



*The World's Largest Open Access Agricultural & Applied Economics Digital Library*

**This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.**

**Help ensure our sustainability.**

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

[aesearch@umn.edu](mailto:aesearch@umn.edu)

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

*No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.*



# Simulating time-to-event data from parametric distributions, custom distributions, competing-risks models, and general multistate models

Michael J. Crowther

Department of Medical Epidemiology and Biostatistics

Karolinska Institutet

Stockholm, Sweden

michael.crowther@ki.se

**Abstract.** In this article, 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 or user-defined distribution, a fitted `merlin` model, a specified cause-specific hazards competing-risks model, or a specified general multistate model (with multiple timescales). Left-truncation (delayed entry) is now also available for all settings. I illustrate the `survsim` command with some examples, demonstrating the huge flexibility that can be used to better evaluate statistical methods.

**Keywords:** `st0275_1`, `survsim`, survival analysis, Monte Carlo simulation, flexible parametric survival models, time to event, time to censoring, clinical trials, competing-risks models, multistate models, `merlin`

## 1 Introduction

Crowther and Lambert (2012) introduced `survsim` with the ability to simulate from a defined parametric distribution, such as exponential, Weibull, and Gompertz, as well as their two-component versions. The first version of `survsim` allowed both time-independent and time-dependent effects and allowed simulation of competing-risks data, providing a useful tool to generate event times. 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 that was also incorporated into `survsim`.

In this article, I introduce some substantial new developments:

1. The ability to simulate from a competing-risks or general multistate 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 approach combines the competing-risks simulation method

described by Beyersmann et al. (2009) with the general algorithm of Crowther and Lambert (2013).

2. The ability to simulate from a conditional distribution, that is, allowing for delayed entry or left-truncation, in all settings.
3. The ability to simulate directly from a fitted `merlin` survival model (Crowther 2020). This is currently limited to univariate, observation-level-only survival models, that is, models with no random effects.

The article is arranged as follows. In section 2, I briefly describe the core algorithm used to simulate event times in `survsim`. In section 2, 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 discussion.

## 2 Simulating survival times

Simulating data from a defined distribution can be simple or 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  $t$ :

$$F(t) = u$$

Hence,

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

Now solving for  $t$  relies on being able to invert the cumulative distribution function, and because 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) developed 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 article. For the competing-risks and more general multistate settings, this is used to simulate from the total hazard function (made up of the sum of cause- or transition-specific hazard functions leaving a particular state) as developed by Beyersmann et al. (2009) for competing risks. Once a survival time is simulated, a multinomial draw is conducted with probabilities equal to the cause- or transition-specific hazards divided by the total hazard to decide which state is then entered. In the multistate setting, this is repeated as the observation transitions from state to state until either an absorbing state or a maximum follow-up time is reached.

## 3 The `survsim` command

The `survsim` command can simulate survival data from a parametric distribution, a user-defined distribution, a fitted `merlin` model, a cause-specific hazards competing-risks model, or a general multistate model.

### 3.1 Syntax for simulating survival data from a parametric distribution

The command syntax is

```
survsim newvar1 [newvar2], distribution(string) lambda(numlist) [options]
```

where *newvar1* specifies the new variable name to contain the generated survival times. *newvar2* is required when the `maxtime()` option is specified, which defines the times of right-censoring.

#### 3.1.1 Options

`distribution(string)` specifies the parametric survival distribution to use, including `exponential`, `gompertz`, or `weibull`. All listed distributions are parameterized in the proportional hazards metric. `distribution()` is required.

`lambda(numlist)` defines the scale parameters in the exponential, Weibull, or Gompertz distribution. The number of values required depends on the model choice. The default is a single number corresponding with a standard parametric distribution. Under a mixture model, two values are required. `lambda()` is required.

`gamma(numlist)` defines the shape parameters of the Weibull or Gompertz parametric distribution. The number of entries must be equal to that of `lambda()`. `gamma()` is required for all but the exponential distribution.

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

`covariates(varname # [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 treatment variable coded 0 or 1 can be included with a log hazard-ratio of 0.5 by `covariates(treat 0.5)`. The variable `treat` must be in the dataset before `survsim` is called.

