Simulating time-to-event data from parametric distributions, custom distributions, competings risk models and general multi-state models

Michael J. Crowther

University of Leicester and Karolinska Institutet

michael.crowther@le.ac.uk

Abstract

In this paper I describe some substantial extensions to the `survsim` command for simulating survival data. `survsim` can now simulate survival data from a parametric distribution, a custom/user-defined distribution, from a fitted `merlin` model, from a specified cause-specific hazards competing risks model, or from a specified general multi-state model. I illustrate the command with some examples from each setting, demonstrating the huge flexibilty that can be used to better evaluate statistical methods.

March 28, 2020

1 Introduction

`survsim` was first introduced with the ability to simulate from a defined parametric distribution, including the exponential, Weibull, Gompertz and 2-component versions of them. It allowed both time-independent and time-dependent effects, and allowed simulation of competings risks data, providing a useful tool to generate event times (Crowther and Lambert, 2012). Following this, Crowther and Lambert (2013) developed a general algorithm to simulate event times from a custom, user-defined hazard or cumulative hazard function, using a combination of root-finding and nested numerical integration.

In this paper, I introduce some substantial extensions, including:

1. The ability to simulate from a competing risks or general multi-state model. Event times can be simulated from transition-specific hazards, where each transition hazard function can be a standard parametric distribution, or a user-defined hazard function. Covariates and time-dependent effects can be specified for each transition-specific hazard independently, and multiple timescales can be incorporated. The simulation method utilises the competing risks simulation method described by Beyersmann et al. (2009), and the general algorithm of Crowther and Lambert (2013).
2. The ability to simulate from a conditional distribution, i.e. allowing for delayed entry/left truncation.
3. The ability to simulate directly from a fitted `merlin` survival model (Crowther, (To appear).

The paper is arranged as follows. In Section 2 I briefly describe the core algorithm used to simulate event times in `survsim`. In Section 3, I describe the syntax for the four core settings that can be used to generate event times, which are then illustrated in Section 4. I conclude in Section 5 with a brief discussion.

2 Simulating survival times

Simulating data from a defined distribution can be simple, or can be extremely complex. Assume we have a random variable T, with associated cumulative distribution function, $F(T)$. To simulate survival times from such a distribution, we can simply let

$$F \sim U(0,1)$$

To simulate an observation, we draw from the uniform distribution, say $u \sim U(0,1)$, and simply substitute and solve for t,

$$F(t) = u$$

and hence,

$$t = F^{-1}(u)$$

Now solving for t relies on being able to invert the cumulative distribution function, and since it is a function of the cumulative hazard function, we must be able to integrate our hazard function. To accommodate these challenges, Crowther and Lambert (2013) dveeloped a combined root-finding and numerical integration algorithm to provide an efficient method of generating event times from arbitrary distribution functions. This forms the engine of the developments in this paper. For the competing risks and more general multi-state setting, this is used to simulate from the total hazard function (made up of the sum of cause/transition-specific hazard functions leaving a particular state), as developed by Beyersmann et al. (2009) for competing risks.

3 The `survsim` command

Simulate survival data from a parametric distribution, a user-defined distribution, from a fitted merlin model, from a cause-specific hazards competing risks model, or from a general multi-state model.

3.1 Syntax - simulate survival data from a parametric distribution

`survsim` *newvarname1* [*newvarname2*] , `distribution(string)` [, *options*]

3.1.1 Options

`noconstant` suppresses the constant (intercept) term and may be specified for the fixed effects equation and for the random effects equations.

`distribution(string)` specifies the parametric survival distribution to use, including `exponential`, `gompertz` or `weibull`.

`lambdas(numlist)` defines the scale parameters in the exponential/Weibull/Gompertz distribution(s). The number of values required depends on the model choice. Default is a single number corresponding to a standard parametric distribution. Under a mixture model, 2 values are required.

`gammas(numlist)` defines the shape parameters of the Weibull/Gompertz parametric distribution(s). Number of entries must be equal to that of `lambdas()`.

