Using R6causal

Juha Karvanen

2023-11-18

Overview

The R package `R6causal` implements an R6 class called `SCM`. The class 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

- defining a structural causal model via functions, text or conditional probability tables
- printing basic information on the model
- plotting the graph for the model using packages `igraph` or `qgraph`
- simulating data from the model
- applying an intervention
- checking the identifiability of a query using the R packages `causaleffect` and `dosearch`
- defining the missing data mechanism
- simulating incomplete data from the model according to the specified missing data mechanism
- checking the identifiability in a missing data problem using the R package `dosearch`
- checking the identifiability of a counterfactual query using the R package `cfid`

In addition, there are functions for

- running experiments
- counterfactual inference using simulation
- evaluating fairness of a prediction model

The class `ParallelWorld` inherits `SCM` and defines a structural causal model that describes parallel worlds for counterfactual inference.

The class `LinearGaussianSCM` inherits `SCM` and defines a structural causal model where all functions are linear and all background variables follow Gaussian distribution.

Setup

```
library(R6causal)
library(data.table)
library(stats)
data.table::setDTthreads(2)
```

Defining the model

Structural causal model (SCM) for a backdoor situation can be defined as follows

```
backdoor <- SCM$new("backdoor",
  uflist = list(
```

```
    uz = function(n) {return(runif(n))},
    ux = function(n) {return(runif(n))},
    uy = function(n) {return(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))}
  )
)
```

A shortcut notation for this is

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

Alternatively the functions of SCM can be specified via conditional probability tables

```
backdoor_condprob <- SCM$new("backdoor",
  uflist = list(
    uz = function(n) {return(runif(n))},
    ux = function(n) {return(runif(n))},
    uy = function(n) {return(runif(n))}
  ),
  vflist = list(
    z = function(uz) {
      return( generate_condprob( ycondx = data.table(z = c(0,1),
                                                     prob = c(0.6,0.4)),
                                 x = data.table(uz = uz),
                                 Umerge_expr = "uz"))},
    x = function(ux, z) {
      return( generate_condprob( ycondx = data.table(x = c(0,1,0,1),
                                                     z = c(0,0,1,1),
                                                     prob = c(0.8,0.2,0.3,0.7)),
                                            x = data.table(z = z, ux = ux),
                                            Umerge_expr = "ux"))},
    y = function(uy, z, x) {
      return( generate_condprob( ycondx = data.table(y= rep(c(0,1), 4),
                                                     z = c(0,0,1,1,0,0,1,1),
                                                     x = c(0,0,0,0,1,1,1,1),
                                                     prob = c(0.9,0.1,0.5,0.5,
                                                              0.5,0.5,0.1,0.9)),
```

```
                                                x = data.table(z = z, x = x, uy = uy),
                                                Umerge_expr = "uy"))}
  )
)
```

It is possible to mix the styles and define some elements of a function list as functions, some as text and some as conditional probability tables.

Defining a linear Gaussian SCM

A linear Gaussian SCM can be defined giving the coefficients for the structural equations:

```
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)))
print(lgbackdoor)
#> Name of the model:  Linear Gaussian Backdoor
#>
#> Graph:
#>  z -> x
#>  x -> y
#>  z -> y
#>
#> Functions of background (exogenous) variables:
#>
#> $ux
#> function(n) {rnorm(n)}
#>
#> $uy
#> function(n) {rnorm(n)}
#>
#> $uz
#> function(n) {rnorm(n)}
#>
#> Functions of endogenous variables:
#>
#> $x
#> function (z, ux)
#> {
#>     return(0 + 0.6 * z + 1 * ux)
#> }
#> <environment: 0x0000024714f15720>
#>
#> $y
#> function (x, z, uy)
#> {
#>     return(0 + 0.4 * x + 0.8 * z + 1 * uy)
#> }
```

```
#> <environment: 0x0000024714f04910>
#>
#> $z
#> function (uz)
#> {
#>     return(0 + 1 * uz)
#> }
#> <environment: 0x0000024714f07dd0>
#>
#> Topological order of endogenous variables:
#> [1] "z" "x" "y"
#>
#> No missing data mechanism
```

