20251003_Progress_Report_XX

Xiaoni Xu

2025-10-06

```
# rmarkdown::render("20251003_Progress_Report_XX.Rmd", clean = TRUE, envir = new.env(parent = em
ptyenv()))
```

```
library(SimDesign)
library(timereg)
library(tidyr)
library(dplyr)
library(mice)
library(splitstackshape)
library(foreach)
library(data.table)
library(survival)
library(mvtnorm)
library(parallel)

knitr::opts_chunk$set(eval = FALSE, warning = FALSE, message = FALSE)
```

Setup and data preparation

This step loads the dataset, samples 10000 rows (instead of 2000 rows from the beginning), and creates mediator names. The column names in the simulated data were changed to fit the med_longitudinal function.

- Created mediator aliases `M_1` , `M_2` and a baseline `M` column so the outcome model can use `const(M)` .
- Created models for the bootstrap sample:

Mediator models: `M_1 ~ E + C` , `M_2 ~ E + M_1 + C` (linear).

Dropout models: `D1 ~ E + M_1 + C` , `D2 ~ E + M_2 + C` (logistic).

Outcome model: Additive Aalen survival model with
`Surv(tstart,tstop,endpt) ~ const(E) + const(M) + const(C)` .

- Standardized IDs (`id_unique` , `idno`) for `tmerge` and later subsetting.
- Properly constructed reshape(…, varying=list(M_cols, TS_cols)) with one list entry per time-varying variable

```
# Load data
load("simData.mimick.SHS.RDa")

set.seed(2)

# Sample to 10,000 rows
simData <- simData[sample(1:nrow(simData), size = 10000, replace = FALSE), ]

# Create mediator aliases
simData[, M_1 := M1]
simData[, M_2 := M2]

# Counting-process helpers and unique ID for tmerge
simData[, `:=`(
  tstart = 0,
  tstop  = time_to_event,
  time.since.first.exam_1 = 0,
  time.since.first.exam_2 = time_to_event,
  id_unique = .I
)]

# Make sure it's a data.frame where needed downstream
simData <- as.data.frame(simData)

# Quick peek
str(simData[, c("E","C","M_1","M_2","tstart","tstop","id_unique")])
```

Models fitted per bootstrap resample

I refitteds mediator, dropout, and outcome models on a bootstrap resample, which is based on the existing codes.

```
# One illustrative iteration (set eval=TRUE to run)
ind  <- sample(seq_len(nrow(simData)), replace = TRUE)
data <- simData[ind, , drop = FALSE]

# IDs for tmerge and selection
data$id_unique <- seq_len(nrow(data))
data$idno      <- if ("id" %in% names(data)) data$id else data$id_unique

# Ensure baseline M exists for const(M) in the Aalen model
if (!("M" %in% names(data))) data$M <- data$M_1

# Mediator and dropout models
M1 <- lm(M_1 ~ E + C, data = data)
L1 <- glm(D1 ~ E + M_1 + C, data = data, family = binomial())
M2 <- lm(M_2 ~ E + M_1 + C, data = data)
L2 <- glm(D2 ~ E + M_2 + C, data = data, family = binomial())

# Counting-process outcome data with time-varying mediator injected via tdc()
df <- survival::tmerge(
  data1 = data,
  data2 = data,
  id    = id_unique,
  endpt = event(time_to_event, eventHappened)
)

# Long form mediator with measurement times
df_tv <- reshape(
  as.data.frame(data),
  direction = "long",
  varying   = list(
    M    = c("M_1", "M_2"),
    time = c("time.since.first.exam_1", "time.since.first.exam_2")
  ),
  v.names = c("M", "time.since.first.exam"),
  times   = c(1, 2),
  idvar   = "id_unique"
)
df_tv <- df_tv[order(df_tv$id_unique, df_tv$time), ]

# Inject time-dependent covariate
df <- survival::tmerge(
  data1 = df,
  data2 = df_tv,
  id    = id_unique,
  M     = tdc(time.since.first.exam, M)
)

# Outcome model: additive hazards
Y <- timereg::aalen(
  Surv(tstart, tstop, endpt) ~ const(E) + const(M) + const(C),
  data = df,
  resample.iid = 1,
  n.sim = 0
)

# Inputs that the mediation function expects
treat <- "E"
L     <- list(L1 = L1, L2 = L2)
Mlist <- list(M1 = M1, M2 = M2)
mvec  <- c("M_1", "M_2")
tpts  <- c("1", "2")

a      <- as.numeric(quantile(data$E, 0.25, na.rm = TRUE))
a_star <- as.numeric(quantile(data$E, 0.75, na.rm = TRUE))
```