`covariates(varname # ...)` defines baseline covariates to be included in the linear predictor of the survival model, along with the value of the corresponding coefficient. For example, a treatent variable coded 0/1 can be included, with a log hazard ratio of 0.5, by `covariates(treat 0.5)`. Variable `treat` must be in the dataset before `survsim` is called.

`tde(varname # ...)` creates non-proportional hazards by interacting covariates with log time for an exponential or Weibull model, or time under a Gompertz model or mixture model. Values should be entered as `tde(trt 0.5)`, for example. Multiple time-dependent effects can be specified, but they will all be interacted with the same function of time.

`maxtime(#|varname)` specifies the right censoring time(s). Either a common maximum follow-up time `#` can be specified for all observations, or observation specific censoring times can be specified by using a `varname`.

`ltruncated(#|varname)` specifies the left truncated/delayed entry time(s), to simulate from a conditional survival distribution. Either a common time `#` can be specified for all observations, or observation specific left truncation times can be specified by using a `varname`.

`mixture` specifies that survival times are simulated from a 2-component mixture model, with mixture component distributions defined by `distribution()`. `lambdas()` and `gammas()` must be of length 2.

`pmix(#)` defines the mixture parameter. Default is 0.5.

`nodes(#)` defines the number of Gauss-Legendre quadrature points used to evaluate the cumulative hazard function when `mixture` and `tde()` are specified together. To simulate survival times from such a mixture model, a combination of numerical integration and root-finding is used. The default is `nodes(30)`.

3.2 Syntax for simulating survival times from a user-defined distribution

`survsim` *newvarname1 newvarname2* `, maxtime(#|varname)` [`,` *options*]

3.2.1 Options

`loghazard(string)` is the user-defined log hazard function. The function can include:

- `{t}` which denotes the main timescale, measured on the time since starting state, `startstate()`, timescale (which may be `ltruncated()`)
- `varname` which denotes a variable in your dataset
- `+-*/^` standard Mata mathematical operators, using colon notation i.e. `2 :* {t}`, see `help [M-2] op_colon`. Colon operators must be used as behind the scenes, `{t}` gets replaced by an `_N` by `nodes()` matrix when numerically integrating the hazard function.
- `mata_function()` any Mata function, e.g. `log()` and `exp()`

`hazard(string)` is the user-defined baseline hazard function. See `loghazard()` for more details, and examples below.

`logchazard(string)` is the user-defined log cumulative baseline hazard function. See `loghazard()` for more details, and examples below.

`chazard(string)` is the user-defined baseline cumulative hazard function. See `loghazard()` for more details, and examples below.

`covariates(varname # ...)` defines baseline covariates to be included in the linear predictor of the survival model, along with the value of the corresponding coefficient. For example, a treatent variable coded 0/1 can be included, with a log hazard ratio of 0.5, by `covariates(treat 0.5)`. Variable treat must be in the dataset before survsim is called. If `chazard()` or `logchazard()` are used, then `covariates()` effects are additive on the log cumulative hazard scale.

`tde(varname # ...)` creates non-proportional hazards by interacting covariates with a function of time, defined by tdefunction(), on the appropriate log hazard or log cumulative hazard scale. Values should be entered as `tde(trt 0.5)`, for example. Multiple time-dependent effects can be specified, but they will all be interacted with the same `tdefunction()`. To circumvent this, you can directly specify them in your user function.

`tdefunction(string)` defines the function of time to which covariates specified in `tde()` are interacted with, to create time-dependent effects. The default is {t}, i.e. linear time. The function can include:

- `{t}` which denotes the main timescale, measured on the time since starting state, `startstate()`, timescale (which may be `ltruncated()`)
- `+-*/^` standard Mata mathematical operators, using colon notation i.e. `2 :* {t}`, see `help [M-2] op_colon`. Colon operators must be used as behind the scenes, `{t}` gets replaced by an _N x nodes() matrix when numerically integrating the hazard function. mata_function any Mata function, e.g. log() and exp()

`maxtime(#|varname)` specifies the right censoring time(s). Either a common maximum follow-up time # can be specified for all observations, or observation specific censoring times can be specified by using a `varname`.

`ltruncated(#|varname)` specifies the left truncated/delayed entry time(s). Either a common time `#` can be specified for all observations, or observation specific left truncation times can be specified by using a `varname`.

