Further documentation on code and methods in the survival package

Terry Therneau

March 2024

Chapter 1

Introduction

> Kernighan's Law: "Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"

The most complex code in the survival package arises out of two aspects. First, some of the mathematical formulas underlying the code are themselves complex, a second is the work I have done to avoid $O(n^2)$ computations in the algorithms. A consequence of this is that a few comment lines interspersed into the source code will never be enough information. Anyone reading it, including myself, is likely to ask "what the heck is he trying to *do* here?" A higher level overview is needed that can include equations and talk about higher level concepts.

For many years I have used the noweb package to intersperse technical documentation with the source code for the survival package. However, despite its advantages, the uptake of noweb by the R community in general has been nearly nil. It is increasingly clear that no future maintainer will continue the work, e.g., on the github page I have yet to recieve a suggested update that actually fixed the .Rnw source file instead of the .R file derived from it. This means that I can't merge a suggested change automatically, but have to replicate it myself.

This document is a start at addressing this. As routines undergo maintainance, I will remove the relevant .Rnw file in the noweb directory and work directly on the C and R code, migrating the extra material into this document. In this vignette are discussions of design issues, algorithms, and detailed formulas. In the .R and .c code I will place comments of the form "See methods document, abc:def", adding an abc:def entry to the index of this document. I am essentially splitting each noweb document into two parts. The three advantages are

- Ease the transition to community involvement and maintainance.
- The methods document will be a more ready source of documentation for those who want to know technical details, but are not currently modifying the code.
- (minor) I expect it will be useful to have the methods document and the R or C code open simultaneously in 2 windows, when editing.

However, this conversion will be a long process.

Notation Throughout this document I will use formal counting process notation, so you may as well get used to it. The primary advantage is that it is completely precise, and part of the goal for this document is to give a fully accurate description of what is computed. In this notation we have

- $Y_i(t) = 1$ if observation i is at risk at time t, 0 otherwise.
- $N_i(t) =$ total number of events, up to time t for subject i.
- $dN_i(t) = 1$ if there was an event at exactly time t
- $X =$ the matrix of covariates, with n rows, one per observation, and a column per covariate.
- $X(s) =$ the time-dependent covariate matrix, if there are time-dependent covariates

For multistate models the extends to $Y_{ij}(t)$, which is 1 if subject i is at risk and in state j at time t, and $N_{ijk}(t)$ which is the number of j to k transitions that have occured, for subject i, up to time t. The number at risk and number of events at time t can be written as

$$\begin{aligned} n(t) &= \sum_{i=1}^{n} w_i Y_i(t) \\ &= \overline{Y}(t) \\ d(t) &= \sum_{i=1}^{n} w_i dN_i(t) \\ &= d\overline{N}(t) \end{aligned}$$

where w_i are optional weights for each observation. $\overline{N}(t)$ is the cumulative number of events up to time t. I will often also use $r_i(t) = \exp(X_i(t)\beta)$ as the per observation risk score in a Cox model.

Chapter 2

Survival Curves

The survfit function was set up as a method so that we could apply the function to both formulas (to compute the Kaplan-Meier) and to coxph objects. The downside to this is that the manual pages get a little odd: `survfit` is generic and `survfit.formula` and `survfit.coxph` are the ones a user will want. But from a programming perspective it has mostly been a good idea. At one time, long long ago, we allowed the function to be called with "Surv(time, status)" as the formula, i.e., without a right hand side of ~1. That was a bad idea, now abandoned: `survfit.Surv` is now a simple stub that prints an error message.

2.1 Roundoff and tied times

One of the things that drove me nuts was the problem of "tied but not quite tied" times. It is a particular issue for both survival curves and the Cox model, as both treat a tied time differently than two close but untied values. As an example consider two values of 24173 = 23805 + 368. These are values from an actual study with times in days: enrollment at age 23805 days and then 368 days of follow-up. However, the user chose to use age in years, and saved those values out in a CSV file, the left hand side of the above equation becomes 66.18206708000000 and the right hand side addition yeilds 66.18206708000001. The R phrase `unique(x)` sees these two values as distinct but `table(x)` and `tapply` see it as a single value since they first apply `factor` to the values, and that in turn uses `as.character`. A transition through CSV is not necessary to create the problem. Consider the small code chunk below. For someone born on 1960-03-10, it caclulates the time interval between a study enrollment on each date from 2010-01-01 to 2010-07-29 (200 unique dates) and a follow up that is exactly 29 days later, but doing so on age scale.

```
> tfun <- function(start, gap, birth= as.Date("1960-01-01")) {
      as.numeric(start-birth)/365.25 - as.numeric((start + gap)-birth)/365.25
  }
> test <- logical(200)
> for (i in 1:200) {
      test[i] <- tfun(as.Date("2010/01/01"), 29) ==
                 tfun(as.Date("2010/01/01") + i, 29)
```

```
  }
> table(test)
test
FALSE  TRUE
  148    52
```

The number of FALSE entries in the table depends on machine, compiler, and possibly several other issues. There is discussion of this general issue in the R FAQ: "why doesn't R think these numbers are equal". The Kaplan-Meier and Cox model both pay careful attention to ties, and so both now use the `aeqSurv` routine to first preprocess the time data. It uses the same rules as `all.equal` to adjudicate ties and near ties. See the vignette on tied times for more detail.

The survfit routine has been rewritten more times than any other in the package, as we trade off simplicty of the code with execution speed. The current version does all of the oranizational work in S and calls a C routine for each separate curve. The first code did everything in C but was too hard to maintain and the most recent prior function did nearly everything in S. Introduction of robust variance prompted a movement of more of the code into C since that calculation is computationally intensive.

The `survfit.formula` routine does a number of data checks, then hands the actual work off to one of three computational routes: simple survival curves using the `survfitKM.R` and `survfitkm.c` functions, interval censored data to `survfitTurnbull.R`, and multi-state curves using the `survfitAJ.R` and `survfitaj.c` pair.

The addition of `+ cluster(id)` to the formula was the suggested form at one time, we now prefer `id` as a separate argument. Due to the long comet's tail of usage we are unlikely to formally depreciate the older form any time soon. The istate argument applies to multi-state models (Aalen-Johansen), or any data set with multiple rows per subject; but the code refrains from complaint if it is present when not needed.

2.2 Single outcome

We bein with classic survival, where there is a single outcome. Multistate models will follow in a separate section.

At each event time we have

- n(t) = weighted number at risk
- d(t) = weighted number of events
- e(t) = unweighted number of events

leading to the Nelson-Aalen estimate of cumulative hazard and the Kaplan-Meir estimate of survival, and the estimate of hazard at each time point.

$$\begin{aligned} KM(t) &= KM(t-)(1 - d(t)/n(t) \\ NA(t) &= NA(t-) + d(t)/n(t)h(t) \qquad\qquad = d(t)/n(t) \end{aligned}$$

An alternative estimate in the case of tied times is the Fleming-Harrington. When there are no case weights the FH idea is quite simple. The assumption is that the real data is not tied,

but we saw a coarsened version. If we see 3 events out of 10 subjects at risk the NA increment is 3/10, but the FH is 1/10 + 1/9 + 1/8, it is what we would have seen with the uncoarsened data. If there are case weights we give each of the 3 terms a 1/3 chance of being the first, second, or third event

$$\begin{aligned} KM(t) &= KM(t-)(1 - d(t)/n(t) \\ NA(t) &= NA(t-) + d(t)/n(t) \\ FH(t) &= FH(t-) + \sum_{i=1}^{3} \frac{(d(t)/3}{n(t) - d(t)(i-1)/3} \end{aligned}$$

If we think of the size of the denominator as a random variable Z, an exact solution would use $E(1/Z)$, the FH uses $1/E(Z)$ and the NA uses $1/\max(Z)$ as the denominator for each of the 3 deaths. Although Fleming and Harrington were able to show that the FH has a lower MSE than the KM, it little used. The primary reasons for its inclusion in the package are first, that I was collaborating with them at the time so knew of these results, but second and more importantly, this is identical to the argument for the Efron approximation in a Cox model. (The NA hazard corresponds to the Breslow approximation.) The Efron approx is the default for the coxph function, so post coxph survival curves need to deal with it.

When one of these 3 subjects has an event but continues to be at risk, which can happen with start/stop data, then the argument gets trickier. Say that the first of the 3 continues, the others do not. We can argue that subject 1 remains at risk for all 3 denominators, or not. The first is more a mathematical viewpoint, the second more medical, e.g., in a study with repeated infections you will never have a second one recorded on the same day. For multi-state, we finally "tossed in the towel" and now use the Breslow approx as default for multi-state hazard (coxph) models. For single state, the FH estimate is rarely reqested but we still do our best to handle all aspects of it (pride and history), but there would be few tears if it were dropped.

When there is (time1, time2) data the code uses a `position` vector, explained further below which is 1 if this is obs is the leftmost for a subjet and 2 if it is the rightmost, 3 if it is both. A primary purpose was to not count a subject as an extra entry and censor in the middle of a string of times such as (0, 10), (20, 35), (35, 50), but we also use it to moderate the FH estimate: only those with an event at the end of their intervals participate in the special computation for ties.

The survfitKM call has arguments of

- y: a matrix y containing survival times, either 2 or 3 columns
- weight: vector of case weights
- ctype: compuation for the cumulative hazard: either the Nelson-Aalen (1) or Efron approximation for ties (2) approach. Number 2 is very rarely used.
- stype: computation for the survival curve: either product-limit (1), also known as the Kaplan-Meier or exp(-cumulative hazard) (2). Use of ctype=2, stype=2 matches a Cox model using the Efron approximation for ties, ctype=1, stype=2 is the Fleming-Harrington estimate of survival or a Cox model with the Breslow approximation for ties.
- type: older form of the ctype/stype argument, retained for backwards compatability. Type 1= (ctype 1/ stype 1), 2= (2, 1), 3 = (1,2), 4= (2,2).

