Introduction to Building and Calibrating Simulation Models in R

Mt Hood/SMDM 2023

Kuala Lumpur, Malaysia

The DARTH Workgroup

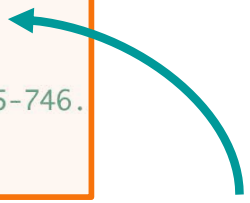
- Materials for this course were largely developed by the Decision Analysis in R for Technologies in Health (DARTH) Workgroup
- Goals: To expand knowledge in decision analysis using R and develop educational materials to empower people to construct R-based decision models.

For more information
http://www.darthworkgroup.com/

Attribution and Acknowledgement

- R code provided with this course are yours to reuse and modify

```
################################################################################
# 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
```



Acknowledgement and citation information in code headers

Cohort models

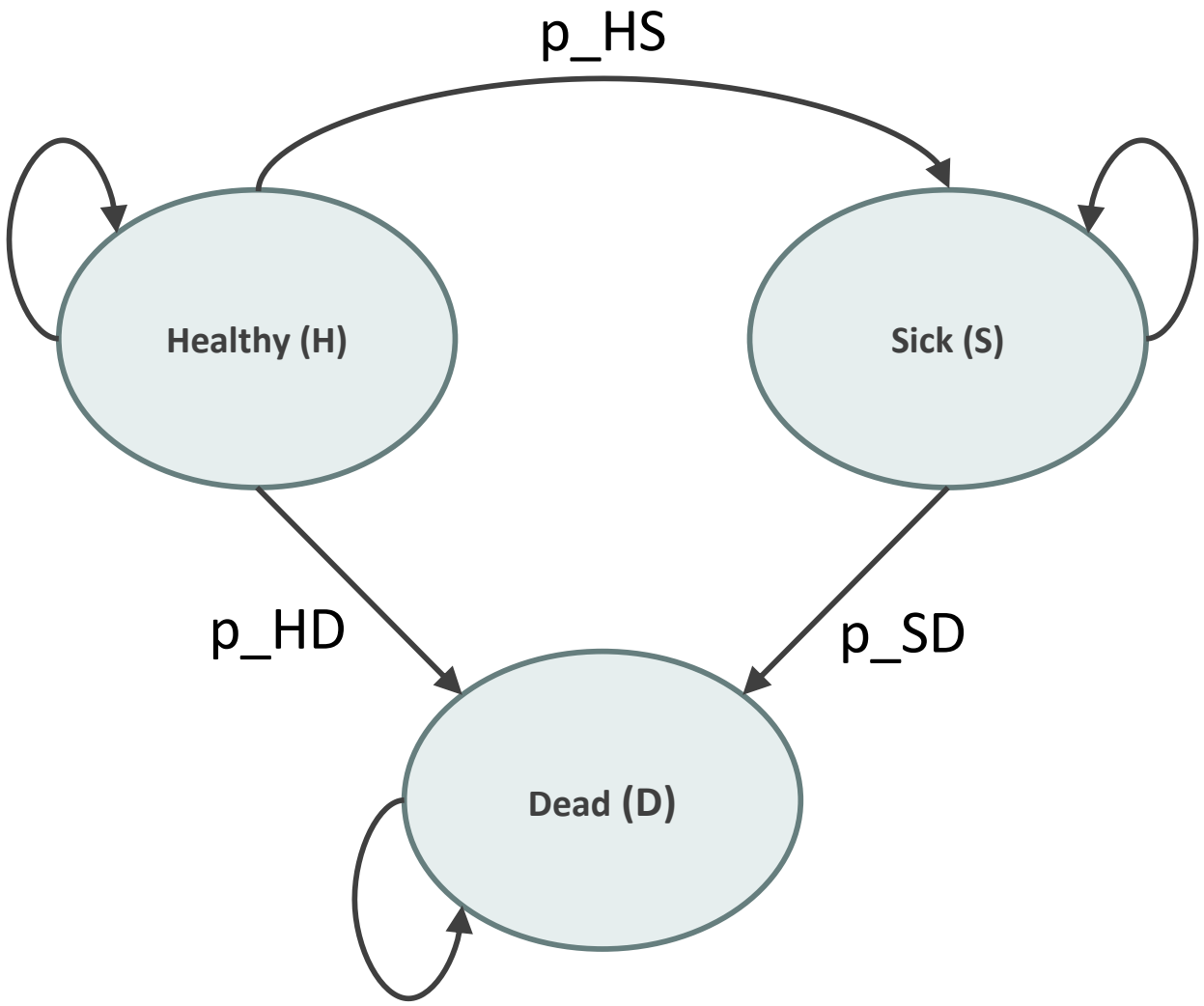
Cohort State-Transition Models (cSTMs)

- Models where proportions of a cohort occupy states at each point in time (e.g., healthy, sick, stable, progressed, dead).
- Transitions allowed between states with some probability.
- Transitions occur in cycles (months, years etc).
- Each state associated with a value associated with a model outcome ($, utility).
- Markov assumption: no “memory” within states.

Building a cohort State-Transition Model

- Determine health states
- Determine transitions
- Choose cycle length
- Estimate transition probabilities
- Estimate state utilities and costs per cycle
- Calculate
- Sensitivity analyses

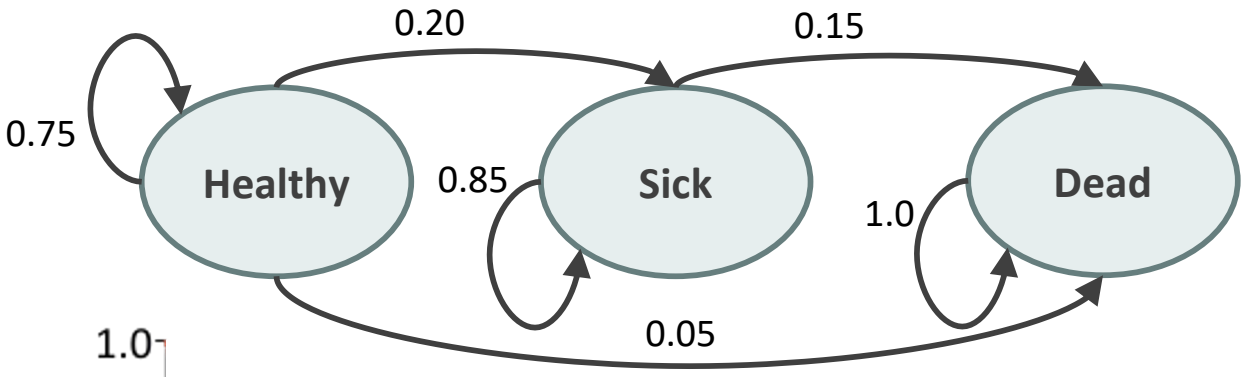
Three-State Model

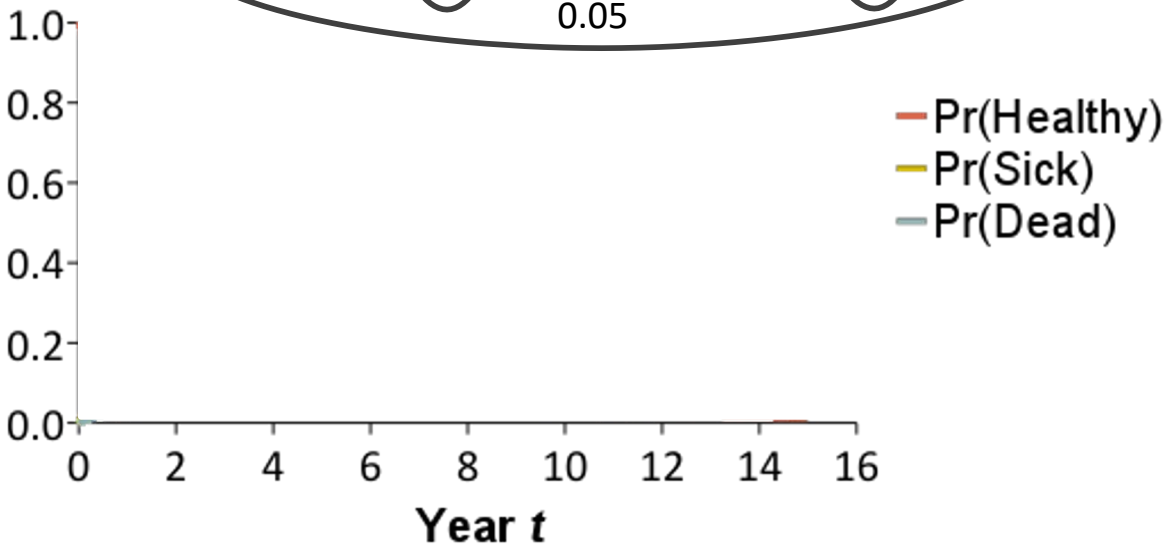


Trace the Cohort Through Time

- Reflects the distribution of a cohort of patients over a set of health states over time

$$x_{10} = [0.06 \quad 0.28 \quad 0.66]$$





$x_t(1)$

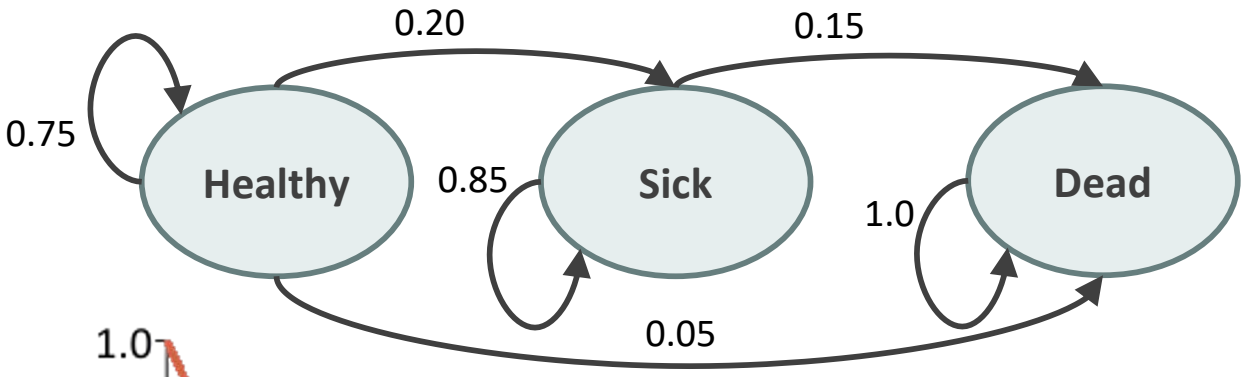
$x_t(2)$

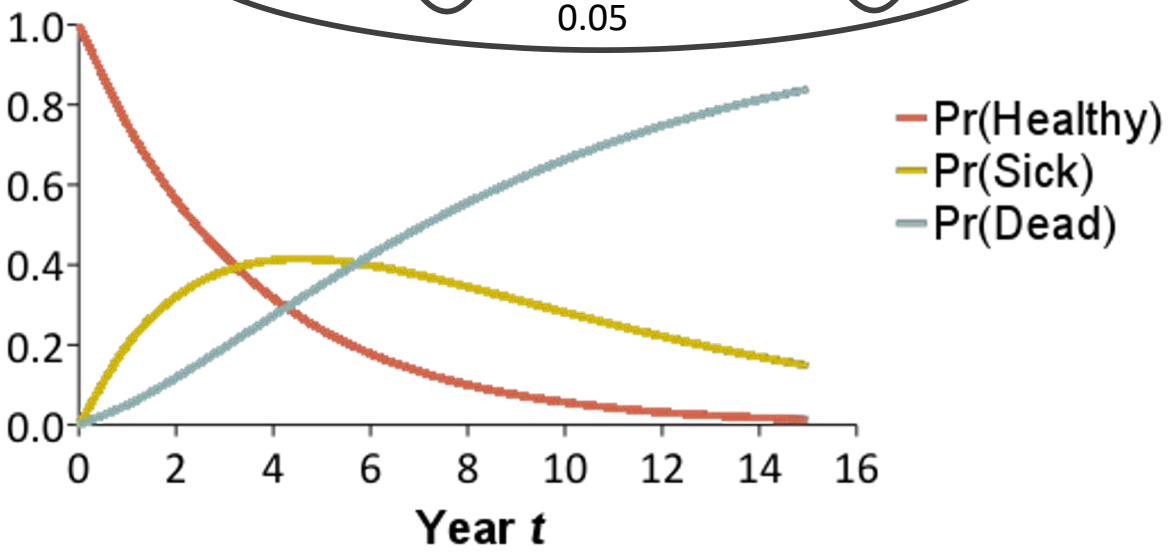
$x_t(3)$

Trace the Cohort Through Time

- Reflects the distribution of a cohort of patients over a set of health states over time

$$x_{15} = [0.01 \quad 0.15 \quad 0.84]$$





$x_t(1)$
$x_t(2)$
$x_t(3)$

Model as a Transition Matrix

- Summarize transition probabilities as a matrix

From:	To Healthy	Sick	Dead
Healthy	0.75	0.20	0.05
Sick	0	0.85	0.15
Dead	0	0	1.0

= P

- Cohort distribution at next time step calculated through matrix multiplication

$$\begin{bmatrix} \text{——} & x_{t+1} & \text{——} \end{bmatrix} = \begin{bmatrix} \text{——} & x_t & \text{——} \end{bmatrix} \begin{bmatrix} & P & \end{bmatrix}$$

"Running" the Model

- Summarize transition probabilities as a matrix

From:	To Healthy	Sick	Dead
Healthy	0.75	0.20	0.05
Sick	0	0.85	0.15
Dead	0	0	1.0

= P

- Cohort distribution at next time step calculated through matrix multiplication

$$\begin{bmatrix} \text{——} & x_{t+1} & \text{——} \end{bmatrix} = \begin{bmatrix} \text{——} & x_{t} & \text{——} \end{bmatrix} \underset{P}{\begin{bmatrix} 0.75 & 0.20 & 0.05 \\ 0 & 0.85 & 0.15 \\ 0 & 0 & 1.0 \end{bmatrix}}$$

"Running" the Model

- Summarize transition probabilities as a matrix

From:	To Healthy	To Sick	To Dead
Healthy	0.75	0.20	0.05
Sick	0	0.85	0.15
Dead	0	0	1.0

= P

- Cohort distribution at next time step calculated through matrix multiplication

$$\underset{x_1}{\begin{bmatrix} 0.75 & 0.20 & 0.05 \end{bmatrix}} = \underset{x_0}{\begin{bmatrix} 1.0 & 0.0 & 0.0 \end{bmatrix}} \underset{P}{\begin{bmatrix} 0.75 & 0.20 & 0.05 \\ 0 & 0.85 & 0.15 \\ 0 & 0 & 1.0 \end{bmatrix}}$$

Markov Trace (Life-Years)

- Calculate expected remaining LE, QALE, costs
  - Multiply cohort distribution by state-specific values to calculate expected value at each time
  - Sum expected values over time (discount if desired)

Life-Years: 1.0 1.0 0.0

Time	Healthy	Sick	Dead	E[LYs]
0	1.0	0.0	0.0	--
1	0.75	0.20	0.05	
2	0.56	0.32	0.12	
3	0.42	0.38	0.19	
	…	…	…	

Sum

* $1/(1+r)$

* $1/(1+r)^2$

* $1/(1+r)^3$

Total life years: 6.77 years

(Remaining life expectancy)

Markov Trace (Costs)

- Calculate expected remaining LE, QALE, costs
  - Multiply cohort distribution by state-specific values to calculate expected value at each time
  - Sum expected values over time (discount if desired)

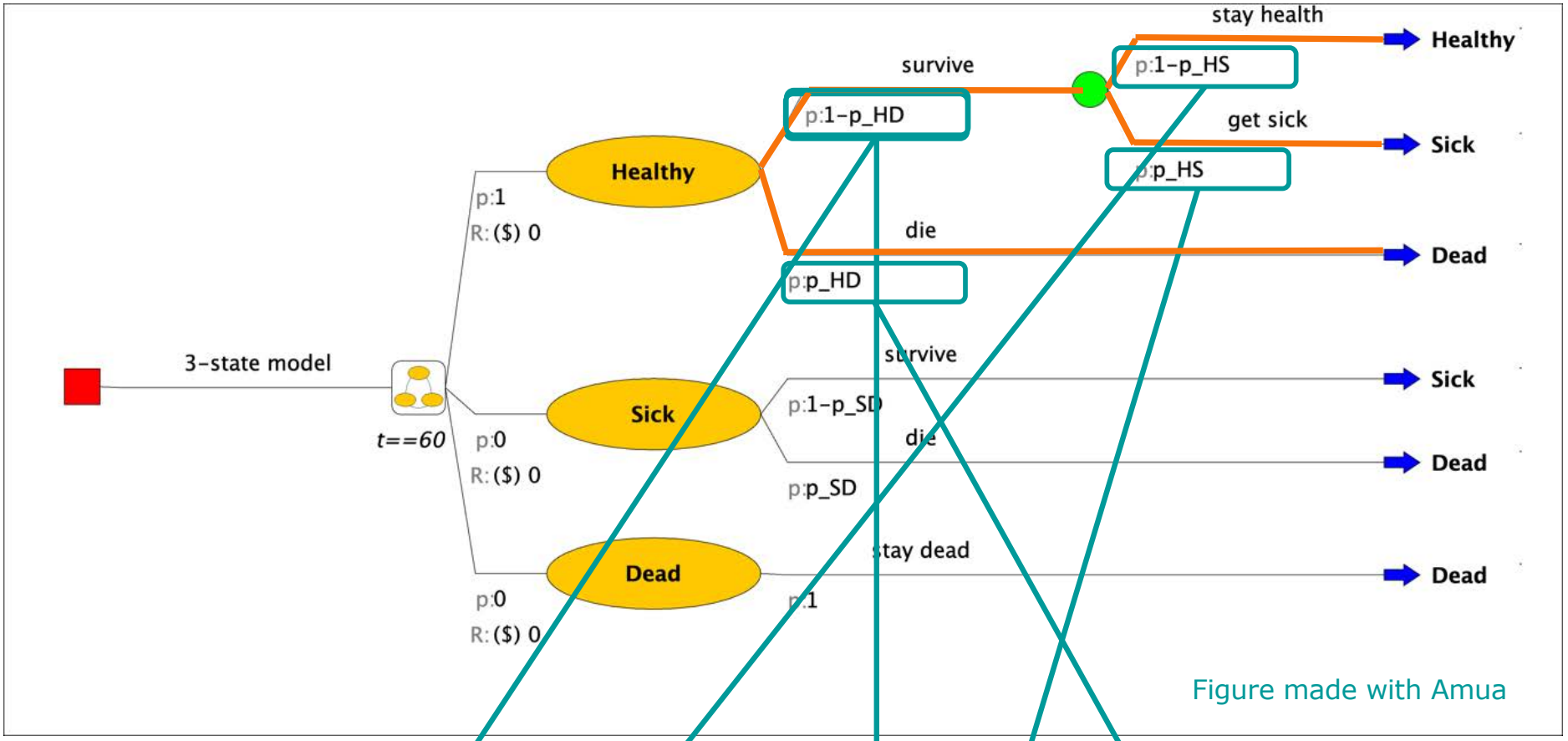
Costs:	$500	$2,500	$0	
Time	Healthy	Sick	Dead	E[Costs]
0	1.0	0.0	0.0	--
1	0.75	0.20	0.05	* 1/(1+r)
2	0.56	0.32	0.12	* $1/(1+r)^2$
3	0.42	0.38	0.19	* $1/(1+r)^3$
	…	…	…	

Sum ↓

Total costs: $11,557

(Total remaining lifetime costs)

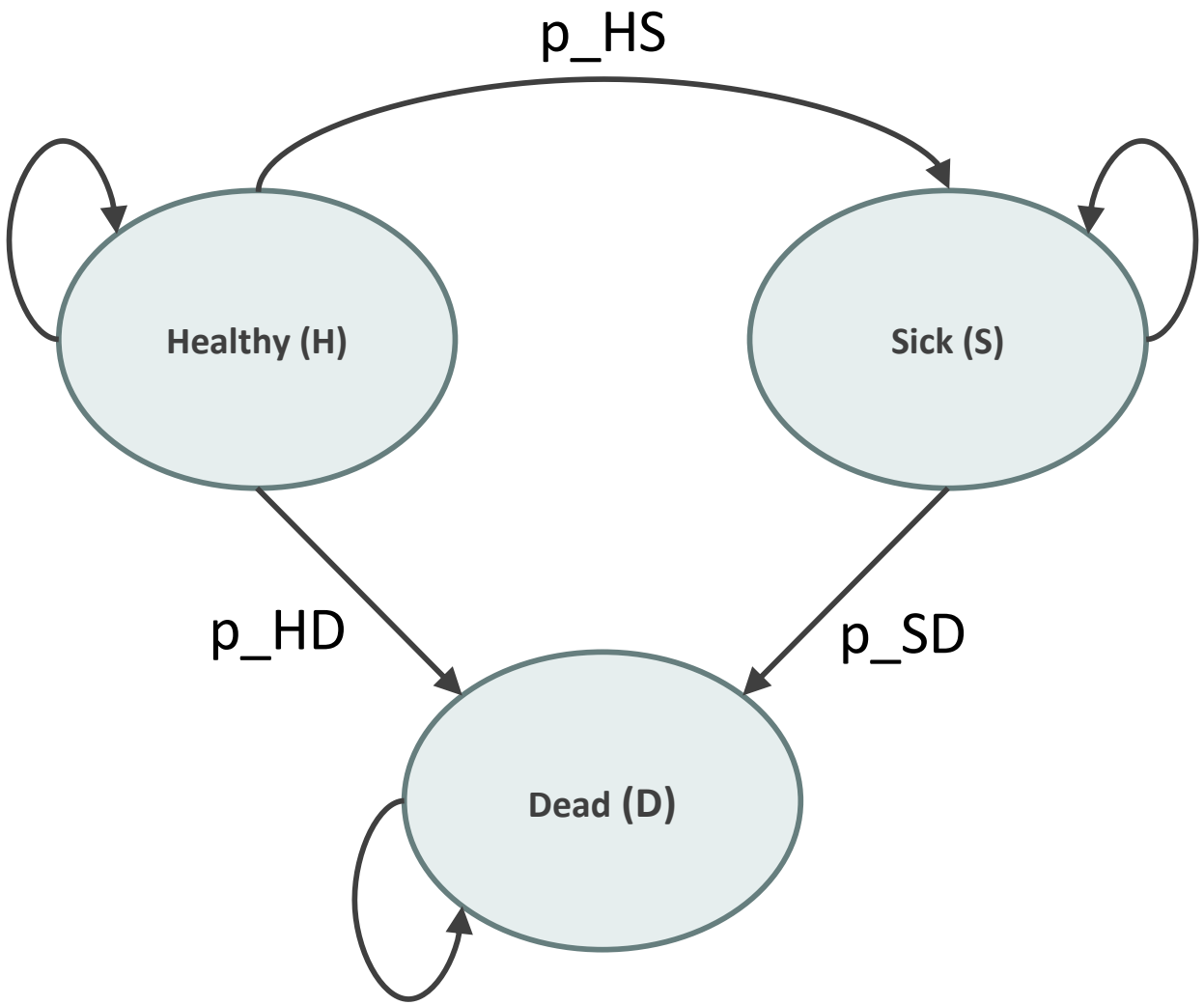
Transition Probabilities



From/To	Healthy	Sick	Dead
Healthy	(1-p_HD) * (1-p_HS)	(1-p_HD) * p_HS	p_HD
…			

Conceptualizing the Markov model in R

Simple state transition model in R



Simple state transition model

Model input:

p_{HS}: transition probability from *Healthy* to *Sick*

p_{HD}: transition probability *Healthy* to *Dead*

p_{SD}: transition probability *Sick* to *Dead*

c_H: cost of being in state *Healthy*

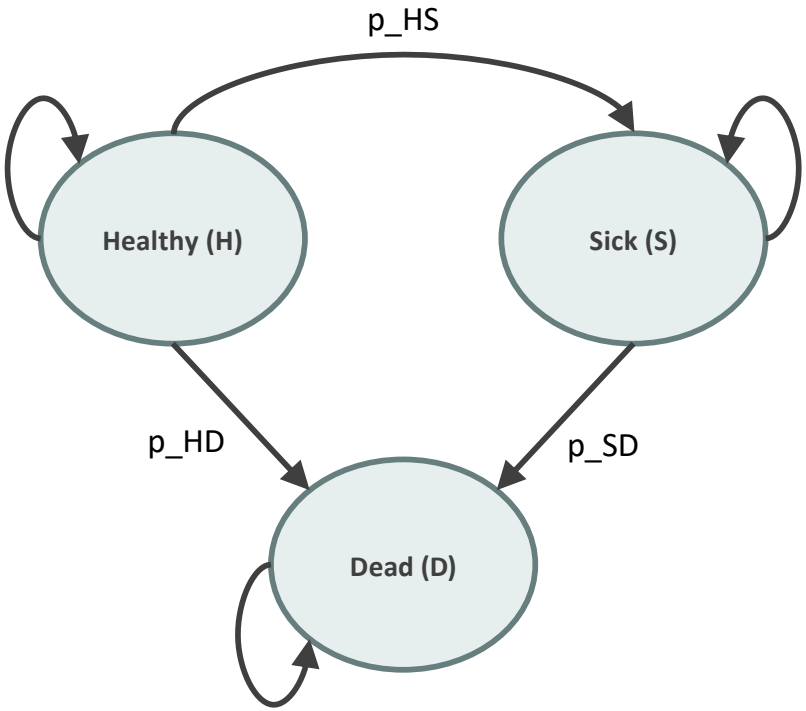
c_S: cost of being in state *Sick*

e_H: outcomes associated with state *Healthy*

e_S: outcomes associated with state *Sick*

No cost or disutility associated with death

Matrix Implementation of the Markov Model



Transition probability matrix

$$P = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{S} \\ \mathbf{D} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{H} & \mathbf{S} & \mathbf{D} \end{array} \\ \begin{bmatrix} 1 - p_{HS} - p_{HD} & p_{HS} & p_{HD} \\ 0 & 1 - p_{SD} & p_{SD} \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Vector of cycle's cost/outcomes

$$C = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{S} \\ \mathbf{D} \end{array} \begin{bmatrix} c_H \\ c_S \\ 0 \end{bmatrix} \qquad E = \begin{array}{c} \\ \mathbf{H} \\ \mathbf{S} \\ \mathbf{D} \end{array} \begin{bmatrix} e_H \\ e_S \\ 0 \end{bmatrix}$$