`nodes(#)` defines the number of Gauss-Legendre quadrature points used to evaluate the cumulative hazard function when `loghazard()` or `hazard()` is specified. To simulate survival times from such a function, a combination of numerical integration and root-finding is used. The default is `nodes(30)`.

3.3 Syntax for simulating survival times from a fitted merlin survival model

`survsim` *newvarname*`1` *newvarname*`2 , model(name) maxtime(#|varname)`

3.3.1 Options

`model(name)` specifies the name of the `estimates store` object containing the estimates of the model fitted. The survival must be estimated using the merlin command. `survsim` will simulate from the fitted model, using covariate values that are in your current dataset. For example,

```
. merlin (_t trt , family(weibull, failure(_d)))
. estimates store m1
. survsim stime died, model(m1) maxtime(10)
```

`maxtime(#|varname)` specifies the right censoring time(s). Either a common maximum follow-up time `#` can be specified for all observations, or observation specific censoring times can be specified by using a `varname`.

3.4 Syntax for simulating survival times from a mult-state model

`survsim` *timestub statestub eventstub* `, hazard1(haz_options) hazard2(haz_options) maxtime(#|varname)` [`, transmatrix(name) hazard3(haz_options)` *options*]

3.4.1 Options

`distribution(string)` specifies the parametric survival distribution to use, including exponential, gompertz or weibull.

`lambda(#)` defines the scale parameter for a exponential/Weibull/Gompertz distribution.

`gamma(#)` defines the shape parameter for a Weibull/Gompertz parametric distribution.

`user(function)` defines a custom hazard function. The function can include:

- `{t}` which denotes the main timescale, measured on the time since starting state, `startstate()`, timescale (which may be `ltruncated()`)
- `{t0}` which denotes the time of entry to the current state of the associated transition hazard, measured on the time since initial starting state timescale
- `varname` which denotes a variable in your dataset
- `+-*/^` standard Mata mathematical operators, using colon notation i.e. `2 :* {t}`, see `help [M-2] op_colon`. Colon operators must be used as behind the scenes, `{t}` gets replaced by an `_N` by `nodes()` matrix when numerically integrating the transition hazard function.
- `mata_function` any Mata function, e.g. `log()` and `exp()`

For example,

```
dist(weibull) lambda(0.1) gamma(1.2)
```

is equivalent to

```
user(0.1:*1.2:*{t}:^(1.2:-1))
```

`covariates(varname # ...)` defines baseline covariates to be included in the linear predictor of the transition-specific hazard function, along with the value of the corresponding coefficient. For example, a treatent variable coded 0/1 can be included, with a log hazard ratio of 0.5, by `covariates(treat 0.5)`. Variable treat must be in the dataset before `survsim` is called.

`tde(varname # ...)` creates non-proportional hazards by interacting covariates with a function of time. Covariates are interacted with `tdefunction()`, on the log hazard scale. Values should be entered as `tde(trt 0.5)`, for example. Multiple time-dependent effects can be specified, but they will all be interacted with the same function of time.

`tdefunction(string)` defines the function of time to which covariates specified in `tde()` are interacted with, to create time-dependent effects in the transition-specific hazard function. The default is `{t}`, i.e. linear time. The function can include:

- `{t}` which denotes the main timescale, measured on the time since starting state, startstate(), timescale (which may be ltruncated())
- `+-*/^` standard Mata mathematical operators, using colon notation i.e. `2 :* {t}`, see `help [M-2] op_colon`. Colon operators must be used as behind the scenes, `{t}` gets replaced by an `_N` by `nodes()` matrix when numerically integrating the transition hazard function.
- `mata_function` any Mata function, e.g. `log()` and `exp()`

`reset` specifies that this transition model is on a clock-reset timescale. The timescale is reset to 0 on entry, i.e. the timescale for this transition is measured on a time since state entry timescale, rather than the default clock-forward. If you specify a `user()` function with reset, then survsim will replace any occurences of `{t}` with `{t}-{t0}`, including those specified in `tdefunction()`.