`med_longitudinal` function

The function is solely based on the existing one.

```
med_longitudinal=function(L=NULL, M, m, Y, treat='logcocr',
                          control.value=a, treat.value=a_star,
                          data, time_points, peryr=100000){

  N  <- dim(data)[1]
  NL <- length(L)
  NM <- length(M)

  boot <- foreach(i=1:10000, .combine='rbind') %dopar%{
    ind <- sample(1:N, replace=TRUE)

    # Parameter draws for mediators
    MModel <- vector("list", NM)
    for (i in 1:NM){
      MModel[[i]] <- mvtnorm::rmvnorm(1, mean = coef(M[[i]]), sigma = vcov(M[[i]]))
    }

    # Parameter draw for outcome (Aalen)
    YModel <- mvtnorm::rmvnorm(1, mean = Y$gamma, sigma = Y$robvar.gamma)

    PredictL_a <- PredictL_astar <- PredictL_astar_a <- matrix(NA,nrow=N,ncol=NL)
    PredictM_a <- PredictM_astar <- PredictM_a_astar <- matrix(NA,nrow=N,ncol=NM)

    # Predict M1 under a and a*
    pred.data.astar.m1 <- pred.data.a.m1 <- model.frame(M[[1]])[ind,]
    pred.data.astar.m1[, treat] <- treat.value
    pred.data.a.m1[, treat]     <- control.value

    m1mat.astar <- model.matrix(terms(M[[1]]), data = pred.data.astar.m1)
    m1mat.a     <- model.matrix(terms(M[[1]]), data = pred.data.a.m1)

    PredictM_astar[,1] <- tcrossprod(MModel[[1]], m1mat.astar)
    PredictM_a[,1]     <- tcrossprod(MModel[[1]], m1mat.a)

    # Predict L1 if present
    if(NL > 0){
      pred.data.astar.l1 <- pred.data.a.l1 <- pred.data.astar.a.l1 <- model.frame(L[[1]])[ind,]
      pred.data.astar.l1[, treat]        <- treat.value
      pred.data.astar.a.l1[, treat]      <- treat.value
      pred.data.a.l1[, treat]            <- control.value
      pred.data.astar.l1[, m[1]]         <- PredictM_astar[,1]
      pred.data.a.l1[, m[1]]             <- PredictM_a[,1]
      pred.data.astar.a.l1[, m[1]]       <- PredictM_a[,1]

      PredictL_astar_a[,1] <- rbinom(N, 1, predict(L[[1]], pred.data.astar.a.l1, type='respons
e'))
      PredictL_a[,1]       <- rbinom(N, 1, predict(L[[1]], pred.data.a.l1,       type='respons
e'))
      PredictL_astar[,1]   <- rbinom(N, 1, predict(L[[1]], pred.data.astar.l1,   type='respons
e'))
    }
    # Later mediators and dropout
    if (NM > 1){
      for (i in 2:NM){
        pred.data.a.m       <- pred.data.astar.m       <- pred.data.a.astar.m       <-
          as.data.frame(matrix(nrow=N, ncol=(dim(model.frame(M[[i]]))[2]-1)))
        colnames(pred.data.a.m) <- colnames(pred.data.astar.m) <- colnames(pred.data.a.astar.m)
<-
          attr(terms(M[[i]]),"term.labels")
        names <- colnames(pred.data.a.m)[which(colnames(pred.data.a.m) %in% attr(terms(M
[[1]]),"term.labels"))]

        pred.data.a.m[, names]       <- model.frame(M[[1]])[ind,names]
        pred.data.astar.m[, names]   <- model.frame(M[[1]])[ind,names]
        pred.data.a.astar.m[, names] <- model.frame(M[[1]])[ind,names]

        pred.data.a.m[, treat]       <- control.value
        pred.data.a.astar.m[, treat] <- control.value
        pred.data.astar.m[, treat]   <- treat.value

        pred.data.a.m[, m[i-1]]       <- PredictM_a[,i-1]
        pred.data.astar.m[, m[i-1]]   <- PredictM_astar[,i-1]
        pred.data.a.astar.m[, m[i-1]] <- PredictM_a_astar[,i-1]
