Package 'optic'

August 8, 2023

Type Package

Title Simulation Tool for Causal Inference Using Longitudinal Data

Version 1.0.1

Description Implements a simulation study to assess the strengths and weaknesses of causal inference methods for estimating policy effects using panel data. See Griffin et al. (2021) <doi:10.1007/s10742-022-00284-w> and Griffin et al. (2022) <doi:10.1186/s12874-021-01471-y> for a description of our methods.

License GPL-3

URL https://randcorporation.github.io/optic/,
https://github.com/randcorporation/optic

BugReports https://github.com/randcorporation/optic/issues

Depends R (>= 4.1.0)

Imports did, dplyr, future.apply, lmtest, magrittr, MASS, methods, purrr, R6, rlang, sandwich, stats, tidyr, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Beth Ann Griffin [aut, cph] (<https://orcid.org/0000-0002-2391-4601>),
Pedro Nascimento de Lima [cre, aut]
(<https://orcid.org/0000-0001-9057-198X>),
Max Griswold [aut] (<https://orcid.org/0000-0002-6387-128X>),
Adam Scherling [aut],
Joseph Pane [aut],
Geoffrey Grimm [aut]

Maintainer Pedro Nascimento de Lima <plima@rand.org>

Repository CRAN

Date/Publication 2023-08-08 13:40:02 UTC

R topics documented:

`calculate_exposure` *Calculates the exposure rate applied to each year provided month of policy implementation and number of years to full implementation*

Description

Calculates the exposure rate applied to each year provided month of policy implementation and number of years to full implementation

Usage

```
calculate_exposure(month, n_years, monthly_effect = (1/n_years)/12)
```

Arguments

`month`	month of year (as integer) that policy takes effect
`n_years`	number of months until full implementation in effect
`monthly_effect`	increment of exposure to apply each month; default is ((1/n_years) / 12) (constant over the period)

Value

A vector of percentages, indicating change in exposure by year (relative to start month)

Examples

```
# Calculate uniform increase in policy effect which ramps up across 10 years

# Assume policy starts in July of the first year, then continues for 10 years
starting_month <- 7
implementation_years <- 10
```

```
# Assume some policy effect (which is the target effect for simulations)
policy_effect <- 2

exposure_by_year <- calculate_exposure(starting_month, implementation_years)

# Based on exposure by year, calculate policy effect by year:
plot(policy_effect*exposure_by_year)
```

---

dispatch_simulations *Execute simulations defined in a optic_simulation object*

Description

Execute simulations defined in a optic_simulation object

Usage

```
dispatch_simulations(object, seed = NULL, use_future = FALSE, verbose = 0, ...)
```

Arguments

object	Simulation scenarios object created using optic_simulation
seed	Specified as either NULL or a numeric. Sets a seed, which is becomes an index in results, for each independent set of simulations in optic_simulation.
use_future	Runs simulation scenarios in parallel. Default FALSE, set to TRUE if you have already setup a future plan (e.g., multiprocess, cluster, etc) and would like for the iterations to be run in parallel.
verbose	Default TRUE. IF TRUE, provides details on what's currently running.
...	additional parameters to be passed to future_apply. User can pass future.globals and future.packages if your code relies on additional packages

Value

A list of dataframes, where each list entry contains results for a set of simulation parameters, with dataframes containing estimated treatment effects and summary statistics by model and draw.

Examples

```
# Set up a basic model and simulation scenario:
data(overdoses)

eff <- 0.1*mean(overdoses$crude.rate, na.rm = TRUE)
form <- formula(crude.rate ~ state + year + population + treatment_level)
mod <- optic_model(name = 'lin',
                   type = 'reg',
                   call = 'lm',
```

```
                   formula = form,
                   se_adjust = 'none')

sim <- optic_simulation(x = overdoses,
                        models = list(mod),
                        method = 'no_confounding',
                        unit_var = 'state',
                        treat_var = 'state',
                        time_var = 'year',
                        effect_magnitude = list(eff),
                        n_units = 2,
                        effect_direction = 'pos',
                        iters = 2,
                        policy_speed = 'instant',
                        n_implementation_periods = 1)

# Finally, dispatch the simulation:
dispatch_simulations(sim)
```

