Contents

OPTIC Introduction

This vignette covers use of the OPTIC package, which contains tools to simulate the performance of commonly-used statistical models. Briefly, OPTIC uses Monte Carlo simulations to estimate the performance of models typically used for state-level policy evaluation (for example, differences-in-differences estimators). Package users provide the simulation procedure with a policy evaluation scenario, a hypothetical treatment effect, an estimator and model specification, and simulation parameters. OPTIC then constructs simulations for all chosen simulation parameters and estimators, returning summary statistics on each simulation's performance.

Currently, OPTIC has been developed for policy evaluation scenarios with the following assumptions:

1. No confounding : There are no confounding variables which determine a observation's selection into treatment. Additionally, treatment effects are assumed to be constant rather than time-varying.
2. Confounding due to co-occurring policies: There is one or more policies which co-occur with the the policy of-interest. Selection into treatment is still random.

Additional details on the simulation procedures within OPTIC are available in Griffin et al., 2021[1] and Griffin et al., 2022[2]. Forthcoming updates to OPTIC will provides simulations which test the effect of selection bias on model performance.

This README covers the use of OPTIC in R and provides examples of the package's features. Section one covers package installation, section two provides a general overview of the package's usage, and section three provides two examples of the package under different policy scenarios.

Installation

Clone the repository to your machine. Once you have the project on your local machine, the first thing you will want to do is build and install the package. Open the project in RStudio - it's easiest to open the optic.Rproj file to do so. Make sure you have `devtools` installed.

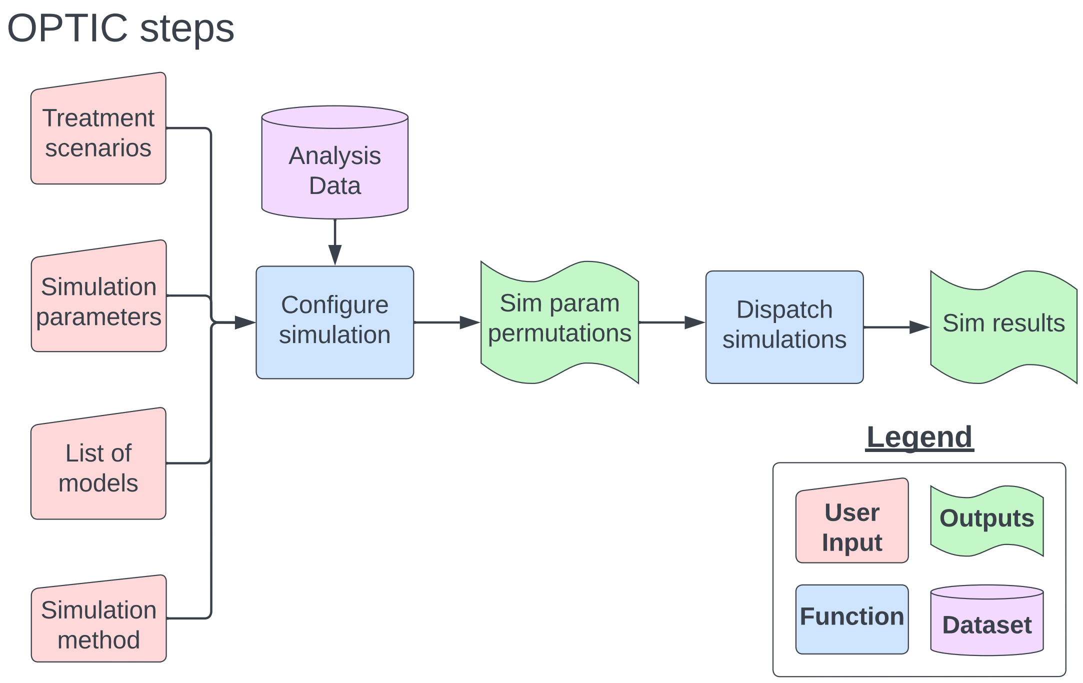
```r
require(devtools)

# Build package documentation and source code
devtools::document()
devtools::build()

# Install optic package and dependencies
devtools::install()
```

Overview of the OPTIC Package

OPTIC contains two core functions for performing simulations, which require four user-provided inputs. Figure 1 below displays a flowchart of OPTIC's required steps to produce simulation results.



The following sections describe OPTIC inputs and steps in detail.

