

Counting process and timeline data sets

Terry Therneau

September 9, 2026

0.1 Dataset types

0.1.1 Counting process

Survival data for the package is supported in four types:

1. The simple `Surv(time, status)` form, where the status is 0/1 or FALSE/TRUE for alive/dead.
2. Variants on the first for left and interval censored data, which are largely used by the parametric survival routine `survreg`.
3. The counting process form `Surv(time1, time2, status)`.
4. Timeline form, which is the newest addition and is particularly useful for multistate models

At its origin, the `survfit` and `coxph` routines supported only the simple style, and this is still true for some of the functions in the survival package, e.g., `survdiff`. The counting process style was originally added solely to support time-dependent covariates. Data sets have a structure like one below, which displays data from the primary biliary cirrhosis study. The counting process data is created from the `pbcseq` data using the `tmerge` routine.

	id	tstart	tstop	death	age	bili	ascites	chol
1	1	0	192	0	58.8	14.5	1	261
2	1	192	400	1	58.8	21.3	1	261
3	2	0	182	0	56.4	1.1	0	302
4	2	182	365	0	56.4	0.8	0	302
5	2	365	768	0	56.4	1.0	0	302
6	2	768	1790	0	56.4	1.9	0	302
7	2	1790	2151	0	56.4	2.6	1	230
8	2	2151	2515	0	56.4	3.6	1	230
9	2	2515	2882	0	56.4	4.2	1	230
10	2	2882	3226	0	56.4	3.6	1	244
11	2	3226	5169	0	56.4	4.6	1	237

For a simple time-dependent data set like this we have the subject id, the time interval (`tstart`, `tstop`], the death indicator, a fixed covariate of age at enrollment, followed by time dependent covariates of bilirubin, ascites, and chol count. Each row contains the covariate values that apply over the interval (`tstart`, `tstop`], along with the indicator of whether the interval ends in an event (death).

The use of open and closed brackets (`tstart`, `tstop`] for the intervals is purposeful and necessary. For example, in the PBC data subject 185 dies on day 2882, and subject 2 listed above is in the risk set for that death, but also has a new bilirubin recorded on that day. In a Cox model assessing the relationship of bilirubin and ascites to the risk of death, should the model use the value of 4.2 (row 9) or 3.6 (row 10) as subject 2's bilirubin at that death time? The proper answer is that the code should use the first. The underlying theory for the Cox model, which is based on martingales and counting processes, requires that covariate values used for an event must be known strictly before the time of that event. In analogy to a gambling game, bets must be placed before the dice leave the shooter's hand.

It was fairly quickly discovered that this form for the data made it easy to handle time-dependent strata, and within another year that it also allows for multiple events per subject, which was useful in a study of recurrent myocardial infarction. (I foresaw neither when the time1, time2 form was first created). The `cdg` data is an example of multiple events of the same type, and use of the Andersen-Gill model for analysis. The status variable can now have a value of 1 multiple times for a subject.

Use of a factor as the “status” variable extends the `Surv(time, status)` form to the case of competing risks and the `Surv(time1, time2, status)` form to general multi-state models. The first level of the factor is assumed to be “censored”, though for a multi-state model “no transition to another state occurred at the end of this interval” is the more accurate description. The label associated with this first or censoring level can be anything the user desires, the author uses both “censored” and “none” fairly commonly.

Multistate counting process data sets can get complex, particularly if there are also time-dependent covariates (which is often the case). The code below builds a time-dependent data set for the PBC data with two different state variables: simple alive/dead, and a second for modeling the progression of bilirubin over time. The second corresponds to the state space illustrated below. If subjects were under continuous observation we would expect only the state changes shown by the black arrows.

