simcausal R Package: Conducting Transparent and Reproducible Simulation Studies of Causal Effect Estimation with Complex Longitudinal Data

Oleg Sofrygin
Division of Research,
Kaiser Permanente Northern California
University of California, Berkeley

Mark J. van der Laan
University of California, Berkeley

Romain Neugebauer
Division of Research,
Kaiser Permanente Northern California

Abstract

The **simcausal** R package is a tool for specification and simulation of complex longitudinal data structures that are based on structural equation models. The package aims to provide a flexible tool for simplifying the conduct of transparent and reproducible simulation studies, with a particular emphasis on the types of data and interventions frequently encountered in real-world causal inference problems, such as, observational data with time-dependent confounding, selection bias, and random monitoring processes. The package interface allows for concise expression of complex functional dependencies between a large number of nodes, where each node may represent a time- varying random variable. The package allows for specification and simulation of counterfactual data under various user- specified interventions (e.g., static, dynamic, deterministic, or stochastic). In particular, the interventions may represent exposures to treatment regimens, the occurrence or non-occurrence of right-censoring events, or of clinical monitoring events. Finally, the package enables the computation of a selected set of user-specified features of the distribution of the counterfactual data that represent common causal quantities of interest, such as, treatment-specific means, the average treatment effects and coefficients from working marginal structural models. The applicability of **simcausal** is demonstrated by replicating the results of two published simulation studies.

Keywords: causal inference, simulation, marginal structural model, structural equation model, directed acyclic graph, causal model, R.

Contents

1. Introduction

This vignette describes the **simcausal** package (Sofrygin *et al.* 2015), a comprehensive set of tools for specification and simulation of complex longitudinal data structures to study causal inference methodologies. The package is developed using the R system for statistical computing (R Core Team 2015) and is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=simcausal`. The main motivation behind the package is to provide a flexible tool to *facilitate* the conduct of *transparent* and *reproducible* simulation studies, with a particular emphasis on the types of data and interventions frequently encountered in real-world causal inference problems. For example, the package simplifies the simulation of observational data based on random clinical monitoring to evaluate the effect of time-varying interventions in the presence of time-dependent confounding and sources of selection bias (e.g., informative right censoring). The package is built to provide a novel user-interface that allows concise and intuitive expression of complex functional dependencies for a large number of nodes that may represent time-varying random variables (e.g., repeated measurements over time of the same subject-matter attribute, such as, blood pressure).

Each data generating distribution is specified via a structural equation model (SEM) (Pearl 1995, 2009, 2010). The package allows for specification and simulation of counterfactual data (referred to as "full data") under various user-specified interventions (e.g., static, dynamic, deterministic, or stochastic), which are referred to as "actions". These actions may represent exposure to treatment regimens, the occurrence or non-occurrence of right- censoring events, or of clinical monitoring events (e.g., laboratory measurements based on which treatment decisions may be made). Finally, the package enables the computation of a selected set of user-specified features of the distribution of the full data that represent common causal quantities of interest, referred to as causal target parameters, such as, treatment-specific means, the average treatment effects (ATE) (on the multiplicative or additive scale) and coefficients from working marginal structural model (MSM) (Neugebauer and van der Laan 2007). We demonstrate an application of the **simcausal** package by replicating the results of two published simulation studies from the causal inference literature (Neugebauer *et al.* 2014, 2015; Lefebvre *et al.* 2008).

We note that the **simcausal** package differs from other R packages that implement data simulation based on structural equation modeling in the following ways. First, **simcausal** does not restrict the set of distributions available to the analyst to conduct a simulation study, i.e., any distribution that is currently available in R or that can be user-defined in the R programming environment can be used to sample observations in the **simcausal** package. In particular, the **simcausal** package is not restricted to data simulation based on linear structural equations only. Thus, this package allows the analyst to specify arbitrary functional dependencies between random variables, and, hence, enables data simulation from a much larger set of data generating mechanisms. Second, the **simcausal** package introduces an intuitive user-interface for specifying complex data-generating distributions to emulate realistic real-world longitudinal data studies characterized by a large number of repeated measurements of the same subject-matter attributes over time. Third, this package is particularly tailored to conduct data simulations to study causal inference methodologies for investigat-

ing the effect of complex intervention regimens such as dynamic and stochastic interventions (not just the common static and deterministic intervention regimens), and summary measures of these effects defined by (working) marginal structural models. The anticipated practical utility of this package thus extends beyond methodological research purposes by providing a tool for simulation-based power calculations to inform the design and analyses of real-world studies. Finally, the **simcausal** package provides a pipeline for conducting the typical steps of most simulation studies that consists of defining the observed data distribution, defining intervention/counterfactual distributions, defining causal parameters, simulating observed and counterfactual data, and evaluating the true value of causal parameters.

The rest of this vignette is organized as follows. In Section 2, we provide an overview of the technical details for a typical use of the package. In Section 3, we describe a template workflow for a simple simulation study with single time point interventions. In Section 4, we describe the use of the package for a more realistic and complex simulation study example based on survival data with repeated measures and dynamic interventions at multiple time points. In Section 5, we apply the **simcausal** package to replicate results of a previously published simulation study by Neugebauer *et al.* (2014, 2015). In Section 6, we apply the **simcausal** package to replicate results of another published simulation study by Lefebvre *et al.* (2008). We conclude with a discussion in Section 7.

2. Technical details

2.1. The workflow

The following schematic shows the order in which **simcausal** routines would be utilized in a typical simulation study:

$$
\begin{array}{ccc}
D = \texttt{DAG.empty}() & & \\
\downarrow & & \\
D = D + \texttt{node}(\ldots) + \texttt{node}(\ldots) & & \\
\downarrow & & \\
D = \texttt{set.DAG}(D) & \longrightarrow & \texttt{sim}(D) \\
\downarrow & & \\
D = D + \texttt{action}(\ldots) & \longrightarrow & \texttt{sim}(D, \texttt{actions} = \ldots) \\
\downarrow & & \\
D = \texttt{set.targetE}(D, \ldots) & & \\
D = \texttt{set.targetMSM}(D, \ldots) & & \\
\downarrow & & \\
\texttt{eval.target}(D) & &
\end{array}
$$

Data structures. The following most common types of output are produced by the package.

parameterized causal* DAG *model - object that specifies the structural equation model, along with interventions and the causal target parameter of interest.

observed data - data simulated from the (pre-intervention) distribution specified by the structural equation model.

full data - data simulated from one or more post-intervention distributions defined by actions on the structural equation model.

causal target parameter - the true value of the causal target parameter evaluated with full data.

Routines. The following routines will be generally invoked by a user, in the same order as presented below.

`DAG.empty` initiates an empty `DAG` object that contains no nodes.

`node` defines a node in the structural equation model and its conditional distribution, i.e., the outcome of one equation in the structural equation model and the formula that links the outcome value to that of earlier covariates, referred to as parent nodes. A call to `node` can specify either a single node or multiple nodes at once, with `name` and `distr` being the only required arguments. To specify multiple nodes with a single `node` call, one must also provide an indexing vector of integers as an argument `t`. In this case, each node shares the same name, but is indexed by distinct values in `t`. The simultaneous specification of multiple nodes is particularly relevant for providing a shorthand syntax

for defining a time-varying covariate, i.e., for defining repeated measurements over time of the same subject-matter attribute, as shown in the example in Section 4.1.

`add.nodes` **or** `D + node` provide two equivalent ways of growing the structural equation model by adding new nodes and their conditional distributions. Informally, these routines are intended to be used to sequentially populate a `DAG` object with all the structural equations that make up the causal model of interest. See Sections 3.1 and 4.1 for examples.

`set.DAG` locks the `DAG` object in the sense that no additional nodes can be subsequently added to the structural equation model. In addition, this routine performs several consistency checks of the user-populated `DAG` object. In particular, the routine attempts to simulate observations to verify that all conditional distributions in the `DAG` object are well-defined.

`sim` simulates independent and identically distributed (iid) observations of the complete node sequence defined by a `DAG` object. The output dataset is stored as a `data.frame` and is referred to as the *observed data*. It can be structured in one of two formats, as discussed in Section 4.5.

`add.action` **or** `D + action` provides two equivalent ways to define one or more actions. An action modifies the conditional distribution of one or more nodes of the structural equation model. The resulting data generating distribution is referred to as the post-intervention distribution. It is saved in the `DAG` object alongside the original structural equation model. See Sections 3.3 and 4.3 for examples.

`sim(..., actions = ...)` can also be used for simulating independent observations from one or more post-intervention distributions, as specified by the `actions` argument. The resulting output is a named list of `data.frame` objects, collectively referred to as the *full data*. The number of `data.frame` objects in this list is equal to the number of post-intervention distributions specified in the `actions` argument, where each `data.frame` object is an iid sample from a particular post-intervention distribution.

`set.targetE` **and** `set.targetMSM` define two distinct types of target causal parameters. The output from these routines is the input `DAG` object with the definition of the target causal parameter saved alongside the interventions. See Sections 3.5 and 4.7 for examples defining various target parameters.

`eval.target` evaluates the causal parameter of interest using simulated full data. As input, it can take previously simulated full data (i.e., the output of a call to the `sim(..., actions = ...)` function) or, alternatively, the user can specify the sample size `n`, based on which full data will be simulated first.

2.2. Specifying a structural equation model

The **simcausal** package encodes a structural equation model using a `DAG` object. The `DAG` object is a collection of nodes, each node represented by a `DAG.node` object that captures a single equation of the structural equation model. `DAG.node` objects are created by calling the `node` function. When the `node` function is used to simultaneously define multiple nodes, these nodes share the same name, but must be indexed by distinct user-specified integer values of

the time variable `t`, as shown in the example in Section 4.1. We will refer to a collection of nodes defined simultaneously in this manner as a *time-varying node* and we will refer to each node of such a collection as a measurement at a specific time point.

Each node is usually added to a growing `DAG` object by using either the `add.nodes` function or equivalently the `'+'` function, as shown in the example in Sections 3.1 and 4.1. Each new node added to a `DAG` object must be uniquely identified by its name or the combination of a name and a value for the time variable argument `t`.

The user may explicitly specify the temporal ordering of each node using the `order` argument of the `node()` function. However, if this argument is omitted, the `add.nodes` function assigns the temporal ordering to a node by using the actual order in which this node was added to the `DAG` object and, if applicable, the value of the time variable that indexes this node (earlier added nodes receive a lower order value, compared to those that are added later; nodes with a lower value for the `t` argument receive a lower order value, compared to those with a higher value of `t`).

The `node` function also defines the conditional distribution of a node, given its parents, with a combination of the sampling distribution specified by the `distr` argument and the distributional parameters specified as additional named arguments to the `node()` function. This `distr` argument can be set to the name of any R function that accepts an integer argument named `n` and returns a vector of size `n`. Examples and specifications for the types of distribution functions that can be used in **simcausal** are provided in Section 3.6.

The distributional parameters (also referred to as *node formulas* thereafter) are specified as additional named arguments of the `node()` function and can be either constants or some summary measures of the parent nodes. Their values can be set to any evaluable `R` expressions that may reference any standard or user-specified `R` function, and also, may invoke a novel and intuitive shorthand syntax for referencing specific measurements of time-varying parent nodes, i.e., nodes identified by the combination of a node name and a time point value `t`. The syntax for identifying specific measurements of time-varying nodes is based on a re-purposed R square-bracket vector subsetting function `'['`: e.g., writing the expression `sum(A[0:5])`, where `A` is the name of a previously defined time-varying node, defines the summary measure that is the sum of the node values over time points `t = 0,...,5`. This syntax may also be invoked to simultaneously define the conditional distribution of the measurements of a time-varying node over multiple time points `t` at once. For example, defining the conditional distribution of a time-varying node with the R expression `sum(A[max(0, t - 5):t]) + t` will resolve to different node formulas for each measurement of the time-varying node, depending on the value of `t`:

1. `A[0]` at `t = 0`;
2. `sum(A[0:1]) + 1` at `t = 1`, ..., `sum(A[0:5]) + 5` at `t = 5`;
3. `sum(A[1:6]) + 6` at `t = 6`, ..., `sum(A[5:10]) + 10` at `t = 10`.

Concrete applications of this syntax are described in Section 4.1, as well as in the documentation of the `node()` function (`?node`).

Note that the user can also define a causal model with one or more nodes that represent the occurrence of *end of follow-up* (EFU) events (e.g., right-censoring events or failure events of interest). Such nodes are defined by calling the `node()` function with the `EFU` argument being set to `TRUE`. The EFU nodes encode binary random variables whose value of 1 indicates that, by default, all of the subsequent nodes (i.e., nodes with a higher temporal order value) are to be replaced with a constant `NA` (missing) value. As an alternative, the user may choose to impute missing values for the time-varying node that represents the failure event of interest using the *last time point value carried forward* (LTCF) imputation method. This imputation procedure consists in replacing missing values for measurements of a time-varying node at time points `t` after an end of follow-up event with its last known measurement value prior to the occurrence of an end of follow-up event. Additional details about this imputation procedure are provided in Sections 2.5 and 4.6 and its relevance is demonstrated in Section 4.7.2 (Example 1 of `set.targetMSM`).

Finally, we note that the package includes pre-written wrapper functions for random sampling from some commonly employed distributions. These routines can be passed directly to the `distr` argument of the node function with the relevant distributional parameters on which they depend. These built-in functions can be listed at any time by calling `distr.list()`. In particular, the routines `"rbern"`, `"rconst"`, and `"rcategor"` can be used for specifying a Bernoulli distribution, a degenerate distribution (constant at a given value), and a categorical distribution, respectively. One can also use any of the standard random generating R functions, e.g., `"rnorm"` for sampling from the normal distribution and `"runif"` for sampling from the uniform distribution, as demonstrated in Sections 3.1 and 3.6.

2.3. Specifying interventions

An intervention regimen (also referred to as action regimen) is defined as a sequence of conditional distributions that replace the original distributions of a subset of nodes in a `DAG` object. To specify an intervention regimen, the user must identify the set of nodes to be intervened upon and provide new node distributions for them. The user may define a static, dynamic, deterministic or stochastic intervention on any given node, depending on the type of distributions specified. A deterministic static intervention is characterized by replacing a node distribution with a degenerate distribution such that the node takes on a constant value. A deterministic dynamic intervention is characterized by a conditional degenerate distribution such that the node takes on a value that is only a function of the values of its parents (i.e., a decision rule). A stochastic intervention is characterized by a non-degenerate conditional distribution. A stochastic intervention is dynamic if it is characterized by a non-degenerate conditional distribution that is defined as a function of the parent nodes and it is static otherwise. Note that a particular intervention may span different types of nodes and consist of different types of distributions, e.g., an intervention on a monitoring node can be static, while the intervention on a treatment node from the same structural equation model may be dynamic.

To define an intervention the user must call `D + action(A, nodes = B)` (or equivalently `add.action(D, A, nodes = B)`), where `D` is a `DAG` object, `A` is a unique character string that

represents the intervention name, and `B` is a list of `DAG.node` objects defining the intervention regimen. To construct `B` the user must first aggregate the output from one or more calls to `node` (using `c(..., ...)`), with the `name` argument of the `node` function call set to node names that already exist in the locked `DAG` object `D`. The example in Section 4.3 demonstrates this functionality. Alternatively, repeated calls to `add.action` or `D+action` with the same intervention name, e.g., `A = "A1"`, allow the incremental definition of an intervention regimen by passing each time a different `node` object, enabling iterative build-up of the collection `B` of the intervened nodes that define the intervention regimen. Note, however, that by calling `D + action` or `add.action(D, ...)` with a new action name, e.g., `action("A2", ...)`, the user initiates the definition of a new intervention regimen.

2.4. Specifying a target causal parameter

The causal parameter of interest (possibly a vector) is defined by either calling the function `set.targetE` or `set.targetMSM`. The function `set.targetE` defines causal parameters as the expected value(s) of `DAG` node(s) under one post-intervention distribution or the contrast of such expected value(s) from two post-intervention distributions. The function `set.targetMSM` defines causal parameters based on a **working** marginal structural model (Neugebauer and van der Laan 2007). In both cases, the true value of the causal parameter is defined by one or several post-intervention distributions and can thus be approximated using full data.

The following types of causal parameters can be defined with the function `set.targetE`:

- The expectation of an outcome node under an intervention regimen denoted by d, where the outcome under d is denoted by Y_d. This parameter can be naturally generalized to a vector of measurements of a time-varying node, i.e., the collection of nodes $Y_d(t)$ sharing the same name, but indexed by distinct time points `t` that represents a sequence of repeated measurements of the same attribute (e.g., a CD4 count or the indicator of past occurrence of a given failure event):

$$E(Y_d) \text{ or } (E(Y_d(t)))_{t=0,1,\ldots}.$$

- The difference between two expectations of an outcome node under two interventions, d_1 and d_0. This parameter can also be naturally generalized to a vector of measurements of a time-varying node:

$$E(Y_{d_1}) - E(Y_{d_0}) \text{ or } (E(Y_{d_1}(t)) - E(Y_{d_0}(t)))_{t=0,1,\ldots}.$$

- The ratio of two expectations of an outcome node under two interventions. This parameter can also be naturally generalized to a vector of measurements of a time-varying node:

$$\frac{E(Y_{d_1})}{E(Y_{d_0})} \text{ or } \left(\frac{E(Y_{d_1}(t))}{E(Y_{d_0}(t))}\right)_{t=0,1,\ldots}.$$

Note that if the `DAG` object contains nodes of type `EFU = TRUE` other than the outcome nodes of interest $Y_d(t)$, the target parameter must be defined by intervention regimens that set all such nodes that precede all outcomes of interest $Y_d(t)$ to 0. Also note that with such intervention regimens, if the outcome node is time-varying of type `EFU = TRUE` then the nodes

$Y_d(t)$ remain well defined (equal to 1) even after the time point when the simulated value for the outcome jumps to 1 for the first time. The nodes $Y_d(t)$ can then be interpreted as indicators of past failures in the absence of right-censoring events. The specification of these target parameters is covered with examples in Sections 3.5.1 and 4.7.1.

When the definition of the target parameter is based on a working marginal structural model, the vector of coefficients (denoted by α) of the working model defines the target parameter. The definition of these coefficients relies on the specification of a particular weighting function when the working model is not a correct model (see Neugebauer and van der Laan (2007) for details). This weighting function is set to the constant function of 1 in this package. The corresponding true value of the coefficients α can then be approximated by running a standard (unweighted) regression routines applied to simulated full data observations. The following types of working models, denoted by $m()$, can be defined with the function `set.targetMSM`:

- The working linear or logistic model for the expectation of one outcome node under intervention d, possibly conditional on baseline node(s) V, where a baseline node is any node preceding the earliest node that is intervened upon, i.e., $E(Y_d \mid V)$:

  $$m(d, V \mid \alpha).$$

  Such a working model can, for example, be used to evaluate the effects of HIV treatment regimens on the mean CD4 count measured at one point in time.

- The working linear or logistic model for the expectation vector of measurements of a time-varying outcome node, possibly conditional on baseline node(s) V, i.e., $E(Y_d(t) \mid V)$:

  $$m(t, d, V \mid \alpha), \text{ for } t = 0, 1, \ldots.$$

  Such a working model can, for example, be used to evaluate the effects of HIV treatment regimens on survival probabilities over time.

- The logistic working model for discrete-time hazards, i.e., for the probabilities that a measurement of a time-varying outcome node of type `EFU=TRUE` is equal to 1 under intervention d, given that the previous measurement of the time-varying outcome node under intervention d is equal to 0, possibly conditional on baseline node(s) V, i.e., $E(Y_d(t) \mid Y_d(t-1) = 0, V)$:

  $$m(t, d, V), \text{ for } t = 0, 1, \ldots.$$

  Such a working model can, for example, be used to evaluate the effects of HIV treatment regimens on discrete-time hazards of death over time.

Examples of the specification of the above target parameters are provided in Sections 3.5.2 and 4.7.2. As shown above, the working MSM formula $m()$ can be a function of t, V and d, where d is a unique identifier of each intervention regimen. In Sections 3.5.2 and 4.7.2 we describe in detail how to specify such identifiers for d as part of the `action` function call. Also note that the working MSM formula, m, may reference time-varying nodes using the square-bracket syntax introduced in Section 2.2, as long as all such instances are embedded within the syntax `S(...)`. This formula syntax can be used to define V, where V is defined by a baseline

measurement of a time-varying node. Example uses of this syntax are provided in Section 4.7.2 (Example 6 of `set.targetMSM`) and Section 4.7.2 (Example 7 of `set.targetMSM`).

2.5. Simulating data and evaluating the target causal parameter

The **simcausal** package can simulate two types of data: 1) observed data, sampled from the (pre-intervention) distribution specified by the structural equation model and 2) full data, sampled from one or more post-intervention distributions defined by actions on the structural equation model. Both types of data are simulated by invoking the `sim` function and the user can set the seed for the random number generator using the argument `rndseed`. The examples showing how to simulate observed data are provided in Sections 3.2 and 4.2, whereas the examples showing how to simulate full data are provided in Sections 3.4 and 4.4.

We note that two types of structural equation models can be encoded with the `DAG` object: 1) models where some or all nodes are defined by specifying the "time" argument `t` to the `node` function, or 2) models where the argument `t` is not used for any of the nodes. For the first type of structural equation models, the simulated data can be structured in either *long* or *wide* formats. A dataset is considered to be in wide format when each simulated observation of the complete sequence of nodes is represented by only one row of data, with each time-varying node represented by columns spanning distinct values of `t`. In contrast, for a dataset in long format, each simulated observation is typically represented by multiple rows of data, with separate rows of the same observation indexed by distinct values of `t` and each time-varying node represented by a single column. The format of the output data is controlled by setting the argument `wide` of the `sim` function to `TRUE` or `FALSE`. The default setting for `sim` is to simulate data in wide format, i.e., `wide = TRUE`. An example describing these two formats is provided in Section 4.5 in the context of time-varying measurements.