It is also possible to generate the underlying DAG and the coefficients randomly:

```
randomlg <- LinearGaussianSCM$new("Random Linear Gaussian",
                                  random_linear_gaussian = list(
                                  nv = 6,
                                  edgeprob=0.5,
                                  vcoefdistr = function(n) {rnorm(n)},
                                  ccoefdistr = function(n) {rnorm(n)},
                                  ucoefdistr = function(n) {rnorm(n)}))
print(randomlg)
#> Name of the model:  Random Linear Gaussian
#>
#> Graph:
#>  v1 -> v2
#>  v2 -> v4
#>  v5 -> v4
#>  v1 -> v6
#>  v2 -> v6
#>  v5 -> v6
#>
#> Functions of background (exogenous) variables:
#>
#> $u1
#> function (n)
#> {
#>     return(rnorm(n))
#> }
#> <environment: 0x00000247186aaf40>
#>
#> $u2
#> function (n)
#> {
#>     return(rnorm(n))
#> }
#> <environment: 0x00000247186b4188>
#>
#> $u3
#> function (n)
#> {
#>     return(rnorm(n))
```

```
#> }
#> <environment: 0x000002471186b1470>
#>
#> $u4
#> function (n)
#> {
#>     return(rnorm(n))
#> }
#> <environment: 0x000002471186ae858>
#>
#> $u5
#> function (n)
#> {
#>     return(rnorm(n))
#> }
#> <environment: 0x000002471186b3b80>
#>
#> $u6
#> function (n)
#> {
#>     return(rnorm(n))
#> }
#> <environment: 0x000002471186befd8>
#>
#> Functions of endogenous variables:
#>
#> $v1
#> function (u1)
#> {
#>     return(-0.874108406310088 + 1.41674293091438 * u1)
#> }
#> <environment: 0x000002471186c1950>
#>
#> $v2
#> function (v1, u2)
#> {
#>     return(0.621408077314456 + 0.877024898333607 * v1 + 0.554819351454142 *
#>         u2)
#> }
#> <environment: 0x000002471186ccc90>
#>
#> $v3
#> function (u3)
#> {
#>     return(-0.144717175501409 + -0.656824690079663 * u3)
#> }
#> <environment: 0x000002471186c3ca8>
#>
#> $v4
#> function (v2, v5, u4)
#> {
#>     return(1.24385329947927 + 0.749819455160944 * v2 + 0.249156721203491 *
#>         v5 + -1.3904985686492 * u4)
```

```
#> }
#> <environment: 0x000002471186d2dd8>
#>
#> $v5
#> function (u5)
#> {
#>     return(-1.11892080931192 + -0.479777443284055 * u5)
#> }
#> <environment: 0x00000247186c7b40>
#>
#> $v6
#> function (v1, v2, v5, u6)
#> {
#>     return(-1.45693866091209 + -0.4811146483387 * v1 + -0.69388664633599 *
#>         v2 + -0.768457818139036 * v5 + 0.923122323068245 * u6)
#> }
#> <environment: 0x00000247186cad60>
#>
#> Topological order of endogenous variables:
#> [1] "v1" "v3" "v5" "v2" "v4" "v6"
#>
#> No missing data mechanism
```

Printing the model

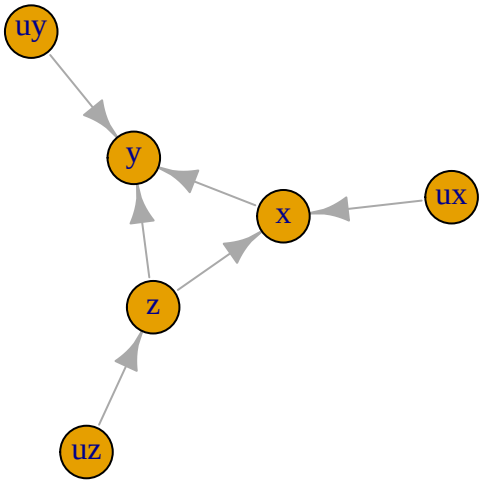
The print method presents the basic information on the model

```
backdoor
#> Name of the model:  backdoor
#>
#> Graph:
#>  z -> x
#>  z -> y
#>  x -> y
#>
#> Functions of background (exogenous) variables:
#>
#> $uz
#> function(n) {return(runif(n))}
#>
#> $ux
#> function(n) {return(runif(n))}
#>
#> $uy
#> function(n) {return(runif(n))}
#>
#> Functions of endogenous variables:
#>
#> $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))}
#>
#> Topological order of endogenous variables:
#> [1] "z" "x" "y"
#>
#> No missing data mechanism
```

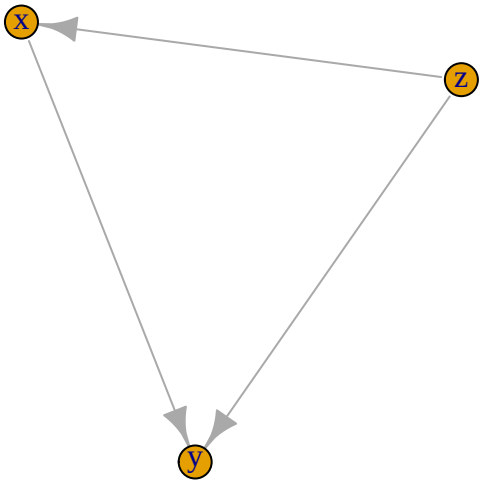
Plotting the graph

The plotting method of the package `igraph` is used by default. If `qgraph` is available, its plotting method can be used as well. The argument `subset` controls which variables are plotted. Plotting parameters are passed to the plotting method.