Here is the code to create the relevant data using `tmerge`. Feel free to skip over the code itself and concentrate on the final printout for subjects 1-2.

```
> p1 <- subset(pbcseq, !duplicated(id))
> pdata <- tmerge(p1[,c(1,3:5)], p1, id=id, death= event(futime, 1*(status==2)))
> pdata <- tmerge(pdata, pbcseq, id=id, bili= tdc(day, bili),
  ascites=tdc(day, ascites),
  chol = tdc(day, chol))
> pdata$age <- round(pdata$age,1)
> bili3 <- cut(pbcseq$bili, c(0, 1, 4, 50), c("normal", "1-4", "4+"))
> # two 0-1 visits in a row is not a transition
> b3e <- nostutter(pbcseq$id, as.numeric(bili3))
> pdata2 <- tmerge(pdata, pbcseq, id= id,
  bili3 = tdc(day, bili3), bstate= event(day, b3e))
> temp <- with(pdata2, ifelse(death==1, 4*death, as.numeric(bstate) -1L))
> pdata2$bstate <- factor(temp, 0:4,
  c("censor", "normal", "1-4", "4+", "death"))
> subset(pdata2, id<3, c(id, tstart, tstop, death, age, bili, ascites, chol,
  bili3, bstate))
```

	id	tstart	tstop	death	age	bili	ascites	chol	bili3	bstate
1	1	0	192	0	58.8	14.5		1	261	4+ censor
2	1	192	400	1	58.8	21.3		1	261	4+ death
3	2	0	182	0	56.4	1.1		0	302	1-4 normal
4	2	182	365	0	56.4	0.8		0	302	normal censor
5	2	365	768	0	56.4	1.0		0	302	normal 1-4
6	2	768	1790	0	56.4	1.9		0	302	1-4 censor
7	2	1790	2151	0	56.4	2.6		1	230	1-4 censor
8	2	2151	2515	0	56.4	3.6		1	230	1-4 4+

9	2	2515	2882	0	56.4	4.2	1	230	4+	1-4
10	2	2882	3226	0	56.4	3.6	1	244	1-4	4+
11	2	3226	5169	0	56.4	4.6	1	237	4+	death

A subtle and often confusing aspect of the counting process involves covariates versus events. The bilirubin, current state (bili3) and transition (bstate) variables in the data frame all originate from the same values, but the first two apply to the entire interval and the last describes any transitions to a new state at the end of the interval. Subject 1, for instance is in the bilirubin > 4 category for both the (0,192] and (192, 400] interval, their status (bevent) variable is censor (no change) at time 192 and death at time 400. Subject 2 moves between the states; the bstate variable looks ahead to the next state and only records changes of state.

An important check for any counting process data set is the survcheck routine.

```
> survcheck(Surv(tstart, tstop, bstate) ~ 1, pdata2, id=id, istate=bili3)
Call:
survcheck(formula = Surv(tstart, tstop, bstate) ~ 1, data = pdata2,
  id = id, istate = bili3)
```

Unique identifiers	Observations	Transitions
312	1945	438

Transitions table:

	to					
from	normal	1-4	4+	death	(censor)	
normal	0	91	3	9	77	
1-4	63	0	109	21	60	
4+	1	31	0	110	35	
death	0	0	0	0	0	

Number of subjects with 0, 1, ... transitions to each state:

	count								
state	0	1	2	3	4	5	6		
normal	259	42	11	0	0	0	0		
1-4	214	77	18	3	0	0	0		
4+	213	88	9	2	0	0	0		
death	172	140	0	0	0	0	0		
(any)	91	96	72	29	14	5	5		

It is important to not just print the table, but carefully *read* it.

- We see no transitions from death to another state, including death:death (died twice), nor any follow-up time after death. This is as it should be.
- There were no warning messages about invalid transitions or time intervals.
- Because PBC is a progressive disease we expect more transitions above than below the diagonal in the transitions table.

The counting process style offers great flexibility. Unfortunately, it is easy to create data sets which are not valid. With rare exceptions, the final rows representing a subject should follow a simple rule: the described path is physically possible. In particular

1. A subject cannot be two places at once (no overlapping intervals)
2. They cannot disappear and then return (no disjoint intervals)
3. Any state that is entered must be occupied for a positive time interval (no zero length intervals, current state and transitions must be consistent)

The `tmerge` routine does a few things to help this process, e.g., the `tdc()` and `event()` operators signal changes that should occur at the start and end of an interval, respectively. One particular action of `tdc()` is to fill in missing values using last value carried forward (LVCF). Looking at the `pbcseq` data, for instance, subject 2 has a cholesterol of 302 at baseline, is missing a cholesterol measurement on visits 2–4, and has a new value of 230 at visit 5. In the `pdata2` data set above has this been filled in. If this were not done then the standard “delete all rows with missing values” action, which is the first step of the `coxph` or `survfit` routines, would result in a data set that violated rule 2 above, leading to an invalid analysis. (Or an error message when those routines do an internal call of the `survcheck` code, see the `survcheckallow` argument of `coxph.control`).

