Microsimulation Sick-Sicker model with time dependency with PSA

Includes individual characteristics: age, age dependent mortality, individual treatment effect modifer, state-residency for the sick (S1) state, increasing change of death in the first 6 year of sickness

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. 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 Making. 2020 Online first. https://doi.org/10.1177/0272989X19893973

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

```
rm(list = ls())      # clear memory (removes all the variables from the workspace)
```

01 Load packages

```
if (!require('pacman')) install.packages('pacman'); library(pacman)
# load (install if required) packages from CRAN
p_load("here", "devtools", "dplyr", "scales", "ellipse", "ggplot2", "lazyeval", "igraph", "truncnorm",
# load (install if required) packages from GitHub
# install_github("DARTH-git/dampack", force = TRUE) # Uncomment if there is a newer version
# install_github("DARTH-git/darthtools", force = TRUE) # Uncomment if there is a newer version
p_load_gh("DARTH-git/dampack", "DARTH-git/darthtools")
```

02 Load functions

```
# No functions needed
```

03 Input model parameters

```
set.seed(1)  # set the seed

# Model structure
n_t   <- 30                       # time horizon, 30 cycles
n_i   <- 100000                   # number of simulated individuals
v_n   <- c("H", "S1", "S2", "D")  # the model states names
n_states <- length(v_n)           # the number of health states
d_r   <- 0.03                     # discount rate of 3% per cycle
v_dwe <- v_dwc <- 1 / ((1 + d_r) ^ (0:n_t))    # discount weight
v_names_str <- c("no treatment", "treatment")  # strategy names
n_str <- length(v_names_str)      # number of strategies

# Event probabilities (per cycle)
# Annual transition probabilities
# (all non-probabilities are conditional on survival)
p_HS1   <- 0.15   # probability of becoming sick when healthy
p_S1H   <- 0.5    # probability of recovering to healthy when sick
p_S1S2  <- 0.105  # probability of becoming sicker when sick

# Annual probabilities of death
# load age dependent probability
p_mort   <- read.csv("mortProb_age.csv")
# load age distribution
dist_Age <- read.csv("MyPopulation-AgeDistribution.csv")
```

```r
# probability to die in S1 by cycle (is increasing)
p_S1D    <- c(0.0149, 0.018, 0.021, 0.026, 0.031, rep(0.037, n_t - 5))
p_S2D    <- 0.048  # probability to die in S2

# Cost inputs
c_H     <- 2000    # cost of one cycle in the healthy state
c_S1    <- 4000    # cost of one cycle in the sick state
c_S2    <- 15000   # cost of one cycle in the sicker state
c_D     <- 0       # cost of one cycle in the dead state
c_Trt   <- 12000   # cost of treatment (per cycle)

# Utility inputs
u_H     <- 1       # utility when healthy
u_S1    <- 0.75    # utility when sick
u_S2    <- 0.5     # utility when sicker
u_D     <- 0       # utility when dead
u_trt   <- 0.95    # utility when sick and being treated
```

04 Sample individual level characteristics

04.1 Static characteristics

```r
v_x     <- runif(n_i, min = 0.95, max = 1.05) # treatment effect modifier at baseline
# sample from age distribution an initial age for every individual
v_age0  <- sample(x = dist_Age$age, prob = dist_Age$prop, size = n_i, replace = TRUE)
```

04.2 Dynamic characteristics

```r
# Specify the initial health state of the individuals
# everyone begins in the healthy state (in this example)
v_M_init  <- rep("H", times = n_i)
v_Ts_init <- rep(0, n_i)  # a vector with the time of being sick at the start of the model
```

04.3 Create a dataframe with the individual characteristics

```r
# create a data frame with each individual's
# ID number, treatment effect modifier, age and initial time in sick state
df_X <- data.frame(ID = 1:n_i, x = v_x, Age = v_age0, n_ts = v_Ts_init, M_init = v_M_init)
head(df_X) # print the first rows of the dataframe
```

