Sick-Sicker case study - as part of the framework paper

DARTH workgroup

2019-02-22

How to read this document

This document is meant to showcase our framework via a fully functional decision 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) This document is supportive material to the 'A need for change! A coding framework for improving transparency in decision modeling' paper from the DARTH workgroup, and you are therefore recommended to first read this paper before you read this document.

We start this document by explainig the case example we use, followed by expalining all the different components as described in the paper. As explained in the paper, our file organization consist of several folder. This markdown manuscript can be found in the *manuscript* folder. Our figures can be found in the *figure*folder, data is the *data* folder and when we talk about functions we use you can find them in the *functions* folder. All other `R` files can be found in the `R` script folder.

The Sick-Sicker model

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. 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"). 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 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 health states from the Sicker health 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" health 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 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.

We are asked to evaluate the cost-effectiveness of this treatment that increases QoL in one of the disease states assuming as willingness to pay of $8000.(Krijkamp et al. 2018) We also identify the uncertainty around our decision based on the CEA using sensitivity analysis. We end the showcase of our framework by performing a value of information (VoI) analysis to see if it is worth investing in extra research projects with the aim to reduce the uncertainty around our decision. All steps of the cost-effectiveness analysis will be described using the different components of the framework.

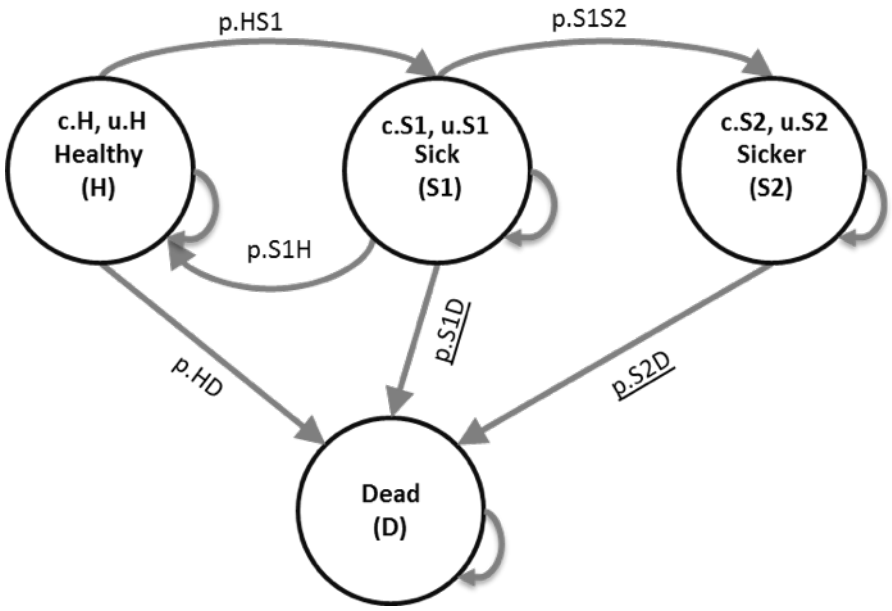


Figure 1: Sick-Sicker

01 Define model inputs

As described in the paper, in this component we declare all model input variables and set their values. The `R` script running all components of this section is the *01_model-inputs.R* file in the `R` script folder.

The input to inform the values is categorized in three categories: external, estimated, and calibrated. The majority of the Sick-Sicker model parameters are informed by external data, while 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, prevalence of the different health states at the start of the model and discount rates. The next step is to declare and inform the external parameter. The model parameter values and `R` variable names are presented in Table 1.

Table 1: Description of parameters with their `R` name and value.

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 ratio of death in S1 vs H	`hr.S1`	3
- Hazard 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

Parameter	R name	Value
- 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

The all-cause mortality of healthy individuals is age specific. We therefore load the *01_all-cause-mortality-USA-2015.csv* file, including the age-, sex- and race- (ASR) specific mortality rates for the US population in 2015 derived from the Human Mortality database.

The other external parameter values are all stored the 01_init-params.csv file in the data folder. Based on this file a base-case parameters set, an `R` data frame, of this information is generated by the `f.generate_init_params()` function. Using a function to create a basecase parameters dataframe might see complex, but it is important for the sensitivity analysis we will do in component 5 of the framework.

