An alternative representation of state-transition model dynamics - Sick-Sicker case study

Appendix of 'Krijkamp EM, Alarid-Escudero F, Enns EA, Hunink MGM, Pechlivanoglou P, Jalal HJ. An alternative representation of state-transition model dynamics'.

2019-04-29

Sick-Sicker model

In the Sick-Sicker model, we simulate a hypothetical cohort of 25-year-old individuals over a lifetime (or reaching age 100-years old) using 75 annual cycles, represented with `n.t`. The cohort start in the "Healthy" health state (denoted H). Healthy individuals are at risk of developing the illness, at which point they would transition to the first stage of the disease (the "Sick" health state, denoted S1). Individuals that become sick incur a one-time utility decrement of 0.01 (`du.HS1`), disutility of transitioning from H to S1) and a one-time cost of $1,000 (`ic.HS1`) that reflect the acute impacts of developing the illness. Sick individuals are at risk of further progressing to a more severe stage (the Sicker health state, denoted S2), which is constant in this case example. There is a chance that individuals in the Sick state eventually recover and return back to the Healthy state. However, once an individual reaches the Sicker health state, they cannot recover; that is, the probability of transitioning to the "Sick" or "Healthy" states from the Sicker state is zero. Individuals in the "Healthy" state face background mortality that is age-specific (i.e., time-dependent). Sick and Sicker individuals face an increased mortality in the form of a hazard rate ratio (HR) of 3 and 10 times, respectively, on the background mortality rate. Sick and Sicker individuals also experience increased health care costs and reduced QoL compared to healthy individuals. Once simulated individuals die, they transition to the Dead state (denoted D), where they remain. When an individual dies, they incur a one-time cost of $2,000 (`ic.D`) that reflects the acute care that might be received immediately preceding death. The state-transition diagram of the Sick-Sicker model is shown in Figure 1. The evolution of the cohort is simulated in one-year discrete-time cycles. Both costs and QALYs are discounted at an annual rate of 3%.

01 Initial setup

We start with loading the packages and functions needed.

01.2 External parameters

In the external parameter set up we specify the staring age of the cohort, the number of cycles, the names of the health states and the discount rate used for costs and QALYs. The age specific mortality rate for those in the healthy state are based on the US overall mortality data from the Human Mortality Database. The base-case parameters are combined in a list using the `f.generate_init_params()` function.

```r
#### 01.2.1 General setup ####
n.age.init  <- 25  # age of starting cohort
n.t         <- 75  # time horizon, number of cycles
v.age.names <- n.age.init:(n.age.init + n.t - 1) # vector with age names
v.n <- c("H", "S1", "S2", "D") # vector with the 4 health states of the model:
# Healthy (H), Sick (S1), Sicker (S2), Dead (D)
n.states <- length(v.n) # number of health states
d.c <- 0.03 # discount rate for costs
d.e <- 0.03 # discount rate for QALYs
v.dwc <- 1 / ((1 + d.e) ^ (0:(n.t))) # vector with discount weights for costs
v.dwe <- 1 / ((1 + d.c) ^ (0:(n.t))) # vector with discount weights for QALYs
```