`tde(varname # [varname # ...])` creates nonproportional hazards by interacting covariates either with log time for an exponential or a Weibull model or with 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 times. 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 or delayed entry times to simulate from a conditional survival distribution. A common time can be specified for all observations by using `#`, or observation-specific left-truncation times can be specified by using *varname*.

`mixture` specifies that survival times be simulated from a two-component mixture model with mixture component distributions defined by `distribution()`. `lambda()` and `gamma()` must be of length 2.

`pmix(#)` defines the mixture parameter. The default is `pmix(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, you use a combination of numerical integration and root finding. The default is `nodes(30)`.

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

The command syntax is

```
survsim newvar1 newvar2, maxtime(# | varname) {loghazard(string) |
    hazard(string) | logchazard(string) | chazard(string)} [options]
```

where *newvar1* specifies the new variable name to contain the generated survival times and *newvar2* specifies the new variable name to contain the generated event indicator.

### 3.2.1 Options

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

`loghazard(string)` is the user-defined log-hazard function. The user must specify one of the following: `loghazard()`, `hazard()`, `logchazard()`, or `chazard()`. The function can include the following:

- `{t}`, which denotes the main timescale. If `ltruncated()` is specified, it is also measured on this timescale.
- *varname*, which denotes a variable in your dataset.
- `+-*/^`, which are standard Mata mathematical operators, using colon notation, that is, `2 :* {t}`; see [M-2] `op_colon`. Colon operators must be used because

behind the scenes, `{t}` gets replaced by an  $_N \times \text{nodes}()$  matrix when numerically integrating the hazard function.

- *mata\_function*, which denotes any Mata function, for example, `log()` and `exp()`.

`hazard(string)` is the user-defined baseline hazard function. The user must specify one of the following: `loghazard()`, `hazard()`, `logchazard()`, or `chazard()`. See `loghazard()` for more details and examples below.

`logchazard(string)` is the user-defined log cumulative-baseline-hazard function. The user must specify one of the following: `loghazard()`, `hazard()`, `logchazard()`, or `chazard()`. See `loghazard()` for more details and examples below.

`chazard(string)` is the user-defined baseline cumulative hazard function. The user must specify one of the following: `loghazard()`, `hazard()`, `logchazard()`, or `chazard()`. See `loghazard()` for more details and examples below.

`covariates(varname # [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 treatment variable coded 0 or 1 can be included with a log hazard-ratio of 0.5 by `covariates(treat 0.5)`. The variable `treat` must be in the dataset before `survsim` is called. If `chazard()` or `logchazard()` is used, then `covariates()` effects are additive on the log cumulative-hazard scale.

`tde(varname # [varname # ...])` creates nonproportional 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 the option `tde()` are interacted with to create time-dependent effects. The default is `tdefunction({t})`, that is, linear time. The function can include the following:

- `{t}`, which denotes the main timescale. If `ltruncated()` is specified, it is also measured on this timescale.
- `+-*/^`, which are standard Mata mathematical operators, using colon notation, that is, `2 :* {t}`; see [M-2] **op\_colon**. Colon operators must be used because behind the scenes, `{t}` gets replaced by an  $_N \times \text{nodes}()$  matrix when numerically integrating the hazard function.
- *mata\_function*, which denotes any Mata function, for example, `log()` and `exp()`.

`ltruncated(#|varname)` specifies the left-truncated or delayed entry times. A common time can be specified for all observations by using `#`, or observation-specific left-truncation times can be specified by using *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, you use a combination of numerical integration and root finding. The default is `nodes(30)`.

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

The command syntax is

```
survsim newvar1 newvar2, model(name) maxtime(#|varname)
```

where *newvar1* specifies the new variable name to contain the generated survival times and *newvar2* specifies the new variable name to contain the generated event indicator.

#### 3.3.1 Options

`model(name)` specifies the name of the `estimates` store object containing the estimates of the fitted model. The survival model must be fit using the `merlin` command. `survsim` will simulate from the fitted model, using covariate values that are in your current dataset. `model()` is required. 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 times. A common maximum follow-up time `#` can be specified for all observations, or observation-specific censoring times can be specified by using a *varname*. `maxtime()` is required.

