Package 'R6causal'

January 20, 2025

Type Package

Title R6 Class for Structural Causal Models

Version 0.8.3

Maintainer Juha Karvanen <juha.karvanen@iki.fi>

Description The implemented R6 class 'SCM' aims to simplify working with structural causal models. The missing data mechanism can be defined as a part of the structural model. The class contains methods for 1) defining a structural causal model via functions, text or conditional probability tables, 2) printing basic information on the model, 3) plotting the graph for the model using packages 'igraph' or 'qgraph', 4) simulating data from the model, 5) applying an intervention, 6) checking the identifiability of a query using the R packages 'causaleffect' and 'dosearch', 7) defining the missing data mechanism, 8) simulating incomplete data from the model according to the specified missing data mechanism and 9) checking the identifiability in a missing data problem using the R package 'dosearch'. In addition, there are functions for running experiments and doing counterfactual inference using simulation.

License AGPL-3

Encoding UTF-8

RoxygenNote 7.3.1

Imports causaleffect, cfid, data.table, dosearch, glue, igraph, R6, stats, MASS

Suggests rmarkdown, knitr, qgraph, sqldf

VignetteBuilder knitr

NeedsCompilation no

Collate 'R6causal-package.R' 'R6causal_utils.R' 'R6causal.R' 'R6causal_examples.R' 'R6causal_counterfactual.R' 'R6causal_functions.R' 'R6causal_linear.R'

Author Juha Karvanen [aut, cre] (<https://orcid.org/0000-0001-5530-769X>)

Repository CRAN

Date/Publication 2024-03-14 16:10:02 UTC

Contents

`analytic_linear_gaussian` *Simulate data from a conditional linear Gaussian SCM*

Description

Simulate data from a conditional linear Gaussian SCM

Usage

```
analytic_linear_gaussian(scm, situation, n)
```

Arguments

`scm`	A linear Gaussian SCM
`situation`	A list with the following element: • condition : either a string that gives an SQL query (e.g. "select x,y,z from DATA where") or a data.table consisting of the valid rows (e.g. data.table::data.table(x = 0, y = 0))
`n`	The number of rows in the data to be simulated

analytic_linear_gaussian_conditining

Return the mean and the covariance matrix of the conditional distribution of a linear Gaussian SCM

Description

Return the mean and the covariance matrix of the conditional distribution of a linear Gaussian SCM

Usage

```
analytic_linear_gaussian_conditining(scm, situation)
```

Arguments

scm	A linear Gaussian SCM
situation	A list with the following element: • condition : either a string that gives an SQL query (e.g. "select x,y,z from DATA where") or a data.table consisting of the valid rows (e.g. data.table::data.table(x = 0, y = 0))

backdoor

SCM "backdoor" used in the examples.

Description

Variable z fulfills the back-door criterion for P(y|do(x))

Usage

```
backdoor
```

Format

An object of class SCM (inherits from R6) of length 42.

Examples

```
backdoor
backdoor$plot()
```

backdoor_md *SCM "backdoor_md" used in the examples.*

Description

Variable z fulfills the back-door criterion for P(y|do(x)). Variable z is missing completely at random. The missingness of variables x and y depend on z.

Usage

```
backdoor_md
```

Format

An object of class `SCM` (inherits from `R6`) of length 42.

Examples

```
backdoor_md
backdoor_md$plot()
```

counterfactual *Counterfactual inference via simulation*

Description

Counterfactual inference via simulation

Usage

```
counterfactual(
  scm,
  situation,
  n,
  target = NULL,
  ifunction = NULL,
  method = NULL,
  returnscm = FALSE,
  control = NULL
)
```