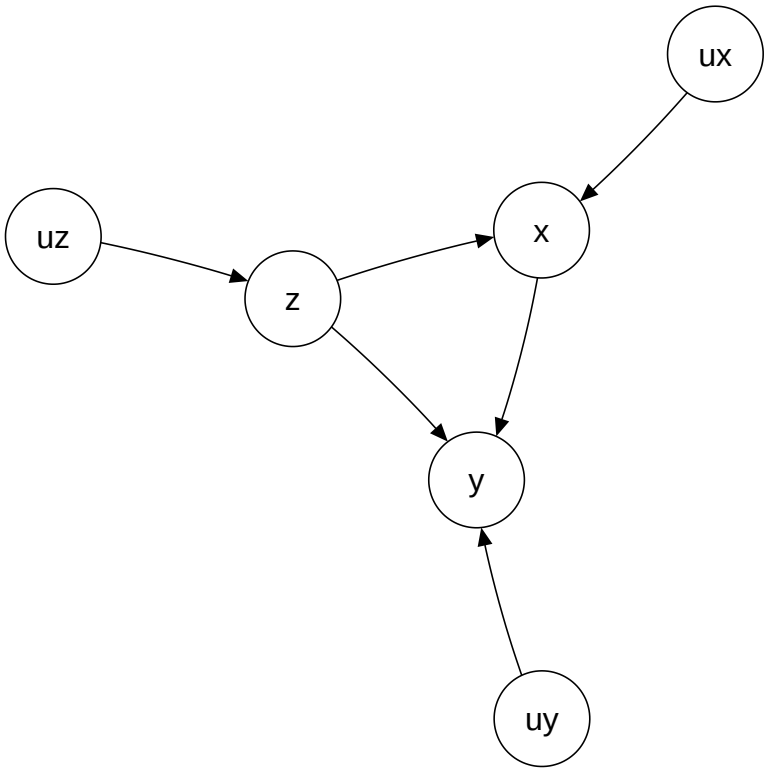
```
backdoor$plot(vertex.size = 25) # with package 'igraph'
```




```
backdoor$plot(subset = "v") # only observed variables
```




```
if (requireNamespace("qgraph", quietly = TRUE)) backdoor$plot(method = "qgraph")
```




```
# alternative look with package 'qgraph'
```

Simulating data

Calling method `simulate()` creates or updates data table `simdata`.

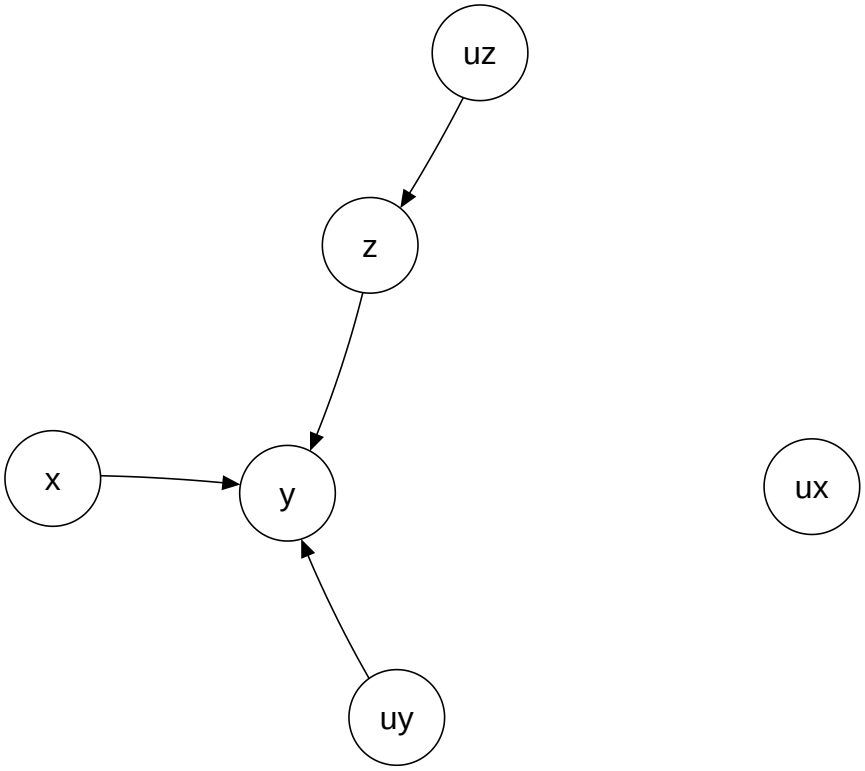
```
backdoor$simulate(10)
backdoor$simdata
#>            uz         ux         uy z x y
#>  1: 0.7728560 0.04783734 0.40316817 0 1 1
#>  2: 0.6557514 0.21003555 0.84930382 0 0 0
#>  3: 0.1498437 0.57696786 0.84806126 1 1 1
#>  4: 0.7447288 0.24595002 0.15895769 0 0 0
#>  5: 0.9976432 0.26092262 0.31783183 0 0 0
#>  6: 0.7159726 0.12435491 0.01870671 0 1 1
#>  7: 0.1958957 0.64853166 0.54125387 1 1 1
#>  8: 0.9931485 0.61877967 0.23995472 0 0 0
#>  9: 0.2697301 0.98479715 0.82915003 1 0 0
#> 10: 0.6505578 0.40424012 0.39835067 0 0 0
backdoor$simulate(8)
backdoor$simdata
#>            uz        ux         uy z x y
#> 1: 0.4833479 0.4007874 0.07220994 0 0 1
#> 2: 0.4462248 0.2344714 0.02897326 0 0 1
#> 3: 0.6069826 0.8923524 0.93320993 0 0 0
#> 4: 0.6357160 0.8017341 0.89557624 0 0 0
#> 5: 0.6751718 0.3245823 0.35437712 0 0 0
#> 6: 0.8872067 0.3097941 0.36439379 0 0 0
#> 7: 0.4310017 0.3422076 0.70888860 0 0 0
```

```
#> 8: 0.2569434 0.5874797 0.07481673 1 1 1
backdoor_text$simulate(20)
backdoor_condprob$simulate(30)
```

