

Microsimulation Modeling for Health Decision Sciences Using R: A Tutorial

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



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

Abstract

Microsimulation models are becoming increasingly common in the field of decision modeling for health. Because microsimulation models are computationally more demanding than traditional Markov cohort models, the use of computer programming languages in their development has become more common. R is a programming language that has gained recognition within the field of decision modeling. It has the capacity to perform microsimulation models more efficiently than software commonly used for decision modeling, incorporate statistical analyses within decision models, and produce more transparent models and reproducible results. However, no clear guidance for the implementation of microsimulation models in R exists. In this tutorial, we provide a step-by-step guide to build microsimulation models in R and illustrate the use of this guide on a simple, but transferable, hypothetical decision problem. We guide the reader through the necessary steps and provide generic R code that is flexible and can be adapted for other models. We also show how this code can be extended to address more complex model structures and provide an efficient microsimulation approach that relies on vectorization solutions.

Keywords

decision-analytic modeling, Markov model, microsimulation, open source software, R project

Date received: September 26, 2016; accepted: December 5, 2017

Healthcare policy makers are often faced with decisions on how to allocate healthcare resources given constrained budgets and the continuous development of new, expensive technologies. Policy makers increasingly rely on health decision modeling tools to guide their decisions, as such models can synthesize evidence from different sources to give indications on the long-term implications and the uncertainty around a decision.¹

One of the most common types of decision models used is that of state-transition cohort models.^{2,3} Cohort models investigate a hypothetical homogeneous cohort of individuals as they transition across health states. In a deterministic cohort model, the result is precisely determined given a set of initial conditions and parameters. As the complexity of decisions increases, deterministic cohort models become inadequate in reflecting the decision problem, and more complex models are needed. For example, an assumption commonly made in cohort

Erasmus MC, Epidemiology Department, Rotterdam, The Netherlands (EMK, MGMH); University of Minnesota School of Public Health, Minneapolis, MN, USA (FAE, EAE); University of Pittsburgh Graduate School of Public Health, Pittsburgh, PA, USA (HJJ); Erasmus MC, Radiology Department, Rotterdam, The Netherlands (MGMH) Harvard T.H. Chan School of Public Health, Center for Health Decision Science, Boston, USA (MGMH); Child Health Evaluative Sciences, The Hospital for Sick Children, Toronto, ON, Canada (PP); Institute of Health Policy Management and Evaluation, University of Toronto, ON, Canada (PP). The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article. Financial support for this study was provided by the Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota (FAE); and a grant from National Institute of Allergy and Infectious Diseases of the National Institutes of Health (award no. K25AI118476) (EAE). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. Royalties are received for the textbook ‘Decision Making in Health and Medicine: Integrating Evidence and Values’ (MGMH). There are no other conflicts of interest to disclose.

Corresponding Author

Petros Pechlivanoglou, The Peter Gilgan Centre for Research and Learning, The Hospital for Sick Children, Toronto, and University of Toronto, 686 Bay St, Toronto, ON M5A 0G4, Canada.
(petros.pechlivanoglou@sickkids.ca)

models is that the transition probabilities only depend on the current health state at any given cycle and cannot depend on the history before that cycle. This is often referred to as the “Markov” assumption and is an inherent limitation of cohort models.⁴ Although this assumption can be relaxed by creating additional states that can capture the cohort’s history, this can result in difficult to manage models due to “state explosion”.⁴

Individual-based state-transition (or microsimulation) models address many of the limitations of deterministic cohort models because they can more accurately reflect individual clinical pathways, incorporate the impact of history on future events, and more easily capture the variation in patients’ characteristics at baseline.^{5,6} Microsimulation models differ from the more traditional cohort-based models because they simulate the impact of interventions or policies on individual trajectories rather than the deterministic mean response of homogeneous cohorts.^{3,7–11} In a microsimulation model, outcomes are generated for each individual and are used to estimate the distribution of an outcome for a sample of potentially heterogeneous individuals. This individual-level simulation allows the inclusion of stochastic variation in disease progression as well as variation due to individual characteristics. Microsimulation models do not require the Markov assumption and so can introduce “memory” to the models’ structure. This “memory” characteristic makes microsimulation models more popular in diseases where the severity of the disease or the cost and health outcomes vary with duration spent in a diseased state.¹²

There have been numerous microsimulation models developed for health applications. For example, the Population Health Model (POHEM) simulates the life-cycle of the Canadian population and assesses the impact of policy and program interventions on the health status of Canadians;^{13–15} the Future Elderly Model (FEM) predicts the impact of changes in health care status on future health and costs;¹⁶ and the Prostate Cancer and Policy (PCOP) model evaluates the benefits and harms of prostate cancer screening.¹⁷

A drawback of the additional functionality of microsimulation models is numerical and computational complexity. To accommodate the computational demands of microsimulation models, health decision scientists increasingly adopt (high-level) programming languages. The increasing availability of patient-level data also requires more advanced statistical analyses to be integrated with microsimulation models. In addition, computational efficiency can be achieved in decision modeling when relying on numerical and statistical computing software, such as Matlab or R, versus either spreadsheet software (e.g., Microsoft Excel) or common specialized software, such as TreeAge.¹⁸ Recently, we illustrated the

increased use of the statistical software R in health decision sciences, and provided a collection of resources for its application in medical decision making.¹⁹

R is an open-source and freely available software environment, where statistical analyses can be combined with decision models within the same framework, and the results can be presented in publication-ready tabular and graphical forms.²⁰ The popularity of R in decision analysis follows similar trends as in other fields of science, which have established R as the second-most often used statistical software²¹ and among the most often used programming languages.²² Implementing a microsimulation model using a programming language provides the ability to use version control capabilities and repositories, such as GitHub. This is particularly useful when collaborating in teams, where multiple members can contribute to different components of the model. Additionally, with R, it is possible to use tools to generate web-based apps and graphical user interfaces (e.g., Shiny).¹⁹

R is not a programming language that was developed with a focus on computational performance. Nevertheless, there are some characteristics of the R programming language that can be leveraged to allow R-coded programs to perform efficiently. A key attribute of R is its ability to process problems more efficiently when presented in a vector rather than in a scalar format. Vectorization can reduce the need to rely on iterative “for” loops, which, in general, increase computation time significantly. Vectorization is particularly important in decision analysis, because processes are frequently repetitive and relatively straightforward to vectorize.

Despite the growing popularity of statistical programming for building simulation models in health decision sciences, few educational tutorials on how to perform these simulations are available.^{12,23,24} This tutorial offers both a theoretical and applied introduction on building microsimulation models in R, and has been created for beginner/intermediate users of R who are interested in developing an R-based microsimulation model.

This tutorial is structured as follows: First, we introduce a conceptual algorithm for model implementation, which can be applied to any (high-level) programming language, such as R. Subsequently, we illustrate how to build a microsimulation model in R using a simple, hypothetical decision problem. We guide the reader through the necessary steps and provide the R code used in the illustration. We additionally present a microsimulation solution that uses a vectorization approach. We then conclude with a discussion of the advantages and disadvantages of using R for microsimulation modeling, and provide guidance on how to extend the presented methods to more complex applications. In Supplementary Appendices and on GitHub (<https://>

github.com/DARTH-git/Microsimulation-tutorial), we provide the full R code used to run the microsimulation model and the vectorized approach.

Methods

Microsimulation models are structured around a set of mutually exclusive and collectively exhaustive health states.^{9,10,25,26} Each hypothetical individual can only be in one health state at any given cycle. Based on individual-specific transition probabilities, individuals are simulated through the model one at a time, while keeping track of their individual trajectories. In an economic evaluation framework, each health state, and potentially each transition between health states, is associated with a particular cost and health outcome value. For example, in the context of a cost-utility analysis, the cost and health outcome values would, respectively, represent the cost and the quality-adjusted life years (QALYs) associated with remaining in each health state for one cycle. Both state and transition outcome values can depend on the individual's characteristics as well as on the past transitions and events of the individual.

State-transition Microsimulation Algorithm

The flexibility of a programming language like R implies that a microsimulation model can be implemented in multiple ways depending on the experience and programming style of the modeler and the research question that needs to be addressed. Below, we outline an algorithm that could be used when implementing a microsimulation model. This algorithm serves our goal to provide a generic approach with a reasonable balance between clarity and efficiency. First, we outline the steps to simulate one hypothetical individual over time.

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.

Repeating these steps for all individuals in the simulation, allows us to describe the distribution of total cost and health outcomes for the population of interest.