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

```
## function ()
## {
##     df.params.init <- read.csv(file = "data/01_init-params.csv")
##     df.params.init <- data.frame(c.H = df.params.init$c.H, c.S1 = df.params.init$c.S1,
##         c.S2 = df.params.init$c.S2, c.D = df.params.init$c.D,
##         c.Trt = df.params.init$c.Trt, u.H = df.params.init$u.H,
##         u.S1 = df.params.init$u.S1, u.S2 = df.params.init$u.S2,
##         u.D = df.params.init$u.D, u.Trt = df.params.init$u.Trt,
##         p.HS1 = df.params.init$p.HS1, p.S1H = df.params.init$p.S1H,
##         p.S1S2 = df.params.init$p.S1S2, hr.S1 = df.params.init$hr.S1,
##         hr.S2 = df.params.init$hr.S2)
##     return(df.params.init)
## }
## <bytecode: 0x7f823b437920>
```

In this code you can see that the functions start calling the .csv file and stores the values of the parameters. All of this is combined in a dataframe. The Sick-Sicker model does not have estimated parameters, but there are 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.

02 Model implementation

In this component we build the backbone of the decision analysis: the implementation of the model. In this section of the framework, all combined in the *02_simulation-model.R* `R` script, a function is used that maps model inputs to outputs, for the Markov model we are using to capture the dynamic process of the Sick-Sicker example. The *02_simulation-model.R* file itself is not very large. It is loading some packages, sources the input from component 01, sources the function that is creating the model, after which the functions is ran and the results are stored. The output of the model is the traditional cohort trace, describing how the cohort

is distributed among the different health states over time. This trace will be used in many of the other component sections.

The key component of this file, is 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 of this section we will describe the code of the function.

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

```
## function (v.params)
## {
##     with(as.list(v.params), {
##         p.HDage <- 1 - exp(-v.r.asr[(n.age.init + 1) + 0:(n.t -
##             1)])
##         p.S1Dage <- 1 - exp(-v.r.asr[(n.age.init + 1) + 0:(n.t -
##             1)] * hr.S1)
##         p.S2Dage <- 1 - exp(-v.r.asr[(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)
##             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)))) {
##             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))
```

```
##         a.A <- 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: 0x7f823e91dca0>
```

The `f.decision_model()` function is informed by just one argument, called `v.params`. This argument informs the model about the model parameters. The parameter values for our Sick-Sicker model are stored in the dataframe `df.params` and is passed into the function as an argument.

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

You probably noticed that the prefix of the argument is telling us it has to be a vector, while we pass a dataframe into the function. This is totally fine. The function is able to deal with both vectors, usefull for small models, as well as dataframes, when you have more parameter info.

The functions starts with calculating the age-specific transition probabilities based on the rates in the data frame. This means that the parameters will become vectors of length `n.t`, describing the probability do die for all ages.

The next part of the function, creates the age-specific transition probability matrices in an array. The transition probability matrices a core component of a state-transition cohort model. This matrix contains the probabilities of transitioning from the current health state, indicated by the rows, towards the new health states, specified in the columns. More information about creating these matrices is described in a paper about State-transition models using R (Alarid-Escudero et al. 2018). Since we have age specific transition probabilities, the transition probability matrix is different each cycle. Since these probabilities are only depending on the age of the cohort, and not on other events, we can generate all matrices at the start of the model. This results `n.t` matrices that we will store in an array, called `a.P`, of dimensions `n.s` x `n.s` x `n.t`. After initiation this array the array is filled based on the parameter values stored in the dataframe. When running the model, we can index the correct transition probability matrix corresponding with the current age of the cohort. The make sure we are using correct values, the functions include some code to check if both the transition probabilities them self, as well as the transition probability matrices are valid. The first three time-points of the probability array, as well as for the last cycle, are shown below.

```
## , , 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
```

```
##            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
```

Comparing these probability matrices, shows you the increased probabilities of transitioning to death form all health states towards the end.

