_direction = "two_sided"
)
```

Arguments

sc.pred List. Synthetic-control estimation result, typically from `estimate_sc`.

dataset Data frame. Original panel dataset.

cores Integer. Number of cores for placebo runs. Default is 1.

verbose Logical. Whether to print progress information. Default is TRUE.

expected_direction
Character. Expected direction of treatment effect: `"negative"`, `"positive"`, or `"two_sided"`. Default is `"two_sided"`.

Value

A list with data, est.results, inference.results, rmse, and abadie_significance.

Examples

```
set.seed(1)
toy <- expand.grid(unit = c("treated", "donor1", "donor2", "donor3"), year = 1:10,
                   stringsAsFactors = FALSE)
toy$y <- 50 + 0.8 * toy$year + ifelse(toy$unit == "treated" & toy$year >= 7, 3, 0) +
  rnorm(nrow(toy), 0, 0.5)
sc_fit <- estimate_sc(
  dataset = toy,
  outcome = "y",
  covagg = list(
    list(var = "outcome_var", partition_periods = list(type = "by_period"))
  ),
  col_name_unit_name = "unit",
  name_treated_unit = "treated",
  col_name_period = "year",
  treated_period = 7,
  min_period = 1,
  end_period = 10,
  outcome_models = "none"
)
inf <- inference_sc(sc_fit, toy, cores = 1, verbose = FALSE)
names(inf)
```

---

perform_bootstrap_inference

Perform Bootstrap Inference

Description

Perform Bootstrap Inference

Usage

```
perform_bootstrap_inference(
  dataset_null,
  sc.pred,
  actual_effects,
  n_bootstrap,
  cores,
  verbose
)
```

Arguments

dataset_null	Dataset with null hypothesis enforced
sc.pred	Original SC estimation results
actual_effects	List of actual treatment effects
n_bootstrap	Number of bootstrap replications
cores	Number of cores for parallel processing
verbose	Logical for verbose output

Value

List with bootstrap effects and estimates

---

plot.spec_curve *Plot Method for Specification Curve Results*

Description

Plot method that creates a specification curve visualization by calling the main plotting function with appropriate defaults.

Usage

```
## S3 method for class 'spec_curve'
plot(x, ...)
```

Arguments

x	An object of class "spec_curve" from spec_curve
...	Additional arguments passed to plot_spec_curve

Details

This is a convenience method that calls plot_spec_curve with the specification curve data. All arguments supported by plot_spec_curve can be passed through the ... parameter.

Value

A ggplot object containing the specification curve plot

See Also

plot_spec_curve for detailed plotting options

Examples

```
specs <- data.table::data.table(
  full_spec_id = paste0("s", 1:4),
  outcome_model = c("none", "ridge", "none", "ridge"),
  const = c("simplex", "lasso", "simplex", "lasso"),
  fw = c("uniform", "uniform", "optimize", "optimize"),
  feat = c("f1", "f1", "f2", "f2")
)
dt <- data.table::CJ(
  unit_name = c("treated", "donor1"),
  full_spec_id = as.character(specs[["full_spec_id"]]),
  post_period = c(FALSE, TRUE),
  sorted = FALSE
)
dt[, unit_type := ifelse(unit_name == "treated", "treated", "control")]
dt <- merge(dt, specs, by = "full_spec_id", all.x = TRUE)
dt[, outcome := "y"]
dt[, rmse := 0.2 + as.numeric(factor(full_spec_id)) * 0.01]
dt[, tau := ifelse(
  unit_name == "treated",
  as.numeric(factor(full_spec_id)) * 0.5 + ifelse(post_period, 0.2, 0),
  as.numeric(factor(full_spec_id)) * 0.1 + ifelse(post_period, 0.05, 0)
)]
spec_results <- list(results = dt)
class(spec_results) <- "spec_curve"
plot(spec_results, show_shap = FALSE, show_pvalues = FALSE)
```

---

plot_spec_curve *Plot Specification Curve with Internal SHAP Computation and Filtering*

Description

Creates specification curve plots directly from long format data with integrated SHAP analysis and flexible filtering capabilities. Key features:

- Internal SHAP computation using XGBoost (automatic when show_shap=TRUE)
- Specification filtering BEFORE SHAP computation and p-value calculation
- Perfect alignment between SHAP values and specifications via full_spec_id
- FAIL HARD error handling with specific guidance when requirements aren't met

Usage