Arguments

`scm`	An SCM object
`situation`	A list or a character string. The list has the following elements: • do : NULL or a list containing named elements 'target' and 'ifunction' that specify the intervention carried out in the situation • dolist : NULL or a list of lists containing named elements 'target' and 'ifunction' that specify the intervention carried out in each parallel world • condition : either a string that gives an SQL query (e.g. "select x,y,z from DATA where") or a data.table consisting of the valid rows (e.g. data.table::data.table(x = 0, y = 0)) • condition_type : (required only if method == "u_find") A character vector giving the type ("continuous" or "discrete") of every variable in `situation$condition`
`n`	The number of rows in the data to be simulated
`target`	NULL or a vector of variable names that specify the target variable(s) of the counterfactual intervention.
`ifunction`	NULL or a list of functions for the counterfactual intervention.
`method`	The simulation method, "u_find", "rejection" or "analytic_linear_gaussian"
`returnscm`	A logical, should the internally created twin SCM or parallel world SCM returned?
`control`	List of parameters to be passed to the simulation method: • batchsize: (u_find, rejection) The size of data from n observations are resampled (default n) • max_iterations: (u_find) The maximum number of iterations for the binary search (default 50) • minu: (u_find) A scalar or a named list that specifies the lower starting value for the binary search (default -10) • maxu: (u_find) A scalar or a named list that specifies the upper starting value for the binary search (default 10) • sampling_replace: (u_find) Logical, resampling with replacement? (default TRUE) • nonunique_jittersd: (u_find) Standard deviation of the noise to be added to the output (default NULL meaning no noise) • maxbatchs: (u_find, rejection) The maximum number of batches for rejection sampling (for discrete variables) • weightfunction: (u_find) A function or a named list of functions to be applied to dedicated error terms to obtain the resampling weights (default stats::dnorm)

Value

A data table representing the situation after the counterfactual intervention

Examples

```
cfdata <- counterfactual(backdoor,
                         situation = list(
                             do = list(target = "x", ifunction = 0),
                             condition = data.table::data.table( x = 0, y = 0)),
                         target = "x",
                         ifunction = 1,
                         method = "rejection",
                         n = 1000)
mean(cfdata$y)

backdoor_parallel <- ParallelWorld$new(backdoor,
                                       dolist=list(
                                         list(target = "x", ifunction = 0),
                                      list(target = list("z","x"), ifunction = list(1,0))
                                         )
)
cfdata2 <- counterfactual(backdoor_parallel,
                         situation = list(
                             do = NULL,
                           condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                         target = "x",
                         ifunction = 1,
                         method = "rejection",
                         n = 1000)
mean(cfdata2$y)
```

`credit` *SCM "credit" used in the credit scoring example.*

Description

Variable `default` is the outcome to be predicted

Usage

```
credit
```

Format

An object of class `SCM` (inherits from `R6`) of length 42.

Examples

```
credit$simulate(100)
summary(credit$simdata)
```

`fairness` *Checking fairness of a prediction via counterfactual simulation*

Description

Checking fairness of a prediction via counterfactual simulation

Usage

```
fairness(
  modellist,
  scm,
  sensitive,
  condition,
  condition_type,
  parents,
  n,
  sens_values = NULL,
  modeltype = "predict",
  method,
  control = NULL,
  ...
)
```

Arguments

`modellist`	A list of model objects that have a predict method or a list of functions that return predictions
`scm`	An SCM object
`sensitive`	A character vector of the names of sensitive variables
`condition`	A data.table consisting of the valid rows (e.g. data.table::data.table(x = 0, y = 0))
`condition_type`	(required only if method == "u_find") A character vector giving the type ("continuous" or "discrete") of every variable in `condition`
`parents`	A character vector of the names of variables that remain fixed
`n`	The number of rows in the data to be simulated by `counterfactual`
`sens_values`	A data.table specifying the combinations of the values of sensitive variables to be considered (default NULL meaning the all possible combinations of the values of sensitive variables)
`modeltype`	"predict" (default) or "function" depending on the type `modellist`
`method`	The simulation method, "u_find", "rejection" or "analytic_linear_gaussian"
`control`	List of parameters to be passed to the simulation method, see `counterfactual`.
`...`	Other arguments passed to `predict` or to the prediction functions.

Value

A list containing a data table for element of `modellist`. Each data table contains the predicted values after counterfactual interventions on the sensitive variables.