`transmatrix(matname)` specifies the transition matrix which governs the multi-state model. Transitions must be numbered as an increasing sequence of integers from 1,...,K, from left to right, top to bottom of the matrix. Reversible transitions are allowed. If `transmatrix()` is not specified, a competing risks model is assumed.

`maxtime(#|varname)` specifies right censoring time(s). Either a common maximum follow-up time `#` can be specified for all observations, or observation specific censoring times can be specified by using a `varname`.

`startstate(#|varname)` specifies the state(s) in which observations begin. Either a common state `#` can be specified for all observations, or observation specific starting states can be specified by using a `varname`. Default is `startstate(1)`.

`ltruncated(#|varname)` specifies left truncated/delayed entry time(s), which is the time(s) at which observations start in the initial starting state(s). Either a common time `#` can be specified for all observations, or observation specific left truncation times can be specified by using a varname. Default is `ltruncated(0)`.

`nodes(#)` defines the number of Gauss-Legendre quadrature points used to evaluate the total cumulative hazard function for each potential next transition. To simulate survival times from such a function, a combination of numerical integration and root-finding is used. The default is `nodes(30)`.

4 Examples

In this section I will go through at least one example of how to use `survsim` to simulate survival data from each of the four main settings.

4.1 Simulating survival times from standard parametric distributions

Let's simulate survival times from a Weibull distribution, with a binary treatment group, `trt`, and a continuous covariate, `age`, under proportional hazards:

$$h(t) = \lambda\gamma t^{\gamma-1}\exp(\text{trt}\beta_1 + \text{age}\beta_2)$$

I'll simulate 300 observations, and pick some distributions for the covariates, which should be self-explanatory,

```
. clear

. set obs 300
number of observations (_N) was 0, now 300

. set seed 134987

. gen trt = runiform()>0.5

. gen age = rnormal(50,3)
```

We then call `survsim`, setting $\lambda = 0.1$, $\gamma = 1.2$, $\beta_1 = -0.5$ and $\beta_2 = 0.01$,

```
. survsim stime, distribution(weibull) lambda(0.1) gamma(1.2)  ///
>                covariates(trt -0.5 age 0.01)
```

which stores our simulated survival times in the new variable `stime`. If we wanted to apply right-censoring, we could first generate some censoring times, we could apply a common censoring time using for example `maxtime(5)`, which would censor all observations if their simulated event times was greater than 5, or we could generate observation specific potential censoring times, such as

```
. gen censtime = runiform() * 5
```

and now add the `maxtime()` option to `survsim`, remembering to also specify a second new variable name for the event indicator,

```
. survsim stime2 died2, distribution(weibull) lambda(0.1) gamma(1.2)     ///
>                       covariates(trt -0.5 age 0.01) maxtime(censtime)
```

We could also:

- add time-dependent effects using the `tde()` option
- add left-truncation/delayed entry using the `ltruncated()` option
- simulate from a 2-component mixture distribition using the `mixture` option

4.2 Simulating survival times from a user-defined (log) (cumulative) hazard function

This section illustrates how to simulate from a user-defined function, first described in Crowther and Lambert (2013). Let's start by simulating 500 observations, and generate a binary treatment group,

```
. clear

. set obs 500
number of observations (_N) was 0, now 500

. set seed 134987

. gen trt = runiform()>0.5
```

The most flexible form of simulating survival data with `survsim` is by specifying a custom hazard or cumulative hazard function, such as:

$$h(t) = h_0(t)\exp(\text{trt}\beta_1)$$

where

$$h_0(t) = \exp(-1 + 0.02t - 0.03t^2 + 0.005t^3)$$

which can be done, on the `loghazard()` scale for simplicity, using

```
. survsim stime1 died1, loghazard(-1:+0.02:*{t}:-0.03:*{t}:^2:+0.005:*{t}:^3)  ///
>                       covariates(trt -0.5) maxtime(1.5)
Warning: 321 survival times were above the upper limit of maxtime()
         They have been set to maxtime()
         You can identify them by _survsim_rc = 3
```