Matrix Implementation of the Markov Model

Create the t x 3 matrix M that will store the proportion of the cohort at each state and cycle:

At $t = 0$:

$$M_0 = [1, 0, 0]$$

For $t < T$:

$$M_{t+1} = M_t \mathrm{P}$$

$$[H_t \quad S_t \quad D_t] \begin{bmatrix} 1 - p_{HS} - p_{HD} & p_{HS} & p_{HD} \\ 0 & 1 - p_{SD} & p_{SD} \\ 0 & 0 & 1 \end{bmatrix}$$

Calculating total costs & effects

Total effects (TE):

$$E = M\,e$$
$$TE = \mathbf{1}_T E$$

Total costs (TC):

$$C = M\,c$$
$$TC = \mathbf{1}_T C$$

$\mathbf{1}_T : 1 \times T$ vector of ones

Calculating total costs & effects (discounted)

Total effects (TE):

$$E = M\,e$$

$$TE = dw_T\,E$$

Total costs (TC):

$$C = M\,c$$

$$TC = dw_T\,C$$

$$dw_T = \frac{1}{(1+d)^{0:T}}$$

Time-dependence

- **Since start of the model**
- Transition probabilities often depend on age
  - Background mortality
  - Risk of developing disease or experiencing an event
- In other words, matrix P is not the same every cycle

- **Depending on state residence**
- Some transition probabilities depend on time since an event, not age
  - e.g., The risk of developing recurrence among newly diagnosed cancer patients declines with time

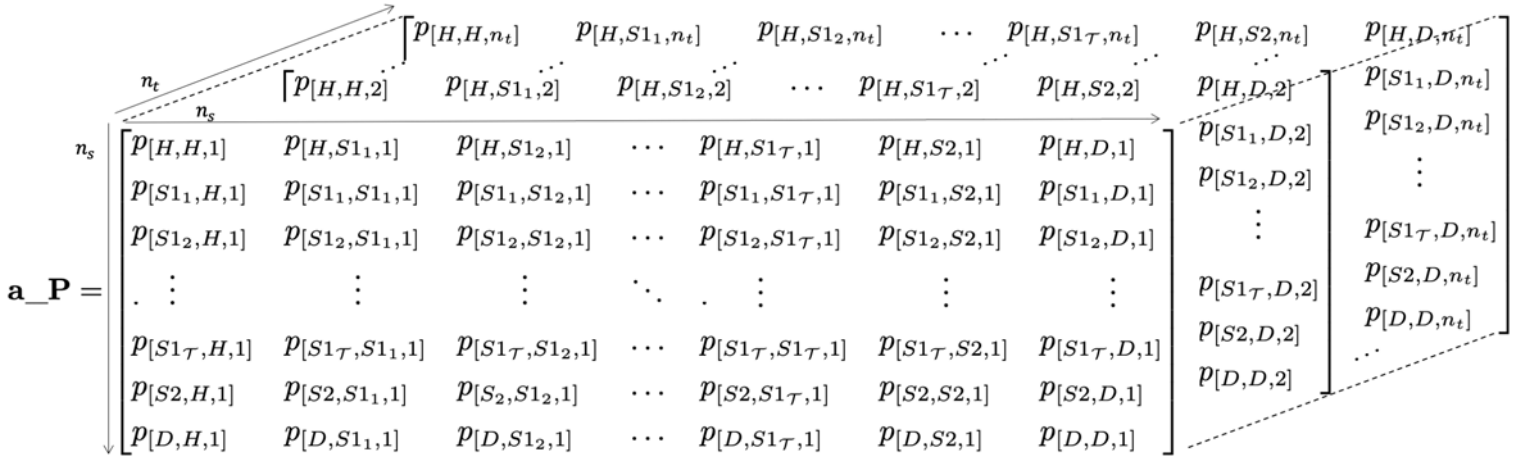
Time-dependence since model start

- Transition probabilities often depend on time since model start
  - Background mortality
  - Risk of developing disease or experiencing an event
- In other words, matrix P is not the same every cycle
- Replace matrix P with matrices P_t, where t is time since model start

Time-varying probabilities in R

- We create a 3D array, `a_P`, that stores a collection of time-varying transition matrices, P_t, in the third dimension

- For example, `a_P` for the Sick-Sicker Markov model:

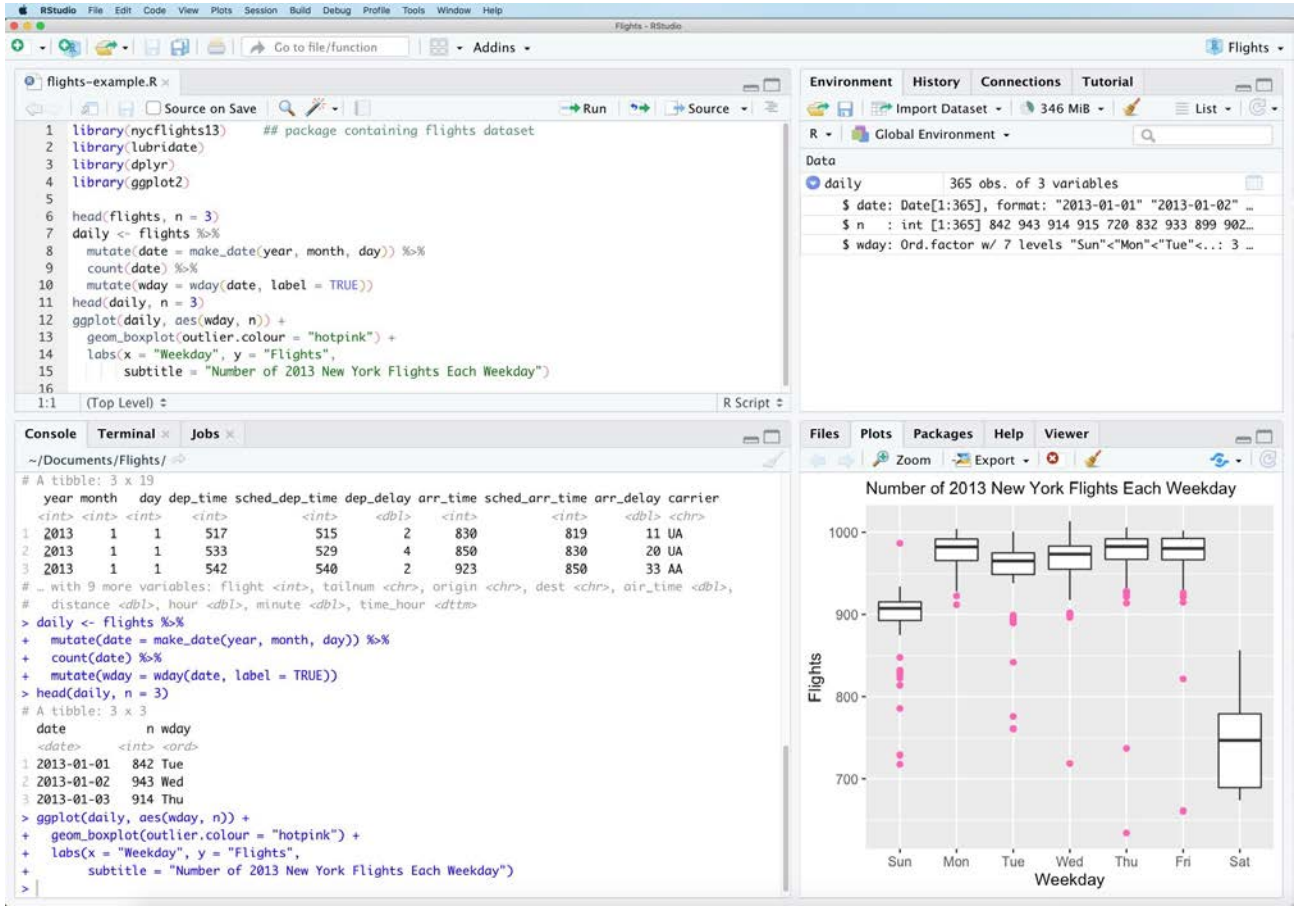


R session

State-Transition Model & Computing CEA Results

R tools

- Programming Language/Interpreter: R
- Integrated Development Environment: R Studio
- Extensions: Various Packages/Libraries



Starting our script

- We clear out the environment
- We load libraries that we need (and we may need to install these first)

```
3  rm(list = ls())      # clear memory (removes all the variables from the workspace)
4
5  # 01 Load packages
6
7  library(devtools)
8  install_github("DARTH-git/darthtools", force = TRUE)
9  p_load_gh("DARTH-git/darthtools")
10 library(darthtools)
11 library(dampack)
12 library(reshape2)
13 library(diagram)
```

R scripts can also load other functions as needed

- In our first example, we do not have another script file

```
15  # 02 Load functions
16
17  # all functions are in the darthtools package
```

- But if we need to do this we could add code like

```
source("path/load_data.R")
source("path/analysis.R")
```

Define model inputs (1)

```r
# 03 Model input

n_time_horizon_yr <- 60                                  # time horizon (in years)
cycle_length    <- 1                                     # cycle length in years (use 1/12 for monthly)
n_cycles        <- n_time_horizon_yr / cycle_length      # number of cycles
v_names_cycles  <- paste("cycle", 0:n_cycles)            # cycle names
v_names_states  <- c("Healthy", "Sick", "Dead")          # state names
n_states        <- length(v_names_states)                # number of health states

### Discounting factors
d_c <- 0.03 # annual discount rate for costs
d_e <- 0.03 # annual discount rate for QALYs

### Strategies
v_names_str     <- c("Standard of Care",          # store the strategy names
                     "Treatment A",
                     "Treatment B")
n_str           <- length(v_names_str)           # number of strategies


### Within-cycle correction (WCC) using Simpson's 1/3 rule
v_wcc <- gen_wcc(n_cycles = n_cycles,  method = "Simpson1/3")

### Transition probabilities
p_HD_yr       <- 0.01  # annual probability of dying when healthy
p_SD_yr       <- 0.10  # annual probability of dying when sick
# Annual probability of becoming sick from Healthy, conditional on surviving cycle
p_HS_yr_SoC   <- 0.05   # under standard of care
p_HS_yr_trtA  <- 0.04  # under treatment A
p_HS_yr_trtB  <- 0.02  # under treatment B
```

Define model inputs (2)

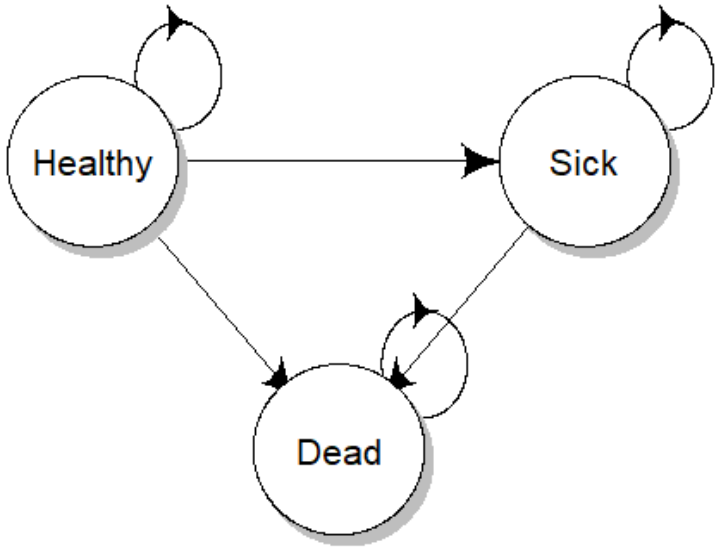
```r
### State rewards

#### Costs
c_H_yr     <- 400   # cost of one year in healthy state
c_S_yr     <- 1000  # cost of one year in sick state
c_D_yr     <- 0     # cost of one year in dead state
c_trtA_yr  <- 800   # cost of treatment A (per year) in healthy state
c_trtB_yr  <- 1500  # cost of treatment B (per year) in healthy state
#### Utilities
u_H        <- 1     # utility when healthy
u_S        <- 0.5   # utility when sick
u_D        <- 0     # utility when dead

## 03.2 Calculate internal model parameters

### Cycle-specific discount weight for costs and effects
v_dwc   <- 1 / ((1 + (d_e * cycle_length)) ^ (0:n_cycles))
v_dwe   <- 1 / ((1 + (d_c * cycle_length)) ^ (0:n_cycles))

### Calculate cycle-specific transition probabilities
p_HD <- 1 - exp(-(-log(1 - p_HD_yr) * cycle_length)) # probability of dying from Healthy
p_SD <- 1 - exp(-(-log(1 - p_SD_yr) * cycle_length)) # probability of dying from Sick
# probability of becoming sick from Healthy
p_HS_SoC  <- 1 - exp(-(-log(1 - p_HS_yr_SoC) * cycle_length))  # Standard of Care
p_HS_trtA <- 1 - exp(-(-log(1 - p_HS_yr_trtA) * cycle_length)) # Treatment A
p_HS_trtB <- 1 - exp(-(-log(1 - p_HS_yr_trtB) * cycle_length)) # Treatment B
```

Define model inputs (3)

```r
### Calculate cycle-specific state rewards
#### Costs
v_c_SoC  <- c(c_H_yr, c_S_yr, c_D_yr) * cycle_length             # Standard of Care
v_c_trtA <- c(c_H_yr + c_trtA_yr, c_S_yr, c_D_yr) * cycle_length # Treatment A
v_c_trtB <- c(c_H_yr + c_trtB_yr, c_S_yr, c_D_yr) * cycle_length # Treatment B
#### QALYs
v_q_SoC  <- c(u_H, u_S, u_D) * cycle_length # Standord of Care
v_q_trtA <- v_q_trtB <- v_q_SoC             # Treatments A and B have same utilities as SoC
```

Using R to make a state-transition diagram

```r
# 04 Construct state-transition models

m_P_diag <- matrix(0, nrow = n_states, ncol = n_states,
                   dimnames = list(v_names_states, v_names_states))
m_P_diag["Healthy", "Sick" ]    = ""
m_P_diag["Healthy", "Dead" ]    = ""
m_P_diag["Healthy", "Healthy" ] = ""
m_P_diag["Sick"   , "Dead" ]    = ""
m_P_diag["Sick"   , "Sick" ]    = ""
m_P_diag["Dead"   , "Dead" ]    = ""
layout.fig <- c(2, 1)
plotmat(t(m_P_diag), t(layout.fig), self.cex = 0.5, curve = 0, arr.pos = 0.8,
        latex = T, arr.type = "curved", relsize = 0.85, box.prop = 0.8,
        cex = 0.8, box.cex = 0.7, lwd = 1)
```



Proportions starting in each state and Markov trace

```r
## 04.1 Initial state vector

# All starting healthy
v_m_init <- c("Healthy" = 1, "Sick" = 0, "Dead" = 0)
v_m_init

## 04.2 Initialize cohort traces

### Initialize cohort trace for SoC
m_M_SoC <- matrix(0,
                  nrow = (n_cycles + 1), ncol = n_states,
                  dimnames = list(v_names_cycles, v_names_states))
# Store the initial state vector in the first row of the cohort trace
m_M_SoC[1, ] <- v_m_init

## Initialize cohort traces for treatments A and B
# Structure and initial states are the same as for SoC
m_M_trtA <- m_M_trtB <- m_M_SoC
```

Fill transition matrix

```r
## 04.3 Create transition probability arrays

## Create transition probability matrices for strategy SoC
### Initialize transition probability matrix
# All transitions to a non-death state are assumed to be conditional on survival
m_P_SoC  <- matrix(0,
                   nrow = n_states, ncol = n_states,
                   dimnames = list(v_names_states, v_names_states))
### Fill in matrix
# from Healthy
m_P_SoC["Healthy", "Healthy"] <- (1 - p_HD) * (1 - p_HS_SoC)
m_P_SoC["Healthy", "Sick"]    <- (1 - p_HD) *      p_HS_SoC
m_P_SoC["Healthy", "Dead"]    <-      p_HD
# from Sick
m_P_SoC["Sick", "Sick"] <- 1 - p_SD
m_P_SoC["Sick", "Dead"] <-     p_SD
# from Dead
m_P_SoC["Dead", "Dead"] <- 1

## Treatment A
# Start with same matrix as SoC, but replace parameters that differ for trtA
m_P_trtA <- m_P_SoC
m_P_trtA["Healthy", "Healthy"] <- (1 - p_HD) * (1 - p_HS_trtA)
m_P_trtA["Healthy", "Sick"]    <- (1 - p_HD) *      p_HS_trtA

## Treatment B
# Start with same matrix as SoC, but replace parameters that differ for trtB
m_P_trtB <- m_P_SoC
m_P_trtB["Healthy", "Healthy"] <- (1 - p_HD) * (1 - p_HS_trtB)
m_P_trtB["Healthy", "Sick"]    <- (1 - p_HD) *      p_HS_trtB

```

Are transition matrices valid?

```r
## Check if transition probability matrices are valid
### Check that transition probabilities are in [0, 1]
check_transition_probability(m_P_SoC,  verbose = TRUE)
check_transition_probability(m_P_trtA, verbose = TRUE)
check_transition_probability(m_P_trtB, verbose = TRUE)
### Check that all rows sum to 1
check_sum_of_transition_array(m_P_SoC,  n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
check_sum_of_transition_array(m_P_trtA, n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
check_sum_of_transition_array(m_P_trtB, n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
```

Console | Terminal | Background Jobs

R 4.1.3 · C:/Users/jeremygf/Dropbox/SharedWork/Courses/AdvancedDSMethods/CURRENT_YEAR/WORKSHOPS/INFECTIOUS_DISEASE_MODEL_WORKSHOP/M

```
>
> ## Check if transition probability matrices are valid
> ### Check that transition probabilities are in [0, 1]
> check_transition_probability(m_P_SoC,  verbose = TRUE)
[1] "Valid transition probabilities"
> check_transition_probability(m_P_trtA, verbose = TRUE)
[1] "Valid transition probabilities"
> check_transition_probability(m_P_trtB, verbose = TRUE)
[1] "Valid transition probabilities"
> ### Check that all rows sum to 1
> check_sum_of_transition_array(m_P_SoC,  n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
[1] "This is a valid transition matrix"
> check_sum_of_transition_array(m_P_trtA, n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
[1] "This is a valid transition matrix"
> check_sum_of_transition_array(m_P_trtB, n_states = n_states, n_cycles = n_cycles, verbose = TRUE)
[1] "This is a valid transition matrix"
```

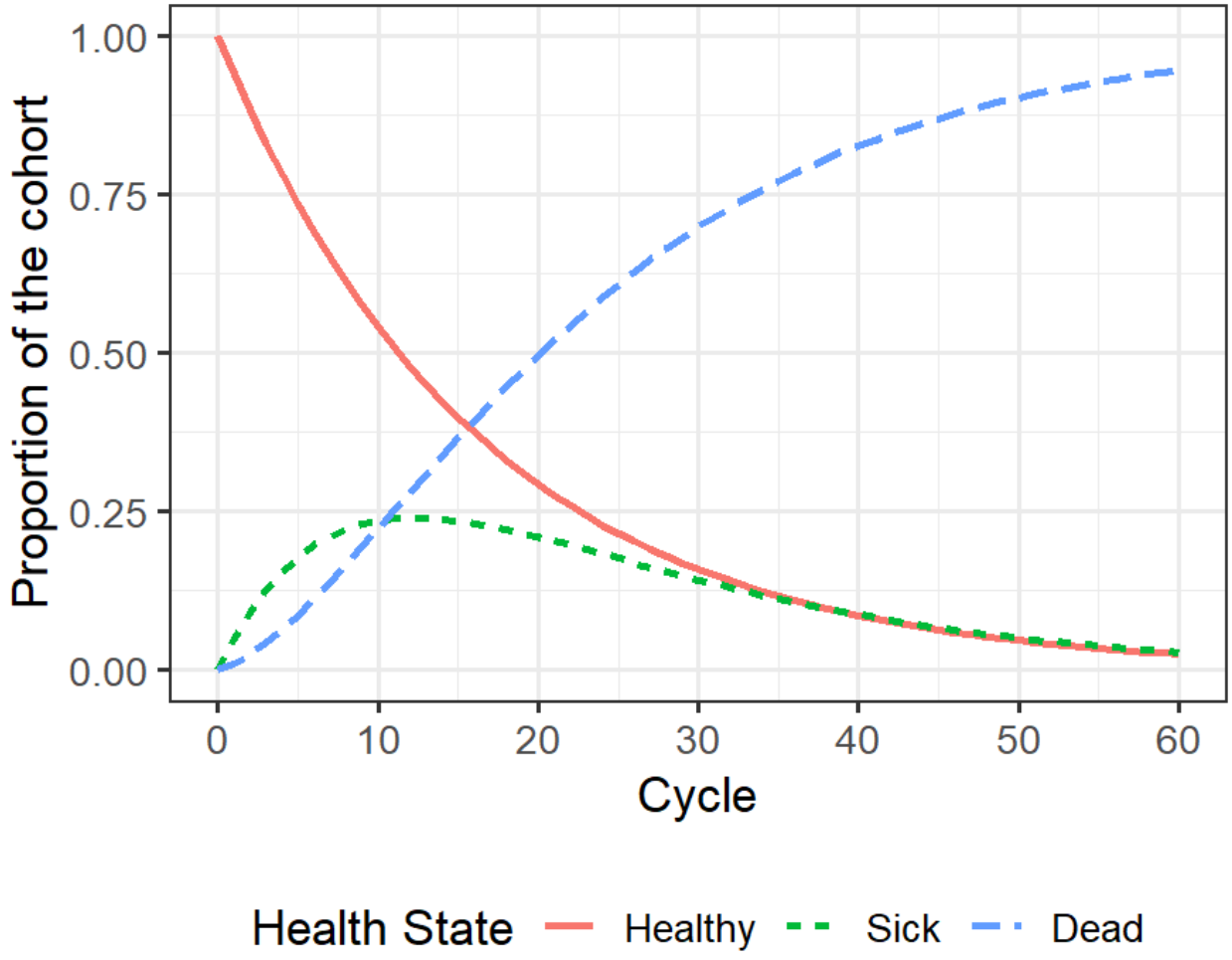
Run the model for each strategy

```r
# 05 Run cohort state transition model

## Loop over time
# Calculating cohort state based on previous state and transition matrix
for (t in 1:n_cycles){
  # For SoC
  m_M_SoC [t + 1, ] <- m_M_SoC [t, ] %*% m_P_SoC
  # For treatment A
  m_M_trtA[t + 1, ] <- m_M_trtA[t, ] %*% m_P_trtA
  # For treatment B
  m_M_trtB[t + 1, ] <- m_M_trtB[t, ] %*% m_P_trtB
}
```

```
> m_M_trtB
            Healthy       Sick       Dead
cycle 0   1.0000000 0.00000000 0.00000000
cycle 1   0.9702000 0.01980000 0.01000000
cycle 2   0.9412880 0.03702996 0.02168200
cycle 3   0.9132377 0.05196447 0.03479788
cycle 4   0.8860232 0.06485013 0.04912670
cycle 5   0.8596197 0.07590837 0.06447194
cycle 6   0.8340030 0.08533800 0.08065898
cycle 7   0.8091497 0.09331746 0.09753281
cycle 8   0.7850371 0.10000688 0.11495605
cycle 9   0.7616430 0.10554993 0.13280711
cycle 10  0.7389460 0.11007547 0.15097853
```

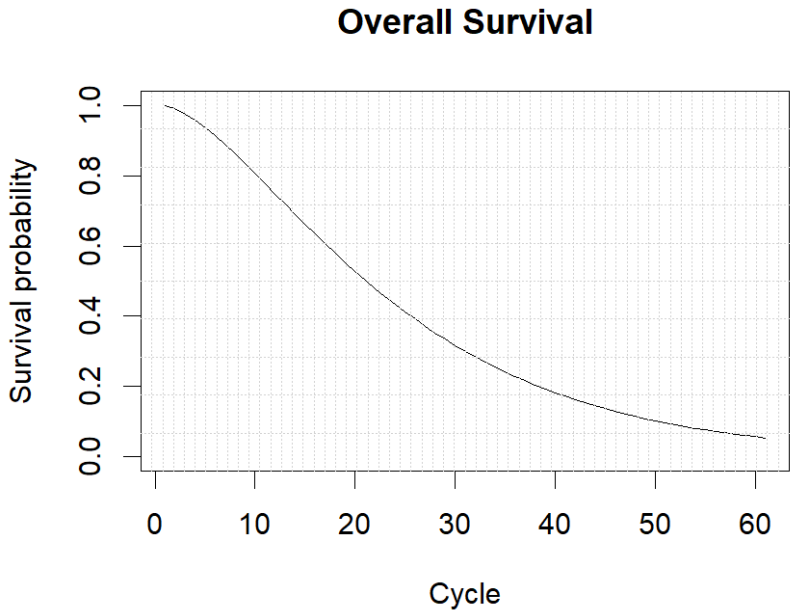
Plot Model trace

```
178  # 06 Plot Outputs
179
180  ## 06.1 Plot the cohort trace for strategies SoC
181
182  plot_trace(m_M_SoC)
```