```
plot_spec_curve(
  long_data,
  name_treated_unit,
```

```
  outcomes = NULL,
  normalize_outcomes = "none",
  rmse_threshold = Inf,
  shap_values = NULL,
  file_path_save = NA,
  width = 6,
  height = 10,
  show_pvalues = FALSE,
  p_threshold = 0.05,
  prefer_bootstrap_pvalues = FALSE,
  curve_stat = "abs_median_tau",
  weighting = c("none", "pre_rmspe_percentile"),
  two_sided = TRUE,
  grid_policy = "strict",
  min_placebos = 1L,
  min_specs = 2L,
  test_statistic = "rmse_ratio",
  null_distribution = "placebo",
  crop_outliers = "none",
  sort_by = "tau",
  filter_specs = NULL,
  show_shap = TRUE,
  shap_label_type = "absolute",
  show_predictions = FALSE,
  predictions = NULL,
  xgboost_params = NULL,
  tune_xgboost = NULL,
  xgboost_grid = NULL,
  xgboost_cv_folds = 5
)
```

Arguments

long_data	Data.table or List. Long format data from spec_curve(). Can be either the results data.table directly, or the full structured results (with results, abadie_inference, bootstrap_inference components).
name_treated_unit	Character. Name of the treated unit.
outcomes	Character vector or NULL. Names of the outcome variables to plot. If NULL, all outcomes in the data will be plotted.
normalize_outcomes	Character. Method for normalizing treatment effects for comparability. Options: "none" (default), "percent_of_mean", "percent_of_preperiod", "standardized", "rmspe_ratio". • "percent_of_mean": (tau / mean(pre-period treated)) * 100 • "percent_of_preperiod": (tau / final_preperiod_value) * 100 • "standardized": tau / sd(control_effects)

- "rmspe_ratio": post_RMSPE / pre_RMSPE (Abadie-style)

`rmse_threshold` Numeric. Threshold for root mean square error to filter results. Default is Inf.

`shap_values` Data.table or NULL. External SHAP values from run_xgboost_shap_analysis(). Should contain columns: unit, full_spec_id, feature_group, feature, shapley_value. If NULL and show_shap=TRUE, SHAP values will be computed internally. Default is NULL.

`file_path_save` Character or NA. File path to save the plot. If NA, plot is not saved. Default is NA.

`width` Numeric. Width of the saved plot in inches. Default is 6.

`height` Numeric. Height of the saved plot in inches. Default is 10.

`show_pvalues` Logical. Whether to include significance coloring for treatment effects. Uses Abadie p-values by default, or bootstrap p-values if available. Default is TRUE.

`p_threshold` Numeric. P-value threshold for significance coloring. Default is 0.05.

`prefer_bootstrap_pvalues`
Logical. Whether to prefer bootstrap p-values over Abadie p-values when both are available. Default is FALSE (prefer Abadie).

`curve_stat` Character vector. Curve-level statistics to compute for placebo-in-space inference. Options:

- `"median_tau"`: $\mathrm{median}_s(\tau_s)$ (signed; the only option that supports `two_sided = FALSE`)
- `"abs_median_tau"`: $|\mathrm{median}_s(\tau_s)|$
- `"median_abs_tau"`: $\mathrm{median}_s(|\tau_s|)$
- `"consistency_ratio"`: $|\sum_s \tau_s| / \sum_s |\tau_s|$

Default is `"abs_median_tau"`.

`weighting` Character vector. Weighting modes to evaluate for curve-level statistics. Options: `"none"` and `"pre_rmspe_percentile"`. Default is `c("none", "pre_rmspe_percentile")`. Weighting is applied generically to each requested curve statistic.

`two_sided` Logical. If `TRUE` (default), curve-level p-values use absolute-tail placebo comparison: `(1 + sum(abs(placebo) >= abs(observed))) / (J + 1)`. If `FALSE`, one-sided comparison is used with `expected_direction` from `long_data` (`"negative"` or `"positive"` required); this requires `curve_stat = "median_tau"`. For `"negative"`, `p = (1 + sum(placebo <= observed)) / (J + 1)`.

`grid_policy` Character. How to handle incomplete placebo specification grids when computing curve-level p-values. Options:

- `"strict"` (default): fail if any placebo unit is missing/has extra specs.
- `"drop_incomplete_units"`: keep full treated spec grid, drop placebo units that do not exactly match it.
- `"intersect_specs"`: keep all units, restrict to specs shared by all units.

`min_placebos` Integer. Minimum number of placebo units required after applying `grid_policy`. Default is 1.

`min_specs` Integer. Minimum number of specifications required after applying `grid_policy`. Default is 2.

`test_statistic` Character. Which test statistic p-values to use for coloring. Options: "rmse_ratio" (default), "treatment_effect", "normalized_te". Only applies to Abadie placebo inference which provides multiple test statistics.

`null_distribution`
Character. The distribution to plot as the null/comparison. Options: "placebo" (default) or "bootstrap".

- "placebo": Use placebo effects from control units as the null distribution.
- "bootstrap": Use the bootstrapped null distribution for the treated unit.

`crop_outliers` Character or Numeric. Method for cropping outliers in Panel A. Options: "none" (default), "percentile", "iqr", "mad", or numeric vector c(ymin, ymax).

- "percentile": Crop to 1st-99th percentile of treated unit effects
- "iqr": Crop to Q1 - 1.5*IQR to Q3 + 1.5*IQR
- "mad": Crop to median +/- 3*MAD (robust outlier detection)
- c(ymin, ymax): Manual y-axis limits

`sort_by` Character. Method for sorting specifications on x-axis. Options: "tau" (default), "pvalue", "rmspe_ratio".

- "tau": Sort by treatment effect magnitude (original behavior)
- "pvalue": Sort by statistical significance (most significant first)
- "rmspe_ratio": Sort by post/pre RMSPE ratio

`filter_specs` Named list or NULL. Filters to apply to specifications BEFORE SHAP computation and p-value calculation. Each element should be named by the feature group column and contain allowed values. Available filters: "constant" (TRUE/FALSE), "const" (simplex/lasso/ridge/pensynth), "outcome_model" (none/augsynth/lasso/ridge/o "fw" (uniform/optimize), "feat" (covariate aggregation labels), "data_sample" (all/most_similar). Examples: list(constant = "TRUE"), list(const = c("simplex", "lasso")), list(constant = "TRUE", outcome_model = c("none", "augsynth")). Default is NULL (no filtering).

`show_shap` Logical. Whether to compute and display SHAP values. When TRUE:

- If shap_values is provided: uses external SHAP values
- If shap_values is NULL: computes SHAP values internally using XGBoost
- Requires at least 3 unique specifications for internal computation
- Automatically detects available specification features (outcome_model, const, fw, feat, data_sample, constant) Default is TRUE.

`shap_label_type`
Character. How to display SHAP values in y-axis labels: "absolute" or "signed".

- "absolute": Shows mean |SHAP| values, e.g., "ridge (0.123)"
- "signed": Shows mean SHAP values with sign, e.g., "ridge (+0.089)" or "lasso (-0.045)" Default is "absolute".

`show_predictions`
Logical. Whether to display predicted treatment effects from XGBoost models alongside actual treatment effects in Panel A. Uses leave-one-out cross-validation predictions for robust evaluation. Default is FALSE.

`predictions` Data.table or NULL. External LOO predictions from run_xgboost_shap_analysis(). Should contain columns: unit, full_spec_id, predicted_loo, actual. If NULL and show_predictions=TRUE, predictions will be computed internally. Default is NULL.

`xgboost_params` List. Custom parameters for XGBoost model training. If NULL, uses defaults: list(objective='reg:squarederror', max_depth=10, eta=0.05, nrounds=500, subsample=0.8, colsample_bytree=0.8, nthread=1, seed=42, verbose=0). Common parameters to modify: nrounds (more = better fit but slower), max_depth (complexity), eta (learning rate). Default is NULL. If tune_xgboost is TRUE, xgboost_params must be NULL.

`tune_xgboost` Logical or NULL. If TRUE, tune XGBoost hyperparameters via k-fold CV on the treated unit before SHAP/LOO. If NULL, defaults to TRUE when xgboost_params is NULL and FALSE otherwise. Default is NULL.

`xgboost_grid` Data.frame, data.table, or list. Optional grid for tuning. Must include max_depth, eta, nrounds, subsample, colsample_bytree. Default is NULL.

`xgboost_cv_folds`
Integer. Number of CV folds for tuning. Default is 5.

Details

Plot Specification Curve Using Direct Long Format Data

Panel B labels are rich text. To save `final_plot` as SVG, use the Cairo device (`grDevices::svg`); svglite misaligns the labels.

Value

List containing:

- final_plot: ggplot object representing the complete specification curve with Panel A (treatment effects) and Panel B (feature groups/SHAP)
- panel_a: ggplot object for Panel A only (treatment effects)
- panel_b: ggplot object for Panel B only (feature groups/SHAP)
- plot_data_p1: data.table with Panel A plotting data including columns: Unit Name, Estimate, RMSE, Specification, unit_type, p_value (if available)
- plot_data_p2: data.table with Panel B plotting data including columns: Specification, feature_group, feature, shapley_value (if SHAP computed)
- computed_shap: Complete results from internal SHAP computation (NULL if external shap_values provided or show_shap=FALSE). Contains: results (feature importance), shapley (SHAP values), predictions (model predictions), models (trained XGBoost models), config (SHAP configuration)
- spec_curve_pvals: List with specification curve-level p-values calculated on filtered data: `treated_summary` (treated-unit curve estimates and p-values) and `stats_by_unit` (curve statistics for treated and placebo units), plus `attrition_report` summarizing unit/spec retention under `grid_policy`. NULL if no inference data available.
- filtered_specs: Integer count of specifications remaining after filtering (before filtering if filter_specs=NULL)

- feature_groups_displayed: Character vector of feature groups shown in Panel B (only groups with variation in filtered data)

Examples