### 3.4 Syntax for simulating survival times from a multistate model

The command syntax is

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

where *timestub* specifies the new variable name stub to contain the generated survival times for each transition. *statstub* specifies the new variable name stub to contain the generated state identifiers, that is, which state each observation has transitioned to or is

in, at the associated times. *eventstub* specifies the new variable name stub to contain the event indicators, that is, whether an event or right-censoring occurred at the associated times. When observations have reached an absorbing state or reached `maxtime()`, they will have missing observations in any further generated variables. When all observations have reached an absorbing state or are right-censored, then the simulation will cease, and the generated variables are returned.

Each `hazard#(haz_options)` represents a transition-specific hazard function, where `#` is used to index the transition in the transition matrix, defined in `transmatrix()`.

### 3.4.1 Options

`hazard#(haz_options)` represents a transition-specific hazard function, where `#` is used to index the transition in the transition matrix, defined in `transmatrix()`. `hazard1()` and `hazard2()` are required. *haz\_options* are the following:

`distribution(string)` specifies the parametric survival distribution to use, including `exponential`, `gompertz`, or `weibull`. All listed distributions are parameterized in the proportional hazards metric. Either `distribution()` or `user()` must be specified.

`lambda(#)` defines the scale parameter for an exponential, Weibull, or Gompertz distribution. `lambda()` is required.

`gamma(#)` defines the shape parameter for a Weibull or Gompertz parametric distribution. `gamma()` is required for all but the exponential distribution.

`user(function)` defines a custom hazard function. Either `user()` or `distribution()` must be specified. The function can include the following:

- `{t}`, which denotes the main timescale. If `ltruncated()` is specified, it is also measured on this timescale.
- `{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.
- `+*/^`, which are standard Mata mathematical operators, using colon notation, that is, `2 :* {t}`; see [M-2] **op\_colon**. Colon operators must be used because behind the scenes, `{t}` gets replaced by an `_N × nodes()` matrix when numerically integrating the transition hazard function.
- *mata\_function*, which denotes any Mata function, for example, `log()` and `exp()`.



For example,

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

is equivalent to

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

`covariates(varname # [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 treatment variable coded 0 or 1 can be included with a log hazard-ratio of 0.5 by `covariates(treat 0.5)`. The variable `treat` must be in the dataset before `survsim` is called.

`tde(varname # [varname # ...])` creates nonproportional hazards by interacting covariates with a function of time. Covariates are interacted with the specification in `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 the option `tde()` are interacted with to create time-dependent effects in the transition-specific hazard function. The default is `tdefunction({t})`, that is, linear time. The function can include the following:

- `{t}`, which denotes the main timescale. If `ltruncated()` is specified, it is also measured on this timescale.
- `+*/^`, which are standard Mata mathematical operators, using colon notation, that is, `2 :* {t}`; see [M-2] `op_colon`. Colon operators must be used because behind the scenes, `{t}` gets replaced by an  $_N \times \text{nodes}()$  matrix when numerically integrating the transition hazard function.
- `mata_function`, which denotes any Mata function, for example, `log()` and `exp()`.

`reset` specifies that this transition model be on a clock-reset timescale. The timescale is reset to 0 on entry; that is, 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 occurrences of `{t}` with `{t} - {t0}`, including those specified in `tdefunction()`.

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

`transmatrix(matname)` specifies the transition matrix that governs the multistate model. Transitions must be numbered as an increasing sequence of integers from  $1, \dots, K$ , from left to right, top to bottom of the matrix. Reversible transitions are allowed. By default, a competing-risks model is assumed.

