Simple 3-state Partitioned Survival model in R

The DARTH workgroup

Developed by the Decision Analysis in R for Technologies in Health (DARTH) workgroup:

Fernando Alarid-Escudero, PhD (1)

Eva A. Enns, MS, PhD (2)

M.G. Myriam Hunink, MD, PhD (3,4)

Hawre J. Jalal, MD, PhD (5)

Eline M. Krijkamp, MSc (3)

Petros Pechlivanoglou, PhD (6)

Alan Yang, MSc (7)

In collaboration of:

1. Drug Policy Program, Center for Research and Teaching in Economics (CIDE) - CONACyT, Aguascalientes, Mexico
2. University of Minnesota School of Public Health, Minneapolis, MN, USA
3. Erasmus MC, Rotterdam, The Netherlands
4. Harvard T.H. Chan School of Public Health, Boston, USA
5. University of Pittsburgh Graduate School of Public Health, Pittsburgh, PA, USA
6. The Hospital for Sick Children, Toronto and University of Toronto, Toronto ON, Canada
7. The Hospital for Sick Children, Toronto ON, Canada

Please cite our publications when using this code:

- Jalal H, Pechlivanoglou P, Krijkamp E, Alarid-Escudero F, Enns E, Hunink MG. An Overview of R in Health Decision Sciences. Med Decis Making. 2017; 37(3): 735-746. https://journals.sagepub.com/doi/abs/10.1177/0272989X16686559
- Krijkamp EM, Alarid-Escudero F, Enns EA, Jalal HJ, Hunink MGM, Pechlivanoglou P. Microsimulation modeling for health decision sciences using R: A tutorial. Med Decis Making. 2018;38(3):400–22. https://journals.sagepub.com/doi/abs/10.1177/0272989X18754513
- Krijkamp EM, Alarid-Escudero F, Enns E, Pechlivanoglou P, Hunink MM, Jalal H. A Multidimensional Array Representation of State-Transition Model Dynamics. Med Decis Mak. 2020;40(2):242-248. doi:10.1177/0272989X19893973

Change `eval` to `TRUE` if you want to knit this document.

01 Load packages

```
if (!require('pacman')) install.packages('pacman'); library(pacman) # use this package to conveniently
# load (install if required) packages from CRAN
p_load( "dplyr", "devtools", "gems", "flexsurv", "survminer", "survHE", "ggplot2", "msm", "igraph", "ms
#install_github("DARTH-git/darthtools", force = TRUE) #Uncomment if there is a newer version
p_load_gh("DARTH-git/darthtools")
```

02 Load functions

```
# No function needed
```

03 Input model parameters

```
v_names_states <- c("Healthy", "Sick", "Dead")  # state names
n_states  <- length(v_names_states)  # No of states
n_i       <- 2000                    # number of simulations
c_l       <- 1 / 12                  # cycle length (a month)
n_t       <- 30                      # number of years (20 years)
times     <- seq(0, n_t, c_l)        # the cycles in years
set.seed(2020)                       # set the seed
```

Create a transition probability matrix with all transitions indicated and numbered.

```
m_P_diag <- matrix(NA, n_states, n_states, dimnames = list(v_names_states, v_names_states))
m_P_diag["Healthy", "Sick"]  <- 1
m_P_diag["Healthy", "Dead"]  <- 2
m_P_diag["Sick"   , "Dead"]  <- 3

layout.fig <- c(2,1)
plotmat(t(m_P_diag), t(layout.fig), self.cex = 0.5, curve = 0, arr.pos = 0.76,
        latex = T, arr.type = "curved", relsize = 0.85, box.prop = 0.8,
        cex = 0.1, box.cex = 0.7, lwd = 1)
```

The following chink generates a hypothetical dataset from a Randomized Controlled trial (RCT). The two outcomes are Sickness (or Progression) and Death. We generate four data structures:

-true_data includes the true trajectory of the patients in the RCT, assuming that we have followed them until their end of life

-sim_data includes the observed data within the RCT period. Patients could be censored anytime during the trial or at 5 years of follow-up (max follow-up in the trial)

-status includes information regarding whether the event occurred or not

- OS_PFS_data include the data as they would typically be presented in the results of an RCT (a PFS and OS outcome with the corresponding event (0, 1) information)

```
n_pat       <- 550                        # cohort size
n_years     <- 30                         # number of years
generate    <- gen_data(n_pat, n_years)   # generates true, censored and OS/PFS data
true_data   <- generate$true_data         # stores the true data
sim_data    <- generate$sim_data          # stores the censored data
status      <- generate$status            # stores the censoring status
OS_PFS_data <- generate$OS_PFS_data       # store the OS / PFS structured data

# Show the data structure
head(true_data)
head(sim_data)
head(status)
head(OS_PFS_data)
```