Applying an intervention

In an intervention, the structural equation of the target variable is changed.

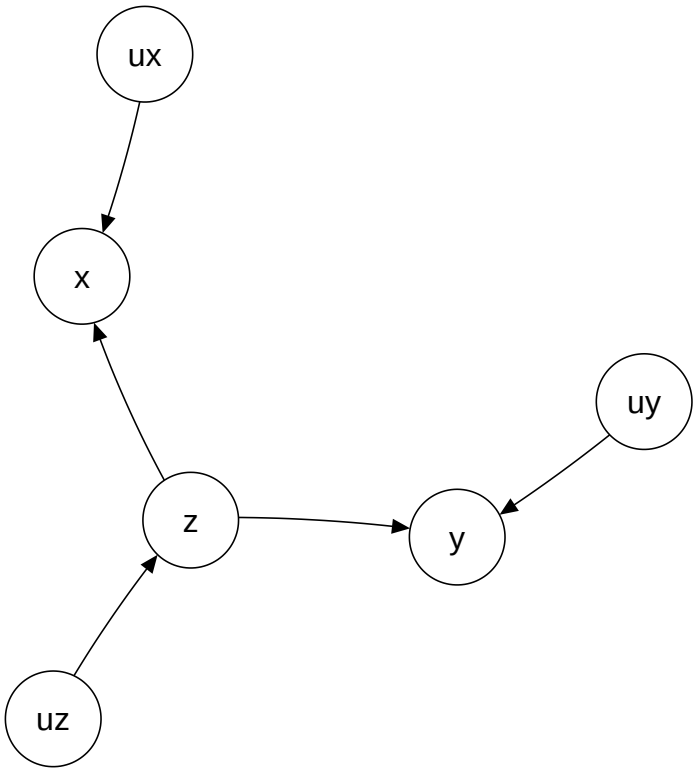
```
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot(method = "qgraph") # to see that arrows incoming to x are cut
```




```
backdoor_x1$simulate(10) # simulating from the intervened model
backdoor_x1$simdata
#>             uz        ux        uy z x y
#>  1: 0.62997290 0.3558503 0.9305637 0 1 0
#>  2: 0.72306475 0.8181661 0.9001112 0 1 0
#>  3: 0.84575187 0.6348200 0.2189730 0 1 1
#>  4: 0.87357792 0.7679340 0.7208477 0 1 0
#>  5: 0.93222775 0.8281281 0.1402974 0 1 1
#>  6: 0.70568889 0.2038273 0.5953877 0 1 0
#>  7: 0.15904410 0.1264877 0.7561948 1 1 1
#>  8: 0.94477340 0.7544117 0.1369158 0 1 1
#>  9: 0.73473279 0.5385419 0.5251055 0 1 0
#> 10: 0.09526467 0.4741884 0.2634645 1 1 1
```

An intervention can redefine 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(method = "qgraph") # to see that arrow x -> y is cut
```



Running an experiment (set of interventions)

The function `run_experiment` applies a set of interventions, simulates data and collects the results.