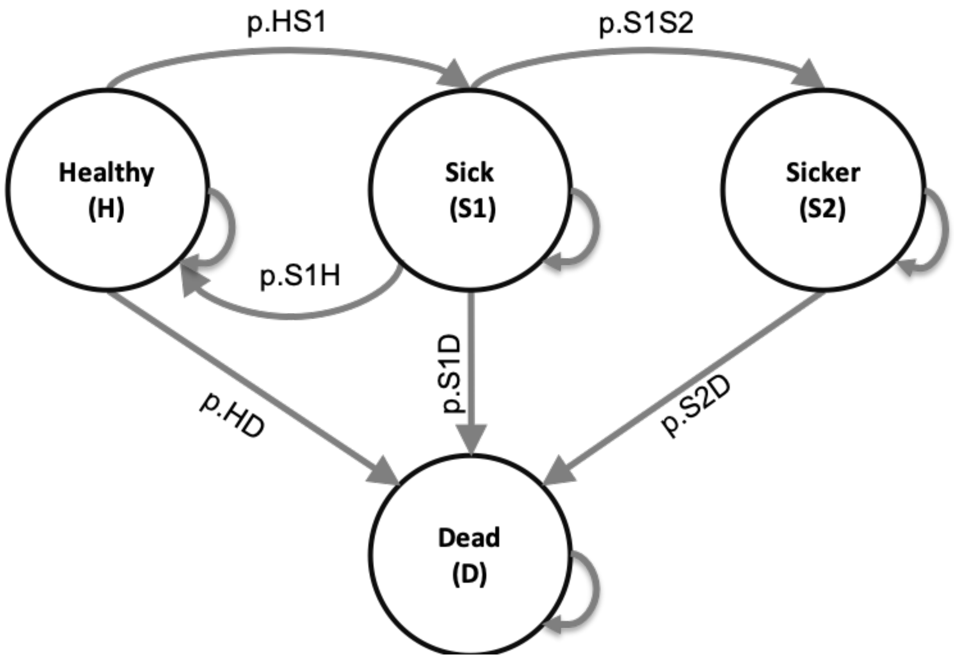


Figure 1: Sick-Sicker

```
#### 01.2.2 All-cause age-, sex- and race- (ASR) specific mortality ####
df.r.asr <- read.csv("../data/01_all-cause-mortality-USA-2015.csv")
v.r.asr  <- df.r.asr %>%
  dplyr::select(Total) %>%
  as.matrix()                # vector with mortality rates

#### 01.2.3 Generate initial set of base-case external parameters ####
v.params.init <- f.generate_init_params()
## Create name of parameters
v.names.params <- names(v.params.init)
```

02 Define and initialize matrices and vectors

In this section we initialize the matrices and vectors used for storing the data. The transition probability matrix is initialized and filled for the current cycle using the function `f.create_transition_prob_matrix` with arguments `v.params` and `t`. The first part of the function calculates the age-specific transition probabilities for the current cycle `t`. In the second part is uses these parameters togehter with the parameters stored in `v.params.init` to fill the transition probability matrix. For the initiation of cycle 0 we set the time argument to 1, since `R` starts indexing at with 1. The next step is to initialize the state vector `v.m0`. In the Sick-Sicker model all individuals start in the Healthy state. This state vector is used to inform the first row of initialized the cohort trace matrix `m.P`.

Equation 1

```
#### 02.1 Transition probability matrix ####
# matrix m.P at the first cycle
f.create_transition_prob_matrix(v.params = v.params.init, t = 1)
```

```
##             H        S1        S2           D
## H  0.8489865 0.1500000 0.0000000 0.001013486
## S1 0.5000000 0.3919626 0.1050000 0.003037378
## S2 0.0000000 0.0000000 0.9899112 0.010088764
## D  0.0000000 0.0000000 0.0000000 1.000000000
```

```
#### 02.2 Initial state vector ####
# the cohort start in the Healthy state
v.m0 <- c(H = 1, S1 = 0, S2 = 0, D = 0)
v.m0
```

```
##  H S1 S2  D
##  1  0  0  0
```

```
#### 02.3 Cohort trace
## Create the Markov cohort trace matrix m.M capturing the proportion of the cohort
# in each state at each cycle

m.M <- matrix(0,  # initialize cohort trace
              nrow = (n.t + 1), ncol = n.states,
              dimnames = list(0:n.t, v.n))

m.M[1, ] <- v.m0   # store the initial state vector
```

Now we specified all parameters for the general set up, we specified our input parameters, initialized all structures and fill the transition probability matrix `m.P` and the first row of our Cohort trace `m.M`. This allows us to start running the Markov model.

03 Traditional cohort trace approach

In this section we show how we can run the Markov model for all cycles. The calculation shown in Equation 2 needs to be performed for all cycles. Therefore, we create a loop starting at `t = 1` until `t = n.t`. Since our transition probabilities are depending on the age of the individuals in the cohort (e.g. we use age dependent mortality rates), we need to update the transition probability matrix `m.P` every cycle. This cycle dependent matrix is used for the matrix multiplication, specified in `R` with `%*%`, with the cohort trace, `m.M[t, ]` to fill the next row of the `m.M[t + 1, ]`. #### Equation 2 - 4

```
for(t in 1:n.t){  # loop through the number of cycles
  # create the transition probability matrix for the current cycle
  m.P <- f.create_transition_prob_matrix(v.params = v.params.init, t = t)
  # estimate the state vector for the next cycle (t + 1)
  m.M[t + 1, ] <- m.M[t, ] %*% m.P   # Equation 2
}
```