To formalize the previous steps, we define the following notation

- nt as the number of discrete time cycles
- t as the current cycle, ranging from 0 to nt
- cl as the cycle length
- ni as the number of individuals
- i as a specific individual from the group of individuals
- nx as the number of individual characteristics
- ns as the number of health states
- n as the ns character vector of health state names
- p as the ns vector containing the transition probabilities for individual i at the end of cycle t
- M as the $ni \times (nt + 1)$ matrix capturing the health states occupied by all individuals during all cycles
- X as the $ni \times (nt + 1) \times nx$ 3-dimensional array of individuals' characteristics for all cycles (e.g. age, gender, disease severity, comorbidities)
- $Probs(M_i, X_i), Costs(M_i, X_i), Effs(M_i, X_i, cl)$ as functions assigning individual-specific probabilities, cost and health outcome values conditional on M_i, X_i and cl . M_i and X_i represent the vectors of health states and individual characteristics, respectively, for all time points for individual i
- C as the $ni \times (nt + 1)$ matrix capturing the costs for all individuals during all cycles
- E as the $ni \times (nt + 1)$ matrix capturing the health outcomes for all individuals during all cycles
- dwc as the nt vector of discount weights for costs, where $dwc = \frac{1}{(1 + dc)^t}$, with dc being the discount rate for the costs at any cycle t . The first cycle is not discounted, as the first time point t is equal to zero.

- dwe as the nt vector of discount weights for health outcomes, where $dwe = \frac{1}{(1 + de)^t}$, with de being the discount rate for the health outcome values at any cycle t . The first cycle is not discounted, as the first time point t is equal to zero.
- tc as the ni vector of total (discounted) costs for all individuals
- te as the ni vector of total (discounted) health outcomes for all individuals

Below, we present an algorithm that can be used to perform a state-transition microsimulation. The enumerated areas on the left correspond to the steps of the microsimulation algorithm outlined before.

```

1  [ for  $i = 1$  to  $ni$  do
    [  $M_{i0}$  is assigned the initial health state for individual  $i$ .
    [  $C_{i0} = Costs(M_{i0}, X_{i0})$  {Cost value for individual  $i$  during cycle 0 as function of the initial health state and individual's characteristics}
    [  $E_{i0} = Effs(M_{i0}, X_{i0}, cl)$  {Health outcomes for individual  $i$  during cycle 0 as function of the initial health state, individual's characteristics and cycle length}
    [ Then,
    [ for  $t = 1$  to  $nt$  do
2  [  $p = Probs(M_{i[0:t]}, X_{i[0:t]})$  {State-transition probabilities for individual  $i$  at cycle  $t$  as function of the complete history of states and individual characteristics up to the current cycle  $t$ }
3  [  $M_{it} \sim Cat(n, p)$  {Sample the state individual  $i$  will transition to during cycle  $t$  from a categorical distribution ( $Cat$ ) of  $ns$  states  $n$  with probabilities  $p$ }
    [ Assign state values for costs and health outcomes using the  $M$  matrix and individual characteristics  $X$ .
4  [  $C_{it} = Costs(M_{i[0:t]}, X_{i[0:t]})$  {Costs for individual  $i$  during cycle  $t$  as function of the complete history of states and individual characteristics up to the current cycle  $t$ }
    [  $E_{it} = Effs(M_{i[0:t]}, X_{i[0:t]}, cl)$  {Health outcomes for individual  $i$  during cycle  $t$  as function of the complete history of states and individual characteristics up to the current cycle  $t$  and the cycle length}

    [ end
    [ end

5  [  $tc = C \cdot dwe$  {Total (discounted) cost per individual for all individuals, “.” denotes inner product multiplication}
    [  $te = E \cdot dwe$  {Total (discounted) health outcomes per individual for all individuals, “.” denotes inner product multiplication}
  ]

```

The implementation of the microsimulation algorithm using R syntax is illustrated in Box 1. The notation in Box 1 follows the algorithm outlined before with inner product multiplication indicated in R by the “%*%” symbol. Because t starts at 0, we increment t by 1 to correspond to the element index in the vectors and matrices in R. Variable names for matrices are preceded by the prefix “m.”, whereas, for vectors, the prefix “v.” is used. At each individual and each cycle, the vector of probabilities “v.p” is replaced with a new set of transition probabilities. The `Probs()` function returns a vector of state transition probabilities. Finally, both `Costs()` and `Effs()` functions yield a single value of the cost and health outcomes per individual per cycle.

```

# The matrices that will store information across the iterations need to be initialized first
m.M <- m.C <- m.E <- matrix(nrow = n.i, ncol = n.t + 1) # create empty matrices
v.n <- 1:n.s # the vector of health states
v.dwc <- 1 / (1 + dc) ^ (0:n.t) # cost discount weights based on the discount rate dc
v.dwe <- 1 / (1 + de) ^ (0:n.t) # health outcome discount weights based on the discount rate de

for (i in 1:n.i) {
  # Step 1:
  m.M[i, 1] <- v.M_1[i] # specify the initial health state per individual
  m.C[i, 1] <- Costs(m.M[i, 1], m.X[i, 1]) # costs per individual during cycle 0
  m.E[i, 1] <- Effs (m.M[i, 1], m.X[i, 1], c1) # health outcome value per individual during cycle 0

  for (t in 1:n.t) {
    # Step 2
    v.p <- Probs(m.M[i, 1:t], m.X[i, 1:t]) # calculate the transition probabilities
    # Step 3
    m.M[i, t + 1] <- sample(v.n, prob = v.p, size = 1) # sample the health state at cycle t+1 and store i
    # Step 4
    m.C[i, t + 1] <- Costs(m.M[i, 1:t + 1], m.X[i, 1:t + 1]) # costs per individual during cycle t + 1
    m.E[i, t + 1] <- Effs (m.M[i, 1:t + 1], m.X[i, 1:t + 1], c1) # health outcome per individual during
    cycle t + 1
  } # close the loop for time cycles
} # close the loop for the individuals

# Step 5
tc <- m.C %*% v.dwc # total discounted cost per individual
te <- m.E %*% v.dwe # total discounted health outcomes per individual

```

Box 1 A generic implementation of the microsimulation algorithm outlined before using R syntax. The R variable names are defined in Table 1.

The algorithm in Box 1 illustrates the microsimulation process for one treatment strategy. In cases where 2 or more strategies are compared, the algorithm should be completed separately for each strategy with strategy-specific values. The vectors of undiscounted total costs and health outcomes per individual can be calculated by setting dc and de equal to zero. The approach above can be readily expanded or simplified. For example, in the context of a cost-utility economic evaluation, the QALYs accumulated within a cycle would be captured in the health outcome matrix E and, consequently, te would capture the total QALYs across all cycles. In extension, \bar{tc} and \bar{te} , the mean of individual-specific costs and QALYs for each strategy, respectively, could be calculated to generate standard incremental cost-effectiveness tables.

Stochastic Variation and Random Sampling in Microsimulation Models

Microsimulation models rely on Monte Carlo (MC) simulation methods to describe the stochastic transition process of individuals through the model.²⁷ Due to this stochastic process, the estimates of a microsimulation

will be different between individuals. Therefore, executing the model for a sufficiently large number of individuals is important to achieve a representative distribution of population outcomes. The variability around the mean model estimate is called Monte Carlo Standard Error (MCSE). In a decision problem that could be described with either a microsimulation or a cohort model, the 2 models types should yield the same estimates as the number of simulated individuals in the microsimulation model increases.^{28,29} In this tutorial, we use graphical diagnostics to show how outcomes produced using a microsimulation model converge to those produced using a cohort model. One such example is a plot of the mean simulation estimates as a function of the number of individuals run in the microsimulation model (i.e., microsimulation sample size).

When designing a microsimulation model for comparative effectiveness or cost-effectiveness purposes, it is important that simulated individuals are as similar as possible across comparators, except for the intervention to which they are exposed. One can think of the random sampling process as 2 parallel processes, where the same hypothetical individual is exposed to both the

intervention and the control. This can be achieved by using pre-sampled values from the distributions used in the model, by explicitly setting a seed number per individual.³⁰ This ensures that individuals across intervention scenarios share the same baseline characteristics, and reduces any difference observed between the intervention scenarios that can be attributed to MC variability and not to intervention allocation. An additional benefit of setting the seed is that it allows the results to be reproducible despite the inherent stochastic process.

Example: The Sick-Sicker Model

In this section, we illustrate the implementation of the algorithm presented above, using a previously published decision model.³¹ This illustration provides an example of how R can be used for the development of a microsimulation model for a specific disease process. We intend for the R code below to be generic to allow it to be customizable. There are many opportunities to improve the efficiency of the code for specific applications but the goal here is to provide a generic code as a starting point.

The illustration relies on a state-transition cohort model of a hypothetical disease (the Sick-Sicker model), first described by Enns et al.³¹ We initially create a microsimulation implementation of the Sick-Sicker cohort model and subsequently extend it to illustrate the advantages of microsimulation models in incorporating individual-level characteristics and “memory” in the disease process. The Sick-Sicker model consists of 4 health states: healthy (H), sick (S1), sicker (S2) and dead (D) (Figure 1). All individuals are assumed to be healthy at baseline. Over time, healthy individuals face a risk of developing the disease and progressing to S1. Individuals in S1 can recover (return to state H), stay in S1, progress further to S2 or die. Individuals in S2 cannot recover (i.e., cannot transition to either S1 or H). Individuals in H are assumed to have a fixed mortality rate. Individuals in S1 and S2 have an increased mortality rate compared with healthy individuals. All individuals are followed for 30 years using annual cycles. This cycle length of 1 year is assumed to be constant throughout the model’s time horizon.