The `loghazard()` function is defined using Mata code, with colon operators representing element by element operations. Time must be referred to using the `{t}` notation. A common right censoring time of 1.5 years is specified using `maxtime(1.5)`. We could make the treatment effect diminish over log time by incorporating a time-dependent effect, where

$$\beta_1(t) = \log(t)\beta_1$$

which is defined using the `tdefunction()` and `tde()` options, setting $\beta_1 = 0.03$

```
. survsim stime2 died2, loghazard(-1:+0.02:*{t}:-0.03:*{t}:^2:+0.005:*{t}:^3)    ///
>                       covariates(trt -0.5) tde(trt 0.03) tdefunction(log({t})) ///
>                       maxtime(1.5)
Warning: 328 survival times were above the upper limit of maxtime()
         They have been set to maxtime()
         You can identify them by _survsim_rc = 3
```

which will form an interaction between `trt`, its coefficient `0.03` and log time. Alternatively, we could instead simulate from a model on the cumulative hazard scale, using the `logchazard()` option instead.

4.3 Simulating survival times from a fitted `merlin` survival model

Rather than simulating from a particular data-generating model specified essentially by hand, we can directly simulate from a fitted model, by passing an `estimates` object to `survsim` through the `model()` option. This is similar to `stsurvsim` [Roystonstsurvsim] which allows the simulation of survival times from a Royston-Parmar flexible parametric model. `survsim` now allows the simulation from survival model that has been fitted with the `merlin` command. Let's fit a Weibull model to a standard survival dataset:

```
. webuse brcancer, clear
(German breast cancer data)

. stset rectime, f(censrec=1) scale(365)

     failure event:  censrec == 1
obs. time interval:  (0, rectime]
 exit on or before:  failure
    t for analysis:  time/365

------------------------------------------------------------------------------
        686  total observations
          0  exclusions
------------------------------------------------------------------------------
        686  observations remaining, representing
        299  failures in single-record/single-failure data
  2,113.425  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
```

```
                                          last observed exit t =  7.284932

. merlin (_t hormon , family(weibull, failure(_d)))

Fitting full model:

Iteration 0:   log likelihood = -2113.4247
Iteration 1:   log likelihood =  -898.2654
Iteration 2:   log likelihood = -868.11925
Iteration 3:   log likelihood =  -868.0269
Iteration 4:   log likelihood = -868.02684

Mixed effects regression model                  Number of obs     =        686
Log likelihood = -868.02684
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
_t:          |
      hormon |  -.3932405   .1248267    -3.15   0.002    -.6378962   -.1485847
       _cons |  -2.196012   .1094092   -20.07   0.000     -2.41045   -1.981574
  log(gamma) |   .2509974   .0496958     5.05   0.000     .1535953    .3483994
------------------------------------------------------------------------------
```

I'm using the benefits of `stset` to declare the survival variables, which I can use directly within `merlin` for simplicity. We then simply store the model object, calling it whatver we like, such as the imaginative name of `m1`,

```
. estimates store m1
```

This we then pass to `survsim` to simulate a dataset, of the same size, using our fitted results,

```
. survsim stime5 died5, model(m1) maxtime(7)
```

The option `maxtime()` is required in this case. We can then fit the same model as before, and of course get slightly different results, because we have sampling variability.

```
. stset stime5, failure(died5)

     failure event:  died5 != 0 & died5 < .
obs. time interval:  (0, stime5]
 exit on or before:  failure

------------------------------------------------------------------------------
        686  total observations
          0  exclusions
------------------------------------------------------------------------------
        686  observations remaining, representing
        447  failures in single-record/single-failure data
  3,166.854  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
```

```
                                          last observed exit t =         7
```

```
. merlin (_t hormon , family(weibull, failure(_d)))

Fitting full model:

Iteration 0:   log likelihood = -3166.8536
Iteration 1:   log likelihood = -1345.4701
Iteration 2:   log likelihood =  -1299.206
Iteration 3:   log likelihood = -1298.8074
Iteration 4:   log likelihood = -1298.8059
Iteration 5:   log likelihood = -1298.8059

Mixed effects regression model                  Number of obs     =        686
Log likelihood = -1298.8059
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
_t:          |
      hormon |  -.3533907   .1014646    -3.48   0.000    -.5522576   -.1545238
       _cons |  -2.349859   .1100583   -21.35   0.000    -2.565569   -2.134148
  log(gamma) |   .2650626   .0421563     6.29   0.000     .1824377    .3476875
------------------------------------------------------------------------------
```

Note, `survsim` will simulate the same number of observations as `_N`, so any covariates in your model must have non-missing observations, otherwise missing values will be produced. The useful aspect of this, is that you can manipulate your covariate distributions in your dataset, and then simply recall `survsim` to simulate new survival times using the same estimated parameters.

