SurvivalSimulation

```r
#remotes::install_github("osofr/simcausal")
library(simcausal)
```

The package sincausal allows one to simulate from user defined structural equation models. The main idea is to define a set of nodes which are causally ordered, each with their own distribution. Their distribution might depend on the values of past nodes. Therefore, the data must be simulated sequentially, following the causal ordering.

For an introduction to the package, see the github: https://github.com/osofr/simcausal

Sample survival data (W, A, Delta, Ttilde) from scratch

```r
#Lets drawn data from a parametric distribution using the package simcausal (github)
# We will assume that this is the real data from our study


D <- DAG.empty()
D <- D +
  node("W", distr = "runif", min =1, max = 25) +
  node("A", distr = "rbinom", size = 1, prob = 0.2 + W/50 ) +
  node("T", distr = "rexp", rate = 0.05*(1 - 0.5*A + 0.3*W) )+
  node("C", distr = "rexp",  rate = 0.03*(1 - 0.4*A + 0.2*W)) +
  node("Delta", distr = "rconst", const = as.numeric(T<=C)  ) +
  node("Ttilde", distr = "rconst", const = round(min(T,C,10) +1  ))
setD <- set.DAG(D)
```

```
## ...automatically assigning order attribute to some nodes...

## node W, order:1

## node A, order:2

## node T, order:3

## node C, order:4

## node Delta, order:5

## node Ttilde, order:6

## as.numeric(T <= C)
## min(cbind_mod(T, C, 10))
```

```r
# Lets draw 200 samples
dat <- sim(setD, n = 200)
```

```
## simulating observed dataset from the DAG object

## as.numeric(T <= C)
## min(cbind_mod(T, C, 10))
```

```r
head(dat)
```

```
##   ID         W A          T         C Delta Ttilde
## 1  1 13.587693 0 10.6352720  1.554030     0      3
## 2  2  8.268513 1  1.1255897  4.686286     1      2
## 3  3  5.852753 1  1.6664215 21.567634     1      3
## 4  4 16.497725 1 11.2699025  4.178770     0      5
## 5  5 24.248153 1  0.8468451  9.698357     1      2
## 6  6 11.544939 0 11.9683682 17.160510     1     11
```

```r
#Lets assume we are outcome blind.
# Thus, we are only given the outcome blind data (W, A)
baseline_data <- dat[, c("W", "A")]
#We can learn P(A |W) from the given data. For simplicty, lets use logistic regression
fit <- glm("A ~ W", family = binomial(), data = baseline_data)
#Lets store P(A=1|W) as a function of w
PA1_W <- function(w){
  as.vector(predict(fit, newdata = list(W = w), type = "response"))
}
PA1_W(0.5)
```

```
## [1] 0.2073618
```

Simulate survival data from a known (empirical distribution) of W and estimated conditional density of binary A and parametric forms for censoring/survival

```r
# Our goal is to sample (W, A, Delta, Ttilde) survival data structure where W and A are sampled from di
# Lets estimate P(W) with the empirical distribution so that sampling from it is trivial
#Next, we assume that A is binary so that once we have drawn W = w, we can draw A using rbinom(1, size 


n = 50
Wsamp = baseline_data$W
#This function samples from an empirical sample
remp <- function(n){
  emp_sample <- Wsamp
  sample(emp_sample, n, replace = T)
}

# Should be the function P(A = 1 |W).
prob_map <- PA1_W
#Make sure this map is vectorized if needed
#prob_map <- Vectorize(prob_map)
#This function samples from A conditional on some value (i.e. W in this case) using our estimated condi
rTrtment <- function(n, value, prob){
  rbinom(n, size = 1, prob= prob(value))
}


#Now, we would like to draw survival data. Specifically, we want to draw a time of death T and time of
#For this, we need to specify the conditional distributions P(T <= t| A, W) and
```

```r
# P(C <= t| A, W).
# We can parametrically model these conditional distributions (as shown below) or we can estimate these


#First, let us use a parametric form for these conditional distributions of survival and censoring. We
#where the shape and scale parameters depend on A and W
D <- DAG.empty()
D <- D +
  # This node represents a sample of W's from our empirical distribution
  node("W", distr = "remp") +
    # This node represents a sample of A's from our conditional distribution of A given the previously
  node("A", distr = "rTrtment", value = W, prob =  prob_map) +
   # This node represents a drawn survival time from a parametric form of the conditional survival dist
  node("T", distr = "rweibull", shape = W + 5*A + W*A, scale = W + 5*A - 0.2*A*W) +
  # Same as above but for censoring
    node("C", distr = "rweibull", shape = W + 5*A + W*A, scale = W + 5*A - 0.205*A*W)+
  # For simplicity, lets make the times discrete by rounding up
  node("Tdiscrete", distr = "rconst", const = ceiling(T)) +
    node("Cdiscrete", distr = "rconst", const = ceiling(C)) +
  #Now we extract Ttilde and Delta
  node("Ttilde", distr = "rconst", const = min(Tdiscrete, Cdiscrete)) +
  node("Delta", distr = "rconst", const = as.numeric(Ttilde  == Tdiscrete) )


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