We evaluate 2 alternative strategies: a no-treatment and a treatment strategy. The treatment improves utility for those individuals in the S1 state but has no impact on utility for those in the S2 state. However, due to the nature of this hypothetical disease, we are not able to distinguish those who are in S1 from those in S2. Therefore, under the treatment strategy, individuals who occupy states S1 or S2 receive treatment and continue doing so until they recover to the healthy state or die.

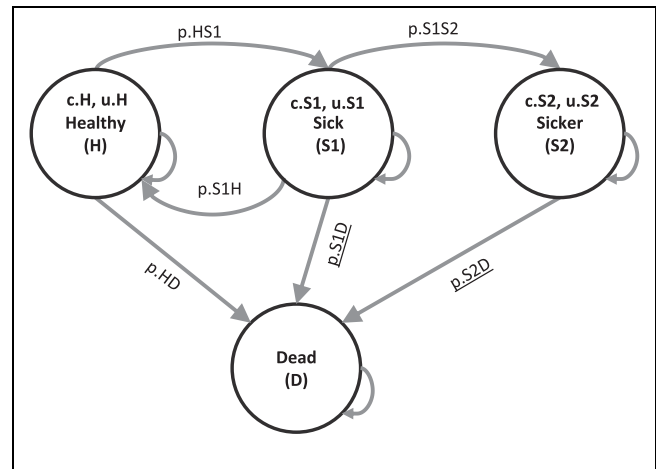


Figure 1 Schematic representation of the Sick-Sicker microsimulation model. In the extended microsimulation model the underlined probabilities ($p.S1D$ and $p.S2D$) are adjusted to incorporate time-dependent transitions.

In the extension of the model that incorporates “memory” and individual-level characteristics, 3 modifications are made to the model. Firstly, we assume that the benefits of treatment wane over time, with the utility of those in S1 on treatment decreasing by 0.03 every year they spend in S1. The second modification involves the dependency of mortality rates on the duration of remaining in the disease states (a 20% increase every cycle in S1 or S2). Finally, we also assume that the improvement on quality of life by the treatment varies across individuals through a characteristic that acts as a treatment effect modifier. All model parameter values and R variable names are presented in Table 1.

The probabilities to die when sick ($p.S1D$) and sicker ($p.S2D$) are calculated by converting ($p.HD$) the probability to die when healthy) to a rate, multiplying it by the rate ratios $rr.S1$ and $rr.S2$, respectively, and then converting them back to a probability. In the extended model, not only the rate ratios but also the variable $rp.S1S2$ and the duration of being sick are used to adjust the rates. See the R code below for more details on the calculations.

In the next section, we provide the R code and some documentation for the most important steps of the algorithm. We first illustrate the direct microsimulation implementation of the Sick-Sicker model and then provide the modifications necessary to extend the model to the scenario where “memory” is incorporated in the model. A full working version of the R codes used for each decision model can be found in the Appendices and on GitHub.

Table 1 Input Parameters for the Illustrative Microsimulation Model

Parameter	R name	Value
Time horizon (<i>nt</i>)	n.t	30 y
Cycle length (<i>cl</i>)	cl	1 y
Number of simulated individuals (<i>ni</i>)	n.i	100,000
Names of health states (<i>n</i>)	v.n	H, S1, S2, D
Annual discount rate (costs/QALYs) (<i>dc/de</i>)	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

Microsimulation Implementation of the Sick-Sicker Model

The R code is broken down into 3 sections: the model input section, the model's main section, and the model output section. In the model input section, input parameters are loaded into the R environment. We assume

that the health state of the individuals is known at the start of the simulation, and stored in the variable `v.M_1`. In our example, all individuals enter the model as healthy (i.e., at health state (H)).

First, we specify the number of individuals, their initial state, the time horizon, the health states, the discount rates, and the intervention strategies.

```
n.i <- 100000 # number of simulated individuals
n.t <- 30 # time horizon, 30 cycles
v.n <- c("H", "S1", "S2", "D") # 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
```

The next lines of R code specify the transition probabilities, cost, and the utility input for individuals in the treatment and the no-treatment group (Table 1).

```
# 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
r.S1 <- 3 # rate ratio of death when sick vs healthy
r.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 sicker and being treated
```

In the model's main section, we define 4 R functions that constitute the core of the microsimulation model. In the following paragraphs, we describe each component of these 4 functions.

The main function is the `MicroSim()` function, which operationalizes the algorithm steps described above in R, adjusted to represent the Sick-Sicker model. The `MicroSim()` function includes arguments that control the input parameters (e.g., names of health states) and the components of the model itself (e.g., the seed number). It also provides the option to the modeler to record the transitions between states at every cycle in a TS matrix of size $ni \times (nt + 1)$ and the calculation of a microsimulation trace TR , which captures the proportion of individuals occupying each state at every cycle. Some

arguments have a default setting, which means that, in case the user does not specify them, they will be assigned a default value (e.g., the arguments `TR.out` and `TS.out` that control whether a microsimulation trace and a transition array will be calculated are by default set to `TRUE` and the default seed number is 1). The `Trt` argument is a scalar with a Boolean value (by default set to `FALSE`), used to run the Sick-Sicker model for a group of treated or non-treated individuals. In other situations, modelers can adjust the code to make use of a vector with Boolean values to model mixed groups of treated and non-treated individuals. The following paragraphs describe each component of the `MicroSim()` function. The R code lines below describe the arguments and functions used in the `MicroSim()` function.


```

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 outcomes (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)

```

In addition to the `MicroSim()` function, we construct 3 functions that need to be specified and executed before execution of the `MicroSim()` function. These are the `Probs()`, `Costs()` and `Effs()` functions, which are used to update the probabilities, costs, and health outcome values at each cycle, respectively. We will provide more details on these functions later in this tutorial.

Within the `MicroSim()` function, 2 vectors with discount weights are generated based on the discount rates for costs and health outcomes, respectively, `d.c` and `d.e`, specified in the argument section of the function. The first cycle is not discounted.

```

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

```

In R, variables used to store information across iterations need to be initialized and their dimensions need to be declared outside of the iterative process. Hence, we first initialize the variables that will capture the matrices M , C , and E .

```

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 = " ")))

```

Next, we specify the health states the individuals occupy at the start of the simulation.

```

m.M[, 1] <- v.M_1    # indicate the initial health state

```

Given M_{i0} , we can calculate C_{i0} and E_{i0} for every individual i and store this information in `m.C` and `m.E`.

```

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

```

At the beginning of all the subsequent cycles t , the transition probabilities $v.p$, given the states occupied at the beginning of the cycle, are specified. The states M_{it} occupied by any individual i are sampled from a categorical distribution

with frequency based on $v.p$. The next step is to calculate C_{it} and E_{it} for each individual i during each cycle t . The costs and QALYs only depend on the current state of the individual and on whether they are receiving treatment or not. Note that $m.C$ and $m.E$ are initialized, whereas $v.p$ is not, as we are not interested in storing it for every iteration and it will therefore be overwritten at every cycle.

```
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
}
```

The following lines of code display the progress of the simulation.

```
} # 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
```

Once the iterative process is completed, the model's output is captured in the initialized variables. To calculate the (discounted) total costs and QALYs per individual across all cycles, we take the inner products of C and E with the discount weight vectors. The average total costs and QALYs within this patient population are estimated by averaging across the individual total costs and QALYs.

```
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
```

The function `MicroSim()` also offers the modeler the option to generate a microsimulation trace matrix TR and a matrix of transitions TS (i.e., argument set to `TRUE`).

```
if (TS.out == TRUE) { # create a matrix of transitions across states
  TS <- paste(m.M, cbind(m.M[, -1], "D"), sep = "->") # transitions from one state to the other
  TS <- matrix(TS, nrow = n.i)
  rownames(TS) <- paste("Ind", 1:n.i, sep = "")      # name the columns
  colnames(TS) <- paste("Cycle", 0:n.t, sep = "")    # name the rows
} else {
  TS <- NULL
}
if (TR.out == TRUE) {
  TR <- t(apply(m.M, 2, function(x) table(factor(x, levels = v.n, ordered = TRUE))))
  TR <- TR / n.i                                     # create a distribution trace
  colnames(TR) <- v.n                               # name the columns
  rownames(TR) <- paste("cycle", 0:n.t, sep = "")   # name the rows
} else {
  TR <- NULL
}
```

Finally, all the function's output is stored and returned as a list object.

```

results <- list(m.M = m.M, m.C = m.C, m.E = m.E, tc = tc, te = te, tc_hat = tc_hat, te_hat
= te_hat, TS = TS, TR = TR) # store the results from the simulation in a list
return(results) # return the results
} # end of the MicroSim function

```

Key components in any microsimulation structure are the functions used to update the transition probabilities and to estimate the costs and health outcomes for every cycle. Below, we present the R code used to describe these functions for the Sick-Sicker model.

