Sick-Sicker case study

Supplementory material to the paper: 'A need for change! A coding framework for improving transparency in decision modeling'

DARTH workgroup

The Sick-Sicker model

In this case-study, we perform a cost-effectiveness analysis (CEA) using a previously published 4-state model called the Sick-Sicker model. (Enns et al. 2015) In the Sick-Sicker model, a hypothetical disease affects individuals with an average age of 25 years and results in increased mortality, increased treatment costs and reduced quality of life (QoL). We simulate this hypothetical cohort of 25-year-old individuals over a lifetime (i.e., reaching an age of 100 years old) using 75 annual cycles, represented with `n.t`. The cohort starts 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"). Sick individuals are at risk of further progressing to a more severe stage (the "Sicker" health state, denoted "S2"), which is a constant probability in this case-study. 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 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 expressed as a hazard rate ratio (HR) of 3 and 10, 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. Figure 1 shows the state-transition diagram of the Sick-Sicker model. The evolution of the cohort is simulated in one-year discrete-time cycles. Both costs and quality-adjusted life years (QALYs) are discounted at an annual rate of 0.03 %.

Two alternative strategies exist for this hypothetical disease: a no-treatment and a treatment strategy. Under the treatment strategy, Sick and Sicker individuals receive treatment and continue doing so until they recover or die. The cost of the treatment is additional to the cost of being Sick or Sicker for one year. The treatment improves QoL for those individuals who are Sick but has no effect on the QoL of those who are sicker. To evaluate these two alternative strategies, we perform a cost-effectiveness analysis (CEA).

We assume that most of the parameters of the Sick-Sicker model and their uncertainty have been previously estimated and are known to the analyst. However, while we can identify those who are afflicted with the illness through obvious symptoms, we can not easily distinguish those in the Sick state from the those in the Sicker state. Thus, we can not directly estimate state-specific mortality hazard rate ratios, nor do we know the transition probability of progressing from Sick to Sicker. Therefore, we calibrate the model to different epidemiological data. We internally validated the calibrated model by comparing the predicted outputs from the model evaluated at the calibrated parameters against the calibration targets.(Eddy et al. 2012, Goldhaber-Fiebert, Stout, and Goldie (2010))

As part of the CEA, we conducted different deterministic sensitivity analysis (SA), including one-way and two-way SA, and tornado plots. To quantify the effect of parameter uncertainty on decision uncertainty, we conducted a probabilistic sensitivity analysis (PSA) and reported our uncertainty analysis results with a cost-effectiveness acceptability curve (CEAC), cost-effectiveness acceptability frontier (CEAF) and expected loss curves (ELC). We also conducted a value of information (VOI) analysis to determine whether potential future research is needed to reduce parameter uncertainty. All steps of the CEA will be described using the different components of the framework.