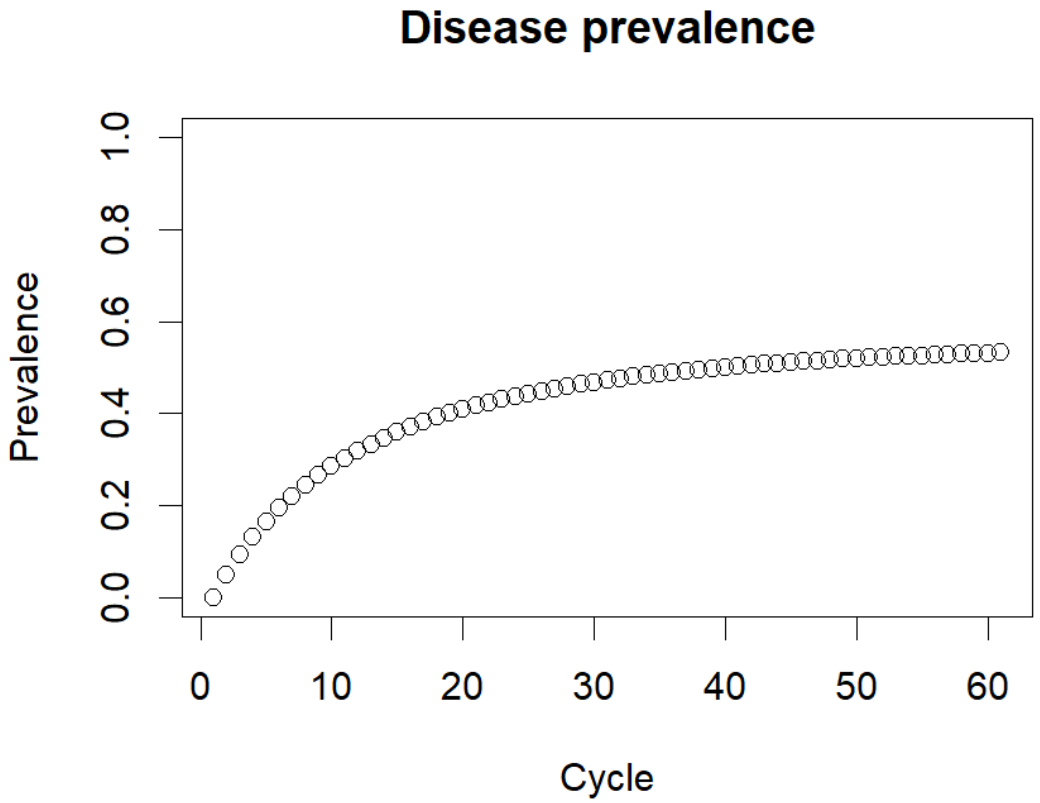
Survival curves & life expectancy

```r
## 06.2 Overall Survival (OS)

#Print the overall survival for the Standard of Care

v_os_SoC <- 1 - m_M_SoC[, "Dead"]    # calculate the overall survival (OS) probability
v_os_SoC <- rowSums(m_M_SoC[, 1:2])  # alternative way of calculating the OS probability

plot(v_os_SoC, type = 'l',
     ylim = c(0, 1),
     ylab = "Survival probability",
     xlab = "Cycle",
     main = "Overall Survival")  # create a simple plot showing the OS

# add grid
grid(nx = n_cycles, ny = 10, col = "lightgray", lty = "dotted", lwd = par("lwd"),
     equilogs = TRUE)

## 06.2.1 Life Expectancy (LE)

le_SoC <- sum(v_os_SoC)  # summing probability of OS over time  (i.e. life expectancy)
```



Disease prevalence

```
205  ## 06.2.2 Disease prevalence
206
207  v_prev <- m_M_SoC[, "Sick"]/v_os_SoC
208  plot(v_prev,
209       ylim = c(0, 1),
210       ylab = "Prevalence",
211       xlab = "Cycle",
212       main = "Disease prevalence")
```



State rewards (initialize values) (1)

```
214  # 07 State Rewards
215
216  ## Scale by the cycle length
217  # Vector of state utilities under strategy SoC
218  v_u_SoC    <- c(H  = u_H,
219                  S  = u_S,
220                  D  = u_D) * cycle_length
221  # Vector of state costs under strategy SoC
222  v_c_SoC    <- c(H  = c_H_yr,
223                  S  = c_S_yr,
224                  D  = c_D_yr) * cycle_length
225  # Vector of state utilities under treatment A
226  v_u_trtA   <- c(H  = u_H,
227                  S  = u_S,
228                  D  = u_D) * cycle_length
229  # Vector of state costs under treatment A
230  v_c_trtA   <- c(H  = c_H_yr + c_trtA_yr,
231                  S  = c_S_yr,
232                  D  = c_D_yr) * cycle_length
233  # Vector of state utilities under treatment B
234  v_u_trtB   <- c(H  = u_H,
235                  S  = u_S,
236                  D  = u_D) * cycle_length
237  # Vector of state costs under treatment B
238  v_c_trtB   <- c(H  = c_H_yr + c_trtB_yr,
239                  S  = c_S_yr,
240                  D  = c_D_yr) * cycle_length
```

State rewards (initialize values) (2)

```
242  ## Store state rewards
243  # Store the vectors of state utilities for each strategy in a list
244  l_u   <- list(SQ = v_u_SoC,
245                A  = v_u_trtA,
246                B  = v_u_trtB)
247  # Store the vectors of state cost for each strategy in a list
248  l_c   <- list(SQ = v_c_SoC,
249                A  = v_c_trtA,
250                B  = v_c_trtB)
251
252  # assign strategy names to matching items in the lists
253  names(l_u) <- names(l_c) <- v_names_str
```

State rewards (apply values to trace)

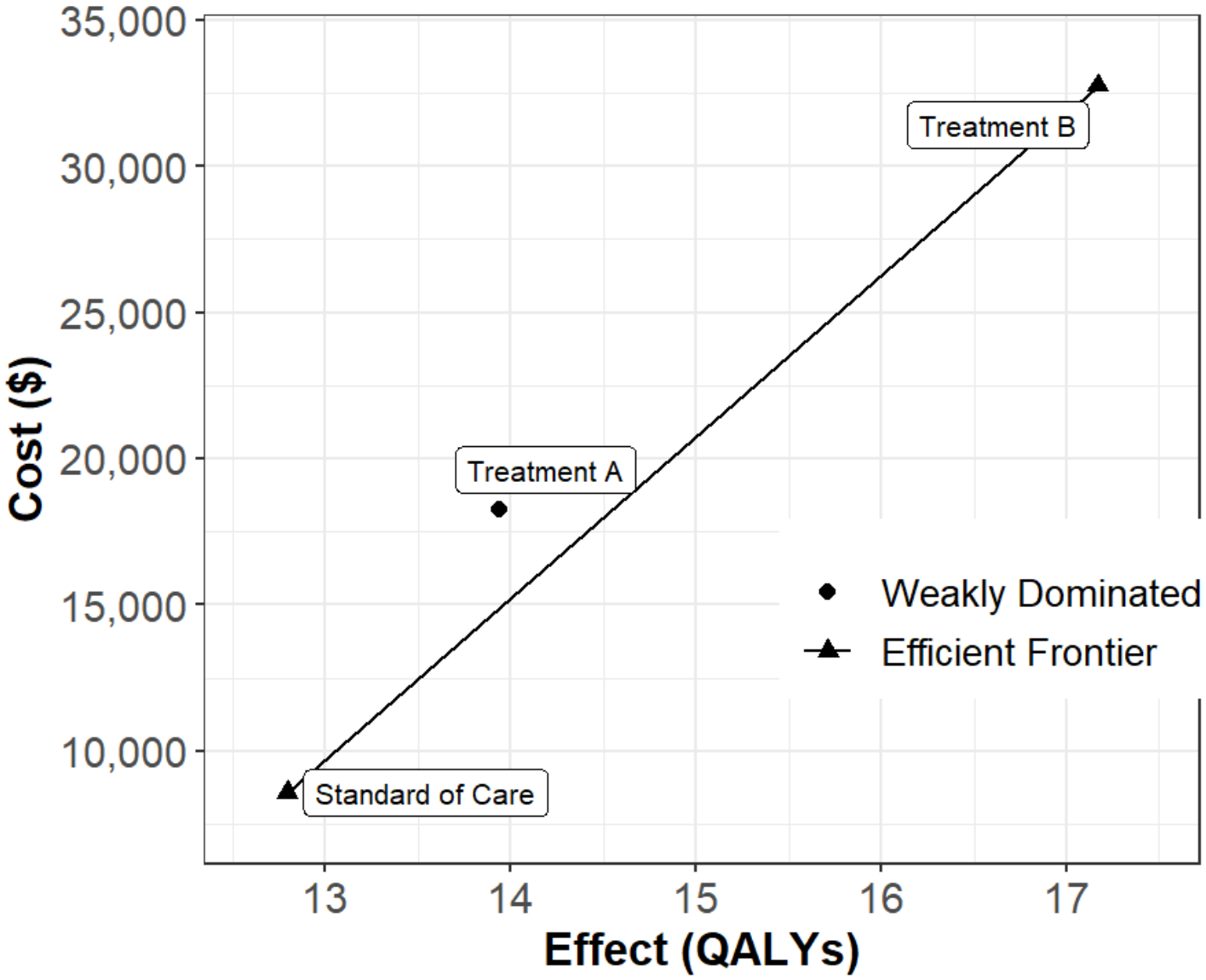
```r
# 08 Compute expected outcomes


# Create empty vectors to store total costs and QALYs
v_tot_cost <- v_tot_qaly <- vector(mode = "numeric", length = n_str)
names(v_tot_qaly) <- names(v_tot_cost) <- v_names_str


### Expected discounted cost for each strategy
v_tot_cost["Standard of Care"] <- t(m_M_SoC %*% v_c_SoC) %*% (v_dwe * v_wcc)
v_tot_cost["Treatment A"] <- t(m_M_trtA %*% v_c_trtA) %*% (v_dwe * v_wcc)
v_tot_cost["Treatment B"] <- t(m_M_trtB %*% v_c_trtB) %*% (v_dwe * v_wcc)

### Expected discounted QALYs for each strategy
v_tot_qaly["Standard of Care"] <- t(m_M_SoC %*% v_q_SoC) %*% (v_dwe * v_wcc)
v_tot_qaly["Treatment A"] <- t(m_M_trtA %*% v_q_trtA) %*% (v_dwe * v_wcc)
v_tot_qaly["Treatment B"] <- t(m_M_trtB %*% v_q_trtB) %*% (v_dwe * v_wcc)
```

CEA (1)

```r
# 09 Cost-effectiveness analysis (CEA)

## Incremental cost-effectiveness ratios (ICERs)
df_cea <- calculate_icers(cost       = v_tot_cost,
                          effect     = v_tot_qaly,
                          strategies = v_names_str)
df_cea

## CEA table in proper format
table_cea <- format_table_cea(df_cea)
table_cea

## CEA frontier
plot(df_cea, label = "all", txtsize = 14) +
  expand_limits(x = max(table_cea$QALYs) + 0.1) +
  theme(legend.position = c(0.8, 0.3))
```

	Strategy	Costs ($)	QALYs	Incremental Costs ($)	Incremental QALYs	ICER ($/QALY)	Status
Standard of Care	Standard of Care	8,575	12.79	<NA>	NA	<NA>	ND
Treatment B	Treatment B	32,745	17.17	24,171	4.37	5,526	ND
Treatment A	Treatment A	18,272	13.94	<NA>	NA	<NA>	ED

CEA (2)


35,000
30,000
25,000
20,000
15,000
10,000
Cost ($)
Treatment B
Treatment A
Standard of Care
Weakly Dominated
Efficient Frontier
13
14
15
16
17
Effect (QALYs)

Sensitivity Analysis

- Vary input parameters within plausible ranges
- For which values is each strategy optimal?

- Deterministic sensitivity analysis (DSA)
  - One-way analysis: vary one parameter, hold rest fixed
  - Two-way analysis: vary two parameters, hold rest fixed

- **Probabilistic sensitivity analysis (PSA)**
  - Simultaneously vary input parameters by randomly sampling from appropriate probability distributions
  - How often is each alternative cost-effective?
  - What strategy has the highest **expected** net benefit?

Sensitivity Analyses (Deterministic) (1)

```
290  # 10 Deterministic Sensitivity Analysis (DSA)
291
292  ## Load model, CEA and PSA functions
293  source('Functions_cSTM_3state.R')
```

```
calculate_ce_out <- function(l_params_all, n_wtp = 10000 ,verbose= FALSE){
```

```
generate_psa_params <- function(n_sim = 1000, seed = 071818){
```

Sensitivity Analyses (Deterministic) (2)

```r
## 10.1 Model input for SA

l_params_all <- list(
    # Transition probabilities
    # probability of dying
  p_HD_yr       = 0.01,  # annual probability of dying when healthy
  p_SD_yr       = 0.10,  # annual probability of dying when sick
  # Annual probability of becoming sick from Healthy, conditional on surviving cycle
  p_HS_yr_SoC   = 0.05,  # under standard of care
  p_HS_yr_trtA  = 0.04,  # under treatment A
  p_HS_yr_trtB  = 0.02,  # under treatment B

  ### State rewards

  #### Costs
  c_H_yr      = 400,   # cost of one year in healthy state
  c_S_yr      = 1000,  # cost of one year in sick state
  c_D_yr      = 0,     # cost of one year in dead state
  c_trtA_yr   = 800,   # cost of treatment A (per year) in healthy state
  c_trtB_yr   = 1500,  # cost of treatment B (per year) in healthy state
  #### Utilities
  u_H         = 1,     # utility when healthy
  u_S         = 0.5,   # utility when sick
  u_D         = 0,     # utility when dead
  # Discount rates
  d_e       = 0.03,  # discount rate per cycle equal discount of costs and QALYs by 3%
  d_c       = 0.03,  # discount rate per cycle equal discount of costs and QALYs by 3%
  # Time horizon
  n_time_horizon_yr = 60,                # time horizon (in years)
  cycle_length      = 1                  # cycle length in years (use 1/12 for monthly)

)
```

Sensitivity Analyses (Deterministic) (3)

```
328 #**Test model functions**
329 #
330 #A function is defined in the `Functions_cSTM_3state.R` file.
331 #The first is the `calculate_ce_out()` function which runs the decision model
332 #and uses the resulting Markov trace to compute the total costs, QALYs,
333 #and net monetary benefit (NMB) for each strategy, returning a data frame of
334 #costs and QALYs.
335 #
336
337 # Try the calculate_ce_out() function
338 df_ce <- calculate_ce_out(l_params_all)
339 df_ce
340
341 # Get strategies names (will be used to label plots)
342 v_names_str <- df_ce$Strategy
343 n_str <- length(v_names_str)
```

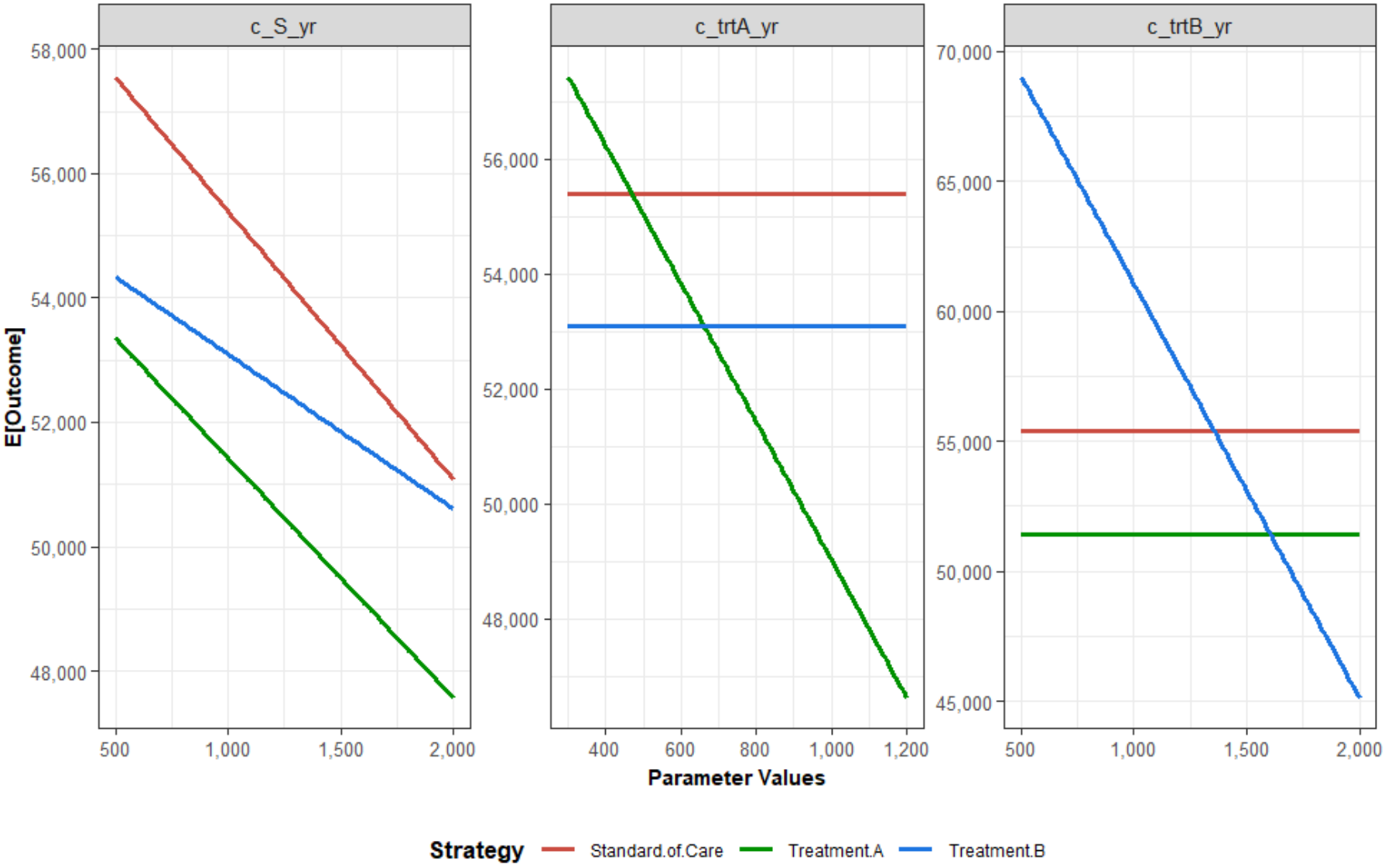
```
> df_ce
                      Strategy      Cost   Effect      NMB
Standard of Care Standard of Care  8574.738 12.79462 119371.4
Treatment A           Treatment A 18272.100 13.94028 121130.7
Treatment B           Treatment B 32745.314 17.16887 138943.4
```

	Strategy	Costs ($)	QALYs	Incremental Costs ($)	Incremental QALYs	ICER ($/QALY)	Status
Standard of Care	Standard of Care	8,575	12.79	<NA>	NA	<NA>	ND
Treatment B	Treatment B	32,745	17.17	24,171	4.37	5,526	ND
Treatment A	Treatment A	18,272	13.94	<NA>	NA	<NA>	ED

Sensitivity Analyses (Deterministic) (4)

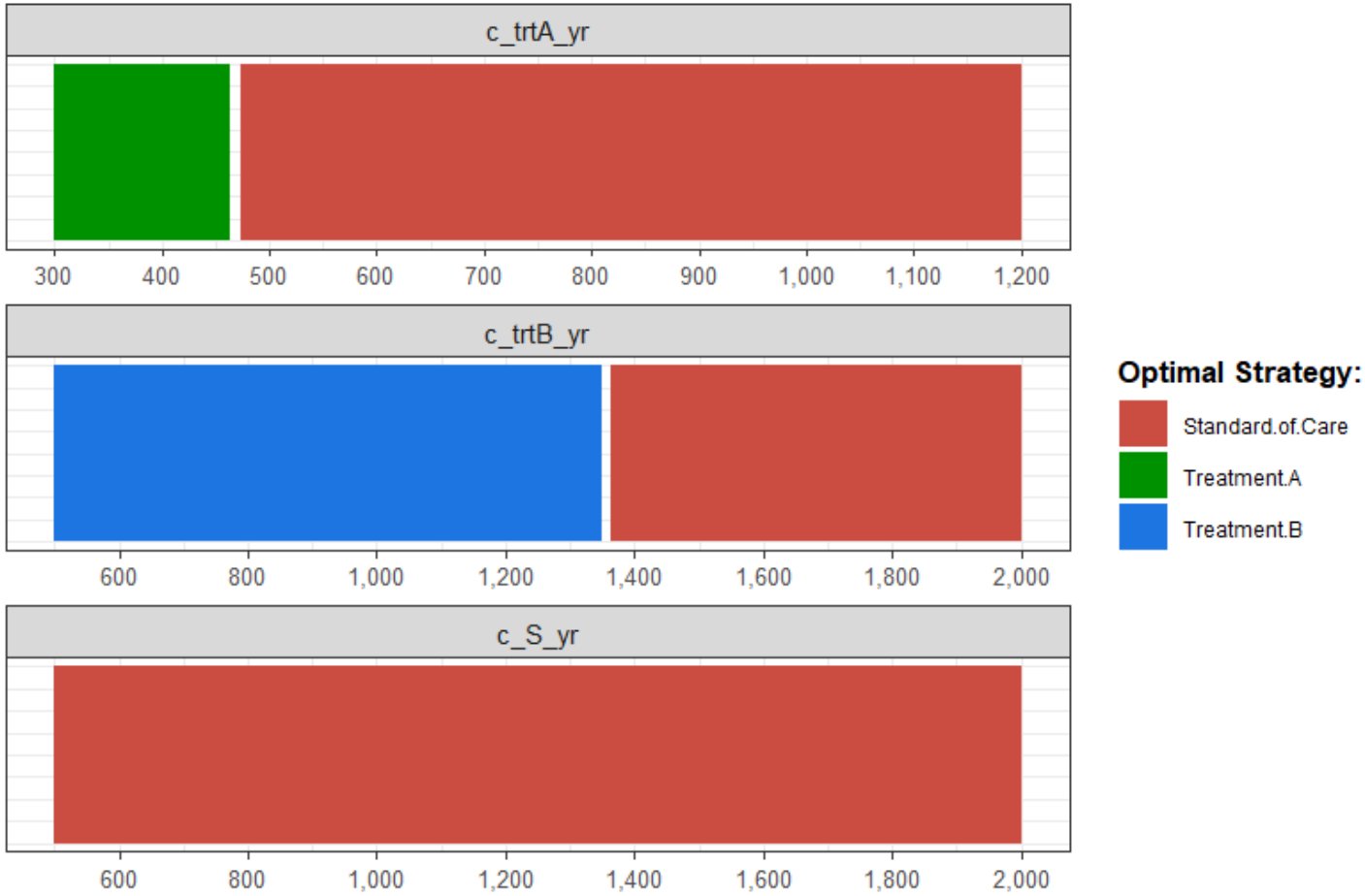
```r
## 10.2 One-way sensitivity analysis (OWSA)

options(scipen = 999) # disabling scientific notation in R
# dataframe containing all parameters, their base case values, and the min and
# max values of the parameters of interest
df_params_owsa <- data.frame(pars = c("c_trtA_yr", "c_trtB_yr", "c_S_yr"),
                             min  = c(300 , 500,   500),  # min parameter values
                             max  = c(1200, 2000, 2000)  # max parameter values
                             )
owsa_nmb  <- run_owsa_det(params_range     = df_params_owsa,    # dataframe with parameters for
                          params_basecase  = l_params_all,      # list with all parameters
                          nsamp            = 100,              # number of parameter values
                          FUN              = calculate_ce_out, # function to compute outputs
                          outcomes         = c("NMB"),         # output to do the OWSA on
                          strategies       = v_names_str,      # names of the strategies
                          n_wtp            = 5000)             # extra argument to pass to FUN

plot(owsa_nmb, txtsize = 10, n_x_ticks = 4,
     facet_scales = "free") +
     theme(legend.position = "bottom")
```

Sensitivity Analyses (Deterministic) (5)