The `Probs()` function is used to update transition probabilities based on the health state occupied at cycle t , as represented in Figure 1. The `M_it` argument in the context of the Sick-Sicker model represents a single character, representing the health state for individual i at cycle t , extracted from the `m.M` matrix.

```

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 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 produce an error
}

```

In this simple example where there is no time variation in the transition probabilities, one could specify a transition probability matrix and rely on linear algebra to simplify the function `Probs()`. At the end of the function, there is a check that all probabilities sum to one. If this condition is not met, an error message appears.

In this example, the function `Costs()` depends on the current health state and treatment status of the individual. Individuals occupying states S1 and S2 have higher costs if treated compared to untreated individuals. If `Trt` is set to `TRUE`, the function will estimate the costs for the treatment strategy; otherwise, the costs for the no treatment group are estimated.

```

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
}

```

The function `Effs()` is used for the estimation of the health outcomes, QALYs in our example, during each cycle based on the current health state. If `Trt` is set to `TRUE`, the function will estimate the QALYs for the treatment strategy; this means that all individuals in the sample are treated when sick or sicker, otherwise the QALYs for the no treatment group are estimated. The variable `c1` indicates the cycle length.

```
Effs <- function (M_it, Trt = FALSE, c1 = 1) {
  # M_it: health state occupied by individual i at cycle t (character variable)
  # Trt: is the individual treated? (default is FALSE)
  # c1: 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 * c1 # calculate the QALYs during cycle t
  return(QALYs) # return the QALYs
}
```

Results of the Simple Sick-Sicker Microsimulation

After all the functions and parameters are defined, the `MicroSim()` function is executed and the outcomes are stored in the lists named `sim_no_trt` and `sim_trt` for the no-treatment and treatment strategies, respectively.

```
# Run the simulation for both no treatment and treatment
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
```

The outcomes stored in these lists are useful to generate graphical representations of individuals' trajectories between health states over all cycles. Figure 2 shows the trajectories of 3 individuals of the Sick-Sicker model. The variation between the individual trajectories is the result of the stochastic nature of the microsimulation model. For example, an individual can stay healthy during the first 4 cycles, can become sick during cycle 5, and become sicker during cycle 6. From cycle 6 until 16, the individual remains in the sicker state and during cycle 17 the individuals dies (see top trajectory Figure 2).

The cost and QALY outcomes for these 3 individuals across the time horizon are represented in graphical form in Figure 3.

Because this model was developed to answer an economic evaluation question, we need to calculate the incremental costs (ΔC), QALYs (ΔE), and the incremental costs effectiveness ratio (ICER). The code below performs these calculations and stores the values in a table.

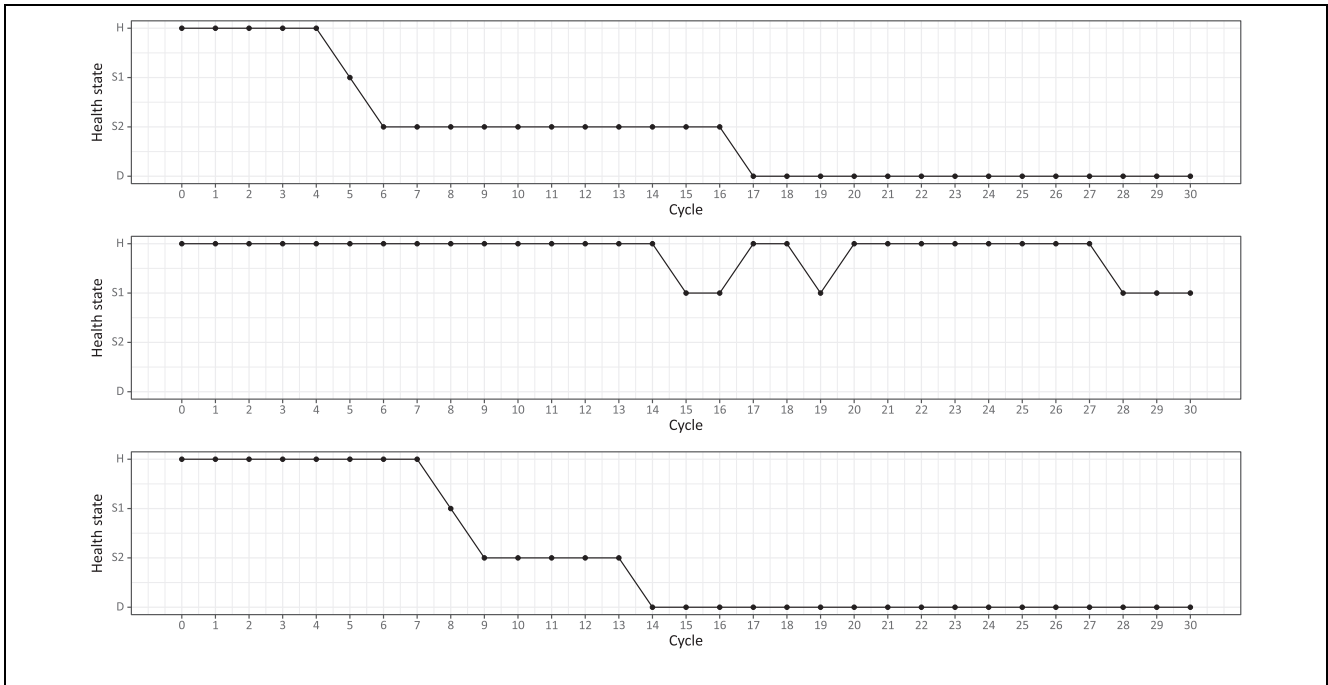


Figure 2 Trajectories between health states for 3 individuals in the Sick-Sicker microsimulation model demonstrating the health state occupied by the individual during each cycle of the simulation. In addition, this figure demonstrates the heterogeneity between individuals. Health state 1: Healthy (H), 2: Sick (S1), 3: Sicker (S2) and 4: Dead (D).

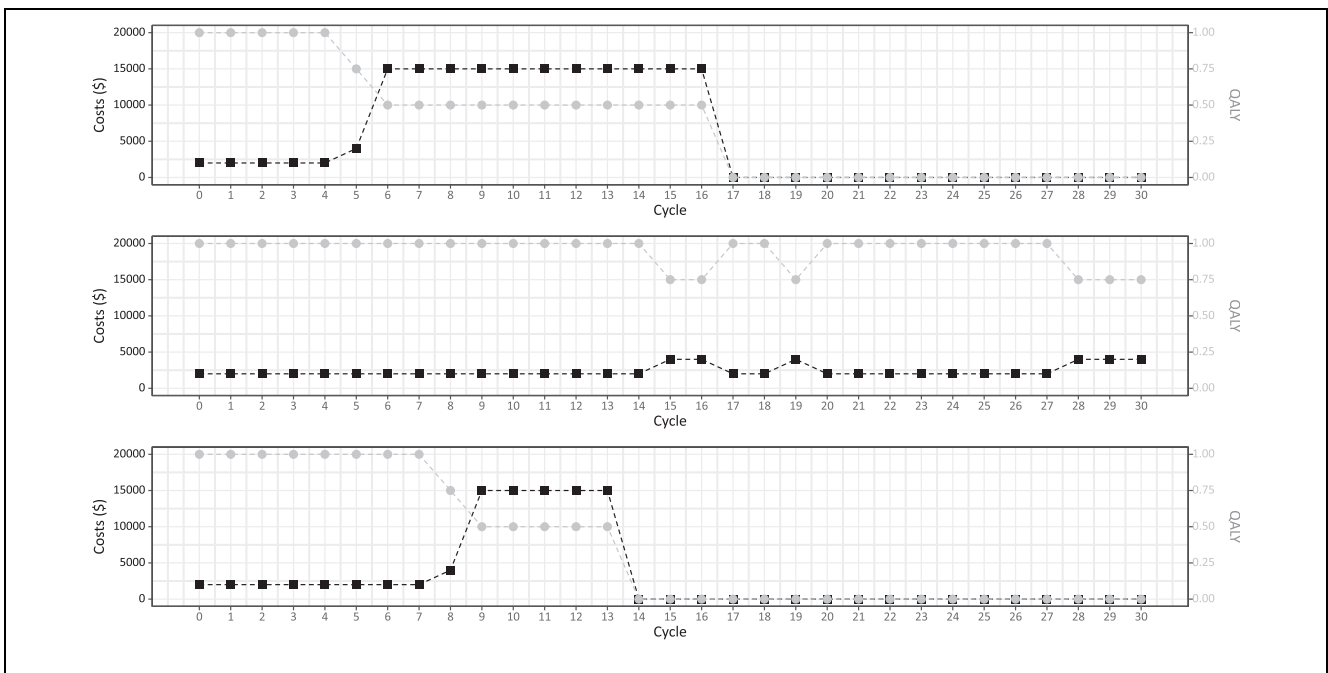


Figure 3 Graphical representation of the state costs (black squares, left y-axis) and QALYs (gray dots, right y-axis) associated with individual trajectories of the first 3 individuals in the simple microsimulation model during all cycles.

```

# 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 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.C <- sd(sim_trt$tc - sim_no_trt$tc)/sqrt(n.i) # Monte Carlo squared error (MCSE) of
incremental costs
se.delta.E <- sd(sim_trt$te - sim_no_trt$te)/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

# 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(se.delta.C, 0)), # incremental costs
  c("", round(sd.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 columns
table_micro # print the table

```

The `table_micro` variable includes the costs and QALYs per strategy, the incremental costs and QALYs, and the ICER of the treatment strategy. Table 2 presents the results of the cost-effectiveness analysis using the microsimulation model for a sample size of 10,000 and 100,000 individuals. For comparison purposes, the results based on the deterministic cohort model are also presented (R code in

Supplementary Appendix C). We can observe that the microsimulation model and the cohort model produce almost identical results after a large number of simulations.