```
backdoor_experiment <- run_experiment(backdoor,
                                      intervene = list(x = c(0,1)),
                                      response = "y",
                                      n = 10000)
str(backdoor_experiment)
#> List of 2
#>  $ interventions:Classes 'data.table' and 'data.frame':  2 obs. of  1 variable:
#>   ..$ x: num [1:2] 0 1
#>   ..- attr(*, ".internal.selfref")=<externalptr>
#>   ..- attr(*, "sorted")= chr "x"
#>  $ response_list:List of 1
#>   ..$ y:Classes 'data.table' and 'data.frame':   10000 obs. of  2 variables:
#>   .. ..$ V1: num [1:10000] 0 0 1 0 0 1 0 0 1 0 ...
#>   .. ..$ V2: num [1:10000] 1 1 0 1 1 0 1 0 1 1 ...
#>   .. ..- attr(*, ".internal.selfref")=<externalptr>
colMeans(backdoor_experiment$response_list$y)
#>     V1     V2
#> 0.2574 0.6491
```

Applying the ID algorithm, Do-search and cfid

There are direct plugins to R packages `causaleffect`, `dosearch` and `cfid` that can be used to solve identifiability problems.

```
backdoor$causal.effect(y = "y", x = "x")
#> [1] "\\sum_{z}P(y|z,x)P(z)"
backdoor$dosearch(data = "p(x,y,z)", query = "p(y|do(x))")
#> \sum_{z}\left(p(z)p(y|x,z)\right)
backdoor$cfid(gamma = cfid::conj(cfid::cf("Y",0), cfid::cf("X",0, c(Z=1))) )
#> The query P(y /\ x_{z'}) is not identifiable from P_*.
```

Counterfactual inference (a simple case)

Let us assume that intervention do(X=0) was applied and the response Y = 0 was recorded. What is the probability that in this situation the intervention do(X=1) would have led to the response Y = 1? We estimate this probability by means of simulation.

```
cfdata <- counterfactual(backdoor, situation = list(do = list(target = "x", ifunction = 0),
                                                    condition = data.table( x = 0, y = 0)),
                         target = "x", ifunction = 1, n = 100000,
                         control = list(method = "rejection"))
mean(cfdata$y)
#> [1] 0.53761
```

The result differs from P(Y = 1 | do(X = 1))

```
backdoor_x1$simulate(100000)
mean(backdoor_x1$simdata$y)
#> [1] 0.6587
```

Counterfactual inference (parallel worlds)

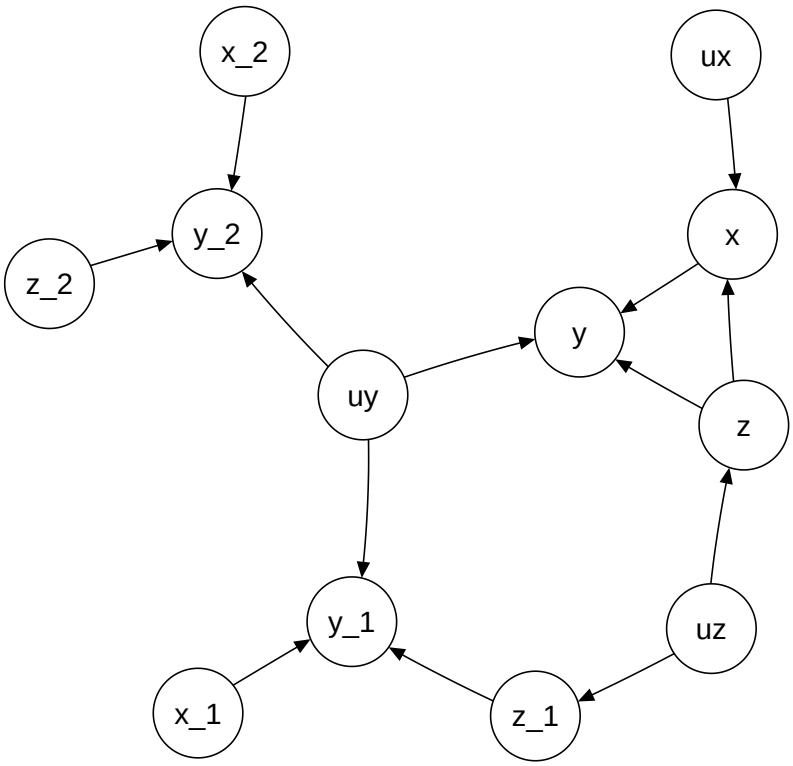
Parallel world graphs (a generalization of a twin graph) are used for counterfactual inference with several counterfactual interventions . The package implements class `ParallelWorld` which heritates class `SCM`. A `ParallelWorld` object is created from an `SCM` object by specifying the interventions for each world. By default the variables of the parallel worlds are named with suffixes "_1", "_2", ...

In the example below, we have the original world (variables `x`, `z`, `y`) and its two variants. In the variant 1 (variables `x_1`, `z_1`, `y_1`), the value of `x` (variable `x_1` in the object) is set to be 0. In the variant 2 (variables `x_2`, `z_2`, `y_2`), the value of `x` (variable `x_2` in the object) is set to be 0 and the value of `z` (variable `z_2` in the object) is set to be 1.