Examples

```
trainingd <- backdoor$simulate(10000, return_simdata = TRUE)
newd <- backdoor$simulate(100, return_simdata = TRUE)
vnames <- backdoor$vnames
m1 <- lm(y ~  x + z, data = trainingd)
m2 <- lm(y ~  z, data = trainingd)
fairlist <- fairness(modellist = list(m1,m2),
                     scm = backdoor,
                     sensitive = c("x"),
                     sens_values = data.table::data.table(x=c(0,1)),
                     condition = newd[1,c("x","y")],
                     condition_type = list(x = "cont",
                                           z = "cont",
                                           y = "cont"),
                     parents = NULL,
                     n = 20,
                     modeltype = "predict",
                     method = "u_find")
```

`frontdoor` *SCM "frontdoor" used in the examples.*

Description

Variable z fulfills the front-door criterion for P(y|do(x))

Usage

```
frontdoor
```

Format

An object of class `SCM` (inherits from `R6`) of length 42.

Examples

```
frontdoor
frontdoor$plot()
```

generate_condprob *Define structural function by a conditional probability table*

Description

Define structural function by a conditional probability table

Usage

```
generate_condprob(ycondx, x, Umerge_expr = NULL)
```

Arguments

ycondx	A data table or a data frame with the following structure • 1st column: variable to be generated, "Y" • middle columns: the parents of the the 1st column variable, "X" • last column: the probability the case specified be the other columns, "P(Y|do(X))"
x	A data table or a data frame that contains data on the variables in the middle columns of ycondx, "X" and one or more columns giving data on U-variables.
Umerge_expr	A character string specifying how the U-variables will be combined when the value "Y" is generated, e.g. "u" or "(u1+u2)/2". The result of the expression should be a random number in the interval [0,1].

Value

A data table containing the generated variable, "Y"

Examples

```
ycondx <- data.table::data.table(y =rep(c(0,1), each = 3), x=rep(1:3, 2),
                      prob = c(0.2,0.6,0.1,0.8,0.4,0.9))
x <- data.table::data.table(x = sample(1:3, 20, replace = TRUE),
                        uy = stats::runif(20), uy2 = stats::runif(20))
generate_condprob(ycondx, x, Umerge_expr = "(uy+uy2)/2")
```

LinearGaussianSCM *R6 Class for linear structural causal models where background variables have Gaussian distribution*

Description

R6 Class for linear structural causal models where background variables have Gaussian distribution

R6 Class for linear structural causal models where background variables have Gaussian distribution

Details

Inherits R6 class SCM.

Super class

R6causal::SCM -> LinearGausianSCM

Active bindings

`linear_gaussian_B` Matrix of structural coefficients of the observed variables

`linear_gaussian_A` Matrix of structural coefficients of the background variables

`linear_gaussian_c` Vector of constants in structural coefficients

Methods

Public methods:

- LinearGaussianSCM$new()
- LinearGaussianSCM$clone()

Method `new()`**:** Create a new linear Gaussian SCM object.

Usage:

```
LinearGaussianSCM$new(
  name = "A linear Gaussian SCM",
  linear_gaussian = NULL,
  random_linear_gaussian = NULL,
  rflist = NULL,
  rprefix = "R_",
  starsuffix = "_md"
)
```

Arguments:

`name` Name.

`linear_gaussian` A list with the following elements:

- uflist: A named list containing the functions for the background variables.
- vnames: A vector of names of the observed variables.
- vcoefmatrix: A matrix of coefficients for observed variables in the structural equations.
- ucoefvector: A vector of the coefficients of dedicated error terms in the structural equations.
- ccoefvector: A vector of constant terms in the structural equations.
- u2coefmatrix: A matrix of the coefficients of confounding background variables in the structural equations. The number of rows equals the number of the observed variables and the number of columns equals the number of confounding background variables.