- id: subject id, used for data checks when there are multiple rows per subject
- cluster: clustering used for the robust variance. Clustering is based on id if cluster is missing (the usual case)
- influence: return the influence matrix for the survival
- start.time: optional starting time for the curve
- entry: return entry times for (time1, time2) data
- time0: include time 0 in the output
- se.fit: compute the standard errors
- conf.type, conf.int, conf.lower: options for confidence intervals

2.2.1 Confidence intervals

If p is the survival probability and $s(p)$ its standard error, we can do confidence intervals on the simple scale of $p \pm 1.96s(p)$, but that does not have very good properties. Instead use a transformation $y = f(p)$ for which the standard error is $s(p)f'(p)$, leading to the confidence interval

$$f^{-1}\left(f(p) + -1.96s(p)f'(p)\right)$$

Here are the supported transformations.

	f	f'	f^{-1}
log	$\log(p)$	$1/p$	$\exp(y)$
log-log	$\log(-\log(p))$	$1/\left[p\log(p)\right]$	$\exp(-\exp(y))$
logit	$\log(p/1-p)$	$1/[p(1-p)]$	$1 - 1/\left[1+\exp(y)\right]$
arcsin	$\arcsin(\sqrt{p})$	$1/(2\sqrt{p(1-p)})$	$\sin^2(y)$

Plain intervals can give limits outside of (0,1), we truncate them when this happens. The log intervals can give an upper limit greater than 1 (rare), again truncated to be ≤ 1; the lower limit is always valid. The log-log and logit are always valid. The arcsin requires some fiddling at 0 and $\pi/2$ due to how the R function is defined, but that is only a nuisance and not a real flaw in the math. In practice, all the intervals except plain appear to work well. In all cases we return NA as the CI for survival=0: it makes the graphs look better.

Some of the underlying routines compute the standard error of p and some the standard error of $\log(p)$. The `selow` argument is used for the modified lower limits of Dory and Korn. When this is used for cumulative hazards the ulimit arg will be FALSE: we don't want to impose an upper limit of 1.

2.2.2 Robust variance

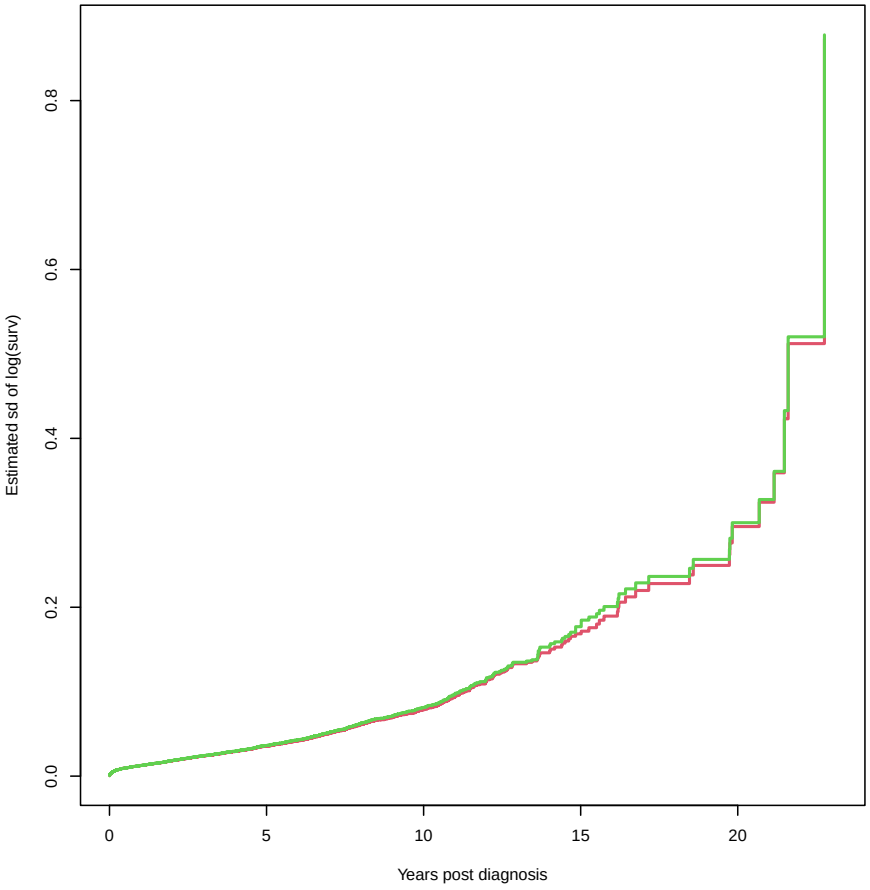
For an ordinary Kapan-Meier curve it can be shown that the infinitesimal jackknife (IJ) variance is identical to the Greenwood estimate, so the extra compuational burden of a robust estimate is unnecessary. The proof does not carry over to curves with (time1, time2) data. The use of (time1, time2) data for a single outcomes arises in 3 cases, with illustrations shown below.

- Delayed entry for a subset of subjects. The robust variance in this case is not identical to the Greenwood formula, but is very close in all the situations I have seen.
- Multiple events of the same type, as arises in reliability data. In this case the cumulative hazard rather than P(state) is the estimate of most interest; it is known as the mean cumulative function (MCF) in the reliability literature. If there are multiple events per subject (the usual) an id variable is required. A study with repeated glycemic events as the endpoint (not shown) had a mean of 48 events per subject over 5.5 months. In this case accounting for within-subject correlation was crucial; the ratio of robust/asymptotic standard error was 4.4. For the valveSeat data shown below the difference is minor.
- Survival curves for a time-dependent covariate, the so called "extended Kaplan-Meier" [2]. I have strong statistical reservations about this method.

Here are examples of each. In the myeloma data set the desired time scale was time since diagnosis, and some of the patients in the study had been diagnosed at another institution before referral to the study institution.

```
> fit1a <- survfit(Surv(entry, futime, death) ~ 1, myeloma)
> fit1b <- survfit(Surv(entry, futime, death) ~ 1, myeloma, id=id, robust=TRUE)
> matplot(fit1a$time/365.25, cbind(fit1a$std.err, fit1b$std.err/fit1b$surv),
         type='s',lwd=2, lty=1, col=2:3, #ylim=c(0, .6),
         xlab="Years post diagnosis", ylab="Estimated sd of log(surv)")
> #
> # when two valve seats failed at the same inspection, we need to jitter one
> #  of the times, to avoid a (time1, time2) interval of length 0
> ties <- which(with(valveSeat, diff(id)==0 & diff(time)==0))  #first of a tie
> temp <- valveSeat$time
> temp[ties] <- temp[ties] - .1
> vdata <- valveSeat
> vdata$time1 <- ifelse(!duplicated(vdata$id), 0, c(0, temp[-length(temp)]))
> vdata$time2 <- temp
> fit2a <- survfit(Surv(time1, time2, status) ~1, vdata)
> fit2b <- survfit(Surv(time1, time2, status) ~1, vdata, id=id)
> plot(fit2a, cumhaz=TRUE, xscale=365.25, xlab="Years in service",
       ylab="Estimated number of repairs")
> lines(fit2b, cumhaz=TRUE, lty=c(1,3,3))
> legend(150, 1.5, c("Estimate", "asymptotic se", "robust se"), lty=1:3, bty='n')
> #
> # PBC data, categorized by most recent bilirubin
> # as an example of the EKM
> pdata <- tmerge(subset(pbcseq, !duplicated(id), c(id, trt, age, sex, stage)),
                  subset(pbcseq, !duplicated(id, fromLast=TRUE)), id,
                  death= event(futime, status==2))
> bcut <- cut(pbcseq$bili, c(0, 1.1, 5, 100), c('normal', 'moderate', 'high'))
> pdata <- tmerge(pdata, pbcseq, id, cbili = tdc(day, bcut))
> pdata$ibili <- pdata$cbili[match(pdata$id, pdata$id)] # initial bilirubin
> ekm <- survfit(Surv(tstart, tstop, death) ~ cbili, pdata, id=id)
```

```
> km  <- survfit(Surv(tstart, tstop, death) ~ ibili, pdata, id=id)
> plot(ekm, fun='event', xscale=365.25, lwd=2, col=1:3, conf.int=TRUE,
      lty=2, conf.time=c(4,8,12)*365.25,
      xlab="Years post enrollment", ylab="Death")
> lines(km, fun='event', lwd=1, col=1:3, lty=1)
> #      conf.time= c(4.1, 8.1, 12.1)*365.25)
> text(c(4600, 4300, 2600), c(.23, .56, .78), c("Normal", "Moderate", "High"),
      col=1:3, adj=0)
> legend("topleft", c("KM", "EKM"), lty=1:2, col=1, lwd=2, bty='n')
```



The EKM plot is complex: dashed are the EKM along with confidence itervals, solid lines are the KM stratified by enrollment bilirubin. The EKM bias for normal and moderate is large, but confidence intervals differ little between the robust and asymptotic se (latter not shown).

As shown above, more often than not the robust IJ variance is not needed for the KM curves themselves. However, they are the central computation for psuedovalues, which are steadily increasing in popularity. Let $U_k(t)$ be the IJ for observation k at time t. This is a vector, but for multi-state data it will be a matrix with 1 column per state. Likewise let $C(t)$ and $A(t)$ be influence vectors for the cumulative hazard and the area under the curve at time t. Let $h(t)$ be

the hazard increment at time t. Then

$$h(t) = \frac{\sum_i w_i dN_i(t)}{\sum_i w_i Y_i(t)} \tag{2.1}$$

$$\frac{\partial h(t)}{\partial w_k} = \frac{dN_i(t) - Y_k(t)h(t)}{\sum_i w_i Y_i(t)} \tag{2.2}$$

$$= C_k(t) - C_k(t-)$$

$$S(t) = \prod_{s \le t} 1 - h(s) \tag{2.3}$$

$$= S(t-)[1 - h(t)]$$

$$U_k(t) = \frac{\partial S(t)}{\partial w_k} \tag{2.4}$$

$$= U_k(t-)[1 - h(t)] - S(t-)\frac{dN_i(t) - Y_k(t)h(t)}{\sum_i w_i Y_i(t)} \tag{2.5}$$

$$U_k(t) = \frac{\partial \prod_{s \le t} 1 - h(s)}{\partial w_k}$$

$$= \sum_{s \le t} \frac{S(t)}{1 - h(s)} \frac{dN_i(s) - Y_k(s)h(s)}{\sum_i w_i Y_i(s)}$$

$$= S(t) \sum_{s \le t} \frac{dN_i(s) - Y_k(s)h(s)}{[1 - h(s)] \sum_i w_i Y_i(s)} \tag{2.6}$$

$$= S(t) \int_0^t \frac{\partial \log[1 - h_k(s)]}{w_k} d\overline{N}(s) \tag{2.7}$$

The simplest case is $C_k(\tau)$ at a single user requested reporting time τ, i.e., a simple use of the pseudo routine. The obvious code will update all n elements of C at each event time, an $O(nd)$ computation where d is the number of unique event times $\le \tau$. For most data d and n grow together, and $O(n^2)$ computations are something we want to avoid at all costs, a large data set will essentially freeze the computer.