```
specs <- data.table::data.table(
  full_spec_id = paste0("s", 1:4),
  outcome_model = c("none", "ridge", "none", "ridge"),
  const = c("simplex", "lasso", "simplex", "lasso"),
  fw = c("uniform", "uniform", "optimize", "optimize"),
  feat = c("f1", "f1", "f2", "f2")
)
long_data <- data.table::CJ(
  unit_name = c("treated", "donor1"),
  full_spec_id = as.character(specs[["full_spec_id"]]),
  post_period = c(FALSE, TRUE),
  sorted = FALSE
)
long_data[, unit_type := ifelse(unit_name == "treated", "treated", "control")]
long_data <- merge(long_data, specs, by = "full_spec_id", all.x = TRUE)
long_data[, outcome := "y"]
long_data[, rmse := 0.2 + as.numeric(factor(full_spec_id)) * 0.01]
long_data[, tau := ifelse(
  unit_name == "treated",
  as.numeric(factor(full_spec_id)) * 0.5 + ifelse(post_period, 0.2, 0),
  as.numeric(factor(full_spec_id)) * 0.1 + ifelse(post_period, 0.05, 0)
)]
out <- plot_spec_curve(
  long_data = long_data,
  name_treated_unit = "treated",
  show_shap = FALSE,
  show_pvalues = FALSE,
  curve_stat = "abs_median_tau",
  weighting = "none"
)
out$panel_a
```

---

print.scdata *Summary Method for Synthetic Control*

Description

The print method for synthetic control data objects.

Usage

```
## S3 method for class 'scdata'
print(x, ...)
```

Arguments

x	Class "scdata" object, obtained by calling scdata.
...	Other arguments.

Value

No return value, called to print scdata results.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

See Also

scdata for synthetic control data preparation.

Supported methods: print.scdata, summary.scdata.

---

print.scest *Print Method for Synthetic Control Methods*

Description

The print method for synthetic control prediction fitted objects.

Usage

```
## S3 method for class 'scest'
print(x, ...)
```

Arguments

x	Class "scest" object, obtained by calling scest.
...	Other arguments.

Value

No return value, called to print scest results.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

See Also

scest for synthetic control prediction.

Supported methods: print.scest, summary.scest, coef.scest.

---

print.scest_formula *Print Method for Formula-Based Synthetic Control*

Description

Print method for scest_formula objects

Usage

```
## S3 method for class 'scest_formula'
print(x, ...)
```

Arguments

x	An scest_formula object
...	Additional arguments

---

run_spec_curve_analysis *Run Specification Curve Analysis with Multiple Configurations*

Description

Performs specification curve analysis across multiple modeling choices and configurations, providing a systematic exploration of researcher degrees of freedom.

Usage

```
run_spec_curve_analysis(dataset, params, cores = 1, output_dir = NULL)
```

Arguments

dataset	Data frame containing the panel data for analysis.
params	List containing ALL analysis parameters including: • outcomes - Character vector of outcome variables (REQUIRED) • col_name_unit_name - Column name for unit identifiers • name_treated_unit - Name of treated unit • covagg - List of covariate specifications • treated_period - Treatment start period

- `min_period` - Minimum period for analysis
- `end_period` - Maximum period for analysis
- `col_name_period` - Column name for time periods
- `feature_weights` - Feature weighting methods
- `donor_sample` - Donor sample selection methods
- `outcome_models` - Outcome modeling approaches
- `constraints` - Weight constraint specifications
- `inference_type` - "placebo", "bootstrap", or "all" (default: "placebo")
- `inference_config` - List with bootstrap_n_replications, verbose, etc.
- `expected_direction` - "negative", "positive", or "two_sided" (default: "two_sided")

`cores` Integer. Number of cores for parallel processing (applies to both bootstrap and placebo inference unless overridden in params$inference_config).

`output_dir` Character. Directory to save results. If NULL, results are not saved.

Details

Run Comprehensive Specification Curve Analysis

Value

List structure containing:

- `results` - Data.table in long format with all specification results
- `abadie_inference` - Abadie placebo inference results (if inference_type includes "placebo")
- `bootstrap_inference` - Bootstrap inference results (if inference_type includes "bootstrap")

Examples

```
set.seed(1)
toy <- expand.grid(unit = c("treated", "donor1", "donor2", "donor3"), year = 1:10,
                   stringsAsFactors = FALSE)
toy$y <- 50 + 0.8 * toy$year + ifelse(toy$unit == "treated" & toy$year >= 7, 3, 0) +
  rnorm(nrow(toy), 0, 0.5)
params <- list(
  outcomes = "y",
  col_name_unit_name = "unit",
  name_treated_unit = "treated",
  covagg = list(
    baseline = list(
      label = "baseline",
    operations = list(list(var = "outcome_var", partition_periods = list(type = "by_period")))
    )
  ),
  treated_period = 7,
  min_period = 1,
  end_period = 10,
  col_name_period = "year",
  feature_weights = "uniform",
  donor_sample = "all",
```

```
  outcome_models = "none",
  constraints = list(list(name = "simplex")),
  inference_type = "placebo",
  inference_config = list(verbose = FALSE, placebo_cores = 1)
)
results <- run_spec_curve_analysis(dataset = toy, params = params, cores = 1)
names(results)
```

`run_xgboost_shap_analysis` *Run Direct XGBoost Analysis on Long Format Data*

Description

Performs XGBoost modeling and SHAP analysis directly on long format data produced by `spec_curve()` or `run_spec_curve_analysis()`.

Usage

```
run_xgboost_shap_analysis(long_data, config, compute_loo = TRUE)
```

Arguments

`long_data`	Data.table. Long format data (typically `spec_results$results`).
`config`	List. Configuration object from `create_xgboost_config`.
`compute_loo`	Logical. If TRUE (default), compute leave-one-out predictions.

Details

Run XGBoost SHAP Analysis on Long Format Data

Value

List containing:

- `results` - Feature importance results
- `shapley` - SHAP values with proper spec_number alignment
- `predictions` - Model predictions
- `config` - Original configuration object

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  long_data <- data.table::data.table(
    unit_name = rep("treated", 6),
    full_spec_id = paste0("s", 1:6),
    spec_number = 1:6,
    post_period = TRUE,
    tau = c(0.4, 0.6, 0.2, 0.8, 0.3, 0.7),
    outcome_model = c("none", "ridge", "none", "ridge", "none", "ridge"),
    const = c("simplex", "lasso", "simplex", "lasso", "simplex", "lasso"),
    fw = c("uniform", "uniform", "optimize", "optimize", "uniform", "optimize"),
    feat = c("f1", "f1", "f2", "f2", "f3", "f3")
  )
  xgb_cfg <- create_xgboost_config(
    dataset_name = "toy",
    treated_unit_name = "treated",
    spec_features = c("outcome_model", "const", "fw", "feat"),
    tune_xgboost = FALSE,
    xgboost_params = list(
      objective = "reg:squarederror",
      max_depth = 2,
      eta = 0.1,
      nrounds = 20,
      subsample = 1,
      colsample_bytree = 1,
      nthread = 1,
      seed = 42,
      verbose = 0
    )
  )
  shap_results <- run_xgboost_shap_analysis(
    long_data = long_data,
    config = xgb_cfg,
    compute_loo = FALSE
  )
  names(shap_results)
}
```

---

scdata *Create Synthetic Control Data Structure*

Description

This function prepares a data structure for use with Synthetic Control Methods (SCM) by formatting and organizing the input data. It validates inputs, structures the data appropriately, and creates matrices needed for synthetic control estimation.