`exposure_list` *Applies a time-varying treatment effect*

Description

Simulates a time-varying treatment effect that starts at zero in time period zero, then linearly increases to a 'full treatment' effect, based on analyst-provided choices concerning time until full treatment effect and 'speed'

Usage

```
exposure_list(
  sampled_time_period,
  mo,
  available_periods,
  policy_speed,
  n_implementation_periods
)
```

Arguments

`sampled_time_period`
: Year that treatment is first enacted

`mo`
: Month that treatment is first enacted

`available_periods`
: Maximum number of time periods in the data (e.g. if policy is between 1950-2000, then available_periods == 50)

`policy_speed`
: A string which is either "instant" for the policy going into immediate effect or "slow" for the policy effect phasing in linearly across n_implement_periods

n_implementation_periods
: Number of periods until full treatment effect is applied. Only used if policy_speed is 'slow'.

Value

A list, containing a vector of policy years of implementation, an integer of the starting policy implementation month, and the effect of treatment within a given implementation year (as a fraction of the total policy effect)

Examples

```
# Set up a policy that starts in first-year of data, in July and takes
# 2 years for full implementation:
exposure_list(1, 7, 3, policy_speed = 'slow', n_implementation_periods = 2)

# Same scenario but effect happens instantaneously:
exposure_list(1, 7, 3, policy_speed = 'instant')
```

---

model_terms — *Parse a formula object into its left-hand-side and right-hand-side components*

Description

Parse a formula object into its left-hand-side and right-hand-side components

Usage

```
model_terms(x)
```

Arguments

x
: Formula to parse

Value

list with named elements "lhs" and "rhs", containing variables on each respective side of the equation

Examples

```
# Set up a hypothetical function, then decompose into left-hand and
# right-hand sides
form <- formula(outcome ~ treatment + confounder + unit + time)
model_terms(form)
```

optic_model *Optic Model*

Description

Generates model object to apply to each simulated dataset

Usage

```
optic_model(name, type, call, formula, se_adjust, ...)
```

Arguments

name	Name of the model object, used to identify the model when reviewing simulation results
type	Estimator used to identify the treatment effect using simulated data. Specified as a string, which can either be 'reg' (regression), 'autoreg' (autoregression, which adds a lag for the outcome variable to a regression model), 'drdid' (doubly-robust difference-in-difference estimator), or 'multisynth' (augmented synthetic control)
call	String which specifies the R function to call for applying the estimator. Package currently supports either 'lm' (linear model), 'feols' (fixed-effect OLS), 'multisynth' (pooled synthetic controls), or 'glm.nb' (negative-binomial generalized nearlized linear model)
formula	Model specification, using R formula formatting. Must include a variable labeled 'treatment' for the 'nonconf' & 'selbias' simulation method or variables labeled 'treatment1' & 'treatment2' for the simulation method 'concurrent'
se_adjust	Adjustments applied to standard errors following model estimation. Specified as a string, OPTIC currently support 'none' for no adjustment or 'cluster' for clustered standard errors. Clustered standard errors will use the 'unit_var' specified in optic_simulation for determining unit used for clustering standard errors.
...	Additional arguments that are passed to the model call. Please refer to documentation for each model call for additional details. If the model call expects a name, you may need to pass your parameter using param = as.name("variable_name") as opposed to param = variable_name.

Value

optic_model An optic_model object to be used as an input within optic_simulations. Details model calls and parameters.