The code solution is to create a running total z of the right hand term of (2.2), which applies to all subjects at risk. Then the leverage for an observation at risk over the interval (a, b) is

$$z(t) = \sum_{s \le t} \frac{h(s)}{\sum_i w_i Y_i(s)}$$

$$C_k(\tau) = fracN_k(\tau) \sum_i w_i Y_i(b) + z(\min(a, \tau)) - z(\min(b, \tau))$$

In R code this becomes an indexing problem. We can create 3 n by m = number of reporting times matrices that point to the correct elements of the vectors `c(0, 1/n.risk)` and `c(0, n.event/n.risk^2)` obtained from the survfit object.

The computation for survival starts out exactly the same if using the exponential estimate $S(t) = \exp(-\Lambda(t))$, otherwise do the same computation but using the rightmost term of equation (2.6). At the end multiply the column for reporting time τ by $-S(\tau)$, for each τ.

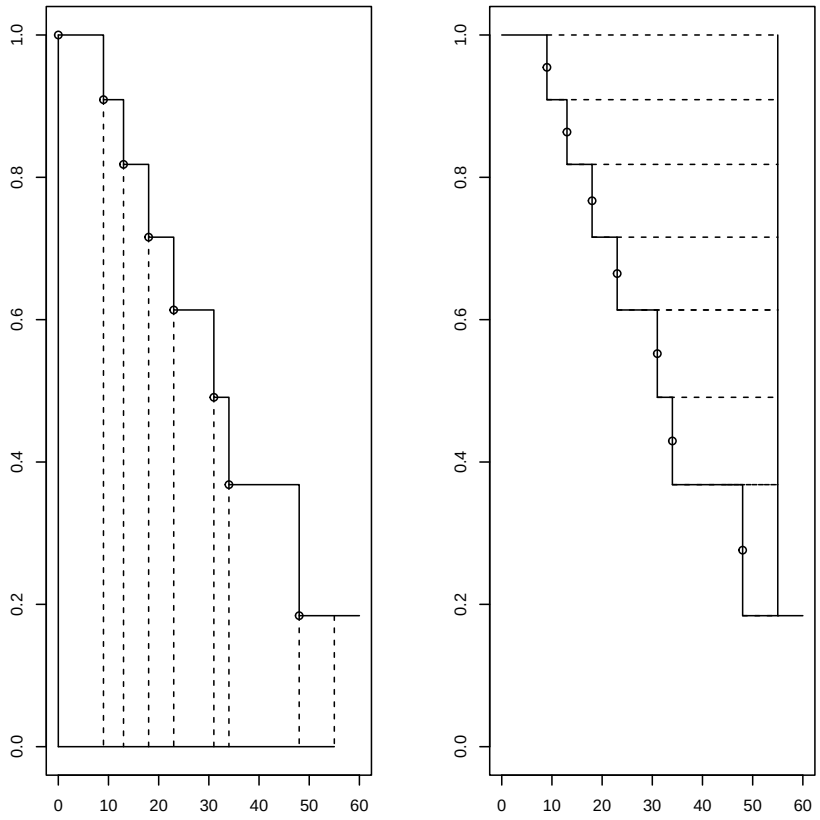


Figure 2.1: Two ways of looking at the AUC

There are two ways to look at the influence on the sojuourn time, which is calculated as the area under the curve. They are shown in the figure 2.1 for an AUC at time 55.

The first uses the obvious rectangle rule. The leverage of observation i on AUC(55) is the sum of the leverage of observation i on the height of each of the circles times the width of the associated rectangle. The leverage on the height of each circle are the elements of U. The second figure depends on the fact that the influence on the AUC must be the same as the influence on 55-AUC. It will be the influence of each observation on the length of each circled segment, times the distance to 55. This is closely related to the asymptotic method used for the ordinary KM, which assumes that the increments are uncorrelated.

Using the first approach, let Let $w_0, w_1, \ldots, w_m$ be the widths of the $m+1$ rectangles under the survival curve $S(t)$ from 0 to τ, and $d_1, \ldots, d_m$ the set of event times that are $\leq \tau$. Further,

let u_{ij} be the additive elements that make up U as found in the prior formulas, i.e.,

$$\begin{aligned}
U_{ik} &= S(d_k) \sum_{j=1}^{k} u_{ij} \\
u_{ij} &= \frac{\partial \log(1 - h(d_j))}{\partial w_i} \text{ KM} \\
&= -\frac{dN_i(d_j) - Y_i(d_j)h(d_j)}{[1 - h(d_j)]\overline{Y}(d_j)} \\
u^*_{ij} &= \frac{\partial - h(d_j)}{\partial w_i} \text{ exp(chaz)} \\
&= -\frac{dN_i(d_j) - Y_i(d_j)h(d_j)}{\overline{Y}(d_j)}
\end{aligned}$$

The second varianct holsd when using the exponential form of S.

Finally, let $A(a, b)$ be the integral under S from a to b.

$$A(0, \tau) = w_0 1 + \sum_{j=1}^{m} w_j S(d_j) \tag{2.8}$$

$$nonumber \tag{2.9}$$

$$\begin{aligned}
\frac{\partial A(0, \tau}{\partial w_k} &= \frac{\partial A(d_1, \tau}{\partial w_k} \\
\frac{\partial A(d_1, \tau}{\partial w_k} &= \sum_{j=1}^{m} w_j U_{kj} \\
&= \sum_{j=1}^{m} w_j S(d_j) \sum_{i=1}^{j} u_{ki} \\
&= \sum_{i=1}^{m} u_{ki} \left[\sum_{j=i}^{m} w_j S(d_j) \right] \\
&= \sum_{i=1}^{m} A(d_i, \tau) u_{ki}
\end{aligned} \tag{2.10}$$

This is a similar sum to before, with a new set of weights. Any given (time1, time2) observation involves u terms over that (time1,time2) range, we can form a single cumulative sum and use the value at time2 minus the value at time1. A single running total will not work for multiple τ values, however, since the weights depend on τ. Write $A(d_i, \tau) = A(d_1, \tau) - A(d_1, d_i)$ and separate the two sums. For the first $A(d_1, \tau)$ can be moved outside the sum, and so will play the same role as $S(d)$ in U, the second becomes a weighted cumulative sum with weights that are independent of tau.

Before declaring victory with this last modification take a moment assess the computational impact of replacing $A(d_i, \tau)$ with two terms. Assume m time points, d deaths, and n observations.

Method 1 will need to $O(d)$ to compute the weights, $O(d)$ for the weighted sum, and $O(2n)$ to create each column of the output for (time1, time2) data, for a total of $O(2m(n+d))$. The second needs to create two running sums, each $O(d)$ and $O(4n)$ for each column of output for $O(2d+4mn)$. The more 'clever' appoach is always slightly slower! (The naive method of adding up rectangles is much slower than either, however, because it needs to compute U at every death time.)

Variance Efficient computation of a the variance of the cumulative hazard within survfit is more complex, since this is computed at every event time. We consider three cases below.

Case 1: The simplest case is unclustered data without delayed entry; not (time1,time2) data.

- Let $W = \sum_i w_i^2$ be the sum of case weights for all those at risk. At the start everyone is in the risk set. Let $v(t) = 0$ and and $z(t) = 0$ be running sums.
- At each event time t
  1. Update $z(t) = z(t-) - h(t)/\overline{Y}(t)$, the running sum of the right hand term in equation (2.2).
  2. For all $i=$ (event or censored at t), fill in their element in C_i, add $(w_i C_i)^2$ to $v(t)$, and subtract w_i^2 from W.
  3. The variance at this time point is $v(t) + Wz^2(t)$; all those still at risk have the same value of C_k at this time point.

Case 2: Unclustered with delayed entry. Let the time interval for each observation i be (s_i, t_i). In this case the contribution at each event time, in step 3 above, for those currently at risk is

$$\sum_j w_j^2 [z(t) - z(s_j)]^2 = z^2(t) \sum_j w_j^2 + \sum_j w_j^2 z^2(s_j) - 2z(t) \sum (w_j z(s_j)$$

This requires two more running sums. When an observation leaves the risk set all three of them are updated. In both case 1 and 2 our algorithm is $O(d+n)$.

Case 3: Clustered variance. The variance at each time point is now

$$\text{varNA(t)} = \sum_{\text{g}} \left(\sum_{\text{i} \in \text{g}} \text{w}_\text{i} \frac{\partial \text{NA(t)}}{\partial \text{w}_\text{i}} \right)^2 \tag{2.11}$$

The observations within a cluster do not necessarily have the same case weight, so this does not collapse to one of the prior cases. It is also more difficult to identify when a group is no longer at risk and could be moved over to the $v(t)$ sum. In the common case of (time1, time2) data the cluster is usually a single subject and the weight will stay constant, but we can not count on that. In a marginal structural model (Robbins) for instance the inverse probability of censoring weights (IPCW) will change over time.