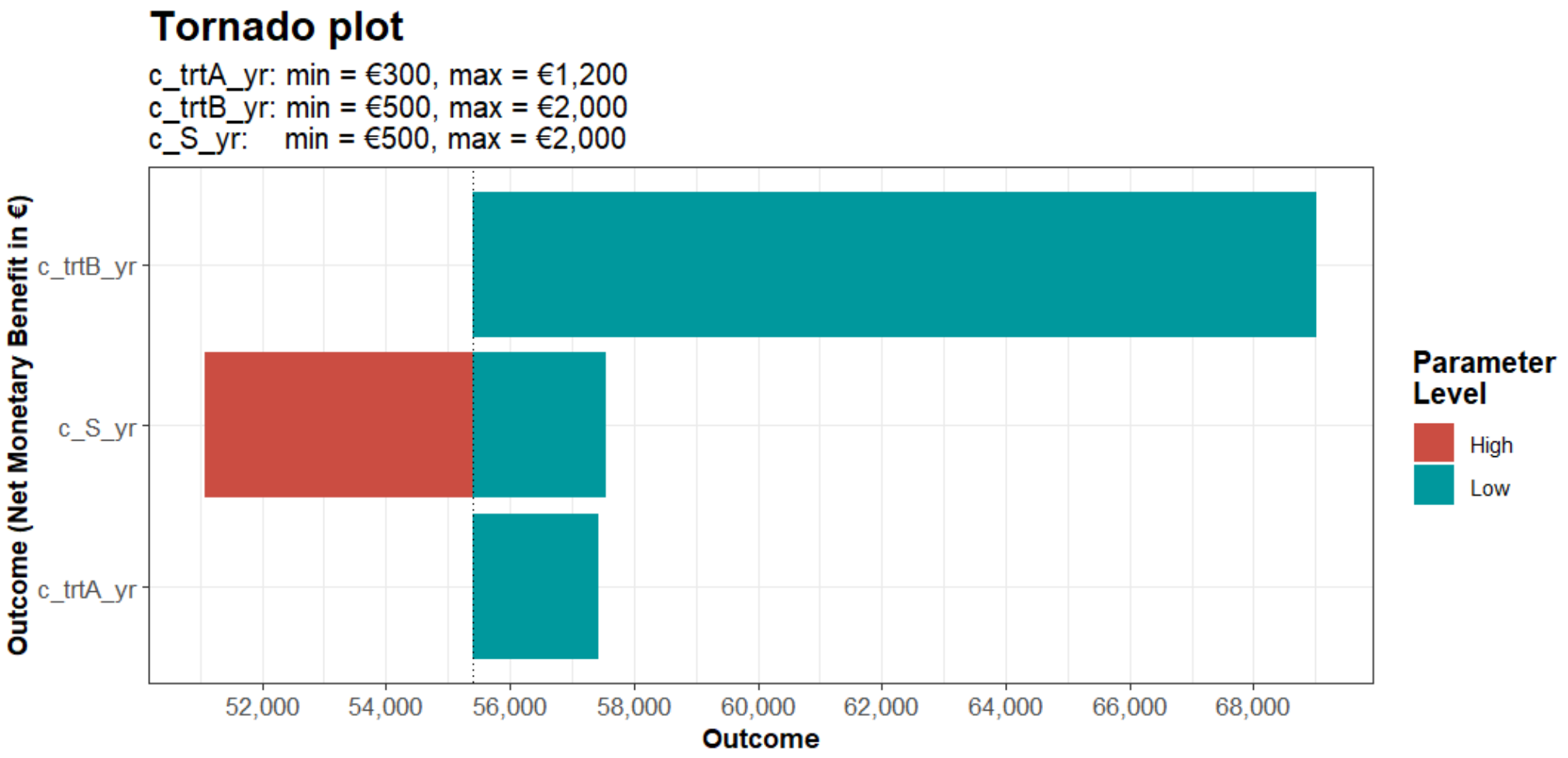
Sensitivity Analyses (Deterministic) (6)

```
### 10.2.1 Optimal strategy with OWSA

owsa_opt_strat(owsa = owsa_nmb, txtsize = 10)
```



Sensitivity Analyses (Deterministic) (6)

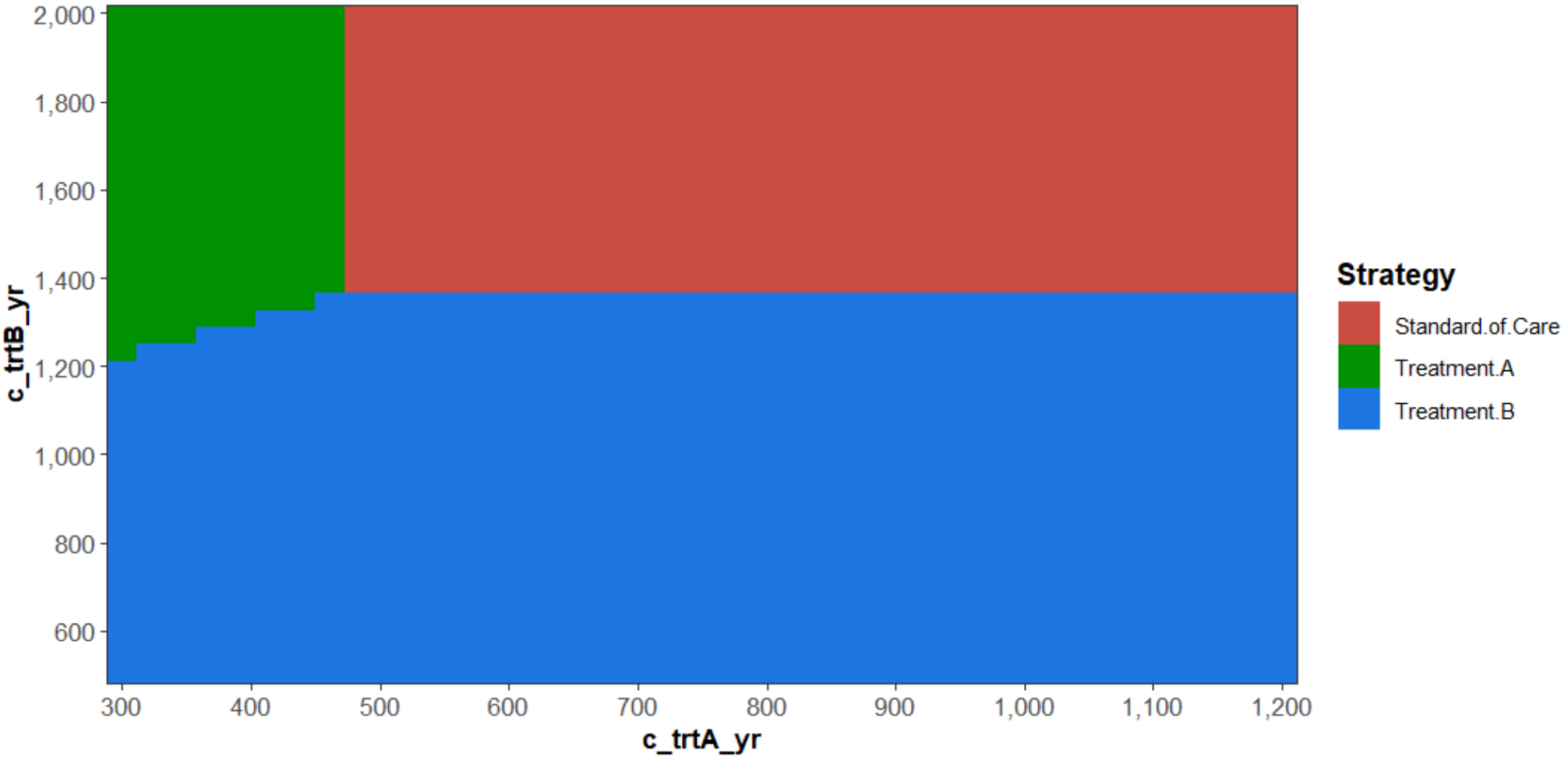
```r
### 10.2.2 Tornado plot

tornado <- owsa_tornado(owsa = owsa_nmb)
tornado + labs(title="Tornado plot",
               subtitle="c_trtA_yr: min = €300, max = €1,200\nc_trtB_yr: min = €500, max = €2,000\nc_S_yr:    min = €500, max = €2,000",
               x = "Outcome (Net Monetary Benefit in €)") +
  theme(plot.subtitle=element_text(size=12, face="plain", color="black"))
```



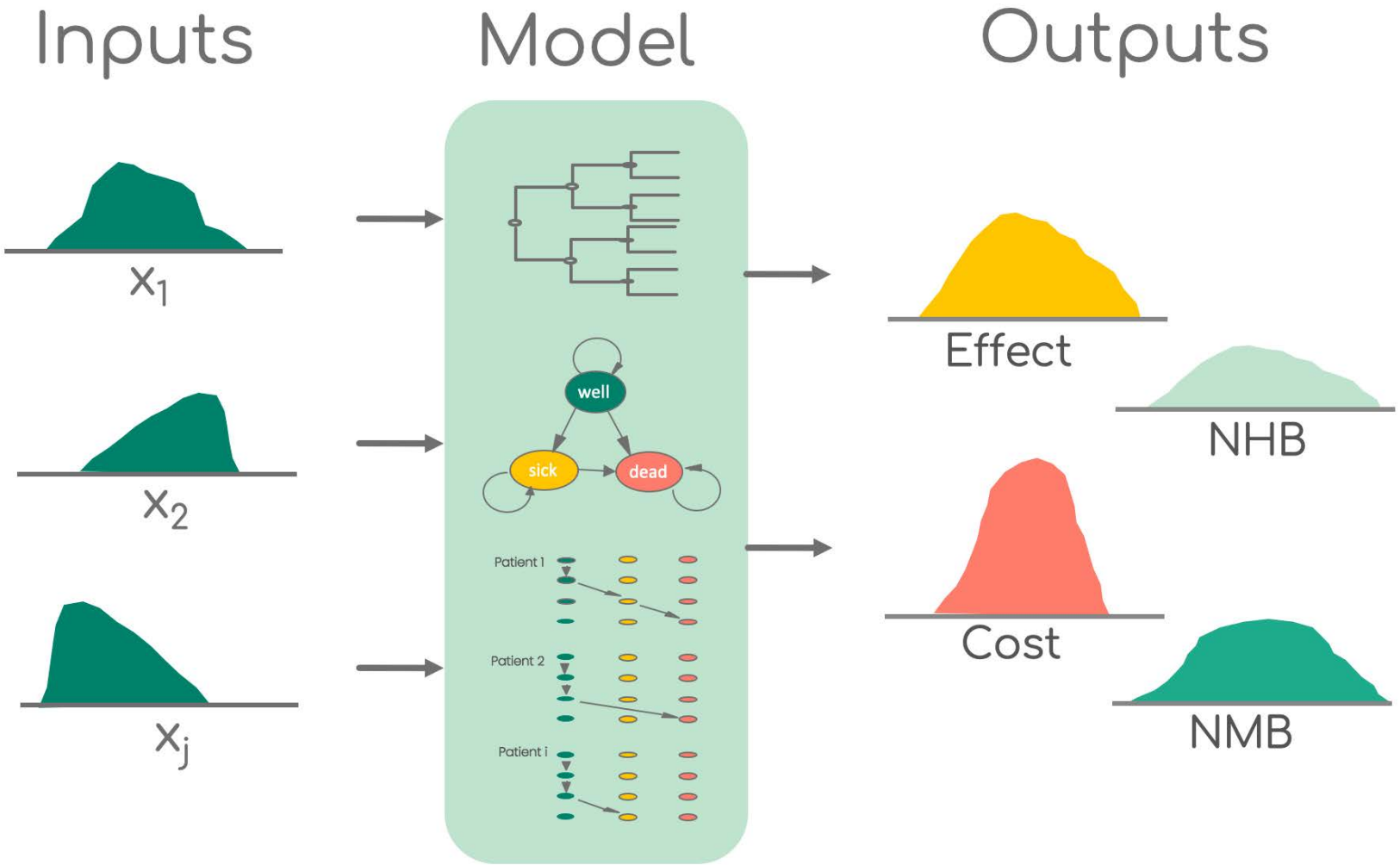
Sensitivity Analyses (Deterministic) (7)

```r
## 10.3 Two-way sensitivity analysis (TWSA)

# dataframe containing all parameters, their basecase values, and the min and
# max values of the parameters of interest
df_params_twsa <- data.frame(pars = c("c_trtA_yr", "c_trtB_yr"),
                             min  = c(300, 500),  # min parameter values
                             max  = c(1200, 2000) # max parameter values
                             )

twsa_nmb <- run_twsa_det(params_range     = df_params_twsa,     # dataframe with parameters for TWSA
                         params_basecase = l_params_all,      # list with all parameters
                         nsamp           = 40,                # number of parameter values
                         FUN             = calculate_ce_out,  # function to compute outputs
                         outcomes        = "NMB",             # output to do the TWSA on
                         strategies      = v_names_str,       # names of the strategies
                         n_wtp           = 5000)              # extra argument to pass to FUN

### 10.3.1 Plot TWSA

plot(twsa_nmb)
```

Sensitivity Analyses (Deterministic) (8)



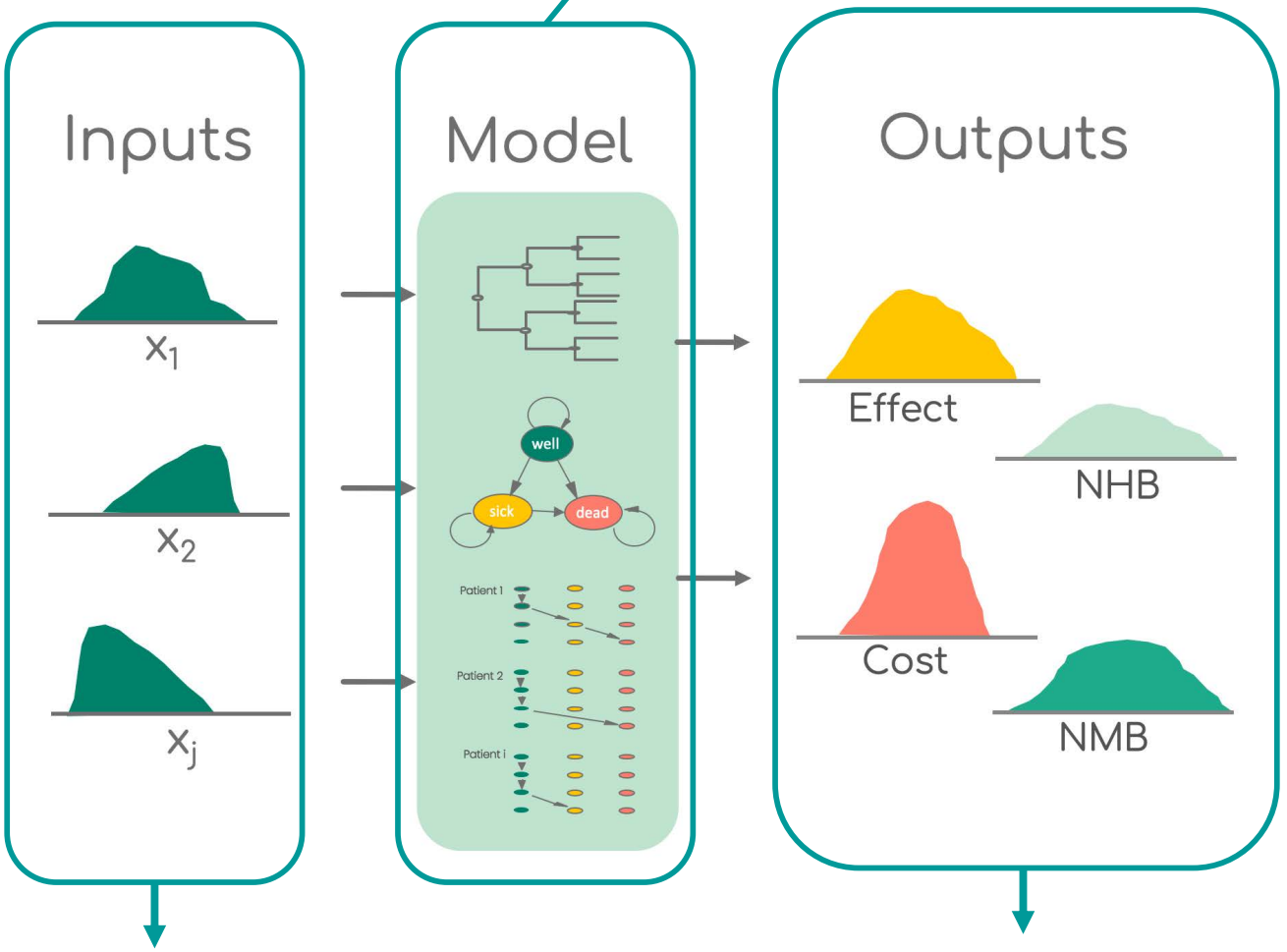
Probabilistic Sensitivity Analysis



NHB: Net health benefit ; NMB: Net monetary benefit

PSA & R

Function which runs the model using the parameter values from the list



List or data set with parameter values for each iteration

Data frame with model outcome estimates for each iteration

Sensitivity Analyses (Probabilistic) (9)

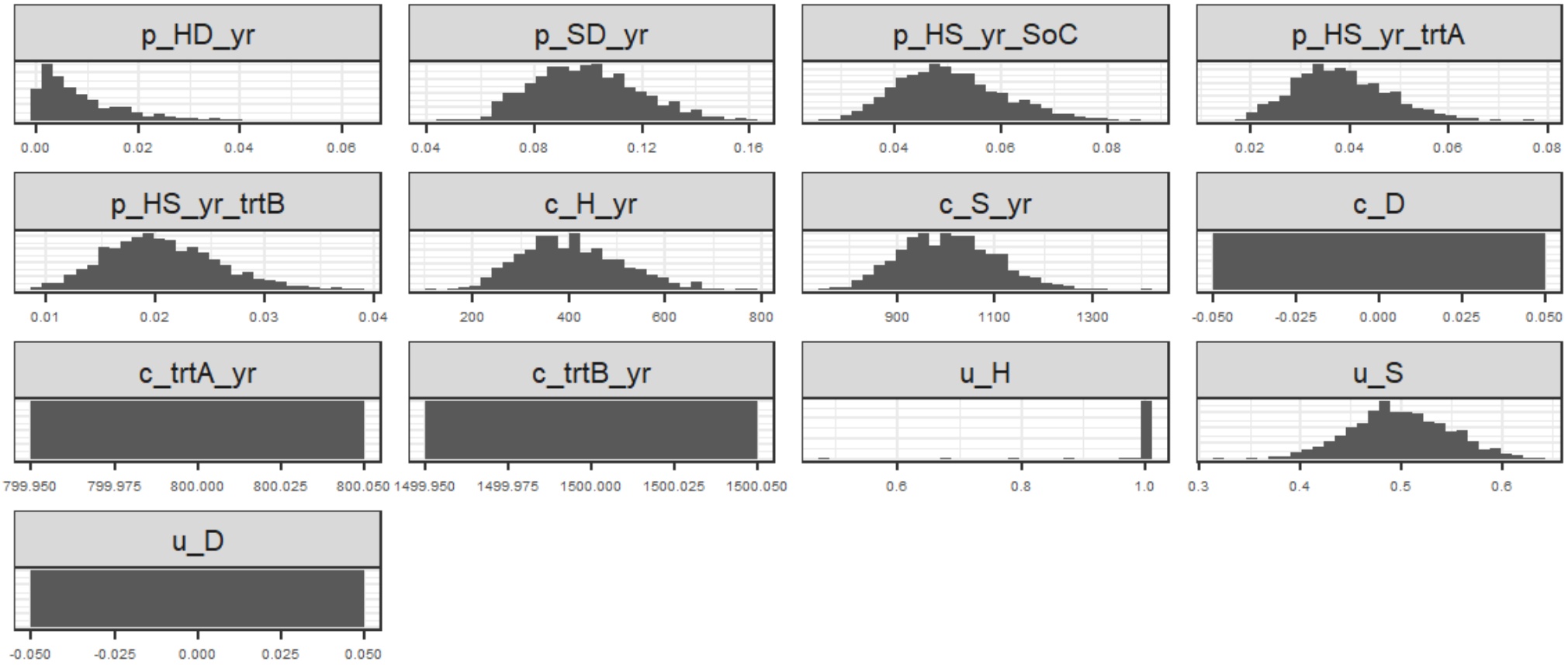
```
399  # 11 Probabilistic Sensitivity Analysis (PSA)
400
401  ## 11.1 Model input
402
403  # Store the parameter names into a vector
404  v_names_params <- names(l_params_all)
405
406  ## Test functions to generate CE outcomes and PSA dataset
407  # Test function to compute CE outcomes
408  calculate_ce_out(l_params_all)
409
410  # Test function to generate PSA input dataset
411  generate_psa_params(10)
```

```
> generate_psa_params(10)
       p_HD_yr    p_SD_yr p_HS_yr_SoC p_HS_yr_trtA p_HS_yr_trtB    c_H_yr    c_S_yr c_D c_trtA_yr c_trtB_yr u_H       u_S u_D
1  0.0008905932 0.07962620  0.04080578   0.07084322   0.01716266 390.7068  984.6521   0       800      1500   1 0.5399242   0
2  0.0092979184 0.14336539  0.06172540   0.05135318   0.02601948 393.7788  820.5960   0       800      1500   1 0.5418116   0
3  0.0069371243 0.08285488  0.04973326   0.03317056   0.01537997 388.4159 1138.0161   0       800      1500   1 0.5599530   0
4  0.0148746692 0.12408528  0.03997063   0.04320045   0.01843236 348.6654  858.9336   0       800      1500   1 0.5384445   0
5  0.0008164595 0.09565414  0.07481755   0.05311687   0.01753576 383.0443 1066.2746   0       800      1500   1 0.5669157   0
6  0.0144162389 0.13613101  0.03950458   0.03384405   0.01839549 337.9317 1056.6110   0       800      1500   1 0.5479511   0
7  0.0291361361 0.10579557  0.05236064   0.01984253   0.02530839 375.8565 1061.5311   0       800      1500   1 0.4594113   0
8  0.0072422302 0.06572414  0.04388866   0.07142634   0.01843345 443.4329 1193.5129   0       800      1500   1 0.4863897   0
9  0.0173178373 0.12304579  0.04945120   0.04120239   0.02108065 265.5241  800.1153   0       800      1500   1 0.5303334   0
10 0.0038339205 0.12801141  0.04737147   0.03291192   0.02603099 474.4756  958.2335   0       800      1500   1 0.5053532   0
>
```

Sensitivity Analyses (Probabilistic) (10)

```r
## Generate PSA dataset
# Number of simulations
n_sim <- 1000

# Generate PSA input dataset
df_psa_input <- generate_psa_params(n_sim = n_sim)
# First six observations
head(df_psa_input)

### Histogram of parameters
ggplot(melt(df_psa_input, variable.name = "Parameter"), aes(x = value)) +
  facet_wrap(~Parameter, scales = "free") +
  geom_histogram(aes(y = after_stat(density))) +
  ylab("") +
  theme_bw(base_size = 16) +
  theme(axis.text = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.y  = element_blank(),
        axis.ticks.y = element_blank())

```

Sensitivity Analyses (Probabilistic) (11)



Sensitivity Analyses (Probabilistic) (12)

```r
## 11.2 Run PSA

# 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

# Conduct probabilistic sensitivity analysis
# Run Markov model on each parameter set of PSA input dataset
n_time_init_psa_series <- Sys.time()
for (i in 1:n_sim) { # i <- 1
  l_psa_input <- update_param_list(l_params_all, df_psa_input[i,])
  # Outcomes
  l_out_ce_temp  <- calculate_ce_out(l_psa_input)
  df_c[i, ]  <- l_out_ce_temp$Cost
  df_e[i, ]  <- l_out_ce_temp$Effect
  # Display simulation progress
  if (i/(n_sim/100) == round(i/(n_sim/100), 0)) { # display progress every 5%
    cat('\r', paste(i/n_sim * 100, "% done", sep = " "))
  }
}
```

Sensitivity Analyses (Probabilistic) (13)

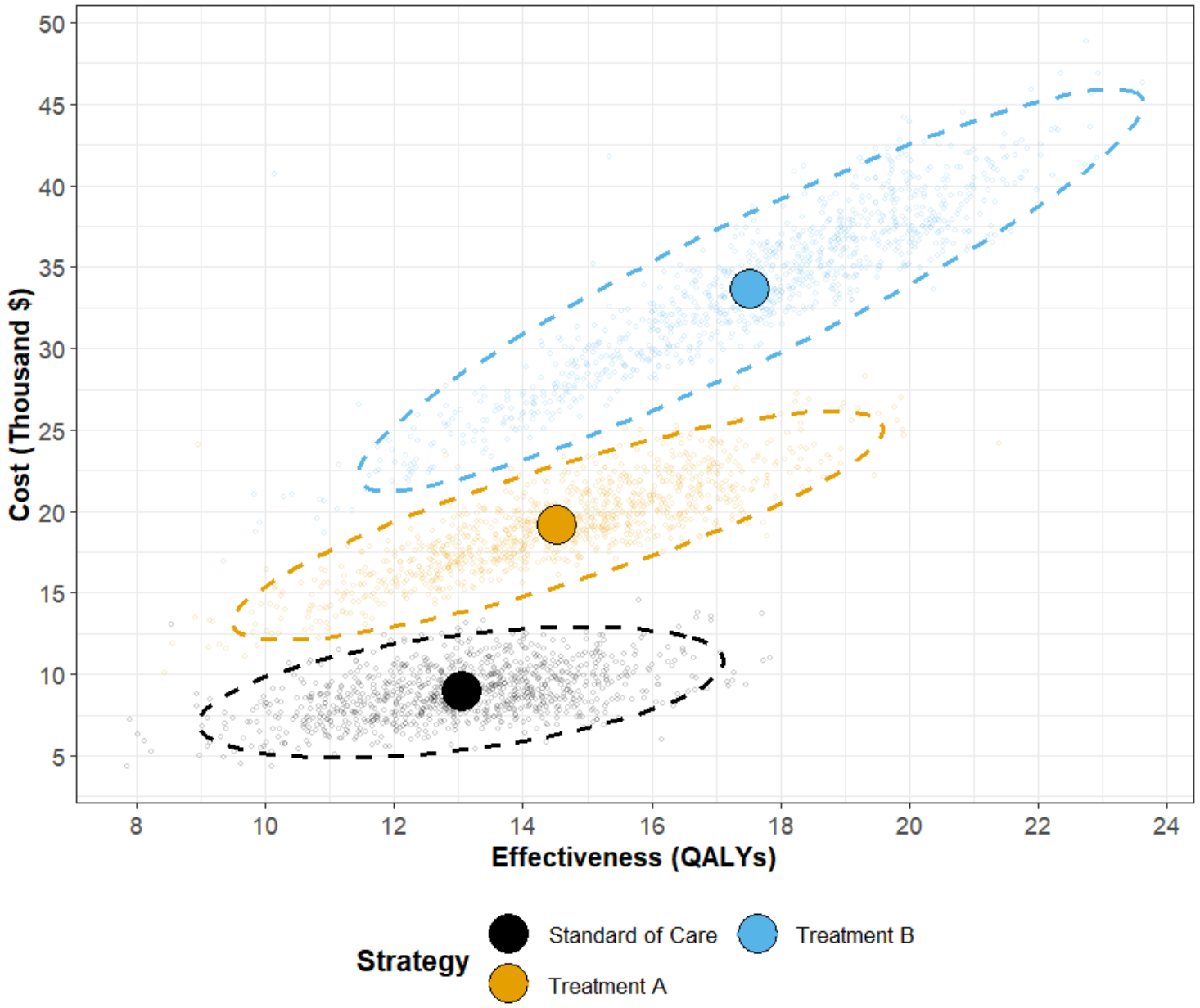
```
462 n_time_end_psa_series <- Sys.time()
463 n_time_total_psa_series <- n_time_end_psa_series - n_time_init_psa_series
464 print(paste0("PSA with ", scales::comma(n_sim), " simulations run in series in ",
465              round(n_time_total_psa_series, 2), " ",
466              units(n_time_total_psa_series)))
```

```
+              units(n_time_total_psa_series)))
[1] "PSA with 1,000 simulations run in series in 9.96 secs"
>
```