Examples

```
# Set up a simple linear model
form <- formula(crude.rate ~ state + year + population + treatment_level)
```

```
mod <- optic_model(name = 'lin',
                   type = 'reg',
                   call = 'lm',
                   formula = form,
                   se_adjust = 'none')

# Deploy an auto-regressive model.
# type = "autoreg" will make AR term
# automatically when the model is deployed; also note
# in formula the use of "treatment_change" as the treatment variable
# rather than "treatment_level" like in the previous example:

form_ar <- formula(crude.rate ~ state + year + population + treatment_change)
mod_ar <- optic_model(name = "auto_regressive_linear",
                      type = "autoreg",
                      call = "lm",
                      formula = form_ar,
                      se_adjust = "none")

# One could also use a different call, assuming the right packages
# are installed and the model uses a familiar formula framework.
# Example with random intercept for states, using lme4 package.

form_me <- formula(crude.rate ~
                   population + year + treatment_level + (1|state))

mod_me <- optic_model(name = "mixed_effect",
                      type = "reg",
                      call = "lmer",
                      formula = form_me,
                      se_adjust = "none")
```

`optic_simulation` *Create a configuration object used to run simulations*

Description

Performs validation on inputs and produces a configuration object that contains all required parameters to dispatch simulation runs for the empirical data provided.

Usage

```
optic_simulation(
  x,
  models,
  iters,
  unit_var,
  time_var,
  conf_var,
```

```
  effect_magnitude,
  n_units,
  effect_direction,
  policy_speed,
  prior_control = "level",
  bias_size = NULL,
  bias_type = NULL,
  treat_var = NULL,
  n_implementation_periods,
  rhos = NULL,
  years_apart = NULL,
  ordered = NULL,
  method,
  method_sample,
  method_model,
  method_results,
  method_pre_model,
  method_post_model,
  globals = NULL,
  verbose = TRUE
)
```

Arguments

`x` Empirical data used to simulate synthetic datasets with specified treatment effect.

`models` List of ‘optic_model‘ objects that should be run for each iteration and simulation scenario. The elements must be created using the ‘optic_model‘ function.

`iters` A numeric, specifying number of iterations for each simulation scenario.

`unit_var` A string variable, used to determine clusters for clustered standard errors.

`time_var` A string variable, specifying time units (e.g. "year", "time to treat", etc). Must be specified in terms of years (fractional years are accepted).

`conf_var` An unobserved confounding variable. Only used for the ’confound-method’.

`effect_magnitude` A vector of numerics, specifying ’true’ effect sizes for treatment scenarios. See vignette for more details. Synthetic datasets will be generated for each entry in the vector.

`n_units` A numeric, determining number of units to simulate treatment effects. Synthetic datasets will be generated for each entry in the vector.

`effect_direction` A vector containing either ’neg’, ’null’, or ’pos’. Determines the direction of the simulated effect. Synthetic datasets will be generated for each entry in the vector.

`policy_speed` A vector of strings, containing either ’instant’ or ’slow’ entries, determining how quickly treated units obtain the simulated effect. Synthetic datasets will be generated for each entry in the vector. Can either be ’instant" (so treatment effect applies fully in the first treated time period) or 'slow' (treatment effect ramps up linearly to the desired effect size, based on ‘n_implementation_periods‘.

`prior_control` Only used for confounding method. Adds an additional set of variables which control for the outcome in previous periods (either a moving average of previous time periods or an autoregressive term)

`bias_size` A string, either "small" "medium" or "large". Specifies relative size of bias for 'confounding' method.

`bias_type` A string, either linear" or "nonlinear". Specifies type of bias for 'confounding' method

`treat_var` A string variable, referring to the unit-of-analysis for treatment (which may not be the same as the unit var argument, e.g. treated classrooms within clustered schools)

`n_implementation_periods` A vector of numerics, determining number of periods after implementation until treated units reach the desired simulated treatment effect. Synthetic datasets will be generated for each entry in the vector.

`rhos` A vector of values between 0-1, indicating the correlation between the primary policy and a concurrent policy. Only applies when 'method' == 'concurrent'. Synthetic datasets will be generated for each entry in the vector.

`years_apart` A numeric, for number of years between the primary policy being implemented and the concurrent policy. Only applies when 'method' == 'concurrent'.