The above, plus the fact that the number of groups may be far less than the number of observations, suggests a different approach. Keep the elements of the weighted grouped C vector, one row per group rather than one row per observation. This corresponds to the inner term of equation (2.11). Now update each element at each event time. This leads to an $O(gd)$

algorithm. A slight improvement comes from realizing that the increment for group g at a given time involves the sum of $Y_i(t)w_i$, and only updating those groups with a non-zero weight.

For the Fleming-Harrington update, the key realization is that if there are d death at a given timepoint, the increment in the cumulative hazard NA(t) is a sum of d 'normal' updates, but with perturbed weights for the tied deaths. Everyone else at risk gets a common update to the hazard. The basic computation and equations for C involve an small loop over d to create the increments, sort of like an aside in a stage play, but then adding them into C is the same as before. The basic steps for case 1–3 do not change.

The C code for survfit has adapted case 3 above, for all three of the hazard, survival, and AUC. A rationale is that if the data set is very large the user can choose `std.err =FALSE` as an option, then later use the residuals function to get values at selected times. When the number of deaths gets over 1000 is where we begin to really notice the $O(nd)$ slowdown, but a plot only needs 100 points to look smooth Most use cases will only need variance at a 1–3 points. Another reason is that for ordinary survival, robust variance is almost never requested unless there is clustering.

2.3 Aalen-Johansen

In a multistate model let the hazard for the jk transition be

$$\begin{aligned} \hat{\lambda}_{jk}(t) &= \frac{\sum_i dN_{ijk}(t)}{\sum_i Y_{ij}(t)} \\ &= \frac{\overline{N}_{jk}(t)}{\overline{Y}(t)} \end{aligned}$$

and gather terms together into the nstate by nstate matrix $A(t)$ with $A_{jk}(t) = \hat{\lambda}_{jk}(t)$, with $A_{jj}(t) = -\sum_{k \neq j} A_{jk}(t)$. That is, the row sums of A are constrainted to be 0.

The cumulative hazard estimate for the $j : k$ transition is the cumulative sum of $\hat{\lambda}_{jk}(t)$. Each is treated separately, there is no change in methods or formula from the single state model.

The estimated probability in state $p(t)$ is a vector with one element per state and the natural constraint that the elements sum to 1; it is a probability distribution over the states. Two estimates are

$$p(t) = p(0) \prod_{s \leq t} e^{A(s)} \tag{2.12}$$

$$\begin{aligned} p(t) &= p(0) \prod_{s \leq t} (I + A(s)) \\ &= p(0) \prod_{s \leq t} H(s) \end{aligned} \tag{2.13}$$

Here $p(0)$ is the estimate at the starting time point. Both $\exp(A(s))$ and $I + A(s)$ are transition matrices, i.e., all elements are positive and rows sum to 1. They encode the state transition probabilities at time s, the diagonal is the probability of no transition for observations in the given state. At any time point with no observed transitions $A(s) = 0$ and $H(s) = I$; wlog the

product is only over the discrete times s at which an event (transition) actually occured. For matrices $\exp(C)\exp(D) \neq \exp(C+D)$ unless $DC = CD$, i.e., equation (2.12) does not simplify.

Equation (2.13) is known as the Aalen-Johansen estimate. In the case of a single state it reduces to the Kaplan-Meier, for competing risks it reduces to the cumulative incidence (CI) estimator. Interestingly, the exponential form (2.12) is almost never used for survfit.formula, but is always used for the curves after a Cox model. Both of the forms obey the basic probability rules that $0 \leq p_k(t) \leq 1$ and $\sum_k p_k(t) = 1$; all probabilities are between 0 and 1 and the sum is 1. The analog of (2.13) has been proposed by some authors for Cox model curves, but it can lead to negative elements of $p(t)$ in certain cases, so the survival package does not support it.

The compuations are straightforward. The code makes two passes through the data, the first to create all the counts: number at risk, number of events of each type, number censored, etc. Since for most data sets the total number at risk decreses over time this pass is done in reverse time order since it leads to more additions than subtractions in the running number at risk and so less prone to round off error. (Unless there are extreme weights this is gilding the lily - there will not be round off error for either case.) The rule for tied times is that events happen first, censors second, and entries third; imagine the censors at $t + \epsilon$ and entries at $t + 2\epsilon$.

When a particular subject has multiple observations, say times of (0,10), (10,15) and (15,24), we don't want the output to count this as a "censoring" and/or entry at time 10 or 15. The `position` vector is 1= first obs for a subject, 2= last obs, 3= both (someone with only one row of data), 0=neither. This subject would be 1,0,0,2. The position vector is created by the `survflag` routine. If the same subject id were used in two curves the counts are separate for each curve; for example curves by enrolling institution in a multi-center trial, where two centers happened to have an overlapping id value.

A variant of this is if a given subject changed curves at time 15, which occurs when users are estimating the "extended Kaplan-Meier" curves proposed by Snapinn [2], in which case the subject will be counted as censored at time 15 wrt the first curve, and an entry at time 15 for the second. (I myself consider Snapinn's estimate to be statistically unsound.)

If a subject changed case weights between the (0,10) and (10,15) interval we do not count them as separate entry and exits for the weighted n.enter and n.censor counts. This means that the running total of weighted n.enter, n.event, and n.censor will not recreate the weighted n.risk value; the latter *does* change when weights change in this way. The rationale for this is that the entry and censoring counts are mostly used for display, they do not participate in further computations.

2.3.1 Data

The survfit routine uses the `survcheck` routine, internally, to verify that the data set follows an overall rule that each subject in the data set follows a path which is physically possible:

1. Cannot be in two places at once (no overlaps)
2. Over the interval from first entry to last follow-up, they have to be somewhere (no gaps).
3. Any given state, if entered, must be occupied for a finite amount of time (no zero length intervals).
4. States must be consistent. For an interval (t1, t2, B) = entered state B at time t2, the current state for the next interval must be B (no teleporting).

I spent some time thinking about whether I should allow the survfit routine to bend the rules, and allow some of these. First off, number 1 and 3 above are not negotiable; otherwise it becomes impossible to clearly define the number at risk. But perhaps there are use cases for 2 and/or 4?

One data example I have used in the past for the Cox model is a subject on a research protocol, over 30 years ago now, who was completely lost to follow-up for nearly 2 years and then reappeared. Should they be counted as "at risk" in that interim? I argued no, on the grounds that the were not at risk for an "event that would have been captured" in our data set — we would never have learned of a disease progression. Someone should not be in the denominator of the Cox model (or KM) at a given time point if they cannot be in the numerator. We decided to put them into the data set with a gap in their follow-up time.

But in the end I've decided no to bending the rules. The largest reason is that in my own experience a data set that breaks any of 1–4 above is almost universally the outcome of a programming mistake. The above example is one of only 2 non-error cases I can think of, in nearly 40 years of clinical research: I *want* the routine to complain. (Remember, I'd like the package to be helpful to others, but I wrote the code for me.) Second is that a gap can easily be accomodated by creating extra state which plays the role of a waiting room. The P(state) estimate for that new state is not interesting, but the others will be identical to what we would have obtained by allowing a gap. Instances of case 4 can often be solved by relabeling the states. I cannot think of an actual use case.

2.3.2 Counting the number at risk

For a model with k states, counting the number at risk is fairly straightforward, and results in a matrix with k columns and one row per time point. Each element contains the number who are currently in that state and not censored. Consider the simple data set shown in figure 2.2.

The obvious algorithm is simple: at any given time point draw a vertical line and count the number in each state who intersect it. At a given time t the rule is that events happen first, then censor, then entry. At time 2 subject 4 has an event and subject 3 enters: the number at risk for the four states at time point 2 is (4, 0, 0, 0). At time point $2 + \epsilon$ it is (4, 1, 0, 1). At time 9 subject 3 is still at risk.

The default is to report a line of output at each unique censoring and event time, adding rows for the unique entry time is an optional argument. Subject 5 has 4 intervals of (1,3b), (3,6+), (6,8b) and (8,11+): time 6 is not reported as a censoring time, nor as an entry time. Sometimes a data set will have been preprocessed by the user with a tool like survSplit, resulting in hundreds of rows for some subjects, most of which are 'no event here', and there is no reason to include all these intermediate times in the output. We make use of an ancillary vector `position` which is 1 if this interval is the leftmost of a subject sequence, 2 if rightmost, 3 if both. If someone had a gap in followup we would treat their re-entry to the risk sets as an entry, however; the survcheck routine will have already generated an error in that case.

The code does its summation starting at the largest time and moving left, adding and subtracting from the number at risk (by state) as it goes. In most data sets the number at risk decreases over time, going from right to left results in more additions than subtractions and thus less potential roundoff error. Since it is possible for a subject's intervals to have different case weights, the number at risk *does* need to be updated at time 6 for subject 5, though that time point is not in the output.