`random_linear_gaussian` A list with the following elements:

- nv: The number of observed variables
- edgeprob: The probability of an edge between a pair of observed variables (provide either edgeprob or avgneighbors)

- avgneighbors: The average number of edges per a vertex (provide either edgeprob or avgneighbors)
- u2prob: The probability of unobserved confounder between a pair of observed variables (provide either u2prob or avgu2)
- avgu2: The average number of unobserved confounders per a vertex (provide either u2prob or avgu2)
- vcoefdistr: A function that generates the coefficients of observed variables in the structural equations. The function must have argument 'n'.
- ucoefdistr: A function that generates the coefficients of dedicated error terms in the structural equations. The function must have argument 'n'.
- ccoefdistr: A function that generates the constants in the structural equations. The function must have argument 'n'.
- u2coefdistr: A function that generates the coefficients of confounding background variables in the structural equations. The function must have argument 'n'.

`rflist` A named list containing the functions for missingness indicators.

`rprefix` The prefix of the missingness indicators.

`starsuffix` The suffix for variables with missing data.

Returns: A new 'LinearGaussianSCM' object that also belongs to class 'SCM'.

Examples:

```
lgbackdoor <- LinearGaussianSCM$new("Linear Gaussian Backdoor",
                linear_gaussian = list(
                  uflist = list(ux = function(n) {rnorm(n)},
                                uy = function(n) {rnorm(n)},
                                uz = function(n) {rnorm(n)}),
                  vnames = c("x","y","z"),
                  vcoefmatrix = matrix(c(0,0.4,0,0,0,0,0.6,0.8,0),3,3),
                  ucoefvector = c(1,1,1),
                  ccoefvector = c(0,0,0)))
randomlg <- LinearGaussianSCM$new("Random Linear Gaussian",
                random_linear_gaussian = list(
                  nv = 10,
                  edgeprob=0.5,
                  vcoefdistr = function(n) {rnorm(n)},
                  ccoefdistr = function(n) {rnorm(n)},
                  ucoefdistr = function(n) {rnorm(n)}))
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LinearGaussianSCM$new`
```

```
## ------------------------------------------------

lgbackdoor <- LinearGaussianSCM$new("Linear Gaussian Backdoor",
                linear_gaussian = list(
                  uflist = list(ux = function(n) {rnorm(n)},
                                uy = function(n) {rnorm(n)},
                                uz = function(n) {rnorm(n)}),
                  vnames = c("x","y","z"),
                  vcoefmatrix = matrix(c(0,0.4,0,0,0,0,0.6,0.8,0),3,3),
                  ucoefvector = c(1,1,1),
                  ccoefvector = c(0,0,0)))
randomlg <- LinearGaussianSCM$new("Random Linear Gaussian",
                random_linear_gaussian = list(
                  nv = 10,
                  edgeprob=0.5,
                  vcoefdistr = function(n) {rnorm(n)},
                  ccoefdistr = function(n) {rnorm(n)},
                  ucoefdistr = function(n) {rnorm(n)}))
```

`ParallelWorld` *R6 Class for parallel world models*

Description

R6 Class for parallel world models

R6 Class for parallel world models

Details

Inherits R6 class SCM.

Super class

`R6causal::SCM` -> `ParallelWorld`

Active bindings

`num_worlds` Number of parallel worlds.

`worldnames` Names of parallel worlds.

`worldsuffix` Suffix used for parallel world variables.

`originalscm` SCM from which the parallel worlds are derived.

`dolist` List containing the interventions for each world.

Methods

Public methods:

- ParallelWorld$new()
- ParallelWorld$clone()

Method `new()`**:** Create a new ParallelWorld object from an SCM object.

Usage:

```
ParallelWorld$new(scm, dolist, worldnames = NULL, worldsuffix = "_")
```

Arguments:

`scm` An SCM object.

`dolist` A list containing the interventions for each world. Each element of the list has the fields:

- target: a vector of variable names that specify the target variable(s) of the counterfactual intervention.
- ifunction: a list of functions for the counterfactual intervention.

`worldnames` A character vector giving the names of the parallel worlds.

`worldsuffix` A text giving the suffix used for parallel world variables before the world number. Defaults to "_" and the worlds have then suffixes "_1", "_2", "_3", ...

Returns: A new 'ParallelWorld' object that also belongs to class 'SCM'.