Usage

```
scdata(
  df,
  id.var,
  time.var,
  outcome.var,
  period.pre,
  period.post,
  unit.tr,
  unit.co,
  anticipation = 0,
  constant = FALSE,
  verbose = TRUE,
  covagg = list(),
  cointegrated.data = FALSE
)
```

Arguments

df	Data frame containing the panel data in long format.
id.var	Character. Name of the column containing unit identifiers (e.g., state names, country codes).
time.var	Character. Name of the column containing time periods (e.g., year, quarter).
outcome.var	Character. Name of the column containing the outcome variable of interest.
period.pre	Numeric vector. Pre-treatment time periods used for fitting the synthetic control.
period.post	Numeric vector. Post-treatment time periods for evaluating treatment effects.
unit.tr	Character. Name or identifier of the treated unit.
unit.co	Character vector. Names or identifiers of the potential control units (donor pool).
anticipation	Numeric. Number of pre-treatment periods to exclude due to anticipation effects. Default is 0.
constant	Logical. Whether to include a global constant (intercept) term in synthetic control estimation. When TRUE, adds a constant feature that allows the synthetic control to have a level shift relative to donors. Requires multiple features (more than one covariate specification). The constant term is estimated alongside donor weights and can be positive or negative. Default is FALSE.
verbose	Logical. Whether to print diagnostic messages and warnings. Default is TRUE.
covagg	List of covariate-aggregation operations. Each entry must be a list with: • var: variable name, or "outcome_var" for outcome.var • select_periods: optional period selector • partition_periods: optional grouping of selected periods • compute: optional aggregation function/function name (default "mean") If covagg is empty, scdata() defaults to outcome per-period matching over all pre-treatment periods.
cointegrated.data	Logical. Whether to treat the data as cointegrated (affects some computations). Default is FALSE.

Details

This function performs several key operations:

- Validates that all required variables exist in the data
- Checks for balanced panel structure
- Creates outcome matrices for treated and control units
- Handles missing data appropriately
- Prepares data structures required by estimation functions

The function expects panel data in long format where each row represents one unit-time observation.

Value

A list of class 'scdata' containing:

- `Y.pre`: Matrix of pre-treatment outcomes (controls × time periods)
- `Y.post`: Matrix of post-treatment outcomes
- `Y.donors`: Full outcome matrix for donor units
- `Y.treated`: Outcome vector for treated unit
- `specs`: List containing specification details
- Additional metadata for estimation and plotting

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
scm_data <- scdata(
  df = toy,
  id.var = "unit",
  time.var = "year",
  outcome.var = "y",
  period.pre = 1:4,
  period.post = 5:6,
  unit.tr = "treated",
  unit.co = c("donor1", "donor2"),
  covagg = list(
    list(var = "outcome_var", partition_periods = list(type = "by_period"))
  )
)
class(scm_data)
```

scest *Synthetic Control Estimation for Single Treated Unit*

Description

This function estimates synthetic control weights for a single treated unit using various constraint specifications and optimization methods. It implements the core synthetic control methodology with flexible constraint options and feature weighting schemes.

Usage

```
scest(
  data,
  w.constr = NULL,
  feature_weights = "uniform",
  V = "separate",
  V.mat = NULL,
  solver = "ECOS",
  save.data = NULL,
  margin.ipop = 0.05,
  sigf.ipop = 5,
  bound.ipop = 10
)
```

Arguments

`data`	A list of class "scdata" containing the prepared data from `scdata()`.
`w.constr`	List specifying constraints on the synthetic control weights. Options include: • `list(name = "simplex")` - Standard simplex constraint (weights sum to 1, non-negative) • `list(name = "lasso", Q = 0.1)` - L1 penalty with regularization parameter Q • `list(name = "ridge", Q = 0.1)` - L2 penalty with regularization parameter Q • `list(name = "ols")` - Ordinary least squares (no constraints) • `list(name = "L1-L2", Q1 = 0.1, Q2 = 0.1)` - Combined L1 and L2 penalties Default is NULL (no constraints).
`feature_weights`	Character. Method for weighting pre-treatment periods/features: • `"uniform"` - Equal weights for all periods (default) • `"optimize"` - Optimize feature weights using nested optimization
`V`	Character or matrix specifying the feature weighting approach: • `"separate"` - Optimize V matrix separately (default)

- "nested" - Nested optimization of V and W matrices
- Custom matrix - User-provided V matrix

V.mat	Matrix. Custom V (feature weighting) matrix. If provided, overrides V parameter.
solver	Character. CVXR solver to use ("ECOS", "OSQP", "GUROBI", etc.). Must be installed on system. Default is "ECOS".
save.data	Logical. Whether to save input data in output object (for debugging). Default is NULL.
margin.ipop	Numeric. Convergence margin passed to the inner interior-point quadratic program when feature_weights = "optimize". Defaults to 0.05. Note that Synth's documented default is 0.0005, i.e. 100x tighter; a loose margin can leave the inner solution short of the optimum, which matters when estimates are compared across different V.
sigf.ipop	Numeric. Significant digits required of the inner solver. Default 5.
bound.ipop	Numeric. Bound passed to the inner solver. Default 10.

Details

The function implements the synthetic control method by solving the optimization problem:

$$\min_W ||X_1 - X_0 W||_V^2 + penalty(W)$$

where X_1 is the treated unit's pre-treatment characteristics, X_0 is the control units' characteristics matrix, W are the synthetic control weights, and V is the feature weighting matrix.

Different constraint types impose different penalty functions:

- Simplex: $\sum W_i = 1, W_i \geq 0$
- Lasso: $Q \sum |W_i|$
- Ridge: $Q \sum W_i^2$
- OLS: No constraints

Value

A list of class "scest" containing:

- est.results$w - Vector of synthetic control weights for donor units
- est.results$constant_term - Constant (intercept) coefficient if constant=TRUE was used in scdata, NULL otherwise
- est.results$b - Full coefficient vector (donor weights + constant term)
- est.results$Y.pre.fit - Fitted pre-treatment outcomes
- est.results$Y.post.fit - Fitted post-treatment outcomes
- est.results$V - Feature weighting matrix used
- est.results$w.constr - Weight constraint specification used
- data - Original scdata object with estimation matrices
- Additional estimation metadata and diagnostics

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
scm_data <- scdata(
  df = toy,
  id.var = "unit",
  time.var = "year",
  outcome.var = "y",
  period.pre = 1:4,
  period.post = 5:6,
  unit.tr = "treated",
  unit.co = c("donor1", "donor2"),
  covagg = list(
    list(var = "outcome_var", partition_periods = list(type = "by_period"))
  )
)
result <- scest(data = scm_data, w.constr = list(name = "simplex"))
names(result$est.results)
```

---

scplot *Plot Synthetic Control Point Estimates and Prediction Interval*

Description

Plot treated and synthetic time series from a fitted `scest` object.

