* [Introduction](http://devinincerti.com/2019/01/01/sim-mstate.html" \l "introduction" \t "_blank)
* [An example 6-state model](http://devinincerti.com/2019/01/01/sim-mstate.html#an-example-6-state-model)
* [Estimation](http://devinincerti.com/2019/01/01/sim-mstate.html#estimation)
  + [Data](http://devinincerti.com/2019/01/01/sim-mstate.html#data)
  + [Fitting](http://devinincerti.com/2019/01/01/sim-mstate.html#fitting)
* [Simulation](http://devinincerti.com/2019/01/01/sim-mstate.html#simulation)
  + [mstate](http://devinincerti.com/2019/01/01/sim-mstate.html#mstate)
  + [hesim](http://devinincerti.com/2019/01/01/sim-mstate.html#hesim)
  + [Comparison](http://devinincerti.com/2019/01/01/sim-mstate.html#comparison)
* [Probabilistic sensitivity analysis](http://devinincerti.com/2019/01/01/sim-mstate.html#probabilistic-sensitivity-analysis)
  + [mstate](http://devinincerti.com/2019/01/01/sim-mstate.html#mstate-1)
  + [hesim](http://devinincerti.com/2019/01/01/sim-mstate.html#hesim-1)
  + [Comparison](http://devinincerti.com/2019/01/01/sim-mstate.html#comparison-1)
* [Conclusion](http://devinincerti.com/2019/01/01/sim-mstate.html#conclusion)

**Introduction**

Multi-state models are used to model a trajectory through multiple states. Survival models are a special case in which there are two states, alive and dead. Multi-state models are therefore useful in clinical settings because they can be used to predict or simulate disease progression in detail. [Putter et al.](https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2712) provide a helpful tutorial.

In this post, we will consider “clock-reset” (i.e., semi-Markov) models rather than “clock-forward” (i.e., Markov) models. In a “clock-reset” model, time refers to time since entering the most recent state, whereas in a “clock-forward” model time refers to time since entering the initial state. When using a “clock-reset” approach, state occupancy probabilities can only be computed in a general fashion via simulation.

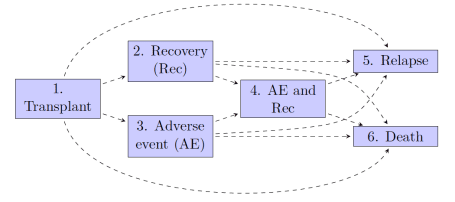
The analysis will be restricted to parametric models, which are useful for extrapolating beyond the the time horizon in the existing data. Example probability distributions include the exponential, Weibull, Gompertz, gamma, log-logistic, lognormal, and generalized gamma.

The [flexsurv](https://cran.r-project.org/web/packages/flexsurv/index.html) package will be used to estimate the parametric models and the [mstate](https://cran.r-project.org/web/packages/mstate/index.html) and [hesim](http://innovationvalueinitiative.github.io/hesim/) (admittedly developed by me) packages will be used to simulate the estimated models. We will compare the computational efficiency of different simulation methods.

**An example 6-state model**

To illustrate, we will follow [Wreede et al.](https://www.jstatsoft.org/article/view/v038i07) and use a 6-state model for leukemia patients following bone marrow transplantation (see figure below). The six states are (1) Transplant (Tx), (2) Recovery (Rec), (3) Adverse Event (AE), (4) AE and Rec, (5) Relapse (Rel), and (6) Death. The following 12 transitions are possible.

1. Tx to Rec
2. Tx to AE
3. Tx to Rel
4. Tx to Death
5. Rec to AE and Rec
6. Rec to Rel
7. Rec to Death
8. AE to AE and Rec
9. AE to Rel
10. AE to Death
11. AE and Rec to Rel
12. AE and Rec to Death



The transitions can be characterized with a 6 x 6 transition matrix, which is a square-matrix where the (i,j) element is a positive integer if a transition from i to j is possible and NA otherwise.

library("mstate")

tmat <- mstate::transMat(x = list(c(2, 3, 5, 6),

c(4, 5, 6),

c(4, 5, 6),

c(5, 6),

c(),

c()),

names = c("Tx", "Rec", "AE", "Rec+AE",

"Rel", "Death"))

print(tmat)

## to

## from Tx Rec AE Rec+AE Rel Death

## Tx NA 1 2 NA 3 4

## Rec NA NA NA 5 6 7

## AE NA NA NA 8 9 10

## Rec+AE NA NA NA NA 11 12

## Rel NA NA NA NA NA NA

## Death NA NA NA NA NA NA

**Estimation**

Parametric multi-state models can be fit using flexsurv and both non-parametric and semi-parametric models can be fit with mstate. In our analysis, we will fit a parametric model to the ebmt4 dataset from the mstate package. For additional information on model fitting and multi-state data beyond what is provided in this post, I recommend the [mstate](https://www.jstatsoft.org/article/view/v038i07) and [flexsurv](https://www.jstatsoft.org/article/view/v070i08) articles published in the *Journal of Statistical Software*.