Because microsimulation models generate outcomes for each individual, it is possible to observe the distribution of the outcomes (costs and QALYs) by creating histograms (Figure 4).

Table 2 Cost-effectiveness Analysis Results of the Simple Microsimulation Model with Two Different Population Sizes (10,000 and 100,000) Compared with Results from a Deterministic Cohort Model of the Sick-Sicker Model

Strategies	Total		Incremental		ICER (\$/QALY)
	Costs (\$)	QALYs	Costs (\$)	QALYs	
<i>Microsimulation model (n = 10,000 and seed = 1)</i>					
No-treatment	75,790 (577)	15.86 (0.049)	-	-	-
Treatment	141,211 (1080)	16.42 (0.051)	65,420 (517)	0.561 (0.004)	116,609
<i>Microsimulation model (n = 100,000 and seed = 1)</i>					
No-treatment	75,996 (183)	15.82 (0.016)	-	-	-
Treatment	141,644 (343)	16.38 (0.016)	65,648 (164)	0.561 (0.001)	117,087
<i>Deterministic cohort model</i>					
No-treatment	75,976	15.83	-	-	-
Treatment	141,623	16.40	65,647	0.562	116,901

All microsimulation results were generated by setting the random seed to 1 at the beginning of the simulation. Monte Carlo standard error in brackets.

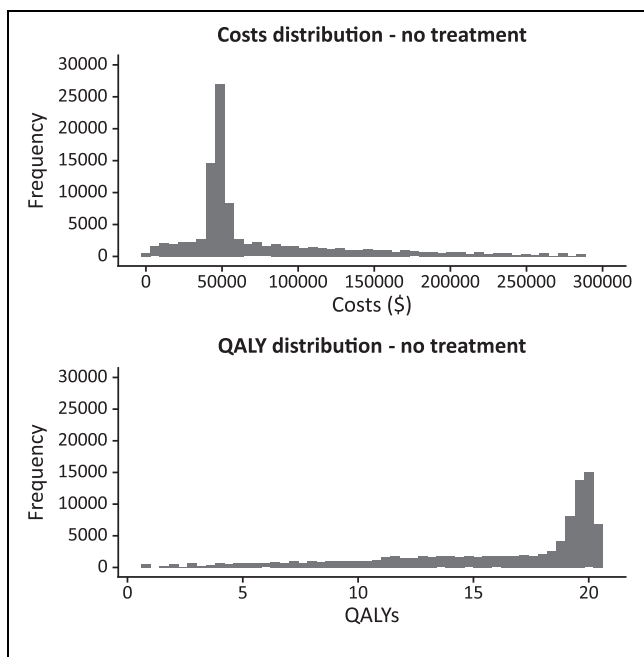


Figure 4 Histograms of the individual costs (top) and individual QALY (bottom) outcomes for the no-treatment strategy for the simple microsimulation model ($n = 100,000$).

Comparing Microsimulation to Cohort Model Outcomes

Outcomes from microsimulation models where no “memory” and no heterogeneity at baseline across individuals is assumed should asymptotically converge to those from a deterministic cohort model as the number of individuals simulated in the microsimulation model

increases.^{11,28,29} The number of hypothetical individuals that need to be generated is related to the magnitude of the MCSE. In Figure 5, we present the proportion of individuals occupying each health state at every cycle of both the microsimulation model and the cohort model. As illustrated in Figure 5, the microsimulation trace converges to the Markov cohort trace as the number of simulated individuals increases.

In addition, the Markov and microsimulation cost-effectiveness outcomes traces are graphically presented to illustrate the convergence of the microsimulation model to the outcomes derived using a cohort representation of the model (Figure 6).

Methods: Adding Memory to the Sick-Sicker Model

In this section, we highlight 2 of the advantages of using a microsimulation implementation of a cohort model, which are to incorporate 1) ‘memory’ into the disease dynamics, and 2) variation in the baseline characteristics for every individual. To illustrate this, we extend the Sick-Sicker microsimulation model to include memory effects and patient heterogeneity at baseline. Specifically, we assume that mortality rate increases the longer a patient spends in one of the sick states (Figure 1 and Table 1) and that effectiveness of treatment is dependent on the duration of stay in the sick states and on baseline characteristics. Below, we outline the modifications and additions that are necessary to incorporate these changes to the model. Supplementary Appendix B includes the complete R code used to build the “extended” microsimulation model.

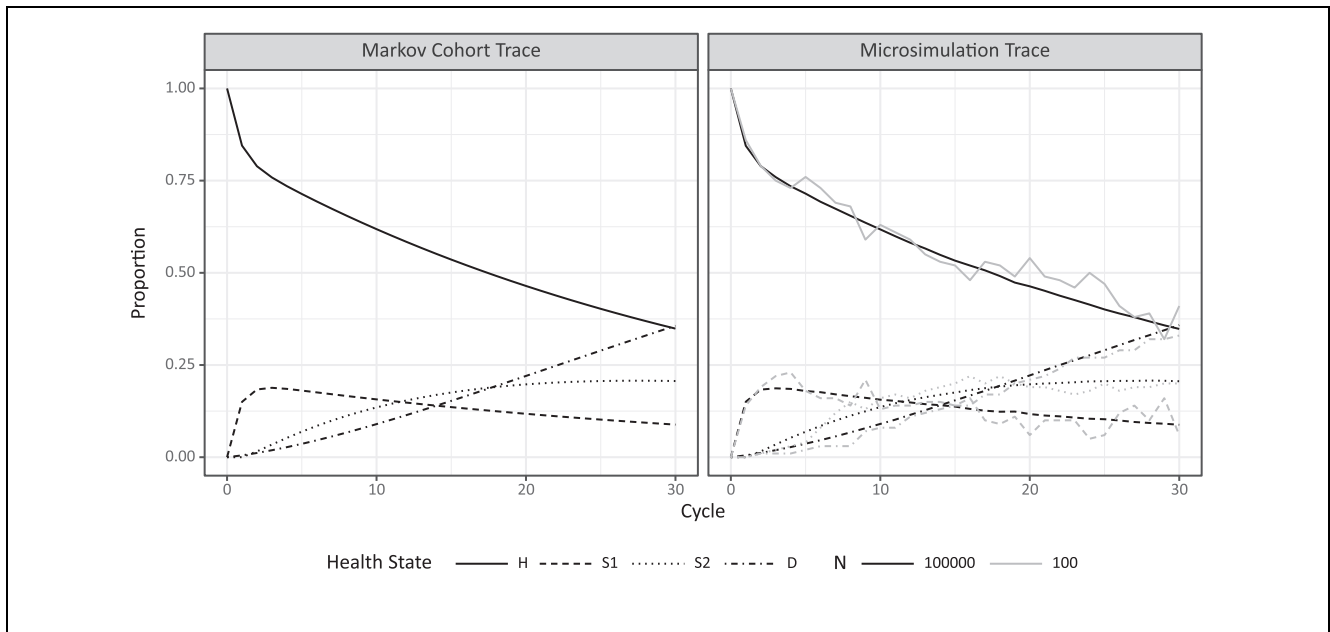


Figure 5 Markov cohort trace (left) and the microsimulation trace of the simple microsimulation (right) for different numbers of individuals (gray line: $n = 100$; black line: $n = 100,000$). The y-axis represents the proportion of individuals in each health state. Because all individuals start healthy, the solid line (H) starts at 1. Over time, the individuals transit from H (solid line going down) towards other health states (other lines increases).