Sensitivity Analyses (Probabilistic) (14)

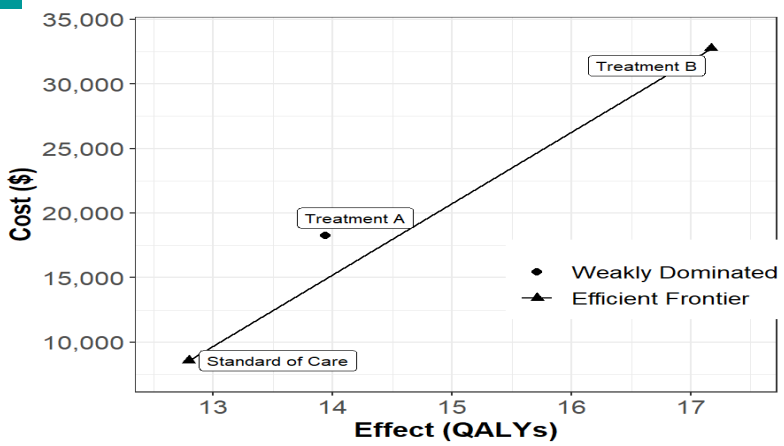
```r
# 11.3 Visualize PSA results for CEA

### Create PSA object
l_psa <- make_psa_obj(cost          = df_c,
                      effectiveness = df_e,
                      parameters    = df_psa_input,
                      strategies    = v_names_str)
l_psa$strategies <- v_names_str
colnames(l_psa$effectiveness) <- v_names_str
colnames(l_psa$cost) <- v_names_str

# Vector with willingness-to-pay (WTP) thresholds.
v_wtp <- seq(0, 30000, by = 1000)

## 11.3.1 Cost-Effectiveness Scatter plot

### Cost-Effectiveness Scatter plot
txtsize <- 13
gg_scattter <- plot_psa(l_psa, txtsize = txtsize) +
  ggthemes::scale_color_colorblind() +
  ggthemes::scale_fill_colorblind() +
  scale_y_continuous("Cost (Thousand $)",
                     breaks = number_ticks(10),
                     labels = function(x) x/1000) +
  xlab("Effectiveness (QALYs)") +
  guides(col = guide_legend(nrow = 2)) +
  theme(legend.position = "bottom")
gg_scattter
```

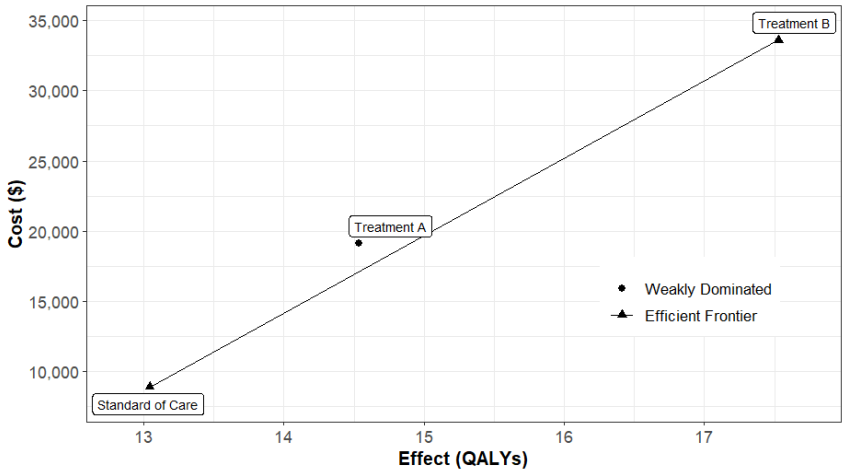
Sensitivity Analyses (Probabilistic) (15)



Sensitivity Analyses (Probabilistic) (16)

```r
## 11.3.2 Incremental cost-effectiveness ratios (ICERs) with probabilistic output

### Incremental cost-effectiveness ratios (ICERs) with probabilistic output
# Compute expected costs and effects for each strategy from the PSA
df_out_ce_psa <- summary(l_psa)
df_cea_psa <- calculate_icers(cost       = df_out_ce_psa$meanCost,
                              effect     = df_out_ce_psa$meanEffect,
                              strategies = df_out_ce_psa$Strategy)
df_cea_psa

## 11.3.3 Plot cost-effectiveness frontier with probabilistic output

### Plot cost-effectiveness frontier with probabilistic output
plot_icers(df_cea_psa, label = "all", txtsize = txtsize) +
  expand_limits(x = max(table_cea$QALYs) + 0.1) +
  theme(legend.position = c(0.8, 0.3))
```

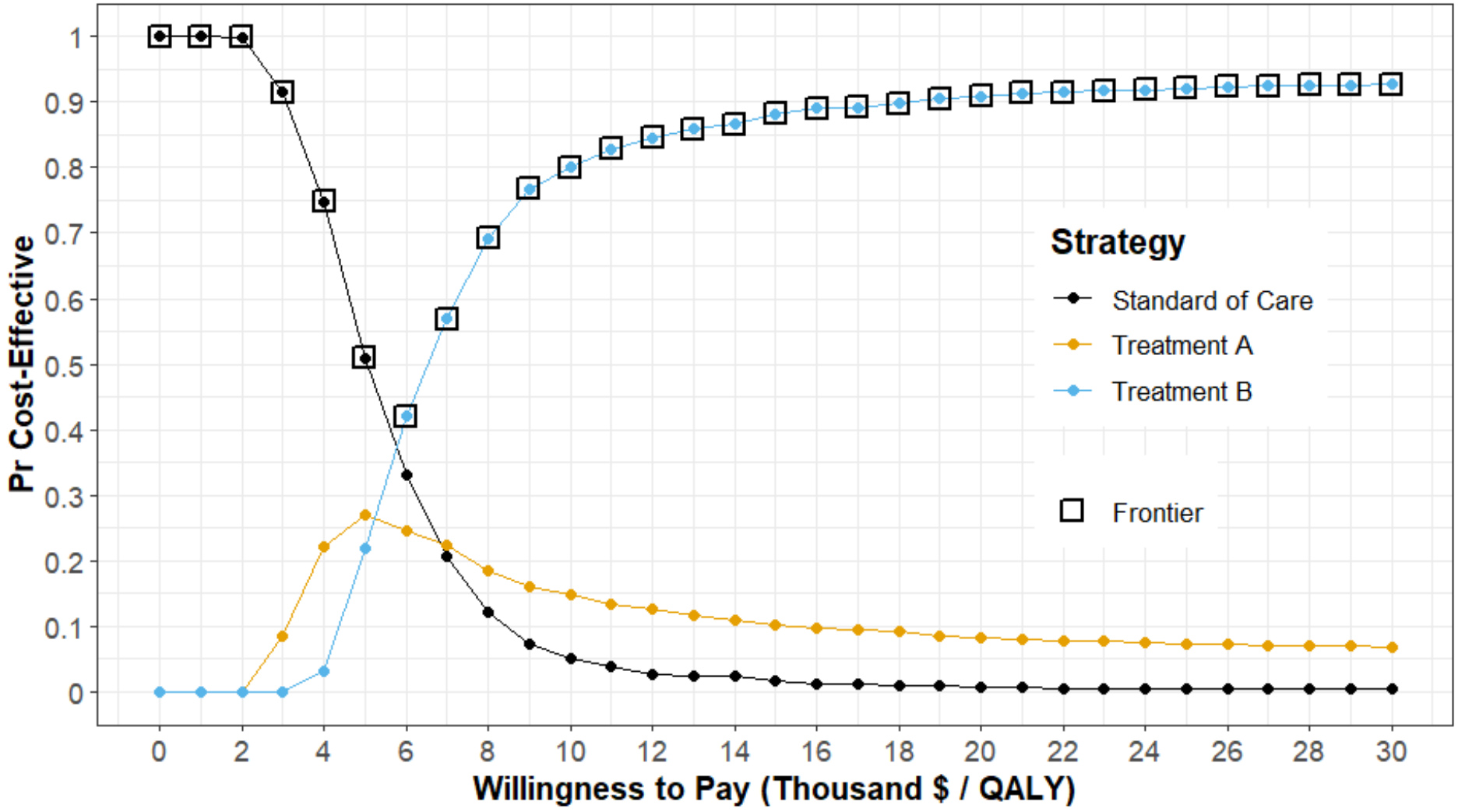




Sensitivity Analyses (Probabilistic) (17)

```
514  ## 11.3.4 Cost-effectiveness acceptability curves (CEACs) and frontier (CEAF)
515
516  ### Cost-effectiveness acceptability curves (CEACs) and frontier (CEAF)
517  ceac_obj <- ceac(wtp = v_wtp, psa = l_psa)
518  # Regions of highest probability of cost-effectiveness for each strategy
519  summary(ceac_obj)
520  # CEAC & CEAF plot
521  gg_ceac <- plot_ceac(ceac_obj, txtsize = txtsize, xlim = c(0, NA), n_x_ticks = 14) +
522    ggthemes::scale_color_colorblind() +
523    ggthemes::scale_fill_colorblind() +
524    theme(legend.position = c(0.8, 0.48))
525  gg_ceac
```

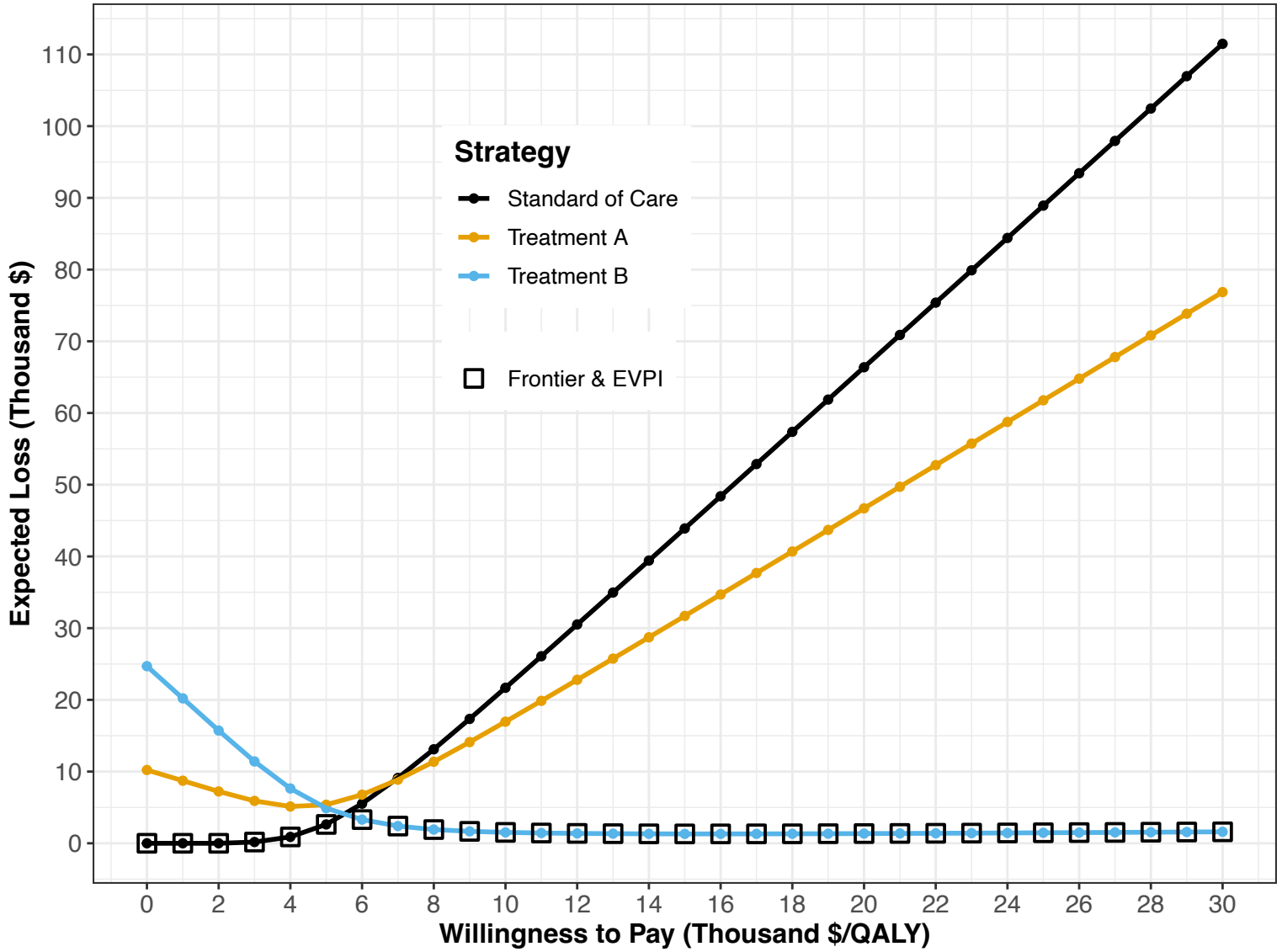
Sensitivity Analyses (Probabilistic) (18)



Advanced Topics: Expected Loss and Value of Information

```
527 ## 11.3.5 Expected Loss Curves (ELCs)
528
529 ### Expected Loss Curves (ELCs)
530 elc_obj <- calc_exp_loss(wtp = v_wtp, psa = l_psa)
531 elc_obj
532
533 # ELC plot
534 gg_elc <- plot_exp_loss(elc_obj, log_y = FALSE,
535                         txtsize = txtsize, xlim = c(0, NA), n_x_ticks = 14,
536                         col = "full") +
537   ggthemes::scale_color_colorblind() +
538   ggthemes::scale_fill_colorblind() +
539   # geom_point(aes(shape = as.name("Strategy"))) +
540   scale_y_continuous("Expected Loss (Thousand $)",
541                      breaks = number_ticks(10),
542                      labels = function(x) x/1000) +
543   theme(legend.position = c(0.4, 0.7),)
544 gg_elc
545
546 ## 11.3.6 Expected value of perfect information (EVPI)
547
548 ### Expected value of perfect information (EVPI)
549 evpi <- calc_evpi(wtp = v_wtp, psa = l_psa)
550 # EVPI plot
551 gg_evpi <- plot_evpi(evpi, effect_units = "QALY",
552                      txtsize = txtsize, xlim = c(0, NA), n_x_ticks = 14) +
553   scale_y_continuous("EVPI (Thousand $)",
554                      breaks = number_ticks(10),
555                      labels = function(x) x/1000)
556 gg_evpi
```

Advanced Topics: Expected Loss and Value of Information



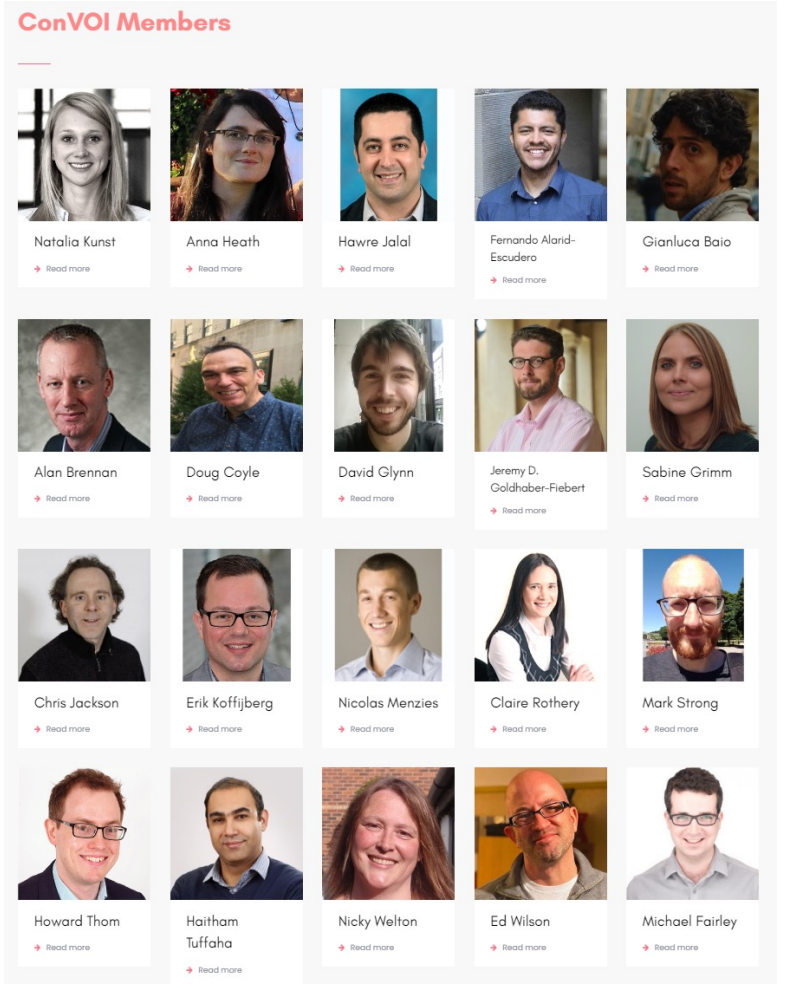
Use the ConVOI website to learn more about these topics (https://convoy-group.org)

ConVOI
Collaborative Network for Value of Information

What is a Value of Information analysis?

Useful resources for Value of Information

- Reading resources
- Video resources
- VoI tools
- ConVOI GitHub
- ConVOI training and courses

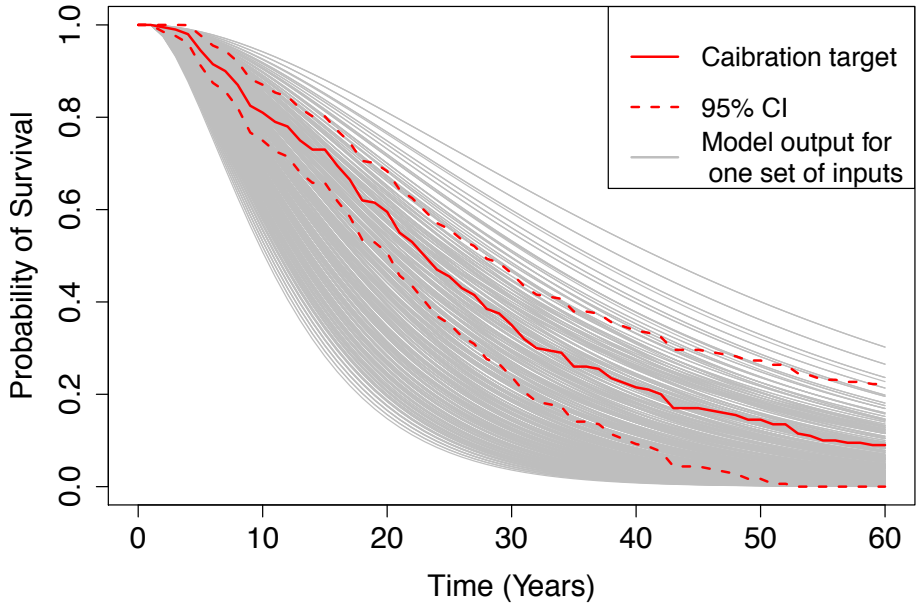


R session

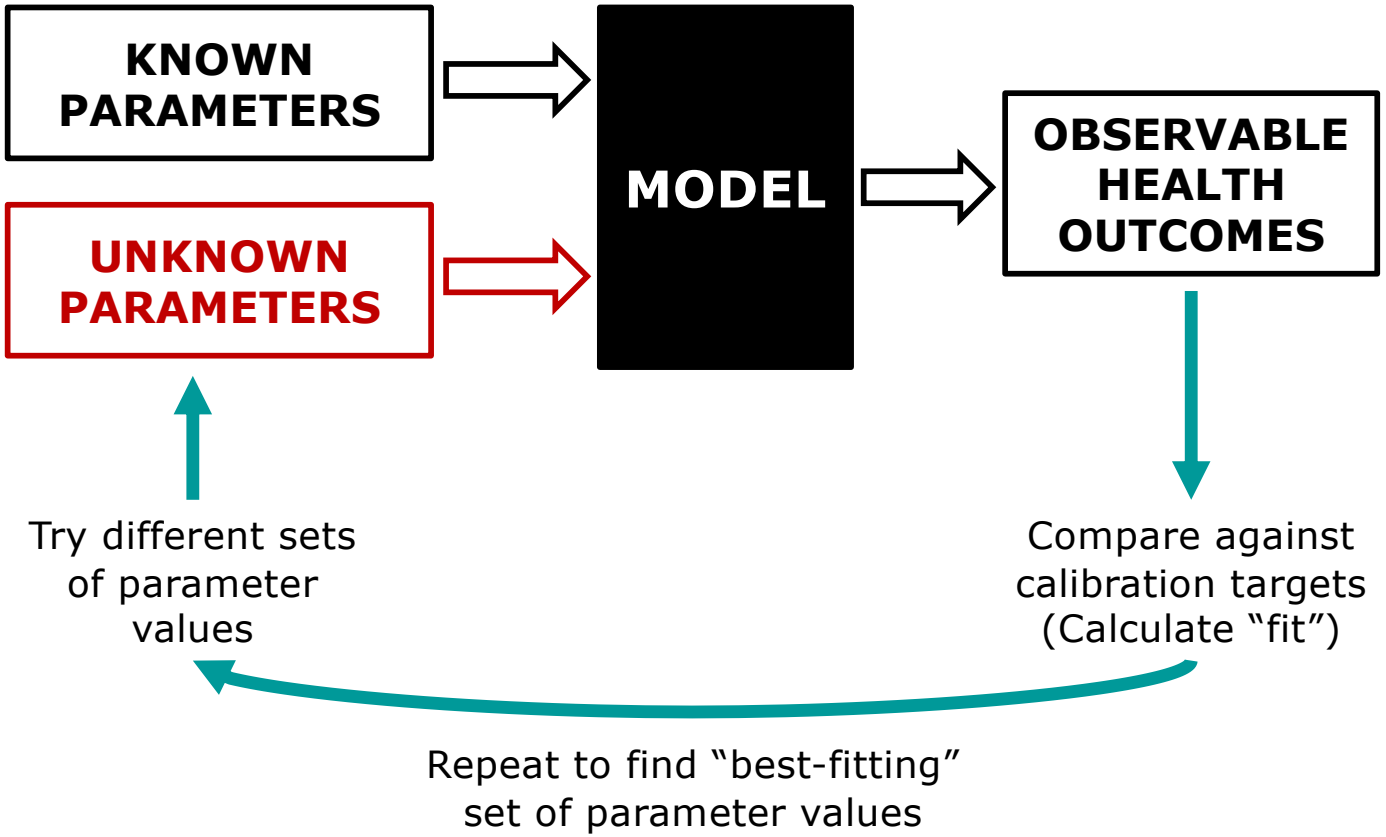
Model Calibration

Calibration definition

- Process of adjusting model input parameter values to match data on an outcome of interest (e.g., survival, prevalence, or incidence)
- Outcome(s) of interest = "calibration targets"



Calibration process



Calibration targets

- Empirical data to be replicated by the model
- Summary statistics (e.g. mean age of cancer diagnosis) or series of observations (e.g. age-specific incidence)
- Can calibrate to multiple targets (e.g. survival and prevalence) simultaneously
- Model should be able to output the outcomes of interest

Calculating “fit”

- Goodness-of-Fit (GoF) is the quantitative measure of how the model is replicating the target data
- Different ways to measure GoF
  - Distance
  - Likelihood
- Notation
  - M : a mathematical model (e.g., Markov model)
  - θ : Set of K parameters to be calibrated
  - $\phi = M(\theta)$: Model output for parameter set θ
  - y : Values of T calibration targets