```
backdoor_parallel <- ParallelWorld$new(
                          backdoor,
                         dolist=list(
                              list(target = "x",
                                  ifunction = 0),
                              list(target = list("z","x"),
                                   ifunction = list(1,0))
                          )
 )
backdoor_parallel
#> Name of the model:  backdoor
#>
```

```
#> Graph:
#>  uz -> z
#>  z -> x
#>  uy -> y
#>  z -> y
#>  x -> y
#>  uz -> z_1
#>  uy -> y_1
#>  z_1 -> y_1
#>  x_1 -> y_1
#>  uy -> y_2
#>  z_2 -> y_2
#>  x_2 -> y_2
#>
#> Functions of background (exogenous) variables:
#>
#> $uz
#> function(n) {return(runif(n))}
#> <bytecode: 0x0000024729eee5b8>
#>
#> $ux
#> function(n) {return(runif(n))}
#> <bytecode: 0x0000024729f84d80>
#>
#> $uy
#> function(n) {return(runif(n))}
#> <bytecode: 0x000002472a01d358>
#>
#> Functions of endogenous variables:
#>
#> $z
#> function(uz) {
#>       return(as.numeric(uz < 0.4))}
#> <bytecode: 0x000002472a0d5150>
#>
#> $x
#> function(ux, z) {
#>       return(as.numeric(ux < 0.2 + 0.5*z))}
#> <bytecode: 0x000002472a251d90>
#>
#> $y
#> function(uy, z, x) {
#>       return(as.numeric(uy < 0.1 + 0.4*z + 0.4*x))}
#> <bytecode: 0x000002472a3c6338>
#>
#> $z_1
#> function (uz)
#> {
#>     return(as.numeric(uz < 0.4))
#> }
#>
#> $x_1
#> function (...)
```

```
#> {
#>     return(constant)
#> }
#> <environment: 0x0000024729e56d30>
#>
#> $y_1
#> function (uy, z_1, x_1)
#> {
#>     return(as.numeric(uy < 0.1 + 0.4 * z_1 + 0.4 * x_1))
#> }
#>
#> $z_2
#> function (...)
#> {
#>     return(constant)
#> }
#> <environment: 0x0000024727fc3458>
#>
#> $x_2
#> function (...)
#> {
#>     return(constant)
#> }
#> <environment: 0x0000024727fce858>
#>
#> $y_2
#> function (uy, z_2, x_2)
#> {
#>     return(as.numeric(uy < 0.1 + 0.4 * z_2 + 0.4 * x_2))
#> }
#>
#> Topological order of endogenous variables:
#> [1] "x_1" "z_2" "x_2" "z"   "z_1" "y_2" "x"   "y_1" "y"
#>
#> No missing data mechanism
if (requireNamespace("qgraph", quietly = TRUE)) backdoor_parallel$plot(method = "qgraph")
```



Counterfactual data can be simulated with function `counterfactual`. In the example below, we know that variable `y` obtained value 0 in the original world as well as variants 1 and 2. We are interested in the counterfactual distribution of `y` if `x` had been set to 1.

```
cfdata <- counterfactual(backdoor_parallel,
                         situation = list(
                            do = NULL,
                            condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                         target = "x",
                         ifunction = 1,
                         n = 100000,
                         control = list(method = "rejection"))
mean(cfdata$y)
#> [1] 0.12277
```

The printed value is a simulation based estimate for the counterfactual probability $P(Y = 1)$.

An alternative way for answering the same question defines the case of interest as one of the parallel worlds (here variant 3).

```
backdoor_parallel2 <- ParallelWorld$new(
                         backdoor,
                        dolist=list(
                             list(target = "x",
                                 ifunction = 0),
                             list(target = list("z","x"),
                                  ifunction = list(1,0)),
                              list(target = "x",
                                 ifunction = 1)
                         )
 )
```