05 Define Simulation Functions

05.1 Probability function

The `Probs` function updates the transition probabilities of every cycle is shown below.

```r
Probs <- function(M_t, df_X, t) {
  # Arguments:
    # M_t:  health state occupied by individual i at cycle t (character variable)
    # df_X: data frame with individual characteristics data
    # 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)
  rownames(m_p_t) <-  v_n  # give the state names to the rows

  # lookup baseline probability and rate of dying based on individual characteristics
  p_HD_all <- inner_join(df_X, p_mort, by = c("Age"))
  p_HD     <- p_HD_all[M_t == "H", "p_HD"]

  # update the m_p with the appropriate probabilities
  # (all non-death probabilities are conditional on survival)
  # transition probabilities when healthy
  m_p_t[, M_t == "H"]  <- rbind((1 - p_HD) * (1 - p_HS1),
                                (1 - p_HD) * p_HS1,
                                       0,
                                    p_HD)
  # transition probabilities when sick
  m_p_t[, M_t == "S1"] <- rbind((1 - p_S1D[df_X$n_ts]) * p_S1H,
                                (1 - p_S1D[df_X$n_ts]) * (1 - ( p_S1H + p_S1S2)),
                                (1 - p_S1D[df_X$n_ts]) * p_S1S2,
                                     p_S1D[df_X$n_ts])

  # transition probabilities when sicker
  m_p_t[, M_t == "S2"] <- rbind(0, 0, 1 - p_S2D, p_S2D)
  # transition probabilities when dead
  m_p_t[, M_t == "D"]  <- rbind(0, 0, 0, 1)

  return(t(m_p_t))
}
```

05.2 Cost function

The `Costs` function estimates the costs at every cycle.

```r
Costs <- function (M_t, Trt = FALSE) {
  # Arguments:
    # M_t: health state occupied by individual i at cycle t (character variable)
    # Trt: is the individual being treated? (default is FALSE)
  # Returns:
    # costs accrued in this cycle
  c_t <- 0                                  # by default the cost for everyone is zero
  c_t[M_t == "H"]  <- c_H                   # update the cost if healthy
  c_t[M_t == "S1"] <- c_S1 + c_Trt * Trt    # update the cost if sick conditional on treatment
  c_t[M_t == "S2"] <- c_S2 + c_Trt * Trt    # update the cost if sicker conditional on treatment
  c_t[M_t == "D"]  <- c_D                   # update the cost if dead
```

```
  return(c_t)
}
```

05.3 Health outcome function

The `Effs` function to update the utilities at every cycle.

```
Effs <- function (M_t, df_X, Trt = FALSE, cl = 1) {
  # Arguments:
    # M_t: health state occupied by individual i at cycle t (character variable)
    # df_X: data frame with individual characteristics data
    # Trt:  is the individual treated? (default is FALSE)
    # cl:   cycle length (default is 1)
  # Returns:
    # QALYs accrued this cycle

  u_t <- 0                                  # by default the utility for everyone is zero
  u_t[M_t == "H"]  <- u_H                   # update the utility if healthy
  u_t[M_t == "S1" & Trt == FALSE] <- u_S1   # update the utility if sick
  # update the utility if sick but on treatment (adjust for individual effect modifier)
  u_t[M_t == "S1" & Trt == TRUE]  <- u_trt * df_X$x[M_t == "S1"]
  u_t[M_t == "S2"] <- u_S2                  # update the utility if sicker
  u_t[M_t == "D"]  <- u_D                   # update the utility if dead

  QALYs <-  u_t * cl  # calculate the QALYs during cycle t
  return(QALYs)
}
```