Examples:

```
backdoor_parallel <- ParallelWorld$new(
                        backdoor,
                        dolist=list(
                            list(target = "x",
                                 ifunction = 0),
                            list(target = list("z","x"),
                                 ifunction = list(1,0))
                        )
)
backdoor_parallel
plot(backdoor_parallel)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ParallelWorld$new`
## ------------------------------------------------

backdoor_parallel <- ParallelWorld$new(
```

```
                        backdoor,
                        dolist=list(
                            list(target = "x",
                                 ifunction = 0),
                            list(target = list("z","x"),
                                 ifunction = list(1,0))
                        )
)
backdoor_parallel
plot(backdoor_parallel)
```

`R6causalimport` *R6causal: R6 class for structural causal models*

Description

Package R6causal implements an R6 class for structural causal models (SCM) with latent variables and missing data mechanism. The class contains methods for 1) defining a structural causal model via functions, text or conditional probability tables, 2) printing basic information on the model, 3) plotting the graph for the model using packages 'igraph' or 'qgraph', 4) simulating data from the model, 5) applying an intervention, 6) checking the identifiability of a query using the R packages 'causaleffect' and 'dosearch', 7) defining the missing data mechanism, 8) simulating incomplete data from the model according to the specified missing data mechanism and 9) checking the identifiability in a missing data problem using the R package 'dosearch'. In addition, there are functions for running experiments and doing counterfactual inference using simulation.

References

J. Pearl (2009). Causality, 2nd edition, Cambridge University Press.

`run_experiment` *Conduct a sequence of interventions and collect the simulated data.*

Description

Conduct a sequence of interventions and collect the simulated data.

Usage

```
run_experiment(scm, intervene, response, n)
```

Arguments

`scm`	An SCM object
`intervene`	A list where the names of the elements are the variables to be intervened and the values of the elements are vectors specifying the values set in the intervention
`response`	A vector of the names of the response variables
`n`	Size of the data to be simulated for each intervention

Value

A list containing the values of the response variables for all intervention combinations

Examples

```
backdoor_experiment <- run_experiment(backdoor,
                                      intervene = list(x = c(0,1)),
                                      response = "y",
                                      n = 10000)
colMeans(backdoor_experiment$response_list$y)
```

SCM *R6 Class for structural causal models*

Description

R6 Class for structural causal models

R6 Class for structural causal models

Details

An R6 class for structural causal models (SCM) with latent variables and missing data mechanism. There are methods for defining, printing, plotting, intervening and simulating SCMs.

Active bindings

`vflist` List of the structural functions of observed variables.

`vnames` List of the names of observed variables.

`vstarnames` List of the names of observed variables with NA's.

`vfsymb` List of the arguments of structural functions of observed variables.

`uflist` List of the structural functions of unobserved variables.

`unames` List of the names of unobserved variables.

`unames_dedicated` List of the names of unobserved variables that have only one child.

`unames_confounder` List of the names of unobserved variables that have two or more children.

`dedicated_u` Named list of the names of unobserved variables that have only one child which is the name of the element.

`is_linear_gaussian` Logical, does the SCM have linear functions and Gaussian background variables?

`rflist` List of the structural functions of missingness indicators.

`rfsymb` List of the names of missingness indicators.

`rprefix` Prefix used to mark missingness indicators.

`starsuffix` Suffix used to mark variables with missing data.

`simdata` Data table containing data simulated from the SCM.

`simdata_obs` Data table containing data simulated from the SCM where missing values are indicated by `NA`.

`igraph` The graph of the SCM in the `igraph` form (without the missing data mechanism).

`igraph_nodedicated` The graph of the SCM in the `igraph` form (without the dedicated U variables and the missing data mechanism).

`igraph_bidirected` The graph of the SCM in the `igraph` form where latent variables are presented by bidirected arcs.

`igraph_md` The graph of the SCM in the `igraph` form including the missing data mechanism.

`toporder` A vector giving the topological order of variables.

`toporderv` A vector giving the topological order of observed variables.

