Using R6causal

Juha Karvanen

2021-08-04

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`

In addition, there are functions for

- running experiments
- counterfactual inference using simulation

Setup

```
library(R6causal)
library(data.table)
library(stats)
```

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.

Printing the model

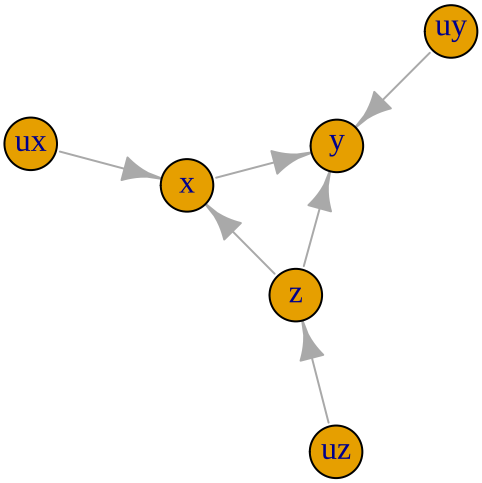
The print method presents the basic information on the model

```
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
```

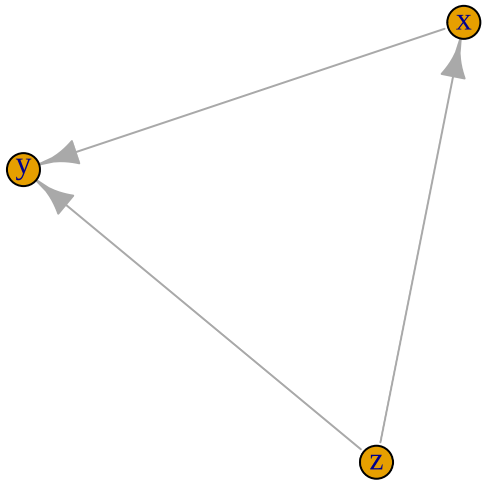
Plotting the graph

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.

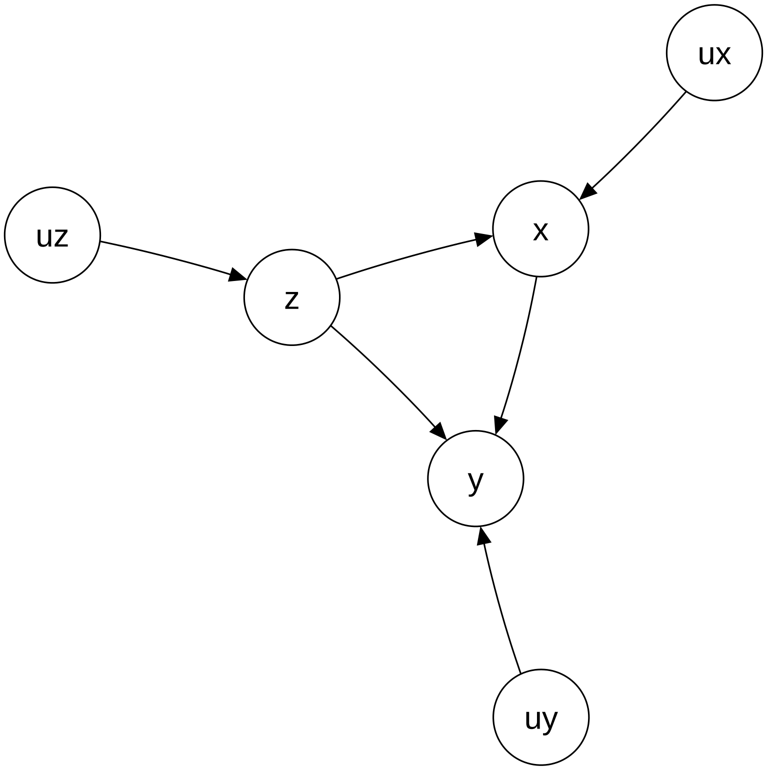
```
backdoor$plot(vertex.size = 25) # with package 'igraph'
```