In addition, as previously described, for nodes representing the occurrence of end of follow-up events (i.e., censoring or outcome nodes declared with `EFU = TRUE`), the value of 1 indicates that, during data simulation, by default, all values of subsequent nodes (including the outcome nodes) are set to missing (`NA`). To instead impute these missing values after a particular end of follow-up event occurs (typically the outcome event) with the *last time point value carried forward* (LTCF) method, the user must set the argument `LTCF` of the `sim` function to the name of the `EFU`-type node that represents the end of follow-up event of interest. This will result in carrying forward the last observed measurement value for all time-varying nodes, after the value of the EFU node whose name is specified by the `LTCF` argument is observed to be 1. For additional details see the package documentation for the function `sim`. Examples demonstrating the use of the `LTCF` argument for data simulation are provided in Sections 4.6 and 4.7.1 (Example 1 of `set.targetMSM`).

In the last step of a typical workflow, the function `eval.target` is generally invoked for estimation of the true value of a previously defined target causal parameter. The true value is estimated using full data simulated from post-intervention distributions. The function `eval.target` can be called with either previously simulated full data, specified by the argument `data` or a sample size value, specified by the argument `n`. In the latter case, full data with the user-specified sample size will be simulated first.

3. Simulation study with single time point interventions

This example describes a typical workflow for specifying a simple structural equation model, defining various interventions, simulating observed and full data, and calculating various causal target parameters. The structural equation model chosen here illustrates a common point treatment problem in which one is interested in evaluating the effect of an intervention on one treatment node on a single outcome node using observational data with confounding by baseline covariates. This example also demonstrates the use of a plotting functionality of the **simcausal** package that builds upon the **igraph** R package (Csardi and Nepusz 2006) to visualize the Directed Acyclic Graph (DAG) (Pearl 1995, 2009, 2010) implied by the structural equation model encoded in the `DAG` object.

3.1. Specifying the structural equation model

The example below shows how to specify a structural equation model with six nodes to represent a single time point intervention study: one categorical baseline covariate with 3 categories (`race`), one normally distributed baseline confounder (`W1`), one uniformly distributed baseline confounder (`W2`), one binary baseline confounder (`W3`), one binary exposure (`Anode`), and one binary outcome (`Y`). This example uses pre- defined R functions `rcategor`, `rbern`, `rnorm` and `runif` for sampling from the categorical, Bernoulli, normal, and uniform distributions, respectively. For details and examples on writing sampling functions for arbitrary distributions see Section 3.6. We also refer to Section 3.6 for a description on how to specify node formulas (distributional parameters), such as with the use of R expressions specified by the `probs`, `mean` and `prob` arguments in the example below.

```
library(simcausal)
D <- DAG.empty()
D <- D +
        node("race",
              distr = "rcategor",
              probs = c(0.5, 0.25, 0.25)) +
        node("W1",
              distr = "rnorm",
              mean = ifelse(race == 1, 0, ifelse(race == 2, 3, 10)),
              sd = 1) +
        node("W2",
              distr = "runif",
              min = 0, max = 1) +
        node("W3",
              distr = "rbern",
              prob = plogis(-0.5 + 0.7 * W1 + 0.3 * W2)) +
        node("Anode",
              distr = "rbern",
              prob = plogis(-0.5 - 0.3 * W1 - 0.3 * W2 - 0.2 * W3)) +
        node("Y",
              distr = "rbern",
              prob = plogis(-0.1 + 1.2 * Anode + 0.1 * W1 + 0.3 * W2 + 0.2 * W3))
Dset <- set.DAG(D)
```

Running the code above results in implicitly assigning a sampling order (temporal order) to each node, based on the order in which the nodes were added to the `DAG` object `D`. Alternatively, one can use the optional `node()` argument `order` to explicitly specify the integer value

of the sampling order of each node, as described in more detail in the documentation for the `node` function.

The resulting internal representation of the structural equation model encoded by the `DAG` object `Dset` can be examined as follows (output showing only the first node of `Dset`).

```
str(Dset[1])
```

```
## List of 1
##  $ race:List of 7
##   ..$ name       : chr "race"
##   ..$ t          : NULL
##   ..$ distr      : chr "rcategor"
##   ..$ dist_params:List of 1
##   .. ..$ probs: chr "c(0.5, 0.25, 0.25)"
##   .. ..- attr(*, "asis.flags")=List of 1
##   .. .. ..$ probs: logi FALSE
##   ..$ EFU        : NULL
##   ..$ order      : num 1
##   ..$ node.env   :<environment: R_GlobalEnv>
##   ..- attr(*, "class")= chr "DAG.node"
```

Figure 1 shows the plot of the DAG that is generated by calling function `plotDAG`. This DAG is implied by the structural equation model specified above and the plotting is accomplished by using the visualization functionality from the **igraph** package (Csardi and Nepusz 2006). The directional arrows represent the functional dependencies in the structural equation model. More specifically, the node of origin of each arrow is an extracted node name from the *node formula(s)*. Note that the appearance of the resulting diagram can be customized with additional arguments, as shown in the example below and in the `plotDAG` example in Section 4.1.

```
plotDAG(Dset, xjitter = 0.3, yjitter = 0.01,
        edge_attrs = list(width = 0.5, arrow.width = 0.4, arrow.size = 0.8),
        vertex_attrs = list(size = 12, label.cex = 0.8))
```

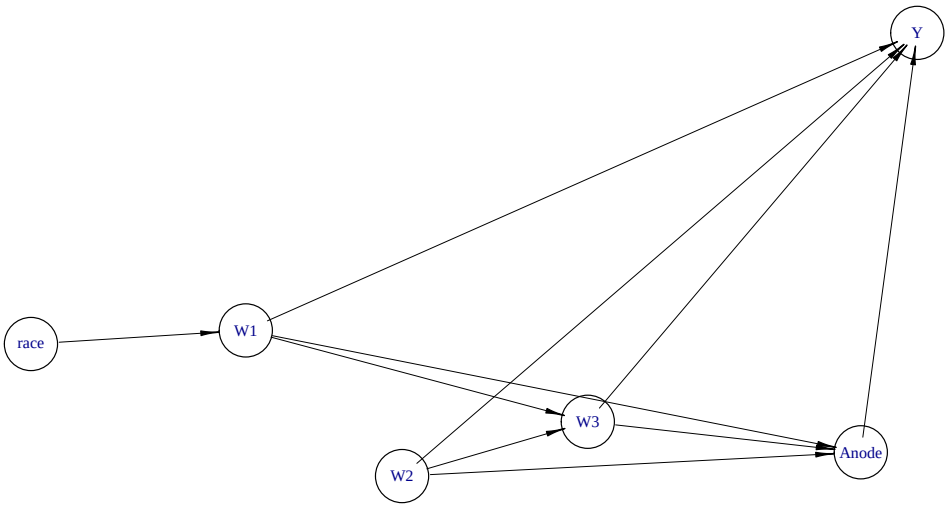

Figure 1: Graphical representation of the structural equation model using a DAG.

3.2. Simulating observed data (`sim`)

Simulating observed data is accomplished by calling the function `sim` and specifying its arguments `DAG` and `n` that indicate the causal model and sample size of interest. Below is an example of how to simulate an observed dataset with 10,000 observations using the causal model defined in the previous section. The output is a `data.frame` object.

```
Odat <- sim(DAG = Dset, n = 10000, rndseed = 123)
```

The format of the output dataset is easily understood by examining the first row of the `data.frame` object:

```
Odat[1,]

##   ID race         W1        W2 W3 Anode Y
## 1  1    1 -0.4941739 0.0878923  0     0 0
```

3.3. Specifying interventions (`+ action`)

The example below defines two actions on the treatment node. The first action named `"A1"` consists in replacing the distribution of the treatment node `Anode` with the degenerate distribution at value 1. The second action named `"A0"` consists in replacing the distribution of the treatment node `Anode` with the degenerate distribution at value 0. As shown below, these interventions are defined by invoking the `+ action` syntax on the existing `DAG` object. This syntax automatically adds and saves the new intervention object within the original `DAG` object, without overwriting it.

```
A1 <- node("Anode", distr = "rbern", prob = 1)
Dset <- Dset + action("A1", nodes = A1)
A0 <- node("Anode", distr = "rbern", prob = 0)
Dset <- Dset + action("A0", nodes = A0)
```

The added actions can be examined by looking at the result of the call `A(Dset)`. Note that `A(Dset)` returns a list of `DAG.action` objects, with each `DAG.action` encoding a particular post-intervention distribution, i.e., it is a modified copy of the original `DAG` object, where the original distribution of the node `Anode` is replaced with the degenerate distribution at value 0 or 1, for actions `"A0"` and `"A1"`, respectively.

```
  names(A(Dset))

## [1] "A1" "A0"

  class(A(Dset)[["A0"]])

## [1] "DAG.action"
```

The following example shows how to display the modified distribution of the intervention node `Anode` under action `"A0"`:

```
A(Dset)[["A0"]]$Anode

## List of 7
##  $ name       : chr "Anode"
##  $ t          : NULL
##  $ distr      : chr "rbern"
##  $ dist_params:List of 1
##   ..$ prob: chr "0"
##   ..- attr(*, "asis.flags")=List of 1
##   .. ..$ prob: logi FALSE
##  $ EFU        : NULL
##  $ order      : num 5
##  $ node.env   :<environment: R_GlobalEnv>
##  - attr(*, "class")= chr "DAG.node"
```

To examine the complete internal representation of the `DAG.action` object for action `"A0"`, invoke the `str()` function as follows (output not shown):

```
str(A(Dset)[["A0"]])
```

3.4. Simulating full data (`sim`)

Simulating full data is accomplished by calling the function `sim` and specifying its arguments `DAG`, `actions` and `n` to indicate the causal model, interventions, and sample size of interest. Full data can be simulated for all actions stored in the `DAG` object or a subset by setting the `actions` argument to the vector of the desired action names.

The example below shows how to use the `sim` function to simulate 100,000 observations for each of the two actions, `"A1"` and `"A0"`. These actions were defined as part of the `DAG` object `Dset` above. The call to `sim` below produces a list of two named `data.frame` objects, where each `data.frame` object contains observations simulated from the same post-intervention distribution defined by one particular action only.

```
  Xdat1 <- sim(DAG = Dset, actions = c("A1", "A0"), n = 100000, rndseed = 123)
  names(Xdat1)

## [1] "A1" "A0"

  nrow(Xdat1[["A1"]])

## [1] 100000

  nrow(Xdat1[["A0"]])

## [1] 100000
```

The format of the output list is easily understood by examining the first row of each `data.frame` object:

```
  Xdat1[["A1"]][1, ]

##   ID race        W1        W2 W3 Anode Y
## 1  1    1 0.2595897 0.7929289  1     1 1

  Xdat1[["A0"]][1, ]

##   ID race        W1        W2 W3 Anode Y
## 1  1    1 0.2595897 0.7929289  1     0 1
```

3.5. Defining and evaluating various causal target parameters

Causal parameters defined with `set.targetE`

The first example below defines the causal quantity of interest as the expectation of node `Y` under action `"A1"`:

```
  Dset <- set.targetE(Dset, outcome = "Y", param = "A1")
```

The true value of the above causal parameter is now evaluated by calling the function `eval.target` and passing the previously simulated full data object `Xdat1`.

```
  eval.target(Dset, data = Xdat1)$res

##  Mean_Y
## 0.83771
```

Alternatively, `eval.target` can be called without the simulated full data, specifying the sample size argument `n` instead. In this case a full dataset with the user-specified sample size is simulated first.

```
  eval.target(Dset, n = 100000, rndseed = 123)$res

##  Mean_Y
## 0.83771
```

The example below defines the causal target parameter as the ATE on the additive scale, i.e. the expectation of `Y` under action `"A1"` minus its expectation under action `"A0"`.

```
  Dset <- set.targetE(Dset, outcome = "Y", param = "A1-A0")
  eval.target(Dset, data = Xdat1)$res

##  Diff_Y
## 0.21973
```

Similarly, the ATE on the multiplicative scale can be evaluated as follows:

```
  Dset <- set.targetE(Dset, outcome = "Y", param = "A1/A0")
  eval.target(Dset, data = Xdat1)$res

##  Ratio_Y
## 1.355562
```

Causal parameters defined with `set.targetMSM`

To specify the MSM target causal parameter, the user must provide the following arguments to `set.targetMSM`: (1) the `DAG` object that contains all and only the actions of interest; (2) `outcome`, the name of the outcome node (possibly time-varying); (3) for a time-varying outcome node, the vector of time points `t` that index the outcome measurements of interest; (4) `form`, the regression formula defining the working MSM; (5) `family`, the working model family that is passed on to `glm`, e.g., `family = "binomial"` or `family = "gaussian"` for a logistic or a linear working model; and (6) for time-to-event outcomes, the logical flag `hazard` that indicates whether the working MSM describes discrete-time hazards (`hazard = TRUE`) or survival probabilities (`hazard = FALSE`).

In the examples above, the two actions `"A1"` and `"A0"` are defined as deterministic static interventions on the node `Anode`, setting it to either constant 0 or 1. Thus, each of these two interventions is uniquely indexed by the post-intervention value of the node `Anode` itself. In the following example, we instead introduce the variable $d \in \{0, 1\}$ to explicitly index each of the two post-intervention distributions when defining the two actions of interest. We then define the target causal parameter as the coefficients of the following linear marginal structural model $m(d \mid \alpha) = \alpha_0 + \alpha_1 d$. As expected, the estimated true value for α_1 obtained below corresponds exactly with the estimated value for the ATE on the additive scale obtained above by running `set.targetE` with the parameter `param = "A1-A0"`.

As just described, we now redefine the actions `"A1"` and `"A0"` by indexing the intervention node formula (the distributional parameter `prob`) with parameter `d` before setting its values to 0 or 1 by introducing an additional new argument named `d` into the `action` function call.

This creates an action- specific attribute variable `d` whose value uniquely identifies each of the two actions and that will be included as an additional column variable to the simulating full data sets.

```
A1 <- node("Anode", distr = "rbern", prob = d)
Dset <- Dset + action("A1", nodes = A1, d = 1)
A0 <- node("Anode",distr = "rbern", prob = d)
Dset <- Dset + action("A0", nodes = A0, d = 0)
```

Creating such an action-specific attribute `d` allows it to be referenced in the MSM regression formula as shown below:

```
msm.form <- "Y ~ d"
Dset <- set.targetMSM(Dset, outcome = "Y", form = msm.form, family = "gaussian")
msm.res <- eval.target(Dset, n = 100000, rndseed = 123)
msm.res$coef

## (Intercept)           d
##     0.61798     0.21973
```

3.6. Defining node distributions and vectorizing node formula functions

To facilitate the comprehension of the following two subsections, we note that, in the **simcausal** package, simulation of observed or full data follows the temporal ordering of the nodes that define the DAG object and is **vectorized**. More specifically, the simulation of a dataset with sample size `n` is carried out by first sampling the vector of all `n` observations of the first node, before sampling the vector of all `n` observations of the second node and so on, where the node ranking is defined by the temporal ordering that was explicitly or implicitly specified by the user during the creation of the DAG object (see Section 2.2 for a discussion of temporal ordering).

Defining node distributions

The distribution of a particular node is specified by passing the name of an evaluable R function to the `distr` argument of the function `node`. Such a distribution function must implement the mapping of `n` independent realizations of the parent nodes into `n` independent realizations of this node. In general, any node with a lower temporal ordering can be defined as a parent. Thus, such a distribution function requires an argument `n`, but will also typically rely on additional input arguments referred to as distributional parameters. In addition, the output of the distribution function must also be a vector of length `n`. Distributional parameters must be either scalars or vectors of `n` realizations of summary measures of the parent nodes. The latter types of distributional arguments are referred to as the *node formula(s)* because they are specified by evaluable R expressions. Node formulas are passed as named arguments to the `node` function so they can be mapped uniquely to the relevant argument of the function that is user-specified by the `distr` argument of the `node` function call. The node formula(s) of any given node may invoke the name(s) of any other node(s) with a lower temporal order value. The parents of a particular node are thus defined as the collection of nodes that are referenced by its node formula(s). Note that unlike the values of distributional parameters, the value of the argument `n` of the `distr` function is internally determined during

data simulation and is set to the sample size value passed to the `sim` function by the user.

For example, as shown below, the pre-written wrapper function for the Bernoulli distribution `rbern` is defined with two arguments, `n` and `prob`. When defining a node with the `distr` argument set to `"rbern"`, only the second argument must be explicitly user-specified by a distributional parameter named `prob` in the call to the `node` function, e.g., `node("N1", distr="rbern", prob = 0.5)`. The argument `prob` can be either a numeric constant as in the previous example or an evaluable R expression. When `prob` is a numeric constant, the distribution function `rbern` returns `n` iid realizations of the Bernoulli random variable with probability `prob`. When `prob` is an R expression (e.g., see the definition of node `W3` in Section 3.1) that involves parent nodes, the `prob` argument passed to the `rbern` function becomes a vector of length `n`. The value of each of its component is determined by the R expression evaluated using one of the `n` iid realizations of the parent nodes simulated previously. Thus, the resulting simulated independent observations of the child node (e.g., `W3` in Section 3.1) are not necessarily identically distributed if the vector `prob` contains distinct values. We note that the R expression in the `prob` argument is evaluated in the environment containing the simulated observations of all previous nodes (i.e., nodes with a lower temporal order value).

To see the names of all pre-written distribution wrapper functions that are specifically optimized for use as `distr` functions in the **simcausal** package, invoke `distr.list()`, as shown below:

```
distr.list()
```

```
## [1] "rbern"           "rcategor"        "rcategor.int"    "rconst"
## [5] "rdistr.template"
```

For a template on how to write a custom distribution function, see the documentation `?rdistr.template` and `rdistr.template`, as well as any of the pre-written distribution functions above. For example, the `rbern` function below simply wraps around the standard R function `rbinom` to define the Bernoulli random variable generator:

```
rbern
```

```
## function (n, prob)
## {
##     rbinom(n = n, prob = prob, size = 1)
## }
## <environment: namespace:simcausal>
```

Another example on how to write a custom distribution function to define a custom left-truncated normal distribution function based on the standard R function `rnorm` with arguments `mean` and `sd` is demonstrated below. The truncation level is specified by an additional distributional parameter `minval`, with default value set to 0.

```
rnorm_trunc <- function(n, mean, sd, minval = 0) {
  out <- rnorm(n = n, mean = mean, sd = sd)
  minval <- minval[1]
```

```
  out[out < minval] <- minval
  out
}
```

The example below makes use of this function to define the outcome node `Y` with positive values only:

```
Dmin0 <- DAG.empty()

Dmin0 <- Dmin0 +
                node("W",    distr = "rbern",
                     prob = plogis(-0.5)) +
                node("Anode", distr = "rbern",
                     prob = plogis(-0.5 - 0.3 * W)) +
                node("Y", distr = "rnorm_trunc",
                     mean = -0.1 + 1.2 * Anode + 0.3 * W,
                     sd = 10)

Dmin0set <- set.DAG(Dmin0)
```

In the next example, we overwrite the previous definition of node `Y` to demonstrate how alternative values for the truncation parameter `minval` may be passed by the user as part of the `node` function call:

```
Dmin0 <- Dmin0 +
                node("Y", distr = "rnorm_trunc",
                     mean = -0.1 + 1.2 * Anode + 0.3 * W,
                     sd = 10,
                     minval = 10)

Dmin10set <- set.DAG(Dmin0)
```

Finally, we illustrate how the `minval` argument can also be defined as a function of parent node realizations:

```
Dmin0 <- Dmin0 +
                node("Y", distr = "rnorm_trunc",
                     mean = -0.1 + 1.2 * Anode + 0.3 * W,
                     sd = 10,
                     minval = ifelse(Anode == 0, 5, 10))

Dminset <- set.DAG(Dmin0)
```

Vectorizing node formula functions (`vecfun.add`)

As just described, the distributional parameters defining a particular node distribution can be evaluable R expressions referred to as *node formulas*. These expressions can contain any build-in or user-defined R functions. By default any function inside such an R expression is assumed non-vectorized, except for functions on the **simcausal** built-in list of known vectorized functions. To see this list, call the `vecfun.all.print` function:

```
  vecfun.all.print()

## [1] "build-in vectorized functions:"
##  [1] "ifelse"     "+"          "-"          "*"          "^"          "/"
##  [7] "=="         "!="         "!"          "<"          ">"          "<="
## [13] ">="         "|"          "&"          "I"          "abs"        "sign"
## [19] "sqrt"       "round"      "signif"     "floor"      "ceil"       "ceiling"
## [25] "trunc"      "sin"        "tan"        "cos"        "acos"       "asin"
## [31] "atan"       "cosh"       "sinh"       "tanh"       "log"        "log10"
## [37] "log1p"      "exp"        "expm1"      "plogis"     "beta"       "lbeta"
## [43] "gamma"      "lgamma"     "psigamma"   "digamma"    "trigamma"   "choose"
## [49] "lchoose"    "factorial"  "lfactorial"
## [1] "user-defined vectorized functions: "
```

When parsing node formulas, the default behavior implemented in **simcausal** is to replace each call to any function not on this list (i.e., a function not recognized as vectorized) with a call to the `apply` function to loop over the rows of the dataset containing the simulated realizations of the parents nodes stored in wide format and to call the unrecognized function individually on every row of the data. For example, with the user-defined function `power2` below, the simulation of observations of the node `W3` first involves automatic replacement of the calls `power2(W1)` and `power2(W2)` in the node formulas `mean` and `sd` with the `apply` function calls `apply(W1, 1, power2)` and `apply(W2, 1, power2)`, respectively:

```
  power2 <- function(arg) arg^2

  D <- DAG.empty()
  D <- D +
          node("W1", distr = "rnorm",
                mean = 0, sd = 1) +
          node("W2", distr = "rnorm",
                mean = 0, sd = 1) +
          node("W3", distr = "rnorm",
                mean = power2(W1), sd = power2(W2))
```

This behavior results in increased computing time that can be avoided as described here. The list of known vectorized functions can be easily expanded to include any function `funname`, by simply invoking the call `vecfun.add(funname)` prior to data simulation with the `sim` function. Doing this avoids the often unnecessary calls to the `apply` loops shown above and typically results in a significant performance boost for data simulation.