**Data**

The ebmt4 dataset is in a “wide” format, which is not suitable for multi-state modeling. Luckily, the mstate package contains a helper function, mstate::msprep(), which can convert data in wide format to a suitable “long” format.

library("mstate")

data("ebmt4")

msebmt <- msprep(data = ebmt4, trans = tmat,

time = c(NA, "rec", "ae","recae", "rel", "srv"),

status = c(NA, "rec.s", "ae.s", "recae.s", "rel.s", "srv.s"),

keep = c("match", "proph", "year", "agecl"))

msebmt[msebmt$id == 2, ]

## An object of class 'msdata'

##

## Data:

## id from to trans Tstart Tstop time status match proph

## 8 2 1 2 1 0 12 12 0 no gender mismatch no

## 9 2 1 3 2 0 12 12 1 no gender mismatch no

## 10 2 1 5 3 0 12 12 0 no gender mismatch no

## 11 2 1 6 4 0 12 12 0 no gender mismatch no

## 12 2 3 4 8 12 29 17 1 no gender mismatch no

## 13 2 3 5 9 12 29 17 0 no gender mismatch no

## 14 2 3 6 10 12 29 17 0 no gender mismatch no

## 15 2 4 5 11 29 422 393 1 no gender mismatch no

## 16 2 4 6 12 29 422 393 0 no gender mismatch no

## year agecl

## 8 1995-1998 20-40

## 9 1995-1998 20-40

## 10 1995-1998 20-40

## 11 1995-1998 20-40

## 12 1995-1998 20-40

## 13 1995-1998 20-40

## 14 1995-1998 20-40

## 15 1995-1998 20-40

## 16 1995-1998 20-40

Notice that the data is set up so that there is a time-to-event for each permitted transition from a given state. In the msebmt dataset, this variable is time, which measures the time elapsed (in days) from Tstart to Tstop. The dataset also contains a variable named status, which denotes whether the transition is observed (status = 1) or whether it was censored (status = 0). From a given state, only one transition is observed and all others are censored.

For example, patient 2 began in state 1 (Tx) at time 0. From state 1, there are 4 possible transitions: transition 2 (Tx to AE) was observed at time 12 while transition 1 (Tx to Rec), transition 3 (Tx to Rel), and transition 4 (Tx to Death) were censored. The patient remained in state 3 until time 29, when a transition to state 4 (AE and Rec) occurred and the transitions to states 5 (Rel) and 6 (Death) were censored. Patient 2 subsequently remained in state 4 until entering state 5 (Rel) at time 422, at which time transition 12 (AE and Rec to Death) was censored.

**Fitting**

Separate hazard functions, $\lambda\_{rs}(t|Z)$ are estimated for each possible transition from state $r$ to state $s$ as a function of time $t$ and covariates $Z$. In “clock-reset” models, the hazard function depends on elapsed time since entering state $r$, or simply Tstop - Tstart. The hazard functions can be estimated using a joint model with patient and transition interaction terms or by fitting separate models for each transition.

Here we illustrate by fitting 12 transition-specific Weibull models with a yearly time scale. The shape parameter for each model does not depend on covariates whereas the scale parameter depends on four prognostic factors known at baseline: (1) an indicator for whether the donor is a gender mismatch (match), prophylaxis (yes or no) (proph), (3) the year of the transplant (1985-1989, 1990-1994, 1995-1998) (year), and (4) age at transplant in years (agecl).

library("flexsurv")

n\_trans <- max(tmat, na.rm = TRUE)

fits\_wei <- vector(mode = "list", length = n\_trans)

msebmt$years <- msebmt$time/365.25

for (i in 1:n\_trans){

fits\_wei[[i]] <- flexsurvreg(Surv(years, status) ~ match + proph + year + agecl ,

data = subset(msebmt, trans == i),

dist = "weibull")

}

**Simulation**

We first simulate the model using the maximum likelihood estimates of the regression coefficients; that is, we assume that there is no parameter uncertainty. Outcomes will be simulated for patient 2 with the covariate profile displayed below.