Distance GoF measures

- Sum of squared errors

$$SSE(\theta) = \sum_{i=1}^{T} \left(y_i - M_i(\theta)\right)^2$$

- Weighted sum of squared errors
  - Assign different weight to different targets (w_i)
  - Often, $w_i = \frac{1}{\sigma^2}$

$$WSSE(\theta) = \sum_{i=1}^{T} w_i \left(y_i - M_i(\theta)\right)^2$$

Likelihood as GoF

- How likely is the observed data to have come from a model M with set of parameter values θ ?
- Assuming targets are independent, overall likelihood is the product of individual likelihoods

$$L(y|M(\theta)) = \prod_{i=1}^{T} L_i(y_i|M(\theta))$$

- Generally, we work with log-likelihood

$$\mathcal{L}(y|M(\theta)) = \sum_{i=1}^{T} \log L_i(y_i|M_i(\theta)$$

Commonly used likelihoods

- **Normal distribution**

$$\log L(y_i, \sigma_i | M_i(\theta)) = -\frac{1}{2}\log\left(2\pi\sigma_i^2\right) - \frac{1}{\sigma_i^2}\left(y_i - M_i(\theta)\right)^2$$

  - In R: `dnorm(x=`y_i`, mean=`$M_i(\theta)$`, sd=`σ_i`, log=T)`

- **Binomial distribution**

$$\log L(y_i, n_i | M_i(\theta)) = \log\left(\binom{n_i}{y_i}\right) + y_i \log\left(M_i(\theta)\right) + (n_i - y_i)\log\left(1 - M_i(\theta)\right)$$

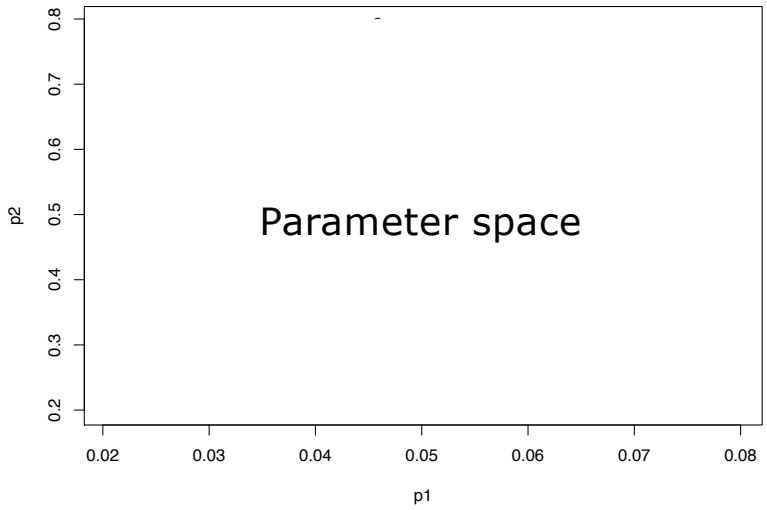
  - In R: `dbinom(x=`y_i`, size=`n_i`, prob=`$M_i(\theta)$`, log=T)`

- **Multinomial distribution**

  - In R: `dmultinom(x=`y_i`, prob=`$M_i(\theta)$`, log=T)`

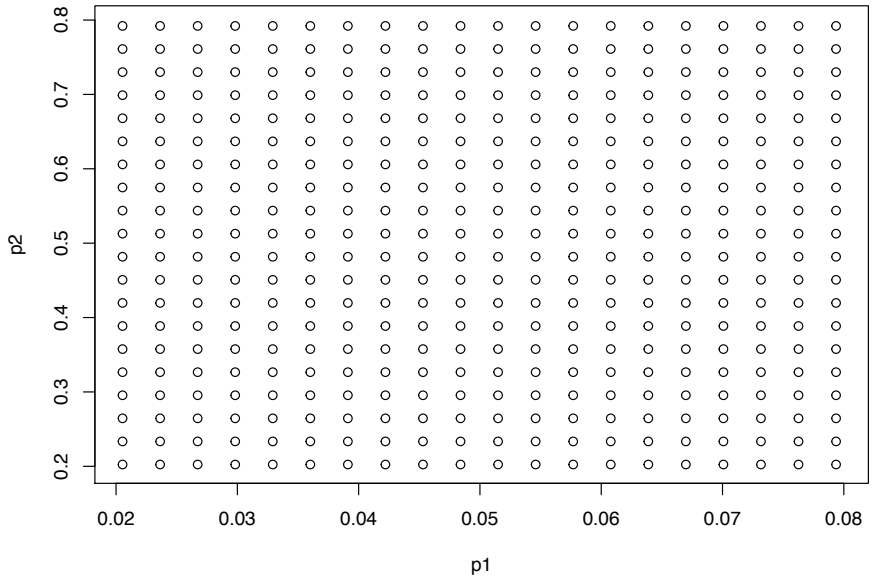
Search Strategy

- Define plausible ranges for parameter whose values are unknown
- Use a search strategy to “search” through the input parameter space
  - Run the model for sets of parameter values generated by search strategy



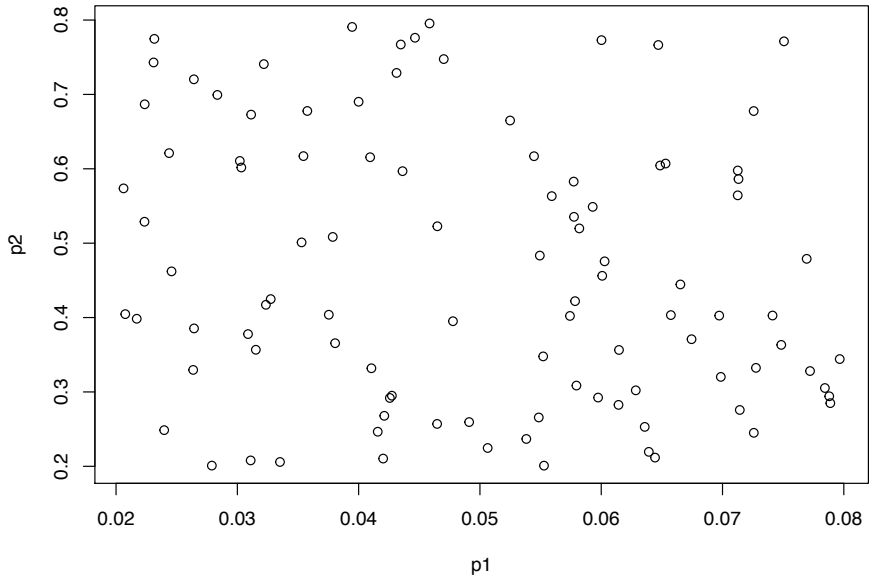
Grid Search

- Run model for all possible combinations of parameter values
- Often infeasible



Random Search

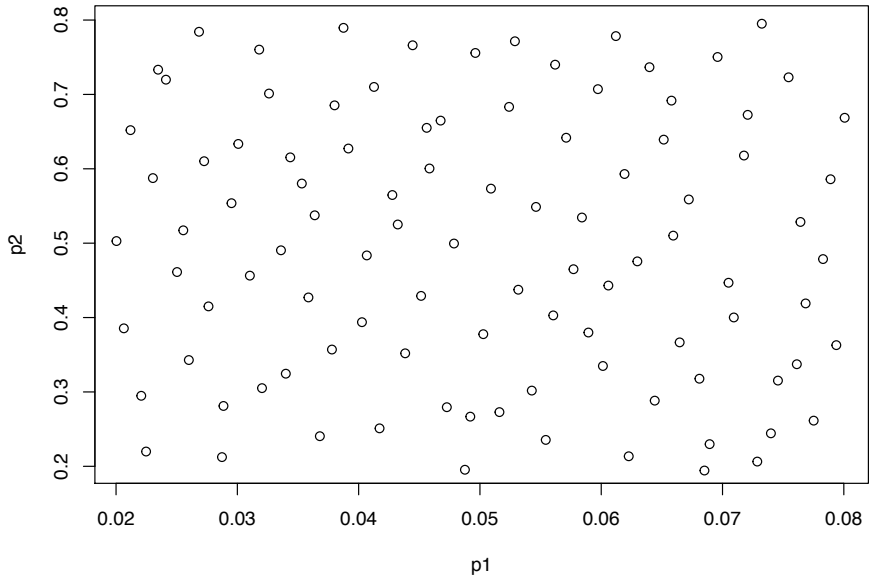
- Randomly sample a large number of parameter value sets from probabilistic distributions
- Use "Latin hypercube sampling" (LHS) to ensure sample captures the full parameter space



Random sample

Random Search

- Randomly sample a large number of parameter value sets from probabilistic distributions
- Use "Latin hypercube sampling" (LHS) to ensure sample captures the full parameter space



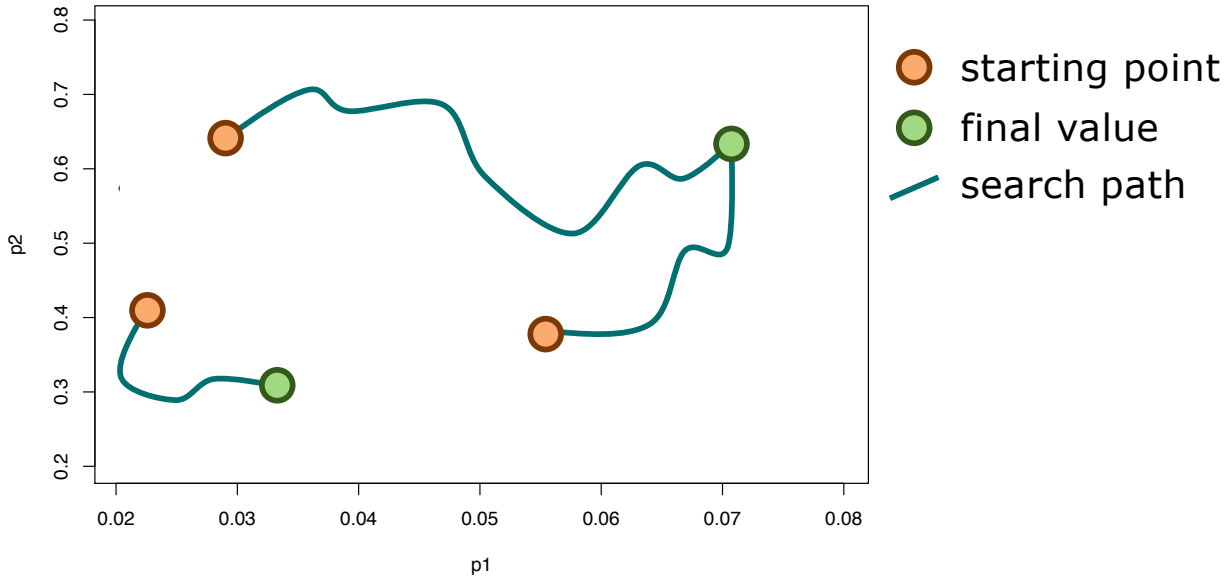
LHS sample

Iterative Search

- Use fits of past input values to determine which input values to try next
- Directed methods
  - Nelder-Mead (simplex method)
  - Gradient–descent and others
- Meta-heuristic algorithms
  - Genetic algorithms
  - Simulated annealing

Nelder-Mead Algorithm

- Downhill simplex method
- Must be run multiple times for different starting points to avoid local extrema



Example:
Calibrating a 3-state cancer model

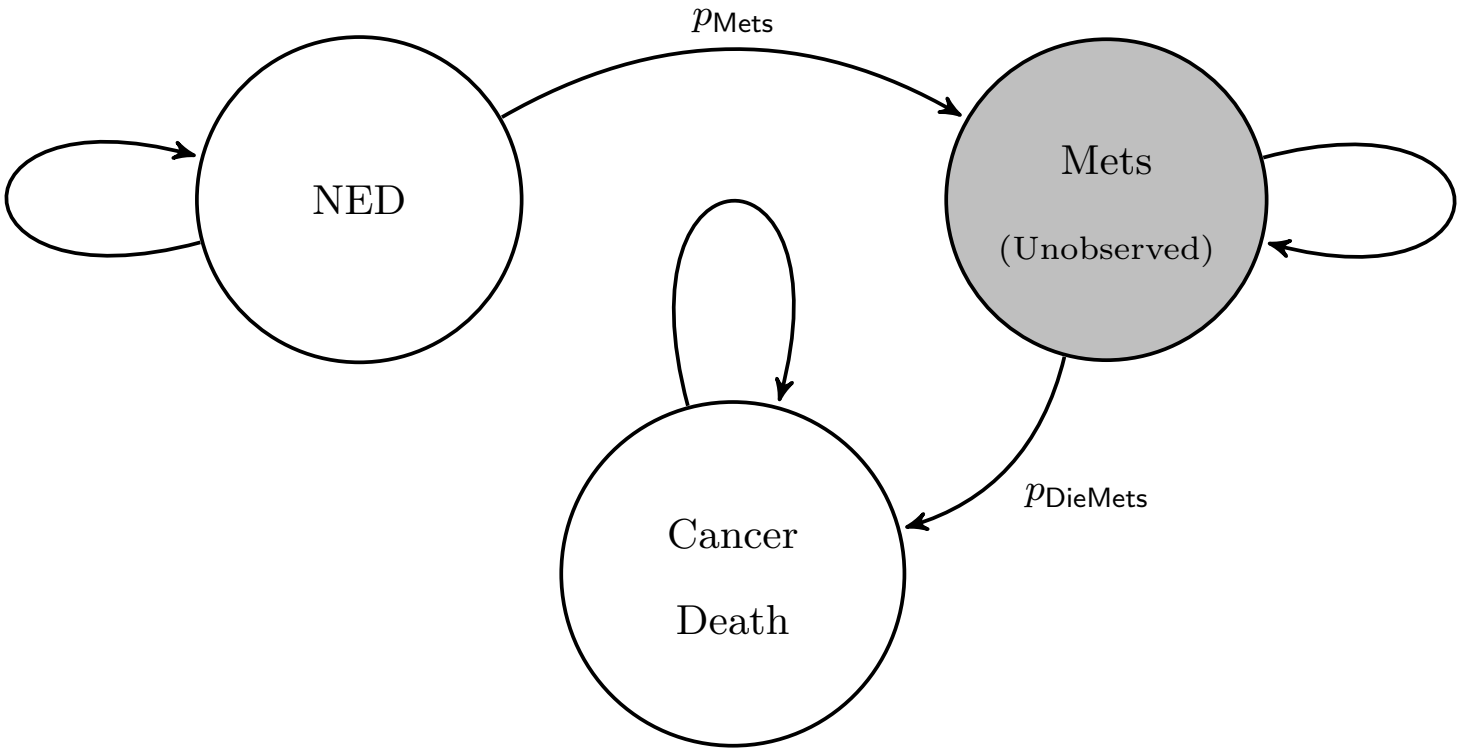
3-state cancer model

- Relative survival as reported by the Surveillance, Epidemiology, and End Results (SEER) Program, represents cancer survival in the absence of other causes of death
- Cancer Markov models often have a distant metastasis state, a state not directly observed in SEER, from which cancer deaths are presumed to occur
- These models can then be used to evaluate interventions that affect transitions to and from this state

Alarid-Escudero F, Maclehose RF, Peralta Y, Kuntz KM, & Enns, EA. Non-identifiability in model calibration and implications for medical decision making. *Medical Decision Making,* 2018;38(7):810–21.

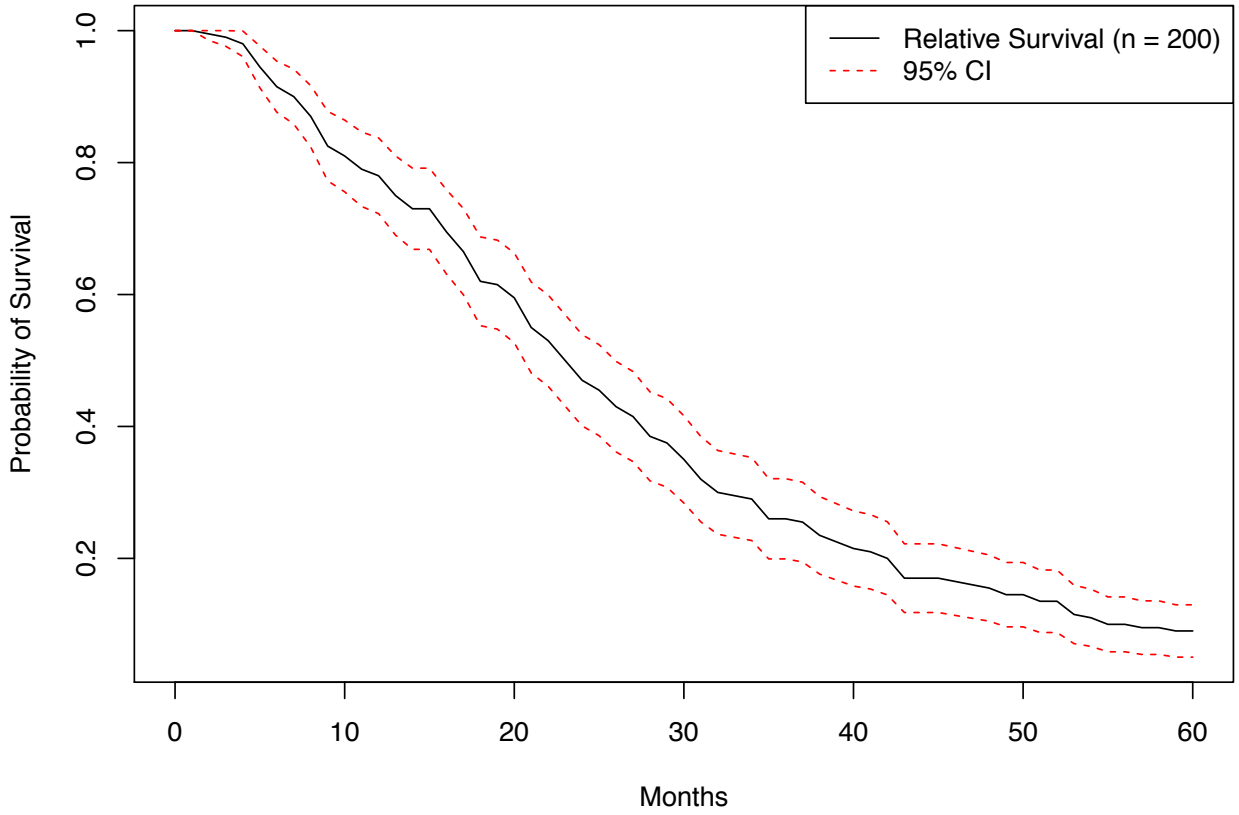
3-state cancer model

- Two unknown transition probabilities
  - p.Mets: Monthly risk of developing distant recurrence, range = [0.04, 0.16]
  - p.DieMets: Monthly risk of cancer death, range = [0.04, 0.16]



Target: Relative survival

- Data frame “CRS_targets$Surv” stored in data file “CRS_CalibTargets.RData”



Calibration R code template

```
### Load calibration targets ###


### Load model as a function ###


### Specify calibration parameters ###


### Calibrate ###


### Explore best-fitting input sets ###
```

R Session

Initialize and load libraries (00_prep.R)

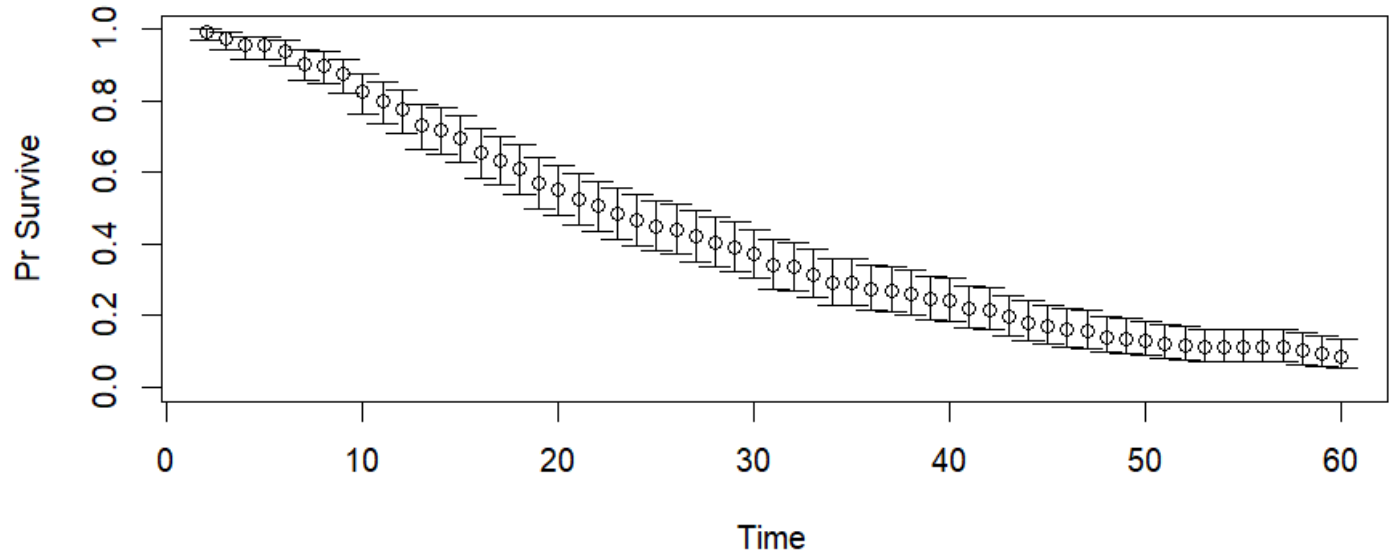
```r
### Run all code in this script to download the desired R packages
### from either CRAN or GitHub


# first download and use this package to conveniently install other packages
if (!require('pacman')) install.packages('pacman'); library(pacman)

# load (install if required) packages from CRAN
p_load("devtools", "lhs", "plotrix", "psych", "DEoptim", "matrixStats", "scatterplot3d")

# Install IMIS
devtools::install_version("IMIS", version = "0.1", repos = "http://cran.us.r-project.org")

```

Random Search (CRS_Calib_RandomSearch.R)

```
32 ##########################################################
33 ######  Load packages and function files  ####
34 ##########################################################
35 # calibration functionality
36 library(lhs)
37
38 # visualization
39 library(plotrix)
40 library(psych)
41
42
43 ##########################################################
44 ######  Load target data  ######
45 ##########################################################
46 load("CRS_CalibTargets.RData")
47 lst_targets <- CRS_targets
```

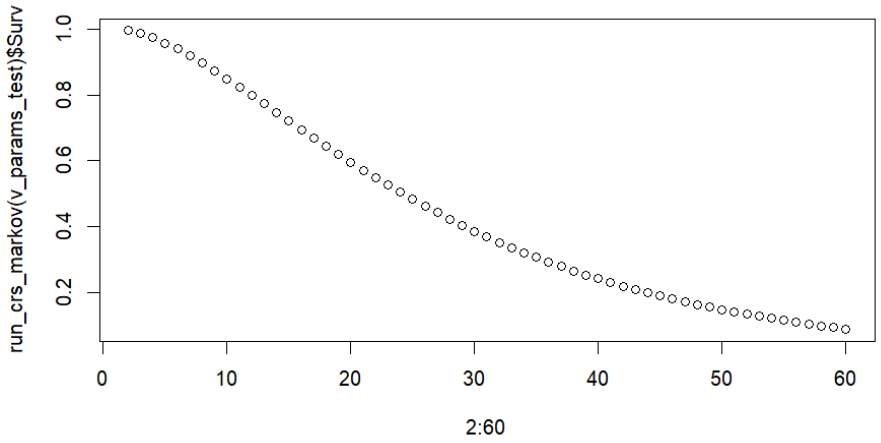
Plot targets

```
49  # Plot the targets
50
51  # TARGET 1: Survival ("Surv")
52  plotrix::plotCI(x = lst_targets$Surv$time, y = lst_targets$Surv$value,
53                  ui = lst_targets$Surv$ub,
54                  li = lst_targets$Surv$lb,
55                  ylim = c(0, 1),
56                  xlab = "Time", ylab = "Pr Survive")
57
58  # TARGET 2: (if you had more...)
59  # plotrix::plotCI(x = lst_targets$Target2$time, y = lst_targets$Target2$value,
60  #                 ui = lst_targets$Target2$ub,
61  #                 li = lst_targets$Target2$lb,
62  #                 ylim = c(0, 1),
63  #                 xlab = "Time", ylab = "Target 2")
64
```



Model as function

```r
#####################################################################
######  Load model as a function  ######
#####################################################################
# - inputs are parameters to be estimated through calibration
# - outputs correspond to the target data

source("CRS_MarkovModel_Function.R") # creates the function run_crs_markov()

# Check that it works
v_params_test <- c(p_Mets = 0.10, p_DieMets = 0.05)
plot(y= run_crs_markov(v_params_test)$Surv, x=2:60) # It works!
```