The performance gain from vectorization is demonstrated with the `power2` function in the example below. We first examine the simulation time when `power2` is not added to the list of recognized vectorized functions as follows:

```
  D1 <- set.DAG(D)

## power2(W1)
## power2(W2)

  (tnonvec <- system.time(sim1nonvec <- simobs(D1, n = 1000000, rndseed = 123)))
```

```
## power2(W1)
## power2(W2)
##    user  system elapsed
##   7.117   0.175   7.327
```

Second, we examine the simulation time after adding `power2` to the list of recognized vectorized functions as follows:

```
  vecfun.add(c("power2"))

## [1] "current list of user-defined vectorized functions: power2"

  D1vec <- set.DAG(D)
  (tvec <- system.time(sim1vec <- simobs(D1vec, n = 1000000, rndseed = 123)))

##    user  system elapsed
##   0.366   0.040   0.406

  all.equal(sim1nonvec,sim1vec)

## [1] TRUE
```

We note that data simulation is approximately 18 times faster with the second approach above.

To see the names of user-added recognized vectorized functions, call `vecfun.print`. To reset this list, call `vecfun.reset`. Note that the built-in list of known vectorized functions cannot be modified/reset.

```
  vecfun.print()

## [1] "current list of user-defined vectorized functions: power2"

  vecfun.reset()
  vecfun.print()

## [1] "current list of user-defined vectorized functions: "
```

It is important to note that all non-vectorized (or unrecognized vectorized) functions referenced in a node formula, such as `ifelse1` in the example below, must be declared with at most one argument. Trying to declare such a function with more than one argument will result in an error. Indeed, the default behavior when parsing node formulas in **simcausal** is to aggregate the arguments of a node formula function not on the list of recognized vectorized functions into a matrix using a call to the `cbind` function. This matrix is then passed as a single argument to the `apply` loop that replaces the call to the user-specified node formula. For example with the custom version of the `ifelse` function defined below, the call to `ifelse1(c(W1, 0.5, 0.1))` is automatically replaced with a call to `apply(cbind(W1, 0.5, 0.1), 1, ifelse1)`:

```
  vecfun.reset()
  ifelse1 <- function(arg) {
    ifelse(arg[1], arg[2], arg[3])
  }

  D <- DAG.empty()
  D <- D +
          node("W1", distr = "rbern",
                prob = 0.05) +
          node("W2", distr = "rbern",
                prob = ifelse1(c(W1, 0.5, 0.1)))

  D2nonvec <- set.DAG(D)

## ifelse1(c(W1, 0.5, 0.1))
```

The restriction in the number of arguments for the node formula functions is easily avoided by adding functions such as `ifelse1` above to the list of recognized vectorized functions. Doing so allows the function to be declared with an arbitrary number of arguments. The example below demonstrates the use of the second custom version of `ifelse`, called `ifelse2`, which is declared with 3 arguments, added to the list of recognized vectorized functions and then called as part of the node formula of `W2`:

```
  ifelse2 <- function(arg, val1, val2) {
    ifelse(arg, val1, val2)
  }

  vecfun.add(c("ifelse2"))

## [1] "current list of user-defined vectorized functions: ifelse2"

  D <- DAG.empty()
  D <- D +
          node("W1", distr = "rbern",
                prob = 0.05) +
          node("W2", distr = "rbern",
                prob = ifelse2(W1, 0.5, 0.1))
  D2vec <- set.DAG(D)
```

The performance gain from vectorization is demonstrated below for the previous two custom versions of the `ifelse` function:

```
  (t2nonvec <- system.time(sim2nonvec <- simobs(D2nonvec, n = 100000, rndseed = 123)))

## ifelse1(c(W1, 0.5, 0.1))
##    user  system elapsed
##   1.350   0.013   1.368

  (t2vec <- system.time(sim2vec <- simobs(D2vec, n = 100000, rndseed = 123)))

##    user  system elapsed
##   0.038   0.007   0.045
```

```
  all(unlist(lapply(seq(ncol(sim2nonvec)),
              function(coli) all.equal(sim2nonvec[, coli], sim2vec[, coli]))))

## [1] TRUE
```

We note that the above data simulation is approximately 30 times faster with the `ifelse2` function compared to the `ifelse1` function. We also note that the above performance gain of vectorization will be larger if the sample size `n` is increased.

4. Simulation study with multiple time point interventions

In this example we replicate results from the longitudinal data simulation protocol used in two published manuscripts Neugebauer *et al.* (2014, 2015). We first describe the structural equation model that implies the data generating distribution of the observed data, with time-to-event outcome, as reported in Section 5.1 of Neugebauer *et al.* (2015). We then show how to specify this model using the **simcausal** R interface, simulate observed data, define static and dynamic time-varying interventions, simulate full data, and calculate various causal parameters based on these interventions. In particular, we replicate estimates of true counterfactual risk differences under the dynamic interventions reported in Neugebauer *et al.* (2014).

4.1. Specifying the structural equation model

In this section, we demonstrate how to specify the structural equation model described by the following longitudinal data simulation protocol (Section 5.1 of Neugebauer *et al.* (2015)):

1. $L_2(0) \sim \mathcal{B}(0.05)$ where $\mathcal{B}$ denotes the Bernoulli distribution (e.g., $L_2(0)$ represents a baseline value of a time-dependent variable such as low versus high A1c)
2. If $L_2(0) = 1$ then $L_1(0) \sim \mathcal{B}(0.5)$, else $L_1(0) \sim \mathcal{B}(0.1)$ (e.g., $L_1(0)$ represents a time-independent variable such as history of cardiovascular disease at baseline)
3. If $(L_1(0), L_2(0)) = (1, 0)$ then $A_1(0) \sim \mathcal{B}(0.5)$, else if $(L_1(0), L_2(0)) = (0, 0)$ then $A_1(0) \sim \mathcal{B}(0.1)$, else if $(L_1(0), L_2(0)) = (1, 1)$ then $A_1(0) \sim \mathcal{B}(0.9)$, else if $(L_1(0), L_2(0)) = (0, 1)$ then $A_1(0) \sim \mathcal{B}(0.5)$ (e.g., $A_1(0)$ represents the binary exposure to an intensified type 2 diabetes pharmacotherapy)
4. $A_2(0) \sim \mathcal{B}(0)$ (e.g., $A_2(0)$ represents occurrence of a right-censoring event)
5. for $t = 1, \ldots, 16$ and as long as $Y(t-1) = 0$ (by convention, $Y(0) = 0$):
   (a) $Y(t) \sim \mathcal{B}(\frac{1}{1+\exp(-(-6.5+L_1(0)+4L_2(t-1)+0.05*\sum_{j=0}^{t-1} I(L_2(j)=0)))})$ (e.g., Y(t) represents the indicator of failure such as onset or progression of albuminuria)
   (b) If $A_1(t-1) = 1$ then $L_2(t) \sim \mathcal{B}(0.1)$, else if $L_2(t-1) = 1$ then $L_2(t) \sim \mathcal{B}(0.9)$, else $L_2(t) \sim \mathcal{B}(\min(1, 0.1 + t/16))$
   (c) If $A_1(t-1) = 1$ then $A_1(t) = 1$, else if $(L_1(0), L_2(t)) = (1, 0)$ then $A_1(t) \sim \mathcal{B}(0.3)$, else if $(L_1(0), L_2(t)) = (0, 0)$ then $A_1(t) \sim \mathcal{B}(0.1)$, else if $(L_1(0), L_2(t)) = (1, 1)$ then $A_1(t) \sim \mathcal{B}(0.7)$, else if $(L_1(0), L_2(t)) = (0, 1)$ then $A_1(t) \sim \mathcal{B}(0.5)$
   (d) If $t = 16$ then $A_2(t) \sim \mathcal{B}(1)$ (e.g., administrative end of study), else $A_2(t) \sim \mathcal{B}(0)$ (e.g., no right-censoring).

First, the example below shows how to define the nodes `L2`, `L1`, `A1` and `A2` at time point `t = 0` as Bernoulli random variables, using the distribution function `"rbern"`:

```
library(simcausal)
D <- DAG.empty()
D <- D +
        node("L2", t = 0, distr = "rbern",
            prob = 0.05) +
        node("L1", t = 0, distr = "rbern",
            prob = ifelse(L2[0] == 1, 0.5, 0.1)) +
        node("A1", t = 0, distr = "rbern",
            prob = ifelse(L1[0] == 1 & L2[0] == 0, 0.5,
                      ifelse(L1[0] == 0 & L2[0] == 0, 0.1,
                        ifelse(L1[0] == 1 & L2[0] == 1, 0.9, 0.5)))) +
        node("A2", t = 0, distr = "rbern",
            prob = 0, EFU = TRUE)
```

Second, the example below shows how one may use the `node` function with node formulas based on the square bracket function `'['` to easily define the time-varying nodes `Y`, `L1`, `A1` and `A2` simultaneously for all subsequent time points `t` ranging from 1 to 16:

```
t.end <- 16
D <- D +
        node("Y", t = 1:t.end, distr = "rbern",
            prob =
              plogis(-6.5 + L1[0] + 4 * L2[t-1] + 0.05 * sum(I(L2[0:(t-1)] == rep(0, t)))),
            EFU = TRUE) +
        node("L2", t = 1:t.end, distr = "rbern",
            prob =
              ifelse(A1[t-1] == 1, 0.1,
                ifelse(L2[t-1] == 1, 0.9, min(1, 0.1 + t / 16)))) +
        node("A1", t = 1:t.end, distr = "rbern",
            prob = ifelse(A1[t-1] == 1, 1,
                      ifelse(L1[0] == 1 & L2[t] == 0, 0.3,
                        ifelse(L1[0] == 0 & L2[t] == 0, 0.1,
                          ifelse(L1[0] == 1 & L2[t] == 1, 0.7, 0.5))))) +
        node("A2", t = 1:t.end, distr = "rbern",
            prob = {if(t == 16) {1} else {0}},
            EFU = TRUE)
lDAG <- set.DAG(D)
```

Note that the `node` formulas specified with the `prob` argument above use the generic time variable `t` both outside and inside the square-bracket vector syntax. For example, the conditional distribution of the time-varying node `Y` is defined by an R expression that contains the syntax `sum(I(L2[0:(t - 1)] == rep(0, t)))`, which evaluates to different R expressions, as `t` ranges from 0 to 16:

1. `sum(I(L2[0] == 0))`, for `t = 1`; and
2. `sum(I(L2[0:1] == c(0, 0)))`, for `t = 2`, ..., `sum(I(L2[0:16] == c(0, ..., 0)))`, for `t = 16`.

For more details on the specification of node formulas, see Section 3.6.

One can visualize the observed data generating distribution defined in the `lDAG` object as shown in Figures 2 and 3 by calling `plotDAG`. The directional arrows represent the functional dependencies in the structural equation model. More specifically, the node of origin of each arrow is an extracted node name from the *node formula(s)*. Note that the appearance of the resulting diagram can be customized with additional arguments, as shown in the following examples. The argument `tmax` can be used to restrict the plotting of the DAG object to only the nodes indexed by time points below the user-specified `tmax` value, as shown in Figure 3.

Figure 2 is created by running the following code:

```
plotDAG(lDAG, xjitter = 0.3, yjitter = 0.01)
```

Figure 3 is created by running the following code:

```
plotDAG(lDAG, tmax = 3, xjitter = 0.3, yjitter = 0.02,
      edge_attrs = list(width = 0.5, arrow.width = 0.4, arrow.size = 0.8),
      vertex_attrs = list(size = 12, label.cex = 0.8))
```

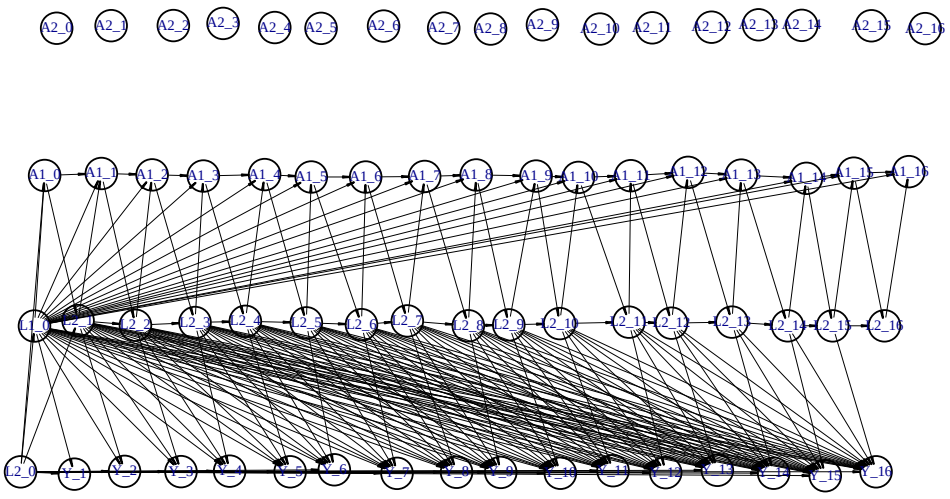

Figure 2: Graphical representation of the structural equation model using a DAG

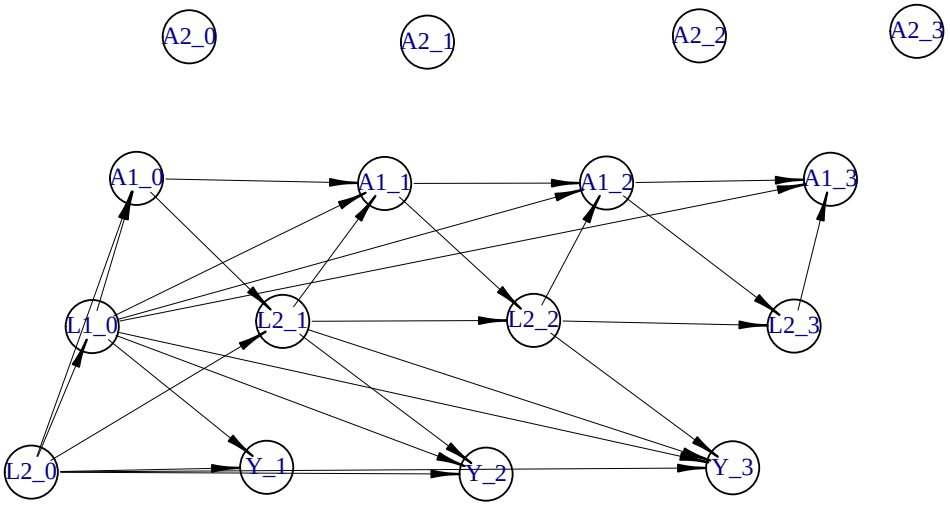

Figure 3: Graphical representation of a portion of the structural equation model using a DAG. Only the nodes indexed by time points lower than or equal to 3 are represented.

4.2. Simulating observed data (`sim`)

Simulating observed data is accomplished by calling the function `sim` and specifying its arguments `DAG` and `n` that indicate the causal model and sample size of interest. Below is an example of how to simulate an observed dataset with $10,000$ observations using the causal model defined previously. The output is a `data.frame` object.

```
Odat <- sim(DAG = lDAG, n = 10000, rndseed = 123)
```

The format of the output dataset is easily understood by examining the first 10 columns of the first row of the `data.frame` object:

```
Odat[1,1:10]

##   ID L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2
## 1  1    0    0    1    0   0    0    1    0   0
```

4.3. Specifying interventions (`+ action`)

Dynamic interventions

The following two dynamic interventions on the time-varying node `A1` of the structural equation model encoded by the previously defined `lDAG` object were studied in Neugebauer *et al.* (2014): '*Initiate treatment* A_1 *the first time* t *that the covariate* L_2 *is greater than or equal to* θ *and continue treatment thereafter (i.e.,* $\bar{A}_1(t-1) = 0$ *and* $A(t) = 1, A(t+1) = 1, \ldots$*)*', for $\theta = 0, 1$. The example below demonstrates how to specify these two dynamic interventions.

First, we define the list of intervention nodes and their post-intervention distributions. Note that these distributions are indexed by the attribute `theta`, whose value is not yet defined:

```
act_theta <-c(node("A1", t = 0, distr = "rbern",
                   prob = ifelse(L2[0] >= theta , 1, 0)),
              node("A1", t = 1:(t.end), distr = "rbern",
                   prob = ifelse(A1[t-1] == 1, 1, ifelse(L2[t] >= theta, 1, 0))))
```

Second, we add the two dynamic interventions to the `lDAG` object while defining the value of `theta` for each intervention:

```
Ddyn <- lDAG
Ddyn <- Ddyn + action("A1_th0", nodes = act_theta, theta = 0)
Ddyn <- Ddyn + action("A1_th1", nodes = act_theta, theta = 1)
```

We refer to the argument `theta` passed to the `+action` function as an *action attribute*.

One can select and inspect particular actions saved in a DAG object by invoking the function `A()`:

```
class(A(Ddyn)[["A1_th0"]])

## [1] "DAG.action"

A(Ddyn)[["A1_th0"]]

## [1] "Action: A1_th0"
## [1] "ActionNodes: A1_0, A1_1, ... , A1_15, A1_16"
## [1] "ActionAttributes: "
## $theta
## [1] 0
```

Figure 4 shows the plot of the `DAG.action` object associated with the dynamic intervention named `"A1_th0"` (with plotting restricted to nodes indexed by time points lower than or equal to 3). Note that the intervention nodes are marked in red and that the action attribute `theta` is represented as a separate node. The following code is used to generate Figure 4:

```
plotDAG(A(Ddyn)[["A1_th0"]], tmax = 3, xjitter = 0.3, yjitter = 0.02,
      edge_attrs = list(width = 0.5, arrow.width = 0.4, arrow.size = 0.8),
      vertex_attrs = list(size = 15, label.cex = 0.7))
```

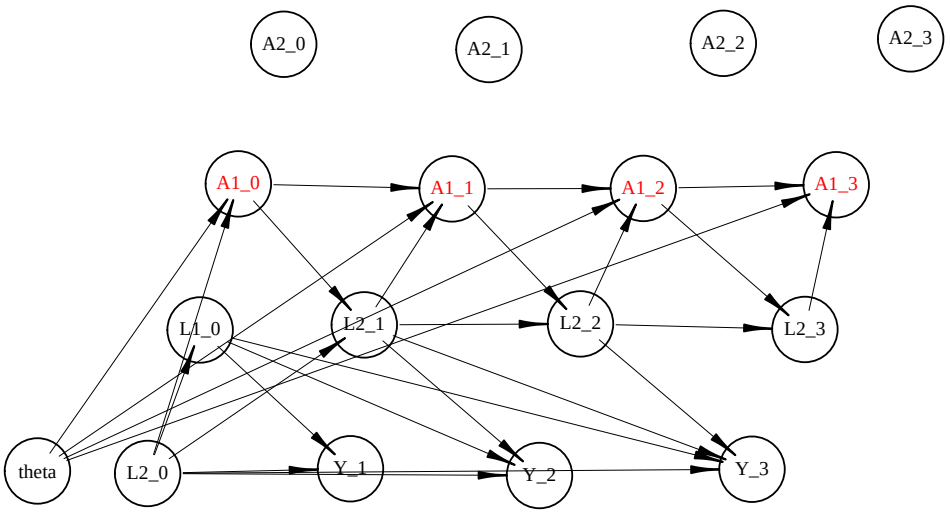

Figure 4: Graphical representation of the modified structural equation model resulting from a dynamic intervention. Only the nodes indexed by time points lower than or equal to 3 are represented.

The distribution of some or all of the the intervention nodes that define an action saved within a `DAG` object can be modified by adding a new intervention object with the same action name to the `DAG` object. The new intervention object can involve actions on only a subset of the original intervention nodes for a partial modification of the original action definition. For example, the code below demonstrates how the existing action `"A1_th0"` with the previously defined dynamic and deterministic intervention on the node `A1[0]` is partially modified by replacing the intervention distribution for the node `A1[0]` with a deterministic and static intervention defined by a degenerate distribution at value 1. Note that the other intervention nodes previously defined as part of the action `"A1_th0"` remain unchanged.

```
  A(Ddyn)[["A1_th0"]]$A1_0

## List of 7
##  $ name       : chr "A1_0"
##  $ t          : num 0
##  $ distr      : chr "rbern"
##  $ dist_params:List of 1
##   ..$ prob: chr "ifelse(L2[0] >= theta, 1, 0)"
##   ..- attr(*, "asis.flags")=List of 1
##   .. ..$ prob: logi FALSE
##  $ EFU        : NULL
##  $ order      : num 3
##  $ node.env   :<environment: R_GlobalEnv>
##  - attr(*, "class")= chr "DAG.node"

  Ddyntry <- Ddyn + action("A1_th0", nodes = node("A1", t = 0, distr = "rbern", prob = 0))
  A(Ddyntry)[["A1_th0"]]$A1_0

## List of 7
##  $ name       : chr "A1_0"
##  $ t          : num 0
```

```
##  $ distr      : chr "rbern"
##  $ dist_params:List of 1
##   ..$ prob: chr "0"
##   ..- attr(*, "asis.flags")=List of 1
##   .. ..$ prob: logi FALSE
##  $ EFU        : NULL
##  $ order      : num 3
##  $ node.env   :<environment: R_GlobalEnv>
##  - attr(*, "class")= chr "DAG.node"
```

Similarly, some or all of the action attributes that define an action saved within a `DAG` object can be modified by adding a new intervention object with the same action name but a different attribute value to the `DAG` object. This functionality is demonstrated with the example below in which the previous value 0 of the action attribute `theta` that defines the action named `"A1_th0"` is replaced with the value 1 and in which a new attribute `newparam` is simultaneously added to the previously defined action `"A1_th0"`:

```
  A(Ddyntry)[["A1_th0"]]


## [1] "Action: A1_th0"
## [1] "ActionNodes: A1_0, A1_1, ... , A1_15, A1_16"
## [1] "ActionAttributes: "
## $theta
## [1] 0

  Ddyntry <- Ddyntry + action("A1_th0", nodes = act_theta, theta = 1, newparam = 100)
  A(Ddyntry)[["A1_th0"]]


## [1] "Action: A1_th0"
## [1] "ActionNodes: A1_0, A1_1, ... , A1_15, A1_16"
## [1] "ActionAttributes: "
## $theta
## [1] 1
##
## $newparam
## [1] 100
```

Finally, we note that an action attribute can also be defined as a time-varying vector, rather than a scalar, i.e., a vector of scalars that are each indexed by a time point. This functionality is demonstrated in the example below to define interventions that are indexed by a scalar `theta` whose value changes over time. Note that in this example the square-bracket syntax `theta[t]` is used for referencing the time-varying values of the action attribute `theta`. More details on the use of time-varying action attributes are provided in the next section.

```
  act_theta_t <-c(node("A1",t = 0, distr = "rbern",
                       prob = ifelse(L2[0] >=  theta[t], 1, 0)),
                  node("A1",t = 1:t.end, distr = "rbern",
                       prob = ifelse(A1[t-1]==1, 1, ifelse(L2[t] >= theta[t], 1, 0)))
                  )
  Ddyntry <- Ddyntry + action("A1_th0", nodes = act_theta_t, theta = rep(0,(t.end)+1))
  A(Ddyntry)[["A1_th0"]]
```

```
## [1] "Action: A1_th0"
## [1] "ActionNodes: A1_0, A1_1, ... , A1_15, A1_16"
## [1] "ActionAttributes: "
## $theta
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##
## $newparam
## [1] 100
```

Static interventions

Here we diverge from the replication of simulation results presented in Neugebauer *et al.* (2014). Instead, we build on the structural equation model introduced in that paper to illustrate the specification of static interventions on the treatment nodes `A1`. These static interventions are defined by more or less early treatment initiation during follow-up followed by subsequent treatment continuation. Each of these static interventions is thus uniquely identified by the time when the measurements of the time-varying node `A1` switch from value 0 to 1. The time of this value switch is represented by the parameter `tswitch` in the code below. Note that the value `tswitch = 16` identifies the static intervention corresponding with no treatment initiation during follow-up in our example while the values 0 through 15 represent 16 distinct interventions representing increasingly delayed treatment initiation during follow-up.