When printing the first six rows of `m.M` we see that everyone stars in the Healthy state and over time the cohort transitions towards the other three health states. Until cycle 5, the proportion in `S1` is increasing after which it starts decreasing, while the proportion in `S2` and `D` is increasing towards the end of the model. We now ran our model using the Matrix approach and have information about state occupation at each cycle. This information allows us to apply state rewards (e.g. `c.Healthy`, `u.Healthy`. `u.S1` etc.), but it is not possible to include the transition rewards (e.g. `ic.HS1`, `du.HS1` and `ic.D`). In order to include these rewards, we need to know when individuals made the transition. Therefore, in the next section we will explain the Array approach which makes it possible to include these rewards.

```
head(round(m.M, 3)) # show the first six lines of the Markov cohort trace
```

```
##       H    S1    S2     D
## 0 1.000 0.000 0.000 0.000
## 1 0.849 0.150 0.000 0.001
## 2 0.796 0.186 0.016 0.002
## 3 0.769 0.192 0.035 0.004
## 4 0.749 0.191 0.055 0.006
## 5 0.731 0.187 0.074 0.008
```

04 Array approach

The Array approach starts similar as the Matrix approach, meaning that secion 01 and 02 are identical for the two appraoches. The biggest difference between Matrix and array approach are the dimensions of the structure to store the dynamics of the cohort. While in the Matrix appoarch, we stored all information in matrix `m.M` of size `n.states` x `n.states`, in this approach we add an dimension for time resulting in an array with dimensions `n.states` x `n.states` x `n.t`. In `R` indexing start at 1, therefore, we initialize the array `a.A` using `n.state` + 1 to allow storing the results from cycle 0 until cycle `n.t`. The initial state vectors `v.m0` is used to inform the initial cycle of the array.

```r
a.A <- array(0, dim = c(n.states, n.states, n.t + 1),
             dimnames = list(v.n, v.n, 0:n.t)) # initialize array
diag(a.A[, , 1]) <- v.m0 # store the initial state vector in the diagonal of A
```

Equation 5 -10

We can now run the model using the Array appraoch. The function `f.create_transition_prob_matrix` is the same for both the Matrix and Array approach and needs to be calculated for each cycle. This `m.P` for the current cycle is multiplied with array `a.A` using element-wise multiplication `*`. The information about all transitions dynamics is stored in the next cycle in `a.A`.

```r
a.A[, , 1]
```

```
##    H S1 S2 D
## H  1  0  0 0
## S1 0  0  0 0
## S2 0  0  0 0
## D  0  0  0 0
```

```r
# run the model
for(t in 1:n.t){                         # loop through the number of cycles
  # create the transition probability matrix for the current cycle
  m.P <- f.create_transition_prob_matrix(v.params = v.params.init, t = t)
#### Equation 4    ####
  a.A[, , t + 1] <- colSums(a.A[, , t]) * m.P  # fill array A for t + 1
}
```

Equation 7

To get an idea about how the information in `a.A` looks like we print it first three cycles. Like in the transition probability matrix `m.P`, the rows specify in which health state the individual started at the beginning of the cycle, while the columns inform you about where individuals transitioned to. In cycle 0 everyone started in the healthy states. At cycle 1 we can see that 0.849 of the cohort stayed healthy, 0.15 transitioned from Healthy to Sick and `r round(a.A["H","D", "1"], 4)` died. Indeed, this looks very similar to the transition probabilities in this case example. From cycle 2 and onwards the information in `a.A` becomes more interesting. In cycle 2, we see that 0.7208 of the cohort stayed healthy, 0.1273 transitioned from Healthy towards Sick and 0.00084 died while they started out healthy. In addition, we see that 0.075 of the cohort recovered from Healthy. 0.0588 stayed Sick, 0.0158 became Sicker out of Sick and 0.00045 died from Sick. All these values sum to 1 since we are still describing what happens to the cohort over time.

```r
a.A[, , 1:3] # shown for two cycles
```

```
## , , 0
##
##    H S1 S2 D
## H  1  0  0 0
## S1 0  0  0 0
## S2 0  0  0 0
## D  0  0  0 0
##
## , , 1
##
##            H   S1 S2           D
## H  0.8489865 0.15  0 0.001013486
## S1 0.0000000 0.00  0 0.000000000
## S2 0.0000000 0.00  0 0.000000000
## D  0.0000000 0.00  0 0.000000000
##
## , , 2
##
##            H         S1      S2           D
## H  0.7207908 0.12734798 0.00000 0.000847714
## S1 0.0750000 0.05880112 0.01575 0.000448877
## S2 0.0000000 0.00000000 0.00000 0.000000000
## D  0.0000000 0.00000000 0.00000 0.001013486
```

```r
sum(a.A[, , 3]) # sum for t = 3
```