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

`ltruncated(# | varname)` specifies left-truncated or delayed entry times, which are the times at which observations start in the initial starting states. A common time can be specified for all observations by using `#`, or observation-specific left-truncation times can be specified by using *varname*. The 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, you use a combination of numerical integration and root finding. 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 will simulate 300 observations and pick some distributions for the covariates, which should be self-explanatory:

```
. set obs 300
Number of observations (_N) was 0, now 300.
. set seed 134987
. generate trt = runiform()>0.5
. generate 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 were greater than 5, or we could generate observation-specific potential censoring times such as

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

Now we 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 or delayed entry using the `ltruncated()` option, or
- simulate from a two-component mixture distribution 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
. generate 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
```

This 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` (Royston 2012), which allows the simulation of survival times from a Royston–Parmar flexible parametric model. `survsim` now allows simulation from a survival model that has been fit 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)
Survival-time data settings
      Failure event: censrec==1
Observed time interval: (0, rectime]
      Exit on or before: failure
      Time 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 = -898.26542
Iteration 2:  log likelihood = -868.11298
Iteration 3:  log likelihood = -868.0269
Iteration 4:  log likelihood = -868.02684
Fixed effects regression model                                Number of obs = 686
Log likelihood = -868.02684
```

	Coefficient	Std. err.	z	P> z	[95% conf. interval]
_t:					
hormon	-.3932404	.1248267	-3.15	0.002	-.6378961    -.1485846
_cons	-2.196012	.1094092	-20.07	0.000	-2.41045    -1.981574
log(gamma)	.2509973	.0496958	5.05	0.000	.1535953    .3483993

I am 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 whatever we like, such as the imaginative name of `m1`:

```
. estimates store m1
```

We then pass this 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)
Survival-time data settings
    Failure event: died5!=0 & died5<.
Observed 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 = -1299.1672
Iteration 3:  log likelihood = -1298.8072
Iteration 4:  log likelihood = -1298.8059
Iteration 5:  log likelihood = -1298.8059

Fixed effects regression model                                Number of obs = 686
Log likelihood = -1298.8059
```

	Coefficient	Std. err.	z	P> z	[95% conf. interval]	
<b>_t:</b>						
hormon	-.3533906	.1014646	-3.48	0.000	-.5522576	-.1545237
_cons	-2.349859	.1100583	-21.35	0.000	-2.565569	-2.134149
log(gamma)	.2650627	.0421563	6.29	0.000	.1824378	.3476876

Note that `survsim` will simulate the same number of observations as `_N`, so any covariates in your model must have nonmissing 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 simulation from multivariate models or a model containing random effects.

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

`survsim` simulates competing-risks data using the cause-specific hazard setting described by Beyersmann et al. (2009), utilizing the general simulation algorithm described by Crowther and Lambert (2013). Let's simulate from a competing-risks model with two 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.
. generate trt = runiform()>0.5
. survsim time state event, hazard1(dist(weibull) lambda(0.1) gamma(0.8))
> hazard2(distribution(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 specification of `hazard#()` defines a cause-specific hazard function with the specified `distribution()` and associated baseline parameters, covariate effects, and time-dependent effects. Each of the `hazard#()` functions can be as similar or as 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

Because the competing-risks simulation is framed within a more general multistate setting, discussed in the next example, all observations are assumed to begin in an initial starting state 1 and 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		Total
	0	1	
1	484	0	484
2	0	414	414
3	0	102	102
Total	484	516	1,000

which shows that by 10 years, 484 observations were right-censored, 414 are in state 2, and 102 are in state 3.

Now let's simulate from a competing-risks model with three 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 distribution 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 three years.

```

. capture 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

```

state1	event1		Total
	0	1	
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 are 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 state 3.
- State 2: An intermediate **ill** state. Observations can come from state 1 and move on to state 3.
- State 3: An absorbing **dead** state. Observations can come from state 1 or state 2 but not leave.

This gives us three potential transitions between states:

- Transition 1: State 1  $\rightarrow$  State 2
- Transition 2: State 1  $\rightarrow$  State 3
- Transition 3: State 2  $\rightarrow$  State 3



This is defined by the following matrix:

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

The key is to think of the column or 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:

```
. matrix colnames tmat = "healthy" "ill" "dead"
. matrix rownames tmat = "healthy" "ill" "dead"
. matrix list tmat
tmat[3,3]
      healthy      ill      dead
healthy      .        1        2
  ill        .        .        3
  dead        .        .        .
```

Now that we have defined the transition matrix, we can use `survsim` to simulate some data. We will simulate 1,000 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
. generate 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 distribution 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 three 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 transition 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 represent the time at which observations entered the initial state, `time0`, and the associated state number, `state0`.