First, we define the list of intervention nodes and their post-intervention distributions. Note that these distributions are indexed by the attribute `tswitch`, whose value is not yet defined:

```
`%+%` <- function(a, b) paste0(a, b)
Dstat <- lDAG
act_A1_tswitch <- node("A1",t = 0:(t.end), distr = "rbern",
                       prob = ifelse(t >= tswitch, 1, 0))
```

Second, we add the 17 static interventions to the `lDAG` object while defining the value of `tswitch` for each intervention:

```
tswitch_vec <- (0:t.end)
for (tswitch_i in tswitch_vec) {
  abar <- rep(0, length(tswitch_vec))
  abar[which(tswitch_vec >= tswitch_i)] <- 1
  Dstat <- Dstat + action("A1_ts"%+%tswitch_i,
                          nodes = act_A1_tswitch,
                          tswitch = tswitch_i,
                          abar = abar)
}
```

Note that in addition to the action attribute `tswitch`, each intervention is also indexed by an additional action attribute `abar` that also uniquely identifies the intervention and that represents the actual sequence of treatment decisions that defines the intervention, i.e., $\bar{a}(tswitch - 1) = 0$, $a(tswitch) = 1$, ...:

```
    A(Dstat)[["A1_ts3"]]

## [1] "Action: A1_ts3"
## [1] "ActionNodes: A1_0, A1_1, ... , A1_15, A1_16"
## [1] "ActionAttributes: "
## $tswitch
## [1] 3
##
## $abar
##  [1] 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The purpose of this additional action attribute `abar` will become clear when we illustrate the definition of target parameters defined by working MSMs based on these 17 static interventions in Section 4.7.2 (Example 7 of `set.targetMSM`).

Figure 5 shows the plot of the `DAG.action` object associated with the action named `"A1_ts3"` (with plotting restricted to nodes indexed by time points lower than or equal to 3 and with exclusion of the action attribute `"abar"` from the plot). Note that the intervention nodes are marked in red by default. The following code is used to generate Figure 5:

```
plotDAG(A(Dstat)[["A1_ts3"]], tmax = 3, xjitter = 0.3, yjitter = 0.02,
      edge_attrs = list(width = 0.5, arrow.width = 0.4, arrow.size = 0.8),
      vertex_attrs = list(size = 15, label.cex = 0.7), excludeattrs = "abar")
```

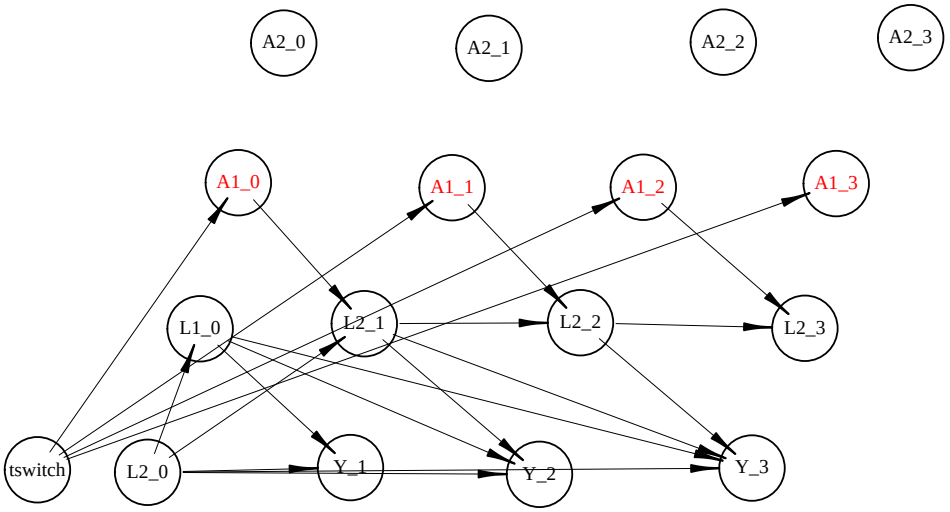

Figure 5: Graphical representation of the modified structural equation model resulting from a static intervention. Only the nodes indexed by time points lower than or equal to 3 are represented. The action attribute `abar` is also not represented.

4.4. Simulating full data (`sim`)

Simulating full data is accomplished by calling the function `sim` and specifying its arguments `DAG`, `actions` and `n` to indicate the causal model, interventions, and sample size of interest.

The full data can be simulated for all actions stored in the `DAG` object or a subset by setting the `actions` argument to the vector of the desired-action names.

Dynamic interventions

The example below shows how to use the `sim` function to simulate 100,000 observations for each of the two dynamic actions, `"A1_th0"` and `"A1_th1"`, defined in Section 4.3.1. The call to `sim` below produces a list of two named `data.frame` objects, where each `data.frame` object contains observations simulated from the same post-intervention distribution defined by one particular action only.

```
  Xdyn <- sim(Ddyn, actions = c("A1_th0", "A1_th1"), n = 10000, rndseed = 123)
  nrow(Xdyn[["A1_th0"]])

## [1] 10000

  nrow(Xdyn[["A1_th1"]])

## [1] 10000

  names(Xdyn)

## [1] "A1_th0" "A1_th1"
```

The default format of the output list generated by the `sim` function is easily understood by examining the first 15 columns of the first row of each `data.frame` object:

```
  Xdyn[["A1_th0"]][1, 1:15]

##   ID theta L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3
## 1  1     0    0    0    1    0   0    0    1    0   0    0    1    0   0

  Xdyn[["A1_th1"]][1, 1:15]

##   ID theta L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3
## 1  1     1    0    0    0    0   0    0    0    0   0    0    0    0   0
```

Static interventions

This example shows how to use the `sim` function to simulate 1,000 observations for each of the 17 static actions defined in Section 4.3.2. The call to the `sim` function below produces a list of named `data.frame` objects, where each `data.frame` object contains observations simulated from the same post-intervention distribution defined by one particular action only.

```
  Xstat <- sim(Dstat, actions = names(A(Dstat)), n = 1000, rndseed = 123)
  length(Xstat)

## [1] 17
```

```
  nrow(Xstat[["A1_ts3"]])

## [1] 1000
```

The default format of the output list generated by the `sim` function is easily understood by examining the first row of the `data.frame` object associated with the action `"A1_ts3"`:

```
  Xstat[["A1_ts3"]][1, ]

##   ID abar_0 abar_1 abar_2 abar_3 abar_4 abar_5 abar_6 abar_7 abar_8 abar_9 abar_10
## 1  1      0      0      0      1      1      1      1      1      1      1       1
##   abar_11 abar_12 abar_13 abar_14 abar_15 abar_16 tswitch L2_0 L1_0 A1_0 A2_0 Y_1 L2_1
## 1       1       1       1       1       1       1       3    0    0    0    0   0    0
##   A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3 L2_3 A1_3 A2_3 Y_4 L2_4 A1_4 A2_4 Y_5 L2_5 A1_5 A2_5
## 1    0    0   0    0    0    0   0    1    1    0   0    0    1    0   0    1    1    0
##   Y_6 L2_6 A1_6 A2_6 Y_7 L2_7 A1_7 A2_7 Y_8 L2_8 A1_8 A2_8 Y_9 L2_9 A1_9 A2_9 Y_10 L2_10
## 1   0    0    1    0   0    0    1    0   0    0    1    0   0    0    1    0    0     0
##   A1_10 A2_10 Y_11 L2_11 A1_11 A2_11 Y_12 L2_12 A1_12 A2_12 Y_13 L2_13 A1_13 A2_13 Y_14
## 1     1     0    0     0     1     0    0     0     1     0    0     0     1     0    0
##   L2_14 A1_14 A2_14 Y_15 L2_15 A1_15 A2_15 Y_16 L2_16 A1_16 A2_16
## 1     0     1     0    0     0     1     0    0     0     1     1
```

4.5. Converting a dataset from *wide* to *long* format (`DF.to.long`)

The specification of structural equation models based on time-varying nodes such as the one described in Section 4.1 allows for simulated (observed or full) data to be structured in either long or wide formats. Below, we illustrate these two alternatives. We note that, by default, simulated (observed or full) data from the `sim` function are stored in wide format. The data output format from the `sim` function can, however, be changed to the long format by setting the `wide` argument of the `sim` function to `FALSE` or, equivalently, by applying the function `DF.to.long` to an existing simulated dataset in wide format.

The following code demonstrates the default data formatting behavior of the `sim` function and how this behavior can be modified to generate data in the long format:

```
  Odat.wide <- sim(DAG = lDAG, n = 1000, wide = TRUE, rndseed = 123)
  Odat.wide[1:2, 1:18]

##   ID L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3 L2_3 A1_3 A2_3 Y_4
## 1  1    0    0    0    0   0    0    0    0   0    0    0    0   0    1    0    0   1
## 2  2    0    0    0    0   0    0    0    0   0    0    0    0   0    0    0    0   0

  Odat.long <- sim(DAG = lDAG, n = 1000, wide = FALSE, rndseed = 123)
  Odat.long[1:10, ]

##     ID L1 t L2 A1 A2  Y
## 1.0  1  0 0  0  0  0 NA
## 1.1  1  0 1  0  0  0  0
## 1.2  1  0 2  0  0  0  0
## 1.3  1  0 3  1  0  0  0
```

```
## 1.4  1  0 4 NA NA NA  1
## 2.0  2  0 0  0  0  0 NA
## 2.1  2  0 1  0  0  0  0
## 2.2  2  0 2  0  0  0  0
## 2.3  2  0 3  0  0  0  0
## 2.4  2  0 4  0  0  0  0
```

As with observed data, the default behavior of the `sim` function can be changed so that simulated full data are instead structured in long format:

```
lXdyn <- sim(Ddyn, actions = c("A1_th0", "A1_th1"), n = 10000, wide = FALSE, rndseed = 123)
head(lXdyn[["A1_th0"]], 5)
```

```
##     ID theta L1 t L2 A1 A2  Y
## 1.0  1     0  0 0  0  1  0 NA
## 1.1  1     0  0 1  0  1  0  0
## 1.2  1     0  0 2  0  1  0  0
## 1.3  1     0  0 3  0  1  0  0
## 1.4  1     0  0 4  0  1  0  0
```

The following example demonstrates how the function `DF.to.long` can be used to convert simulated (full or observed) data stored in wide format to long format:

```
Odat.long2 <- DF.to.long(Odat.wide)
all.equal(Odat.long, Odat.long2)
```

```
## [1] TRUE
```

4.6. Implementing imputation by *last time point value carried forward* (`doLTCF`)

As described in Sections 2.2 and 2.5, the default behavior of the `sim` function consists in setting all nodes that temporally follow an `EFU` node whose simulated value is 1 to missing (i.e., `NA`). The argument `LTCF` of the `sim` function can however be used to change this default behavior and impute some of these missing values with *last time point value carried forward* (LTCF). More specifically, only the missing values of time-varying nodes (i.e., those with non-missing `t` argument) that follow the end of follow-up event encoded by the `EFU` node specified by the `LTCF` argument will be imputed. Equivalently, one can use the function `doLTCF` to apply the same *last time point value carried forward* imputation to an existing simulated dataset obtained from the function `sim` that was called with its default imputation setting (i.e., with no `LTCF` argument). Illustration of the use of this LTCF imputation functionality is given in Section 4.7.2 (Example 1 of `set.targetMSM`).

The following code demonstrates the default data format of the `sim` function after an end of follow-up event and how this behavior can be modified to generate data with LTCF imputation by either using the `LTCF` argument of the `sim` function or by calling the `doLTCF` function.

```
  Odat.wide <- sim(DAG = lDAG, n = 1000, rndseed = 123)
  Odat.wide[c(11,76), 1:18]

##    ID L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3 L2_3 A1_3 A2_3 Y_4
## 11 11    1    0    1    0   1   NA   NA   NA  NA   NA   NA   NA  NA   NA   NA   NA  NA
## 76 76    0    0    0    0   0    1    0    0   1   NA   NA   NA  NA   NA   NA   NA  NA

  Odat.wideLTCF <- sim(DAG = lDAG, n = 1000, LTCF = "Y", rndseed = 123)
  Odat.wideLTCF[c(11,76), 1:18]

##    ID L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3 L2_3 A1_3 A2_3 Y_4
## 11 11    1    0    1    0   1    1    1    0   1    1    1    0   1    1    1    0   1
## 76 76    0    0    0    0   0    1    0    0   1    1    0    0   1    1    0    0   1
```

The code below demonstrates how to use the `doLTCF` function to perform LTCF imputation on already existing data simulated with the `sim` function based on its default non-imputation behavior:

```
  Odat.wideLTCF2 <- doLTCF(data = Odat.wide, LTCF = "Y")
  Odat.wideLTCF2[c(11,76), 1:18]

##    ID L2_0 L1_0 A1_0 A2_0 Y_1 L2_1 A1_1 A2_1 Y_2 L2_2 A1_2 A2_2 Y_3 L2_3 A1_3 A2_3 Y_4
## 11 11    1    0    1    0   1    1    1    0   1    1    1    0   1    1    1    0   1
## 76 76    0    0    0    0   0    1    0    0   1    1    0    0   1    1    0    0   1

  all.equal(Odat.wideLTCF, Odat.wideLTCF2)

## [1] TRUE
```

4.7. Defining and evaluating various causal target parameters

Causal parameters defined with `set.targetE`

Example 1. In the example below, we first define two causal target parameters as two vectors, each containing the expectations of the node `Y[t]`, for time points `t`=1, ..., 16, under the post-intervention distribution defined by one of the two dynamic interventions `"A1_th0"` and `"A1_th1"` defined in Section 4.3.1. Second, we evaluate these target parameters using the full data simulated previously in Section 4.4.1. Third, we map the resulting estimates of cumulative risks into estimates of survival probabilities. Fourth, we plot the corresponding two counterfactual survival curves using the **simcausal** routine `plotSurvEst` as shown in Figure 6. Finally, we note that Figure 6 replicates the simulation study results reported in Figure 4 of Neugebauer *et al.* (2014).

```
  Ddyn <- set.targetE(Ddyn, outcome = "Y", t = 1:16, param = "A1_th1")
  surv_th1 <- 1 - eval.target(Ddyn, data = Xdyn)$res
  Ddyn <- set.targetE(Ddyn, outcome = "Y", t = 1:16, param = "A1_th0");
  surv_th0 <- 1 - eval.target(Ddyn, data = Xdyn)$res
```

```
plotSurvEst(surv = list(d_theta1 = surv_th1, d_theta0 = surv_th0),
            xindx = 1:17,
            ylab = "Counterfactual survival for each intervention",
            ylim = c(0.75,1.0))
```

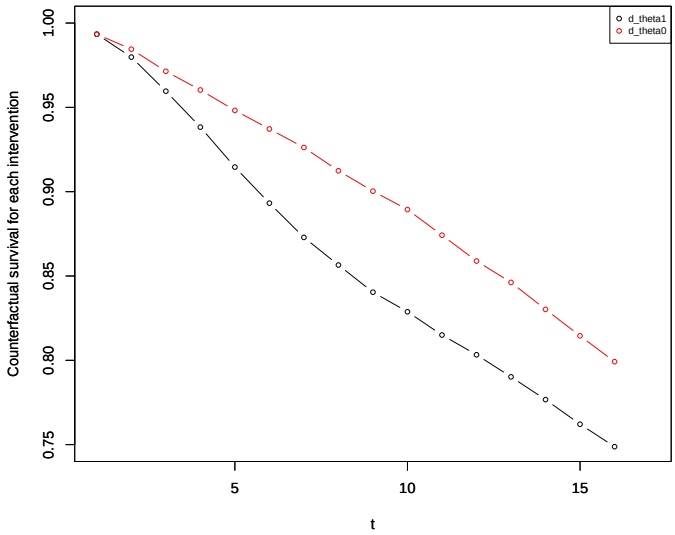

Figure 6: Estimates of the true survival curves under the two dynamic interventions

Example 2. In the example below, we first define the causal target parameters as the ATE on the additive scale (cumulative risk differences) and the ATE on the multiplicative scale (cumulative risk ratios), for the two dynamic interventions (`"A1_th1"` and `"A1_th0"`) defined in Section 4.3.1 at time point `t = 12`. Second, we evaluate these target parameters using the previously simulated full data from Section 4.4.1.

ATE on the additive scale:

```
Ddyn <- set.targetE(Ddyn, outcome = "Y", t = 12, param = "A1_th1-A1_th0")
(psi <- eval.target(Ddyn, data = Xdyn)$res)

## Diff_Y_12
##    0.0556
```

ATE on the multiplicative scale:

```
Ddyn <- set.targetE(Ddyn, outcome = "Y", t = 12, param = "A1_th0/A1_th1")
eval.target(Ddyn, data = Xdyn)$res

## Ratio_Y_12
##   0.717336
```

We also note that the above result for the ATE on the additive scale (`psi`=0.0556) replicates the simulation result reported for ψ in Section 5.1 and Figure 4 of Neugebauer *et al.* (2014), where ψ was defined as the difference between the cumulative risks of failure at $t_0 = 12$ for the two dynamic interventions d_1 and d_0.

Causal parameters defined with `set.targetMSM`

In Section 3.5.2, we described the arguments of the function `set.targetMSM` that the user must specify to define MSM target causal parameters. They include the specification of the argument `form` which encodes the working MSM formula. This formula can only be a function of the time index `t`, action attributes that uniquely identify each intervention of interest, and baseline nodes (defined as nodes that precede the earliest intervention node). Both baseline nodes that are measurements of time-varying nodes and time-varying action attributes must be referenced in the R expression passed to the `form` argument within the wrapping syntax `S(...)` as illustrated in several examples below.

Example 1. Working dynamic MSM for survival probabilities over time. Here, we illustrate the evaluation of the counterfactual survival curves $E(Y_{d_\theta}(t))$ for $t = 1, \ldots, 16$ under the dynamic interventions d_θ for $\theta = 0, 1$ introduced in Section 4.3.1 using the following pooled working logistic MSM:

$$\text{expit}\left(\alpha_0 + \alpha_1\theta + \alpha_2 t + \alpha_3 t\theta\right),$$

where the true values of the coefficients $(\alpha_i, i = 0, \ldots, 3)$ define the target parameters of interest. First, we define these target parameters:

```
msm.form <- "Y ~ theta + t + I(theta*t)"
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, form = msm.form,
                      family = "binomial", hazard = FALSE)
```

Note that when the outcome is a time-varying node of type `EFU`, the argument `hazard = FALSE` indicates that the working MSM of interest is a model for survival probabilities. The argument `family = "binomial"` indicates that the working model is a logistic model. Second, we evaluate the coefficients of the working model:

```
MSMres <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres$coef

##  (Intercept)        theta            t I(theta * t)
##  -3.81485291   0.57306753   0.16134206  -0.01343472
```

We also note that no previously simulated full data were passed as argument to the function `eval.target` above. Instead, the sample size argument `n` was specified and the routine will thus first sample `n = 10,000` observations from each of the two post-intervention distributions before fitting the working MSM with these full data to derive estimates of the true coefficient values. Alternatively, the user could have passed the previously simulated full data. Note however that in this case, the user must either simulate full data by calling the `sim` function with the argument `LTCF = "Y"` or convert the previously simulated full data with the *last time point value carried forward* imputation function `doLTCF`. Both approaches are described

in Section 4.6 and the latter approach is demonstrated in the example below, where we first impute the `EFU` outcome `Y` in the previously simulated full data `Xdyn`.