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

Equation 9

When you sum the values in a column of `a.A`, e.g. Sick, you get which proportion of the cohort was in Sick at the end of that cycle.

```r
sum(a.A[, "S1", 3]) # sum the column of S1 at t = 3
```

```
## [1] 0.1861491
```

By using the `colSums` function, summing over all columns of `a.A` we can do this for all points in time and when we transpose these results we get the traditional cohort trace `m.M`. Here names `m.M_A` to indicate it is generated via the Array approach.

```r
# calculating M from A
m.M_A <- t(colSums(a.A))   # sum over the columns of a.A and transpose
```

Since a Markov model is stochastic, these two approached should give identical resulst. We check this using the `==` function. We use rounding on 10 decimals, to avoid wrong `FALSE` results that have to do with floating point comparison issues. This means that functions allows you to test for equality with a specified difference tolerance.

4.1 Array approach for a Microsimulation

The Array appraoch is not limited to state transition cohort models. Also for Microsimulation models this is very usefull. In fact our previously published code showing how to run a microsimulation already created code to keep track on how individuals transitioned and how the cohort trace can be extracted based on that. We use the code in the `Appendix D_online_supp.R` file of the paper and load it into our environment.

The structure `TS` generated by the `MicroSim` function shows which transition an individual made at each cycle, while `TR` gives the cohort trace of the microsimulation. From `TS` one can fill array `A` as is shown below. [DARTH: Any idea on how can we make this code more efficient to create array A from the MicroSim?]

```
head(sim_no_trt$TS[1:4, 10:14])  # the transition array
```

```
##       Cycle 9 Cycle 10 Cycle 11 Cycle 12 Cycle 13
## Ind 1 "H->H"  "H->H"   "H->H"   "H->H"   "H->H"
## Ind 2 "H->H"  "H->H"   "H->H"   "H->H"   "H->H"
## Ind 3 "H->S1" "S1->S1" "S1->H"  "H->H"   "H->H"
## Ind 4 "S1->H" "H->H"   "H->H"   "H->H"   "H->S1"
```

```
head(sim_no_trt$TR)              # the cohort trace from the microsimulation
```

```
##                H      S1      S2       D
## Cycle 0 1.00000 0.00000 0.00000 0.00000
## Cycle 1 0.84466 0.15025 0.00000 0.00509
## Cycle 2 0.78981 0.18283 0.01609 0.01127
## Cycle 3 0.75742 0.18893 0.03462 0.01903
## Cycle 4 0.73396 0.18598 0.05271 0.02735
## Cycle 5 0.71449 0.17933 0.07023 0.03595
```

```
a.A_MicroSim <- array(0, dim = c(n.states, n.states, n.t + 1),
             dimnames = list(v.n, v.n, 0:n.t)) # initialize array

diag(a.A_MicroSim[, , 1]) <- v.m0 #

for (t in 2:n.t + 1){
a.A_MicroSim["H", "H",  t] <- sum((sim_no_trt$TS[, t] == "H->H") , na.rm = TRUE) / n.i
a.A_MicroSim["H", "S1", t] <- sum((sim_no_trt$TS[, t] == "H->S1"), na.rm = TRUE) / n.i
a.A_MicroSim["H", "S2", t] <- sum((sim_no_trt$TS[, t] == "H->S2"), na.rm = TRUE) / n.i
a.A_MicroSim["H", "D",  t] <- sum((sim_no_trt$TS[, t] == "H->D") , na.rm = TRUE) / n.i

a.A_MicroSim["S1", "H",  t] <- sum((sim_no_trt$TS[, t] == "S1->H") , na.rm = TRUE) / n.i
a.A_MicroSim["S1", "S1", t] <- sum((sim_no_trt$TS[, t] == "S1->S1"), na.rm = TRUE) / n.i
a.A_MicroSim["S1", "S2", t] <- sum((sim_no_trt$TS[, t] == "S1->S2"), na.rm = TRUE) / n.i
a.A_MicroSim["S1", "D",  t] <- sum((sim_no_trt$TS[, t] == "S1->D") , na.rm = TRUE) / n.i