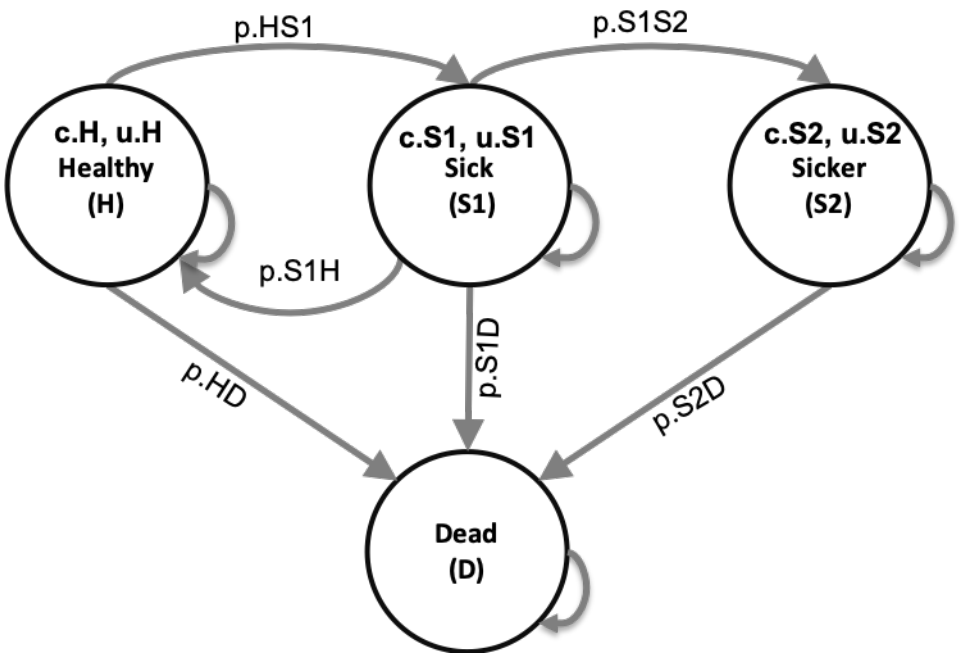

Figure 1: State-transition diagram of the Sick-Sicker model. Healthy individuals can get Sick, die or stay healthy. Sick individuals can recover, transitioning back to healthy, can die, or stay sick. Once individuals are Sicker, they stay Sicker until they die.

Set up

This report is a supplementary material meant to guide you through the `R` code of a full functional decision model we use to showcase the DARTH framework described in the manuscript titled *A need for change! A coding framework for improving transparency in decision modeling*. The code following the framework can be downloaded from the DARTH-git on GitHub (https://github.com/DARTH-git/Decision-Modeling-Framework). We recommend downloading this complete framework as a .zip file containing all directories. Unzip the folder and save it on your computer. The framework is divided into different directories, described in Table 1, that could be accessed from the RStudio project **Decision-Modeling-Framework.Rproj**. In this framework, you will find multiple directories as described in Table 1 of the main manuscript. We refer to the directory names of this framework using *italic* style. This report is created with Markdown and is located in the *reports* directory of the framework. The figures for the case-study can be found in the *figs* directory, data required to conduct some of the analyses of the different components are in the *data* directory and the `R` scripts with functions, are located in the *functions* directory. The main `R` scripts that conduct the analyses of the different components of the framework are stored in the `R` directory. In this document we do not show all the `R` code we refer to. Therefore, it is important to follow along while reading this document. To make sure you have all the required packages needed to run all the code, first install all the required packages by running the **00_general-setup.R** script in the `R` directory.

```
source("R/app0_packages-setup.R")
```

01 Define model inputs

As described in the main manuscript, in this component we declare all model input variables and set their values. The `R` script running the analysis of this component is the **01_model-inputs.R** file in the `R` directory.

The input to inform the values is divided in three categories: external, estimated, and calibrated. The majority of the Sick-Sicker model parameters are informed by external data. Only three parameter values

need to be estimated using model calibration.

In this component, we start with the general setup of the model, specifying among others the time horizon, name and number of health states, proportion of the cohort in each of the different health states at the start of the simulation and discount rates. The next step is to specify the external parameters. The initial model parameter values and `R` variable names are presented in Table 1.

Table 1: Description of the initial parameters with their `R` name and value of the Sick-Sicker model.

Parameter	**R name**	**Value**
Time horizon (n_t)	`n.t`	75 years
Names of health states (n)	`v.n`	H, S1, S2, D
Annual discount rate (costs/QALYs)	`d.c`/`d.e`	3%
Annual transition probabilities		
- Disease onset (H to S1)	`p.HS1`	0.15
- Recovery (S1 to H)	`p.S1H`	0.5
- Disease progression (S1 to S2) in the time-homogenous model	`p.S1S2`	0.105
Annual mortality		
- All-cause mortality (H to D)	`p.HD`	age-specific
- Hazard rate ratio of death in S1 vs H	`hr.S1`	3
- Hazard rate ratio of death in S2 vs H	`hr.S2`	3
Annual costs		
- Healthy individuals	`c.H`	$2,000
- Sick individuals in S1	`c.S1`	$4,000
- Sick individuals in S2	`c.S2`	$15,000
- Dead individuals	`c.D`	$0
- Additional costs of sick individuals treated in S1 or S2	`c.Trt`	$12,000
Utility weights		
- Healthy individuals	`u.H`	1.00
- Sick individuals in S1	`u.S1`	0.75
- Sick individuals in S2	`u.S2`	0.50
- Dead individuals	`u.D`	0.00
Intervention effect		
- Utility for treated individuals in S1	`u.Trt`	0.95