Currently, `survsim` can simulate from any of the available survival models in `merlin`, but does not support simuation from multivariate models or a model containing random effects.

4.4 Simulating competing risks data from specified cause-specific hazard functions

`survsim` simulates competing risk data using the cause-specific hazard setting described by Beyersmann et al. (2009), utilising the general simulation algorithm described by Crowther and Lambert (2013). Let's simulate from a competing risk model with 2 competing events. The first cause-specific hazard has a Weibull distribution, with no covariates. The second cause-specific hazard model has an exponential distribution, with a beneficial treatment effect. Right censoring is applied at 10 years.

```
. clear

. set seed 398

. set obs 1000
number of observations (_N) was 0, now 1,000
```

```
. gen trt = runiform()>0.5

. survsim time state event , hazard1(dist(weibull) lambda(0.1) gamma(0.8))  ///
>                            hazard2(dist(exponential) lambda(0.02)         ///
>                            covariates(trt -0.5)) maxtime(10)
variables time0 to time1 created
variables state0 to state1 created
variables event1 to event1 created
```

Each `hazard#()` defines a cause-specific hazard function, with specified `distribution()` and associated baseline parameters, covariate effects and time-depdent effects. Each of the `hazard#()` functions can be as similar, or different, as required. `survsim` creates some new variables, based on the *newvarstubs* that we specified in the call.

```
. list if _n<=5
```

	trt	time0	state0	time1	state1	event1
1.	0	0	1	4.5792847	2	1
2.	1	0	1	10	1	0
3.	1	0	1	10	1	0
4.	1	0	1	2.8415219	3	1
5.	0	0	1	1.576534	2	1

Since the competing risks simulation is framed within a more general multi-state setting, discussed in the next example, all observations are assumed to begin in an initial starting state 1, at an initial starting time 0. These are stored in `state0` and `time0`, respectively. The starting time can be changed using the `ltruncated()` option.

From the starting state, observations have two places to go:

- State 1 to State 2, with the transition rate governed by `hazard1()`
- State 1 to State 3, with the transition rate governed by `hazard2()`

We can see which events occurred with

```
. tabulate state1 event1
```

state1 \ event1	0	1	Total
1	484	0	484
2	0	414	414
3	0	102	102
Total	484	516	1,000

which shows that at by ten years, 484 observations were right-censored, 414 are in State 2, and 102 are in State 3.

Now let's simulate from a competing risk model with 3 competing events. The first cause-specific hazard has a user defined baseline hazard function, with a harmful treatment effect. The second cause-specific hazard model has a Weibull distribition, with a beneficial treatment effect. The third cause-specific hazard has a user-defined baseline hazard function, with an initially beneficial treatment effect that reduces linearly with respect to log time. Right censoring is applied at 3 years. Phew.

```
. cap drop time* state* event*

. set seed 32984575

. survsim time state event,                                        ///
>         hazard1(user(exp(-2 :+ 0.2:* log({t}) :+ 0.1:*{t}))  ///
>                 covariates(trt 0.1))                             ///
>         hazard2(dist(weibull) lambda(0.01) gamma(1.3)        ///
>                 covariates(trt -0.5))                            ///
>         hazard3(user(0.1 :* {t} :^ 1.5) covariates(trt -0.5) ///
>                 tde(trt 0.1) tdefunction(log({t})))          ///
>         maxtime(3)
variables time0 to time1 created
variables state0 to state1 created
variables event1 to event1 created

. tabulate state1 event1

           |        event1
    state1 |         0          1 |     Total
-----------+----------------------+----------
         1 |       341          0 |       341
         2 |         0        345 |       345
         3 |         0         30 |        30
         4 |         0        284 |       284
-----------+----------------------+----------
     Total |       341        659 |     1,000
```

You can see that this can get as complex as necessary. I currently let you use up to 50 cause-specific hazards, just in case you're feeling particularly adventurous.