a.A_MicroSim["S1", "H",  t] <- sum((sim_no_trt$TS[, t] == "S2->H") , na.rm = TRUE) / n.i
a.A_MicroSim["S2", "S1", t] <- sum((sim_no_trt$TS[, t] == "S2->S1"), na.rm = TRUE) / n.i
a.A_MicroSim["S2", "S2", t] <- sum((sim_no_trt$TS[, t] == "S2->S2"), na.rm = TRUE) / n.i
a.A_MicroSim["S2", "D",  t] <- sum((sim_no_trt$TS[, t] == "S2->D") , na.rm = TRUE) / n.i

a.A_MicroSim["D", "H",  t] <- sum((sim_no_trt$TS[, t] == "D->H") , na.rm = TRUE) / n.i
a.A_MicroSim["D", "S1", t] <- sum((sim_no_trt$TS[, t] == "D->S1"), na.rm = TRUE) / n.i
a.A_MicroSim["D", "S2", t] <- sum((sim_no_trt$TS[, t] == "D->S2"), na.rm = TRUE) / n.i
a.A_MicroSim["D", "D",  t] <- sum((sim_no_trt$TS[, t] == "D->D") , na.rm = TRUE) / n.i
}

a.A_MicroSim[, , 1:3]
```

```
## , , 0
##
##    H S1 S2 D
## H  1  0  0 0
## S1 0  0  0 0
## S2 0  0  0 0
## D  0  0  0 0
##
## , , 1
##
##    H S1 S2 D
## H  0  0  0 0
## S1 0  0  0 0
## S2 0  0  0 0
## D  0  0  0 0
##
## , , 2
##
##          H      S1      S2       D
## H  0.66732 0.11826 0.00000 0.00423
## S1 0.00000 0.07067 0.01922 0.00284
## S2 0.00000 0.00000 0.01540 0.00069
## D  0.00000 0.00000 0.00000 0.01127
```

05 Apply state and transition rewards

We now showed how to run a state-transition cohort model using the Array approach and how to interpret or summarize the results. In this section we demonstrate how to apply state and transition rewards. We start by initiating and filling two matrices for both costs and effects. The function `f.create_transition_reward_matrix_costs` is used to create a matrix of state and transition costs. This function is informed by the vector `v.params.init` and since our state and transition rewards are not time dependent we don't need an argument for time. The costs on the diagonal are the costs for staying one cycle in that state, while the costs off the diagonal are the costs of staying one cycle in that stated plus the transition costs. The function `f.create_transition_reward_matrix_effects` does the same but then for utilities.

In this Sick-Sicker example we use three different functions one for creating a transition probability matrix, one for creating a cost matrix and one for the effect matrix. Since these functions are based on the same information, one could decide to reduce the number of functions by combining all these steps.

```r
#### 05.1 Create reward matrices for both costs and effects ####
m.R_costs   <- f.create_transition_reward_matrix_costs(v.params = v.params.init)
m.R_effects <- f.create_transition_reward_matrix_effects(v.params = v.params.init)
```

Equation 11

These matrices look as follow. We see that staying healthy costs $2000, while someone that transitions from healthy towards sick costs makes $3000. In the effects matrix we see that an individual gets a utility of 1 assign for staying healthy, while when the individual transitions towards Sick the decrement of `ic.HS1` is included.

```r
m.R_costs    # show the reward matrix for costs
```

```
##        H    S1    S2     D
## H   2000  3000  2000  4000
## S1  4000  4000  4000  6000
## S2 15000 15000 15000 17000
## D      0     0     0     0
```

```
m.R_effects  # show the reward matrix for effects
```

```
##       H   S1   S2    D
## H  1.00 0.99 1.00 1.00
## S1 0.75 0.75 0.75 0.75
## S2 0.50 0.50 0.50 0.50
## D  0.00 0.00 0.00 0.00
```

