An alternative representation of state-transition model dynamics - Generic case study using a 3-state model

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

2019-02-19

Simple 3-state model

In this simple 3-state model, we simulate a hypothetical cohort of 70-year-old individuals over a lifetime using 50 annual cycles, represented with `n.t`.The healthy individuals can become sick, which makes them transition to the sick health state or they can die. Sick individuals can recover, transitioning back to healthy, stay sick or die. Each health state is associated with utilities and costs, state rewards. In addition to these state rewards, transition rewards apply. Becoming sick is associated with a disability as well as some costs. Also transitioning to dead is associated with a one-time costs. The model parameters are described in the table below.

Parameter	**R name**	**Value**
Time horizon (n_t)	`n.t`	50 years
Names of health states (n)	`v.n`	H, S, D
Annual discount rate (costs/QALYs)	`d.c/d.e`	3%
Annual transition probabilities		
- Disease onset (H to S)	`p.HS1`	0.15
- Recovery (S to H)	`p.S1H`	0.5
Annual mortality		
- All-cause mortality (H to D)	`p.HD`	0.05
- Disease specific mortality (S to D)	`p.SD`	0.10
Annual costs		
- Healthy individuals	`c.H`	$1,000
- Sick individuals	`c.S1`	$3,000
- Dead individuals	`c.D`	$0
Utility weights		
- Healthy individuals	`u.H`	1.00
- Sick individuals	`u.S`	0.60
- Dead individuals	`u.D`	0.00
Transition rewards		
- Utility decrement of healthy individuals when transitioning to S	`du.HS`	0.10
- Cost of healthy individuals when transitioning to S	`ic.HS`	$500
- Cost of dying	`ic.D`	$4,000

01 Initial setup

We start with loading the packages and functions needed.

```
#### 01.1 Load packages and functions ####
library(dplyr)    # to manipulate data
library(reshape2) # to transform data
```

```r
library(ggplot2)  # for nice looking plots
library(scales)   # for dollar signs and commas
library(tensorA)  # for tensor calculations

#### 01.1.2 Load functions ####
source("../functions/01_model-inputs_functions.R")
source("../functions/02_simulation-model_functions.R")
```

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 parameters described in the table are combined in a dataframe using the `f.generate_init_params()` function.

```r
#### 01.2.1 General setup ####
n.age.init  <- 70  # age of starting cohort
n.t         <- 50  # 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", "S", "D") # vector with the 3 health states of the model:
# Healthy (H), Sick (S), 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
v.s.init <- c(H = 1, S = 0, D = 0) # initial state vector

#### 01.2.3 Generate initial set of base-case external parameters ####
df.params.init <- f.generate_init_params()
## Create name of parameters
df.names.params <- names(df.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 argument `df.params`. The parameters stored in `df.params.init` are used 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 `s0`. In this example model all individuals start in the Healthy health state. This state vector is used to inform the first row of initialized the cohort trace matrix `m.P`.

Equation 1

```r
#### 02.1 Transition probability matrix ####
# matrix m.P at the first cycle
m.P <- f.create_transition_prob_matrix(df.params = df.params.init)

#### 02.2 Initial state vector ####
# the cohort start in the Healthy health state
s0 <- c(H = 1, S = 0, D = 0)
s0
```

```
## H S D
```

```
## 1 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, ] <- s0   # 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`. The transition probability matrix is mulitplied with the cohort trace `m.M[t, ]`, using matrix multiplication, specified in `R` with `%*%`, to fill the next row of the `m.M[t + 1, ]`. #### Equation 2

```
for(t in 1:n.t){  # loop through the number of cycles
  # 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 starts in the Healthy health state and over time the cohort transitions towards sick and dead.

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

```
##       H     S     D
## 0 1.000 0.000 0.000
## 1 0.780 0.170 0.050
## 2 0.668 0.226 0.106
## 3 0.600 0.238 0.162
## 4 0.551 0.233 0.216
## 5 0.512 0.222 0.267
```

We now ran our model using the traditional cohort trace approach and have information about state occupation at each cycle. This information allows us to apply state rewards (e.g. `c.H`, `u.H`, `u.S` etc.), but it is not possible to include the transition rewards (e.g. `ic.HS`, `du.HS` 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.

04 Array approach

The Array approach starts similar as the traditional cohort trace approach, meaning that secion 01 and 02 are identical for the two appraoches. The biggest difference between the approaches is the dimensions of the structure to store the dynamics of the cohort. While in the cohort trace appoarch, we stored all information in matrix `m.M` of size `n.states` x `n.states`, in the array 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 `s0` 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]) <- s0 # store the initial state vector in the diagonal of A
```

Equation 3 & 4

We can now run the model using the Array appraoch. The function `f.create_transition_prob_matrix` is the same for both approaches 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 S D
## H 1 0 0
## S 0 0 0
## D 0 0 0
```