`graphtext` A character string that gives the edges of the graph of the SCM (without the missing data mechanism).

`graphtext_md` A character string that gives the edges of the graph of the SCM including the missing data mechanism.

`name` The name of the SCM.

Methods

Public methods:

- `SCM$new()`
- `SCM$print()`
- `SCM$plot()`
- `SCM$tikz()`
- `SCM$pa()`
- `SCM$ch()`
- `SCM$an()`
- `SCM$de()`
- `SCM$add_variable()`
- `SCM$remove_variable()`
- `SCM$causal.effect()`
- `SCM$dosearch()`
- `SCM$cfid()`
- `SCM$intervene()`
- `SCM$simulate()`
- `SCM$clone()`

Method `new()`**:** Create a new SCM object.

Usage:

```
SCM$new(
  name = "An SCM",
  uflist = NULL,
  vflist = NULL,
  rflist = NULL,
```

```
  rprefix = "R_",
  starsuffix = "_md"
)
```

Arguments:

`name` Name.

`uflist` A named list containing the functions for the background variables.

`vflist` A named list containing the functions for the observed variables.

`rflist` A named list containing the functions for missingness indicators.

`rprefix` The prefix of the missingness indicators.

`starsuffix` The suffix for variables with missing data.

Returns: A new 'SCM' object.

Examples:

```
backdoor <- SCM$new("backdoor",
 uflist = list(
  uz = function(n) {return(stats::runif(n))},
  ux = function(n) {return(stats::runif(n))},
  uy = function(n) {return(stats::runif(n))}
 ),
 vflist = list(
  z = function(uz) {
    return(as.numeric(uz < 0.4))},
  x = function(ux, z) {
    return(as.numeric(ux < 0.2 + 0.5*z))},
  y = function(uy, z, x) {
    return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
 )
)
```

Method `print()`**:** Print a summmary of the SCM object.

Usage:

```
SCM$print()
```

Examples:

```
backdoor
```

Method `plot()`**:** Plot the DAG of the SCM object.

Usage:

```
SCM$plot(subset = "uvr", method = "igraph", ...)
```

Arguments:

`subset` Variable groups to be plotted: "uvr", "u2vr","vr","uv", "u2v" or "v".

`method` Plotting method: "qgraph" or "igraph".

`...` other parameters passed to the plotting method

Examples:

```
backdoor$plot()
backdoor$plot("v")
```

Method `tikz()`**:** Return a TikZ code for drawing the DAG of the SCM object in LaTeX.

Usage:

```
SCM$tikz(
  subset = "uvr",
  layoutfunction = igraph::layout_with_lgl,
  labels = NULL,
 settings = list(force = FALSE, borders = TRUE, shape = "circle", size = 5, scale = 2),
  ...
)
```

Arguments:

`subset` Variable groups to be plotted: "uvr","vr","uv", or "v".

`layoutfunction` A layout function from igraph package.

`labels` A named list that gives the names of vertices in TikZ.

`settings` A list with the following elements:

`...` Arguments to be passed to layoutfunction

Method `pa()`**:** Return the parents of a set of vertices.

Usage:

```
SCM$pa(vnames, includeself = TRUE)
```

Arguments:

`vnames` A vector of vertex names

`includeself` Logical, should `vnames` to be included in the results (defaults TRUE)

Method `ch()`**:** Return the children of a set of vertices.

Usage:

```
SCM$ch(vnames, includeself = TRUE)
```

Arguments:

`vnames` A vector of vertex names

`includeself` Logical, should `vnames` to be included in the results (defaults TRUE)

Method `an()`**:** Return the ancestors of a set of vertices.

Usage:

```
SCM$an(vnames, includeself = TRUE)
```

Arguments:

`vnames` A vector of vertex names

`includeself` Logical, should `vnames` to be included in the results (defaults TRUE)

Method `de()`**:** Return the descendants of a set of vertices.

Usage:

```
SCM$de(vnames, includeself = TRUE)
```

Arguments:

`vnames` A vector of vertex names

`includeself` Logical, should `vnames` to be included in the results (defaults TRUE)

Method `add_variable()`**:** Add a new variable to the SCM object.