After the array is filled, the cohort trace matrix, `m.M`, of dimensions `n.t` x `n.s` is initiated. 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 indexed from array `a.P`. The model results 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
```

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

03 Model calibration

In this component, unknown or highly uncertain model parameters are estimated by calibrating model outputs to match specified calibration targets. This proces is executed by the *03_calibration.R* file. The target data is stored in the *03_calibration-targets.RData* file. The *03_calibration.R* file include much more code compared to the `R` scripts of the previous components. Like in compontent 02, we start with loading inputs and functions. In addition, we load the calibration targets data into the `R` environment. In the next section, **03.2 Visualize targets**, we create a graph for each of our targets, survival, prevalence and the proportion who are Sicker, among all those afflicted (Sick+Sicker).

```
## Warning: package 'lhs' was built under R version 3.5.2
```

```
## Loading required package: mvtnorm
```

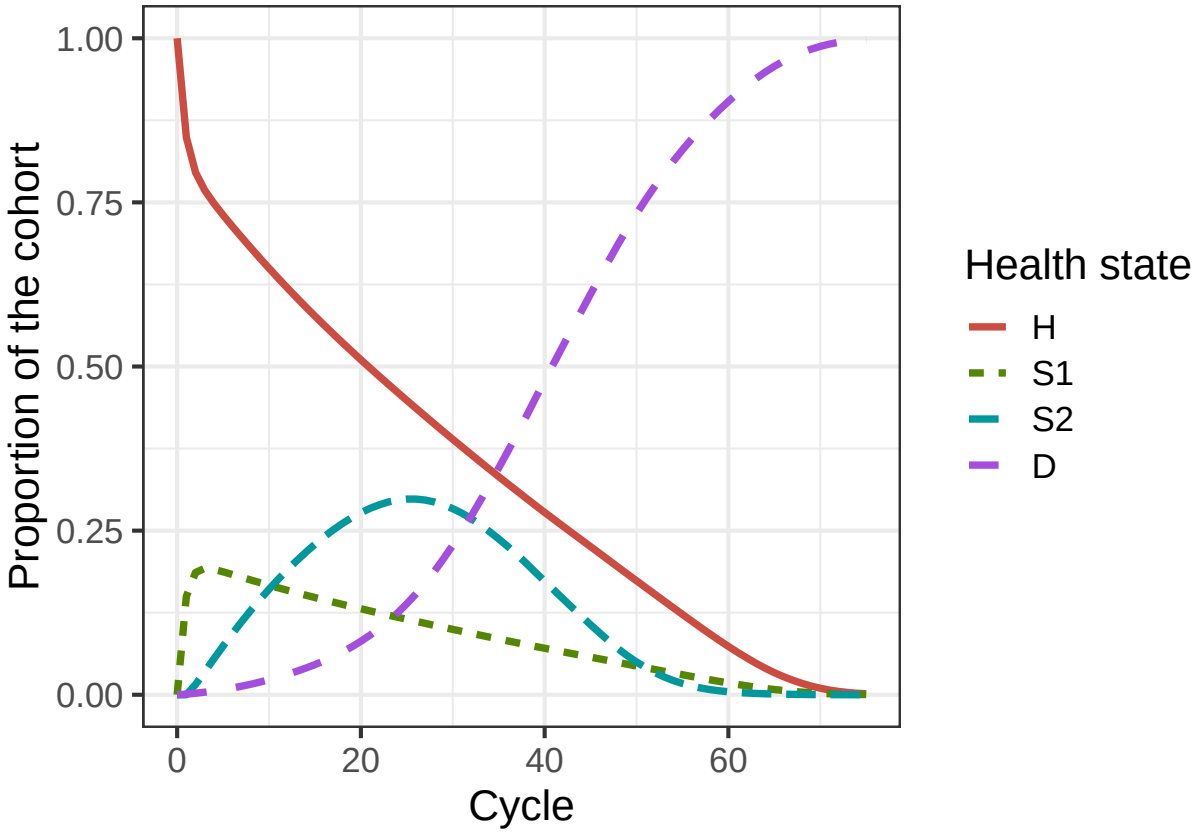
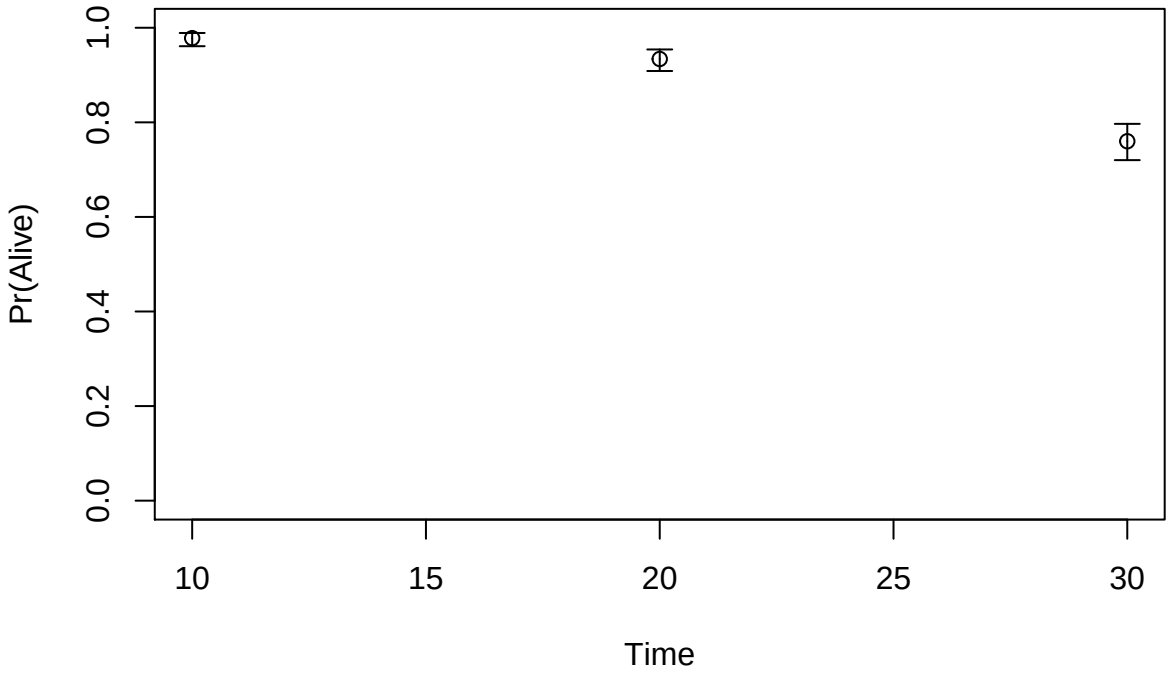
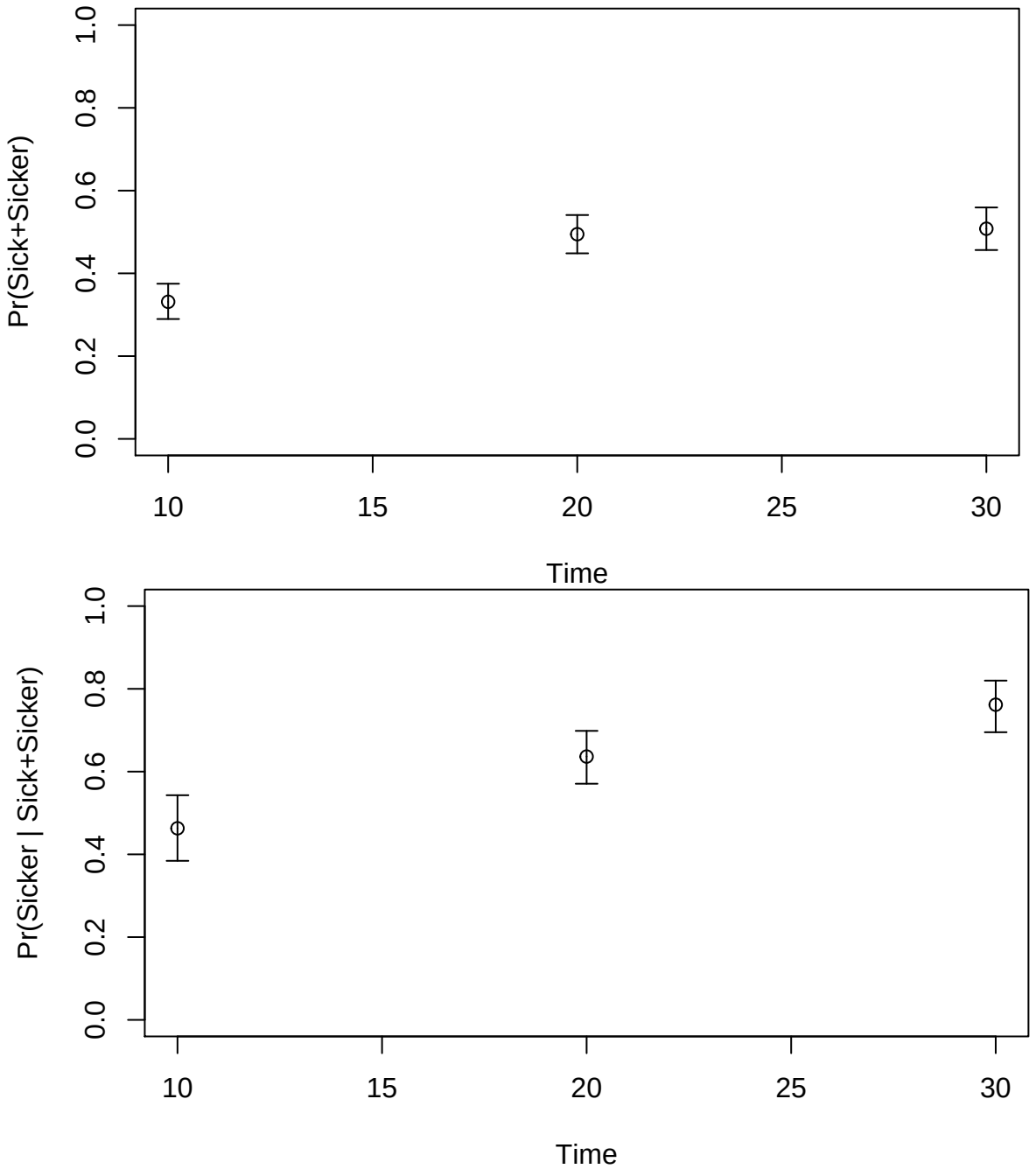


Figure 2: Cohort trace of the Sick-Sicker cohort model