```
cfdata <- counterfactual(backdoor_parallel2,
                         situation = list(
                            do = NULL,
                            condition = data.table::data.table( y = 0, y_1 = 0, y_2 = 0)),
                         n = 100000,
                         control = list(method = "rejection"))
mean(cfdata$y_3)
#> [1] 0.12314
```

The printed value is a simulation based estimate for the counterfactual probability $P(Y = 1)$.

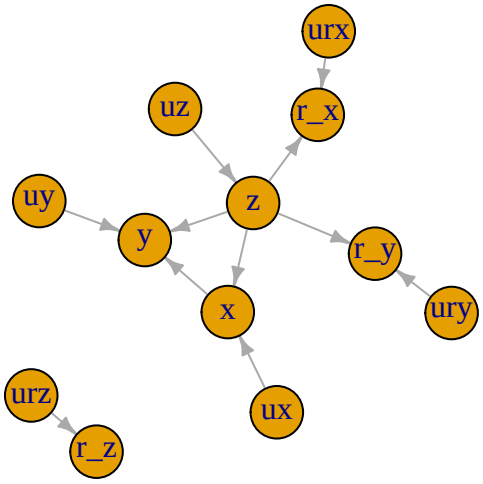
A model with a missing data mechanism

The missing data mechanism is defined in similar manner as the other variables.

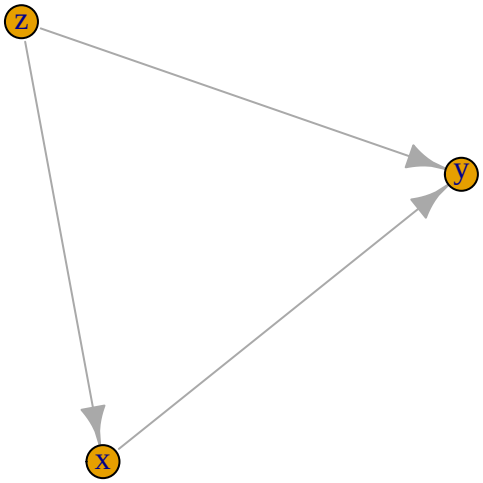
```
backdoor_md <- SCM$new("backdoor_md",
                       uflist = list(
                         uz = "n : runif(n)",
                         ux = "n : runif(n)",
                         uy = "n : runif(n)",
                         urz = "n : runif(n)",
                         urx = "n : runif(n)",
                         ury = "n : runif(n)"
                       ),
                       vflist = list(
                         z = "uz : as.numeric(uz < 0.4)",
                         x = "ux, z : as.numeric(ux < 0.2 + 0.5*z)",
                         y = "uy, z, x : as.numeric(uy < 0.1 + 0.4*z + 0.4*x)"
                       ),
                       rflist = list(
                         z = "urz : as.numeric( urz < 0.9)",
                         x = "urx, z : as.numeric( (urx + z)/2 < 0.9)",
                         y = "ury, z : as.numeric( (ury + z)/2 < 0.9)"
                       ),
                       rprefix = "r_"
)
```

Plotting the graph for a model with missing data mechanism

```
backdoor_md$plot(vertex.size = 25, edge.arrow.size=0.5) # with package 'igraph'
```




```
backdoor_md$plot(subset = "v") # only observed variables a
```




```
if (!requireNamespace("qgraph", quietly = TRUE)) backdoor_md$plot(method = "qgraph")
# alternative look with package 'qgraph'
```

Simulating incomplete data

By default both complete data and incomplete data are simulated. The incomplete dataset is named as `$simdata_obs`.

```
backdoor_md$simulate(100)
summary(backdoor_md$simdata)
#>        uz                  ux                  uy                  urz
#>  Min.   :0.0009881   Min.   :0.006888   Min.   :0.006665   Min.   :0.0249
#>  1st Qu.:0.2005260   1st Qu.:0.253816   1st Qu.:0.194837   1st Qu.:0.2308
#>  Median :0.4625494   Median :0.512815   Median :0.478325   Median :0.4764
#>  Mean   :0.4708455   Mean   :0.487967   Mean   :0.488647   Mean   :0.4880
#>  3rd Qu.:0.7123149   3rd Qu.:0.720430   3rd Qu.:0.741221   3rd Qu.:0.7114
#>  Max.   :0.9930208   Max.   :0.993450   Max.   :0.998812   Max.   :0.9965
#>
#>       urx                 ury                 z                x
#>  Min.   :0.01912   Min.   :0.01456   Min.   :0.00   Min.   :0.00
#>  1st Qu.:0.25383   1st Qu.:0.31613   1st Qu.:0.00   1st Qu.:0.00
```