```r
# run the model
for(t in 1:n.t){                    # loop through the number of cycles
#### Equation 4    ####
  a.A[, , t + 1] <- colSums(a.A[, , t]) * m.P  # fill array A for t + 1
}
```

Equation 5

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 health states. At cycle 1 we can see that the values are 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 0.78 of the cohort stayed healthy, 0.17 transitioned from Healthy towards Sick and 0.05 died. In addition, we see that 0 of the cohort recovered, 0 stayed Sick and 0.05 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 S D
## H 1 0 0
## S 0 0 0
## D 0 0 0
##
## , , 1
##
##      H    S    D
## H 0.78 0.17 0.05
## S 0.00 0.00 0.00
## D 0.00 0.00 0.00
##
## , , 2
##
##        H      S     D
## H 0.6084 0.1326 0.039
```

```
## S 0.0595 0.0935 0.017
## D 0.0000 0.0000 0.050
```

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

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

Equation 7

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.

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

```
## [1] 0.2261
```

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.

```
# 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.

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.

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

Equation 8

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.

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

```
##      H    S    D
## H 1000 1500 5000
## S 3000 3000 7000
## D    0    0    0
```

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

```
##     H   S   D
## H 1.0 0.9 1.0
## S 0.6 0.6 0.6
## D 0.0 0.0 0.0
```

Equation 9

In this section we create outcome array `a.O`, one for costs `a.O_costs` and one for effects `a.O_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.O_costs <- a.O_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.O_costs[, , t]   <- a.A[, , t] * m.R_costs
a.O_effects[, , t] <- a.A[, , t] * m.R_effects
}
```

Equation 10

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.O_costs)))
# calculate the expected QALYs per cycle
v.QALYs <- rowSums(t(colSums(a.O_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
## 20569.86143    10.40485
```

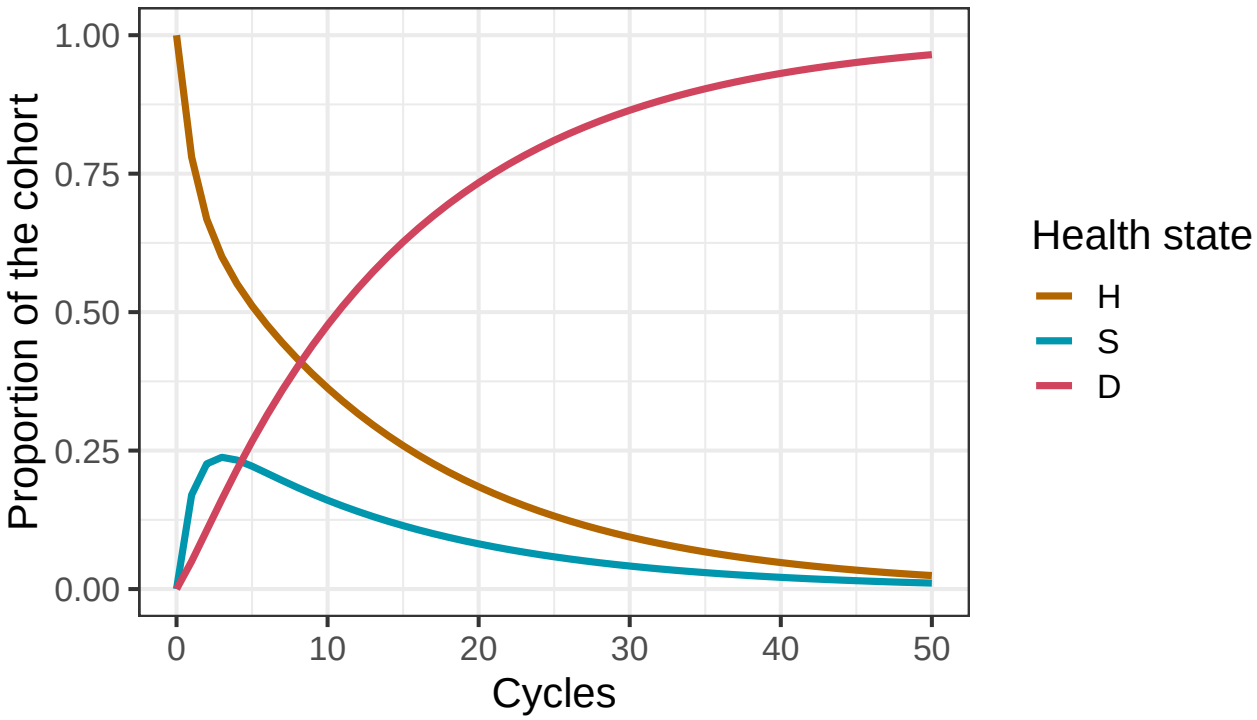

Figure 1: Cohort trace of the model

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).