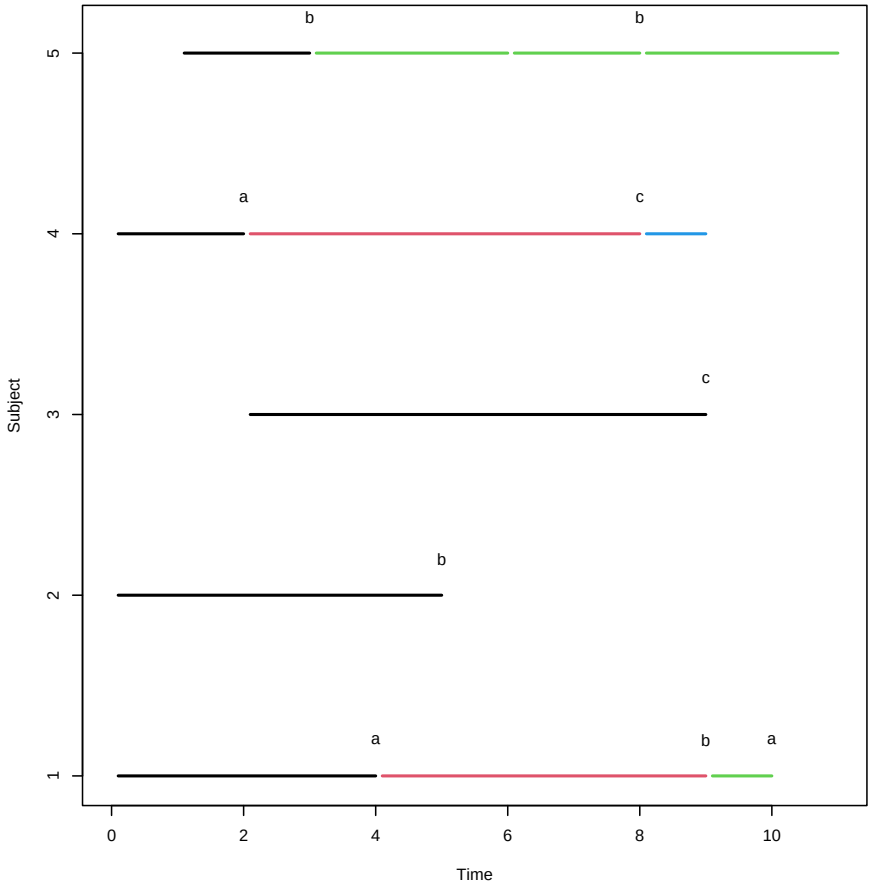


Figure 2.2: A simple multi-state data set. Colors show the current state of entry (black), a (red), b(green) or c (blue), letters show the occurrence of an event.

In the example outcome b can be a repeated event, e.g., something like repeated infections. At time 8 the cumulative hazard for event b will change, but the proability in state will not. It would be quite unusual to have a data set with some repeated states and others not, but the code allows it.

Consider the simple illness-death model with recovery. There are 3 states and 4 transtions in the model. The number at risk (n.risk), number censored (n.censor), probability in state (pstate) and number of entries (n.enter) components of the survfit call will have 3 columns, one per state. The number of events (n.event) and cumulative hazard (cumhaz) components will have 4 columns, one per transition, labeled as 1.2, 1.3, 2.1, and 2.3. In a matrix of counts with row= starting state and column = ending state, this is the order in which the non-zero elements appear. The order of the transitions is yet another consequence of the fact that R stores matrices in column major order.

There are two tasks for which the standard output is insufficient: drawing the curves and computing the area under the curve. For both these we need to know $t0$, the starting point for the curves. There are 4 cases:

1. This was specified by the user using `start.time`
2. All subjects start at the same time
3. All subjects start in the same state
4. Staggered entry in diverse states

For case 1 use what the user specified, of course. For (time, status) data, i.e. competing risks use the same rule as for simple survival, which is min(time, 0). Curves are assumed to start at 0 unless there are negative time values. Otherwise for case 2 and 3 use min(time1), the first time point found; in many cases this will be 0 as well.

The hardest case most often arises for data that is on age scale where there may be a smattering of subjects at early ages. Imagine that we had subjects at ages 19, 23, 28, another dozen from 30-40, and the first transition at age 41, 3 states A, B, C, and the user did not specify a starting time. Suppose we start at age=19 and that subject is in state B. Then p(19)= c(0,1,0), and AJ p(t) estimate will remain (0,1,0) until there is a B:A or B:C transition, no matter what the overall prevalence of inital states as enrollment progresses. Such an outcome is not useful. The default for case 4 is to use the smallest event time at time 0, with intial prevalence the distribution of state for those observations which are at risk at that time. The prevalence at the starting time is saved in the fit object as p0. The best choice for case 4 is a user specified time, however.

Since we do not have an estimate of p= probability in state before time 0, the output will not contain any rows before that time point. Note that if there is an event exactly at the starting time: a death at time 0 for competing risks, a transition at start.time, or case 4 above, that the first row of the pstate matrix will not be the same as p0. The output never has repeats of the same time value (for any given curve). One side effect of this is that a plotted curve never begins with a vertical drop.

2.4 Influence

Let $C(t)$ be the influence matrix for the cumulative hazard $\Lambda(t)$ and $U(t)$ that for the probability in state $p(t)$. In a multistate model with s states there are s^2 potential transitions, but only a few are observed in any given data set. The code returns the estimated cumulative hazard $\hat{\Lambda}$ as a matrix with one column for each $j:k$ transition that is actually observed, the same will be true for C. We will slightly abuse the usual subscript notation; read C_{ijk} as the ith row and the $j:k$ transition (column). Remember that a transition from state j to the same state can occur with multiple events of the same type. Let

$$\overline{Y}_j(t) = \sum_i Y_{ij}(t)$$

be the number at risk in state j at time t. Then

$$\begin{aligned} C_i(t) &= \frac{\partial \Lambda(t)}{\partial w_i} \\ &= C_i(t-) + \frac{\partial \lambda(t)}{\partial w_i} \\ \lambda_{jk}(t) &= \frac{\sum w_i dN_{ijk}}{\overline{Y}_j(t)} \qquad (2.14) \\ \frac{\partial \lambda_{jk}(t)}{\partial w_i} &= \frac{dN_{ijk}(t) - Y_{ij}(t)\lambda_{jk}(t)}{\overline{Y}_j(t)} \qquad (2.15) \\ V_c(t) &= \sum_i [w_i C_i(t)]^2 \qquad (2.16) \end{aligned}$$

At any time t, formula (2.15) captures the fact that an observation has influence only on potential transitions from its current state at time t, and only over the (time1, time2) interval it spans. At any given event time, each non-zero h_{jk} at that time will add an increment to only those rows of C representing observations currently at risk and in state j. The IJ variance is the weighted sum of squares. (For replication weights, where w=2 means two actual observations, the weight is outside the brackets. We will only deal with sampling weights.)

The (weighted) grouped estimates are

$$\begin{aligned} C_{gjk}(t) &= \sum_{i \in g} w_i C_{ijk}(t) \\ &= C_{gjk}(t-) + \sum_{i \in g} w_i \frac{dN_{ijk}(t) - Y_{ij}(t) lambda_{jk}(t)}{\overline{Y}_j(t)} \qquad (2.17) \\ &= C_{gjk}(t-) + \sum_{i \in g} w_i \frac{dN_{ijk}(t)}{n_j(t)} - \left(\sum_{i \in g} Y_{ij}(t) w_i \right) \frac{\lambda_{jk}(t)}{\overline{Y}_j(t)} \qquad (2.18) \\ & \qquad (2.19) \end{aligned}$$

A $j:k$ event at time t adds only to the jk column of C. The last line above (2.18) spits out the dN portion, which will normally be 1 or at most a few events at this time point, from the

second, which is identical for subjects at risk in group g. This allows us to split out the sum of weights. Let $w_{gj}(t)$ be the sum of weights by group and state, which is kept in a matrix of the same form as U and can be efficiently updated. Each row of the grouped C is computed with the same effort as a row of the ungrouped matrix.

Let C be the per observation influence matrix, W a diagonal matrix of per-observation weights, and B the n by g 01/ design matrix that collapses observations by groups. For instance, `B = model.matrix( factor(grp) -1)`. The the weighted and collapsed influence is $V = B'WC$, and the infinitesimal jackknife variances are the column sums of the squared elements of V, `colSums(V*V)`. A feature of the IJ is that the column sums of WC and of $B'WC$ are zero. In the special case where each subject is a group and the weight for a subject is constant over time, then one can collapse and then weight rather than weight and then collapse. The survfitci.c routine (now replaced by survfitaj.c) made this assumption, but unfortunately the code had no test cases with disparate weights within a group and so the error was not caught for several years. There is now a test case.

For C above, where each row is a simple sum over event times, formula (2.18) essentially does the scale + sum step at each event time and so only needs to keep one row per group. The parent routine will warn if any subject id is in two different groups. Doing otherwise makes no statistical sense, IMHO, so any instance if this is almost certainly an error in the data setup. However, someone may one day find a counterexample, hence a warning rather than an error.

For the probability in state $p(t)$, if the starting estimate $p(0)$ is provided by the user, then $U(0) = 0$. If not, $p(0)$ is defined as the distribution of states among those at risk at the first event time τ.

$$p_j = \frac{\sum w_i Y_{ij}(\tau)}{\sum w_i Y_i(\tau)}$$
$$\frac{\partial p_j}{\partial w_k} = \frac{Y_{ij}(\tau) - Y_i(\tau) p_j}{\sum w_i Y_i(\tau)}$$

Assume 4 observations at risk at the starting point, with weights of (1, 4, 6, 9) in states (a, b, a, c), respectively. Then $p(0) = (7/20, 4/20, 9/20)$ and the unweighted U matrix for those observations is

$$U(0) = \begin{pmatrix} (1-7/20)/20 & (0-4/20)/20 & (0-9/20)/20 \\ (0-7/20)/20 & (1-4/20)/20 & (0-9/20)/20 \\ (1-7/20)/20 & (0-4/20)/20 & (0-9/20)/20 \\ (0-7/20)/20 & (0-4/20)/20 & (1-9/20)/20 \end{pmatrix}$$

with rows of 0 for all observations not at risk at the starting time. Weighted column sums are $wU = 0$.

The AJ estimate of the probablity in state vector $p(t)$ is defined by the recursive formula $p(t) = p(t-)H(t)$. Remember that the derivative of a matrix product AB is $d(A)B + Ad(B)$ where $d(A)$ is the elementwise derivative of A and similarly for B. (Write out each element of

the matrix product.) Then ith row of U satisfies

$$\begin{aligned} U_i(t) &= \frac{\partial p(t)}{\partial w_i} \\ &= \frac{\partial p(t-)}{\partial w_i} H(t) + p(t-)\frac{\partial H(t)}{\partial w_i} \\ &= U_i(t-)H(t) + p(t-)\frac{\partial H(t)}{\partial w_i} \end{aligned} \tag{2.20}$$

The first term of 2.20 collapses to ordinary matrix multiplication, the second to a sparse multiplication. Consider the second term for any chosen observation i, which is in state j. This observation appears only in row j of $H(t)$, and thus dH is zero for all other rows of $H(t)$ by definition. If observation i is not at risk at time t then dH is zero. The derviative vector thus collapses an elementwise multiplication of $p(t-)$ and the appropriate row of dH, or 0 for those not at risk.