pat\_2 <- data.frame(msebmt[msebmt$id == 2,

c("match", "proph", "year", "agecl")][1, ])

head(pat\_2)

## match proph year agecl

## 8 no gender mismatch no 1995-1998 20-40

State occupancy probabilities will be computed from baseline to year 10.

yr\_grid <- seq(0, 10, .1)

**mstate**

Multi-state models can be simulated using mstate::mssample(), which simulates state occupancy probabilities from predicted cumulative hazards. flexsurv can be used to predict cumulative hazards for a given patient (i.e., patient 2) given a covariate profile. When predicting the cumulative hazards, it is critical that the time grid (the t argument) is not too coarse. A time step of .01 is used for the time grid, which (after trial and error) was deemed to be sufficiently accurate — simulations with coarser grids differed significantly from simulations based on continuous time using hesim. Finer grids would further increase accuracy but at the cost of slower simulation times.

cumhaz\_grid <- seq(0, max(msebmt$years), .01)

cumhaz\_pat\_2 <- msfit.flexsurvreg(fits\_wei, trans = tmat,

t = cumhaz\_grid,

newdata = pat\_2,

variance = FALSE)

head(cumhaz\_pat\_2$Haz)

## time Haz trans

## 1 0.00 0.0000000 1

## 2 0.01 0.1136587 1

## 3 0.02 0.1535433 1

## 4 0.03 0.1830811 1

## 5 0.04 0.2074241 1

## 6 0.05 0.2285137 1

tail(cumhaz\_pat\_2$Haz)

## time Haz trans

## 20431 16.97 0.3752974 12

## 20432 16.98 0.3754027 12

## 20433 16.99 0.3755079 12

## 20434 17.00 0.3756132 12

## 20435 17.01 0.3757183 12

## 20436 17.02 0.3758235 12

The mstate::mssample() function works by sampling survival times from each possible transition from the cumulative hazards. More precisely, the cumulative hazards are used to simulate the (discrete) times at which patients transition between health states using the base R function sample(), which is, in turn, used to count the number of patients in each health state at the times specified by the argument tvec.

The function below uses mstate::mssample() to simulate state occupancy probabilities with a “clock-reset” model at the times specified in yr\_grid.

sim\_stprobs\_mstate\_2 <- function(n\_pats){

mstate::mssample(Haz = cumhaz\_pat\_2$Haz,

trans = tmat,

tvec = yr\_grid,

clock = "reset",

M = n\_pats)

}

**hesim**

hesim's approach to simulating multi-state models differs from mstate's in a couple of ways. First, if parametric models are estimated, then hesim samples survival times directly from parametric probability distributions (e.g., the Weibull distribution). This increases accuracy (since no discrete time approximation is required) and speed (since it is considerably faster to sample from probability distributions with known functional forms than by sampling from cumulative hazards with sample()). Second, the simulation code is vectorized (i.e., the simulation code is written in C++ by leveraging [Rcpp](https://cran.r-project.org/web/packages/Rcpp/index.html)) across heterogeneous patients and treatment strategies. In other words, hesim can be used to quickly simulate multiple covariate profiles and treatment alternatives.

We set up input data for the simulation by creating a dataset of many identical patients each with the covariate profile of patient 2 and a single treatment strategy (i.e., bone marrow transplantation). An example dataset of 1,000 identical patients is displayed. (Note that hesim uses [data.table](https://github.com/Rdatatable/data.table/wiki) to increase speed.)

library("hesim")

create\_input\_data\_2 <- function(n\_pats){

# Patients

patients <- pat\_2[rep(1, n\_pats), ]

patients$patient\_id <- 1:n\_pats

# Treatment strategies

strategies <- data.frame(strategy\_id = 1)

# Input data

hesim\_dat <- hesim\_data(strategies = strategies,

patients = patients)

input\_data <- hesim::expand(hesim\_dat, by = c("strategies", "patients"))

return(input\_data[, ])

}

create\_input\_data\_2(n\_pats = 1000)

## strategy\_id patient\_id match proph year agecl

## 1: 1 1 no gender mismatch no 1995-1998 20-40

## 2: 1 2 no gender mismatch no 1995-1998 20-40

## 3: 1 3 no gender mismatch no 1995-1998 20-40

## 4: 1 4 no gender mismatch no 1995-1998 20-40

## 5: 1 5 no gender mismatch no 1995-1998 20-40

## ---

## 996: 1 996 no gender mismatch no 1995-1998 20-40

## 997: 1 997 no gender mismatch no 1995-1998 20-40

## 998: 1 998 no gender mismatch no 1995-1998 20-40

## 999: 1 999 no gender mismatch no 1995-1998 20-40

## 1000: 1 1000 no gender mismatch no 1995-1998 20-40

hesim combines the input data with the model fit from flexsurvreg() to set up a disease model—specifically, an individual-level continuous time state transition model (CTSTM). Similar to mstate::mssample, the $sim\_stateprobs() function simulates state occupancy probabilities by first simulating a trajectory through the multi-state model for each patient and treatment strategy combination and then counting the number of simulated patients (for each treatment strategy) in each state over time.