04 Analysis

Showcasing the use of packages `survival`, `flexsurv`.

```
fit_KM <- survfit(Surv(time = OS_time, event = OS_status) ~ 1, data = OS_PFS_data,
                  type ="fleming-harrington")
plot(fit_KM, mark.time = T)

# a prettier way of plotting!!
ggsurvplot(
  fit_KM,
  data = OS_PFS_data,
  size = 1,                 # change line size
  palette = c("orange2"),   # custom color palettes
  conf.int = TRUE,          # Add confidence interval
  pval = TRUE,              # Add p-value
  risk.table = TRUE,        # Add risk table
  risk.table.height = 0.25, # Useful to change when you have multiple groups
  ggtheme = theme_bw(),     # Change ggplot2 theme
  xlab = 'Time in days',    # Change X-axis label
  title    = "Survival curve for Progression-Free Survival (PFS)",
  subtitle = "Based on Kaplan-Meier estimates"
)
```

04.1 Partitioned Survival model

Fit parametric survival models.

```
# R package flexsurv allows parametric fitting of curves
fit_weib <- flexsurvreg(Surv(time = OS_time, event = OS_status) ~ 1, data = OS_PFS_data,
                        dist = "weibull")
plot(fit_weib)
```

- Fit all parametric models to the data and extract the AIC/BIC.

- Select the one with the most appropriate fit
- Repeat for PFS and OS

As a reminder, AIC stands for Akaike information criterion and is a measure of goodness of fit. AIC is a method that helps us understand which model fits well on the observed portion of the data. This might not necessarily mean that this model is the most appropriate to use when predicting future values. We use AIC comparative and so the best model is the one that has the lowest AIC among all the other models. So A lower AIC value indicates a better fit.

```
fit_PFS <- fit.fun(time = "PFS_time", status = "PFS_status", data = OS_PFS_data,
                   extrapolate = TRUE, times = times)
fit_OS  <- fit.fun(time = "OS_time", status  = "OS_status" , data = OS_PFS_data,
                   extrapolate = TRUE, times = times)

# Check AIC of each model to assess goodness-of-fit
GoF_PFS <- data.frame(AIC = fit_PFS$AIC, BIC = fit_PFS$BIC)
GoF_OS  <- data.frame(AIC = fit_OS$AIC,  BIC = fit_OS$BIC)

# Select best-fitting models
choose_PFS <- rownames(GoF_PFS)[which.min(GoF_PFS$AIC)]
choose_OS  <- rownames(GoF_OS)[which.min(GoF_OS$AIC)]
```

Fit a partitioned survival model

```
# construct a partitioned survival model out of the chosen models
m_M_PSM <- partsurv(pfs_survHE = fit_PFS,
                    os_survHE  = fit_OS,
                    choose_PFS = choose_PFS,
                    choose_OS  = choose_OS,
                    time = times,
                    v_names_states = v_names_states)
head(m_M_PSM$trace)
```

04.2 MultiState modeling method 1

- Fit all parametric multi-state models simultaneously.

```
# The existing functions in R require the data in a long rather than a wide format
# convert the data in a way that flexsurv understands using the mstate package
data_long       <- msprep(time = sim_data, status = status, trans = m_P_diag)
data_long$trans <- as.factor(data_long$trans) # convert trans to a factor

data_long$from  <- case_when(data_long$from == 1 ~ "Healthy",
                             data_long$from == 2 ~ "Sick",
                             data_long$from == 3 ~ "Dead")

data_long$to    <- case_when(data_long$to == 1 ~ "Healthy",
                             data_long$to == 2 ~ "Sick",
                             data_long$to == 3 ~ "Dead")
#
# # fit all parametric multistate models simultaneously to the data and extract the AIC/BIC
# # Select the one with the lowest AIC
```

```
# fits <- fit.mstate(time ="time", status = "status", trans, data = data_long,
#                    times = times, extrapolate = T)
# # Check AIC of each model to assess goodness-of-fit
# GoF_MSM1 <- data.frame(AIC = fits$AIC, BIC = fits$BIC)
#
# # Choose best-fitting model
# best.fit <- fits$Loglogistic
```

- Construct a DES model out of the simultaneously fitted multistate model.

```
# Construct a DES model out of the simultaneously fitted multistate model
# DES_data <- sim.fmsm(best.fit, start = 1, t = n_years, trans = tmat, M = n_i)
# m_M_DES  <- trace.DES(DES_data, n_i  = n_i , times = times, tmat = tmat)
# head(m_M_DES)
```

04.3 MultiState modeling method 2

- Multistate models can be fitted independently for each transition.

```
# Multistate models can be fitted independently for each transition. This is more flexible!
# Create subsets for each transition
data_HS <- subset(data_long, trans == 1)
data_HD <- subset(data_long, trans == 2)
data_SD <- subset(data_long, trans == 3)

# fit independent models for each transition and pick the one with the lowest AIC
fit_HS <- fit.fun(time = "time", status = "status", data = data_HS, times = times,
                  extrapolate = T)
fit_HD <- fit.fun(time = "time", status = "status", data = data_HD, times = times,
                  extrapolate = T)
fit_SD <- fit.fun(time = "time", status = "status", data = data_SD, times = times,
                  extrapolate = T)

best.fit_HS <- fit_HS$model.objects$models$`log-Normal`
best.fit_HD <- fit_HD$model.objects$models$`Gompertz`
best.fit_SD <- fit_SD$model.objects$models$`Royston-Parmar`
```

- A microsimulation can be fitted instead of a DES.
- It's more computationally expensive but it provides more freedom to the modeller.
- For the Microsimulation to be run, we need transition probabilities per unit of time.