        if(i==2) pred.data.a.astar.m[, m[1]] <- PredictM_a[,1]

        if(NL > 1){
          m1mat.a.m         <- model.matrix(~., data = pred.data.a.m      [which(PredictL_a[,i-
1]==0),])
          m1mat.astar.m     <- model.matrix(~., data = pred.data.astar.m  [which(PredictL_astar
[,i-1]==0),])
          m1mat.a.astar.m   <- model.matrix(~., data = pred.data.a.astar.m[which(PredictL_astar_
a[,i-1]==0),])

          PredictM_a       [which(PredictL_a[,i-1]==0),i]         <- tcrossprod(MModel[[i]], m1m
at.a.m)
          PredictM_astar   [which(PredictL_astar[,i-1]==0),i]     <- tcrossprod(MModel[[i]], m1m
at.astar.m)
          PredictM_a_astar [which(PredictL_astar_a[,i-1]==0),i]   <- tcrossprod(MModel[[i]], m1m
at.a.astar.m)
        } else {
          PredictM_a[,i]       <- tcrossprod(MModel[[i]], model.matrix(~., data = pred.data.a.
m))
          PredictM_astar[,i]   <- tcrossprod(MModel[[i]], model.matrix(~., data = pred.data.asta
r.m))
          PredictM_a_astar[,i] <- tcrossprod(MModel[[i]], model.matrix(~., data = pred.data.a.as
tar.m))
        }

        if(NL > 1 & i<=NL){
          pred.data.a.l <- pred.data.astar.l <- pred.data.astar.a.l <-
            as.data.frame(matrix(nrow=N, ncol=(dim(model.frame(L[[i]]))[2]-1)))
          colnames(pred.data.a.l) <- colnames(pred.data.astar.l) <- colnames(pred.data.astar.a.
l) <-
```

```
          attr(terms(L[[i]]),"term.labels")
        names <- colnames(pred.data.a.l)[which(colnames(pred.data.a.l) %in% attr(terms(L
[[1]]),"term.labels"))]

        pred.data.a.l[, names]       <- model.frame(L[[1]])[ind,names]
        pred.data.astar.l[, names]   <- model.frame(L[[1]])[ind,names]
        pred.data.astar.a.l[, names] <- model.frame(L[[1]])[ind,names]

        pred.data.a.l[, treat]       <- control.value
        pred.data.astar.l[, treat]   <- treat.value
        pred.data.astar.a.l[, treat] <- treat.value

        pred.data.a.l[, m[i]]       <- PredictM_a[,i]
        pred.data.astar.l[, m[i]]   <- PredictM_astar[,i]
        pred.data.astar.a.l[, m[i]] <- PredictM_a_astar[,i]

        PredictL_a       [which(PredictL_a[,i-1]==0),i]         <- rbinom(length(which(Predict
L_a[,i-1]==0)), 1, predict(L[[i]], pred.data.a.l      [which(PredictL_a[,i-1]==0),], type='respo
nse'))
        PredictL_astar   [which(PredictL_astar[,i-1]==0),i]     <- rbinom(length(which(Predict
L_astar[,i-1]==0)), 1, predict(L[[i]], pred.data.astar.l  [which(PredictL_astar[,i-1]==0),], typ
e='response'))
        PredictL_astar_a [which(PredictL_astar_a[,i-1]==0),i]   <- rbinom(length(which(Predict
L_astar_a[,i-1]==0)), 1, predict(L[[i]], pred.data.astar.a.l[which(PredictL_astar_a[,i-1]==0),],
type='response'))
      }
    }
  }

  # Build counterfactual design matrices for Y
  pred.data.a.y <- pred.data.astar.y <- pred.data.astar.a.y <- pred.data.astar.a.astar.a.y <-
    data[ind, c('idno', getvarnames(Y$call)$xvar[-2], m, colnames(data)[grep('time.since.firs
t.exam', colnames(data))])]

  pred.data.a.y[, treat] <- control.value
  pred.data.astar.y[, treat] <- pred.data.astar.a.y[, treat] <- pred.data.astar.a.astar.a.y[,
treat] <- treat.value
  pred.data.a.y[, m] <- pred.data.astar.a.y[, m] <- PredictM_a
  pred.data.astar.y[, m] <- PredictM_astar