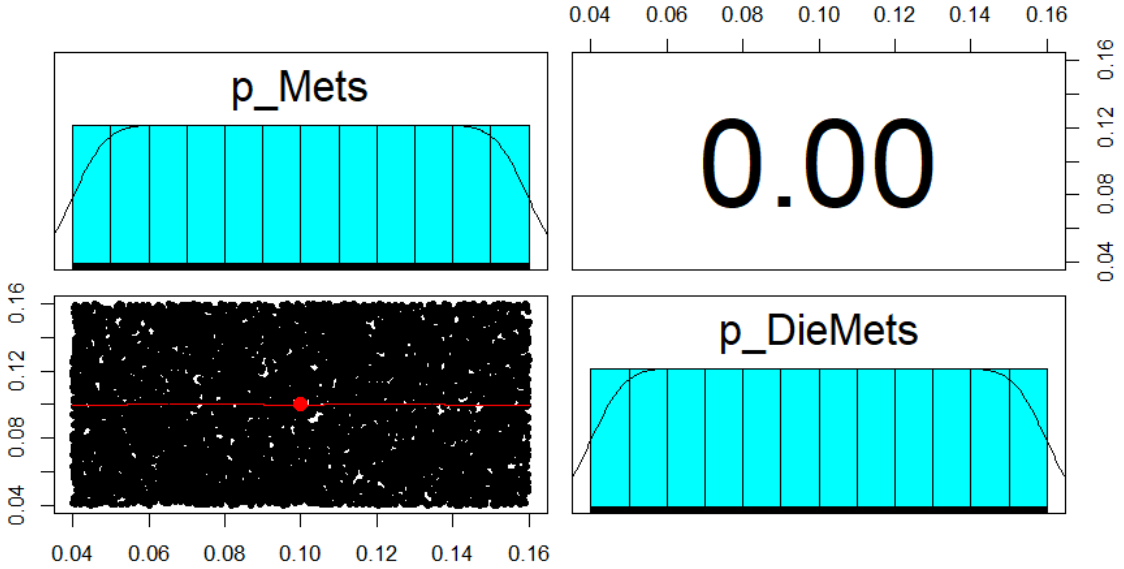

```r
#### CRS Markov model in a function ####
run_crs_markov <- function(v_params) {
  with(as.list(v_params), {
    ## Markov model parameters
    n_t  <- 60                        # time horizon, number of cycles
    v_n  <- c("NED", "Mets", "Death") # the 3 states of the model
    n_s <- length(v_n)                # number of health states

    # Transition probabilities
    # p_Mets    = 0.10          # probability to become sicker when sick
    # p_DieMets = 0.05            # hazard ratio of death in sick vs healthy

    ####### INITIALIZATION ########################################
    # create the cohort trace
    m_M <- matrix(NA, nrow = n_t + 1 ,
                  ncol = n_s,
                  dimnames = list(0:n_t, v_n)) # create Markov trace (n_t + 1 b

    m_M[1, ] <- c(1, 0, 0)                     # initialize Markov trace

    # create transition probability matrix for NO treatment
    m_P <- matrix(0,
                  nrow = n_s,
                  ncol = n_s,
                  dimnames = list(v_n, v_n))
    # fill in the transition probability array
    ### From NED
    m_P["NED", "NED"]   <- 1 - (p_Mets)
    m_P["NED", "Mets"]  <- p_Mets
    m_P["NED", "Death"] <- 0            # Not allowed to die from cancer in NED
    ### From Mets
    m_P["Mets", "NED"]   <- 0
    m_P["Mets", "Mets"]  <- 1 - (p_DieMets)
    m_P["Mets", "Death"] <- p_DieMets
    ### From Death
    m_P["Death", "Death"] <- 1

    # check rows add up to 1
    if (!isTRUE(all.equal(as.numeric(rowSums(m_P)), as.numeric(rep(1, n_s))))) {
      stop("This is not a valid transition Matrix")
    }

    ############# PROCESS ##########################################

    for (t in 1:n_t){                  # throughout the number of cycles
      m_M[t + 1, ] <- m_M[t, ] %*% m_P  # estimate the Markov trace for cycle t
    }

    ####### EPIDEMIOLOGICAL OUTPUT  ################################
    #### Overall Survival (OS) ####
    v_os <- 1 - m_M[, "Death"]  # calculate the overall survival (OS) probabili

    ####### RETURN OUTPUT  #########################################
    out <- list(Surv = v_os[-c(1:2)])

    return(out)
  }
  )
}
```

Set up calibration parameters

```r
#################################################################
######  Specify calibration parameters  ######
#################################################################
# Specify seed (for reproducible sequence of random numbers)
set.seed(072218)

# number of random samples
n_samp <- 10000

# names and number of input parameters to be calibrated
v_param_names <- c("p_Mets","p_DieMets")
n_param <- length(v_param_names)

# range on input search space
lb <- c(p_Mets = 0.04, p_DieMets = 0.04) # lower bound
ub <- c(p_Mets = 0.16, p_DieMets = 0.16) # upper bound

# number of calibration targets
v_target_names <- c("Surv")
n_target <- length(v_target_names)

```

Calibrate – Priors (1)

```
101 ##########################################################
102 ######  Calibrate!  ######
103 ##########################################################
104 # record start time of calibration
105 t_init <- Sys.time()
106
107 ###  Generate a random sample of input values  ###
108
109 # Sample unit Latin Hypercube
110 m_lhs_unit <- randomLHS(n_samp, n_param)
111
112 # Rescale to min/max of each parameter
113 m_param_samp <- matrix(nrow=n_samp,ncol=n_param)
114 for (i in 1:n_param){
115   m_param_samp[,i] <- qunif(m_lhs_unit[,i],
116                             min = lb[i],
117                             max = ub[i])
118 }
119 colnames(m_param_samp) <- v_param_names
120
121 # view resulting parameter set samples
122 pairs.panels(m_param_samp)
```



Calibrate – Run model w/ priors (2)

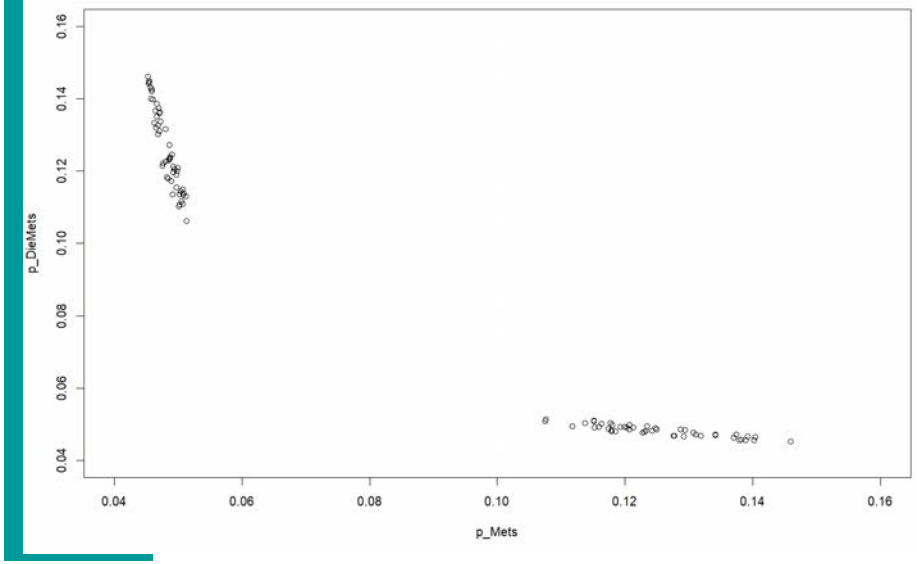
```r
###  Run the model for each set of input values ###

# initialize goodness-of-fit vector
m_GOF <- matrix(nrow = n_samp, ncol = n_target)
colnames(m_GOF) <- paste0(v_target_names, "_fit")

# loop through sampled sets of input values
for (j in 1:n_samp){

  ###  Run model for a given parameter set  ###
  model_res <- run_crs_markov(v_params = m_param_samp[j, ])


  ###  Calculate goodness-of-fit of model outputs to targets  ###

  # TARGET 1: Survival ("Surv")
  # log likelihood
  m_GOF[j,1] <- sum(dnorm(x = lst_targets$Surv$value,
                          mean = model_res$Surv,
                          sd = lst_targets$Surv$se,
                          log = T))

  # weighted sum of squared errors (alternative to log likelihood)
  # w <- 1/(lst_targets$Surv$se^2)
  # m_GOF[j,1] <- -sum(w*(lst_targets$Surv$value - v_res)^2)


  # TARGET 2: (if you had more...)
  # log likelihood
  # m_GOF[j,2] <- sum(dnorm(x = lst_targets$Target2$value,
  #                         mean = model_res$Target2,
  #                         sd = lst_targets$Target2$se,
  #                         log = T))


} # End loop over sampled parameter sets
```

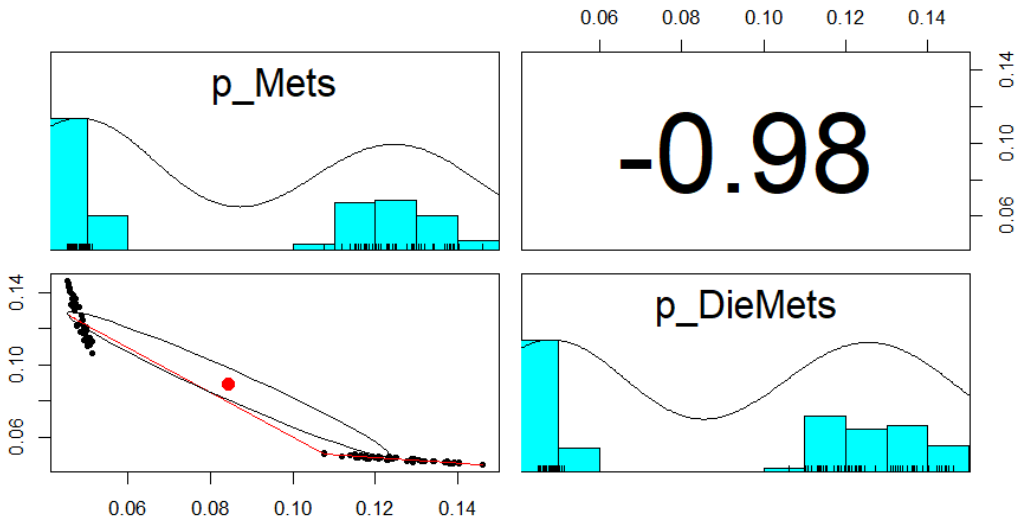
Calibrate – Compute GOF for each (3)

```
163  ###  Combine fits to the different targets into single GOF  ###
164  # can give different targets different weights
165  v_weights <- matrix(1, nrow = n_target, ncol = 1)
166  # matrix multiplication to calculate weight sum of each GOF matrix row
167  v_GOF_overall <- c(m_GOF%*%v_weights)
168  # Store in GOF matrix with column name "Overall"
169  m_GOF <- cbind(m_GOF,Overall_fit=v_GOF_overall)
170
171  # Calculate computation time
172  comp_time <- Sys.time() - t_init
173
```

Calibrate – Posteriors (4)

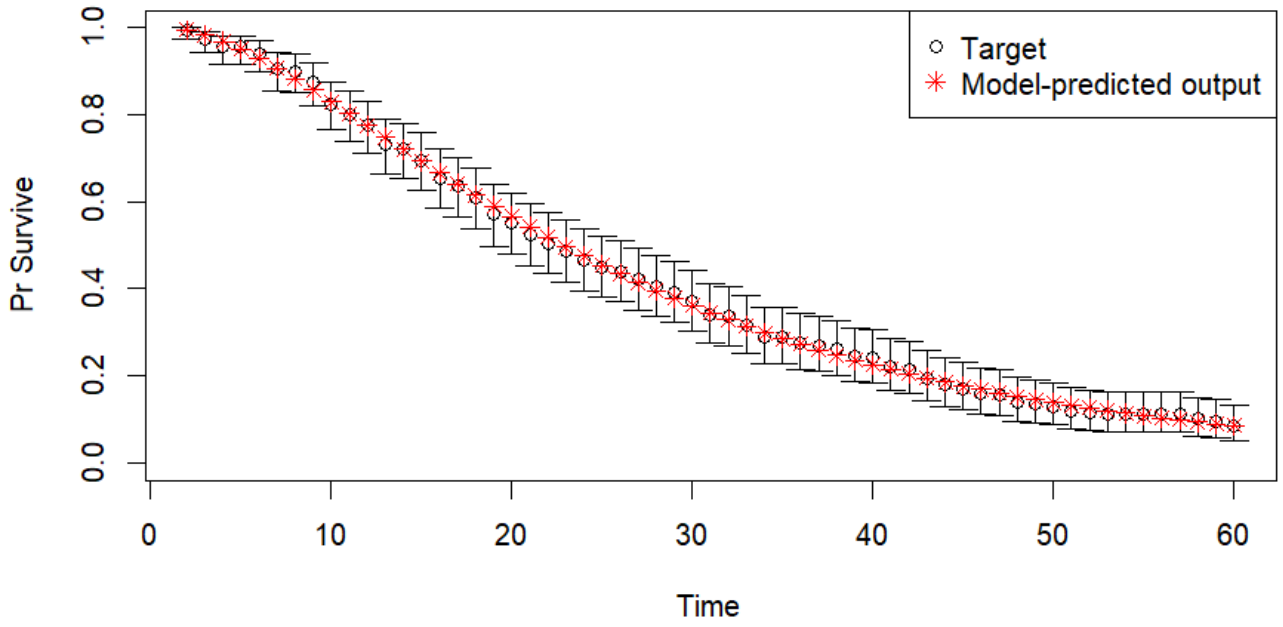
```
174 ##################################################################
175 ######  Exploring best-fitting input sets  ######
176 ##################################################################
177
178 # Arrange parameter sets in order of fit
179 m_calib_res <- cbind(m_param_samp,m_GOF)
180 m_calib_res <- m_calib_res[order(-m_calib_res[,"Overall_fit"]),]
181
182 # Examine the top 10 best-fitting sets
183 m_calib_res[1:10,]
184
185 # Plot the top 100 (top 10%)
186 plot(m_calib_res[1:100,1],m_calib_res[1:100,2],
187      xlim=c(lb[1],ub[1]),ylim=c(lb[2],ub[2]),
188      xlab = colnames(m_calib_res)[1],ylab = colnames(m_calib_res)[2])
189
190 # Pairwise comparison of top 100 sets
191 pairs.panels(m_calib_res[1:100,v_param_names])
```





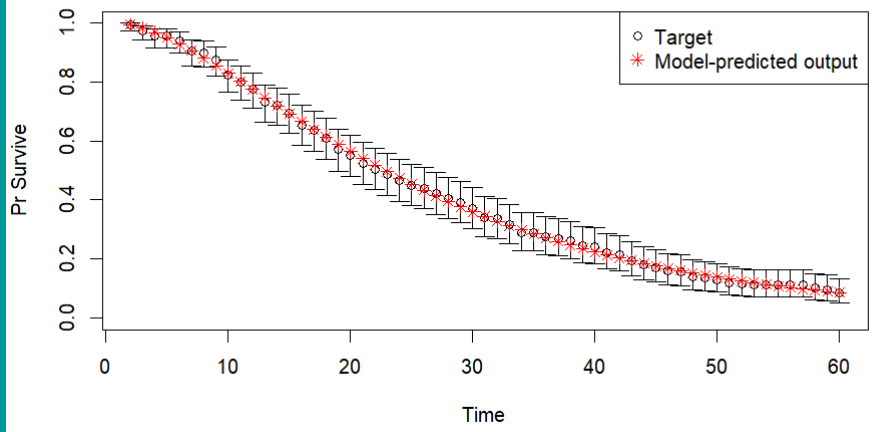
Calibrate – Compute GOF for each (5)

```
193  ### Plot model-predicted output at best set vs targets ###
194  v_out_best <- run_crs_markov(m_calib_res[1,])
195
196  # TARGET 1: Survival ("Surv")
197  plotrix::plotCI(x = lst_targets$Surv$time, y = lst_targets$Surv$value,
198                  ui = lst_targets$Surv$ub,
199                  li = lst_targets$Surv$lb,
200                  ylim = c(0, 1),
201                  xlab = "Time", ylab = "Pr Survive")
202  points(x = lst_targets$Surv$time,
203         y = v_out_best$Surv,
204         pch = 8, col = "red")
205  legend("topright",
206         legend = c("Target", "Model-predicted output"),
207         col = c("black", "red"), pch = c(1, 8))
```

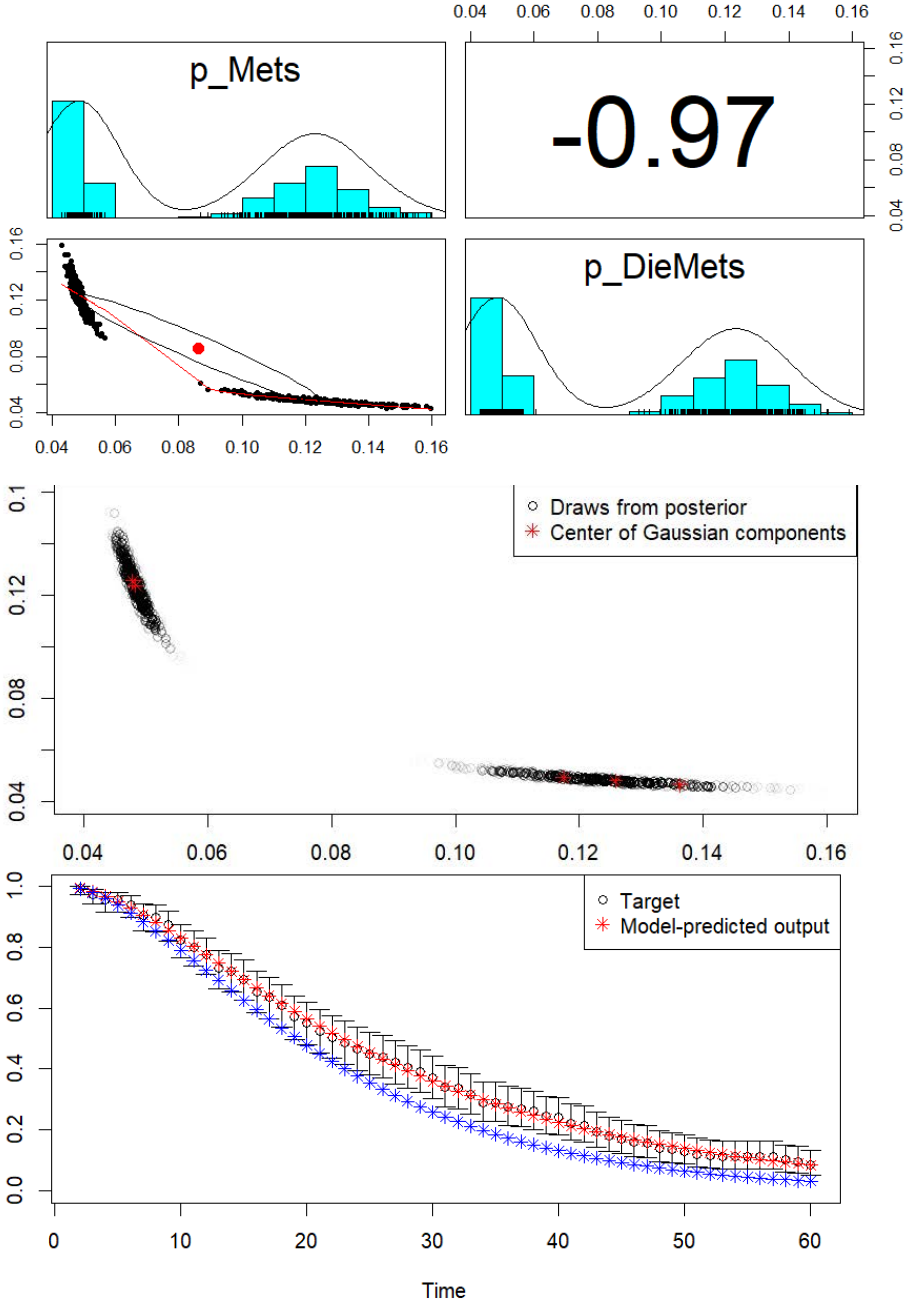


Advanced Calibration

CRS_Calib_NelderMead

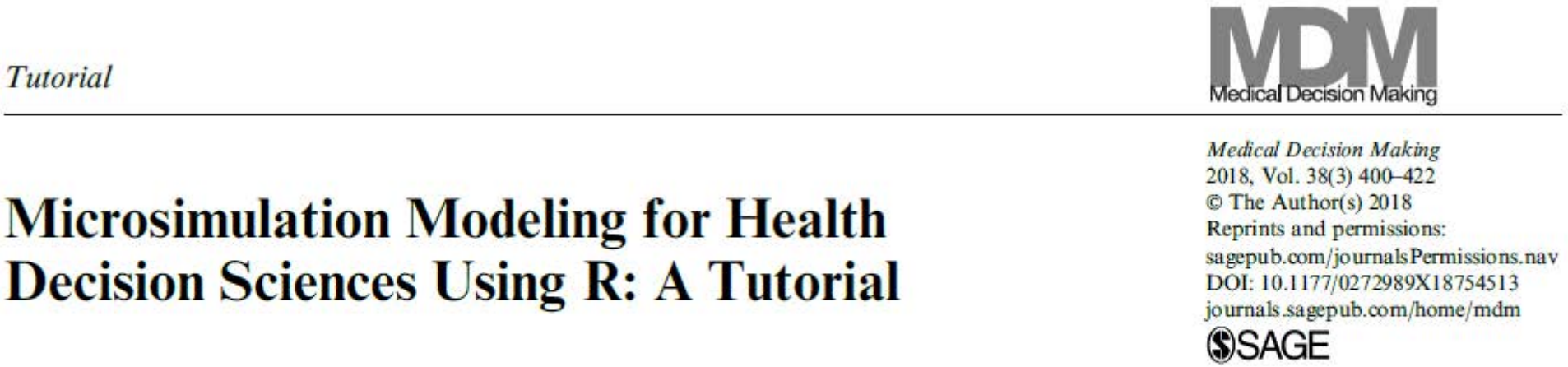


CRS_Calib_IMIS



R session
Microsimulation

Paper on Microsimulation in R



Tutorial

MDM Medical Decision Making

Medical Decision Making
2018, Vol. 38(3) 400–422
© The Author(s) 2018
Reprints and permissions:
sagepub.com/journalsPermissions.nav
DOI: 10.1177/0272989X18754513
journals.sagepub.com/home/mdm
SAGE

Microsimulation Modeling for Health Decision Sciences Using R: A Tutorial

Eline M. Krijkamp, Fernando Alarid-Escudero, Eva A. Enns, Hawre J. Jalal, M. G. Myriam Hunink, and Petros Pechlivanoglou

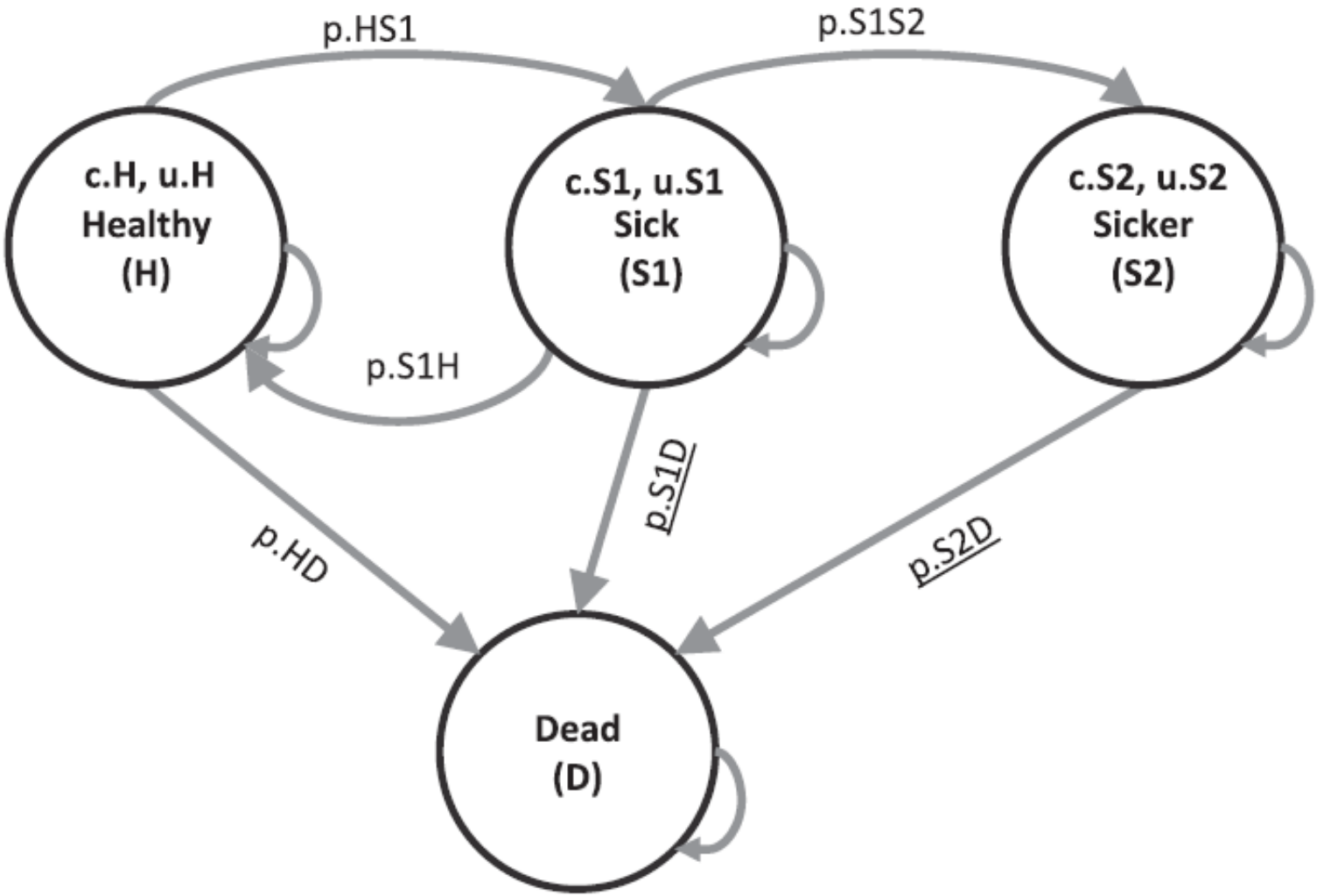
Two biggest differences from previous R code:

1. What happens inside model function: each individual of a specific # of people is simulated instead proportions of a cohort without a real defined size
2. Analysis must deal with Monte Carlo noise (first-order uncertainty) due to simulating a finite sample

Microsimulation Algorithm

1. The individual starts the simulation in an initial health state and is assigned a cost and health outcome value associated with staying in this initial state for one cycle, taking into consideration any individual-level characteristics.
2. During each cycle, the individual's probability to transition to a different health state (or remain in the current health state) in the next cycle is assigned based on the health states previously occupied and the individual's characteristics.
3. The health state to which the individual will transition to in the next cycle is sampled from a categorical distribution based on the probability of transitioning to each possible health state. The individual will then either transition to a new health state or remain in the same health state at the end of the cycle.
4. Each health state is associated with a particular cost or health outcome value attributed to remaining in the health state for one cycle. This could represent the costs associated with a health state and in a cost-utility context, the utility of remaining in a certain health state for one cycle. State-specific costs and health outcome values can depend on the individual's characteristics (age, gender, etc.) as well as on past transitions of the individual. Transition values, one-time costs and one-time changes in health outcomes associated with the transition, may apply.
5. By aggregating all the state and transition values over the model's cycles (and applying discounting if needed) we can estimate the (discounted) total model outcomes for that individual's lifetime.