TASK: figure out make.surv to run PA

```
# Extract transition probabilities from the best fitting models
p_HS <- trans_prob(surv_prob(best.fit_HS))
p_HD <- trans_prob(surv_prob(best.fit_HD))
p_SD <- trans_prob(surv_prob(best.fit_SD))

# everyone starts in the "Healthy" state and therefore has not spent time in "Sick"
v_M_init  <- rep("Healthy", times = n_i)
v_Ts_init <- rep(0, n_i)  # a vector with the time of being Sick at the start of the model
```

```r
# function that generates the transition probabilities per cycle
Probs <- function(M_t, v_Ts, t) {
  # Arguments:
    # M_t: health state occupied by at cycle t (character variable)
    # v_Ts: vector with the duration of being Sick
    # t:     current cycle
  # Returns:
    # transition probabilities for that cycle

  # create matrix of state transition probabilities
  m_p_t           <- matrix(0, nrow = n_states, ncol = n_i)
  # give the state names to the rows
  rownames(m_p_t) <-  v_names_states

  # update m_p_t with the appropriate probabilities
  # transition probabilities when Healthy
  m_p_t[, M_t == "Healthy"] <- rbind( (1 - p_HD[t]) * (1 - p_HS[t]),
                                      (1 - p_HD[t]) *      p_HS[t],
                                                           p_HD[t])
  # transition probabilities when Sick
  m_p_t[, M_t == "Sick"]    <- rbind(0,
                                     1 - p_SD[v_Ts],
                                         p_SD[v_Ts])
  # transition probabilities when Dead
  m_p_t[, M_t == "Dead"]    <- rbind(0, 0, 1)
  return(t(m_p_t))
}
```

04.3.1 Run Microsimulation

```r
MicroSim <- function(n_i,  seed = 1) {
  # Arguments:
  # n_i:     number of individuals
  # seed:    default is 1

  set.seed(seed) # set the seed

  # m_M is used to store the health state information over time for every individual
  times     <- seq(0, n_t, c_l)  # the cycles in years

  m_M  <-  matrix(nrow = n_i, ncol = length(times) ,
                  dimnames = list(paste("ind" , 1:n_i, sep = " "),
                                  paste("year", times, sep = " ")))

  m_M[, 1] <- v_M_init           # initial health state for individual i
  v_Ts     <- v_Ts_init          # initialize time since illness onset for individual i

  # open a loop for time running cycles 1 to n_t
  for (t in 1:(length(times) - 1)) {
    # calculate the transition probabilities for the cycle based on health state t
    m_p <- Probs(m_M[, t], v_Ts, t)
```

```
    # sample the current health state and store that state in matrix m_M
    m_M[, t + 1]  <- samplev(m_p)

    # update time since illness onset for t + 1
    v_Ts <- ifelse(m_M[, t + 1] == "Sick", v_Ts + 1, 0)

    # Display simulation progress
    if(t %in% seq(1,(length(times)),10)) { # display progress every 10%
      cat('\r', paste(round(t/length(times)*100,0), "% done", sep = " "))
    } else if (t == (length(times)-1)) {cat('\r', paste("100% done"))}

  } # close the loop for the time points

  # store the results from the simulation in a list
  results <- list(m_M = m_M)

  return(results)  # return the results

} # end of the MicroSim function

# Run the simulation model
Micro_data <- MicroSim(n_i, seed = 1)

# create the microsimulation trace
m_M_Micro <- t(apply(Micro_data$m_M, 2, function(x) table(factor(x, levels = v_names_states,
                                                                 ordered = TRUE))))
m_M_Micro <- m_M_Micro / n_i    # calculate the proportion of individuals
colnames(m_M_Micro) <- v_names_states
rownames(m_M_Micro) <- paste("Cycle", times, sep = " ")
head(m_M_Micro)
```

05 Compare all methods

```
# Calculate trace for the real data
m_M_data <- transitionProbabilities(generate$cohort, times = times)@probabilities

# visually compare all methods
matplot(times, m_M_data, type = 'l', lty = 1, col = 1, ylab = "proportion of cohort", xlab = "Time", ma
#matlines(times, m_M_DES,   col = 2, lty = 1)
matlines(times, m_M_Micro, col = 2, lty = 1)
matlines(times, m_M_PSM$trace, col = 3, lty = 1)
legend("right", c("True Data", "Microsim",  "PSM"),
       col = 1:3, lty = rep(1, 3), bty = "n")
```