```
## ...automatically assigning order attribute to some nodes...

## node W, order:1

## node A, order:2

## node T, order:3

## node C, order:4

## node Tdiscrete, order:5

## node Cdiscrete, order:6

## node Ttilde, order:7

## node Delta, order:8

## The node A expression(s): W.
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how

## min(cbind_mod(Tdiscrete, Cdiscrete))
## as.numeric(Ttilde == Tdiscrete)
```

```r
#Now we simulate the data by drawing n = 200 samples
dat <- sim(setD, n = 200)
```

```
## simulating observed dataset from the DAG object
## The node A expression(s): W.
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how

## min(cbind_mod(Tdiscrete, Cdiscrete))
## as.numeric(Ttilde == Tdiscrete)
```

```
# only grab ID, W's, A, T.tilde, Delta
head(dat)
```

```
##   ID         W A         T         C Tdiscrete Cdiscrete Ttilde Delta
## 1  1  3.746504 1  8.064032  7.844140         9         8      8     0
## 2  2 23.168090 0 23.502988 23.083755        24        24     24     1
## 3  3  5.625735 0  4.257976  5.781437         5         6      5     1
## 4  4 14.655132 0 14.093016 14.557424        15        15     15     1
## 5  5 17.521842 0 16.007818 17.599248        17        18     17     1
## 6  6 11.472238 1 14.254654 14.124535        15        15     15     1
```

```
#Now we can extract our simulated observed data
observed_data <- dat[, c("W", "A", "Ttilde", "Delta")]
head(observed_data)
```

```
##           W A Ttilde Delta
## 1  3.746504 1      8     0
## 2 23.168090 0     24     1
## 3  5.625735 0      5     1
## 4 14.655132 0     15     1
## 5 17.521842 0     17     1
## 6 11.472238 1     15     1
```

Simulate data from an estimated survival distribution

Now, lets assume we have an estimated conditional survival function.

```
#Lets assume our estimated survival is exactly the pweibull distribution we previously used for simulat
est_surv <- function(t, A, W){
  #This could be any estimated survival function (e.g. survival from cox fit)
  1 - pweibull(t, shape = W + 5*A + W*A, scale = W + 5*A - 0.2*A*W)
}
# We need the CDF for sampling
est_CDF <- function(t, A ,W){
  return(1 - est_surv(t, A, W))
}
library(stats)
#This function takes a (absolutely) continuous CDF and returns the inverse
#Note if our survival data is discrete time then we will need to drawn in a different way using the haz
# Function to numerically compute inverse:
est_cdf_inverse <- function(p, A ,W){
  f_zero <- function(t){
    est_CDF(t, A, W) - p
  }
  grid_endpoints <- c(0, 30)
  root <- stats::uniroot(f_zero, interval = grid_endpoints)$root
  return(root)
}
#We need this to be vectorized
est_cdf_inverse <- Vectorize(est_cdf_inverse)
# True inverse using qweibull
true_inverse <- function(p , A , W){
  qweibull(p, shape = W + 5*A + W*A, scale = W + 5*A - 0.2*A*W)
}
```