Sick-Sicker Model



Simple Microsimulation – Inputs

```r
# Model input
n.i   <- 100000                # number of simulated individuals
n.t   <- 30                    # time horizon, 30 cycles
v.n   <- c("H","S1","S2","D")  # the model states: Healthy (H), Sick (S1), Sicker (S2), Dead (D)
n.s   <- length(v.n)           # the number of health states
v.M_1 <- rep("H", n.i)         # everyone begins in the healthy state
d.c   <- d.e <- 0.03           # equal discounting of costs and QALYs by 3%
v.Trt <- c("No Treatment", "Treatment") # store the strategy names

# Transition probabilities (per cycle)
p.HD    <- 0.005               # probability to die when healthy
p.HS1   <- 0.15                # probability to become sick when healthy
p.S1H   <- 0.5                 # probability to become healthy when sick
p.S1S2  <- 0.105               # probability to become sicker when sick
rr.S1   <- 3                   # rate ratio of death when sick vs healthy
rr.S2   <- 10                  # rate ratio of death when sicker vs healthy
r.HD    <- -log(1 - p.HD)      # rate of death when healthy
r.S1D   <- rr.S1 * r.HD        # rate of death when sick
r.S2D   <- rr.S2 * r.HD        # rate of death when sicker
p.S1D   <- 1 - exp(- r.S1D)    # probability to die when sick
p.S2D   <- 1 - exp(- r.S2D)    # probability to die when sicker

# Cost and utility inputs
c.H     <- 2000                # cost of remaining one cycle healthy
c.S1    <- 4000                # cost of remaining one cycle sick
c.S2    <- 15000               # cost of remaining one cycle sicker
c.Trt   <- 12000               # cost of treatment (per cycle)

u.H     <- 1                   # utility when healthy
u.S1    <- 0.75                # utility when sick
u.S2    <- 0.5                 # utility when sicker
u.Trt   <- 0.95                # utility when sick(er) and being treated
```

Simple Microsimulation – MicroSim function (1)

```r
MicroSim <- function(v.M_1, n.i, n.t, v.n, d.c, d.e,
                     TR.out = TRUE, TS.out = TRUE, Trt = FALSE, seed = 1) {
# Arguments:
  # v.M_1:   vector of initial states for individuals
  # n.i:     number of individuals
  # n.t:     total number of cycles to run the model
  # v.n:     vector of health state names
  # d.c:     discount rate for costs
  # d.e:     discount rate for health outcome (QALYs)
  # TR.out:  should the output include a microsimulation trace? (default is TRUE)
  # TS.out:  should the output include a matrix of transitions between states? (default is TRUE)
  # Trt:     are the n.i individuals receiving treatment? (scalar with a Boolean value, default is FALSE)
  # seed:    starting seed number for random number generator (default is 1)
# Makes use of:
  # Probs:   function for the estimation of transition probabilities
  # Costs:   function for the estimation of cost state values
  # Effs:    function for the estimation of state specific health outcomes (QALYs)

```

Simple Microsimulation – MicroSim function (2)

```r
  v.dwc <- 1 / (1 + d.c) ^ (0:n.t)    # calculate the cost discount weight based on the discount rate d.c
  v.dwe <- 1 / (1 + d.e) ^ (0:n.t)    # calculate the QALY discount weight based on the discount rate d.e

# create the matrix capturing the state name/costs/health outcomes for all individuals at each time point
  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] <- v.M_1                    # indicate the initial health state

  for (i in 1:n.i) {
    set.seed(seed + i)                 # set the seed for every individual for the random number generator
    m.C[i, 1] <- Costs(m.M[i, 1], Trt)  # estimate costs per individual for the initial health state conditional on treatment
    m.E[i, 1] <- Effs (m.M[i, 1], Trt)  # estimate QALYs per individual for the initial health state conditional on treatment

    for (t in 1:n.t) {
      v.p <- Probs(m.M[i, t])          # calculate the transition probabilities at cycle t

      m.M[i, t + 1] <- sample(v.n, prob = v.p, size = 1)  # sample the next health state and store that state in matrix m.M
      m.C[i, t + 1] <- Costs(m.M[i, t + 1], Trt)   # estimate costs per individual during cycle t + 1 conditional on treatment
      m.E[i, t + 1] <- Effs( m.M[i, t + 1], Trt)   # estimate QALYs per individual during cycle t + 1 conditional on treatment

    } # close the loop for the time points
    if(i/100 == round(i/100,0)) {          # display the progress of the simulation
      cat('\r', paste(i/n.i * 100, "% done", sep = " "))
    }
  } # close the loop for the individuals

  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) QALYs
```

Simple Microsimulation – MicroSim function (3)

```
115   if (TS.out == TRUE) {  # create a  matrix of transitions across states
116     TS <- paste(m.M, cbind(m.M[, -1], NA), sep = "->") # transitions from one state to the other
117     TS <- matrix(TS, nrow = n.i)
118     rownames(TS) <- paste("Ind",   1:n.i, sep = " ")   # name the rows
119     colnames(TS) <- paste("Cycle", 0:n.t, sep = " ")   # name the columns
120   } else {
121     TS <- NULL
122   }
123
124   if (TR.out == TRUE) { # create a trace from the individual trajectories
125     TR <- t(apply(m.M, 2, function(x) table(factor(x, levels = v.n, ordered = TRUE))))
126     TR <- TR / n.i                                      # create a distribution trace
127     rownames(TR) <- paste("Cycle", 0:n.t, sep = " ")     # name the rows
128     colnames(TR) <- v.n                                 # name the columns
129   } else {
130     TR <- NULL
131   }
132
133   results <- list(m.M = m.M, m.C = m.C, m.E = m.E, tc = tc, te = te,
134                   tc_hat = tc_hat, te_hat = te_hat,
135                   TS = TS, TR = TR) # store the results from the simulation in a list
136   return(results)  # return the results
137 }  # end of the MicroSim function
138
```

Simple Microsimulation – Probs function

```r
#### Probability function
# The Probs function that updates the transition probabilities of every cycle is shown below.

Probs <- function(M_it) {
  # M_it:    health state occupied by individual i at cycle t (character variable)

  v.p.it <- rep(NA, n.s)      # create vector of state transition probabilities
  names(v.p.it) <- v.n        # name the vector

  # update v.p.it with the appropriate probabilities
  v.p.it[M_it == "H"]  <- c(1 - p.HS1 - p.HD, p.HS1, 0, p.HD)                  # transition probabilities when healthy
  v.p.it[M_it == "S1"] <- c(p.S1H, 1- p.S1H - p.S1S2 - p.S1D, p.S1S2, p.S1D)   # transition probabilities when sick
  v.p.it[M_it == "S2"] <- c(0, 0, 1 - p.S2D, p.S2D)                            # transition probabilities when sicker
  v.p.it[M_it == "D"]  <- c(0, 0, 0, 1)                                        # transition probabilities when dead
  ifelse(sum(v.p.it) == 1, return(v.p.it), print("Probabilities do not sum to 1")) # return the transition probabilities
}
```

Simple Microsimulation – Costs function

```r
Costs <- function (M_it, Trt = FALSE) {
  # M_it: health state occupied by individual i at cycle t (character variable)
  # Trt:  is the individual being treated? (default is FALSE)

  c.it <- 0                                    # by default the cost for everyone is zero
  c.it[M_it == "H"]  <- c.H                    # update the cost if healthy
  c.it[M_it == "S1"] <- c.S1 + c.Trt * Trt     # update the cost if sick conditional on treatment
  c.it[M_it == "S2"] <- c.S2 + c.Trt * Trt     # update the cost if sicker conditional on treatment
  return(c.it)                                 # return the costs
}
```

Simple Microsimulation – Effs function

```r
### Health outcome function
# The Effs function to update the utilities at every cycle.

Effs <- function (M_it, Trt = FALSE, cl = 1) {
  # M_it: health state occupied by individual i at cycle t (character variable)
  # Trt:  is the individual treated? (default is FALSE)
  # cl:   cycle length (default is 1)

  u.it <- 0                       # by default the utility for everyone is zero
  u.it[M_it == "H"]  <- u.H       # update the utility if healthy
  u.it[M_it == "S1"] <- Trt * u.Trt + (1 - Trt) * u.S1  # update the utility if sick conditional on treatment
  u.it[M_it == "S2"] <- u.S2      # update the utility if sicker
  QALYs <-  u.it * cl             # calculate the QALYs during cycle t
  return(QALYs)                   # return the QALYs
}
```

Simple Microsimulation – Running the model and computing results

```r
#################################### Run the simulation ###################################
sim_no_trt  <- MicroSim(v.M_1, n.i, n.t, v.n, d.c, d.e, Trt = FALSE) # run for no treatment
sim_trt     <- MicroSim(v.M_1, n.i, n.t, v.n, d.c, d.e, Trt = TRUE)  # run for treatment
################################## Cost-effectiveness analysis ############################

# store the mean costs (and the MCSE) of each strategy in a new variable v.C (vector costs)
v.C  <- c(sim_no_trt$tc_hat, sim_trt$tc_hat)
se.C <- c(sd(sim_no_trt$tc), sd(sim_trt$tc)) / sqrt(n.i)
# store the mean QALYs (and the MCSE) of each strategy in a new variable v.E (vector health outcomes)
v.E  <- c(sim_no_trt$te_hat, sim_trt$te_hat)
se.E <- c(sd(sim_no_trt$te), sd(sim_trt$te)) / sqrt(n.i)

delta.C <- v.C[2] - v.C[1]                    # calculate incremental costs
delta.E <- v.E[2] - v.E[1]                    # calculate incremental QALYs
se.delta.E <- sd(sim_trt$te - sim_no_trt$te) / sqrt(n.i) # Monte Carlo squared error (MCSE) of incremental costs
se.delta.C <- sd(sim_trt$tc - sim_no_trt$tc) / sqrt(n.i) # Monte Carlo squared error (MCSE) of incremental QALYs
ICER    <- delta.C / delta.E                  # calculate the ICER
results <- c(delta.C, delta.E, ICER)          # store the values in a new variable
```

Simple Microsimulation – Running the model and computing results

```r
# Create full incremental cost-effectiveness analysis table
table_micro <- data.frame(
  c(round(v.C, 0),  ""),          # costs per arm
  c(round(se.C, 0), ""),          # MCSE for costs
  c(round(v.E, 3),  ""),          # health outcomes per arm
  c(round(se.E, 3), ""),          # MCSE for health outcomes
  c("", round(delta.C, 0),   ""), # incremental costs
  c("", round(se.delta.C, 0),""), # MCSE for incremental costs
  c("", round(delta.E, 3),   ""), # incremental QALYs
  c("", round(se.delta.E, 3),""), # MCSE for health outcomes (QALYs) gained
  c("", round(ICER, 0),      "")  # ICER
)
rownames(table_micro) <- c(v.Trt, "* are MCSE values")  # name the rows
colnames(table_micro) <- c("Costs", "*",  "QALYs", "*", "Incremental Costs", "*", "QALYs Gained", "*", "ICER") # name the c
table_micro  # print the table
```

```
                 Costs   *  QALYs     * Incremental Costs   * QALYs Gained     *   ICER
No Treatment      75996 183 15.823 0.016
Treatment        141644 343 16.384 0.016            65648 164        0.561 0.001 117087
* are MCSE values
```

Sick-Sicker Model

Table 1 Input Parameters for the Illustrative Microsimulation Model

Parameter	R name	Value
Time horizon (*nt*)	`n.t`	30 y
Cycle length (*cl*)	`cl`	1 y
Number of simulated individuals (*ni*)	`n.i`	100,000
Names of health states (*n*)	`v.n`	H, S1, S2, D
Annual discount rate (costs/QALYs) (*dc*/*de*)	`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)	`p.S1S2`	0.105
Annual risks of death		
- H to D	`p.HD`	0.005
- Rate ratio of death in S1 v. healthy	`rr.S1`	3
- Rate ratio of death in S2 v. healthy	`rr.S2`	10
Annual costs		
- Healthy individuals	`c.H`	$2,000
- Sick individuals in S1	`c.S1`	$4,000
- Sick individuals in S2	`c.S2`	$15,000
- Annual treatment cost per sick individual (S1 and 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
Intervention effect		
- Utility for treated individuals in S1 (SD)	`u.Trt`	0.95
Time varying extension of Sick-Sicker model (Figure 1)		
- Treatment effect modifier at baseline	`v.x`	*Uniform* (0.95, 1.05)
- Utility decrement of treated sick individuals with every additional year of being sick (S1 and S2)	`ru.S1S2`	0.03
- Proportional increase of the mortality rate with every additional year of being sick/sicker (S1 and S2)	`rp.S1S2`	0.2

Extended Microsimulation

Same inputs as with simple Microsimulation but now with individual heterogeneity of effect and dwell-time effects on mortality and utility of being sick/sicker

```
47  rp.S1S2 <- 0.2                  # increase of the mortality rate with every additional year being sick

59  ru.S1S2 <- 0.03                   # decrease in utility of treated sick individuals with every
60                                    # additional year being sick/sicker
61  v.x     <- runif(n.i, 0.95, 1.05) # vector capturing individuals' effect modifier at baseline
```

Extended Microsimulation – MicroSim function (1)

Same MicroSim function as before except now

```r
# The MicroSim functions for the extended microsimulation of the 'Sick-Sicker' model
# keeps track of what happens to each individual during each cycle.
MicroSim <- function(v.M_1, n.i, n.t, v.n, X = NULL, d.c, d.e,
                     TR.out = TRUE, TS.out = TRUE, Trt = FALSE, seed = 1) {
```

```r
  # create the dur variable that stores the number of consecutive cycles the individual occupies either when sick or sicker
  dur <- 0                              # the individual start without history
  m.C[i, 1] <- Costs(m.M[i, 1], Trt)  # estimate costs per individual for the initial health state conditional on treatment
  m.E[i, 1] <- Effs(m.M[i, 1], dur, Trt, X = X[i])  # estimate QALYs per individual for the initial health state conditional

  for (t in 1:n.t) {
    v.p <- Probs(m.M[i, t], dur)         # calculate the transition probabilities at cycle t conditional on the duration of

    m.M[i, t + 1] <- sample(v.n, prob = v.p, size = 1)  # sample the new health state and store that state in matrix m.M
    m.C[i, t + 1] <- Costs(m.M[i, t + 1], Trt)     # estimate the cost per individual during cycle t + 1 conditional on trea
    m.E[i, t + 1] <-  Effs(m.M[i, t + 1], dur, Trt, X = X[i])    # estimate the utility per individual during cycle t + 1 co

    if (m.M[i, t + 1] == "S1" | m.M[i, t + 1] == "S2") {  # expression to identify sick/sicker individuals
      dur <- dur + 1   # update the duration of being sick/sicker
    } else {
      dur <- 0}        # reset duration variable

  } # close the loop for the time points
  if(i/100 == round(i/100,0)) {           # display the progress of the simulation
    cat('\r', paste(i/n.i * 100, "% done", sep = ""))
  }
} # close the loop for the individuals
```

Extended Microsimulation – MicroSim function (2)

Same MicroSim function now individual characertistics X are used as is duration in Sick/Sicker

```
65  # The MicroSim functions for the extended microsimulation of the 'Sick-Sicker' model
66  # keeps track of what happens to each individual during each cycle.
67  MicroSim <- function(v.M_1, n.i, n.t, v.n, X = NULL, d.c, d.e,
68                       TR.out = TRUE, TS.out = TRUE, Trt = FALSE, seed = 1) {
```

```
 99    # create the dur variable that stores the number of consecutive cycles the individual occupies either when sick or sicker
100    dur <- 0                              # the individual start without history
101    m.C[i, 1] <- Costs(m.M[i, 1], Trt)  # estimate costs per individual for the initial health state conditional on treatment
102    m.E[i, 1] <- Effs(m.M[i, 1], dur, Trt, X = X[i])  # estimate QALYs per individual for the initial health state conditional
103
104    for (t in 1:n.t) {
105      v.p <- Probs(m.M[i, t], dur)          # calculate the transition probabilities at cycle t conditional on the duration of
106
107      m.M[i, t + 1] <- sample(v.n, prob = v.p, size = 1)  # sample the new health state and store that state in matrix m.M
108      m.C[i, t + 1] <- Costs(m.M[i, t + 1], Trt)     # estimate the cost per individual during cycle t + 1 conditional on trea
109      m.E[i, t + 1] <-  Effs(m.M[i, t + 1], dur, Trt, X = X[i])    # estimate the utility per individual during cycle t + 1 co
110
111      if (m.M[i, t + 1] == "S1" | m.M[i, t + 1] == "S2") {  # expression to identify sick/sicker individuals
112        dur <- dur + 1   # update the duration of being sick/sicker
113      } else {
114        dur <- 0}        # reset duration variable
115
116    } # close the loop for the time points
117    if(i/100 == round(i/100,0)) {            # display the progress of the simulation
118      cat('\r', paste(i/n.i * 100, "% done", sep = ""))
119    }
120  } # close the loop for the individuals
```

Extended Microsimulation – Probs function

```r
#### Probability function
# The Probs function that updates the transition probabilities of every cycle is shown below.

Probs <- function(M_it, dur) {
  # M_it:   health state occupied by individual i at cycle t (character variable)
  # dur:    the duration of being sick (sick/sicker)

  v.p.it <- rep(NA, n.s)      # create vector of state transition probabilities
  names(v.p.it) <- v.n        # name the vector

  # update probabilities of death after first converting them to rates and applying the rate ratio
  r.S1D <-  - log(1 - p.S1D)
  r.S2D <-  - log(1 - p.S2D)
  p.S1D <- 1 - exp(- r.S1D * (1 + dur * rp.S1S2)) # calculate p.S1D conditional on duration of being sick/sicker
  p.S2D <- 1 - exp(- r.S2D * (1 + dur * rp.S1S2)) # calculate p.S2D conditional on duration of being sick/sicker

  # update v.p.it with the appropriate probabilities
  v.p.it[M_it ==  "H"] <- c(1 - p.HS1 - p.HD, p.HS1, 0, p.HD)                    # transition probabilities when healthy
  v.p.it[M_it == "S1"] <- c(p.S1H, 1- p.S1H - p.S1S2 - p.S1D, p.S1S2, p.S1D)     # transition probabilities when sick
  v.p.it[M_it == "S2"] <- c(0, 0, 1 - p.S2D, p.S2D)                              # transition probabilities when sicker
  v.p.it[M_it ==  "D"] <- c(0, 0, 0, 1)                                          # transition probabilities when dead
  ifelse(sum(v.p.it) == 1, return(v.p.it), print("Probabilities do not sum to 1")) # return the transition probabilities or
}
```

Extended Microsimulation – Effs function

```r
### Health outcome function
# The Effs function to update the utilities at every cycle for the extended microsimulaton.

Effs <- function (M_it, dur, Trt = FALSE, cl = 1, X = NULL) {
  # M_it: health state occupied by individual i at cycle t (character variable)
  # dur:  the duration of being sick/sicker
  # Trt:  is the individual being treated? (default is FALSE)
  # cl:   the cycle length (default = 1 )
  # X:    the vector or matrix of individual characteristics (optional)

  u.it               <- 0        # by default the utility for everyone is zero
  u.it[M_it == "H"]  <- u.H      # update the utility if healthy
  u.it[M_it == "S1"] <- X * Trt * (u.Trt - dur * ru.S1S2) + (1 - Trt) * u.S1 # update the utility if sick conditional on
  u.it[M_it == "S2"] <- u.S2     # update the utility if sicker
  QALYs <- u.it * cl             # calculate the QALYs during cycle t
  return(QALYs)                  # return the results
}
```

	Costs	*	QALYs	*	Incremental Costs	*	QALYs Gained	*	ICER
No Treatment	62667	120	15.279	0.017					
Treatment	117455	231	15.785	0.017	54787	117	0.507	0.001	108090

* are MCSE values

Extended Microsimulation – Running the model and computing results

```
# Create full incremental cost-effectiveness analysis table
table_micro <- data.frame(
  c(round(v.C, 0),  ""),          # costs per arm
  c(round(se.C, 0), ""),          # MCSE for costs
  c(round(v.E, 3),  ""),          # health outcomes per arm
  c(round(se.E, 3), ""),          # MCSE for health outcomes
  c("", round(delta.C, 0),   ""), # incremental costs
  c("", round(se.delta.C, 0),""), # MCSE for incremental costs
  c("", round(delta.E, 3),   ""), # incremental QALYs
  c("", round(se.delta.E, 3),""), # MCSE for health outcomes (QALYs) gained
  c("", round(ICER, 0),      "")  # ICER
)
rownames(table_micro) <- c(v.Trt, "* are MCSE values")  # name the rows
colnames(table_micro) <- c("Costs", "*",  "QALYs", "*", "Incremental Costs", "*", "QALYs Gained", "*", "ICER") # name the c
table_micro  # print the table
```

```
                  Costs   *  QALYs     * Incremental Costs   * QALYs Gained     *   ICER
No Treatment       62667 120 15.279 0.017
Treatment         117455 231 15.785 0.017             54787 117        0.507 0.001 108090
* are MCSE values
```

```
                  Costs   *  QALYs     * Incremental Costs   * QALYs Gained     *   ICER
No Treatment       75996 183 15.823 0.016
Treatment         141644 343 16.384 0.016             65648 164        0.561 0.001 117087
* are MCSE values
```

Wrap-up

DARTH Publications

An Overview of R in Health Decision Sciences

Hawre Jalal, MD, PhD, Petros Pechlivanoglou, MSc, PhD, Eline Krijkamp, MSc, Fernando Alarid-Escudero, MSc, Eva Enns, MS, PhD, M. G. Myriam Hunink, MD, PhD

Microsimulation Modeling for Health Decision Sciences Using R: A Tutorial

Eline M. Krijkamp, Fernando Alarid-Escudero, Eva A. Enns, Hawre J. Jalal, M. G. Myriam Hunink, and Petros Pechlivanoglou

MDM Medical Decision Making

Medical Decision Making
2018, Vol. 38(3) 400–422

DOI: 10.1177/0272989X18754513
journals.sagepub.com/home/mdm
SAGE

Brief Report

A Multidimensional Array Representation of State-Transition Model Dynamics

Eline M. Krijkamp*, Fernando Alarid-Escudero*, Eva A. Enns, Petros Pechlivanoglou, M.G. Myriam Hunink, Alan Yang, and Hawre J. Jalal

MDM Medical Decision Making

Medical Decision Making
1–7

DOI: 10.1177/0272989X198
journals.sagepub.com/home
SAGE

PharmacoEconomics (2019) 37:1329–1339
https://doi.org/10.1007/s40273-019-00837-x

PRACTICAL APPLICATION

A Need for Change! A Coding Framework for Improving Transparency in Decision Modeling

Fernando Alarid-Escudero[1] · Eline M. Krijkamp[2] · Petros Pechlivanoglou[3] · Hawre Jalal[4] · Szu-Yu Zoe Kao[5] · Alan Yang[6] · Eva A. Enns[5]

Tutorial

An Introductory Tutorial on Cohort State-Transition Models in R Using a Cost-Effectiveness Analysis Example

Fernando Alarid-Escudero, Eline Krijkamp, Eva A. Enns, Alan Yang, M. G. Myriam Hunink, Petros Pechlivanoglou, and Hawre Jalal

Tutorial

A Tutorial on Time-Dependent Cohort State-Transition Models in R Using a Cost-Effectiveness Analysis Example

Fernando Alarid-Escudero*, Eline Krijkamp, Eva A. Enns, Alan Yang, M. G. Myriam Hunink, Petros Pechlivanoglou, and Hawre Jalal

MDM Medical Decision Making

Medical Decision Making
1–21

DOI: 10.1177/0272989X221121747
journals.sagepub.com/home/mdm
SAGE

Acknowledgements

Mt Hood Asia/ SMDM

jeremygf@stanford.edu

falarid@stanford.edu

http://darthworkgroup.com/

https://github.com/organizations/DARTH-git

https://www.linkedin.com/groups/8635339

@DARTHworkgroup