Let $Q(t)$ be the matrix containing $p(t-)Y_i(t)dH_{j(i)\cdot}(t)/dw_i$ as its ith row, $j(i)$ the state for row j. Then

$$\begin{aligned} U(t_2) &= U(t_1)H(t_2) + Q(t_2) \\ U(t_3) &= \left(U(t_1)H(t_2) + Q(t_2)\right) H(t_3) + Q(t_3) \\ &\vdots = \vdots \end{aligned}$$

Can we collapse this in the same way as C, retaining only one row of U per group containing the weighted sum? The equations above are more complex due to matrix multiplication. The answer is yes, because the weight+collapse operation can be viewed as multiplication by a design matrix B on the left, and matrix multiplication is associative and additive. That is $B(U+Q) = BU+BQ$ and $B(UH) = (BU)H$. However, we cannot do the rearrangment that was used in the single endpoint case to turn this into a sum, equation (2.6), as matrix multiplication is not commutative.

For the grouped IJ, we have

$$\begin{aligned} U_g(t) &= \sum_{i \in g} w_i U_i(t) \\ &= U_g(t-)H(t) + \sum_{i \in g} w_i Y_{ij}(t) \frac{\partial H(t)}{\partial w_i} \end{aligned}$$

The above has the potential to be a very slow computation. If U has g rows and p columns, at each event time there is a matrix multiplication UH, creation of the H and H' matrices, addition of the correct row of H' to each row of U, and finally adding up the sqared elements of U to get a variance. This gives $O(d(gp^2 + 2p^2 + gp + gp))$. Due to the matrix multiplication, every element of U is modified at every event time, so there is no escaping the final $O(gp)$ sum of squares for each column. The primary gain is to make g small.

At each event time t the leverage for the AUC must be updated by $\delta U(t-)$, where δ is the amount of time between this event and the last. At each reporting which does not coincide with an event time, there is a further update with δ the time between the last event and the reporting time, before the sum of squares is computed. The AUC at the left endpoint of the curve is 0 and likewise the leverage. If a reporting time and event time coincide, do the AUC first.

At each event time

1. Update $w_{gj}(\tau) = \sum_{i \in g} Y_{ij}(\tau) w_i$, the number currently at risk in each group.
2. Loop across the events ($dN_{ijk}(\tau) = 1$). Update C using equation (2.21) and create the transformation matrix H.
3. Compute $U(\tau) = U(\tau-)H$ using sparse methods.
4. Loop a second time over the $dN_{ijk}(\tau)$ terms, and for all those with $j \neq k$ apply (2.22) and(2.23).
5. For each type of transition jk at this time, apply equation (2.24), and if $j \neq k$, (2.25) and (2.26).
6. Update the variance estimates, which are column sums of squared elements of C, U, and AUC leverage.

$$\lambda_{jk}(\tau) = \frac{\sum_i dN_{ijk}(\tau)}{\sum_i w_i Y_{ij}(\tau)}$$

$$C_{g(i)jk}(\tau) = C_{gjk}(\tau-) + w_{g(i)} \frac{dN_{ijk}(\tau)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.21}$$

$$U_{g(i)j}(\tau) = U_{gj}(\tau-) - w_{g(i)} \frac{dN_{ijk}(\tau) p_j(\tau-)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.22}$$

$$U_{g(i)k}(\tau) = U_{gk}(\tau-) + w_{g(i)} \frac{dN_{ijk}(\tau) p_j(\tau-)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.23}$$

$$C_{ghk}(\tau) = C_{gjk}(\tau-) - w_{gj}(\tau) \frac{\lambda_{jk}(\tau)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.24}$$

$$U_{gj}(\tau) = U_{gj}(\tau-) + w_{gj}(\tau) \frac{\lambda_{jk}(\tau) p_j(\tau-)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.25}$$

$$U_{gk}(\tau) = U_{gk}(\tau-) - w_{gj}(\tau) \frac{\lambda_{jk}(\tau) p_j(\tau-)}{\sum_i w_i Y_{ij}(\tau)} \tag{2.26}$$

In equations (2.21)–(2.23) above $g(i)$ is the group to which observed event $dN_i(\tau)$ belongs. At any reporting time there is often 1 or at most a handful of events. In equations (2.24)–(2.26) there is an update for each row of C and U, each row a group. Most often, the routine will be called with the default where each unique subject is their own group: if only 10% of the subjects are in group j at time τ then 90% of the w_{gj} weights will be 0. It may be slightly faster to test for w positive before multiplying by 0?

For the 'sparse multiplication' above we employ simple approach: if the $H - I$ matrix has only 1 non-zero diagonal, then U can be updated in place. For the NAFLD example in the survival vignette, for instance, this is true for all but 161 out of the 6108 unique event times (5%). For the remainder of the event times it suffices to create a temporary copy of $U(t-)$.

Chapter 3

Cox model

3.1 Specials

R formulas allow for special terms. When parsing the coxph formula using the standard `terms` function, the optional argument `specials= 'strata'` will cause the returned object to specially mark any strata terms. We then deal with this properly in the fit; a stratum is not just another covariate.

The advent of hundreds of libraries, more users have been using double colon notation ensure that they actually invoke the function that they think they are invoking, e.g.,

```
> survival::coxph(survival::Surv(time, status) ~
                        age + sex + survival::strata(inst), data=lung)
```

Saying library(tidyverse), alone, attaches 60 (19 packages + 41 name spaces). A problem with this is that the terms function searches for and recognizes the character string "strata" and so will not detect "survival::strata" as special. At least one package assigns the result of `strata::(group)` to a temporary variable and then uses that variable in their formula, i.e.; it would not suffice to add survival::strata to my list of specials.

In version 3.7 of the package I made a valiant effort to combat this by essentially re-writing the user's formula. It worked pretty well, but I've now thrown in the towel and ensured that all of my special functions return a recognized class and key off of that. The functions that were special were Surv, strata, cluster, pspline, frailty, ridge and tt.

- The pspline, frailty, and ridge functions (and frailty subfunctions like frailty.t) already had the class coxph.penalty. It suffices to recognize that rather use terms.
- The Surv function also has a class. We never use it on the right hand side, however, so have never used specials to find it.
- The tt() function did not have to change, since I never created a dummy function in the package to correspond to it, it only exists "on the fly" within the package and will always be recognized first.
- An update is to do the same with cluster(). Using this within a formula is on its way to deprecation in any case.

- That leaves strata. It appears in lots and lots of places, and all will need to be changed.

3.2 Missing values

When the formula is a list, then the possiblility exists that different transitions will have different covariates. If a particular row of the data is missing variable x3 say, but x3 is only used in transitions for which that row is not at risk, then there is no need to remove said observation. Assume for instance that one of the states was surgical recection of a cancer, then pathology results will only be available after that time point.

This leads to a simple dichotomy: if the model formula is simple then call the model.frame matrix in the usual way. If not, use na.pass to retain all observations until a proper determination can be made. To make that determination we need the current state for each data row, which is either obtained or validated by survcheck.

The model.frame and the tmap object suffice for missings, we don't need to create the X matrix. (Someone might have saved X and not the model frame, though, in which case we can use cmap and X).

Steps for a multi-formula model

1. Get the raw model frame, no missings removed
2. Extract Y, istate if present, weights if present
3. Use survcheck2 to get the

3.3 Absolute risk

3.3.1 Cumulative hazard and survival curves

The survival curves from a fitted coxph model default to the Breslow estimate = exp(cumulative hazard). We also support the simple product-limit estimate (for which I have concerns, see more in the multi-state section), and the Kalfleisch-Prentice modification of the product-limit form.

Predicted survival curves are always for a particular set of covariates. We do not have much sympathy for the notion of *the* baseline hazard, which textbooks define as the predicites survival for all x equal to zero, since it can easily lead to overflow or underflow errors in the exponential function. The coxph object includes a `means` component, which contains a workable vector of centering constants, i.e., a set for which the exponential will not be a problem. The labels of "means" is historical, as individual elements of the vector are not necessarily a mean, 0/1 covariates for instance are given a centering constant of zero. Let c be that centering vector, and define $r(z; c) = \exp[(z - c)\beta]$ be the recentered risk score for a hypothetical subject with

covariate vector z. The predicted cumulative hazard for a new subject with covariate vector z is

$$\begin{aligned}
\hat{\lambda}(t;z) &= \frac{\sum_i w_i dN_i(t)}{\sum_i Y_i(t) w_i r(x_i;z)} \\
&= e^{(z-c)\beta} \hat{\lambda}(t;c) \\
\Lambda(t;z) &= \int_0^t \hat{\lambda}(s;z) ds \\
&= e^{(z-c)\beta} \Lambda(t;c) \qquad (3.1) \\
\hat{S}(t;z) &= e^{-\Lambda(t;z)} \\
&= \hat{S}(t;c)^{e^{(z-c)\beta}} \qquad (3.2)
\end{aligned}$$

Per above we can compute the survival curve for any covariate z from the "baseline" curve for $x = c$. Any vector c that will ensure that the above can be calculated without roundoff error will suffice for safety, the survival code uses the mean component from the fit. To be clear, for most data sets use of $c = 0$ as a centering constant will be just fine; it is sufficient to keep the $r(x_i, c)$ values far enough away from the `double.min.exp` and `double.max.exp` components of `.Machine` (around 1000 on many machines) such that weighted means and variances, using $r(x_i, c)$ as the weights, do not lose precision. One case we have seen is where someone is looking at a seasonal effect and uses a Date variable for the covariate. Say that the study were centered in 2022, then the mean x value is approximately 19000. (Even then we would be okay if β is small enough).