An example where we once allowed a violation of rule 2 was a cancer clinical trial with long follow-up. One patient completely disappeared from the surveillance system for 3 years, then re-appeared at the clinic one day and resumed regular appointments. After some discussion, we assumed that neither death nor progression would have been discovered had they occurred in the interim, but that the information post rejoining was useful, so we allowed disjoint intervals of follow up. But this remains a rare case.

Below are two fits, the first with bilirubin as a time-dependent covariate, the second with it a state. The latter is a valid way to create time-dependent survival curves.

```
> pcox1 <- coxph(Surv(tstart, tstop, death) ~ bili3, pdata2,
                 id=id)
> psurv1a <- survfit(Surv(tstart, tstop, bstate) ~ 1, pdata2,
                   id= id, istate= bili3, p0=c(1,0,0,0))
> psurv1b <- survfit(Surv(tstart, tstop, bstate) ~ 1, pdata2,
                   id= id, istate= bili3, p0=c(0,1,0,0))
> psurv1c <- survfit(Surv(tstart, tstop, bstate) ~ 1, pdata2,
                   id= id, istate= bili3, p0=c(0,0,1,0))
> plot(psurv1c[4], col=3, lwd=2, conf.int=F, xscale=365.25,
       xlab= "Years from randomization", ylab="Death")
> lines(psurv1b[4], col=2, lwd=2, conf.int=F)
> lines(psurv1a[4], col=1, lwd=2, conf.int=F)
```

0.2 Timeline data

A more recent addition to the package is to allow what we call a *timeline* data sets. Such data has a unique (subject identifier, time) pair that identifies each row of data, along with information

that was observed at that time. Time constant covariates such a genotype can appear on all rows, or only when first recorded, the latter obeying the strict “observed at that time” rule. For simple survival for instance, the timeline form would have two rows per subject. The first 3 subjects of the lung cancer data set would appear as below (some variables omitted from the printout for width).

	id	time	death	inst	age	sex	ph.ecog	pat.karno
1	1	0	0	3	74	1	1	100
2	1	306	1	NA	NA	NA	NA	NA
3	2	0	0	3	68	1	0	90
4	2	455	1	NA	NA	NA	NA	NA
5	3	0	0	3	56	1	0	90
6	3	1010	0	NA	NA	NA	NA	NA

This is clearly not an advantageous approach for either simple survival or competing risks data, which have only one row per subject in the standard form. The advantage comes with multistate data sets: we no longer need to distinguish between covariates and outcomes in the data, and data sets can be created with any number of tools, we are not tied to tmerge. Here is code for the sequential pbc data set earlier.

```
> # separate out death, and add it on the end
> # death yes/no *is* observed every visit
> temp <- with(subset(pbcseq, !duplicated(id)),
               data.frame(id=id, day =fuptime, death= 1*(status==2)))
> pdata3 <- cbind(pbcseq[, -(2:3)], death=0)
> pdata3 <- merge(pdata3, temp, all=TRUE)
> # create a factor for joint outcome
> pdata3$bili3 <- cut(pdata3$bili, c(0,1,4, 50), c("normal","1-4", "4+"))
> temp <- with(pdata3, ifelse(!duplicated(id, fromLast=TRUE), 4*death,
                           as.numeric(bili3)))
> pdata3$bstate <- factor(temp, 0:4, c("censor", "normal","1-4", "4+", "death"))
> subset(pdata3, id<3, c(id, day, death, age, bili, ascites, chol,
                        bili3, bstate))
```