4.5 Simulating from an illness-death model

We first define the transition matrix for an illness-death model. It has three states:

- State 1 - A "healthy" state. Observations can move from state 1 to state 2 or 3.
- State 2 - An intermediate "illness" state. Observations can come from state 1, and move on to state 3.
- State 3 - An absorbing "death" state. Observations can come from state 1 or 2, but not leave.

This gives us three potential transitions between states:

- Transition 1 - State 1 -> State 2
- Transition 2 - State 1 -> State 3
- Transition 3 - State 2 -> State 3

which is defined by the following matrix:

```
. matrix tmat = (.,1,2\.,.,3\.,.,.)
```

The key is to think of the column/row numbers as the states, and the elements of the matrix as the transition numbers. Any transitions indexed with a missing value `.` means that the transition between the row state and the column state is not possible. Let's make it obvious, sticking with our "healthy", "ill" and "dead" names for the states:

```
. mat colnames tmat = "healthy" "ill" "dead"

. mat rownames tmat = "healthy" "ill" "dead"

. mat list tmat

tmat[3,3]
         healthy      ill     dead
healthy        .        1        2
    ill        .        .        3
   dead        .        .        .
```

Now we've defined the transition matrix, we can use `survsim` to simulate some data. We'll simulate 1000 observations, and generate a binary treatment group indicator, remembering to `set seed` first.

```
. clear

. set obs 1000
number of observations (_N) was 0, now 1,000

. set seed 9865

. gen trt = runiform()>0.5
```

The first transition-specific hazard has a user defined baseline hazard function, with a harmful treatment effect. The second transition-specific hazard model has a Weibull distribition, with a beneficial treatment effect. The third transition-specific hazard has a user-defined baseline hazard function, with an initially beneficial treatment effect that reduces linearly with respect to log time. Right censoring is applied at 3 years.

```
. survsim time state event, transmatrix(tmat)                   ///
>         hazard1(user(exp(-2 :+ 0.2:* log({t}) :+ 0.1:*{t}))  ///
>                 covariates(trt 0.1))                          ///
>         hazard2(dist(weibull) lambda(0.01) gamma(1.3)         ///
>                 covariates(trt -0.5))                         ///
>         hazard3(user(0.1 :* {t} :^ 1.5) covariates(trt -0.5) ///
>                 tde(trt 0.1) tdefunction(log({t})))           ///
>         maxtime(3)
variables time0 to time2 created
```

```
variables state0 to state2 created
variables event1 to event2 created
```

The hazard number # in each `hazard#()`, represents the transition number in the transiton matrix. Simple as that. `survsim` has created variables storing the times at which states were entered, with the associated state number and the associated event indicator. It begins by creating the `0` variables, which represents the time at which observatations entered the inital state, `time0`, and the associated state number, `state0`. As `ltruncated()` and `startstate()` were not specified, all observations are assumed to start in state 1 at time 0. Subsequent transitions are simulated until all observations have either entered an absorbing state, or are right-censored at their `maxtime()`. For simplicity, I will assume time is measured in years. We can see what `survsim` has created:

```
. list if inlist(_n,1,4,16,112), compress
```

	trt	time0	sta~0	time1	sta~1	eve~1	time2	sta~2	eve~2
1.	0	0	1	3	1	0	.	.	.
4.	1	0	1	.95636156	2	1	3	2	0
16.	0	0	1	1.0755764	2	1	2.4401409	3	1
112.	1	0	1	2.3290322	3	1	.	.	.