sim\_stprobs\_hesim\_2 <- function(n\_pats, fits, point\_estimate = TRUE,

n\_samples = 1000){

input\_dat\_2 <- create\_input\_data\_2(n\_pats)

dismod <- create\_IndivCtstmTrans(hesim::flexsurvreg\_list(fits),

input\_dat\_2,

trans\_mat = tmat,

clock = "reset",

point\_estimate = point\_estimate,

n = n\_samples)

return(dismod$sim\_stateprobs(t = yr\_grid))

}

**Comparison**

We simulate state occupancy probabilities using both mstate and hesim and vary the number of simulated patients to examine the impact on precision.

As shown in the plot below, the differences in state occupancy probabilities generally become smaller as the number of simulated patients increases. Further, (although not shown) the simulation results from mstate become increasingly close to hesim as the time grid becomes finer.

library("ggplot2")

stprobs2[, state\_name := factor(state\_id, levels = 1:6,

labels = colnames(tmat))]

n\_pats\_levels <- paste0("Patients = ", unique(stprobs2$n\_pats))

stprobs2[, n\_pats\_name := factor(n\_pats, labels = n\_pats\_levels)]

ggplot(stprobs2, aes(x = time, y = prob, col = lab)) +

geom\_line() +

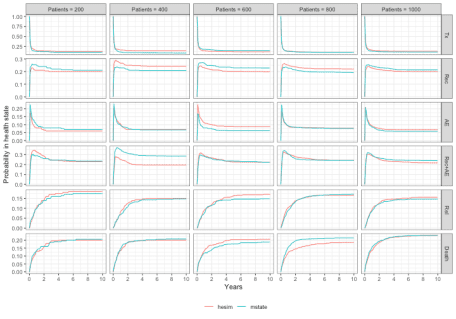
facet\_grid(state\_name ~ n\_pats\_name, scales = "free\_y") +

xlab("Years") + ylab("Probability in health state") +

scale\_x\_continuous(breaks = seq(0, max(yr\_grid), 2)) +

scale\_color\_discrete(name = "") + theme\_bw() +

theme(legend.position = "bottom")



Since 1,000 iterations generate reasonably accurate estimates, we will assess speed by simulating 1,000 patients.

time\_mstate <- system.time(sim\_stprobs\_mstate\_2(1000))

time\_hesim <- system.time(sim\_stprobs\_hesim\_2(1000, fits\_wei))

print(time\_mstate)

## user system elapsed

## 19.759 2.674 24.842

print(time\_hesim)

## user system elapsed

## 0.036 0.000 0.037

The elapsed times (in seconds) suggest that hesim is 671.41 faster than mstate when simulating a parametric multi-state model.

**Probabilistic sensitivity analysis**

The results presented so far have ignored the impact of parameter uncertainty. In contrast, probabilistic sensitivity analysis (PSA) propagates uncertainty in the parameters to the state occupancy probabilities. In our case, the regression coefficients from the multi-state model are drawn from a suitable probability distribution and the multi-state simulation is run for each draw of the coefficients. There are a number of ways to simulate the distribution of the coefficients including bootstrapping, Bayesian modeling, and [asymptotic Monte Carlo approximation](https://www.tandfonline.com/doi/abs/10.1080/00031305.2013.783880). We will take the latter approach by sampling the maximum likelihood estimates from an asymptotic multivariate distribution, which is the fastest option.

Although sampling from a multivariate normal distribution is not computationally intensive, repeatedly rerunning the simulation for each draw of the regression coefficients is. In this section, we show how to perform PSA using both mstate and hesim and compare performance.

**mstate**

PSA can be performed using mstate::mssample() by looping through a distribution of cumulative hazards for a covariate profile and running mstate::mssample() for each iteration of the loop. The most straightforward way to predict a distribution of cumulative hazards is with the CTSTMs from the hesim package. The $cumhazard() function predicts cumulative hazards by transition number, parameter sample, treatment strategy, patient, and time. In our example, we predict cumulative hazards for a single patient (id = 2) and treatment strategy.