  pred.data.astar.a.astar.a.y[, m] <- PredictM_a_astar
  pred.data.astar.a.astar.a.y[, m[1]] <- PredictM_a[,1]

  M_cols  <- m
  TS_cols <- paste0("time.since.first.exam_", time_points)
  stopifnot(length(M_cols) == length(TS_cols))

  reshape_to_long <- function(df_wide){
    df_wide$id_boot <- seq_len(nrow(df_wide))
    df_long <- reshape(
      df_wide,
      direction = "long",
      varying   = list(M_cols, TS_cols),
        v.names   = c("M","time_since"),
        times     = time_points,
        timevar   = "tidx",
        idvar     = "id_boot"
      )
      df_long[order(df_long$id_boot, df_long$tidx), , drop = FALSE]
    }

    # Align to the Aalen xvar ordering
    build_X <- function(dfw){
      dfl <- reshape_to_long(dfw)
      X   <- dfl[, match(getvarnames(Y$call)$xvar, colnames(dfl)), drop = FALSE]
      model.matrix(~., data = X)[, -1, drop = FALSE]
    }

    X_DEIEM   <- build_X(pred.data.astar.a.y)
    X_TEDE_2  <- build_X(pred.data.a.y)
    X_IEMIED  <- build_X(pred.data.astar.a.astar.a.y)
    X_IEDTE_1 <- build_X(pred.data.astar.y)

    PredictY_DEIEM   <- mean(tcrossprod(YModel, X_DEIEM))
    PredictY_TEDE_2  <- mean(tcrossprod(YModel, X_TEDE_2))
    PredictY_IEMIED  <- mean(tcrossprod(YModel, X_IEMIED))
    PredictY_IEDTE_1 <- mean(tcrossprod(YModel, X_IEDTE_1))

    DE  <- mean(PredictY_DEIEM -  PredictY_TEDE_2)  * peryr
    IEM <- mean(PredictY_IEDTE_1 - PredictY_IEMIED) * peryr
    IED <- mean(PredictY_IEMIED -  PredictY_DEIEM)  * peryr
    TE  <- mean(PredictY_IEDTE_1 - PredictY_TEDE_2) * peryr

    cbind(DE, IEM, IED, TE)
  }

  # Summaries of the inner bootstrap
  DE <- quantile(boot[,1], 0.5); DE_low <- quantile(boot[,1], 0.025); DE_up <- quantile(boot[,
1], 0.975)
  IEM <- quantile(boot[,2], 0.5); IEM_low <- quantile(boot[,2], 0.025); IEM_up <- quantile(boot
[,2], 0.975)
  IED <- quantile(boot[,3], 0.5); IED_low <- quantile(boot[,3], 0.025); IED_up <- quantile(boot
[,3], 0.975)
  TE  <- quantile(boot[,4], 0.5); TE_low  <- quantile(boot[,4], 0.025); TE_up  <- quantile(boot
[,4], 0.975)

  DE_result  <- paste0(round(DE,2),  " (", round(DE_low, 2), ", ", round(DE_up,  2), ")")
  IEM_result <- paste0(round(IEM,2), " (", round(IEM_low,2), ", ", round(IEM_up, 2), ")")
  IED_result <- paste0(round(IED,2), " (", round(IED_low,2), ", ", round(IED_up, 2), ")")
  TE_result  <- paste0(round(TE,2),  " (", round(TE_low, 2), ", ", round(TE_up,  2), ")")

  Q <- round(IEM/TE*100, 2)

  list(DE=DE_result, IEM=IEM_result, IED=IED_result, TE=TE_result, Q=Q)
}
```

Results obtained from running med_longitudinal

After debugging the pipeline and ensuring all variable names matched correctly, the function ran successfully on the resampled dataset.

The estimated mediation effects (per 100,000 person-years) from one full run are shown below.

DE = 146.8186,

IEM = -19.19714,

IED = -0.5660544,

TE = 127.0554

The negative IED and IEM value suggests a negative mediation pathway in this simulated setting, while the direct effect (DE) remains the primary positive contribution to the total effect (TE).

The negative values are not expected from the dataet, which might be a result of subsampling. A new dataset is put into place to try to fix the issue.