```
##
## Attaching package: 'matrixStats'
## The following object is masked from 'package:dplyr':
##
##     count
##
## Attaching package: 'plotrix'
## The following object is masked from 'package:scales':
##
##     rescale
## Warning: package 'psych' was built under R version 3.5.2
##
## Attaching package: 'psych'
## The following object is masked from 'package:plotrix':
##
##     rescale
## The following objects are masked from 'package:scales':
##
##     alpha, rescale
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```





In section *03.3 Run calibration algorithms* we run # Check that it works v.params.calib = c(p.S1S2 = 0.105, hr.S1 = 3, hr.S2 = 10) f.calibration_out(v.params.calib = v.params.calib)

```
print.function(f.calibration_out) # print the funtions
```

```
## function (v.params.calib)
## {
##     v.params <- df.params.init
##     v.params["p.S1S2"] <- v.params.calib["p.S1S2"]
##     v.params["hr.S1"] <- v.params.calib["hr.S1"]
##     v.params["hr.S2"] <- v.params.calib["hr.S2"]
##     l.out.stm <- f.decision_model(v.params = v.params)
```

```
##     v.os <- 1 - l.out.stm$m.M[, "D"]
##     v.prev <- rowSums(l.out.stm$m.M[, c("S1", "S2")])/v.os
##     v.prop.S2 <- l.out.stm$m.M[, "S2"]/rowSums(l.out.stm$m.M[,
##         c("S1", "S2")])
##     l.out <- list(Surv = v.os[c(11, 21, 31)], Prev = v.prev[c(11,
##         21, 31)], PropSicker = v.prop.S2[c(11, 21, 31)])
##     return(l.out)
## }
```

This component involves both the setup of the calibration (specification of plausible ranges or prior distributions for input parameters to be calibrated, specification of calibration targets, calculation of corresponding values from model outputs and assessment of fit to targets) as well as the carrying out the calibration itself with a chosen algorithm.