If a user requested the product-limit form, then the predicted survival is

$$\begin{aligned}
\hat{S}(t;z) &= \prod_{s \le t} [1 - lhat(t;z)] \\
&\neq \left(\prod_{s \le t} [1 - lhat(t;c)] \right)^{e^{(z-c)\beta}}
\end{aligned}$$

A key issue is that if $r(x_i; z)$ is very small, e.g., when computing the predicted curve for a very high risk subject, then $1 - \hat{\lambda}(t;z)$ may be negative, leading to an invalid survival estimate. For this reason the survival package does not support using a product-limit form.

An alternative estimate of $\hat{\lambda}$ proposed by Kalbfleisch and Prentice avoids this issue of an invalid estimate. Also, it collapses to the Kaplan-Meier when $\beta = 0$, something that some consider to be a major plus. In this author's opinion it is a "meh"; a numerical difference that is tiny until the number at risk falls below 20, and even then much smaller than the se of the estimate. For ordinary survival, Fleming and Harrington showed the exp(-cumulative hazard) actually had a smaller MSE than the KM, but again the difference is so small that no one really cares. For completeness, our software allows the KP as an option. The variance of $\log(S)$, however, is assumed to be exactly the same as that of the exponential estimate, i.e., the variance of Λ. Creating a special variance estimate is just too much work

The KP approach computes a multiplicative increment $z(t)$ as the solution to equation (3.3). If there is a single event at time t, this reduces to the closed form of (3.4), where d is shorthand for the index of the death at that time point. Otherwise the code uses simple bisection on the

interval [0,1), which must contain the solution. A bit of algebra confirms that the final estimate is mathematically independent of the centering vector c.

$$\sum w_i Y_i(t) r(x_i; c) = \sum dN_i(t) w_i \frac{r(x_i; c)}{1 - z(t)^{r(x_i;c)}} \tag{3.3}$$

$$z(t) = 1 - w_d \frac{r(x_d; c)}{\left(\sum w_i Y_i(t) r(x_i; c)\right)^{1/r(x_d;c)}} \tag{3.4}$$

$$\hat{S}(t; z) = \left(\prod_{s \le t} z(t) \right)^{e^{(z-c)\beta}} \tag{3.5}$$

3.3.2 Variance

The variance of $\hat{\Lambda}(t; z)$ has two terms, the first an analog to the variance increment for the Nelson-Aalen estimate, the second accounts for the variance of $\hat{\beta}$. Let $\tilde{\mu}(t; z)$ be the cumulative weighted differnce between z and the running mean

$$\tilde{\mu}(t; z) = \int_0^t (z - \overline{x}(s)) d\Lambda(s; z) e^{(z-c)\beta} \int_0^t (z - \overline{x}(s)) d\Lambda(s; c) \tag{3.6}$$

This is a weighted average distance between z and the center of the data. The variance is

$$\mathrm{var}\hat{\Lambda}(\mathrm{t}; \mathrm{z}) = \left[\mathrm{e}^{2(\mathrm{z}-\mathrm{c})\beta} \int_0^{\mathrm{t}} \frac{\hat{\lambda}(\mathrm{s}; \mathrm{c})}{\sum_{\mathrm{i}} \mathrm{w_i Y_i(s) r(x_i; c)}} \right] + \tilde{\mu}(\mathrm{t}; \mathrm{z})' \mathrm{V} \tilde{\mu}(\mathrm{t}; \mathrm{z})$$

The `agsurv` routine returns individual terms of the integral as a vector `varhaz`, the first term of the variance is then a weighted cumulative sum. A matrix with a row for each time point is returned as `xbar`, containing the product of $\hat{\lambda}(t; c)$ with $\overline{x}(t)$. (Note the $\overline{x}$ itself does not depend on centering.) The routine also returns the vector $\lambda(t; c)$.

The second term of the variance cannot be treated as a cumulative sum of squares, we need to first add and *then* multiply. In the pseudo code below, hazard and xbar are results from agsurv and vmat the variance matrix from the coxph model.

```
> temp1 <- outer(hazard, z, '*') - xbar
> temp2 <- apply(temp1, 2, cumsum)
> v2 <-  rowSums((temp2 %*% vmat)* temp2)
```

- temp1 is a matrix where each row is $\lambda(t; c)(z - \overline{x}(t))$
- each row of temp2 is the integral up to that point
- element i of v2 contains the result of `temp2[i,] %*% vmat %*% t(temp2[i,])`. The use of both matrix multiplication and elementwise multiplication, in the right order, is important.

We can offset the use of $\lambda(t; c)$ instead of $\lambda(t; z)$ at the end, but creation and summation of the $z - \overline{x}$ matrix needs to be done separately for each z.

3.3.3 Time-dependent coefficients

Look at an example from the time-dependent vignette.

```
> vet2 <- survSplit(Surv(time, status) ~ ., data= veteran, cut=c(90, 180),
                    episode= "tgroup", id="id", start="time1", end="time2")
> vfit2 <- coxph(Surv(time1, time2, status) ~ trt + prior +
                    I(karno/10):strata(tgroup), data=vet2)
> vfit2
Call:
coxph(formula = Surv(time1, time2, status) ~ trt + prior + I(karno/10):strata(tgroup),
    data = vet2)

                                        coef exp(coef)
trt                                  -0.011025  0.989035
prior                                -0.006107  0.993912
I(karno/10):strata(tgroup)tgroup=1 -0.487550  0.614129
I(karno/10):strata(tgroup)tgroup=2  0.080504  1.083834
I(karno/10):strata(tgroup)tgroup=3 -0.083487  0.919903
                                     se(coef)      z
trt                                  0.189062 -0.058
prior                                0.020355 -0.300
I(karno/10):strata(tgroup)tgroup=1  0.062217 -7.836
I(karno/10):strata(tgroup)tgroup=2  0.128228  0.628
I(karno/10):strata(tgroup)tgroup=3  0.146204 -0.571
                                         p
trt                                  0.953
prior                                0.764
I(karno/10):strata(tgroup)tgroup=1 4.64e-15
I(karno/10):strata(tgroup)tgroup=2    0.530
I(karno/10):strata(tgroup)tgroup=3    0.568

Likelihood ratio test=63.04  on 5 df, p=2.857e-12
n= 225, number of events= 128
> cdata <- data.frame(time1 = rep(c(0,90,180), 2),
                      time2 =  rep(c(90,180, 365), 2),
                      status= rep(0,6),   #necessary, but ignored
                      tgroup= rep(1:3, 2),
                      trt  =  rep(1,6),
                      prior=  rep(0,6),
                      karno=  rep(c(40, 75), each=3),
                      curve=  rep(1:2, each=3))
> cdata
  time1 time2 status tgroup trt prior karno curve
1     0    90      0      1   1     0    40     1
2    90   180      0      2   1     0    40     1
3   180   365      0      3   1     0    40     1
```

```
4     0    90       0       1    1      0     75      2
5    90   180       0       2    1      0     75      2
6   180   365       0       3    1      0     75      2
> vsurv <- survfit(vfit2, newdata=cdata, id=curve)
```

In the code `slist` will have one curve per stratum, 3 in the above case. The `id` for this new curve has nothing to do with any id variable in the original fit, here it is an id for the resulting curve. The algorithm is executed once per new curve

- For each row of data corresponding to this id, pluck off the appropriate rows of slist[[stratum of this row]], i.e. the time, n.censor, n.event, etc components.
- Glue these together into a new curve

The (time1, time2) values in the new data correspond to the time values in the chosen stratum; in this case those are consistent, but one could have strata on different time scales. The time values in the new curve will be

- time values for the stratum of row 1
- (time2[1] - time1[2]) + time values for the stratum of row 2
- (time2[2] - time2[3]) + time values for the stratum of row 3
- ...

The idea is that if stratum 1 went from 30 to 100 and stratum 2 from 150 to 200, we want our new time scale to be from 30 to 150. In the case above these splicing constants are all zero, which is what I expect, but users do things I have never thought of.

The underlying routine agsurv returns results for the hazard function centered at c = the means component of the coxph result. But from this point forward, we need to do separate computations for each new id, a simple loop since there are unlikely to be many separate id values. Let j be the rows of newdata that coresond to the first id. For each of these rows pull off the hazard increments, increments of term 1 of the variance, and the $\tilde{\mu}$ values of (3.6); multiple the first and third by risk[j], the second by risk[j] squared, and save them. When that is done,

- Compute the cumulate hazard, cumsum(unlist(hazard increments))
- Exponential survial is exp(-cum haz), KP survival is cumprod of the KP contributions
- The first term of the variance is again cumsum(unlist(var1))
- For the second term we need the tmu matrix, using cumsum(unlist) for each column, separately. Then the quadratic form can be computed.

I am moderately confident of the statisical theory for this in the case of a time-dependent covariate, where each new id covers the same range of time, without holes or overlaps. For the more complex cases I am much less certain, and would myself use a jackknife or boostrap.

Chapter 4

Multistate hazard model

4.1 Absolute risk

To begin, first remember that predicitons from a Cox model are always for *someone*, that is, some particular covariate vector z. This includes survival curves and other estimates of absolute risk. Put aside the notion of "the" baseline survival curve from the start. For a multistate model the predicted curve is a matrix product

$$\begin{aligned} r_i(z) &= \exp((x_i - z)\beta) \\ \lambda_{jk}(t; z) &= \frac{\sum_i w_i dN_{ijk}(t)}{\sum_i w_i r_i(z) Y_{ij}(t)} \\ p(t; z) &= p(0) \prod_{s \le t} e^{A(s;z)} \\ A(s; z) &= \begin{pmatrix} \square & \lambda_{12}(s; z) & \lambda_{13}(s; z) & \dots \\ \lambda_{21}(s; z) & \square & \lambda_{23}(s; z) & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{aligned}$$

In the above j, k are states, i is an observation. The diagonal elements are such that row sums are zero.

The mstate package uses the first order Taylor series $\exp(A) \approx I + A$, for $p(t; z)$, which is parallel to the formula in the Aalen-Johansen. We do not, for a couple of reasons.