User inputs

Data

Analysts will need to shape their data to work properly with OPTIC. The package contains an example dataset which demonstrates the format required by the function "configure_simulation".

state	year	pop	unemployment_rate	opioid_death_rate
Alabama	2015	4858979	6.1	15.1
Alabama	2016	4863300	5.8	15.5
Alabama	2017	4874747	4.4	17.1
Alaska	1999	624779	6.5	7.4
Alaska	2000	626932	6.4	7.7

The data should generally be structured to work as an input to a two-way fixed effect model, using a "long-format", with variables for "group", "time", covariates, and an outcome variable. The example data included in the package is derived from data provided by the US Bureau of Labor Statistics and the Centers for Disease Control and Prevention.

Policy evaluation scenarios

Policy evaluation scenarios are single string inputs into the "configure_simulation" function, representing either the no confounding scenario ("noconf") or the confounding due to co-occurring policies scenario ("concurrent"). For the "concurrent" scenario, there are additional parameters required within "configure_simulation", which are discussed in the "parameter" section below. Additional details on policy evaluation scenarios are provided in Griffin et al., 2021[@http://zotero.org/users/3390799/items/ZNCVTPJF] and Griffin et al., 2022[@http://zotero.org/users/3390799/items/V3Q6ARUA].

Treatment scenarios

This input represents the "true treatment effects" that OPTIC will simulate across model iterations. OPTIC is currently designed to work on static treatment effects (rather than dynamic treatment effects, such as time-varying treatment effects). Users should structure their treatment scenarios in a list, corresponding to a change in the outcome variable. The list will contain two effects within a vector, if the user is simulating models for the "concurrent" policy evaluation scenario. Using the example_data:

```r
# Calculate 5% and 10% changes in mean opioid_death_rate,
# across states and years.
five_percent_effect <- 0.05 * mean(df$opioid_death_rate)
ten_percent_effect <- 0.1 * mean(df$opioid_death_rate)

# Calculate a confounding policy effect
confound_effect <- -0.02 * mean(df$opioid_death_rate)

# Scenario object for 'no confounding' evaluation scenario:
scenarios_no_confounding <- list(five_percent_effect, ten_percent_effect)

# Scenario object for 'co-occuring policy' evaluation
# scenario:
scenarios_co_occur <- list(c(five_percent_effect, confound_effect),
    c(ten_percent_effect, confound_effect))
```

List of models

For each treatment scenario, OPTIC will simulate a treatment effect and then attempt to estimate this effect based on user-provided models. The `configure_simulation` function takes a list of lists, with inner lists requiring several named arguments. Model lists should contain:

- `name`: A name for the model to identify the model type in results.
- `type`: Users can set this as either "autoreg" or "reg". The "autoreg" option adds an outcome lag to the model formula.
- `model_call`: The call for the model in R (e.g. "lm", "glm", etc).
- `model_formula`: The model specification, in an R formula. Needs to include a variable labeled "treament" for the policy scenario "noconf" or variables labeled "treatment1", "treatment2", . . ., "treatment{n}" for the policy scenario "concurrent". If using
- `model_args`: Any additional arguments passed to the model_call (e.g. "weights", "family", "control" etc.).
- `se_adjust`: Any adjustments to standard errors after estimation. OPTIC recognizes either "none" for no adjustment or "cluster" for clustered standard errors (OPTIC will use the `configure_simulation` parameter "unit_var" to determine clusters for clustered standard errors).

Below provides an example of a model list using the `example_data`:

```r
sim_models <- list(
  list(
    name = "fixed_effect_linear",
    type = "reg",
    model_call = "lm",
    model_formula = opioid_death_rate ~ year + state + treatment,
    model_args = list(weights = 'population'),
    se_adjust = "none"),

  list(
    name = "Auto-regressive linear",
    type = "autoreg",
    model_call = "lm",
    model_formula = opioid_death_rate ~ unemployment_rate + year + treatment_change,
    model_args=list(offset = 'population'),
    se_adjust = c("cluster")
  )
)
```

OPTIC functions

`configure_simulation`

This function takes the policy evaluation scenario, treatment scenarios, model list, and function parameters to generate synthetic datasets with simulated treatment effects. `configure_simulation` has the following additional arguments:

- `x`: The prepped analysis dataset. See section above for more details.
- `models`: List of models. See section above for more details.
- `meth`: Policy evaluation scenario. Can either be "noconf" or "concurrent".
- `unit_var`: Variable for groups within the dataset. Used to determine clusters for clustered standard errors
- `treat_var`: The variable to use to simulate treatment across units.
- `effect_magnitude`: A vector of treatment scenarios. See section above for more details. Synthetic datasets will be generated for each entry in the vector.
- `n_units`: A vector with the number of units to simulate treatment. Synthetic datasets will be generated for each entry in the vector.
- `effect_direction`: A vector containing either 'neg', 'null', or 'pos'. Synthetic datasets will be generated for each entry in the vector. Determines the direction of the simulated effect.
- `policy_speed`: A vector 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`.
- `n_implementation_periods` A vector with 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.