```
#>  Median :0.49155   Median :0.49779   Median :0.00   Median :0.00
#>  Mean   :0.50928   Mean   :0.49643   Mean   :0.46   Mean   :0.44
#>  3rd Qu.:0.79531   3rd Qu.:0.70900   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :0.98021   Max.   :0.97863   Max.   :1.00   Max.   :1.00
#>
#>        y             z_md             x_md             y_md
#>  Min.   :0.00   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
#>  1st Qu.:0.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
#>  Median :0.00   Median :0.0000   Median :0.0000   Median :0.0000
#>  Mean   :0.46   Mean   :0.4725   Mean   :0.4432   Mean   :0.4409
#>  3rd Qu.:1.00   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
#>  Max.   :1.00   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
#>                 NA's   :9        NA's   :12       NA's   :7
#>       r_z             r_x             r_y
#>  Min.   :0.00   Min.   :0.00   Min.   :0.00
#>  1st Qu.:1.00   1st Qu.:1.00   1st Qu.:1.00
#>  Median :1.00   Median :1.00   Median :1.00
#>  Mean   :0.91   Mean   :0.88   Mean   :0.93
#>  3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :1.00   Max.   :1.00   Max.   :1.00
#>
summary(backdoor_md$simdata_obs)
#>       z_md             x_md             y_md             r_z
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00
#>  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.00
#>  Median :0.0000   Median :0.0000   Median :0.0000   Median :1.00
#>  Mean   :0.4725   Mean   :0.4432   Mean   :0.4409   Mean   :0.91
#>  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.00
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00
#>  NA's   :9        NA's   :12       NA's   :7
#>       r_x             r_y
#>  Min.   :0.00   Min.   :0.00
#>  1st Qu.:1.00   1st Qu.:1.00
#>  Median :1.00   Median :1.00
#>  Mean   :0.88   Mean   :0.93
#>  3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :1.00   Max.   :1.00
#>
```

By using the argument `fixedvars` one can keep the complete data unchanged and re-simulate the missing data mechanism.

```
backdoor_md$simulate(100, fixedvars = c("x","y","z","ux","uy","uz"))
summary(backdoor_md$simdata)
#>        uz                 ux                 uy                 urz
#>  Min.   :0.0009881   Min.   :0.006888   Min.   :0.006665   Min.   :0.006917
#>  1st Qu.:0.2005260   1st Qu.:0.253816   1st Qu.:0.194837   1st Qu.:0.222695
#>  Median :0.4625494   Median :0.512815   Median :0.478325   Median :0.429692
#>  Mean   :0.4708455   Mean   :0.487967   Mean   :0.488647   Mean   :0.449283
#>  3rd Qu.:0.7123149   3rd Qu.:0.720430   3rd Qu.:0.741221   3rd Qu.:0.628679
#>  Max.   :0.9930208   Max.   :0.993450   Max.   :0.998812   Max.   :0.995040
#>
#>       urx                ury                 z              x
#>  Min.   :0.002533   Min.   :0.009308   Min.   :0.00   Min.   :0.00
```

```
#>  1st Qu.:0.176621   1st Qu.:0.208492   1st Qu.:0.00   1st Qu.:0.00
#>  Median :0.462471   Median :0.544313   Median :0.00   Median :0.00
#>  Mean   :0.440669   Mean   :0.516955   Mean   :0.46   Mean   :0.44
#>  3rd Qu.:0.628319   3rd Qu.:0.786014   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :0.994485   Max.   :0.992411   Max.   :1.00   Max.   :1.00
#>
#>        y              z_md              x_md              y_md
#>  Min.   :0.00   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
#>  1st Qu.:0.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
#>  Median :0.00   Median :0.0000   Median :0.0000   Median :0.0000
#>  Mean   :0.46   Mean   :0.4681   Mean   :0.4239   Mean   :0.4318
#>  3rd Qu.:1.00   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
#>                 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
#>                 NA's   :6        NA's   :8        NA's   :12
#>       r_z             r_x             r_y
#>  Min.   :0.00   Min.   :0.00   Min.   :0.00
#>  1st Qu.:1.00   1st Qu.:1.00   1st Qu.:1.00
#>  Median :1.00   Median :1.00   Median :1.00
#>  Mean   :0.94   Mean   :0.92   Mean   :0.88
#>  3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :1.00   Max.   :1.00   Max.   :1.00
#>
summary(backdoor_md$simdata_obs)
#>       z_md              x_md              y_md              r_z
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00
#>  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.00
#>  Median :0.0000   Median :0.0000   Median :0.0000   Median :1.00
#>  Mean   :0.4681   Mean   :0.4239   Mean   :0.4318   Mean   :0.94
#>  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.00
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00
#>  NA's   :6        NA's   :8        NA's   :12
#>       r_x             r_y
#>  Min.   :0.00   Min.   :0.00
#>  1st Qu.:1.00   1st Qu.:1.00
#>  Median :1.00   Median :1.00
#>  Mean   :0.92   Mean   :0.88
#>  3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :1.00   Max.   :1.00
#>
```

Applying Do-search to a missing data problem

```
backdoor_md$dosearch(data = "p(x*,y*,z*,r_x,r_y,r_z)", query = "p(y|do(x))")
#> \sum_{z}\left(\frac{p(z,r_z = 1)}{p(r_z = 1)}p(y|z,r_z = 1,x,r_x = 1,r_y = 1)\right)
```

It is automatically recognized that the problem is a missing data problem when `rflist != NULL`.