Age-specific background mortality for healthy individuals is represented by the US population in 2015 and obtained from the Human Mortality database. This information is stored in the **01_all-cause-mortality.csv** file in the *data* directory. Based on this .csv file a vector with mortality rates by age is created using the `f.load_mort_data` function in the **01_model-inputs_functions** script. This functions gives us the flexibility to easily import data from other coutries or years.

```
print.function(f.load_mort_data) # print the function
```

```
## function (file = "data/01_all-cause-mortality.csv")
## {
##     df.r.mort_by_age <- read.csv(file = file)
##     v.r.mort_by_age <- df.r.mort_by_age %>% dplyr::select(Total) %>%
##         as.matrix()
##     return(v.r.mort_by_age)
## }
```

An other function in the **01_model-inputs_functions** script, is the `f.load_all_parms` function. This function, which is actually using the `f.load_mort_data` function, loads all parameters for the decision model from multiple sources and creates a list that contains all parameters and their values.

```r
print.function(f.load_all_params)  # print the function
```

```
## function (file.init = "data/01_init-params.csv", file.mort = "data/01_all-cause-mortality.csv")
## {
##     df.params.init <- read.csv(file = file.init)
##     v.r.mort_by_age <- f.load_mort_data(file = file.mort)
##     l.params.all <- with(as.list(df.params.init), {
##         v.age.names <- n.age.init:(n.age.init + n.t - 1)
##         v.n <- c("H", "S1", "S2", "D")
##         n.states <- length(v.n)
##         v.s.init <- c(H = 1, S1 = 0, S2 = 0, D = 0)
##         l.params.all <- list(n.age.init = n.age.init, n.t = n.t,
##             v.age.names = v.age.names, v.n = v.n, n.states = n.states,
##             v.s.init = c(H = 1, S1 = 0, S2 = 0, D = 0), v.r.mort_by_age = v.r.mort_by_age)
##         return(l.params.all)
##     })
##     l.params.all <- c(l.params.all, df.params.init)
## }
## <bytecode: 0x7fd89b8c2430>
```

The `f.load_all_params` function is informed by the arguments `file.init` and `file.mort`. The `file.init` argument is a string with the location and name of the file with initial set of parameters. The inital parameter values for our case-study are stored in the **01__init-params.csv** file located in the *data* directory. The `f.load_all_params` function read this .csv file into the function environment as a dataframe called, `df.params.init`.

The `file.mort` argument is a string with the location and name of the file with mortality data. As described before, in our case-study this is the **01__all-cause-mortality.csv** file. Within the `f.load_all_parms` function, the `f.load_mort_data` function is used to create a vector with mortality rates from the .csv data.

After loading all the information, the `f.load_all_params` generates a list called,`l.params.all`, including all parameters for the model including the general setup parametes and the vector of mortality rates. The function also stores the dataframe `df.params.init` with the initial set of parameters in the list. This is all executed in the in the **01__model-inputs.R** script by running the code below.

```r
l.params.all <- f.load_all_params(file.init = "data/01_init-params.csv",
                                  file.mort = "data/01_all-cause-mortality.csv")
```

For the Sick-Sicker model we do not have to estimated parameters, but we d o have three parameters that need to be estimated via model calibration. In this stage of the framework, we simply set these parameters to valid "dummy" values that are compatible with the next phase of the analysis, model implementation, but are ultimately just placeholder values until we conduct the calibration phase. This means that these values will be replaced by the best-fitted calibrated values after we performed the calibration in component 3.

Using a function to create a list of base-case parameters to have all model parameters in a single object is very useful, because this object will have to be updated for the calibration and the different sensitivity analyses in components 3 and 5 of the framework, respectively. Below, we guide you through the components of the function.

02 Model implementation

In this component, we build the backbone of the decision analysis: the implementation of the model. This component is performed by the **02__simulation-model.R** script. This file itself is not very large. It simply loads some packages, sources the input from component 01, sources the function `f.decision_model` that is used the capture the dynamic process of the Sick-Sicker example, runs this function and stores the output. The output of the model is the traditional cohort trace, describing how the cohort is distributed among the

different health states over time, which is plotted at the end of this script. This trace will be used in many of the other components.