Usage:

```
SCM$add_variable(
  vfnew = NULL,
  ufnew = NULL,
  rfnew = NULL,
  rprefixnew = NULL,
  starsuffixnew = NULL
)
```

Arguments:

`vfnew` NULL or a named list containing the functions for the new observed variables.

`ufnew` NULL or a named list containing the functions for the new latent variables.

`rfnew` NULL or a named list containing the functions for the new missingness indicators.

`rprefixnew` NULL or the prefix of the missingness indicators.

`starsuffixnew` NULL orthe suffix for variables with missing data.

Examples:

```
backdoor2 <- backdoor$clone()
backdoor2$add_variable(
   vfnew = list(
             w = function(uw, x) {
             return(as.numeric(uw < 0.4 + 0.3*x))}),
   ufnew = list(
            uw = function(n) {return(stats::runif(n))})
)
```

Method `remove_variable()`**:** Remove variables from the SCM object.

Usage:

```
SCM$remove_variable(variablenames)
```

Arguments:

`variablenames` Names of the variables to be removed.

Examples:

```
backdoor2 <- backdoor$clone()
backdoor2$remove_variable(c("uy","y"))
#' @include R6causal.R R6causal_examples.R
NULL
```

Method `causal.effect()`**:** Is a causal effect identifiable from observational data? Calls the implementation of ID algorithm from package **causaleffect**. See the documentation of `causal.effect` for the details.

Usage:

```
SCM$causal.effect(y, x, ...)
```

Arguments:

`y` A vector of character strings specifying target variable(s).

`x` A vector of character strings specifying intervention variable(s).

`...` Other parameters passed to `causal.effect`.

Returns: An expression for the joint distribution of the set of variables (y) given the intervention on the set of variables (x) conditional on (z) if the effect is identifiable. Otherwise an error is thrown describing the graphical structure that witnesses non-identifiability. @examples backdoor$causal.effect(y = "y", x = "x")

Method `dosearch()`**:** Is a causal effect or other query identifiable from given data sources? Calls `dosearch` from the package **dosearch**. See the documentation of **dosearch** for the details.

Usage:

```
SCM$dosearch(
  data,
  query,
  transportability = NULL,
  selection_bias = NULL,
  missing_data = NULL,
  control = list()
)
```

Arguments:

`data` Character string specifying the data sources.

`query` Character string specifying the query of interest.

`transportability` Other parameters passed to `dosearch()`.

`selection_bias` Other parameters passed to `dosearch()`.

`missing_data` Other parameters passed to `dosearch()`.

`control` List of control parameters passed to `dosearch()`.

Returns: An object of class `dosearch::dosearch`.

Examples:

```
backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")
```

Method `cfid()`**:** Is a counterfactual query identifiable from given data sources? Calls `identifiable` from the package **cfid**. See the documentation of **cfid** for the details.

Usage:

```
SCM$cfid(gamma, ...)
```

Arguments:

`gamma` An R object that can be coerced into a `cfid::counterfactual_conjunction` object that represents the counterfactual causal query.

`...` Other arguments passed to `cfid::identifiable`.

Returns: An object of class `cfid::query`.

Examples:

```
backdoor$cfid(gamma = cfid::conj(cfid::cf("Y",0), cfid::cf("X",0, c(Z=1))) )
```

Method `intervene()`**:** Apply an intervention to the SCM object.

Usage:

```
SCM$intervene(target, ifunction)
```

Arguments:

`target` Name(s) of the variables (in vflist, uflist or rflist) to be intervened.

`ifunction` Either numeric value(s) or new structural function(s) for the target variables.