```
#They are the same
true_inverse(0.5, 1, 10)
```

```
## [1] 12.8108
```

```
est_cdf_inverse(0.5, 1, 10)
```

```
## [1] 12.8108
```

```
#Inverse transform (CDF) sampling
n_samples <- 100
A = 1
W = 10
#draws n samples from condiitonal dist given A and W
do_sample <- function(n, A, W){
  #Draw uniform random variables and feed into inverse CDF
  est_cdf_inverse(runif(n), A = A, W= W)
}

#simulated data summary stat
samples <- do_sample(n_samples, A , W)
mean(samples)
```

```
## [1] 12.748
```

```
sd(samples)
```

```
## [1] 0.6327257
```

```
# simulate from rweibull
samples <- rweibull(n_samples, shape = W + 5*A + W*A, scale = W + 5*A - 0.2*A*W)

mean(samples)
```

```
## [1] 12.80633
```

```
sd(samples)
```

```
## [1] 0.5648252
```

```
#As expected, the two samples have similar means and variances
#As they are sampling from the same distribution.
```

```
#Now, lets use our "do_sample" function to simulate the W A T C data.
# We will use the parametric form of the censoring for simplicity. Though the previous work can be appl
# The only changes are in the "T" node
D <- DAG.empty()
D <- D +
  # This node represents a sample of W's from our empirical distribution
  node("W", distr = "remp") +
    # This node represents a sample of A's from our conditional distribution of A given the previously
  node("A", distr = "rTrtment", value = W, prob =  prob_map) +
   # This node represents a drawn survival time from a parametric form of the conditional survival dist
  node("T", distr = "do_sample", A = A, W = W) +
  # Same as above but for censoring
    node("C", distr = "rweibull", shape = W + 5*A + W*A, scale = W + 5*A - max(0.205*A*W, 0))+
  # For simplicity, lets make the times discrete by rounding up
  node("Tdiscrete", distr = "rconst", const = ceiling(T)) +
```

```r
    node("Cdiscrete", distr = "rconst", const = ceiling(C)) +
  #Now we extract Ttilde and Delta
  node("Ttilde", distr = "rconst", const = min(Tdiscrete, Cdiscrete)) +
  node("Delta", distr = "rconst", const = as.numeric(Ttilde  == Tdiscrete) )


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

```
## ...automatically assigning order attribute to some nodes...

## node W, order:1

## node A, order:2

## node T, order:3

## node C, order:4

## node Tdiscrete, order:5

## node Cdiscrete, order:6

## node Ttilde, order:7

## node Delta, order:8

## The node A expression(s): W.
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how t
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how t

## max(cbind_mod(0.205 * A * W, 0))
## min(cbind_mod(Tdiscrete, Cdiscrete))
## as.numeric(Ttilde == Tdiscrete)
```

```r
#Now we simulate the data by drawing n = 200 samples
dat <- sim(setD, n = 200)
```

```
## simulating observed dataset from the DAG object
## The node A expression(s): W.
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how t
##  One of the formulas evaluated to unknown data type. Make sure the distribution function knowns how t

## max(cbind_mod(0.205 * A * W, 0))
## min(cbind_mod(Tdiscrete, Cdiscrete))
## as.numeric(Ttilde == Tdiscrete)
```

```r
head(dat)
```

```
##   ID         W A         T         C Tdiscrete Cdiscrete Ttilde Delta
## 1  1 20.222432 0 19.794488 19.210557        20        20     20     1
## 2  2 12.631978 1 15.480901 14.344579        16        15     15     0
## 3  3 12.514024 0 12.352374 10.462044        13        11     11     0
## 4  4 17.323425 0 18.139418 17.744051        19        18     18     0
## 5  5  7.339862 0  6.944741  7.608381         7         8      7     1
## 6  6 18.729595 1 20.469917 19.792758        21        20     20     0
```

```r
#Now we can extract our simulated observed data
observed_data <- dat[, c("W", "A", "Ttilde", "Delta")]
head(observed_data)
```

```
##           W A Ttilde Delta
## 1 20.222432 0     20     1
## 2 12.631978 1     15     0
## 3 12.514024 0     11     0
## 4 17.323425 0     18     0
## 5  7.339862 0      7     1
## 6 18.729595 1     20     0
```

```
# We now have our survival data using a custom conditional survvial fit.
```