All observations start initially in state 1 at time 0, which are stored in `state0` and `time0`, respectively. Then,

- Observation 1 is right-censored at 3 years, remaining in state 1
- Observation 4 moves to state 2 at 0.956 years, and is subsequently right-censored at 3 years, still in state 2
- Observation 16 moves to state 2 at 1.076 years, and then moves to state 3 at 2.440 years. Since state 3 is an absorbing state, there are no further transitions
- Observation 112 moves to state 3 at 2.329 years. Again, since state 3 is absorbing, there are no further transitions

There's a variety of extensions we could incorporate, for example, we could simulate from a semi-Markov model by using the `reset` option in `hazard3()`, which would reset the clock when State 2 is entered. The simulated event times that `survsim` returns will still be calculated on the main timescale in this case, time since initial `startstate()`. We could of course have a much more complex multi-state structure, i.e. more states or reversible transitions. Both of these are supported by `survsim`.

4.6 Simulating from an illness-death model with multiple timescales

A further capability of `survsim` is to simulate from an event time model with *multiple* timescales. Such a setting is rarely considered, as both the estimation of such a model is computationally challenging, let alone simulating from one. However, `survsim` now provides this, in extremely general way, and indeed `merlin` has the capability of fitting such models.

Continuing with the illness-death model, the transition between the illness and death state may depend not only on time since the initial healthy state entry, but also on the time *since* illness.

More formally, we can define the transition rate for state 2 to 3 as,

$$h_3(t) = h_{30}(t)\exp(X\beta_{31} + f(t - t_{30})\beta_{32})$$

where $h_{30}(t)$ is the baseline hazard function on the main timescale, time since entry to the healthy state 1. We have a vector of baseline covariates with associated log hazard ratios, X and β_{31}, respectively. Finally, we define t_{30} to be the time at which the observation entered State 2, and hence $(t - t_{30})$ defines the time since entry to state 2. Our function $f()$ can be as simple or complex as required, with associated coefficient vector β_{32}.

For simplicity, I'll assume a linear effect of time since entry to state 2, incorporating it into the illness-death transition models from the previous section, setting $\beta_{32} = -0.05$, meaning a longer time to illness reduces the risk of death.

```
. capture drop time* state* event*

. set seed 98798

. survsim time state event, transmatrix(tmat)                          ///
>         hazard1(user(exp(-2 :+ 0.2:* log({t}) :+ 0.1:*{t}))          ///
>                 covariates(trt 0.1))                                  ///
>         hazard2(dist(weibull) lambda(0.01) gamma(1.3)                 ///
>                 covariates(trt -0.5))                                 ///
>         hazard3(user(0.1 :* {t} :^ 1.5 :* exp(-0.05 :* ({t}:-{t0})))  ///
>                 covariates(trt -0.5)                                  ///
>                 tde(trt 0.1) tdefunction(log({t})))                   ///
>         maxtime(3)
variables time0 to time2 created
variables state0 to state2 created
variables event1 to event2 created
```

To refer to the entry time for a particular transition, we use `{t0}` alongside our usual `{t}` denoting time since study origin, and hence `({t} :- {t0})` allows us to define time since state entry. Of course, this can be extended in numerous ways.

5 Conclusion

In this paper I have introduced a range of extensions to the `survsim` command for simulating survival data from parametric distributions, custom/user-defined hazard or cumulative hazard functions, an estimated `merlin` survival model, or from a general competing risks or multi-state model. Further work will allow the ability to simulate from multivariate and hierarchical/multilevel `merlin` models.

About the author

Michael J. Crowther is an Associate Professor of Biostatistics at the University of Leicester. He works heavily in methods and software development, particularly in the field of survival analysis. He is currently part funded by a MRC New Investigator Research Grant (MR/P015433/1).

References

Beyersmann, J., A. Latouche, A. Buchholz, and M. Schumacher. 2009. Simulating competing risks data in survival analysis. *Stat Med* 28(6): 956–971. URL http://dx.doi.org/10.1002/sim.3516.

Crowther, M. J. (To appear). merlin - a unified modelling framework for data analysis and methods development in Stata. *The Stata Journal* . URL https://arxiv.org/abs/1806.01615.

Crowther, M. J., and P. C. Lambert. 2012. Simulating complex survival data. *The Stata Journal* 12(4): 674–687.

———. 2013. Simulating biologically plausible complex survival data. *Stat Med* 32(23): 4118–4134. URL http://dx.doi.org/10.1002/sim.5823.