Because `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 either have 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)
```

1.	trt 0	time0 0	sta_0 1	time1 3	sta_1 1	eve_1 0	time2 .	sta_2 .
eve_2 .								
4.	trt 1	time0 0	sta_0 1	time1 .95636156	sta_1 2	eve_1 1	time2 3	sta_2 2
eve_2 0								
16.	trt 0	time0 0	sta_0 1	time1 1.0755764	sta_1 2	eve_1 1	time2 2.4401409	sta_2 3
eve_2 1								
112.	trt 1	time0 0	sta_0 1	time1 2.3290322	sta_1 3	eve_1 1	time2 .	sta_2 .
eve_2 .								

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

- 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. Because state 3 is an absorbing state, there are no further transitions.
- Observation 112 moves to state 3 at 2.329 years. Again, because state 3 is absorbing, there are no further transitions.

We could incorporate a variety of extensions; 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 have, of course, a much more complex multistate structure, that is, 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 because the estimation of such a model is computationally challenging, let alone simulating from one. However, `survsim` now provides this in an extremely general way, and indeed `merlin` can fit 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 state 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  $\beta_{31}$ , 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  $\beta_{32}$ .

For simplicity, I will 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 mortality rate.

```
. 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.

## 4.7 Simulating from a reversible illness-death model

I now extend the example from section 4.5 to allow for recovery; that is, observations can move from the `ill` state back to the `healthy` state. Our transition matrix now looks like this:

```
. matrix tmat = (.,1,2\3,.,4\.,.,.)
. matrix colnames tmat = "healthy" "ill" "dead"
. matrix rownames tmat = "healthy" "ill" "dead"
. matrix list tmat
tmat[3,3]
      healthy      ill      dead
healthy      .        1        2
  ill        3        .        4
  dead        .        .        .
```

Remember, transitions must be indexed going from top left to bottom right, along columns, and then down rows. So transition 3 is our new transition, going from `ill` to `healthy`.

Once again, we will simulate 1,000 observations and generate a binary treatment-group indicator, remembering to `set seed` first. However, now I will add delayed entry, allowing each observation to enter the study at a time greater than 0. Remember, entry and transition times are all simulated and recorded on the main overall timescale, even if the clock is reset on state entry. I will also vary the starting state of the observations, allowing observations to enter in either state 1 or state 2.

```
. clear
. set obs 1000
Number of observations (_N) was 0, now 1,000.
. set seed 9865
. generate trt = runiform()>0.5
. generate lt = 1.5 * runiform()
. generate startstate = 1 + (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 distribution with a beneficial treatment effect. The third transition has a Weibull distribution with no covariate effects. The fourth 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 three 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(dist(weibull) lambda(0.05) gamma(1))
> hazard4(user(0.1:* {t}:^ 1.5)
> covariates(trt -0.5)
> tde(trt 0.1) tdefunction(log({t})))
> ltruncated(1t) startstate(startstate) maxtime(3)
variables time0 to time4 created
variables state0 to state4 created
variables event1 to event4 created