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



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



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

Simulating data

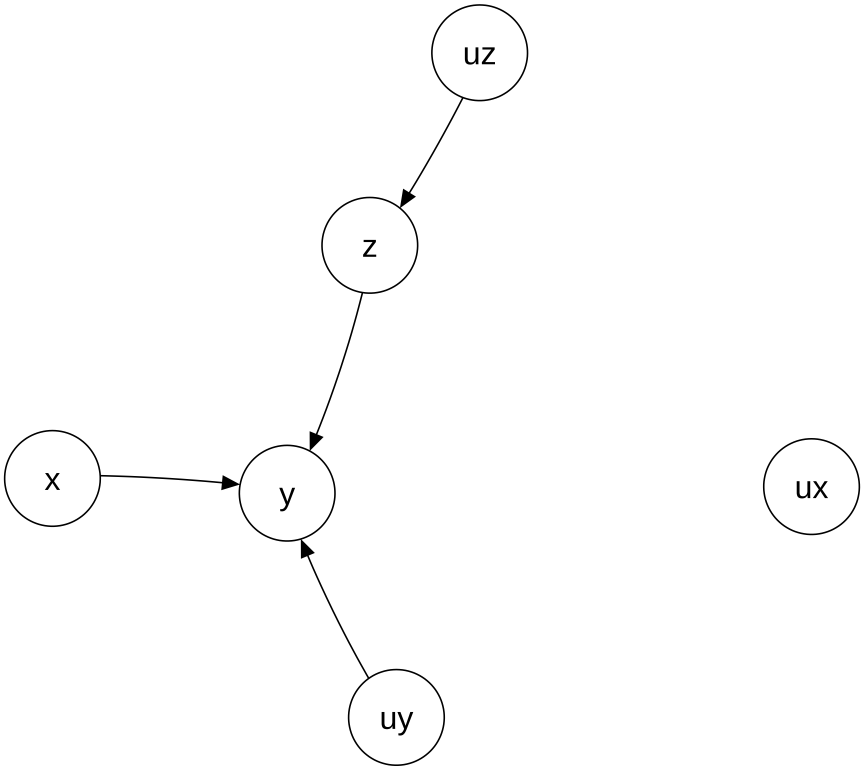
Calling method `simulate()` creates or updates data table `simdata`.

```
backdoor$simulate(10)
backdoor$simdata
#>              uz        ux          uy z x y
#>  1: 0.08825727 0.2134524 0.144842838 1 1 1
#>  2: 0.39153788 0.8432795 0.140244378 1 0 1
#>  3: 0.67716922 0.5522505 0.798552506 0 0 0
#>  4: 0.24316595 0.5277762 0.774681081 1 1 1
#>  5: 0.16763421 0.4238919 0.844601495 1 1 1
#>  6: 0.88362075 0.2350016 0.009793869 0 0 1
#>  7: 0.92164964 0.2178123 0.303537107 0 0 0
#>  8: 0.78831191 0.2436196 0.521628107 0 0 0
#>  9: 0.67479687 0.8198795 0.563194058 0 0 0
#> 10: 0.53531451 0.4059065 0.853722318 0 0 0
backdoor$simulate(8)
backdoor$simdata
#>            uz          ux          uy z x y
#> 1: 0.07952332 0.870030816 0.009026842 1 0 1
#> 2: 0.97010495 0.146590343 0.342205652 0 1 1
#> 3: 0.67273276 0.499134110 0.624593021 0 0 0
#> 4: 0.81241975 0.007645814 0.043582779 0 1 1
#> 5: 0.13089738 0.815788944 0.627113224 1 0 0
#> 6: 0.08545426 0.749640221 0.197122378 1 0 1
#> 7: 0.35595625 0.054852621 0.400546687 1 1 1
#> 8: 0.24686794 0.377255672 0.320359227 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.

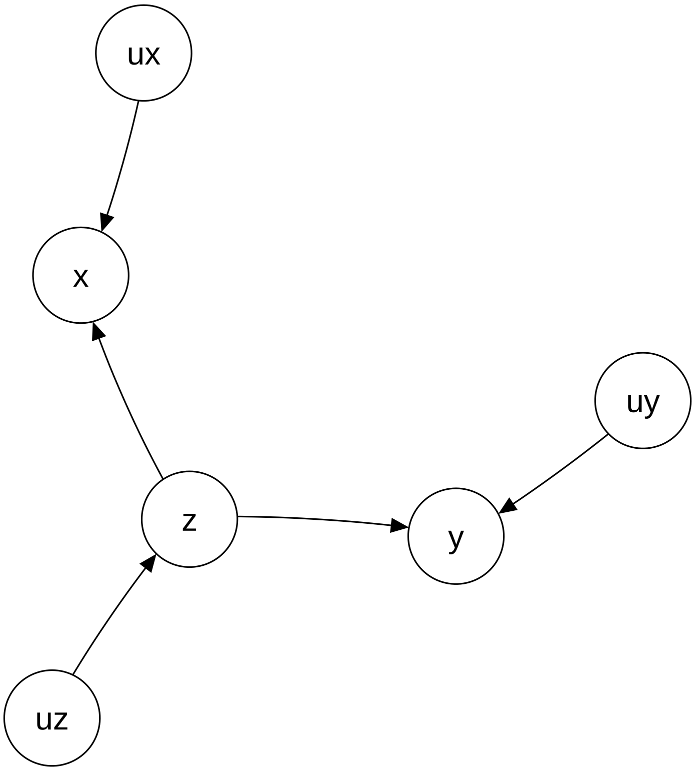
```
backdoor_x1 <- backdoor$clone()  # making a copy
backdoor_x1$intervene("x",1) # applying the intervention
backdoor_x1$plot(method = "qgraph") # to see that arrows incoming to x are cut
```