Companion Stata and Python packages are described in Cattaneo, Feng, Palomba, and Titiunik (2022).

Companion commands are scdata for data preparation, scest for point estimation, and inference_sc for inference procedures.

For an introduction to synthetic control methods, see Abadie (2021) and references therein.

Usage

```
scplot(
  result,
  fig.path = NULL,
  fig.name = NULL,
  fig.format = "png",
  col.treated = "black",
  col.synth = "mediumblue",
  label.xy = NULL,
```

```
  plot.range = NULL,
  x.ticks = NULL,
  event.label = NULL,
  plot.specs = NULL,
  save.data = NULL
)
```

Arguments

result	A class "scest" object from scest.
fig.path	a string indicating the path where the plot(s) should be saved.
fig.name	a string indicating the name of the plot(s). If multiple plots will be saved the command automatically generates a numeric suffix to avoid overwriting them.
fig.format	a string indicating the format in which the plot(s) should be saved.
col.treated	a string specifying the color for the treated unit series. Find the full list at https://sape.inf.usi.ch/quick-reference/ggplot2/colour.
col.synth	a string specifying the color for the synthetic unit series. Find the full list at https://sape.inf.usi.ch/quick-reference/ggplot2/colour.
label.xy	a character list with two elements indicating the name of the axes (eg. label.xy = list(x.lab = "Year", y.lab = "GDP growth (%)")).
plot.range	a numeric array indicating the time range of the plot(s).
x.ticks	a numeric list containing the location of the ticks on the x axis.
event.label	a list containing a character object ('lab') indicating the label of the event and a numeric object indicating the height of the label in the plot.
plot.specs	a list containing some specifics to be passed to ggsave (eg. img.width, img.height, dpi)
save.data	a character specifying the name and the path of the saved dataframe containing the processed data used to produce the plot.

Value

plots	a list containing standard ggplot object(s) that can be used for further customization.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

References

- Abadie, A. (2021). Using synthetic controls: Feasibility, data requirements, and methodological aspects. *Journal of Economic Literature*, 59(2), 391-425.

See Also

scdata, scest, inference_sc

Examples

```
# Create example data
df <- data.frame(
  unit = rep(c("treated", "control1", "control2"), each = 20),
  time = rep(1:20, 3),
  outcome = c(rnorm(20, mean = 1:20), rnorm(20, mean = 1:20), rnorm(20, mean = 1:20))
)

# Prepare data for SCM
scm_data <- scdata(df = df, id.var = "unit", time.var = "time",
                   outcome.var = "outcome", period.pre = 1:10,
                   period.post = 11:20, unit.tr = "treated",
                   unit.co = c("control1", "control2"))

# Estimate synthetic control
result <- scest(scm_data, w.constr = list(name = "simplex"))

# Plot results
scplot(result)
```

---

solve_qp_clarabel *Solve QP using Clarabel for V optimization*

Description

Solve quadratic programming problem using clarabel solver

Usage

```
solve_qp_clarabel(c, H, A, b, l, u)
```

Arguments

c	Linear objective coefficients
H	Quadratic objective matrix (Hessian)
A	Equality constraint matrix
b	Equality constraint RHS
l	Lower bounds
u	Upper bounds

Value

Solution vector as matrix

SpecResultsManager *Specification Results Storage Manager*

Description

R6 class for managing large specification curve results with disk-based storage, compression, and memory-efficient retrieval patterns.

Public fields

`storage_dir` Directory for storing specification results

`compression_level` Compression level (0-9, higher = more compression)

`max_memory_mb` Maximum memory usage in MB before offloading to disk

`current_memory_usage` Current memory usage tracking

Methods

Public methods:

- `SpecResultsManager$new()`
- `SpecResultsManager$store_results()`
- `SpecResultsManager$retrieve_results()`
- `SpecResultsManager$get_filtered_results()`
- `SpecResultsManager$update_metadata()`
- `SpecResultsManager$get_metadata()`
- `SpecResultsManager$check_memory_usage()`
- `SpecResultsManager$cleanup_memory()`
- `SpecResultsManager$get_storage_summary()`
- `SpecResultsManager$cleanup_storage()`
- `SpecResultsManager$merge_specification_sets()`
- `SpecResultsManager$clone()`

`SpecResultsManager$new()`**:** Initialize storage manager

Usage:

```
SpecResultsManager$new(
  storage_dir = tempdir(),
  max_memory_mb = 1000,
  compression_level = 6
)
```

Arguments:

`storage_dir` Character, directory path for storage

`max_memory_mb` Numeric, maximum memory in MB

`compression_level` Integer 0-9, compression level

`SpecResultsManager$store_results()`: Store specification results efficiently

Usage:

```
SpecResultsManager$store_results(spec_results, spec_id, metadata = list())
```

Arguments:

`spec_results` List or data.table with specification results

`spec_id` Character, unique identifier for this specification set

`metadata` List with additional metadata

`SpecResultsManager$retrieve_results()`: Retrieve specification results

Usage:

```
SpecResultsManager$retrieve_results(spec_id, lazy = TRUE)
```

Arguments:

`spec_id` Character, specification set identifier

`lazy` Logical, whether to use lazy loading for large results

`SpecResultsManager$get_filtered_results()`: Get filtered subset of results without loading all data

Usage:

```
SpecResultsManager$get_filtered_results(
  spec_id,
  filter_function = NULL,
  select_cols = NULL
)
```

Arguments:

`spec_id` Character, specification set identifier

`filter_function` Function to apply for filtering

`select_cols` Character vector of columns to select

`SpecResultsManager$update_metadata()`: Update metadata for stored results

Usage:

```
SpecResultsManager$update_metadata(
  spec_id,
  file_path,
  spec_results,
  additional_metadata = list()
)
```

Arguments:

`spec_id` Character, specification identifier

`file_path` Character, path to results file

`spec_results` The results object

`additional_metadata` List with additional metadata

`SpecResultsManager$get_metadata()`: Get metadata for all stored results

Usage:

```
SpecResultsManager$get_metadata()
```

`SpecResultsManager$check_memory_usage()`: Check current memory usage and trigger cleanup if needed