```
XdynLTCF <- lapply(Xdyn, doLTCF, LTCF = "Y")
eval.target(Ddyn, data = XdynLTCF)$coef

##  (Intercept)        theta            t I(theta * t)
##  -3.81485291   0.57306753   0.16134206  -0.01343472
```

The resulting coefficient estimates can be mapped into estimates of the two counterfactual survival curves and plotted as shown in Figure 7 using the `plotSurvEst` function:

```
surv_th0 <- 1 - predict(MSMres$m, newdata = data.frame(theta = rep(0, 16), t = 1:16),
                        type = "response")
surv_th1 <- 1 - predict(MSMres$m, newdata = data.frame(theta = rep(1, 16), t = 1:16),
                        type = "response")
plotSurvEst(surv = list(MSM_theta1 = surv_th1, MSM_theta0 = surv_th0),
            xindx = 1:16,
            ylab = "MSM Survival, P(T>t)",
            ylim = c(0.75, 1.0))
```

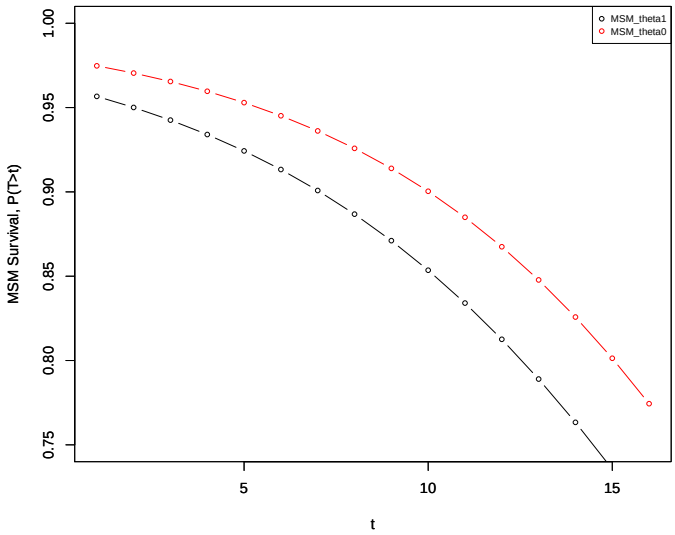

Figure 7: Survival curve estimates evaluated based on working MSM 1

Example 2. More complex working dynamic MSM for survival probabilities over time. The previous example can be modified to illustrate the evaluation of the survival curves of interest with a more flexible (i.e., more non-parametric) working MSM as follows:

```
msm.form <- "Y ~ theta + t + I(t^2) + I(t^3) + I(t^4) + I(t^5) + I(t*theta) + I(t^2*theta) +
             I(t^3*theta) + I(t^4*theta) + I(t^5*theta)"
```

```
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, formula = msm.form,
                      family = "binomial", hazard = FALSE)
MSMres2 <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres2$coef
```

```
##    (Intercept)          theta              t         I(t^2)         I(t^3)         I(t^4)
##  -6.194377e+00  -1.309635e-01   1.439451e+00  -2.512705e-01   2.432731e-02  -1.164989e-03
##         I(t^5)   I(t * theta) I(t^2 * theta) I(t^3 * theta) I(t^4 * theta) I(t^5 * theta)
##   2.175838e-05   1.972163e-01  -5.036459e-03  -2.443627e-03   2.104174e-04  -4.983988e-06
```

```
surv_th0 <- 1 - predict(MSMres2$m, newdata = data.frame(theta = rep(0, 16), t = 1:16),
                        type = "response")
surv_th1 <- 1 - predict(MSMres2$m, newdata = data.frame(theta = rep(1, 16), t = 1:16),
                        type = "response")
plotSurvEst(surv = list(MSM_theta1 = surv_th1, MSM_theta0 = surv_th0),
            xindx = 1:16,
            ylab = "MSM Survival, P(T>t)",
            ylim = c(0.75, 1.0))
```

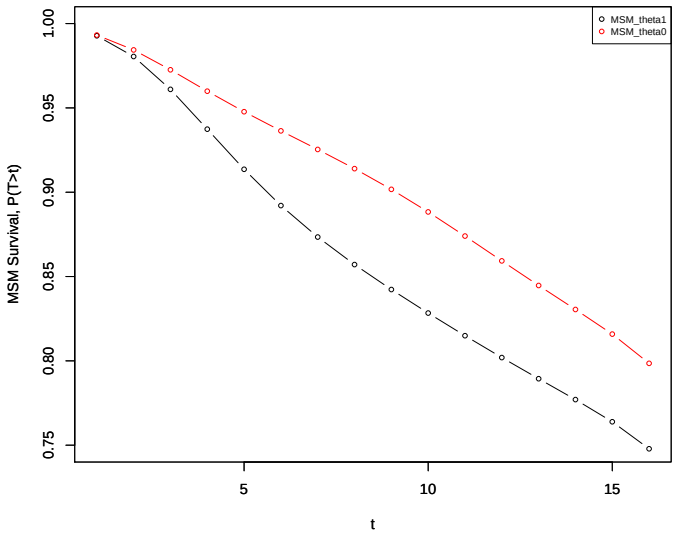

Figure 8: Survival curve estimates evaluated based on working MSM 2

The resulting estimates of the survival curves shown in Figure 8 are indeed visually closer to the true survival curves reported in Figure 4 of Neugebauer *et al.* (2014).

Example 3. Saturated dynamic MSM for survival probabilities over time. Here, we further modify the working model formula by specifying a saturated MSM to directly replicate the results reported in Figure 4 of Neugebauer *et al.* (2014) that are based on a non-parametric MSM approach:

```
msm.form <- "Y ~ theta + as.factor(t) + as.factor(t):theta "
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, formula = msm.form,
                      family = "binomial", hazard = FALSE)
MSMres3 <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres3$coef
```

```
##          (Intercept)                theta        as.factor(t)2        as.factor(t)3
##        -5.014064e+00        -1.217855e-12         8.627698e-01         1.488732e+00
##        as.factor(t)4        as.factor(t)5        as.factor(t)6        as.factor(t)7
##         1.828169e+00         2.106888e+00         2.311122e+00         2.484332e+00
##        as.factor(t)8        as.factor(t)9       as.factor(t)10       as.factor(t)11
##         2.670766e+00         2.813501e+00         2.929437e+00         3.075448e+00
##       as.factor(t)12       as.factor(t)13       as.factor(t)14       as.factor(t)15
##         3.207880e+00         3.308961e+00         3.427018e+00         3.534544e+00
##       as.factor(t)16  theta:as.factor(t)2  theta:as.factor(t)3  theta:as.factor(t)4
##         3.632762e+00         2.696280e-01         3.576449e-01         4.641088e-01
##  theta:as.factor(t)5  theta:as.factor(t)6  theta:as.factor(t)7  theta:as.factor(t)8
##         5.360346e-01         5.790890e-01         6.028846e-01         5.567782e-01
##  theta:as.factor(t)9 theta:as.factor(t)10 theta:as.factor(t)11 theta:as.factor(t)12
##         5.393551e-01         5.074806e-01         4.557836e-01         3.991351e-01
## theta:as.factor(t)13 theta:as.factor(t)14 theta:as.factor(t)15 theta:as.factor(t)16
##         3.789713e-01         3.405073e-01         3.152922e-01         2.890793e-01
```

```
surv_th0 <- 1 - predict(MSMres3$m, newdata = data.frame(theta = rep(0, 16), t = 1:16),
                        type = "response")
surv_th1 <- 1 - predict(MSMres3$m, newdata = data.frame(theta = rep(1, 16), t = 1:16),
                        type = "response")
plotSurvEst(surv = list(MSM_theta1 = surv_th1, MSM_theta0 = surv_th0),
            xindx = 1:16,
            ylab = "MSM Survival, P(T>t)",
            ylim = c(0.75, 1.0))
```

The resulting estimates of the survival curves shown in Figure 9 approximate that reported in Figure 4 of Neugebauer *et al.* (2014). Differences can be explained by the relative small sample size of the full data based on which the following plot is based (`n=1,000` here vs. `n=1,000,000` in the simulation study from Neugebauer *et al.* (2014)).

Example 4. Working dynamic MSM for discrete-time hazards over time. Here, we illustrate the evaluation of discrete-time hazards $E(Y_{d_\theta}(t))|Y_{d_\theta}(t-1)=0)$, for $t=1,\ldots,16$ under the dynamic interventions d_θ, for $\theta=0,1$ introduced in Section 4.3.1 using the following pooled working logistic MSM:

$$\text{expit}\left(\alpha_0+\alpha_1\theta+\alpha_2 t+\alpha_3 t\theta\right),$$

where the true values of coefficients $(\alpha_i, i=0,\ldots,3)$ define the target parameters of interest. First, we define these target parameters:

```
msm.form <- "Y ~ theta + t + I(theta*t)"
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, form = msm.form,
                      family = "binomial", hazard = TRUE)
```

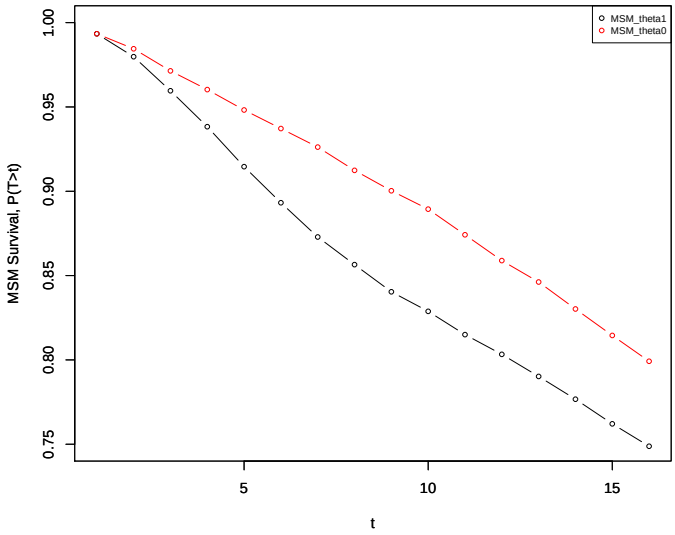

Figure 9: Survival curve estimates evaluated based on working MSM 3

Note that when the outcome is a time-varying node of type `EFU`, the argument `hazard = TRUE` indicates that the working MSM of interest is now a model for hazards.

Second, we evaluate the coefficients of the working model:

```
MSMres <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres$coef
```

```
##  (Intercept)        theta            t I(theta * t)
##  -4.71857385   0.69819739   0.05069993  -0.04871408
```

Note that no previously simulated full data were passed as argument to the call to `eval.target` above. Instead, the sample size argument `n` was specified and the routine will thus first sample `n=10,000` observations from each of the two post-intervention distributions before fitting the working MSM with these full data to derive estimates of the true coefficient values. Alternatively, the user could have passed the previously simulated full data using the argument `data = Xdyn`.

The resulting coefficient estimates can be mapped into estimates of the two counterfactual hazard curves and plotted as shown in Figure 10 using the `plotSurvEst` function:

```
h_th0 <- predict(MSMres$m, newdata = data.frame(theta = rep(0, 16), t = 1:16),
                 type = "response")
h_th1 <- predict(MSMres$m, newdata = data.frame(theta = rep(1, 16), t = 1:16),
                 type = "response")
plotSurvEst(surv = list(MSM_theta1 = h_th1, MSM_theta0 = h_th0),
            xindx = 1:16,
            ylab = "MSM hazard function",
            ylim = c(0.0, 0.03))
```

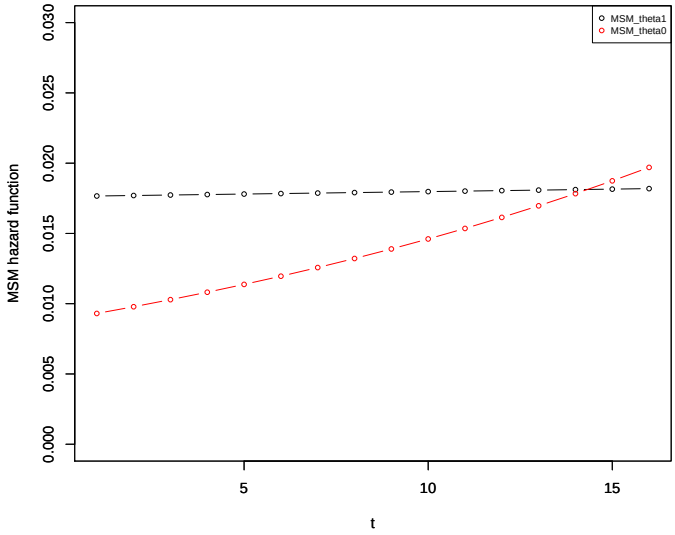

Figure 10: Hazard estimates evaluated based on working MSM 4

Alternatively, the resulting coefficient estimates can also be mapped into estimates of the two counterfactual survival curves and plotted as shown in Figure 11 using the `plotSurvEst` function:

```
Surv_h_th0 <- cumprod(1 - h_th0)
Surv_h_th1 <- cumprod(1 - h_th1)
plotSurvEst(surv = list(MSM_theta1 = Surv_h_th1, MSM_theta0 = Surv_h_th0),
            xindx = 1:16,
            ylab = "Survival P(T>t) derived from MSM hazard",
            ylim = c(0.75, 1.0))
```

Example 5. Working dynamic MSM to evaluate effect modification by a time-independent covariate. The previous example can be modified to illustrate the evaluation of effect modification by a baseline covariate through the inclusion of an interaction term between θ and L_1 in the working logistic MSM for $E(Y_{d_\theta}(t))|Y_{d_\theta}(t-1)=0, L_1)$:

$$\text{expit}\left(\alpha_0 + \alpha_1\theta + \alpha_2 t + \alpha_3 t\theta + \alpha_4\theta L_1\right), \text{ for } t = 1,\ldots,16 \text{ and } \theta = 0, 1,$$

where the true values of coefficients α_i, for $i = 0,\ldots,4$ define the target parameters of interest. First, we define and estimate these target parameters:

```
msm.form_sum <- "Y ~ theta + t + I(theta*t) + I(theta*L1)"
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, form = msm.form_sum,
                      family = "binomial", hazard = TRUE)
MSMres <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres$coef

##   (Intercept)         theta             t  I(theta * t) I(theta * L1)
##   -4.71857385    0.54547251    0.05069993   -0.04617993    0.91876760
```

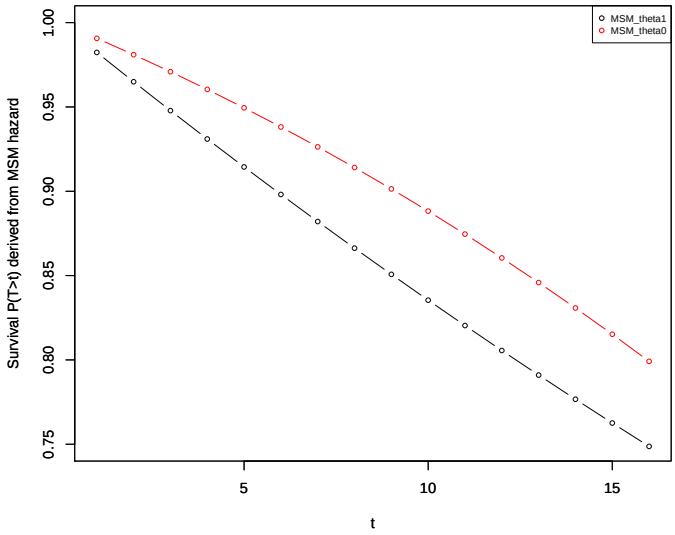

Figure 11: Survival curve estimates evaluated based on working MSM 4

Second, we map the coefficient estimates into counterfactual survival curves within subgroups defined by the baseline covariate `L1` levels:

```
get_haz <- function(thetaval, L1val) {
  predict(MSMres$m, newdata = data.frame(theta = rep(thetaval, 16), t = 1:16, L1 = L1val),
          type = "response")
}
Sth0L1_0 <- cumprod(1 - get_haz(thetaval = 0, L1val = 0))
Sth1L1_0 <- cumprod(1 - get_haz(thetaval = 1, L1val = 0))
Sth0L1_1 <- cumprod(1 - get_haz(thetaval = 0, L1val = 1))
Sth1L1_1 <- cumprod(1 - get_haz(thetaval = 1, L1val = 1))
```

Third, we plot the resulting survival curves for each subgroup of interest as shown in Figure 12:

```
par(mfrow = c(1,2))
plotSurvEst(surv = list(MSM_theta1 = Sth1L1_0, MSM_theta0 = Sth0L1_0),
            xindx = 1:16,
            ylab = "Survival P(T>t), for L1=0",
            ylim = c(0.5, 1.0))
plotSurvEst(surv = list(MSM_theta1 = Sth1L1_1, MSM_theta0 = Sth0L1_1),
            xindx = 1:16,
            ylab = "Survival P(T>t), for L1=1",
            ylim = c(0.5, 1.0))
```

Example 6. Working dynamic MSM to evaluate effect modification by the baseline measurement of a time-dependent covariate. Here, the previous example is modified to illustrate the evaluation of effect modification by the baseline measurement of the time-varying node `L2` using the following working logistic MSM:

$$\text{expit}\left(\alpha_0 + \alpha_1\theta + \alpha_2 t + \alpha_3 t\theta + \alpha_4\theta L_2(0)\right),$$

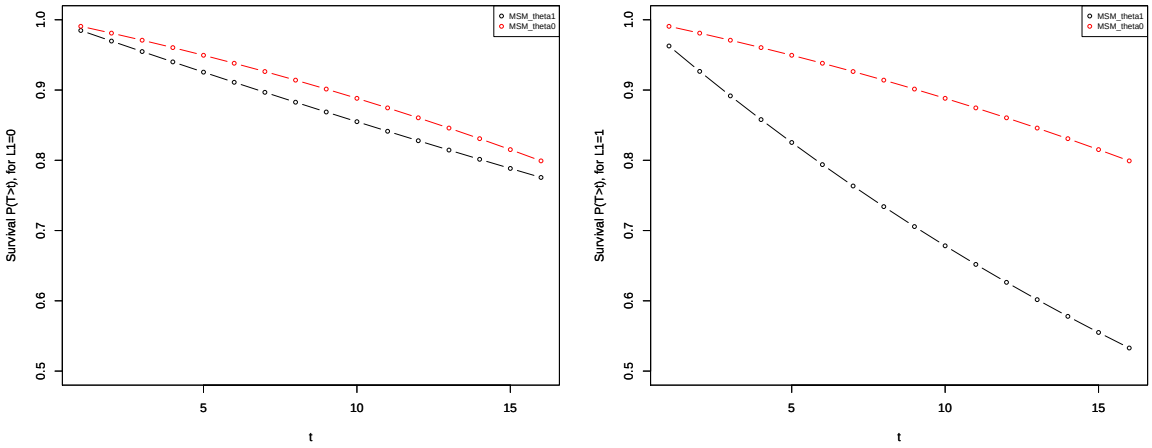

Figure 12: Survival curve estimates evaluated based on working MSM 5

where the true values of coefficients α_i for $i = 0, \ldots, 4$ define the target parameters of interest. First, we define and estimate these target parameters:

```
msm.form.correct <- "Y ~ theta + t + I(theta*t) + I(theta * S(L2[0]))"
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = 1:16, form = msm.form.correct,
                      family = "binomial", hazard = TRUE)
MSMres.correct <- eval.target(Ddyn, n = 10000, rndseed = 123)
MSMres.correct$coef
```

```
##         (Intercept)               theta                   t        I(theta * t)
##         -4.71857385          0.67293187          0.05069993          -0.04862217
## I(theta * S(L2[0]))
##          0.46000951
```

Note that the baseline measurement of the time-varying node `L2` must be referenced within the `S(...)` in the working MSM formula.

Example 7. Working static MSM for survival probabilities over time. Here, we illustrate the evaluation of discrete-time hazards $E(Y_{\bar{a}}(t))|Y_{\bar{a}}(t-1) = 0)$, for $t = 1, ..., 16$ under the 17 static interventions introduced in Section 4.3.2 using the following pooled working logistic MSM:

$$expit\big(\alpha_0 + \alpha_1 t + \alpha_2 \frac{1}{t} \sum_{j=0}^{t-1} a(j) + \alpha_3 \sum_{j=0}^{t-1} a(j)\big),$$

where we use the notation $\bar{a} = (a(0), a(1), ..., a(16))$ to denote the 17 static intervention regimens on the time- varying treatment node `A1`. Note that the time-varying action attribute `abar` introduced in Section 4.3.2 directly encodes the 17 treatment regimens values $\bar{a}$ referenced in the MSM working model above. To evaluate the target parameters α_j above, for $j = 0, ..., 3$, we first simulate full data for the 17 static interventions of interest as follows:

```
  Xts <- sim(Dstat, actions = names(A(Dstat)), n = 1000, rndseed = 123)
```

Second, we define the target parameters and estimate them using the full data just simulated as follows:

```
  msm.form_1 <- "Y ~  t + S(mean(abar[0:(t-1)])) + I(t*S(mean(abar[0:(t-1)])))"
  Dstat <- set.targetMSM(Dstat, outcome = "Y", t = 1:16, form = msm.form_1,
                         family = "binomial", hazard = TRUE)
  MSMres <- eval.target(Dstat, data = Xts)
  MSMres$coef

##                     (Intercept)                               t
##                      -3.6345057                       0.1001448
##        S(mean(abar[0:(t - 1)])) I(t * S(mean(abar[0:(t - 1)])))
##                      -1.2086860                      -0.1165954
```

Note that the working MSM formulas can reference arbitrary summary measures (functions) of time-varying action attributes such as `abar`. The square-bracket `'['` syntax can then be used to identify specific elements of the time-varying action attributes in the same way it can be used in node formulas to reference particular measurements of time-varying nodes. For example, the term `sum(abar[0:t])` indicates a summation over the elements of the action attribute `abar` indexed by time points lower than or equal to value `t` and the syntax `S(abar[max(0, t - 2)])` creates a summary measure representing time-lagged values of `abar` that are equal to `abar[0]` if `t`<2 and to `abar[t-2]` if `t`≥ 2. Note also that references to time-varying action attributes in the working MSM formula must be wrapped within a call to the `S(...)` function, e.g., `Y`$\sim$`t + S(mean(abar[0:t]))`.

The `eval.target` function returns a list with the following named attributes: the working MSM fit returned by a `glm` function call (`msm`), the coefficient estimates (`coef`), the mapping (`S.msm.map`) of the formula terms defined by expressions enclosed within the `S(...)` function into the corresponding variable names in the design matrix that was used to implement the regression, and the design matrix (`df_long`) stored as a list of `data.table` objects from the R package **data.table** (Dowle *et al.* 2014). Each of these `data.table` objects contains full data indexed by a particular intervention. These full data are stored in long format with possibly additional new columns representing terms in the working MSM formula defined by expressions enclosed with the `S()` function. The design matrix can be derived by row binding these `data.table` objects.