	id	day	death	age	bili	ascites	chol	bili3	bstate
1	1	0	0	58.76523	14.5	1	261	4+	4+
2	1	192	0	58.76523	21.3	1	NA	4+	4+
3	1	400	1	NA	NA	NA	NA	<NA>	death
4	2	0	0	56.44627	1.1	0	302	1-4	1-4
5	2	182	0	56.44627	0.8	0	NA	normal	normal
6	2	365	0	56.44627	1.0	0	NA	normal	normal
7	2	768	0	56.44627	1.9	0	NA	1-4	1-4
8	2	1790	0	56.44627	2.6	1	230	1-4	1-4
9	2	2151	0	56.44627	3.6	1	NA	1-4	1-4
10	2	2515	0	56.44627	4.2	1	NA	4+	4+
11	2	2882	0	56.44627	3.6	1	244	1-4	1-4
12	2	3226	0	56.44627	4.6	1	237	4+	4+
13	2	5169	0	NA	NA	NA	NA	<NA>	censor

```

> pcox2 <- coxph(Surv2(day, death) ~ bili3, pdata3, id= id, ties="efron")
> all.equal(coef(pcox1), coef(pcox2))
[1] TRUE
> psurv2a <- survfit(Surv2(day, bstate) ~1, pdata3, id=id, p0=c(1,0,0,0))
> all.equal(psurv1a$pstate, psurv2a$pstate)
[1] TRUE

```

The call is different, but the result of the fit is the same. Internally, the timeline data set is first rewritten into counting process form, and then computations are done using the counting process data. The use of `Surv2` instead of `Surv` informs the routine that the data is of timeline form. A user version of the transformed data can be obtained using the `fromtimeline` function. Such a call may sometimes be necessary, i.e., if one wants to change the arguments of `lvcf=TRUE` and `repeated=FALSE`, which are the default arguments for the internal call.

Footnote: We seriously considered making the detection of the timeline form be automatic, i.e., any time there was an `id` statement, multiple rows per subject, and simple `Surv(time, status)` form. There are, however, a few exceptions. One example is the diabetic retinopathy study, where each patient has one treated and one untreated eye. There are two rows per subject (eyes) that both start at time 0. A `survcheck` call will complain about overlapping time intervals, but the following `survfit` call is perfectly legal, both statistically and in the code.

```

> rfit1 <- survfit(Surv(futime, status) ~ trt, id=id, retinopathy)
> rfit2 <- survfit(Surv(futime, status) ~ trt, cluster=id, retinopathy)

```

The two calls give the same result. The help file for the retinopathy data shows the second form, and indeed for all valid cases that we can think of with simple (time, status) data and multiple rows per subject the cluster form is both valid and clearer. However, the current decision is to require `Surv2` and not try to auto detect the timeline case, a primary argument being that it makes the code somewhat clearer to a user.

0.3 Multistate models and missing values

One of the features of the multistate `coxph` call is the ability to have certain covariates apply to only a subset of the transitions. Say for instance that one of the states was surgical intervention, such as in the Crohn's disease data set, and that a particular covariate depended on availability of tissue from that procedure. Then by definition that covariate is only applicable to further transitions after the surgical state. A user might choose to use a particular covariate for only a subset of transitions for multiple reasons, however.

Say that a covariate 'zed' is only used for the 2:3 transition within a 3 state model. Any observation (time1, time2, status, istate) with istate not equal to 2 will be unaffected by the value of zed, or whether zed is NA. The `coxph` routine will, properly, not remove this observation from the risk sets for a 2:3 transition due to a missing value for zed. What about a subject with a (time1, time2) row with a current state (istate) of 2 and missing value for zed? This row will be removed due to missing, but other rows of data for this subject are still informative for other transitions.

Here is a constructed example using the myeloid data.

```

> tdata <- tmerge(myeloid[,1:4], myeloid, id=id, death=event(futime,death),
  priortx = tdc(txttime), sct= event(txttime))
> tdata$event <- factor(with(tdata, sct + 2*death), 0:2,
  c("censor", "sct", "death"))
> tdata$sex[tdata$id %in% 273:275] <- NA # obs 425 to 428
> tdata$flt3[tdata$id %in% 271:273] <- NA # obs 422 to 425
> tdata$event[tdata$id==270 & tdata$tstart>0] <- NA
> subset(tdata, id %in% 270:275)