We calibrated the Sick-Sicker model using a Bayesian approach using the incremental mixture importance sampling (IMIS) algorithm [Teele2006], which has been used to calibrate health policy models (Raftery2010, Menzies, Pandya, and Kim 2017, Rutter2018). Bayesian methods allow us to quantify the uncertainty in the calibrated parameters even in the presence of non-identifiability (F. Alarid-Escudero et al. 2018). We assumed a normal likelihood and uniform priors. For a more detailed description of IMIS for Bayesian calibration, different likelihood functions and prior distributions, we refer the reader to the tutorial for Bayesian calibration by Menzies et al. (Menzies, Pandya, and Kim 2017). For illustration purposes, we also calibrated the Sick-Sicker model with the Nelder-Mead algorithm (Nelder and Mead 1965), which was initialized at 100 different starting points sampled from the prior distributions of the calibrated parameters. This calibration exercise can be found in the file `app2_calibration-nelder-mead.R`.

We have external data for the calibration of three parameters from the Sick-Sicker model. This data, stored in the `03_calibration-tarted.RData` file, contains information about survival, the disease prevalence and the proportion who are sick among those afflicted (sick and sicker). The `03_calibration_functions.R` file contains two functions for the calibration process. One to generate model outputs for calibration from a parameters set called `f.calibration_out` and a Goodness of fit function for calibration from a parameter set called `f.gof`.

In the first section of the calibration we load the data, we visualize our targets and we check if the calibration function runs by using the default values of the parameters of interest. In the section 03.3.3 we run the Nelder-Mead for each starting point and we explore the best fitting input sets. The best set of parameters from the Nelder-Mead is saved in the file `03_nm-best-set.RData`.

The last part of the calibration includes internal validation of the values predicted by the model using the best set of parameters and our targets.

The figures show that our best-set of parameters is quite good for the first two parameters since the model-predicted output is quite close to our targets. Our calibrated parameters are not very good in reaching the target values for the proportion of those who are sick among those who are afflicted.

Once appropriate values, ranges, and/or distributions have been identified for calibrated parameters, these values will be used in the analysis component, as appropriate. This means that we will replace the placeholder values we created in component 1 by the calibrated values.

05c Value of information

References

Alarid-Escudero, F, EM Krijkamp, P Pechlivanoglou, EA Enns, MGM Hunink, and H Jalal. 2018. "State-transition cohort models in R, from conceptualization to implementation: A tutorial." *Medical Decision Making : An International Journal of the Society for Medical Decision Making* XX (X). SAGE

PublicationsSage CA: Los Angeles, CA: XX. http://journals.sagepub.com/doi/10.1177/0272989X16686559 http://www.ncbi.nlm.nih.gov/pubmed/28061043.

Alarid-Escudero, Fernando, Richard F. MacLehose, Yadira Peralta, Karen M. Kuntz, and Eva A. Enns. 2018. "Nonidentifiability in Model Calibration and Implications for Medical Decision Making." *Medical Decision Making* 38 (7): 810–21. doi:10.1177/0272989X18792283.

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.

Krijkamp, EM, F Alarid-Escudero, EA Enns, H Jalal, MGM Hunink, and P Pechlivanoglou. 2018. "Microsimulation Modeling for Health Decision Sciences Using R: A Tutorial." *Medical Decision Making : An International Journal of the Society for Medical Decision Making* 38 (3): 400–422. doi:10.1177/0272989X18754513.

Menzies, Nicolas A, Ankur Pandya, and Jane J Kim. 2017. "HHS Public Access." *Pharmacoeconomics* 35 (6): 613–24. doi:10.1007/s40273-017-0494-4.Bayesian.

Nelder, J. A., and R. Mead. 1965. "A Simplex Method for Function Minimization." *The Computer Journal* 7 (4): 308–13. doi:10.1093/comjnl/7.4.308.