Three arguments to `configure_simulation` only apply within the "concurrent" policy scenario:

- `rhos`: A vector of 0-1 values indicating the correlation between the primary policy and a confounding policy. Synthetic datasets will be generated for each entry in the vector.

- `years_apart`: Number of years between the primary policy being implemented and the confounding policy.
- `ordered`: Determines if the primary policy always occurs before the confounding policy (`TRUE`) or if the policies are randomly ordered (`FALSE`).

The function returns a configuration object that's used as an input to `dispatch_simulation`. This object contains a dataset listing all possible simulations that will be run for each model (. An example call of `configure_simulation` is displayed below:

```
sim_config <- configure_simulation(

  x      = df,
  models = sim_models,
  iters  = 10,
  meth   = "concurrent",

  #Max note: We should unpack these arguments from being embedded within a list and change some of the

  params = list(
    unit_var                  = "state",
    treat_var                 = "state",
    time_var                  = "year",
    effect_magnitude          = scenarios_co_occur,
    n_units                   = c(30),
    effect_direction          = c("neg"),
    policy_speed              = c("instant", "slow"),
    n_implementation_periods = c(3),
    rho                       = c(0.5),
    years_apart               = 2,
    ordered                   = TRUE
  )
)
```

`dispatch_simulation`

Starting point for this section, based on Adam's previous work:

To run a simulation, use the `dispatch_simulation` method, which will iterate over the rows in the `"simulation_params"` element and run the iterations for each simulation either in a loop or in parallel depending on how you choose to dispatch the job.

Some short examples

Putting all the pieces together from section X

Simulations for a single policy (without confounding)

Simulations with co-occurring policies (without confounding)

```
# We will define two scenarios for different effect magnitudes for
# a hypothetical policies using 5, 10, and 15 percent changes in the outcome. In general,
# the package user should specify scenarios using numbers that make sense for their
# outcome variable (for example, if the outcome is mortality rates, then a 5%
# average treatment effect would be equivalent to 0.05*average mortality
# rate across units.)

# Below scenarios correspond to percent changes in average opioid mortality rates
# across US states between 1999-2017.
linear5 <- 0.22
linear10 <- 0.45
linear15 <- 0.68

scenario1 <- c(linear10, linear10)
scenario2 <- c(linear5, linear15)

my_models <- list(
  list(
    name="fixedeff_linear",
    type="reg",
    model_call="lm",
    model_formula = crude.rate ~ unemploymentrate + as.factor(year) + as.factor(state) + treatment1_leve
    model_args=list(weights=as.name('population')),
    se_adjust=c("none")
  ),
  list(
    name="autoreg_linear",
    type="autoreg",
    model_call="lm",
    model_formula = crude.rate ~ unemploymentrate + as.factor(year) + treatment1_change + treatment2_cha
    model_args=list(weights=as.name('population')),
    se_adjust=c("none")
  )
)

sim_config <- configure_simulation(
  x=df,
  models=my_models,
  iters=10,
  meth = "concurrent",

  params=list(
    unit_var="state",
    time_var="year",

    #Below are the linear effect chagnes occuring from a     hypothetical policy
    effect_magnitude=list(scenario1, scenario2),
    n_units=c(30),
    effect_direction=c("neg"),
    policy_speed=c("instant", "slow"),
    n_implementation_periods=c(3),
    rho=c(0.5),
    years_apart=2,
    ordered=TRUE
  )
)

lm_results <- dispatch_simulations(
    sim_config,
    use_future=TRUE,
    seed=9782,
    verbose=2,
    future.globals=c("cluster_adjust_se"),
    future.packages=c("dplyr", "optic")
)

# Include plots
```

Bibliography

1 Griffin BA, Schuler MS, Stuart EA, *et al.* Moving beyond the classic difference-in-differences model: a simulation study comparing statistical methods for estimating effectiveness of state-level policies. *BMC Medical Research Methodology* 2021; **21**: 279.

2 Griffin BA, Schuler MS, Pane J, *et al.* Methodological considerations for estimating policy effects in the context of co-occurring policies. *Health Serv Outcomes Res Method* 2022; published online July 9. DOI:10.1007/s10742-022-00284-w.