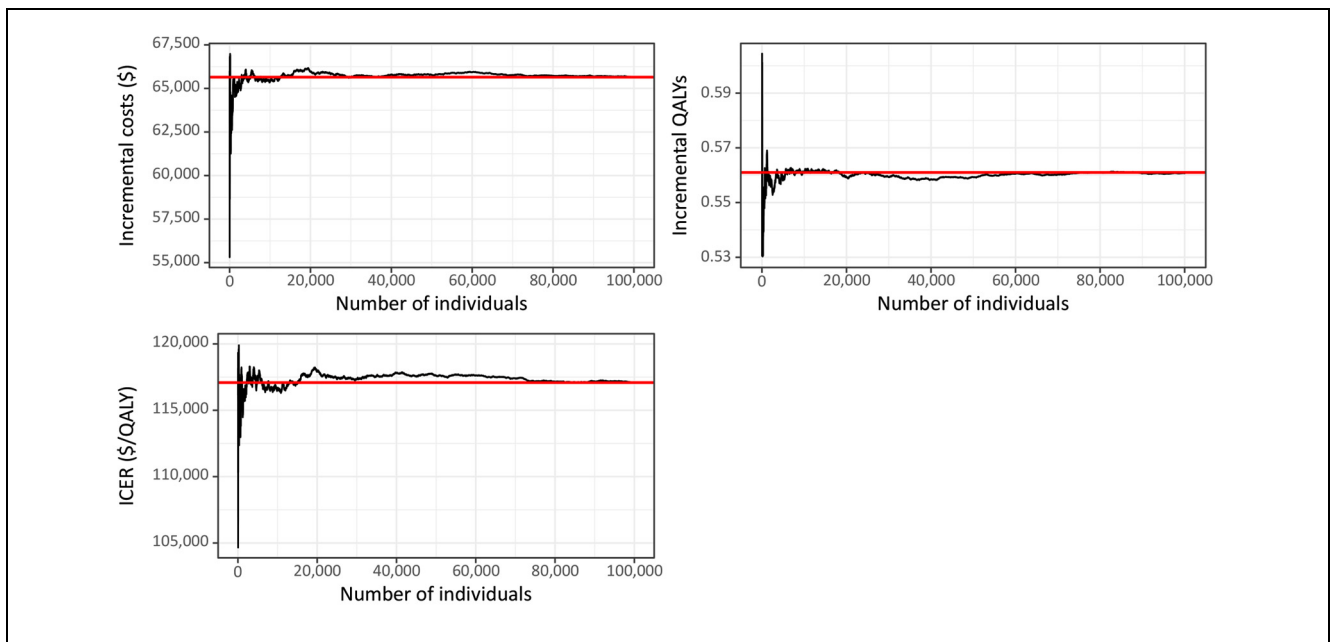


Figure 6 Cost-effectiveness analysis results from the simple microsimulation model with an increasing number of individuals (up to $n = 100,000$). The x-axis represents the number of individuals in the microsimulation model, the y-axis is the values for the incremental costs (\$), incremental QALYs, and ICER (\$/QALY). Horizontal line: cohort model results. Left top: Convergence of incremental costs, right top: convergence of QALYs left bottom: convergence of the ICER. QALY, quality-adjusted life years.

We initially assume that individuals in our cohort have a baseline characteristic that acts as an effect modifier to the treatment effect. This baseline characteristic x is assumed to follow a uniform distribution within the population of interest.

```
v.x <- runif(1, 0.95, 1.05) # vector
capturing individuals' effect modifier
at baseline
```

To include this extra variable, the `MicroSim()` function in the extended model has an extra argument X . This argument is assigned the name of the vector or matrix capturing any individual characteristics.

Additionally, we introduce 2 new variables—the annual increase in mortality rate (`rp.S1S2`) and the annual decrease in utility for treated sick patients (`ru.S1S2`)—with every additional cycle spent in the sick states (S1 or S2).

```
rp.S1S2 <- 0.2 # increase of the mortality
rate with every additional year being sick/
sicker
ru.S1S2 <- 0.03 # decrease in utility of
treated sick individuals with every
additional year being sick/sicker
```

Subsequently, we introduce the help variable `dur`, which stores the number of consecutive cycles the individual remains in either S1 or S2. Because every individual starts in the healthy state, the `dur` variable takes the value 0 at cycle 0. With every additional cycle that an individual spent in S1 or S2, a value of 1 is added to `dur`. When the individual recovers (transitions back to state H), the `dur` variable is set again to zero. This means that all previous cycles spent in one of the sick health states does not influence the current transition probabilities. The process is repeated every time an individual transitions to S1.

```
dur <- 0 # the individual start without history
m.C[i, 1] <- Costs(m.M[i, 1], Trt) # estimate the cost per individual of the initial health state
m.E[i, 1] <- Effs(m.M[i, 1], dur, Trt, X = X[i]) # estimate the health outcome per individual at the
initial health state conditional on treatment, duration of being sick/sicker and individual characteristics

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 being sick/sicker
  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 treatment
  m.E[i, t + 1] <- Effs(m.M[i, t + 1], dur, Trt, X = X[i]) # estimate the health outcome per individual
during cycle t + 1 conditional on treatment, duration of being sick/sicker and individual characteristics

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

We introduce minor modifications on the `Probs()` function to take the history of the individual into account when updating the probability of transitioning to death from S1 and S2. The modified function is presented 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.
  new [
    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
    r.S2D <- 1 - exp(-r.S2D * (1 + dur * rp.S1S2)) # calculate p.S2D conditional on duration of being sick/sicker

  # update the 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 produce an error
}

```

The next step is to estimate the cost and QALYs during each cycle. The new `Effs()` function includes the effect modifier `v.x`, the variable `dur` and `ru.S1S2` as input parameters.

```

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? (defaults is FALSE)
  # cl: the cycle length (default = 1)
  # X: the vector or matrix of personal 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
  new [
    u.it[M_it == "S1"] <- X * Trt * (u.Trt - dur * ru.S1S2) + (1 - Trt) * u.S1 # update the
    utility if sick conditional on treatment and duration of being sick/sicker
    [M_it == "S2"] <- u.S2 # update the utility if sicker

  QALYs <- u.it * cl # calculate the QALYs at cycle t
  return(QALYs) # return the QALYs
}

```

Table 3 Cost-effectiveness Analysis Results for the Extended Microsimulation Model for the Sick-Sicker Model with a Population Size of 100,000 Individuals

Strategies	Total		Incremental		ICER (\$/QALY)
	Costs (\$)	QALYs	Costs (\$)	QALYs	
<i>Extended microsimulation model (n = 100,000 and seed = 1)</i>					
No-treatment	62,667 (120)	15.28 (0.017)	-	-	-
Treatment	117,455 (231)	15.79 (0.017)	54,787 (117)	0.507 (0.001)	107,986

The microsimulation results were generated by setting the random seed to 1 at the beginning of the simulation. Monte Carlo standard error in brackets.

No modification of the costs is needed, hence the function `Costs()` remained the same as in the simple microsimulation model. The only thing left is to run the model with all the adjustments.

```
# Run the simulation for both no treatment and
treatment
sim_no_trt <- MicroSim(v.M_1, n.i, n.t, v.n,
X = v.x, d.c, d.e, Trt = FALSE) # run for no treatment
sim_trt <- MicroSim(v.M_1, n.i, n.t, v.n,
X = v.x, d.c, d.e, Trt = TRUE) # run for treatment
```

Results of the Extended Sick-Sicker Model

The cost-effectiveness analysis based on the extended microsimulation model is presented in Table 3. The expected costs, QALYs, and ICER of this simulation are lower in both the no-treatment and treatment strategy compared with the simple microsimulation (and cohort model) (Table 2 and Table 3).

The results are consistent with the modifications that were applied to the model. With an increasing probability of death for each additional year spent in the sick health states, patient life-expectancy is reduced, resulting in fewer QALYs as well as lower healthcare costs. The gains in QALYs due to treatment are also reduced compared with the simple model due to the decreasing effectiveness of treatment over time in the extended model. The microsimulation traces presented in Figure 7 confirm the increased mortality rate. Finally, the uncertainty around the health outcomes, QALYs, has increased due to the introduction of the treatment effect modifier.

Vectorized Implementation of the Microsimulation Model

A microsimulation process was presented above as an iterative process at an individual level over the model's time horizon. Because of the causal dependence of future

events on past event histories in the simulation, temporal iterations in the model cannot be easily vectorized. However, the repetitive process of a microsimulation across the individuals can be intuitively represented in a vectorized form. In this case, at each time iteration, all simulated individuals are transitioned simultaneously through the model. Vectorization can improve computational performance but often results in decision models with high-dimensionality components. For example, what was a vector of transition probabilities for an individual on a given point in time will become a matrix of probabilities, with the number of individuals in rows and the number of states in columns. Such higher dimensionality sometimes increases the complexity of the model structure. A frequently used tool in vectorization is linear algebra, as it can provide a convenient way of performing operations in a vector or matrix scale. Although linear algebra can facilitate quick calculations, it can also generate more complex coding and larger matrices that will require more computer memory. Hence, with vectorization, modelers must achieve a balance between conceptual complexity, memory, and computational efficiency.

Vectorization can be achieved in the context of our microsimulation example by extending the dimension of the health state sampling process. The function `sample()` used in this tutorial can support a vectorized solution only if the state transition probabilities are the same across individuals. However, in a microsimulation context, individuals' risk often depends on a number of individual-specific characteristics, which restrict us from using the `sample()` function on a vectorized context. To overcome this, and take advantage of vectorization solutions, we developed the `samplev()` function by modifying a random number generating function for multinomial variables from the `Hmisc` package.³² The modification allows for a health state-specific vectorization solution based on vectors of random numbers generated from a $U(0,1)$ distribution. The `samplev()` function randomly draws the state name at $t + 1$ based on the probability of each individual occupying each state at $t + 1$.