```
  names(MSMres)

## [1] "msm"       "coef"      "S.msm.map" "hazard"    "call"      "df_long"

  MSMres$S.msm.map

##             S_exprs_vec  XMSMterms
## 1 mean(abar[0:(t - 1)]) XMSMterm.1

  names(MSMres$df_long)
```

```
##  [1] "A1_ts0"  "A1_ts1"  "A1_ts2"  "A1_ts3"  "A1_ts4"  "A1_ts5"  "A1_ts6"  "A1_ts7"
##  [9] "A1_ts8"  "A1_ts9"  "A1_ts10" "A1_ts11" "A1_ts12" "A1_ts13" "A1_ts14" "A1_ts15"
## [17] "A1_ts16"
```

```
  MSMres$df_long[["A1_ts2"]]
```

```
##           ID tswitch L1  t abar L2 A1 A2 Y XMSMterm.1
##      1:    1       2  0  1    0  0  0  0 0  0.0000000
##      2:    1       2  0  2    1  0  1  0 0  0.0000000
##      3:    1       2  0  3    1  1  1  0 0  0.3333333
##      4:    1       2  0  4    1  0  1  0 0  0.5000000
##      5:    1       2  0  5    1 NA NA NA 1  0.6000000
##     ---
## 14351: 1000       2  0 12    1  0  1  0 0  0.8333333
## 14352: 1000       2  0 13    1  0  1  0 0  0.8461538
## 14353: 1000       2  0 14    1  0  1  0 0  0.8571429
## 14354: 1000       2  0 15    1  0  1  0 0  0.8666667
## 14355: 1000       2  0 16    1  0  1  1 0  0.8750000
```

Finally, we plot the resulting counterfactual survival curve estimates using the function `survbyMSMterm` (source code provided in the appendix) as shown in Figure 13.

```
  survMSMh_wS <- survbyMSMterm(MSMres = MSMres, t_vec = 1:16,
                               MSMtermName = "mean(abar[0:(t - 1)])")
```

```
## [1] "MSMtermName used"
## [1] "XMSMterm.1"
```

```
  print(plotsurvbyMSMterm(survMSMh_wS))
```

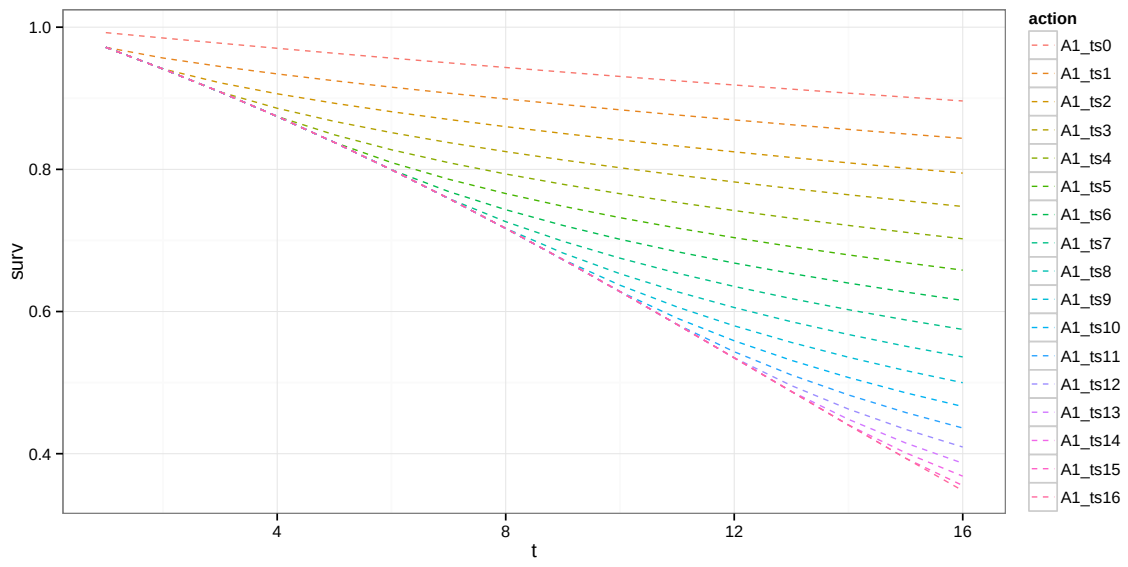

Figure 13: Survival curve estimates evaluated based on working MSM 7

5. Replication study of the comparative performances of two estimators

In this section, we demonstrate how the **simcausal** package can be used to conduct a complex but transparent and reproducible simulation study to compare the finite sample properties (bias an relative efficiency) of two causal effect estimators. Specifically, we aim to replicate the results of the simulation study described by Neugebauer *et al.* (2014) that compared targeted minimum loss based estimation (TMLE) and inverse probability weighting (IPW) estimation of a causal risk difference defined by two dynamic treatment regimens (see ATE in *Example 2* of `set.targetE` in Section 4.7.1). We carried out this replication study using simulation protocol 3 described in **Section 5.3.** of Neugebauer *et al.* (2014) and compared the bias and relative efficiency of TMLE and IPW estimation using the same performance metrics as that reported in **Table 6** of Neugebauer *et al.* (2014). The observed data were thus generated using the following slightly modified version of the structural equation model from Section 4.1 to match the data generating distribution described in protocol 3 of **Section 5.3.** of Neugebauer *et al.* (2014):

```
t.end <- 12
D <- DAG.empty()
D <- D +
        node("L2", t = 0, distr = "rbern",
             prob = 0.05) +
        node("m1L2", t = 0, distr = "rconst",
             const = 1 - L2[0]) +
        node("L1", t = 0, distr = "rbern",
             prob = ifelse(L2[0] == 1, 0.8, 0.3)) +
        node("A1", t = 0, distr = "rbern",
             prob = ifelse(L1[0] == 1 & L2[0] == 0, 0.5,
                      ifelse(L1[0] == 0 & L2[0] == 0, 0.2,
                        ifelse(L1[0] == 1 & L2[0] == 1, 0.8, 0.5)))) +
        node("A2", t = 0, distr = "rbern", prob = 0, EFU = TRUE)

D <- D +
        node("Y", t = 1:t.end, distr = "rbern",
             prob = plogis(-7 + 3 * L1[0] + 5 * L2[t-1] +
                           0.1 * sum(I(L2[0:(t-1)] == rep(0, t)))),
             EFU = TRUE) +
        node("L2", t = 1:t.end, distr = "rbern",
             prob = ifelse(A1[t-1] == 1, 0.1,
                      ifelse(L2[t-1] == 1, 0.9, min(1,0.1 + t/16)))) +
        node("m1L2", t = 1:t.end, distr = "rconst", const = 1-L2[t]) +
        node("A1", t = 1:t.end, distr = "rbern",
             prob = ifelse(A1[t-1] == 1, 1,
                      ifelse(L1[0] == 1 & L2[t] == 0, 0.4,
                        ifelse(L1[0] == 0 & L2[t] == 0, 0.2,
                          ifelse(L1[0] == 1 & L2[t] == 1, 0.8, 0.6))))) +
        node("A2", t = 1:t.end, distr = "rbern",
             prob = {if(t == .(t.end)) {1} else {0}},
             EFU = TRUE)

lDAG <- set.DAG(D)
Ddyn <- lDAG
```

The target causal parameter ψ_0 in this replication study is defined as the difference between

the cumulative risks of failure at $t_0 = 11$ under the two dynamic regimes d_0 and d_1 introduced in Section 4.3.1, i.e., $\psi_0 = P(Y_{d_1}(t_0 + 1) = 1) - P(Y_{d_0}(t_0 + 1) = 1)$:

```
t0 <- 12
Ddyn <- set.targetE(Ddyn, outcome = "Y", t = (1:t0), param = "A1_th1-A1_th0")

getNP.truetarget <- function() {
  resNP <- eval.target(Ddyn, n = 150000, rndseed = 123)$res
  return(as.vector(resNP[paste0("Diff_Y_", t0)]))
}

f1name <- "vignette_dat/repstudy1_psi0.t0.NP.Rdata"
if (file.exists(f1name)) {
  load(f1name)
} else {
  psi0.t0.NP <- getNP.truetarget()
  save(list = "psi0.t0.NP", file = f1name)
}
psi0.t0.NP
```

```
## [1] 0.1079467
```

The above estimate of the true value of ψ_0 closely matches the value reported in Figure 10 of Neugebauer *et al.* (2014).

We note that the target parameter ψ_0 can also be defined and thus evaluated with the following saturated MSM:

$$\text{logit}(P(Y_{d_\theta}(t+1)=1)) = \alpha_0 + \alpha_0^1\theta + \sum_{k=1}^{t_0}\alpha_k^0 I(t=k) + \sum_{k=1}^{t_0}\alpha_k^1\theta I(t=k),$$

for $t = 0, 1, \ldots, t_0$ and $\theta \in \{0, 1\}$. Indeed, the risk difference ψ_0 can then be derived from the coefficients of this MSM as follows:

$$\psi_0 = \text{expit}(\alpha_0 + \alpha_0^1 + \alpha_{t_0}^0 + \alpha_{t_0}^1) - \text{expit}(\alpha_0 + \alpha_{t_0}^0). \tag{1}$$

Evaluation of the target parameter ψ_0 can thus be implemented through fitting of a saturated MSM as shown below:

```
MSM_RD_t <- function(resMSM, t) {
  invlogit <- function(x) 1 / (1 + exp(-x))
  Riskth0 <- invlogit(resMSM["(Intercept)"] + resMSM[paste0("as.factor(t)",t)])
  Riskth1 <- invlogit(resMSM["(Intercept)"] + resMSM[paste0("as.factor(t)",t)] +
                        resMSM["theta"] + resMSM[paste0("theta:as.factor(t)",t)])
  return(as.vector(Riskth1-Riskth0))
}

msm.form <- "Y ~ theta + as.factor(t) + as.factor(t):theta "
Ddyn <- set.targetMSM(Ddyn, outcome = "Y", t = (1:t0), formula = msm.form,
                      family = "binomial", hazard = FALSE)

getMSM.truetarget <- function() {
  resMSM <- eval.target(Ddyn, n = 150000, rndseed = 123)$coef
```

```
    return(as.vector(MSM_RD_t(resMSM = resMSM, t = t0)))
  }

  f2name <- "vignette_dat/repstudy1_psi0.t0.MSM.Rdata"
  if (file.exists(f2name)) {
    load(f2name)
  } else {
    psi0.t0.MSM <- getMSM.truetarget()
    save(list = "psi0.t0.MSM", file = f2name)
  }

  psi0.t0.MSM

## [1] 0.1079467

  all.equal(psi0.t0.NP, psi0.t0.MSM)

## [1] TRUE
```

To evaluate the bias and relative efficiency of TMLE and IPW estimation of the risk difference ψ_0 with observed data, we generated $1,000$ observed datasets, each with sample size $50,000$. With each simulated observed data set, the coefficients of the saturated MSM were then estimated by TMLE and IPW estimation using the `ltmeMSM` function from the **ltmle** R package (Schwab *et al.* 2014) as shown in the appendix. For an in-depth description of TMLE, we refer to Petersen *et al.* (2014) and the **ltmle** package manual. The IPW and TMLE estimates of ψ_0 were then derived from the estimated MSM coefficients α using formula (1). As in Neugebauer *et al.* (2014), both TMLE and IPW estimators of ψ_0 were derived using a correctly specified model for the treatment mechanism but also a misspecified model (covariate `L2[0]` missing from the logistic model for the propensity scores).

We report our simulation results in Table 1. For each estimator, we report the empirical mean of the $1,000$ estimates (ψ_n), corresponding bias and the empirical standard deviation of the $1,000$ estimates (σ_{emp}). We also report the relative efficiency for IPW vs. TMLE, evaluated as the ratio of their respective empirical standard deviations. The function `simrun_ltmleMSM()` (source code provided in the appendix) can be used to generate the results presented in Table 1.

Our TMLE results match those reported in **Table 6** by Neugebauer *et al.* (2014), while our IPW results differ from those reported in Neugebauer *et al.* (2014). Specifically, in our simulations the IPW was shown to have a smaller empirical standard deviation, compared to Neugebauer *et al.* (2014), resulting in a smaller reported relative efficiency of 12%, compared to 39% relative efficiency reported by Neugebauer *et al.* (2014). We note though that the IPW estimator implemented by Neugebauer *et al.* (2014) is different from the IPW estimator implemented in the **ltmle** package. The latter IPW estimator is defined based on the fitting of a separate treatment mechanism model for each time point. Additionally, the IPW estimator for ψ_0 implemented with the **ltmle** package is based on a saturated MSM for the counterfactual *survival* functions, whereas the IPW estimator implemented in Neugebauer *et al.* (2014) was constructed using the survival probability estimates derived from a saturated MSM for the

counterfactual *hazard* functions. To our knowledge, such an approach is not automated currently in an existing R package. Therefore, the IPW estimation performance in this simulation study should not be expected to match that reported in Neugebauer *et al.* (2014).

Estimator	ψ_n	Bias	σ_{emp}	$\sigma_{emp}^{IPTW}/\sigma_{emp}^{TMLE}$
Results from this replication study				
TMLE (correct model)	0.108	-0.0002	0.0055	1.12
IPTW (correct model)	0.108	-0.0001	0.0061	
Results reported by Neugebauer et al. (2014)				
TMLE (correct model)	0.108	0.0010	0.0060	1.392
IPTW (correct model)	0.108	0.0011	0.0078	
Results from this replication study				
TMLE (incorrect model 1)	0.112	0.0041	0.0067	1.91
IPTW (incorrect model 1)	0.094	-0.0144	0.0128	
Results reported by Neugebauer et al. (2014)				
TMLE (incorrect model 1)	0.109	0.0019	0.0074	
IPTW (incorrect model 1)	0.182	0.0750	0.0107	

Table 1: Replication of the results from simulation protocol 3 reported in Table 6 of Neugebauer *et al.* (2014) based on two models for estimating the treatment mechanism: 1) a correctly specified model and 2) a misspecified model missing a term for the time-dependent variable. ψ_n - mean point estimates over 1,000 simulated data sets; σ_{emp} - empirical standard deviation (SD) of point estimates over 1,000 simulated data sets; $\sigma_{emp}^{IPTW}/\sigma_{emp}^{TMLE}$ - the relative efficiency measured by the ratio of the empirical SDs associated with the IPW and TMLE point estimates.

6. Replication study of the impact of misspecification of propensity score models

In this section, we demonstrate how the **simcausal** package can be used to replicate a simulation study from Lefebvre *et al.* (2008). Specifically, we replicate the results first reported in Tables II and IV of that paper. We first specify the observed data generating distribution using the two structural equation models corresponding with Scenarios 1 and 3 described in Lefebvre *et al.* (2008). Second, for each scenario, we evaluate the true values of the coefficients of the MSM using full data and compare them to those reported by Lefebvre *et al.* (2008). Finally for each scenario, we implement the same IPW estimators of these MSM coefficients and evaluate their performances using the same two metrics (bias and mean squared error) as in Lefebvre *et al.* (2008). Each IPW estimator is defined by a distinct model for the propensity scores. Our replication results are reported in Tables 2 and 4, and we show the simulations results as they were reported by Lefebvre *et al.* (2008) in Tables 3 and 5.

To carry out the simulation study, we first define a new distribution function `rbivNorm` for simulating observations from a bivariate normal distribution with a user-specified mean vector (specified by the argument `mu`) and a user-specified covariance matrix (specified by the arguments `var1`, `var2`, and `rho` to represent the diagonal and off-diagonal scalars, respectively). This new distribution function is based on Cholesky decomposition of the covariance matrix and independent observations simulated from the standard normal distribution which are provided by the input argument `norms`. The argument `whichbiv` indicates whether the function should return independent observations from the first or second element of the bivariate normal vector.

```
rbivNorm <- function(n, whichbiv, norms, mu, var1 = 1, var2 = 1, rho = 0.7) {
  whichbiv <- whichbiv[1]; var1 <- var1[1]; var2 <- var2[1]; rho <- rho[1]
  sigma <- matrix(c(var1, rho, rho, var2), nrow = 2)
  Scol <- chol(sigma)[, whichbiv]
  bivX <- (Scol[1] * norms[, 1] + Scol[2] * norms[, 2]) + mu
  bivX
}
```

Second, using this distribution function, we define the structural equation model specified for data simulation according to Scenario 1 in Lefebvre *et al.* (2008).

```
`%+%` <- function(a, b) paste0(a, b)
Lnames <- c("LO1", "LO2", "LO3", "LC1")
D <- DAG.empty()

for (Lname in Lnames) {
  D <- D +
          node(Lname%+%".norm1", distr = "rnorm", mean = 0, sd = 1) +
          node(Lname%+%".norm2", distr = "rnorm", mean = 0, sd = 1)
}

D <- D +
        node("LO1", t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
             norms = c(LO1.norm1, LO1.norm2),
```

```
            mu = 0) +
      node("LO2", t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
            norms = c(LO2.norm1, LO2.norm2),
            mu = 0) +
      node("LO3", t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
            norms = c(LO3.norm1, LO3.norm2),
            mu = 0) +
      node("LC1", t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
            norms = c(LC1.norm1, LC1.norm2),
            mu = {if (t == 0) {0} else {-0.30 * A[t-1]}}) +
      node("alpha", t = 0:1, distr = "rconst",
            const = {if(t == 0) {log(0.6)} else {log(1.0)}}) +
      node("A", t = 0:1, distr = "rbern",
            prob = plogis(alpha[t] +
                          log(5)*LC1[t] + {if(t == 0) {0} else {log(5)*A[t-1]}})) +
      node("Y", t = 1, distr = "rnorm",
            mean = (0.98 * LO1[t] + 0.58 * LO2[t] + 0.33 * LO3[t] +
                    0.98 * LC1[t] - 0.37 * A[t]),
            sd = 1)

DAGO.sc1 <- set.DAG(D)
```

Third, we define the target parameter as the coefficients β_1 and β_2 of the following correctly specified marginal structural model:

$$E[Y_{a(0),a(1)}] = \beta_0 + \beta_1 a(0) + \beta_2 a(1),$$

defined by the following four possible static and deterministic interventions $(a(0), a(1))$ on the treatment process $(A(0), A(1))$: $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$.

```
defAct <- function (Dact) {
  act.At <- node("A", t = 0:1, distr = "rbern", prob = abar[t])
  Dact <- Dact +
              action("A00", nodes = act.At, abar = c(0, 0)) +
              action("A10", nodes = act.At, abar = c(1, 0)) +
              action("A01", nodes = act.At, abar = c(0, 1)) +
              action("A11", nodes = act.At, abar = c(1, 1))
  return(Dact)
}

Dact.sc1 <- defAct(DAGO.sc1)
msm.form <- "Y ~ S(abar[0]) + S(abar[1])"
Dact.sc1 <- set.targetMSM(Dact.sc1, outcome = "Y", t = 1,
                          form = msm.form, family = "gaussian")
```

Fourth, we evaluate the true values of these MSM coefficients using the `eval.target` function and note that our results closely match the true value of the MSM coefficients reported in Table II of Lefebvre *et al.* (2008):

```
repstudy2.sc1.truetarget <- function() {
  trueMSMreps.sc1 <- NULL
  reptrue <- 50
  for (i in (1:reptrue)) {
    res.sc1.i <- eval.target(Dact.sc1, n = 500000)$coef
```

```
      trueMSMreps.sc1 <- rbind(trueMSMreps.sc1, res.sc1.i)
    }
    return(trueMSMreps.sc1)
  }

  f1name <- "vignette_dat/trueMSMreps.sc1.Rdata"
  if (file.exists(f1name)) {
    load(f1name)
  } else {
    trueMSMreps.sc1 <- repstudy2.sc3.truetarget()
    save(list = "trueMSMreps.sc1", file = f1name)
  }
  (trueMSM.sc1 <- apply(trueMSMreps.sc1, 2, mean))

##   (Intercept)    S(abar[0])    S(abar[1])
##  0.0001540635 -0.2941187264 -0.3700397969
```

Note that the true values of the MSM coefficients above were obtained from the averages of coefficient estimates obtained from several simulated full data sets. This approach was implemented to avoid the memory limitation that can be encountered when trying to simulate a single very large full data set.

Finally, we replicate the IPW estimation results for Scenario 1 presented originally in Table II of Lefebvre *et al.* (2008) using the source R code provided in the appendix. To estimate the propensity scores $P(A(0)|L(0))$ and $P(A(1)|A(0), L(1))$ that define each of the three IPW estimators considered, we used the same three models presented in Table I of Lefebvre *et al.* (2008). For the three sample sizes $N = 300$; $1,000$; and $10,000$, we report the bias of each IPW estimator, multiplied by 10 (*Bias*10*) and the mean-squared error, also multiplied by 10 (*MSE*10*) in Table 2. We note that our results closely match those in Table 3 which were originally reported in Lefebvre *et al.* (2008).

Covariates in $P(A\|L)$	N	A(0) Bias*10	A(0) MSE*10	A(1) Bias*10	A(1) MSE*10
Confounder(s) only	300	0.600	1.791	0.726	1.760
	1000	0.327	0.764	0.382	0.695
	10000	0.041	0.138	0.063	0.129
Confounder(s) &	300	0.546	1.757	0.814	1.461
risk factors	1000	0.303	0.712	0.364	0.668
	10000	0.053	0.132	0.043	0.130

Table 2: Replication of the simulation results from Lefebvre *et al.* (2008) for Scenario 1.

Next, using the same approach described above, we replicate the simulation results for Scenario 3 reported in Table IV of Lefebvre *et al.* (2008). We start by defining the structural equation model specified for data simulation according to Scenario 3 in Lefebvre *et al.* (2008) as follows:

Covariates in $P(A\|L)$	N	A(0) Bias*10	A(0) MSE*10	A(1) Bias*10	A(1) MSE*10
Lefebvre et al.: Confounder(s) only	300	0.768	1.761	0.889	1.728
	1000	0.265	0.761	0.312	0.723
	10000	0.057	0.146	0.086	0.120
Lefebvre et al.: Confounder(s) & risk factors	300	0.757	1.642	0.836	1.505
	1000	0.283	0.718	0.330	0.638
	10000	0.056	0.139	0.081	0.114

Table 3: Simulation results for Scenario 1 as reported in Table II of Lefebvre *et al.* (2008).