Usage:

```
SpecResultsManager$check_memory_usage()
```

`SpecResultsManager$cleanup_memory()`: Clean up memory by removing old cached objects

Usage:

```
SpecResultsManager$cleanup_memory()
```

`SpecResultsManager$get_storage_summary()`: Get summary statistics about stored specifications

Usage:

```
SpecResultsManager$get_storage_summary()
```

`SpecResultsManager$cleanup_storage()`: Clean up storage directory

Usage:

```
SpecResultsManager$cleanup_storage(older_than_days = 30)
```

Arguments:

`older_than_days` Remove results older than this many days

`SpecResultsManager$merge_specification_sets()`: Merge multiple specification result sets efficiently

Usage:

```
SpecResultsManager$merge_specification_sets(spec_ids, output_spec_id)
```

Arguments:

`spec_ids` Character vector of specification IDs to merge

`output_spec_id` Character, ID for merged results

`SpecResultsManager$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SpecResultsManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

spec_curve **CORE ENGINE** - *Specification Curve Computation*

Description

INTERNAL COMPUTATIONAL CORE - This function performs the core computation for specification curve analysis by systematically varying modeling choices and estimating treatment effects across all combinations. Called internally by `run_spec_curve_analysis()`. For user-facing specification curve analysis, use `run_spec_curve_analysis()` instead.

Usage

```
spec_curve(
  dataset,
  outcomes,
  col_name_unit_name,
  name_treated_unit,
  covagg = list(),
  treated_period,
  min_period,
  end_period,
  col_name_period,
  feature_weights = c("uniform", "optimize"),
  outcome_models = c("none", "augsynth", "ridge", "lasso", "ols"),
  donor_sample = c("all", "most_similar"),
  sim_function = most_similar,
 constraints = list(list(name = "ols"), list(name = "simplex"), list(name = "lasso"),
    list(name = "ridge"), list(name = "L1-L2")),
  constants = c(FALSE, TRUE),
  cores = 1,
  verbose = TRUE,
  inference_type = "placebo",
  inference_config = list(),
  expected_direction = "two_sided"
)
```

Arguments

`dataset`	Data frame containing panel data in long format with units, time periods, and outcomes.
`outcomes`	Character vector of outcome variable names to analyze. Multiple outcomes will be processed separately.
`col_name_unit_name`	Character. Name of column containing unit identifiers (e.g., "state", "country").
`name_treated_unit`	Character. Identifier of the treated unit as it appears in the data.

`covagg` List of covariate specifications for pre-treatment matching. Each top-level entry must be a named specification with optional `label` and required `operations`, where `operations` is a non-empty list of operation signatures: `list(var, select_periods, partition_periods, compute)`. Use `"outcome_var"` in `var` to reference the current outcome. Default is empty list.

`treated_period` Numeric. First time period when treatment is active for the treated unit.

`min_period` Numeric. Earliest time period available in the dataset.

`end_period` Numeric. Latest time period available in the dataset.

`col_name_period`
Character. Name of column containing time period identifiers.

`feature_weights`
Character vector of feature weighting methods:

- `"uniform"` - Equal weights for all pre-treatment periods.
- `"optimize"` - Data-driven optimization of feature weights.

Default is `c("uniform", "optimize")`.

`outcome_models` Character vector of outcome modeling approaches:

- `"none"` - Standard synthetic control (no outcome model).
- `"augsynth"` - Augmented synthetic control method.
- `"ridge"` - Ridge regression outcome model.
- `"lasso"` - Lasso regression outcome model.
- `"ols"` - OLS regression outcome model.

Default is `c("none", "augsynth", "ridge", "lasso", "ols")`.

`donor_sample` Character vector specifying donor pool selection:

- `"all"` - Use all available control units.
- `"most_similar"` - Use only most similar control units.

Default is `c("all", "most_similar")`.

`sim_function` Function used when `donor_sample` includes `"most_similar"`.

`constraints` List of constraint specifications for weight optimization. Each element should be a list with constraint parameters (e.g., `list(name = "simplex")` or `list(name = "lasso", Q = 0.1)`).

`constants` Logical vector indicating whether to include a constant term in synthetic control estimation for each specification.

`cores` Integer. Number of CPU cores for parallel processing. Default is 1 (sequential). Higher values increase speed at higher memory cost.

`verbose` Logical. Whether to display progress messages and warnings. Default is TRUE.

`inference_type` Character. Type of inference to perform:

- `"placebo"` - Abadie-style placebo inference only (default).
- `"bootstrap"` - Bootstrap null-hypothesis inference only.
- `"all"` - Both placebo and bootstrap inference.

`inference_config`
List. Configuration settings for inference methods:

- `bootstrap_n_replications` - Number of bootstrap replications (default: 1000).
- `bootstrap_cores` - Number of cores for bootstrap processing (default: 1).

`expected_direction`
: Character scalar used by direction-aware inference helpers: `"negative"`, `"positive"`, or `"two_sided"` (default).

Details

Generate Specification Curve (Core Computational Engine)

The function creates a Cartesian product of all specification choices and estimates synthetic controls for each combination. This implementation includes:

- **Parallel Processing**: Distributes computations across multiple cores.
- **Error Handling**: Captures failed estimations and returns diagnostics.
- **Progress Tracking**: Provides estimation progress updates.
- **Memory Management**: Handles large specification spaces.

The resulting specification curve can reveal:

- Sensitivity of treatment-effect estimates to modeling choices.
- Distribution of effect sizes across specifications.
- Which modeling choices are associated with estimate shifts.
- Overall robustness of findings.

Value

A list structure containing:

- `results`: data.table in long format with all specification results.
- `abadie_inference`: list containing placebo-in-space inference outputs, including p-values and post/pre RMSPE ratios.
- `bootstrap_inference`: list containing bootstrap inference outputs.

The `results` data.table contains treatment effects (tau), unit information, and specification metadata for all units (treated, control placebo, bootstrap iterations) across all specifications. Inference results are separated to avoid duplication and provide cleaner access to statistical tests.

References

- Simonsohn, U., Simmons, J. P., and Nelson, L. D. (2020). *Specification curve analysis*. Nature Human Behaviour, 4(11), 1208-1214.
- Huntington-Klein, N. et al. (2021). The influence of hidden researcher decisions in applied microeconomics. *Economic Inquiry*, 59(3), 944-960.

Examples

```
set.seed(1)
toy <- expand.grid(unit = c("treated", "donor1", "donor2", "donor3"), year = 1:10,
                   stringsAsFactors = FALSE)
toy$y <- 50 + 0.8 * toy$year + ifelse(toy$unit == "treated" & toy$year >= 7, 3, 0) +
  rnorm(nrow(toy), 0, 0.5)
results <- spec_curve(
  dataset = toy,
  outcomes = "y",
  col_name_unit_name = "unit",
  name_treated_unit = "treated",
  covagg = list(
    baseline = list(
      label = "baseline",
    operations = list(list(var = "outcome_var", partition_periods = list(type = "by_period")))
    )
  ),
  treated_period = 7,
  min_period = 1,
  end_period = 10,
  col_name_period = "year",
  feature_weights = "uniform",
  outcome_models = "none",
  donor_sample = "all",
  constraints = list(list(name = "simplex")),
  constants = FALSE,
  cores = 1,
  inference_type = "placebo",
  inference_config = list(verbose = FALSE, placebo_cores = 1)
)
names(results)
```

---

spec_curve-methods *S3 Methods for Specification Curve Objects*

Description

This file contains S3 methods for objects of class "spec_curve" created by the spec_curve function. These methods provide a professional interface for examining and summarizing specification curve analysis results.