Equation 12

In this section we create outcome array `a.Y`, one for costs `a.Y_costs` and one for effects `a.Y_effects`. These arrays show the costs and QALYs generated with each transition at each cycle. By iteratively element-wise multiplication of the reward matrices with `a.A` we can fill the outcome arrays. Again, we are now showing all these steps in a step wise approach, resulting in having a couple of loops for time. This can all be combined in one iterative process.

```
#### 05.2 Expected QALYs and Costs per cycle for each strategy ####
a.Y_costs <- a.Y_effects <- array(0, dim = c(n.states, n.states, n.t + 1),
             dimnames = list(v.n, v.n, 0:n.t))

for(t in 1:n.t){
# element-wise-multiplication of array A with the rewards matrices
a.Y_costs[, , t]   <- a.A[, , t] * m.R_costs
a.Y_effects[, , t] <- a.A[, , t] * m.R_effects
}
```

Equation 13

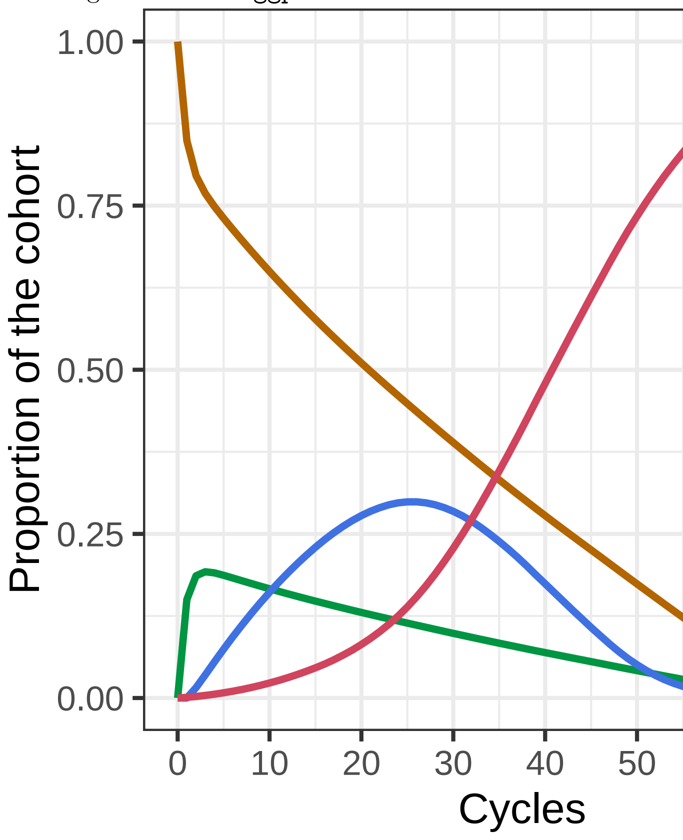
The final step is to calculated to the total expected discounted costs and QALYs. We start by calculating the expected cost and QALYs per cycle. These values, stored in the vectors `v.Costs` and `v.QALYs`, in turn are multiplied with the vector of discount weights, `v.dwc1` and `v.dwe`, respectively. This gives us the total expected discounted cost (`TC`) and QALYs (`TE`.

```
# calculate the expected costs per cycle
v.Costs <- rowSums(t(colSums(a.Y_costs)))
# calculate the expected QALYs per cycle
v.QALYs <- rowSums(t(colSums(a.Y_effects)))

TC <- t(v.Costs) %*% v.dwc   # calculate the total expected discounted costs
TE <- t(v.QALYs) %*% v.dwe   # calculate the total expected discounted QALYS

v.Results <- c(TC, TE)       # combine the total expected costs and QALYs
names(v.Results) <- c("Costs", "Effect") # name the vector
v.Results                    # print the results
```

```
##        Costs       Effect
## 115104.13376     20.37685
```

06 Plot cohort trace

The results of a cohort trace are much easier to interpret via a graph. Using the function `ggplot` can show the



proportion of the cohort in each state (y-axis) at each cycle (x-axis).