```

Because `ltruncated()` and `startstate()` were specified, observations start in different states and at different times. Subsequent transitions are simulated until all observations either have 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,1,695,704)
```

1.	trt 0	lt .0026511	starts_e 1	time0 .00265108	state0 1	time1 3	state1 1
	event1 0	time2 .	state2 .	event2 .	time3 .	state3 .	event3 .
	time4 .		state4 .		event4 .		
695.	trt 1	lt .7018383	starts_e 2	time0 .70183825	state0 2	time1 1.244147	state1 1
	event1 1	time2 1.9200901	state2 2	event2 1	time3 2.2587202	state3 1	event3 1
	time4 3		state4 1		event4 0		
704.	trt 0	lt 1.279466	starts_e 2	time0 1.2794662	state0 2	time1 2.3864832	state1 1
	event1 1	time2 2.464492	state2 2	event2 1	time3 2.9616442	state3 3	event3 1
	time4 .		state4 .		event4 .		

Starting state and times are stored in `state0` and `time0`, respectively. We see the following:

- Observation 1 enters state 1 at 0.0026 years and remains there until being right-censored at three years.
- Observation 695 enters state 2 at 0.7018 years, recovers to state 1 at 1.244 years, has a relapse at 1.920 years, moving to state 2, recovers again at 2.259 years, and remains there until the end of follow-up.
- Observation 704 enters state 2 at 1.279 years, recovers to state 1 at 2.386 years, has a relapse at 2.464 years, moving to state 2, and subsequently dies at 2.9616 years, moving to state 3.

Note that in this case, we had four sets of new variables created. If we extend follow-up, then we will obtain more because we have a reversible process and `survsim` will simply continue to simulate transitions for observations until all observations have reached an absorbing state or the maximum follow-up time.

## 5 Conclusion

In this article, I introduced several extensions to the `survsim` command for simulating continuous time survival data from parametric distributions, custom or user-defined hazard or cumulative hazard functions, a fitted `merlin` survival model, or a general competing-risks or multistate model.

### 5.1 Further topics

`survsim` focuses on the simulation of continuous time survival data; however, it would be rather simple to apply some discretizing to the simulated event times to generate discrete-time or interval-censored survival data.

Incorporating a frailty or random effects can also be accommodated. To allow for a frailty (random intercept), one can simply generate a normally distributed random effect with mean 0 and standard deviation  $\sigma$  and include the generated variable in the `covariate()` option with a coefficient of 1 (note that this would not be valid if including left-truncation; see van den Berg and Drepper [2016]). Further examples of using `survsim` to simulate multilevel survival data and joint longitudinal-survival data are available at <https://reddooranalytics.se/products/survsim>.

For those interested in competing-risks and multistate modeling, I refer you to the `multistate` package, where, for example, the simulated datasets shown above may be `msset` and subsequently analyzed within a multistate setting (Crowther and Lambert 2017).

Further work will allow the ability to simulate from multivariate and hierarchical or multilevel fitted `merlin` models.

## 6 Programs and supplemental materials

To install a snapshot of the corresponding software files as they existed at the time of publication of this article, type

```
. net sj 22-1
. net install st0275_1      (to install program files, if available)
. net get st0275_1          (to install ancillary files, if available)
```

## 7 References

- Beyersmann, J., A. Latouche, A. Buchholz, and M. Schumacher. 2009. Simulating competing risks data in survival analysis. *Statistics in Medicine* 28: 956–971. <https://doi.org/10.1002/sim.3516>.
- Crowther, M. J. 2020. merlin—A unified modeling framework for data analysis and methods development in Stata. *Stata Journal* 20: 763–784. <https://doi.org/10.1177/1536867X20976311>.
- Crowther, M. J., and P. C. Lambert. 2012. Simulating complex survival data. *Stata Journal* 12: 674–687. <https://doi.org/10.1177/1536867X1201200407>.
- . 2013. Simulating biologically plausible complex survival data. *Statistics in Medicine* 32: 4118–4134. <https://doi.org/10.1002/sim.5823>.
- . 2017. Parametric multistate survival models: Flexible modelling allowing transition-specific distributions with application to estimating clinically useful measures of effect differences. *Statistics in Medicine* 36: 4719–4742. <https://doi.org/10.1002/sim.7448>.
- Royston, P. 2012. Tools to simulate realistic censored survival-time distributions. *Stata Journal* 12: 639–654. <https://doi.org/10.1177/1536867X1201200405>.
- van den Berg, G. J., and B. Drepper. 2016. Inference for shared-frailty survival models with left-truncated data. *Econometric Reviews* 35: 1075–1098. <https://doi.org/10.1080/07474938.2014.975640>.

### About the author

Michael J. Crowther is a biostatistician at the Karolinska Institutet and a consultant Stata developer. He works heavily in methods and software development, particularly in the field of survival analysis.