# n\_samples = # of PSA iterations

predict\_cumhaz\_dist\_2 <- function(n\_samples){

input\_dat\_2 <- create\_input\_data\_2(n\_pats = 1)

dismod <- create\_IndivCtstmTrans(hesim::flexsurvreg\_list(fits\_wei),

input\_dat\_2,

trans\_mat = tmat,

clock = "reset",

n = n\_samples,

point\_estimate = FALSE)

cumhaz\_pat2\_dist <- dismod$cumhazard(t = cumhaz\_grid)

setnames(cumhaz\_pat2\_dist, c("t", "cumhazard"), c("time", "Haz"))

return(cumhaz\_pat2\_dist)

}

predict\_cumhaz\_dist\_2(n\_samples = 3)

## trans sample strategy\_id patient\_id time Haz

## 1: 1 1 1 1 0.00 0.0000000

## 2: 1 1 1 1 0.01 0.1106806

## 3: 1 1 1 1 0.02 0.1472472

## 4: 1 1 1 1 0.03 0.1740076

## 5: 1 1 1 1 0.04 0.1958947

## ---

## 61304: 12 3 1 1 16.98 0.3965328

## 61305: 12 3 1 1 16.99 0.3966496

## 61306: 12 3 1 1 17.00 0.3967664

## 61307: 12 3 1 1 17.01 0.3968832

## 61308: 12 3 1 1 17.02 0.3969999

A distribution of simulated state occupancy probabilities for patient 2 can then be simulated by looping over the cumulative hazards for each parameter sample.

psa\_stprobs\_mstate\_2 <- function(n\_pats, n\_samples){

cumhaz\_dist\_2 <- predict\_cumhaz\_dist\_2(n\_samples)

stprobs\_mstate\_2 <- vector(mode = "list", length = n\_samples)

for (s in 1:n\_samples){

cumhaz\_s <- cumhaz\_dist\_2[sample == s]

stprobs\_mstate\_2[[s]] <- sim\_stprobs\_mstate\_2(n\_pats)

stprobs\_mstate\_2[[s]]$sample <- s

}

stprobs\_mstate\_2 <- rbindlist(stprobs\_mstate\_2)

return(stprobs\_mstate\_2)

}

**hesim**

With hesim, the entire analysis is inherently Bayesian so PSA is seamless. When creating a CTSTM from a flexsurvreg object, the user must simply set the argument point\_estimate = FALSE and choose the number of samples of the parameters to draw. The distribution of the regression coefficients is then drawn by sampling from the multivariate normal distribution. Furthermore, the PSA is vectorized since the loops over the parameter samples are written in C++.

**Comparison**

In our comparisons, we continue to simulate 1,000 patients. With hesim, we consider both 100 and 1,000 parameter samples; with mstate we restrict the analysis to 100 parameter samples because it becomes increasingly slow as the number of parameter samples increases.

time\_mstate <- system.time(psa\_stprobs\_mstate\_2(n\_pats = 1000, n\_samples = 100))

time\_hesim\_100 <- system.time(sim\_stprobs\_hesim\_2(n\_pats = 1000, fits\_wei,

point\_estimate = FALSE,

n\_samples = 100))

time\_hesim\_1000 <- system.time(sim\_stprobs\_hesim\_2(n\_pats = 1000, fits\_wei,

point\_estimate = FALSE,

n\_samples = 1000))

print(time\_mstate)

## user system elapsed

## 1764.489 270.389 2038.770

print(time\_hesim\_100)

## user system elapsed

## 0.403 0.037 0.440

print(time\_hesim\_1000)

## user system elapsed

## 4.665 0.422 5.093

hesim is fast, even when performing a PSA. With 1,000 patients, 100 PSA iterations can be simulated in less than a second and 1,000 PSA iterations in approximately 5 seconds. Computational efficiency becomes increasingly important as the computational demands grow: with 1,000 patients and 100 PSA iterations, mstate takes around 34 minutes to run and hesim is 4633.57 times faster.

**Conclusion**

In this post, we describe some of the R packages that facilitate multi-state modeling. The flexsurv package is particularly useful for estimating parametric models. When “clock-reset” models are fit, state occupancy probabilities can only be predicted for general multi-state models using simulation. Both the hesim and mstate packages provide functionality for running such simulations. However, when parametric models are fit, hesim is considerably faster and this computational advantage grows with the number of required iterations. The speed advantage may be particularly useful when running a PSA, multiple subgroup analyses, and/or simulations using competing survival distributions.