05.4 The Microsimulation function

```
MicroSim <- function(n_i, df_X, Trt = FALSE, seed = 1) {
  # Arguments:
    # n_i: number of individuals
    # df_X: data frame with individual characteristics data
    # Trt: is this the individual receiving treatment? (default is FALSE)
    # seed: seed for the random number generator, default is 1
  # Returns:
    # results: data frame with total cost and QALYs

  set.seed(seed) # set the seed

  n_states <- length(v_n) # the number of health states

  # create three matrices called m_M, m_C and m_E
  # number of rows is equal to the n_i, the number of columns is equal to n_t
  # (the initial state and all the n_t cycles)
  # m_M is used to store the health state information over time for every individual
  # m_C is used to store the costs information over time for every individual
  # m_E is used to store the effects information over time for every individual
```

```r
  m_M <- m_C <- m_E <-  matrix(nrow = n_i, ncol = n_t + 1,
                               dimnames = list(paste("ind"  , 1:n_i, sep = " "),
                                               paste("cycle", 0:n_t, sep = " ")))

    m_M [, 1] <- as.character(df_X$M_init) # initial health state at cycle 0 for individual i

    # calculate costs per individual during cycle 0
    m_C[, 1]  <- Costs(m_M[, 1], Trt)
    # calculate QALYs per individual during cycle 0
    m_E[, 1]  <- Effs (m_M[, 1], df_X, Trt)

    # open a loop for time running cycles 1 to n_t
    for (t in 1:n_t) {
      # calculate the transition probabilities for the cycle based on  health state t
      m_P <- Probs(m_M[, t], df_X, t)
      # check if transition probabilities are between 0 and 1
      check_transition_probability(m_P, verbose = TRUE)
      # check if checks if each of the rows of the transition probabilities matrix sum to one
      check_sum_of_transition_array(m_P, n_states = n_i, n_t = n_t, verbose = TRUE)
      # sample the current health state and store that state in matrix m_M
      m_M[, t + 1]  <- samplev(m_P)
      # calculate costs per individual during cycle t + 1
      m_C[, t + 1]  <- Costs(m_M[, t + 1], Trt)
      # calculate QALYs per individual during cycle t + 1
      m_E[, t + 1]  <- Effs(m_M[, t + 1], df_X, Trt)

      # update time since illness onset for t + 1
      df_X$n_ts <- if_else(m_M[, t + 1] == "S1", df_X$n_ts + 1, 0)
      # update the age of individuals that are alive
      df_X$Age[m_M[, t + 1] != "D"]  <- df_X$Age[m_M[, t + 1] != "D"] + 1

      # Display simulation progress
      if(t/(n_t/10) == round(t/(n_t/10), 0)) { # display progress every 10%
        cat('\r', paste(t/n_t * 100, "% done", sep = " "))
      }

    } # close the loop for the time points

  # calculate
  tc <- m_C %*% v_dwc  # total (discounted) cost per individual
  te <- m_E %*% v_dwe  # total (discounted) QALYs per individual
  tc_hat <- mean(tc)   # average (discounted) cost
  te_hat <- mean(te)   # average (discounted) QALY
  # store the results from the simulation in a list
  results <- list(m_M = m_M, m_C = m_C, m_E = m_E, tc = tc , te = te, tc_hat = tc_hat,
                  te_hat = te_hat)

  return(results)  # return the results

} # end of the MicroSim function

# By specifying all the arguments in the `MicroSim()` the simulation can be started
# In this example the outcomes are of the simulation are stored in the variables `outcomes_no_tr` and `
```

06 Run Microsimulation

```
# Run the simulation for both no treatment and treatment options
outcomes_no_trt  <- MicroSim(n_i, df_X, Trt = FALSE, seed = 1)
outcomes_trt     <- MicroSim(n_i, df_X, Trt = TRUE,  seed = 1)
```

07 Visualize results

```
options(scipen = 999) # disable scientific notation in R
# No treatment
plot(density(outcomes_no_trt$tc), main = paste("Total cost per person"), xlab = "Cost ($)")
plot(density(outcomes_no_trt$te), main = paste("Total QALYs per person"), xlab = "QALYs")
plot_m_TR(outcomes_no_trt$m_M)  # health state trace
# Treatment
plot(density(outcomes_trt$tc), main = paste("Total cost per person"), xlab = "Cost ($)")
plot(density(outcomes_trt$te), main = paste("Total QALYs per person"), xlab = "QALYs")
plot_m_TR(outcomes_trt$m_M)     # health state trace
```

08 Cost-Effectiveness Analysis

```
# store the mean costs of each strategy in a new variable C (vector of costs)
v_C <- c(outcomes_no_trt$tc_hat, outcomes_trt$tc_hat)
# store the mean QALYs of each strategy in a new variable E (vector of effects)
v_E <- c(outcomes_no_trt$te_hat, outcomes_trt$te_hat)

# use dampack to calculate the ICER
df_cea <- calculate_icers(cost       = v_C,
                          effect     = v_E,
                          strategies = v_names_str)
df_cea

# Plot the CEA frontier
plot(df_cea, effect_units = "Quality of Life")
```