```
backdoor_x1$simulate(10) # simulating from the intervened model
backdoor_x1$simdata
#>             uz         ux         uy z x y
#>  1: 0.01355571 0.07589350 0.16523142 1 1 1
#>  2: 0.46773195 0.79690430 0.79729327 0 1 0
#>  3: 0.88419213 0.04956716 0.05327442 0 1 1
#>  4: 0.17917345 0.57871292 0.62271447 1 1 1
#>  5: 0.76973670 0.87320844 0.32136078 0 1 1
#>  6: 0.27850679 0.29007694 0.40752861 1 1 1
#>  7: 0.74907192 0.36657186 0.49661998 0 1 1
#>  8: 0.42173200 0.94241092 0.83722295 0 1 0
#>  9: 0.84223875 0.80206929 0.99891863 0 1 0
#> 10: 0.37151395 0.49432979 0.10927442 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 1 1 0 1 0 1 1 1 1 ...
#>   .. ..$ V2: num [1:10000] 1 1 0 1 1 1 1 0 1 1 ...
#>   .. ..- attr(*, ".internal.selfref")=<externalptr>
colMeans(backdoor_experiment$response_list$y)
#>     V1     V2
#> 0.2604 0.6613
```

Applying the ID algorithm and Do-search

There are direct plugins to R packages `causaleffect` and `dosearch` 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)
```

Counterfactual inference

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)
mean(cfdata$y)
#> [1] 0.54219
```

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