`ordered` A boolean, determines if the primary policy always occurs before the concurrent policy (‘TRUE‘) or if the policies are randomly ordered (‘FALSE‘).

`method` A string, determing the simulation method. Can be either 'no_confounding', 'confounding' or 'concurrent'

`method_sample` Underlying function for the sampling method to determine treatment status. Provided here for convenience so that the user does not need to modify the actual underlying function's script.

`method_model` Another convenience function, which can be modified to control the model call.

`method_results` Another convenience function, which can be modified to control the simulation results that are returned.

`method_pre_model` Similar to method_sample argument, this variable is provided as a convenience for the user. This function transforms the treatment effect, after it's simulated within the synthetic data.

`method_post_model` Another convenience function, which can be modified to control transformations to the simulated effect, after modeling.

`globals` Additional globals to pass to the simulate function, such as parallelization packages or additional R packages used by method calls (e.g. modeling packages, like "FEOLS").

`verbose` Boolean, default True. If TRUE, provides summary details on simulation runs across iterations

Details

The resulting configuration object is used to pass simulation scenarios to the 'simulate' function. Provided as a convenience function to the user so they can investigate simulation arguments prior to running models.

Value

An OpticSim object, which contains simulation and model parameters for simulation runs, which is used as an input for dispatch_simulations.

Examples

```
# Load data for simulation and set up a hypothetical policy effect:

data(overdoses)
eff <- 0.1*mean(overdoses$crude.rate, na.rm = TRUE)

# Set up a simple linear model
form <- formula(crude.rate ~ state + year + population + treatment_level)
mod <- optic_model(name = 'lin',
                   type = 'reg',
                   call = 'lm',
                   formula = form,
                   se_adjust = 'none')

# Create simulation object, with desired parameters for simulations:
sim <- optic_simulation(x = overdoses,
                        models = list(mod),
                        method = 'no_confounding',
                        unit_var = 'state',
                        treat_var = 'state',
                        time_var = 'year',
                        effect_magnitude = list(eff),
                        n_units = 10,
                        effect_direction = 'pos',
                        iters = 10,
                        policy_speed = 'instant',
                        n_implementation_periods = 1)
```

overdoses *OPTIC Overdoses example data.*

Description

An example dataset for performing simulations using the OPTIC library, consisting of state-year overdose data from the US Bureau of Labor Statistics, the Centers from Disease Control and Prevention, and IQVIA Xponent.

Usage

```
overdoses
```

Format

A data frame with 969 rows and 7 variables:

state US state

year Year

population Population estimate (Centers for Disease Control and Prevention, National Center for Health Statistics)

unemploymentrate Average annual unemployment rate (US Bureau of Labor Statistics)

opioid_rx Estimated number of annual opioid prescriptions dispensed per 100 residents (Centers for Disease Control and Prevention, IQVIA)

deaths Annual number of drug-induced deaths (all drug overdose) (Centers for Disease Control and Prevention, National Center for Health Statistics)

crude.rate Crude rate of drug-induced deaths (all drug overdose) per 100,000 residents (Centers for Disease Control and Prevention, National Center for Health Statistics)

Source

US Bureau of Labor Statistics. Local Area Unemployment Statistics April 2019 release. Accessed at https://www.bls.gov/lau/.

Centers for Disease Control and Prevention, National Center for Health Statistics. Multiple Cause of Death 1999-2019 on CDC WONDER Online Database, released in 2020. Data are from the Multiple Cause of Death Files, 1999-2019, as compiled from data provided by the 57 vital statistics jurisdictions through the Vital Statistics Cooperative Program. Accessed at `http://wonder.cdc.gov/mcd-icd10.html`.

Centers for Disease Control and Prevention, IQVIA Xponent 2006–2019. U.S. Opioid Dispensing Rate Maps. Accessed at https://www.cdc.gov/drugoverdose/rxrate-maps/index.html.

Index