09 Probabilistic Sensitivity Analysis (PSA)

```
# Function that generates random sample for PSA
gen_psa <- function(n_sim = 1000, seed = 1){
  # Arguments:
    # n_statesim: number of PSA simulations
    # seed: seed: seed for the random number generator, default is 1
  # Returns:
    # df_psa: data frame of sampled parameter values
```

```r
  set.seed(seed) # set a seed to be able to reproduce the same results

  df_psa <- data.frame(
    # Transition probabilities (per cycle)  - see exercise Table II
    # (all non-death probabilities are conditional on survival)
    p_HS1   = rbeta(n_sim, 30, 170),  # probability to become sick when healthy
    p_S1H   = rbeta(n_sim, 60, 60) ,  # probability to become healthy when sick
    p_S1S2  = rbeta(n_sim, 84, 716),  # probability to become sicker when sick
    p_S2D   = rbeta(n_sim, 22, 434),  # probability to die in S2

    # Cost vectors with length n_sim
    # cost of remaining one cycle in state H
    c_H     = rgamma(n_sim, shape = 100, scale = 20),
    # cost of remaining one cycle in state S1
    c_S1    = rgamma(n_sim, shape = 177.8, scale = 22.5),
    # cost of remaining one cycle in state S2
    c_S2    = rgamma(n_sim, shape = 225, scale = 66.7),
    # cost of treatment (per cycle)
    c_trt   = rgamma(n_sim, shape = 73.5, scale = 163.3),
    # cost of being in the death state
    c_D     = 0,

    # Utility vectors with length n_sim
    u_H    = rbeta(n_sim, shape1 = 200, shape2 = 3),   # utility when healthy
    u_S1   = rbeta(n_sim, shape1 = 130, shape2 = 45),  # utility when sick
    u_S2   = rbeta(n_sim, shape1 = 230, shape2 = 230), # utility when sicker
    u_D    = 0                                       , # utility when dead
    u_trt  = rbeta(n_sim, shape1 = 300, shape2 = 15),  # utility when being treated
    lb_eff = 0.95,                                    # lower bound of effect modifier
    ub_eff = 1.05                                     # upper bound of effect modifier
    )

  return(df_psa)
}
# Try it
gen_psa(10)

# Decrease number of individuals since PSA takes a lot of time
n_i <- 1000

# update Sample individual level characteristics

# Dynamic characteristics
# Specify the initial health state of the individuals
# everyone begins in the healthy state (in this example)
v_M_init  <- rep("H", n_i)  # a vector with the initial health state for all individuals

# Number of PSA simulations
n_sim <- 100

# Generate PSA input dataset
df_psa_input <- gen_psa(n_sim = n_sim)
# First six observations
```

```r
head(df_psa_input)

## Histogram of parameters
# Make sure the Plots window is large enough to plot all the histograms
ggplot(melt(df_psa_input, variable.name = "Parameter"), aes(x = value)) +
       facet_wrap(~Parameter, scales = "free") +
       geom_histogram(aes(y = ..density..)) +
       theme_bw(base_size = 16)

# Initialize data frames with PSA output
# Data frame of costs
df_c <- as.data.frame(matrix(0,
                      nrow = n_sim,
                      ncol = n_str))
colnames(df_c) <- v_names_str
# Data frame of effectiveness
df_e <- as.data.frame(matrix(0,
                      nrow = n_sim,
                      ncol = n_str))
colnames(df_e) <- v_names_str
```

09.1 Load function of microsimulation model

```r
source("Function_Microsim_Sick-Sicker_time.R")
# Test microsimulation function
calculate_ce_out(df_psa_input[1,])
```

09.2 Run microsimulation model on each parameter set of PSA input dataset

```r
start.time <- Sys.time() # track computation time
for(i in 1:n_sim){
  df_ce_psa <- calculate_ce_out(df_psa_input[i, ]) # run model
  df_c[i, ] <- df_ce_psa$Cost   # take the cost from the PSA run and store in df_c
  df_e[i, ] <- df_ce_psa$Effect # take the QALY from the PSA run in store in df_e
  # Display simulation progress
  if(i/(n_sim/10) == round(i/(n_sim/10),0)) { # display progress every 10%
    cat('\r', paste('            ', 'Overall progress: ', i/n_sim * 100, "% done",
                    sep = " "))
  }
}
elapsed.time <- Sys.time() - start.time # total computation time
elapsed.time

### CreaTe PSA object for dampack
l_psa <- make_psa_obj(cost = df_c,
                      effectiveness = df_e,
                      parameters = df_psa_input,
                      strategies = v_names_str)
```

09.3 Cost Effectiveness Analysis

Vector with willingness-to-pay (WTP) thresholds your considering and would like to have in your plot.

```
v_wtp <- seq(0, 300000, by = 10000)
```

09.3.1 ICER

```
# use dampack to calculate the ICER
calculate_icers(cost       = df_ce_psa$Cost,
                effect     = df_ce_psa$Effect,
                strategies = df_ce_psa$Strategy)
```

09.3.2 Cost-Effectiveness Acceptability Curves (CEAC) and Frontier (CEAF)

```
out_ceaf <- ceac(v_wtp, l_psa)
plot(out_ceaf)
```

09.3.3 Cost-Effectiveness Scatter plot

```
plot(l_psa)
```

09.4 Expected value of perfect information (EVPI)

```
evpi <- calc_evpi(wtp = v_wtp, psa = l_psa) # calculate EVPI
plot(evpi, effect_units = "QALY") # EVPI plot
```