The function `f.decision_model` is defined in the **02__simulation-model__functions.R file**. As described in the paper, constructing a model as a function at this stage facilitates subsequent stages of the model development and analysis, as these processes will all call the same model function, but pass different parameter values and/or calculate different final outcomes based on the model outputs. In the next part, we will describe the code within the function.

```
print.function(f.decision_model) # print the code of the function
```

```
## function (l.params.all, verbose = FALSE)
## {
##     with(as.list(l.params.all), {
##         p.HDage <- 1 - exp(-v.r.mort_by_age[(n.age.init + 1) +
##             0:(n.t - 1)])
##         p.S1Dage <- 1 - exp(-v.r.mort_by_age[(n.age.init + 1) +
##             0:(n.t - 1)] * hr.S1)
##         p.S2Dage <- 1 - exp(-v.r.mort_by_age[(n.age.init + 1) +
##             0:(n.t - 1)] * hr.S2)
##         a.P <- array(0, dim = c(n.states, n.states, n.t), dimnames = list(v.n,
##             v.n, 0:(n.t - 1)))
##         a.P["H", "H", ] <- (1 - p.HDage) * (1 - p.HS1)
##         a.P["H", "S1", ] <- (1 - p.HDage) * p.HS1
##         a.P["H", "D", ] <- p.HDage
##         a.P["S1", "H", ] <- (1 - p.S1Dage) * p.S1H
##         a.P["S1", "S1", ] <- (1 - p.S1Dage) * (1 - (p.S1S2 +
##             p.S1H))
##         a.P["S1", "S2", ] <- (1 - p.S1Dage) * p.S1S2
##         a.P["S1", "D", ] <- p.S1Dage
##         a.P["S2", "S2", ] <- 1 - p.S2Dage
##         a.P["S2", "D", ] <- p.S2Dage
##         a.P["D", "D", ] <- 1
##         m.indices.notvalid <- arrayInd(which(a.P < 0 | a.P >
##             1), dim(a.P))
##         try(if (dim(m.indices.notvalid)[1] != 0) {
##             v.rows.notval <- rownames(a.P)[m.indices.notvalid[,
##                 1]]
##             v.cols.notval <- colnames(a.P)[m.indices.notvalid[,
##                 2]]
##             v.cycles.notval <- dimnames(a.P)[[3]][m.indices.notvalid[,
##                 3]]
##             df.notvalid <- data.frame(`Transition probabilities not valid:` = matrix(paste0(paste(v.
##                 v.cols.notval, sep = "->"), "; at cycle ", v.cycles.notval),
##                 ncol = 1), check.names = FALSE)
##             if (verbose) {
##                 message("Not valid transition probabilities")
##                 stop(print(df.notvalid), call. = FALSE)
##             }
##         })
##         valid <- apply(a.P, 3, function(x) all.equal(sum(rowSums(x)),
##             n.states))
##         if (!isTRUE(all.equal(as.numeric(sum(valid)), as.numeric(n.t)))) {
##             if (verbose) {
##                 stop("This is not a valid transition Matrix")
```

```
##             }
##         }
##         m.M <- matrix(0, nrow = (n.t + 1), ncol = n.states, dimnames = list(0:n.t,
##             v.n))
##         m.M[1, ] <- v.s.init
##         for (t in 1:n.t) {
##             m.M[t + 1, ] <- m.M[t, ] %*% a.P[, , t]
##         }
##         return(list(a.P = a.P, m.M = m.M))
##     })
## }
## <bytecode: 0x7fd89b9e6008>
```

The `f.decision_model` function is informed by the argument `l.params.all`. Via this argument we give the function a list with with all parameters of the decision model. For the Sick-Sicker model, these parameters are stored in the list `l.params.all`, which we passed into the function as shown below.

```
l.out.stm <- f.decision_model(l.params.all = l.params.all) # run the function
```

This function itself has all the mathematical equations of the decision models coded inside. It starts by calculating the age-specific transition probabilities from all non-dead states based on the vector of age-specific mortality rates `v.r.asr`. These parameters will become vectors of length `n.t`, describing the probability to die for all ages from all non-dead states.