Print method for specification curve objects that provides a concise overview of the analysis results including number of specifications, treatment effect distribution, and available inference results.

Usage

```
## S3 method for class 'spec_curve'
print(x, ...)
```

Arguments

x	An object of class "spec_curve" from spec_curve
...	Additional arguments (currently unused)

Details

The print method displays:

- Number of specifications estimated
- Number of outcomes analyzed
- Treatment effect summary statistics
- Available inference methods and significance rates
- Specification dimension summary

Value

No return value, called for side effects (printing)

Examples

```
specs <- data.table::data.table(
  full_spec_id = paste0("s", 1:4),
  outcome_model = c("none", "ridge", "none", "ridge"),
  const = c("simplex", "lasso", "simplex", "lasso"),
  fw = c("uniform", "uniform", "optimize", "optimize"),
  feat = c("f1", "f1", "f2", "f2")
)
dt <- data.table::CJ(
  unit_name = c("treated", "donor1"),
  full_spec_id = as.character(specs[["full_spec_id"]]),
  post_period = c(FALSE, TRUE),
  sorted = FALSE
)
dt[, unit_type := ifelse(unit_name == "treated", "treated", "control")]
dt <- merge(dt, specs, by = "full_spec_id", all.x = TRUE)
dt[, outcome := "y"]
dt[, rmse := 0.2 + as.numeric(factor(full_spec_id)) * 0.01]
dt[, tau := ifelse(
  unit_name == "treated",
  as.numeric(factor(full_spec_id)) * 0.5 + ifelse(post_period, 0.2, 0),
  as.numeric(factor(full_spec_id)) * 0.1 + ifelse(post_period, 0.05, 0)
)]
spec_results <- list(results = dt)
class(spec_results) <- "spec_curve"
print(spec_results)
```

summary.scdata *Summary Method for Synthetic Control Prediction*

Description

The summary method for synthetic control prediction objects.

Usage

```
## S3 method for class 'scdata'
summary(object, ...)
```

Arguments

object	Class "scest" object, obtained by calling scdata.
...	Additional arguments

Value

No return value, called to summarize scdata results.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

See Also

scdata

Supported methods: print.scdata, summary.scdata.

summary.scest *Summary Method for Synthetic Control Prediction*

Description

The summary method for synthetic control prediction fitted objects.

Usage

```
## S3 method for class 'scest'
summary(object, ...)
```

Arguments

object	Class "scest" object, obtained by calling scest.
...	Additional arguments

Value

No return value, called to summarize scest results.

Author(s)

Matias Cattaneo, Princeton University. <cattaneo@princeton.edu>.

Yingjie Feng, Tsinghua University. <fengyj@sem.tsinghua.edu.cn>.

Filippo Palomba, Princeton University (maintainer). <fpalomba@princeton.edu>.

Rocio Titiunik, Princeton University. <titiunik@princeton.edu>.

See Also

scest

Supported methods: print.scest, summary.scest, coef.scest.

---

summary.scest_formula *Summary Method for Formula-Based Synthetic Control*

Description

Summary method for scest_formula objects

Usage

```
## S3 method for class 'scest_formula'
summary(object, ...)
```

Arguments

object	An scest_formula object
...	Additional arguments

summary.spec_curve *Summary Method for Specification Curve Results*

Description

Detailed summary method for specification curve objects that provides comprehensive information about the analysis results, including breakdown by specification dimensions and detailed inference results.

Usage

```
## S3 method for class 'spec_curve'
summary(object, ...)
```

Arguments

object	An object of class "spec_curve" from spec_curve
...	Additional arguments (currently unused)

Details

The summary method provides detailed information including:

- Treatment effect statistics by specification dimension
- Inference results breakdown
- Model performance statistics (RMSE, fit)
- Specification robustness indicators

Value

No return value, called for side effects (printing detailed summary)

Examples

```
specs <- data.table::data.table(
  full_spec_id = paste0("s", 1:4),
  outcome_model = c("none", "ridge", "none", "ridge"),
  const = c("simplex", "lasso", "simplex", "lasso"),
  fw = c("uniform", "uniform", "optimize", "optimize"),
  feat = c("f1", "f1", "f2", "f2")
)
dt <- data.table::CJ(
  unit_name = c("treated", "donor1"),
  full_spec_id = as.character(specs[["full_spec_id"]]),
  post_period = c(FALSE, TRUE),
  sorted = FALSE
)
dt[, unit_type := ifelse(unit_name == "treated", "treated", "control")]
```

```
dt <- merge(dt, specs, by = "full_spec_id", all.x = TRUE)
dt[, outcome := "y"]
dt[, rmse := 0.2 + as.numeric(factor(full_spec_id)) * 0.01]
dt[, tau := ifelse(
  unit_name == "treated",
  as.numeric(factor(full_spec_id)) * 0.5 + ifelse(post_period, 0.2, 0),
  as.numeric(factor(full_spec_id)) * 0.1 + ifelse(post_period, 0.05, 0)
)]
spec_results <- list(results = dt)
class(spec_results) <- "spec_curve"
summary(spec_results)
```

`synth` *Synthetic Control with Formula Interface*

Description

Estimate synthetic control using standard R formula syntax. This function provides a convenient interface that automatically handles data preparation and estimation based on formula specification.

Usage

```
synth(
  formula,
  data,
  time.var,
  id.var,
  treated.period,
  pre.period,
  post.period,
  constraint = "simplex",
  feature.weights = "uniform",
  covariate.matching = "average",
  ...
)
```

Arguments

`formula`	A formula object specifying the model. Format: `outcome ~ covariates | treated_unit` where covariates specify matching variables and treated_unit identifies the treated unit. Example: `gdp ~ population + investment | California`
`data`	A data.frame or data.table containing the panel data
`time.var`	Character. Name of time variable column
`id.var`	Character. Name of unit identifier column
`treated.period`	Numeric. First period when treatment begins

`pre.period` Numeric vector. Pre-treatment periods to use for matching

`post.period` Numeric vector. Post-treatment periods for outcome prediction

`constraint` Character or list. Weight constraint specification:

- `"simplex"` - Standard synthetic control (default)
- `"lasso"` - L1 penalized weights
- `"ridge"` - L2 penalized weights
- `"ols"` - Unconstrained weights
- List with detailed constraint parameters

`feature.weights` Character. Feature weighting method ("uniform" or "optimize")

`covariate.matching` Character. How to aggregate covariates:

- `"average"` - Use pre-period means (default)
- `"each"` - Include each pre-period separately
- `"last"` - Use only the last pre-treatment period

`...` Additional arguments passed to `scest()`

Details

The formula interface translates standard R formula syntax into the appropriate scdata/scest calls:

Formula Components:

- Left side: Outcome variable
- Right side (before |): Covariates for matching
- Right side (after |): Treated unit identifier

Special Formula Syntax:

- `I(log(var))` - Transform variables
- `var1 + var2` - Include multiple covariates
- `.` - Include all available covariates
- `-var` - Exclude specific variables from matching

Value

An object of class "scest_formula" inheriting from "scest" with additional formula-related metadata

Examples

```
panel_data <- data.frame(
  state = rep(c("California", "Texas", "Florida"), each = 6),
  year = rep(1:6, 3),
  gdp = c(10, 11, 12, 14, 15, 16,
          9, 10, 11, 12, 13, 14,
          11, 12, 13, 13, 14, 15),
  population = c(1, 1.1, 1.2, 1.3, 1.35, 1.4,
```

```
                0.9, 0.95, 1.0, 1.05, 1.1, 1.15,
                1.2, 1.25, 1.3, 1.33, 1.36, 1.4),
  investment = c(2, 2.1, 2.2, 2.4, 2.5, 2.6,
                1.8, 1.9, 2.0, 2.1, 2.2, 2.3,
                2.0, 2.1, 2.2, 2.25, 2.3, 2.35)
)
result <- synth(
  gdp ~ population + investment | California,
  data = panel_data,
  time.var = "year",
  id.var = "state",
  treated.period = 5,
  pre.period = 1:4,
  post.period = 5:6,
  constraint = "simplex"
)
class(result)
```

`tidy-methods` *Broom-Compatible Tidy Methods for Synthetic Control Objects*

Description

These methods provide broom-compatible interfaces for extracting information from synthetic control estimation results in tidy data.table format. They follow the broom convention of tidy(), glance(), and augment() methods while returning data.table objects for efficiency.

Details

Tidy Methods for SCMs Objects

`tidy.scest` *Extract Synthetic Control Weights in Tidy Format*

Description

Extract donor unit weights and related information from synthetic control estimation results in a tidy data.table format compatible with broom workflows.

Usage

```
## S3 method for class 'scest'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

x	An object of class "scest" from scest()
conf.int	Logical. Whether to include confidence intervals (if available)
conf.level	Numeric. Confidence level for intervals (default 0.95)
...	Additional arguments (not currently used)

Value

A data.table with columns:

- term - Character. Donor unit names
- estimate - Numeric. Synthetic control weights
- donor_rank - Integer. Rank of donor by weight (1 = highest)
- weight_percentage - Numeric. Weight as percentage of total
- is_active - Logical. Whether weight is meaningfully non-zero
- constraint_type - Character. Type of constraint used
- std.error - Numeric. Standard error (if conf.int=TRUE and available)
- conf.low - Numeric. Lower confidence bound (if conf.int=TRUE)
- conf.high - Numeric. Upper confidence bound (if conf.int=TRUE)

Examples

```
toy <- data.frame(
  unit = rep(c("treated", "donor1", "donor2"), each = 6),
  year = rep(1:6, 3),
  y = c(10, 11, 12, 14, 15, 16,
        9, 10, 11, 12, 13, 14,
        11, 12, 13, 13, 14, 15)
)
sc_obj <- scdata(
  df = toy,
  id.var = "unit",
  time.var = "year",
  outcome.var = "y",
  period.pre = 1:4,
  period.post = 5:6,
  unit.tr = "treated",
  unit.co = c("donor1", "donor2"),
  covagg = list(list(var = "outcome_var", partition_periods = list(type = "by_period")))
)
fit <- scest(sc_obj, w.constr = list(name = "simplex"))
generics::tidy(fit)
```

`tidy.spec_curve` *Extract Specification Curve Results in Tidy Format*

Description

Extract treatment effect estimates and metadata from specification curve analysis results in tidy data.table format.

Usage

```
## S3 method for class 'spec_curve'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

`x`	An object of class "spec_curve" from `run_spec_curve_analysis()`
`conf.int`	Logical. Whether to include confidence intervals from inference
`conf.level`	Numeric. Confidence level for intervals (default 0.95)
`...`	Additional arguments (not currently used)

Value

A data.table with columns:

- `spec_id` - Character. Specification identifier
- `estimate` - Numeric. Treatment effect estimate
- `outcome` - Character. Outcome variable name
- `unit_name` - Character. Treated unit name
- `rmse` - Numeric. Root mean squared error
- `outcome_model` - Character. Outcome modeling approach
- `constraint_type` - Character. Weight constraint type
- `feature_weights` - Character. Feature weighting method
- `p.value` - Numeric. P-value from inference (if available)
- `statistic` - Numeric. Test statistic (if available)
- `method` - Character. Inference method used
- `significant` - Logical. Whether $p < 0.05$

Examples

```
spec_results <- list(
  results = data.table::data.table(
    full_spec_id = c("s1", "s2"),
    tau = c(1.2, 0.8),
    outcome = "y",
    unit_name = "treated",
    unit_type = "treated",
    post_period = TRUE,
    rmse = c(0.2, 0.3)
  )
)
class(spec_results) <- "spec_curve"
generics::tidy(spec_results)
```

Index

augment.scest, 3

b.est.clarabel.ols, 4
b.est.clarabel.simplex, 5
b.est.clarabel.unified, 5
b.est.cvxr.fallback, 6
bootstrap_null_inference, 7

calculate_abadie_significance, 8
calculate_bootstrap_pvalues, 9
calculate_spec_curve_pvalues_filtered, 9
coef.scest, *11*, 11, *31*, *51*
create_analysis_config, 12
create_bootstrap_dataset, 13
create_scm_dataset, 13
create_xgboost_config, 15

enforce_null_hypothesis, 16
estimate_sc, 17, *21*

formula-interface, 18

glance.scest, 19

inference_placebo, 20, *21*
inference_sc, 21, *39*, *41*

perform_bootstrap_inference, 22
plot.spec_curve, 23
plot_spec_curve, *23*, 24
print.scdata, 29, *30*, *50*
print.scest, *11*, 30, *31*, *51*
print.scest_formula, 31
print.spec_curve *(*spec_curve-methods*)*, 48

run_spec_curve_analysis, 31
run_xgboost_shap_analysis, 33

scdata, *14*, *17*, *30*, 34, *39*, *41*, *50*
scest, *11*, *30*, *31*, 37, *39–41*, *51*
scplot, 39
solve_qp_clarabel, 41
spec_curve, *23*, 45, *48*, *49*, *52*
spec_curve-methods, 48
SpecResultsManager, 42
summary.scdata, *30*, *50*, 50
summary.scest, *11*, *31*, 50, *51*
summary.scest_formula, 51
summary.spec_curve, 52
synth, 53

tidy-methods, 55
tidy.scest, 55
tidy.spec_curve, 57