Examples:

```
# A simple intervention
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot() # to see that arrows incoming to x are cut

# An intervention that redefines a structural equation
backdoor_yz <- backdoor$clone()  # making a copy
backdoor_yz$intervene("y",
   function(uy, z) {return(as.numeric(uy < 0.1 + 0.8*z ))}) # making y a function of z only
backdoor_yz$plot() # to see that arrow x -> y is cut
```

Method `simulate()`**:** Simulate data from the SCM object. Returns simulated data as a data.table and/or creates or updates `simdata` in the SCM object. If `no_missing_data = FALSE`, creates or updates also `simdata_obs`

Usage:

```
SCM$simulate(
  n = 1,
  no_missing_data = FALSE,
  seed = NULL,
  fixedvars = NULL,
  store_simdata = TRUE,
  return_simdata = FALSE
)
```

Arguments:

`n` Number of observations to be generated.

`no_missing_data` Logical, should the generation of missing data skipped? (defaults FALSE).

`seed` NULL or a number for `set.seed`.

`fixedvars` List of variable names that remain unchanged or a data table/frame that contains the values of the fixed variables.

`store_simdata` Logical, should the simulated data to be stored in the SCM object (defaults TRUE)

`return_simdata` Logical, should the simulated data to be returned as the output (defaults FALSE)

Examples:

```
backdoor$simulate(8, return_simdata = TRUE, store_simdata = FALSE)
backdoor$simulate(10)
backdoor$simdata
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SCM$new`
## ------------------------------------------------

backdoor <- SCM$new("backdoor",
 uflist = list(
  uz = function(n) {return(stats::runif(n))},
  ux = function(n) {return(stats::runif(n))},
  uy = function(n) {return(stats::runif(n))}
 ),
 vflist = list(
  z = function(uz) {
    return(as.numeric(uz < 0.4))},
  x = function(ux, z) {
    return(as.numeric(ux < 0.2 + 0.5*z))},
  y = function(uy, z, x) {
    return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
 )
)

## ------------------------------------------------
## Method `SCM$print`
## ------------------------------------------------

backdoor

## ------------------------------------------------
## Method `SCM$plot`
## ------------------------------------------------

backdoor$plot()
backdoor$plot("v")

## ------------------------------------------------
## Method `SCM$add_variable`
## ------------------------------------------------

backdoor2 <- backdoor$clone()
backdoor2$add_variable(
   vfnew = list(
             w = function(uw, x) {
              return(as.numeric(uw < 0.4 + 0.3*x))}),
   ufnew = list(
            uw = function(n) {return(stats::runif(n))})
)

## ------------------------------------------------
## Method `SCM$remove_variable`
## ------------------------------------------------

backdoor2 <- backdoor$clone()
```

```
backdoor2$remove_variable(c("uy","y"))
#' @include R6causal.R R6causal_examples.R
NULL

## ------------------------------------------------
## Method `SCM$dosearch`
## ------------------------------------------------

backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")

## ------------------------------------------------
## Method `SCM$cfid`
## ------------------------------------------------

backdoor$cfid(gamma = cfid::conj(cfid::cf("Y",0), cfid::cf("X",0, c(Z=1))) )

## ------------------------------------------------
## Method `SCM$intervene`
## ------------------------------------------------

# A simple intervention
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot() # to see that arrows incoming to x are cut

# An intervention that redefines a structural equation
backdoor_yz <- backdoor$clone()  # making a copy
backdoor_yz$intervene("y",
   function(uy, z) {return(as.numeric(uy < 0.1 + 0.8*z ))}) # making y a function of z only
backdoor_yz$plot() # to see that arrow x -> y is cut

## ------------------------------------------------
## Method `SCM$simulate`
## ------------------------------------------------

backdoor$simulate(8, return_simdata = TRUE, store_simdata = FALSE)
backdoor$simulate(10)
backdoor$simdata
```

`trapdoor` *SCM "trapdoor" used in the examples.*

Description

Variable z is a trapdoor variable for P(y|do(x))

Usage

```
trapdoor
```

Format

An object of class SCM (inherits from R6) of length 42.

References

J. Helske, S. Tikka, J. Karvanen (2021). Estimation of causal effects with small data in the presence of trapdoor variables, Journal of the Royal Statistical Society Series A, 184(3), 1030-1051, http://doi.org/10.1111/rssa.12699

Examples

```
trapdoor
trapdoor$plot()
```

Index