The next part of the function, creates an array that stores the age-specific transition probability matrices in each of the third dimension. The transition probability matrix is a core component of a state-transition cohort model(Iskandar 2018). This matrix contains the probabilities of transitioning from the current health state, indicated by the rows, to the other health states, specified by the columns. Since we have age-specific transition probabilities, the transition probability matrix is different at each cycle. These probabilities are only depending on the age of the cohort, and not on other events; therefore, we can generate all matrices at the start of the model. This results in `n.t` different age-specific matrices that are stored in an array, called `a.P`, of dimensions `n.s` x `n.s` x `n.t`. After initializing the array, it is filled with the transition probability stored in the list. When running the model, we can index the correct transition probability matrix corresponding with the current age of the cohort. We then added some sanity checks to make sure that the transition matrices and the transition probabilities are valid. The transition probability matrices stored in the array `a.P`, for the first three and last cycle, are shown below.

```
l.out.stm$a.P[, , 1:3] # show the first three time-points of a.P
```

```
## , , 0
##
##            H        S1        S2           D
## H  0.8491385 0.1498480 0.0000000 0.001013486
## S1 0.4984813 0.3938002 0.1046811 0.003037378
## S2 0.0000000 0.0000000 0.9899112 0.010088764
## D  0.0000000 0.0000000 0.0000000 1.000000000
##
## , , 1
##
##            H        S1        S2            D
## H  0.8491513 0.1498502 0.0000000 0.0009985012
## S1 0.4985037 0.3938180 0.1046858 0.0029925135
## S2 0.0000000 0.0000000 0.9900597 0.0099402657
## D  0.0000000 0.0000000 0.0000000 1.0000000000
##
## , , 2
```

```
##
##            H        S1        S2           D
## H  0.8490910 0.1498396 0.0000000 0.001069428
## S1 0.4983976 0.3937341 0.1046635 0.003204853
## S2 0.0000000 0.0000000 0.9893570 0.010642959
## D  0.0000000 0.0000000 0.0000000 1.000000000
```

```
l.out.stm$a.P[, , l.params.all$n.t] # show the last
```

```
##            H        S1         S2         D
## H  0.6055199 0.1068564 0.00000000 0.2876237
## S1 0.1807584 0.1427991 0.03795926 0.6384833
## S2 0.0000000 0.0000000 0.03365849 0.9663415
## D  0.0000000 0.0000000 0.00000000 1.0000000
```

By comparing these probability matrices, we observe an increase in the probabilities of transitioning to death from all health states.

After the array is filled, the cohort trace matrix, `m.M`, of dimensions `n.t` x `n.s` is initialized. This matrix will store the state occupation at each point in time. The first row of the matrix is informed by the initial state vector `v.s.init`. For the remaining points in time, we iteratively multiply the cohort trace with the age-specific transition probability matrix corresponding to the specific cycle obtained by indexing the array `a.P` appropriately. All the outputs and relevant elements of the decision model are stored in a list, called `l.out.stm`. This list contains the array of the transition probability matrix for all cycles `t` and the cohort trace `m.M`.

```
head(l.out.stm$m.M)    # show the top part of the cohort trace
```

```
##           H        S1         S2           D
## 0 1.0000000 0.0000000 0.00000000 0.000000000
## 1 0.8491385 0.1498480 0.00000000 0.001013486
## 2 0.7957468 0.1862564 0.01568695 0.002309774
## 3 0.7684912 0.1925699 0.03501425 0.003924648
## 4 0.7484793 0.1909659 0.05478971 0.005765035
## 5 0.7306193 0.1873106 0.07413838 0.007931783
```

```
tail(l.out.stm$m.M)    # show the bottom part of the cohort trace
```

```
##              H           S1           S2         D
## 70 0.009928317 0.0022433565 2.035951e-04 0.9876247
## 71 0.007153415 0.0015935925 1.311619e-04 0.9911218
## 72 0.005058845 0.0011174473 8.674540e-05 0.9937370
## 73 0.003460336 0.0007552206 5.436484e-05 0.9957301
## 74 0.002289594 0.0004937081 3.298632e-05 0.9971837
## 75 0.001475636 0.0003151589 1.985106e-05 0.9981894
```

Using the code below, we can graphically show the model dynamics by plotting the cohort trace. Figure 2 shows the distribution of the cohort among the different health states at each time point.

Eddy, David M., William Hollingworth, J. Jaime Caro, Joel Tsevat, Kathryn M. McDonald, and John B. Wong. 2012. "Model transparency and validation: A report of the ISPOR-SMDM modeling good research practices task force-7." *Medical Decision Making* 32 (5): 733–43. doi:10.1177/0272989X12454579.