Simulation of survival times via hazard estimates for censoring and survival

```
time_points = 1:10
hazard_f <- function(t, A, W){
  #some hazard of t, A and W
  plogis(-2 - 0.2*A + W/100)
}
hazard_f <- Vectorize(hazard_f)

hazard_g <- function(t, A, W){
  #some hazard of t, A and W
  plogis(-6)
}
hazard_g <- Vectorize(hazard_f)
D <- DAG.empty()
D <- D +
  node("W", distr = "runif", min =1, max = 25) +
  node("A", distr = "rbinom", size = 1, prob = 0.2 + W/50 ) +
  node("dNt", t = time_points,  distr = "rbern",  p = hazard_f(t, A, W) , EFU  = T)+  node("dAt", t = t

# Lets draw 200 samples
setD <- set.DAG(D, vecfun = c("hazard_f", "hazard_g"))
```

```
## ...automatically assigning order attribute to some nodes...

## node W, order:1

## node A, order:2

## node dNt_1, order:3

## node dAt_1, order:4

## node dNt_2, order:5

## node dAt_2, order:6

## node dNt_3, order:7

## node dAt_3, order:8

## node dNt_4, order:9

## node dAt_4, order:10

## node dNt_5, order:11

## node dAt_5, order:12

## node dNt_6, order:13
## node dAt_6, order:14
## node dNt_7, order:15
## node dAt_7, order:16
## node dNt_8, order:17
## node dAt_8, order:18
## node dNt_9, order:19
## node dAt_9, order:20
## node dNt_10, order:21
## node dAt_10, order:22
## [1] "current list of user-defined vectorized functions: hazard_f,hazard_g"
```

```r
dat <- sim(setD, n = 200)
```

```
## simulating observed dataset from the DAG object
```

```r
head(dat)
```

```
##   ID         W A dNt_1 dAt_1 dNt_2 dAt_2 dNt_3 dAt_3 dNt_4 dAt_4 dNt_5 dAt_5
## 1  1  4.500224 0     0     0     0     0     1    NA    NA    NA    NA    NA
## 2  2 10.994375 0     1    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 3  3 19.816788 1     0     0     0     0     0     0     0     0     0     0
## 4  4 11.797239 0     0     0     0     0     0     1    NA    NA    NA    NA
## 5  5  1.137216 0     0     0     1    NA    NA    NA    NA    NA    NA    NA
## 6  6 10.546724 1     1    NA    NA    NA    NA    NA    NA    NA    NA    NA
##   dNt_6 dAt_6 dNt_7 dAt_7 dNt_8 dAt_8 dNt_9 dAt_9 dNt_10 dAt_10
## 1    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA
## 2    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA
## 3     0     0     0     0     0     0     0     0      0      0
## 4    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA
## 5    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA
## 6    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA
```

```r
dNt_data <- dat[, grep( "dNt", colnames(dat))]
survival_times <- apply(dNt_data,1, function(row){
  result = time_points[!is.na(row) & row ==1]
  if(length(result)==0){
    return(NA)
  }
  return(result)
})

dAt_data <- dat[, grep( "dAt", colnames(dat))]
censor_times <- apply(dAt_data,1, function(row){
  result = time_points[!is.na(row) & row ==1]
  if(length(result)==0){
    return(max(time_points))
  }
  return(result)
})
```

```r
#NA means censoring happened first
head(data.frame(cbind(survival_times, censor_times)))
```

```
##   survival_times censor_times
## 1              3           10
## 2              1           10
## 3             NA           10
## 4             NA            3
## 5              2           10
## 6              1           10
```