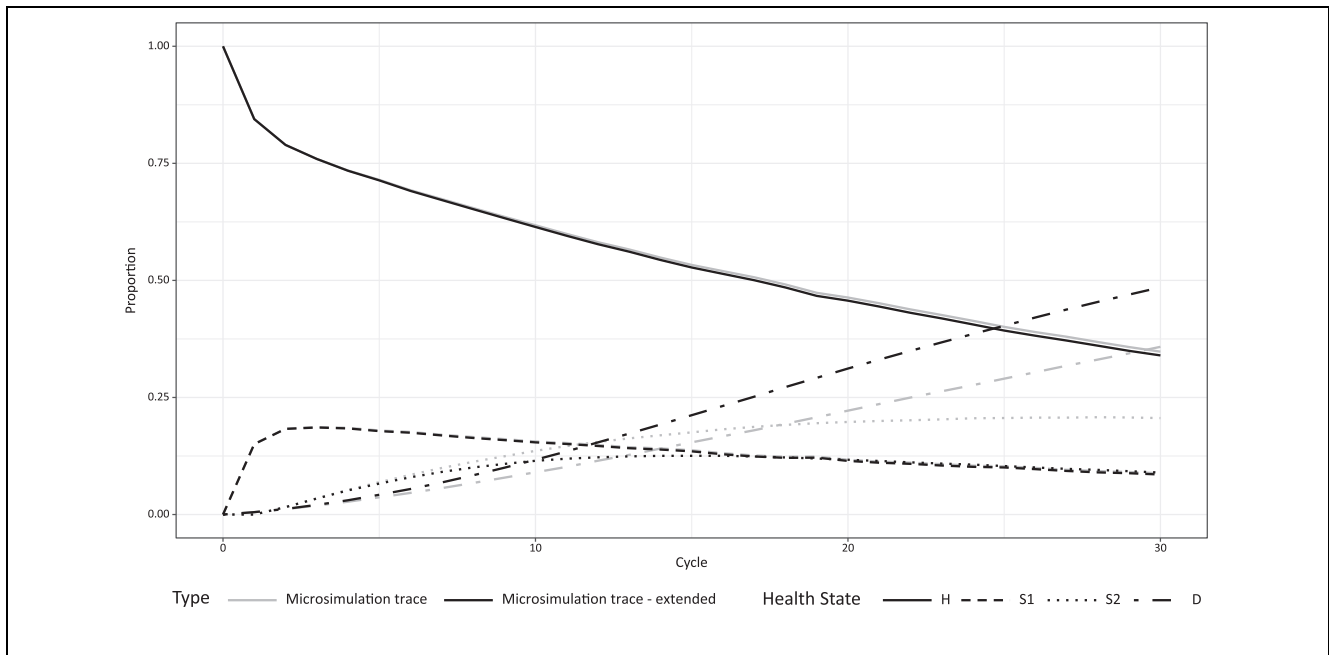


Figure 7 Trace of the simple microsimulation (gray line) and the extended microsimulation (black line) during all cycles. The y-axis represents the proportion of individuals in each health state. Because all individuals start healthy, the solid line starts at 1. Over time, the individuals transit from healthy (H, solid line going down) to other health states; sick (S1, dashed line), sicker (S2, dotted line) and dead (D, dot-dash line) (other lines increase). The gray and black line of the states H (solid) and S1 (dashed) overlap.

Table 4 Time Comparison between the Iterative (Standard) Approach Using `sample()` and the Vectorized Approach Using `samplev()` to Run the Simple Sick-Sicker Model

Sample size	Time to run (in seconds)	
	<code>sample()</code>	<code>samplev()</code>
1,000	5.42	0.16
10,000	38.41	1.21
100,000	378.76	11.71
1,000,000	4538.80	128.79

Computer: MacBook Pro, macOS Sierra version 10.12.6
 Processor 2.3 GHz Intel Core i5, 2 cores, 4GB RAM, 1333 MHz DDR3.

To make use of the `samplev()` function, the `Probs()` function must also be modified to return a matrix of transition probabilities for the whole cohort rather than a vector of probabilities for a single individual at a time. The output of the modified `Probs()` function is therefore a matrix of size $ni \times ns$. Because the `samplev()` function samples the name of the future states of individuals conditional on their probability of experiencing

these states, state names are required as column names. In Appendix D, the code with all necessary modifications is presented.

The new `samplev()` function achieves a 97% reduction in the time needed to run the analysis (Table 4). The results in Table 4 are a time comparison, between the iterative standard approach for the simple Sick-Sicker model using the `sample()` function (Appendix A) and the vectorized approach for the simple Sick-Sicker model using the `samplev()` function (Appendix D). The time shown is the time (average of 3 runs) it takes to run the model for both the treatment and no treatment arm using the `sample()` or `samplev()` function.

Discussion

This tutorial provides guidance on the implementation of a microsimulation model using a programming language such as R. We outline the conceptual steps involved and provide an algorithm to operationalize these steps using a programming language. In addition, the tutorial illustrates an implementation of microsimulation models in R and presentation of the results in graphical form. This

tutorial focuses on the implementation rather than the conceptualization of a microsimulation model. For interested readers, Roberts and others present recommendations and best practices regarding the process of decision model conceptualization.³³ In this paper, we focus on state-transition models with discrete time intervals. Another class of models consider time as continuous and simulate discrete events. Researchers interested in the implementation of discrete event simulation models might benefit from making themselves familiar with the *simmer* package. The core of the *simmer* package is written in C++ with automatic monitoring capabilities, which makes it a fast and robust framework to develop DES models and continuous time Markov models.³⁴

There are limited examples in the literature where researchers relied on R to build a microsimulation model for health decision making. However, most of these examples have appeared recently, which is consistent with the observed trend of a recent increase in the use of R in health decision modeling.¹⁹ Manca and others published an early example of a microsimulation model built in R³⁵ and Choi and others more recently published a hypertension, stroke, and myocardial infarction microsimulation model.³⁶ For 2 other microsimulation models, MILC and CANTRANCE, the authors created R packages, which include all components of the model.^{37,38} Thus, R can be used not only to develop the model but also to provide an open-source platform for distributing the model to other users.

Building a microsimulation model in a programming language such as R has several advantages. Models are written in script form, which facilitates readability and reproducibility. The fact that R is open-source and freely available increases transparency and reproducibility of the analysis. In addition, due to R's architecture, efficient computational performance can be achieved. The output obtained from the model in R is not limited to the output generated by the proprietary software; it is possible to extract every specific outcome estimate (e.g., incidence, prevalence, and average time spent in a certain state). However, decisions on the generated output should be limited to only what is ultimately necessary for the analysis to minimize computational and memory storage demands. Finally, statistical analyses performed to provide input to the decision model could be embedded directly within the decision model.

R may not always be the most appropriate platform, depending on the nature of the microsimulation model and the skills of the researchers. R is also not necessarily an ideal language for computationally intensive processes. Other programming languages, such as C/C++ or Fortran, may

achieve significantly better performance. Combining the strengths of different languages is sometimes possible. For example, the package *Rcpp* makes it possible to interface C++ code in R functions and packages.³⁹

The flexibility of R implies that there can be multiple approaches to the operationalization of a microsimulation model. This tutorial provides one such approach but is not necessarily the most efficient. The implementation of the Sick-Sicker model was developed with a focus on clarity rather than computational efficiency. For example, one could employ matrix operations more extensively rather than relying on iterative or logical procedures. Avoiding iterative loops through vectorization in R can increase efficiency by multifold,⁴⁰ as confirmed by our results (Table 4). When iterative loops are inevitable, the use of parallel processing can drastically improve computational times for iterative procedures. Alternatively, a microsimulation model in R can be developed using object-oriented programming with S4 or References Classes. Under this approach, simulated individuals, together with their characteristics, are encoded as a class and followed over time. In such a modeling approach, interaction between individuals can be incorporated in a more straightforward manner. Implementation of an object-oriented microsimulation in R can be facilitated through the *simecol* package.⁴¹

We encourage modelers to develop and distribute implementations that are more efficient than the one presented in this tutorial. We are also in the process of developing an additional tutorial on advanced methods around microsimulation using R. This upcoming tutorial will have a strong focus on computational efficiency. However, the modeler should also be aware of the trade-off between the time savings associated with an efficient algorithm and the time investment required to achieve such efficiency. One way to reduce computational time in microsimulation models is the use of parallel processing techniques. Most modern computers are equipped with multi-core processors, which increase significantly their computational power. However, by default, R uses only a single core, thereby limiting its computational capacity. With the use of appropriate packages (e.g., *parallel*) R can achieve parallel processing capabilities, and thereby drastically reduce computational times.⁴² One caveat of parallel processing is that the interaction of R with each of the core processors is time consuming, and modelers should be aware of this when employing parallel processing techniques. Overall, it is important that the additional complexity and flexibility introduced using a programming language be justified by the complexity needed to answer the research question.