Enns, EA, LE Cipriano, CT Simons, and CY Kong. 2015. "Identifying Best-Fitting Inputs in Health-Economic Model Calibration: A Pareto Frontier Approach." *Medical Decision Making* 35 (2): 170–82. doi:10.1177/0272989X14528382.

Goldhaber-Fiebert, Jeremy D., Natasha K. Stout, and Sue J. Goldie. 2010. "Empirically evaluating

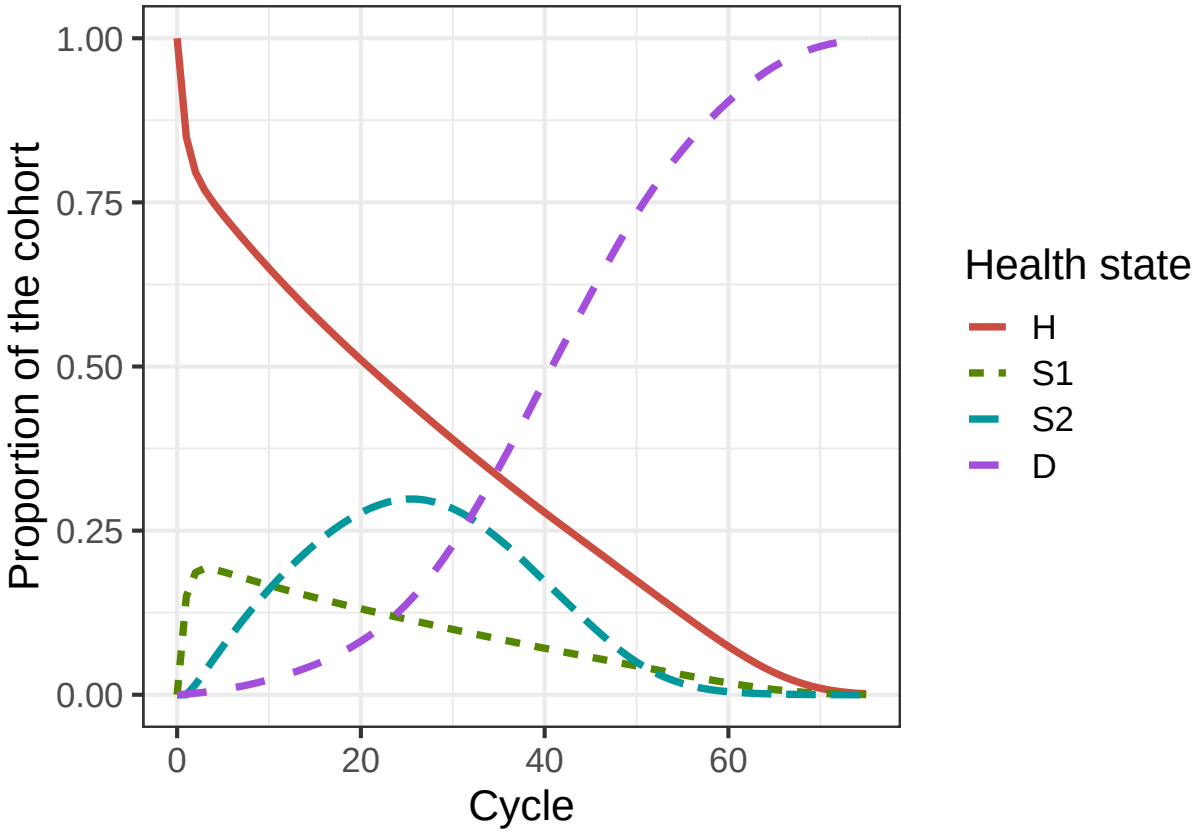

Figure 2: Cohort trace of the Sick-Sicker cohort model

decision-analytic models." *Value in Health* 13 (5): 667–74. doi:10.1111/j.1524-4733.2010.00698.x.

Iskandar, Rowan. 2018. "A theoretical foundation for state-transition cohort models in health decision analysis." *PloS One* 13 (12). Public Library of Science: e0205543–e0205543. doi:10.1371/journal.pone.0205543.