```
`%+%` <- function(a, b) paste0(a, b)
Lnames <- c("LO1", "LO2", "LO3", "LE1", "LE2", "LE3", "LC1", "LC2", "LC3")
D <- DAG.empty()

for (Lname in Lnames) {
  D <- D +
          node(Lname%+%".norm1", distr = "rnorm") +
          node(Lname%+%".norm2", distr = "rnorm")
}
```

```
coefAi <- c(-0.10, -0.20, -0.30)
sdLNi <- c(sqrt(1), sqrt(5), sqrt(10))

for (i in (1:3)) {
  D <- D +
          node("LO"%+%i, t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
                mu = 0,
                params = list(norms = "c(LO"%+%i%+%".norm1, LO"%+%i%+%".norm2)")) +
          node("LE"%+%i, t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
                mu = 0, var1 = 1, var2 = 1, rho = 0.7,
                params = list(norms = "c(LE"%+%i%+%".norm1, LE"%+%i%+%".norm2)")) +
          node("LC"%+%i, t = 0:1, distr = "rbivNorm", whichbiv = t + 1,
                mu = {if (t == 0) {0} else {.(coefAi[i]) * A[t-1]}},
                params = list(norms = "c(LC"%+%i%+%".norm1, LC"%+%i%+%".norm2)")) +
          node("LN"%+%i, t = 0:1, distr = "rnorm",
                mean = 0, sd = .(sdLNi[i]))
}

D <- D +
        node("alpha", t = 0:1, distr = "rconst",
              const = {if(t == 0) {log(0.6)} else {log(1.0)}}) +
        node("A", t = 0:1, distr = "rbern",
              prob = plogis(alpha[t] +
                            log(5) * LC1[t] + log(2) * LC2[t] + log(1.5) * LC3[t] +
                            log(5) * LE1[t] + log(2) * LE2[t] + log(1.5) * LE3[t] +
                            {if (t == 0) {0} else {log(5) * A[t-1]}})) +
        node("Y", t = 1, distr = "rnorm",
              mean = 0.98 * LO1[t] + 0.58 * LO2[t] + 0.33 * LO3[t] +
                      0.98 * LC1[t] + 0.58 * LC2[t] + 0.33 * LC3[t] - 0.39 * A[t],
              sd = 1)
```

```
DAGO.sc3 <- set.DAG(D)
```

Similar to Scenario 1, we then define the same four actions on the new `DAG` object before defining and evaluating the causal target parameter of interest. We note that our results match the true value of the MSM coefficients reported in Table IV of Lefebvre *et al.* (2008):

```
Dact.sc3 <- defAct(DAGO.sc3)
msm.form <- "Y ~ S(abar[0]) + S(abar[1])"
Dact.sc3 <- set.targetMSM(Dact.sc3, outcome = "Y", t = 1,
                          form = msm.form, family = "gaussian")

repstudy2.sc3.truetarget <- function() {
  trueMSMreps.sc3 <- NULL
  reptrue <- 50
  for (i in (1:reptrue)) {
    res.sc3.i <- eval.target(Dact.sc3, n = 500000)$coef
    trueMSMreps.sc3 <- rbind(trueMSMreps.sc3, res.sc3.i)
  }
  return(trueMSMreps.sc3)
}

f2name <- "vignette_dat/trueMSMreps.sc3.Rdata"
if (file.exists(f2name)) {
  load(f2name)
} else {
  trueMSMreps.sc3 <- repstudy2.sc3.truetarget()
  save(list = "trueMSMreps.sc3", file = f2name)
}
(trueMSM.sc3 <- apply(trueMSMreps.sc3, 2, mean))

##   (Intercept)    S(abar[0])    S(abar[1])
## -0.0004548424 -0.3125245583 -0.3897569393
```

Finally, using the R code provided in the appendix, we replicate in Table 4 the IPW estimation results for Scenario 3 presented originally in Table IV of Lefebvre *et al.* (2008). We note that our simulation results closely match those in Table 5 which were originally reported by Lefebvre *et al.* (2008).

Covariates in $P(A\|L)$	N	A(0) Bias*10	A(0) MSE*10	A(1) Bias*10	A(1) MSE*10
Confounder(s) only	300	-0.133	1.236	0.110	1.110
	1000	-0.278	0.398	-0.104	0.346
	10000	-0.353	0.055	-0.178	0.044
Confounder(s) &	300	-0.166	1.143	0.179	0.860
risk factors	1000	-0.293	0.360	-0.116	0.283
	10000	-0.356	0.050	-0.173	0.035
Confounder(s) &	300	1.285	3.900	2.153	3.998
IVs	1000	0.883	2.010	1.243	2.041
	10000	0.461	0.603	0.434	0.622
Confounder(s),	300	1.289	3.868	1.992	3.637
IVs & risk factors	1000	0.984	1.934	1.267	1.951
	10000	0.446	0.605	0.481	0.625
Mis-specified	300	2.675	3.252	5.384	5.358
	1000	2.644	1.758	5.221	3.805
	10000	2.422	0.817	4.994	2.720
Full Model	300	1.294	4.156	2.117	3.892
	1000	0.929	2.132	1.172	1.990
	10000	0.403	0.614	0.442	0.613

Table 4: Replication of the simulation results from Lefebvre *et al.* (2008) for Scenario 3.

Covariates in $P(A\|L)$	N	A(0) Bias*10	A(0) MSE*10	A(1) Bias*10	A(1) MSE*10
Lefebvre et al.: Confounder(s) only	300	-0.080	1.170	0.099	1.155
	1000	-0.371	0.385	-0.035	0.331
	10000	-0.368	0.056	-0.203	0.043
Lefebvre et al.: Confounder(s) &	300	-0.110	1.092	0.112	0.865
risk factors	1000	-0.330	0.340	-0.108	0.245
	10000	-0.378	0.051	-0.207	0.037
Lefebvre et al.: Confounder(s) &	300	1.611	3.538	2.069	3.841
IVs	1000	0.824	2.063	1.245	2.188
	10000	0.241	0.684	0.379	0.622
Lefebvre et al.: Confounder(s),	300	1.600	3.477	2.143	3.598
IVs & risk factors	1000	0.867	2.053	1.170	2.043
	10000	0.235	0.676	0.372	0.625
Lefebvre et al.: Mis-specified	300	3.146	3.326	5.591	5.494
	1000	2.460	1.700	5.258	3.851
	10000	2.364	0.832	4.943	2.705
Lefebvre et al.: Full Model	300	1.524	3.648	2.221	3.907
	1000	0.878	2.099	1.185	2.099
	10000	0.240	0.679	0.377	0.630

Table 5: Simulation results for Scenario 3 as reported in Table IV of Lefebvre *et al.* (2008).

7. Discussion

We demonstrated that the **simcausal** R package is a flexible tool that facilitates the conduct of transparent and reproducible simulation studies to evaluate causal inference methodologies. The package allows the user to simulate complex longitudinal data structures based on structural equation models using a novel interface which allows concise and intuitive expression of complex functional dependencies for a large number of nodes. The package allows the user to specify and simulate counterfactual data under various interventions (e.g., static, dynamic, deterministic, or stochastic). These interventions may represent exposures to treatment regimens, the occurrence or non- occurrence of right-censoring events, or of specific monitoring events. The package also enables the computation of a selected set of user-specified features of the distribution of the counterfactual data that represent common causal target parameters, such as, treatment-specific means, average treatment effects and coefficients from working marginal structural models. In addition, the package provides a flexible graphical component that produces plots of directed acyclic graphs for observed or post-intervention data generating distributions.

We demonstrated the functionality of the package with a single time point intervention simulation study in Section 3 and a complex multiple time point simulation study in Section 4. We also showed two real-world applications of the **simcausal** package in Sections 5 and 6, first, by replicating results of the simulation study by Neugebauer *et al.* (2014, 2015) that evaluated the comparative performance of two estimation procedures, and second, by replicating results of the simulation study by Lefebvre *et al.* (2008) that evaluated the impact of model misspecification of the treatment mechanism on IPW inferences about MSM coefficients.

Finally, we acknowledge that the **simcausal** package is in the early stages of its development and that implementation of additional functionalities in future releases of the package should further expand its utility for methods research. Among such possible improvements is the evaluation of additional causal parameters, e.g., the average treatment effect on the treated (Holland 1986; Imbens 2004; Shpitser and Pearl 2009), survivorship causal effects (Joffe *et al.* 2007; Greene *et al.* 2013) and direct/indirect effects (Pearl 2001; Petersen *et al.* 2006; VanderWeele 2009; VanderWeele and Vansteelandt 2014; Hafeman and VanderWeele 2011). Additionally, future versions of the **simcausal** package may allow simulation of non-iid observations, to study causal inference methodologies to analyze data resulting from experiments with more complex sampling methodologies, e.g., survey-based sampled data (Särndal *et al.* 2003) or network-based sampled (dependent) data (Eckles *et al.* 2014).

Acknowledgments

FUNDING ACKNOWLEDGEMENT: This study was partially funded through internal operational funds provided by the Kaiser Permanente Center for Effectiveness & Safety Research (CESR). This work was also partially supported through a Patient-Centered Outcomes Research Institute (PCORI) Award (ME-1403-12506) and an NIH grant (R01 AI074345-07).

DISCLAIMER: All statements in this report, including its findings and conclusions, are solely those of the authors and do not necessarily represent the views of the Patient-Centered Outcomes Research Institute (PCORI), its Board of Governors or Methodology Committee.

References

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. doi:10.1109/ICCSN.2010.34. URL http://igraph.org.

Dowle M, Short T, Lianoglou S, with contributions from R Saporta AS, Antonyan E (2014). *data.table: Extension of data.frame*. R package version 1.9.4, URL http://CRAN.R-project.org/package=data.table.

Eckles D, Karrer B, Ugander J (2014). "Design and analysis of experiments in networks: Reducing bias from interference." *arXiv preprint arXiv:1404.7530*.

Greene T, Joffe M, Hu B, Li L, Boucher K (2013). "The balanced survivor average causal effect." *The international journal of biostatistics*, **9**(2), 291–306.

Hafeman DM, VanderWeele TJ (2011). "Alternative assumptions for the identification of direct and indirect effects." *Epidemiology*, **22**(6), 753–764.

Holland PW (1986). "Statistics and Causal Inference." *Journal of the American Statistical Association*, **81**(396), 945–960. doi:10.1080/01621459.1986.10478354.

Imbens GW (2004). "Nonparametric estimation of average treatment effects under exogeneity: A review." *Review of Economics and statistics*, **86**(1), 4–29.

Joffe MM, Small D, Hsu CY, Others (2007). "Defining and estimating intervention effects for groups that will develop an auxiliary outcome." *Statistical Science*, **22**(1), 74–97.

Lefebvre G, Delaney JA, Platt RW (2008). "Impact of mis-specification of the treatment model on estimates from a marginal structural model." *Stat Med*, **27**(18), 3629–3642.

Neugebauer R, Schmittdiel JA, van der Laan MJ (2014). "Targeted learning in real-world comparative effectiveness research with time-varying interventions." *Stat Med*, **33**(14), 2480–2520.

Neugebauer R, Schmittdiel JA, Zhu Z, Rassen JA, Seeger JD, Schneeweiss S (2015). "High-dimensional propensity score algorithm in comparative effectiveness research with time-varying interventions." *Statistics in Medicine*, **34**(5), 753–781. ISSN 1097-0258. doi:10.1002/sim.6377. URL http://dx.doi.org/10.1002/sim.6377.

Neugebauer R, van der Laan M (2007). "Nonparametric causal effects based on marginal structural models." *Journal of Statistical Planning and Inference*, **137**(2), 419–434.

Pearl J (1995). "Causal diagrams for empirical research." *Biometrika*, **82**(4), 669–688.

Pearl J (2001). "Direct and Indirect Effects." In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pp. 411–420. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-800-1. URL http://dl.acm.org/citation.cfm?id=2074022.2074073.

Pearl J (2009). *Causality: Models, Reasoning and Inference*. 2nd edition. Cambridge University Press, New York, NY, USA. ISBN 052189560X, 9780521895606.

Pearl J (2010). "An introduction to causal inference." *The international journal of biostatistics*, **6**(2).

Petersen M, Schwab J, van der Laan M, Gruber S, Blaser N, Schomaker M (2014). "Targeted Maximum Likelihood Estimation for Dynamic and Static Longitudinal Marginal Structural Working Models." *Journal of Causal Inference*, **2**(2), 39. URL http://ideas.repec.org/a/bpj/causin/v2y2014i2p39n1.html.

Petersen ML, Sinisi SE, van der Laan MJ (2006). "Estimation of direct causal effects." *Epidemiology*, **17**(3), 276–284.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Särndal CE, Swensson B, Wretman J (2003). *Model assisted survey sampling*. Springer Science & Business Media.

Schwab J, Lendle S, Petersen M, van der Laan M (2014). *ltmle: Longitudinal Targeted Maximum Likelihood Estimation*. R package version 0.9.3-1, URL http://CRAN.R-project.org/package=ltmle.

Shpitser I, Pearl J (2009). "Effects of Treatment on the Treated: Identification and Generalization." In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 514–521. AUAI Press, Montreal, Quebec.

Sofrygin O, van der Laan MJ, Neugebauer R (2015). *simcausal: Simulating Longitudinal Data with Causal Inference Applications*. R package version 0.1.

VanderWeele TJ (2009). "Marginal structural models for the estimation of direct and indirect effects." *Epidemiology*, **20**(1), 18–26.

VanderWeele TJ, Vansteelandt S (2014). "Mediation Analysis with Multiple Mediators." *Epidemiologic methods*, **2**(1), 95–115. doi:10.1515/em-2012-0010. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4287269/.

Source code

.1. Source code for obtaining and plotting MSM survival curves

```
# get MSM survival predictions from the full data.table in long format (melted) by time (t_vec), and by the MSM term (MSMtermName)
# predictions from the estimated msm model (based on observational data) can be obtained by passing estimated msm model, est.msm
# given vector of t (t_vec), results of MSM target.eval and an MSM term get survival table by action
survbyMSMterm <- function(MSMres, t_vec, MSMtermName, use_actions=NULL, est.msm=NULL) {
  library(data.table)
  # look up MSMtermName in MSMterm map, if exists -> use the new name, if doesn't exist use MSMtermName
  if (!is.null(MSMres$S.msm.map)) {
    mapS_exprs <- as.character(MSMres$S.msm.map[,"S_exprs_vec"])
    XMSMterms <- as.character(MSMres$S.msm.map[,"XMSMterms"])
    map_idx <- which(mapS_exprs%in%MSMtermName)
    XMSMtermName <- XMSMterms[map_idx]
    if (!is.null(XMSMtermName)&&length(XMSMtermName)>0) {
      MSMtermName <- XMSMtermName
    }
  }
  print("MSMtermName used"); print(MSMtermName)
  t_dt <- data.table(t=as.integer(t_vec)); setkey(t_dt, t)
  get_predict <- function(actname) {
    # setkey(MSMres$df_long[[actname]], t)
    setkeyv(MSMres$df_long[[actname]], c("t", MSMtermName))
    # MSMterm_vals <- as.numeric(MSMres$df_long[[actname]][t_dt, mult="first"][[MSMtermName]])
    # print("MSMterm_vals"); print(MSMterm_vals)
    MSMterm_vals <- as.numeric(MSMres$df_long[[actname]][t_dt, mult="last"][[MSMtermName]])
    # print("MSMterm_vals last"); print(MSMterm_vals)
    newdata=data.frame(t=t_vec, MSMterm_vals=MSMterm_vals)
    colnames(newdata) <- c("t", MSMtermName)
    # print("newdata"); print(newdata)
    if (!is.null(est.msm)) {
      pred <- predict(est.msm, newdata=newdata, type="response")
    } else {
      pred <- predict(MSMres$m, newdata=newdata, type="response")
    }
    return(data.frame(t=t_vec, pred=pred))
  }
  action_names <- names(MSMres$df_long)
  if (!is.null(use_actions)) {
    action_names <- action_names[action_names%in%use_actions]
  }
  surv <- lapply(action_names, function(actname) {
                               res <- get_predict(actname)
                               if (MSMres$hazard) {
                                 res$surv <- cumprod(1-res$pred)
                               } else {
                                 res$surv <- 1-res$pred
                               }
                               res$pred <- NULL
                               res$action <- actname
                               res
                             })
  names(surv) <- names(MSMres$df_long)
  surv_melt <- do.call('rbind', surv)
  surv_melt$action <- factor(surv_melt$action, levels=unique(surv_melt$action), ordered=TRUE)
  surv_melt
}
plotsurvbyMSMterm <- function(surv_melt_dat) {
  library(ggplot2)
  f_ggplot_surv_wS <- ggplot(data= surv_melt_dat, aes(x=t, y=surv)) +
                      geom_line(aes(group = action, color = action), size=.4, linetype="dashed") +
                      theme_bw()

}
plotsurvbyMSMterm_facet <- function(surv_melt_dat1, surv_melt_dat2, msm_names=NULL) {
  library(ggplot2)
  if (is.null(msm_names)) {
    msm_names <- c("MSM1", "MSM2")
  }
  surv_melt_dat1$MSM <- msm_names[1]
  surv_melt_dat2$MSM <- msm_names[2]

  surv_melt_dat <- rbind(surv_melt_dat1,surv_melt_dat2)
  f_ggplot_surv_wS <- ggplot(data= surv_melt_dat, aes(x=t, y=surv)) +
                      geom_line(aes(group = action, color = action), size=.4, linetype="dashed") +
                      theme_bw() +
                      facet_wrap( ~ MSM)
}
```

.2. Source code for replicating the simulation study in Neugebauer et al., 2014

Source code for the function that creates inputs for the `ltmleMSM` *function of the* ***ltmle*** *package (Schwab* et al. *2014).*

```
# @param DAG Object specifying the directed acyclic graph for the observed data,
  # must have a well-defined MSM target parameter (\code{set.target.MSM()})
# @param obs_df Simulated observational data
# @param Aname Generic names of the treatment nodes (can be time-varying)
# @param Cname Generic names of the censoring nodes (can be time-varying)
# @param Lnames Generic names of the time-varying covariates (can be time-varying)
# @param tvec Vector of time points for Y nodes
# @param actions Which actions (regimens) should be used in estimation from the observed simulated data.
  # If NULL then all actions that were defined in DAG will be considered.
# @param package Character vector for R package name to use for estimation. Currently only "ltmle" is implemented.
# @param fun Character name for R function name to employ for estimation. Currently only "ltmleMSM" is implemented.
# @param ... Additional named arguments that will be passed on to ltmleMSM function
est.targetMSM <- function(DAG, obs_df, Aname="A", Cname="C", Lnames, Ytvec, ACLtvec, actions=NULL, package="ltmle", fun="ltmleMSM", ...) {
  outnodes <- attr(DAG, "target")$outnodes
  param_name <- attr(DAG, "target")$param_name
  if (is.null(outnodes$t)) stop("estimation is only implemented for longitudinal data with t defined")
  if (!param_name%in%"MSM") stop("estimation is only implemented for MSM target parameters")
  if (is.null(actions)) {
    message("actions argument underfined, using all available actions")
    actions <- A(DAG)
  }
  # all time points actually used in the observed data
  t_all <- attr(obs_df, "tvals")
  tvec <- outnodes$t
  t_sel <- ACLtvec
  # ltmle allows for pooling Y's over smaller subset of t's (for example t=(2:8))
  # in this case summary measures HAVE TO MATCH the dimension of finYnodes, not t_sel
  # currently this is not supported, thus, if tvec is a subset of t_sel this will cause an error
  finYnodes <- outnodes$gen_name%+%"_"%+%Ytvec
  Ynodes <- outnodes$gen_name%+%"_"%+%Ytvec
  Anodes <- Aname%+%"_"%+%ACLtvec
  Cnodes <- Cname%+%"_"%+%ACLtvec
  Lnodes <- t(sapply(Lnames, function(Lname) Lname%+%"_"%+%ACLtvec[-1]))
  Lnodes <- as.vector(matrix(Lnodes, nrow=1, ncol=ncol(Lnodes)*length(Lnames), byrow=FALSE))
  Nobs <- nrow(obs_df)
  #-----------------------------------------------------------
  # getting MSM params
  #-----------------------------------------------------------
  params.MSM <- attr(DAG, "target")$params.MSM
  working.msm <- params.MSM$form
  msm.family <- params.MSM$family
  if (params.MSM$hazard) stop("ltmleMSM cannot estimate hazard MSMs...")
  # the number of attributes and their dimensionality have to match between different actions
  n_attrs <- length(attr(actions[[1]], "attnames"))
  #-----------------------------------------------------------
  # define the final ltmle arrays
  regimens_arr <- array(dim = c(Nobs, length(ACLtvec), length(actions)))
  summeas_arr <- array(dim = c(length(actions), (n_attrs+1), length(Ytvec)))
  # loop over actions (regimes) creating counterfactual mtx of A's for each action:
  for (action_idx in seq(actions)) {
      # I) CREATE COUNTERFACTUAL TREATMENTS &
      # II) CREATE summary.measure that describes each attribute by time + regimen
      #-----------------------------------------------------------
      # needs to assign observed treatments and replace the action timepoints with counterfactuals
      A_mtx_act <- as.matrix(obs_df[,Anodes])
      #-----------------------------------------------------------
      action <- actions[[action_idx]]
       # action-spec. time-points
      t_act <- as.integer(attr(action, "acttimes"))
      # action-spec. attribute names
      attnames <- attr(action, "attnames")
      # list of action-spec. attributes
      attrs <- attr(action, "attrs")
      # time points for which we need to evaluate the counterfactual treatment assignment as determined by action:
      #-----------------------------------------------------------
      # Action t's need always be the same subset of t_sel (outcome-based times), otherwise we are in big trouble
      t_act_idx <- which(t_sel%in%t_act)
      t_chg <- t_sel[t_act_idx]
      # modify only A's which are defined in this action out of all Anodes
      As_chg <- Anodes[t_act_idx]
      #-----------------------------------------------------------
      # creates summary measure array that is of dimension (length(t_chg)) - time-points only defined for this action
      # which t's are in the final pooled MSM => need to save the summary measures only for these ts
      t_vec_idx <- which(t_act%in%tvec)
      summeas_attr <- matrix(nrow=length(attnames), ncol=length(tvec))
      #-----------------------------------------------------------
      # extract values of terms in MSM formula: get all attribute values from +action(...,attrs)
      #-----------------------------------------------------------
      obs_df_attr <- obs_df # add all action attributes to the observed data
      for (attr_idx in seq(attnames)) { # self-contained loop # grab values of the attributes, # loop over all attributes
        if (length(attrs[[attnames[attr_idx]]])>1) {
```