```

	id	trt	sex	flt3	tstart	tstop	death	priortx	sct	event
420	270	A	m	C	0	121	0	0	1	sct
421	270	A	m	C	121	191	1	1	0	<NA>
422	271	A	f	<NA>	0	223	0	0	1	sct
423	271	A	f	<NA>	223	707	1	1	0	death
424	272	B	m	<NA>	0	103	1	0	0	death
425	273	A	<NA>	<NA>	0	283	1	0	0	death
426	274	B	<NA>	C	0	293	1	0	0	death
427	275	B	<NA>	A	0	102	0	0	1	sct
428	275	B	<NA>	A	102	1442	0	1	0	censor

```

> check1 <- survcheck(Surv(tstart, tstop, event) ~1, tdata, id=id)
> check1

```

Call:

```

survcheck(formula = Surv(tstart, tstop, event) ~ 1, data = tdata,
  id = id)

```

Unique identifiers	Observations	Transitions
646	1008	683

1 observations removed due to missing

Transitions table:

	to		
from	sct	death	(censor)
(s0)	364	140	142
sct	0	179	183
death	0	0	0

Number of subjects with 0, 1, ... transitions to each state:

	count		
state	0	1	2
sct	282	364	0
death	327	319	0
(any)	142	325	179

```

> check2 <- survcheck(Surv(tstart, tstop, event) ~sex, tdata, id=id)
> fit <- coxph(list(Surv(tstart, tstop, event) ~ trt,
  1:3 + 2:3 ~ sex,
  1:2 + 2:3 ~ flt3), tdata, id=id)

```



```
> fit
Call:
coxph(formula = list(Surv(tstart, tstop, event) ~ trt, 1:3 +
  2:3 ~ sex, 1:2 + 2:3 ~ flt3), data = tdata, id = id)
```

```
1:2      coef exp(coef) robust se      z      p
trtB  -0.1467   0.8635   0.1050 -1.397 0.1625
flt3B  0.4447   1.5600   0.1401  3.175 0.0015
flt3C  0.4856   1.6251   0.1547  3.139 0.0017
```

```
1:3      coef exp(coef) robust se      z      p
trtB  -0.39382  0.67447  0.17111 -2.302 0.0214
sexm  0.03863  1.03939  0.17056  0.227 0.8208
```

```
2:3      coef exp(coef) robust se      z      p
trtB  -0.2976   0.7426   0.1643 -1.811 0.0701
sexm   0.2474   1.2807   0.1630  1.518 0.1291
flt3B  0.2351   1.2651   0.2492  0.944 0.3454
flt3C  0.5352   1.7078   0.2632  2.034 0.0420
```

```
States: 1= (s0), 2= sct, 3= death
```

```
Likelihood ratio test=31.05 on 9 df, p=0.0002906
n= 1005, unique id= 645, number of events= 679
(4 observations deleted due to missingness)
```

```
> check1$transitions- fit$transitions
```

```
      to
from  sct death (censor)
(s0)   1     2         0
sct    0     1         0
death  0     0         0
```

The data set has 1009 observations on 646 subjects. Since there is no `istate` option, all subjects are assumed to start in state 1, which will be labeled as “(s0)” on the printout. The result of `check1` will have 646/1008, one observation is lost due to the missing event on row 421, but no subjects. The `check2` result has 643/1004 since all rows with missing sex are removed. For the `coxph` model the transition table reflects events that are actually counted for a transition.

- One 1:2 transition is lost, obs 427, due to missing `flt`. Observation 422 is missing sex, but that is not used in the 1:2 model
- One 2:3 transition is lost, obs 423
- Two 1:3 transitions are lost, obs 425–426

Three observations are removed from the model frame, 421 which is also removed in `check1`, 423 which is at risk for only the 2:3 transition, and 425 which has missings for both the 1:2 and 1:3 transitions and is not at risk for 2:3. This drops the number of unique subjects to 645 and the rows to 1006.

Internally, the `coxph` code calls the `survcheck` function to verify legality of the data, but it does so before removal of any rows due to missing covariates. If the `survcheck` routine were called post removal, and some subjects' now had an induced discontinuity in follow-up an error would be signaled; but we consider this to be a false positive. (Reporting is further modified by the `survcheckfail` argument of `control.coxph`.) Likewise, user calls to `survcheck` will almost always want to use `~1` as the right hand side of the formula.