There is always a degree of uncertainty in the parameter values used in health decision models. Such parameter uncertainty can be incorporated in a decision model via probabilistic sensitivity analysis (PSA; second-order Monte Carlo simulation).^{7,43} The microsimulation approach presented above is only limited to the random variation of the individuals (first-order uncertainty) and ignores any parameter uncertainty. Efficient programming is particularly important in models that incorporate both individual variation and parameter uncertainty. Models where PSA is incorporated will require particularly long computational times. Using efficient methods when conducting PSA in R is a topic of relevance to any form of decision analysis. Therefore, we have started to prepare a separate tutorial on incorporating sensitivity analysis in decision modeling using R.

In summary, R is increasingly used for building simulation models in health decision sciences; however, information on how to perform these simulations is lacking. This tutorial provides a step-by-step guide to implementing a microsimulation model in R with the aim of supporting health decision scientists who are new to high-level programming languages to develop models in a more flexible, open-source and transparent manner, and encouraging increased transparency and reproducibility in health decision sciences. In subsequent tutorials, and as part of the Decision Analysis in R for Technologies in Health (DARTH) group efforts, we will expand on model optimization, calibration, and value of information analysis among other topics. Future code updates will be placed on our GitHub <https://github.com/DARTH-git/Microsimulation-tutorial>.

References

1. Friedman SG. Current management of the patient with internal carotid artery occlusion. *Eur J Vasc Surg*. 1989;3:97–101.
2. Kattan M. *Encyclopedia of Medical Decision Making*. California: SAGE Publications, Inc.; 2009. doi:10.4135/9781412971980.
3. Kreke JE, Schaefer AJ, Roberts MS. Simulation and critical care modeling. *Current Opin Crit Care*. 2004;10:395–8.
4. Roberts M, Russell LB, Paltiel AD, et al. Conceptualizing a model: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force–2. *Value Health*. 2012;15:804–11.
5. Caro JJ, Briggs AH, Siebert U, Kuntz KM, ISPOR-SMDM Modeling Good Research Practices Task Force. Modeling good research practices—overview: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force–1. *Value Health*. 2012;15:796–803.
6. Statistics Canada. Microsimulation approaches. 2016. Available from: <http://www.statcan.gc.ca/eng/microsimulation/modgen/new/chap2/chap2>. [Accessed July 13, 2016].
7. Hunink MGM, Weinstein MC, Wittenberg E, et al. *Decision Making in Health and Medicine. Integrating Evidence and Values*. 2nd ed. Cambridge: Cambridge University Press; 2014.
8. Lay-Yee R, Cotterell G. The role of microsimulation in the development of public policy; 2015. doi:10.1007/978-3-319-12784-2_14.
9. Siebert U, Alagoz O, Bayoumi AM, et al. State-transition modeling: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force-3. *Med Decis Making*. 2012;32:690–700.
10. Zucchelli E, Jones AM, Rice N. The evaluation of health policies through microsimulation methods. *Health, Econometrics and Data Group (HEDG) Working Papers*. 2010;5:2–20. Available from: <http://ideas.repec.org/p/yor/hectdg/10-03.html>.
11. Brennan A, Chick SE, Davies R. A taxonomy of model structures for economic evaluation of health technologies. *Health Econ*. 2006;15:1295–310.
12. Davis S, Stevenson M, Tappenden P, Wailoo AJ. Cost-effectiveness modelling using patient-level simulation; 2014. Available from: http://www.nicedsu.org.uk/TSD15_Patient-level_simulation.pdf.
13. Wolfson MC. POHEM—a framework for understanding and modelling the health of human populations. *World Health Stat Qrtly*. 1994;47(3-4):157–76.
14. Hennessy DA, Flanagan WM, Tanuseputro P, et al. The Population Health Model (POHEM): an overview of rationale, methods and applications. *Pop Health Metrics*. 2015;13:24.
15. Statistics Canada. Health models: Population health model. 2016. Available from: <http://www.statcan.gc.ca/eng/microsimulation/health/health>. [Accessed May 24, 2016].
16. Goldman DP, Shekelle PG, Bhattacharya J, et al. Health status and medical treatment of the future elderly: Final Report; 2004. Available from: http://www.rand.org/pubs/technical_reports/TR169.html.
17. Muhlberger N, Kurzthaler C, Iskandar R, et al. The ONCOTYROL Prostate Cancer Outcome and Policy Model: Effect of Prevalence Assumptions on the Benefit-Harm Balance of Screening. *Med Decis Making*. 2015;35(August):758–772.
18. Hollman C, Paulden M, Pechlivanoglou P, McCabe C. A comparison of four software programs for implementing decision analytic cost effectiveness models. *PharmacoEconomics*. 2017.
19. Jalal H, Pechlivanoglou P, Krijkamp E, Alarid-Escudero F, Enns E, Hunink MGM. An overview of R in health decision sciences. *Med Decis Making*. 2017;37:735–46.
20. R Core Team. R: A language and environment for statistical computing. Vienna, Austria: R Foundation for

- Statistical Computing; 2015. Available from: <http://www.R-project.org/>.
21. Muenchen RA. The popularity of data science software. Available from: r4stats.com/articles/popularity/
 22. The 2017 top programming languages - IEEE spectrum. Available from: <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>
 23. Baio G. Bayesian Methods in Health Economics. 1st ed. London: Chapman and Hall/CRC; 2012.
 24. Williams C, Lewsey JD, Briggs AH, Mackay DF. Cost-effectiveness analysis in R using a multi-state modeling survival analysis framework: A tutorial. *Med Decis Making*. 2017;37:340–52.
 25. Weeks M, Mitton L, Sutherland H. *Microsimulation Modelling for Policy Analysis Challenges and Innovations*. Cambridge: Cambridge University Press; 2000 p 1–11.
 26. Soares MO, Canto e Castro L. Simulation or cohort models? Continuous time simulation and discretized Markov models to estimate cost-effectiveness. CHE Research Paper 56. 2010:1-20. Available from: http://www.york.ac.uk/media/che/documents/papers/researchpapers/rp56_markov_models_to_estimate_cost-effectiveness.pdf
 27. Sawilowsky SS, Fahoom GC. *Statistics via Monte Carlo Simulation with Fortran*. Rochester Hills, MI: JMASM; 2003.
 28. Kurtz TG. Limit Theorems for sequences of jump Markov processes approximating ordinary differential processes. *J Appl Probab*. 1971;8:344–56.
 29. Whitt W. Stochastic-Process limits: An introduction to stochastic-process limits and their application to queues. New York, NY: Springer; 2002. Available from: <http://www.columbia.edu/~ww2040/preflongno.pdf>.
 30. Tezuka S. *Uniform Random Numbers: Theory and Practice*. Boston, MA: Springer US; 1995. doi:10.1007/978-1-4615-2317-8.
 31. Enns EA, Cipriano LE, Simons CT, Kong CY. Identifying best-fitting inputs in health-economic model calibration: A Pareto frontier approach. *Med Decis Making*. 2015;35:170–82.
 32. Harrell FE. Hmisc: Harrell Miscellaneous. 2017: R package version 4.0-3. Available from: <https://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf>.
 33. Roberts M, Russell LB, Paltiel AD, et al. Conceptualizing a model: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force-2. *Med Decis Making*. 2012;32:678–89.
 34. Ucar I, Smeets B. simmer: Discrete-Event Simulation for R. 2017:R package version 3.6.3. Available from: <https://cran.r-project.org/package=simmer>.
 35. Manca A, Hawkins N, Sculpher MJ. Estimating mean QALYs in trial-based cost-effectiveness analysis: The importance of controlling for baseline utility. *Health Econ*. 2005;14:487–96.
 36. Choi SE, Brandeau ML, Basu S. Expansion of the National Salt Reduction Initiative: A mathematical model of benefits and risks of population-level sodium reduction. *Med Decis Making*. 2015;36:1–14.
 37. Chrysanthopoulou SA. MILC: MICOsimulation Lung Cancer (MILC) model; 2014. Available from: <https://cran.r-project.org/web/packages/MILC/index.html>
 38. Birnbaum JK, Ademuyiwa FO, Carlson JJ, Mallinger L, Mason MW, Etzioni R. Comparative effectiveness of biomarkers to target cancer treatment: Modeling implications for survival and costs. *Med Decis Making*. 2015:1–10.
 39. Eddelbuettel D, François R. Rcpp: Seamless R and C++ Integration. *J Stat Software*. 2011;40:1–18.
 40. Gillespie C, Lovelace R. *Efficient R Programming: A Practical Guide to Smarter Programming*. 1st ed. O'Reilly Media; 2016.
 41. Petzoldt T, Rinke K. simecol: An object-oriented framework for ecological modeling in R. *J Stat Software*. 2007;22:1–31.
 42. R Core Team. Package “parallel.” Requires R version 2.14.0. Package Available from: <https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf>; 2015.
 43. Briggs AH, Weinstein MC, Fenwick EAL, et al. Model parameter estimation and uncertainty analysis: a report of the ISPOR-SMDM Modeling Good Research Practices Task Force Working Group-6. *Med Decis Making*. 2012;32:722–32.