```
          attr_i <- attnames[attr_idx]%+%"_"%+%t_chg
          val_attr_i <- attrs[[attnames[attr_idx]]][t_act_idx]
        } else {
          attr_i <- attnames[attr_idx]
          val_attr_i <- attrs[[attnames[attr_idx]]]
        }
        # summary measures, for each action/measure
        summeas_attr[attr_idx,] <- matrix(val_attr_i, nrow=1, ncol=length(t_chg))[,t_vec_idx]
        # observed data values of the attribute
        df_attr_i <- matrix(val_attr_i, nrow=Nobs, ncol=length(val_attr_i), byrow=TRUE)
        # create the combined data.frame (attrs + O.dat)
        colnames(df_attr_i) <- attr_i; obs_df_attr <- cbind(data.frame(df_attr_i), obs_df_attr)
      } # end of loop
      summeas_attr <- rbind(summeas_attr, t_chg[t_vec_idx])
      rownames(summeas_attr) <- c(attnames, "t")
      #----------------------------------------------------------
      # add action specific summary measures to the full array
      summeas_arr[action_idx, , ] <- summeas_attr
      dimnames(summeas_arr)[[2]] <- rownames(summeas_attr)
      #----------------------------------------------------------
      # GENERATING A MATRIX OF COUNTERFACTUAL TREATMENTS:
      for (Achange in As_chg) { # for each A defined in the action, evaluate its value applied to the observed data
        cur.node <- action[As_chg][[Achange]]
        t <- cur.node$t
        newAval <- with(obs_df_attr, {  # for static no need to sample from a distr
          ANCHOR_VARS_OBSDF <- TRUE
          simcausal:::eval_nodeform(as.character(cur.node$dist_params$prob), cur.node)$evaled_expr
        })

        if (length(newAval)==1) {
          newA <- rep(newAval, Nobs)
        } else {
          newA <- newAval
        }
        A_mtx_act[,which(Anodes%in%Achange)] <- newA
      }
      # Result matrix A_mtx_act has all treatments that were defined in that action replaced with
      # their counterfactual values
      #----------------------------------------------------------
      # add action specific summary measures to the full array
      regimens_arr[, , action_idx] <- A_mtx_act
      #----------------------------------------------------------
  }
  list(regimens_arr=regimens_arr, summeas_arr=summeas_arr)
}
```

Source code for setting up some of the parameters of the `ltmleMSM` *function of the* ***ltmle*** *package (Schwab* et al. *2014).*

```
times <- c(0:(t.end-1))
gforms <- c("A1_0 ~ L1_0 + L2_0","A2_0 ~ L1_0")
timesm0 <- times[which(times > 0)]
# correctly specified g:
gforms <- c("A1_0 ~ L1_0 + L2_0","A2_0 ~ L1_0")
gformm0 <- as.vector(sapply(timesm0, function(t)
             c("A1_"%+%t%+%" ~ A1_"%+%(t-1)%+%" + L1_0"%+%" + L2_"%+%t%+%" + I(L2_"%+%t%+%"*A1_"%+%(t-1)%+%")+ I(L1_0*A1_"%+%(t-1)%+%")",
               "A2_"%+%t%+%" ~ L1_0")))
gforms <- c(gforms, gformm0)
# mis-specified g (no TV covar L2):
gforms_miss <- c("A1_0 ~ L1_0","A2_0 ~ L1_0")
gformm0_miss <- as.vector(sapply(timesm0, function(t)
                           c("A1_"%+%t%+%" ~ L1_0*A1_"%+%(t-1),
                             "A2_"%+%t%+%" ~ L1_0")))
gforms_miss <- c(gforms_miss, gformm0_miss)
Qformallt <- "Q.kplus1 ~ L1_0"
Lterms <- function(var, tlast){
  tstr <- c(0:tlast)
  strout <- paste(var%+%"_"%+%tstr, collapse = " + ")
  return(strout)
}
tY <- (0:11)
Ynames <- paste("Y_"%+%c(tY+1))
Qforms <- unlist(lapply(tY, function(t)  {
                 a <- Qformallt%+%" + I("%+%Lterms("m1L2",t)%+%") + "%+%Lterms("L2",t)
                 return(a)
  }))
names(Qforms) <- Ynames
survivalOutcome <- TRUE
stratify_Qg <- TRUE
mhte.iptw <- TRUE
Anodesnew <- "A1_"%+%(0:(t.end-1))
Cnodesnew <- "A2_"%+%(0:(t.end-1))
L2nodesnew <- "L2_"%+%(1:(t.end-1))
mL2nodesnew <- "m1L2_"%+%(1:(t.end-1))
Lnodesnew <- as.vector(rbind(L2nodesnew, mL2nodesnew))
Ynodesnew <- "Y_"%+%(1:t.end)
finYnodesnew <- Ynodesnew
```

```
dropnms <- c("ID","L2_"%+%t.end,"m1L2_"%+%t.end, "A1_"%+%t.end, "A2_"%+%t.end)
pooledMSM <- FALSE
weight.msm <- FALSE
```

Source code for running the data simulation and estimation with the `ltmleMSM` *function of the* ***ltmle*** *package (Schwab et al. 2014).*

```
simrun_ltmleMSM <- function(sim, DAG, N, t0,gbounds, gforms) {
  library(ltmle)
  O_datnew <- sim(DAG = DAG, n = N)
  ltmleMSMparams <- est.targetMSM(DAG, O_datnew, Aname = "A1", Cname = "A2", Lnames = "L2",
                                  Ytvec = (1:t.end), ACLtvec = (0:t.end), package = "ltmle")
  summeas_arr <- ltmleMSMparams$summeas_arr
  regimens_arr <- ltmleMSMparams$regimens_arr[,c(1:t.end),]
  O_datnewLTCF <- doLTCF(data = O_datnew, LTCF = "Y")
  O_dat_selCnew <- O_datnewLTCF[,-which(names(O_datnewLTCF)%in%dropnms)]
  O_dat_selCnew[,Cnodesnew] <- 1-O_dat_selCnew[,Cnodesnew]
  reslTMLE.MSM <- ltmleMSM(data = O_dat_selCnew, Anodes = Anodesnew, Cnodes = Cnodesnew,
                           Lnodes = Lnodesnew, Ynodes = Ynodesnew,
                           survivalOutcome = survivalOutcome,
                           gform = gforms, Qform = Qforms,
                           stratify = stratify_Qg, mhte.iptw = mhte.iptw,
                           iptw.only = FALSE,
                           working.msm = msm.form, pooledMSM = pooledMSM,
                           final.Ynodes = finYnodesnew, regimes = regimens_arr,
                           summary.measures = summeas_arr, weight.msm=weight.msm,
                           estimate.time = FALSE, gbounds = gbounds)
  iptwMSMcoef <- summary(reslTMLE.MSM, estimator = "iptw")$cmat[,1]
  iptwRD <- MSM_RD_t(resMSM = iptwMSMcoef, t = t0)
  tmleMSMcoef <- summary(reslTMLE.MSM, estimator = "tmle")$cmat[,1]
  tmleRD <- MSM_RD_t(resMSM = tmleMSMcoef, t = t0)
  return(c(simN = sim, iptwRD = iptwRD, tmleRD = tmleRD))
}
```

```
t0 <- 12
Nltmle <- 50000
Nsims <- 1000
source("./determineParallelBackend.R")
sim50K.stratQg.notrunc.g <- foreach(sim = seq(Nsims), .combine = 'rbind')%dopar%{
                      simrun_ltmleMSM(sim = sim,DAG = Ddyn, N = Nltmle, t0 = t0,
                        gbounds = c(0.0000001, 1), gforms = gforms)
            }
save(list = "sim50K.stratQg.notrunc.g", file = "vignette_dat/sim50K.stratQg.notrunc.g.Rdata")
sim50K.stratQg.notrunc.missg <- foreach(sim = seq(Nsims), .combine = 'rbind')%dopar%{
                      simrun_ltmleMSM(sim = sim,DAG = Ddyn, N = Nltmle, t0 = t0,
                        gbounds = c(0.0000001, 1), gforms = gforms_miss)
            }
save(list = "sim50K.stratQg.notrunc.missg", file = "vignette_dat/sim50K.stratQg.notrunc.missg.Rdata")
```

.3. Source code for replicating the simulation study in Lefebvre et al., 2008

Source code for replicating the IPW estimator used by Lefebvre et al. (2008).

```
runMSMsw <- function(DAG0, Lnames, trueA, nsamp, nsims) {
  Lnames_0 <- Lnames%+%"_0"
  Lnames_1 <- Lnames%+%"_1"
  gforms <- c("A_0 ~ "%+%paste(Lnames_0, collapse = " + "), "A_1 ~ A_0 + "%+%paste(Lnames_1, collapse = " + "))
  res_sw <- NULL
  for (sims in (1:nsims)) {
    datO <- sim(DAG0, n = nsamp)
    glmA_0 <- glm(datO[,c("A_0",Lnames_0)], formula = gforms[1], family = "binomial")
    glmA_1 <- glm(datO[,c("A_1","A_0",Lnames_0,Lnames_1)], formula = gforms[2], family = "binomial")
    probA0_1 <- predict(glmA_0,  type = "response")
    weight_t0 <- 1 / (probA0_1^(datO$A_0) * (1-probA0_1)^(1-datO$A_0))
    probA1_1 <- predict(glmA_1,  type = "response")
    weight_t1 <- 1 / (probA1_1^(datO$A_1) * (1-probA1_1)^(1-datO$A_1))
    sw1 <- weight_t0*weight_t1
    emp.pA1cA0 <- table(datO$A_1,datO$A_0)/nrow(datO)
    empPA1 <- data.frame(A_0 = c(0,0,1,1),A_1 = c(0,1,1,0))
    empPA1$empPA_1_cA_0 <- apply(empPA1, 1, function(rowA) emp.pA1cA0[as.character(rowA["A_1"]), as.character(rowA["A_0"])])
    empPA1 <- merge(datO[, c("ID","A_0","A_1")],empPA1, sort = FALSE)
    empPA1 <- empPA1[order(empPA1$ID),]
    swts <- empPA1$empPA_1_cA_0*(weight_t0*weight_t1)
    datO$swts <- swts
    MSMres_sw <- glm(datO, formula = "Y_1 ~ A_0 + A_1", weights = swts, family = "gaussian")
    res_sw <- rbind(res_sw, coef(MSMres_sw))
  }
  meanres <- apply(res_sw, 2, mean)
  Varres <- apply(res_sw, 2, var)
  bias <- c(meanres["A_0"]-trueA["A_0"], meanres["A_1"]-trueA["A_1"])
  MSE <- c(bias^2+Varres[c("A_0","A_1")])
  bias10 <- sprintf("%.3f",bias*10)
  MSE10 <- sprintf("%.3f",MSE*10)
  resrow <- c(bias10[1], MSE10[1], bias10[2], MSE10[2])
  col36names <- c("\\specialcell[t]{A(0)\\\\ Bias*10}",
                  "\\specialcell[t]{A(0)\\\\ MSE*10}",
                  "\\specialcell[t]{A(1)\\\\ Bias*10}",
                  "\\specialcell[t]{A(1)\\\\ MSE*10}")
  names(resrow) <- col36names
  return(resrow)
}
```

Source code for recreating Tables II and IV from Lefebvre et al. (2008).

```
# recreating Tables 2 & 4 reported in Lefebvre et al.
nsamp <- c(300,1000,10000)
# Lefebvre et al. Tab 2:
covnmT2 <- c(c("\\emph{Lefebvre et al.}: Confounder(s) only", rep("",2)),
             c("\\emph{Lefebvre et al.}: Confounder(s) &", "risk factors", rep("",1)))
lefebvreT2 <- data.frame(
  covnm = covnmT2,
  N = rep(nsamp,2),
  A0Bias10 = sprintf("%.3f",c(0.768, 0.265, 0.057, 0.757, 0.283, 0.056)),
  A0MSE10 = sprintf("%.3f",c(1.761, 0.761, 0.146, 1.642, 0.718, 0.139)),
  A1Bias10 = sprintf("%.3f",c(0.889, 0.312, 0.086, 0.836, 0.330, 0.081)),
  A1MSE10 = sprintf("%.3f",c(1.728, 0.723, 0.120, 1.505, 0.638, 0.114)),stringsAsFactors = FALSE)
# Lefebvre et al. Tab 4:
covnmT4 <- c(c("\\emph{Lefebvre et al.}: Confounder(s) only", rep("",2)),
             c("\\emph{Lefebvre et al.}: Confounder(s) &", "risk factors", ""),
             c("\\emph{Lefebvre et al.}: Confounder(s) &", "IVs", ""),
             c("\\emph{Lefebvre et al.}: Confounder(s),", "IVs & risk factors",""),
             c("\\emph{Lefebvre et al.}: Mis-specified", rep("",2)),
             c("\\emph{Lefebvre et al.}: Full Model", rep("",2)))
lefebvreT4 <- data.frame(
  covnm = covnmT4,
  N = rep(nsamp,6),
  A0Bias10 = sprintf("%.3f",c(-0.080, -0.371, -0.368, -0.110, -0.330, -0.378, 1.611,
                              0.824, 0.241, 1.600, 0.867, 0.235, 3.146, 2.460, 2.364,
                              1.524, 0.878, 0.240)),
  A0MSE10 = sprintf("%.3f",c(1.170, 0.385, 0.056, 1.092, 0.340, 0.051, 3.538, 2.063,
                              0.684, 3.477, 2.053, 0.676, 3.326, 1.700, 0.832, 3.648,
                              2.099, 0.679)),
  A1Bias10 = sprintf("%.3f",c(0.099, -0.035, -0.203, 0.112, -0.108, -0.207, 2.069, 1.245,
                              0.379, 2.143, 1.170, 0.372, 5.591, 5.258, 4.943, 2.221, 1.185,
                              0.377)),
  A1MSE10 = sprintf("%.3f",c(1.155, 0.331, 0.043, 0.865, 0.245, 0.037, 3.841, 2.188, 0.622,
                              3.598, 2.043, 0.625, 5.494, 3.851, 2.705, 3.907, 2.099, 0.630)),
                  stringsAsFactors = FALSE)
col1name <- "Covariates in $P(A|L)$"
colnames(lefebvreT2)[1] <- colnames(lefebvreT4)[1] <- col1name
col36names <- c("\\specialcell[t]{A(0)\\\\ Bias*10}",
                "\\specialcell[t]{A(0)\\\\ MSE*10}",
                "\\specialcell[t]{A(1)\\\\ Bias*10}",
                "\\specialcell[t]{A(1)\\\\ MSE*10}")
colnames(lefebvreT2)[3:6] <- colnames(lefebvreT4)[3:6] <- col36names
```

Source code for replicating the simulation results for Scenario 1 in Lefebvre et al. (2008).

```
trueA <- c(A_0 = -0.294, A_1 = -0.370)
nsims <- 10000; restab <- NULL
runsim <- function(Lnames, DAGO) {
  for (nsamp in c(300,1000,10000)) {
    resSc <- runMSMsw(DAGO = DAGO, Lnames = Lnames, trueA = trueA, nsamp = nsamp, nsims = nsims)
    restab <- rbind(restab, c(N = nsamp, resSc))
  }
  restab
}
Lnames <- c("LC1")
covnm <- c("Confounder(s) only", rep("",2))
restab_1 <- cbind(covnm, runsim(Lnames, DAGO.sc1))
# restab_1 <- rbind(restab_1, as.matrix(lefebvreT2[1:3,]))
Lnames <- c("LC1", "LO1", "LO2", "LO3")
covnm <- c("Confounder(s) &", "risk factors", rep("",1))
restab_2 <- cbind(covnm, runsim(Lnames, DAGO.sc1))
# restab_2 <- rbind(restab_2, as.matrix(lefebvreT2[4:6,]))
restab <- rbind(restab_1, restab_2)
col1name <- "Covariates in $P(A|L)$"
colnames(restab)[1] <- col1name
# restabwLef <- restab
# save(list = "restabwLef", file = "vignette_dat/restabwLefSc1_all_1Ksims.Rdata");
# restab <- restab[c(1:3, 7:9),]
save(list = "restab", file = "vignette_dat/restabSc1_all_1Ksims.Rdata");
```

```
library(Hmisc)
load(file = "vignette_dat/restabSc1_all_1Ksims.Rdata");
cat("\n")
latex(restab, file = "", where = "!htpb", caption.loc = 'bottom',
    caption = "Replication of the simulation results from \\citet{lefebvre2008} for Scenario 1.",
    label = 'tab2Lefebvre',booktabs = TRUE,rowname = NULL,landscape = FALSE,
    col.just = c("l", rep("r", 5)), size = "small")
```

```
cat("\n")
latex(lefebvreT2, file = "", where = "!htpb", caption.loc = 'bottom',
    caption = "Simulation results for Scenario 1 as reported in Table II of \\citet{lefebvre2008}.",
    label = 'origtab2Lefebvre', booktabs = TRUE, rowname = NULL, landscape = FALSE,
    col.just = c("l", rep("r", 5)), size = "small")
```

Source code for replicating the simulation results for Scenario 3 in Lefebvre et al. (2008).

```
trueA <- c(A_0 = -0.316, A_1 = -0.390)
nsims <- 10000; restab <- NULL
runsim <- function(Lnames, DAGO) {
  for (nsamp in c(300,1000,10000)) {
    resSc <- runMSMsw(DAGO = DAGO, Lnames = Lnames, trueA = trueA, nsamp = nsamp, nsims = nsims)
    restab <- rbind(restab, c(N = nsamp, resSc))
  }
  restab
}
Lnames <- c("LC1", "LC2", "LC3")
covnm <- c("Confounder(s) only", rep("",2))
restab_1 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_1 <- rbind(restab_1, as.matrix(lefebvreT4[1:3,]))
Lnames <- c("LO1", "LO2", "LO3", "LC1", "LC2", "LC3")
covnm <- c("Confounder(s) &", "risk factors", "")
restab_2 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_2 <- rbind(restab_2, as.matrix(lefebvreT4[4:6,]))
Lnames <- c("LE1", "LE2", "LE3", "LC1", "LC2", "LC3")
covnm <- c("Confounder(s) &", "IVs", "")
restab_3 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_3 <- rbind(restab_3, as.matrix(lefebvreT4[7:9,]))
Lnames <- c("LO1", "LO2", "LO3", "LE1", "LE2", "LE3", "LC1", "LC2", "LC3")
covnm <- c("Confounder(s),", "IVs & risk factors","")
restab_4 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_4 <- rbind(restab_4, as.matrix(lefebvreT4[10:12,]))
Lnames <- c("LE1", "LE2", "LE3", "LC1")
covnm <- c("Mis-specified", rep("",2))
restab_5 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_5 <- rbind(restab_5, as.matrix(lefebvreT4[13:15,]))
Lnames <- c("LO1", "LO2", "LO3", "LE1", "LE2", "LE3", "LC1", "LC2", "LC3", "LN1", "LN2", "LN3")
covnm <- c("Full Model", rep("",2))
restab_6 <- cbind(covnm, runsim(Lnames, DAGO.sc3))
# restab_6 <- rbind(restab_6, as.matrix(lefebvreT4[16:18,]))
restab <- rbind(restab_1, restab_2, restab_3, restab_4, restab_5, restab_6)
col1name <- "Covariates in $P(A|L)$"
colnames(restab)[1] <- col1name
# restabwLef <- restab
# save(list = "restabwLef", file = "vignette_dat/restabwLefSc3_all_1Ksims.Rdata");
# restab <- restab[c(1:3, 7:9, 13:15, 19:21, 25:27, 31:33),]
save(list = "restab", file = "vignette_dat/restabSc3_all_1Ksims.Rdata");
```

```
library(Hmisc)
load(file = "vignette_dat/restabSc3_all_1Ksims.Rdata");
cat("\n")
latex(restab,file = "",where = "!htpb", caption.loc = 'bottom',
    caption = "Replication of the simulation results from \\citet{lefebvre2008} for Scenario 3.",
    label = 'tab4Lefebvre',booktabs = TRUE,rowname = NULL,landscape = FALSE,
    col.just = c("l", rep("r", 5)), size = "small")
```

```
cat("\n")
latex(lefebvreT4,file = "",where = "!htpb", caption.loc = 'bottom',
    caption = "Simulation results for Scenario 3 as reported in Table IV of \\citet{lefebvre2008}.",
    label = 'origtab4Lefebvre',booktabs = TRUE,rowname = NULL,landscape = FALSE,
    col.just = c("l", rep("r", 5)), size = "small")
```

Affiliation:

Oleg Sofrygin
Division of Research
Kaiser Permanente Northern California
Oakland, CA 94612
and
Division of Biostatistics, School of Public Health
University of California, Berkeley
Berkeley, CA 94720
E-mail: oleg.sofrygin@gmail.com

Mark J. van der Laan
Division of Biostatistics, School of Public Health
University of California, Berkeley
Berkeley, CA 94720
E-mail: laan@berkeley.edu

Romain Neugebauer
Division of Research
Kaiser Permanente Northern California
2000 Broadway
Oakland, CA 94612
E-mail: Romain.S.Neugebauer@kp.org