```
backdoor_x1$simulate(100000)
mean(backdoor_x1$simdata$y)
#> [1] 0.659
```

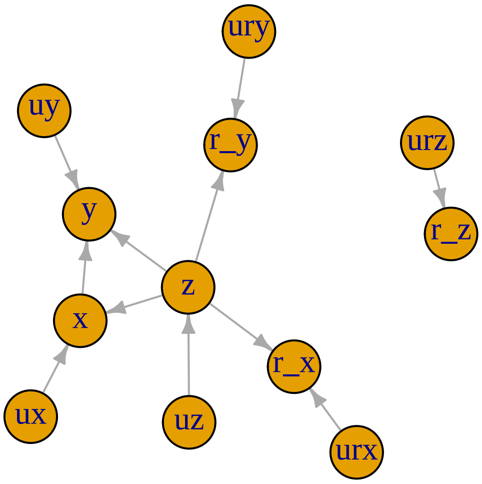
A model with a missing data mechanism

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

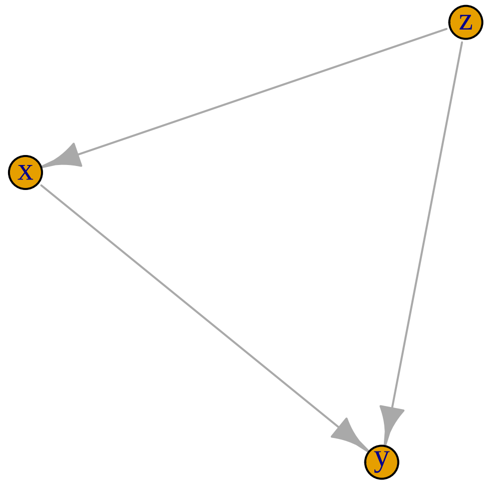
```
backdoor_md <- SCM$new("backdoor_md",
                       uflist = list(
                         uz = "n : runif(n)",
                         ux = "n : runif(n)",
                         uy = "n : runif(n)",
                         urz = "n : runif(n)",
                         urx = "n : 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_md`.

```
backdoor_md$simulate(100)
summary(backdoor_md$simdata)
#>        uz                 ux                 uy                 urz          
#>  Min.   :0.01017   Min.   :0.00731   Min.   :0.03904   Min.   :0.005941  
#>  1st Qu.:0.24106   1st Qu.:0.31420   1st Qu.:0.21873   1st Qu.:0.234579  
#>  Median :0.49156   Median :0.47677   Median :0.45914   Median :0.518430  
#>  Mean   :0.50415   Mean   :0.51465   Mean   :0.48064   Mean   :0.488372  
#>  3rd Qu.:0.75188   3rd Qu.:0.75245   3rd Qu.:0.74231   3rd Qu.:0.700892  
#>  Max.   :0.99997   Max.   :0.99723   Max.   :0.99983   Max.   :0.983133  
#>       urx                ury                z              x        
#>  Min.   :0.002825   Min.   :0.0184   Min.   :0.00   Min.   :0.00  
#>  1st Qu.:0.212025   1st Qu.:0.2587   1st Qu.:0.00   1st Qu.:0.00  
#>  Median :0.494262   Median :0.4777   Median :0.00   Median :0.00  
#>  Mean   :0.509594   Mean   :0.5029   Mean   :0.35   Mean   :0.36  
#>  3rd Qu.:0.769464   3rd Qu.:0.7837   3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :0.999450   Max.   :0.9944   Max.   :1.00   Max.   :1.00  
#>        y       
#>  Min.   :0.00
#>  1st Qu.:0.00
#>  Median :0.00
#>  Mean   :0.33
#>  3rd Qu.:1.00
#>  Max.   :1.00
summary(backdoor_md$simdata_md)
#>       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   :0.3407   Mean   :0.3404   Mean   :0.3043   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   :6        NA's   :8
#>       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   :0.94   Mean   :0.92
#>  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.   :0.01017   Min.   :0.00731   Min.   :0.03904   Min.   :0.006097
#>  1st Qu.:0.24106   1st Qu.:0.31420   1st Qu.:0.21873   1st Qu.:0.285620
#>  Median :0.49156   Median :0.47677   Median :0.45914   Median :0.532844
#>  Mean   :0.50415   Mean   :0.51465   Mean   :0.48064   Mean   :0.524816
#>  3rd Qu.:0.75188   3rd Qu.:0.75245   3rd Qu.:0.74231   3rd Qu.:0.761368
#>  Max.   :0.99997   Max.   :0.99723   Max.   :0.99983   Max.   :0.986495
#>       urx               ury                 z               x
#>  Min.   :0.02144   Min.   :0.001754   Min.   :0.00   Min.   :0.00
#>  1st Qu.:0.27624   1st Qu.:0.212716   1st Qu.:0.00   1st Qu.:0.00
#>  Median :0.60300   Median :0.459770   Median :0.00   Median :0.00
#>  Mean   :0.55069   Mean   :0.457759   Mean   :0.35   Mean   :0.36
#>  3rd Qu.:0.81051   3rd Qu.:0.687357   3rd Qu.:1.00   3rd Qu.:1.00
#>  Max.   :0.98697   Max.   :0.985962   Max.   :1.00   Max.   :1.00
#>        y
#>  Min.   :0.00
#>  1st Qu.:0.00
#>  Median :0.00
#>  Mean   :0.33
#>  3rd Qu.:1.00
#>  Max.   :1.00
summary(backdoor_md$simdata_md)
#>       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   :0.3793   Mean   :0.3298   Mean   :0.2872   Mean   :0.87  
#>  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   :13       NA's   :6        NA's   :6                      
#>       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   :0.94   Mean   :0.94  
#>  3rd Qu.:1.00   3rd Qu.:1.00  
#>  Max.   :1.00   Max.   :1.00  
#> 
```

Applying Do-search for 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`.