- If the chosen x corresponds to a particularly high relative hazard, then the diagonal element if $I + A$ can be negative, and $I + A$ is no longer a valid transition matrix.
- Consistency with the single state Cox model's Breslow estimate, which uses the exponential. The I+A approximation has never been seriously proposed for the simple Cox model, probably because of the point just above.

For matrices, $\exp(A)\exp(B) \neq \exp(A + B)$ unless $AB = BA$, which does not hold for this application: we cannot avoid the matrix multiplication. For transition matrices such as these $\exp(A(s))$ will have row sums of 1 and all elements non-negative, numerically they are very well

Figure 4.1: State space for the NAFLD data. The dashed lines are transitions that would not be observed if monitoring were continuous, but are present in the observed data.

behaved arguments to the matrix exponential function. Per section 7.1, for non-tied death times (only 1 event at this time) A and $\exp(A)$ have a simple closed form and computation will be fast.

For simple survival the above reduces to $\exp(-\Lambda(t;z)$, and in that case the variance of the cumulative hazard estimate has been derived by (ref), not using the IJ as

$$v(t;z) = d'\mathcal{I}^- 1d + \int_0^t \frac{\sum_i dN_i(t)}{\sum_i Y_i(t) r_i(z)}$$
$$d(t;z) = \int_0^t (\overline{x}(s) - z) d\Lambda(s;z)$$

4.2 Shared hazards

The vignette on shared coefficients and shared hazards in the survivalVignettes package gives important background on this problem.

As a first example look at nafld data, the state space is shown in figure 4.1. The code below sets up the data and fits a first model.

```
> ndata <- nafld1
> ndata$nafld <- with(ndata, 1*(id==case.id))
> caseid <-  subset(nafld1, id==case.id)$id
> tdata1 <- subset(ndata, case.id %in% caseid)
> # the endpoints of interest
> enddata <- subset(nafld3, (id %in% tdata1$id) &
                            (event %in% c("diabetes","dyslipidemia",  "htn")))
> ndata <- tmerge(tdata1[c(1,2,3,6,10)], tdata1, id=id,
                  death= event(futime, status),
                  options=list(tstartname="day1", tstopname="day2"))
> ndata <- tmerge(ndata, enddata, id=id, mcount=cumtdc(days),
                  mevent= cumevent(days))
> ndata$cstate <- factor(ndata$mcount, 0:3, mstate[1:4])
> ndata$event  <- with(ndata, factor(ifelse(death==1, 4, mevent), 0:4,
                       c("censor", mstate[-1])))
> ndata$age1 <- ndata$age + ndata$day1/365.25   # analysis on age scale
> ndata$age2 <- ndata$age + ndata$day2/365.25
> check <- survcheck(Surv(age1, age2, event) ~ 1, ndata, id=id, istate=cstate)
> ndata$notzero <- 1*(ndata$cstate != "0mc")
> nfit <- coxph(list(Surv(age1, age2, event) ~ nafld + male,
                     1:2 + 1:3 + 1:4 ~ nafld + male / common,
                     2:3 + 2:4 ~ nafld + male/common,
                     c(2,3,4):5 ~ notzero,
                     0:5  ~ 1 / common),
```

```
                data=ndata, id= id, istate= cstate)
> print(nfit, digits=2)
Call:
coxph(formula = list(Surv(age1, age2, event) ~ nafld + male,
    1:2 + 1:3 + 1:4 ~ nafld + male/common, 2:3 + 2:4 ~ nafld +
        male/common, c(2, 3, 4):5 ~ notzero, 0:5 ~ 1/common),
    data = ndata, id = id, istate = cstate)


1:2, 1:3, 1:4  coef exp(coef) robust se    z      p
        nafld 0.938     2.554     0.069 13.6 <2e-16
        male  0.185     1.203     0.049  3.7  2e-04


2:3, 2:4  coef exp(coef) robust se   z      p
   nafld 0.538     1.712     0.057 9.3 <2e-16
   male  0.222     1.249     0.051 4.4  1e-05


3:4      coef exp(coef) robust se   z     p
  nafld 0.484     1.623     0.070 6.9 6e-12
  male  0.123     1.131     0.067 1.8  0.07


1:5     coef exp(coef) robust se   z     p
  nafld 0.50      1.65      0.23 2.2 0.026
  male  0.37      1.44      0.12 3.0 0.003


2:5         coef exp(coef) robust se    z     p
  nafld     0.49      1.63      0.15  3.2 0.001
  male      0.43      1.54      0.13  3.4 8e-04
  notzero* -0.29      0.75      0.13 -2.3 0.022


3:5         coef exp(coef) robust se    z     p
  nafld     0.54      1.72      0.11  5.0 7e-07
  male      0.37      1.44      0.10  3.7 3e-04
  notzero* -0.18      0.84      0.12 -1.5   0.1


4:5        coef exp(coef) robust se   z     p
  nafld    0.10      1.11      0.10 1.0   0.3
  male     0.16      1.18      0.10 1.6   0.1
  notzero* 0.59      1.81      0.12 4.8 2e-06
```

```
 States:  1= 0mc, 2= 1mc, 3= 2mc, 4= 3mc, 5= death
 (*) = coef for proportional baselines

Likelihood ratio test=533  on 17 df, p=<2e-16
n= 20972, unique id= 16400, number of events= 5881
> nfit$cmap
        1:2 1:3 2:3 1:4 2:4 3:4 1:5 2:5 3:5 4:5
nafld     1   1   3   1   3   5   7   9  12  15
male      2   2   4   2   4   6   8  10  13  16
notzero   0   0   0   0   0   0   0  11  14  17
> nfit$smap
           1:2 1:3 2:3 1:4 2:4 3:4 1:5 2:5 3:5 4:5
(Baseline)   1   2   3   4   5   6   7   7   7   7
> nfit$share
$vtype
[1] 0 0 2

$scale
 [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
 [6] 1.0000000 1.0000000 0.7462403 0.8381356 1.8102515
```

A base model would have 10 hazards and 20 coefficients. The above restricts this to 7 hazards and 17 coefficients. Here is what was fit, using 0:4 as the state labels, $beta_{m1}$ as the first male coefficient, X_i= (male, nafld) as the data vector for each.

$$\lambda_{01}(t;x) = \lambda_{01}(t)e^{x_{i1}\beta_{m1}+x_{i2}\beta_{n1}} \tag{4.1}$$
$$\lambda_{02}(t;x) = \lambda_{02}(t)e^{x_{i1}\beta_{m1}+x_{i2}\beta_{n1}} \tag{4.2}$$
$$\lambda_{03}(t;x) = \lambda_{03}(t)e^{x_{i1}\beta_{m1}+x_{i2}\beta_{n1}} \tag{4.3}$$
$$\lambda_{12}(t;x) = \lambda_{12}(t)e^{x_{i1}\beta_{m2}+x_{i2}\beta_{n2}} \tag{4.4}$$
$$\lambda_{13}(t;x) = \lambda_{13}(t)e^{x_{i1}\beta_{m2}+x_{i2}\beta_{n2}} \tag{4.5}$$
$$\lambda_{23}(t;x) = \lambda_{23}(t)e^{x_{i1}\beta_{m3}+x_{i2}\beta_{n3}} \tag{4.6}$$
$$\lambda_{04}(t;x) = \lambda_{04}(t)e^{x_{i1}\beta_{m4}+x_{i2}\beta_{n4}} \tag{4.7}$$
$$\lambda_{14}(t;x) = \lambda_{04}(t)e^{\gamma_2+x_{i1}\beta_{m4}+x_{i2}\beta_{n5}} \tag{4.8}$$
$$\lambda_{24}(t;x) = \lambda_{04}(t)e^{\gamma_3+x_{i1}\beta_{m4}+x_{i2}\beta_{n6}} \tag{4.9}$$
$$\lambda_{34}(t;x) = \lambda_{04}(t)e^{\gamma_4+x_{i1}\beta_{m4}+x_{i2}\beta_{n7}} \tag{4.10}$$

Equation (4.7) has no $\gamma_1 = 0$ term since we have `(2:4):5` rather than `(1:4):5`. Think of the right hand side as $Z\gamma + X\beta$, a covariate/coefficien block for shared proportional hazards (Z= notzero) and another for covariates (X= nafld, male).

The `stacker` routine creates a working X matrix with a block of rows for each of the 7 baseline hazards and a column for each of the 17 coefficients, along with Y and strata vectors.

- Block 1 contains all those rows at risk for a 1:2 transition, which is all those rows with initial state of 1. The 0/1 status variable for this block is 1 if the row ended in a transition to state 2. Covariate columns that don't play a role in this transition are 0.
- Likewise block 2 for 2:3 transitons, etc., per the Baseline row in smap above.
- For the shared 1:5, 2:5, 3:5, 4:5 the working X matrix will have first the rows at risk for the 1:5 transition, then rows for the 2:5, etc., all marked with the same value of `share` but different values for `subshare`. The subshare for these 4 will have values of 7, 8, 9, and 10, i.e., their position in the smap vector.

The coxph routine uses this information to reason out two things. The first is what covariates in this working matrix are time dependent, and which of those are constant within every subshare value. Such covariates can be shifted from the $X\beta$ to the $X\gamma$ portion of our model. The second is a multiplier for every transition, based on these constant values. This information is returned as a list `shared` with two elements: the first is a vector of 0/1/2 values, identifying each row of cmap as a variable that is ordinary, time-dependent, or "moved to Z". The second is a vector of $exp(Z\gamma)$ values, one per hazard. The Z values themselves do not need to be saved.

In creating the cumulative hazard and probability in state estimates, the survfit.coxphms function, for each curve

1. Recenter each column of the X matrix to match the target x of the curve, then call stacker to recreate the working X matrix.
2. Form $\eta = X\beta$, ignoring those coefficients that were moved to Z. This is done by zeroing those elements of the coefficient vector.
3. Sums of events and risk, for each subshare are computed by the coxsurv1 or coxsurv2 routine, within the multihaz function.
4. Compute the shared hazards as sum(event)/sum(risk * exp(Zgamma)), where the sum is over the subhazards that are bundled. Then the individual hazards as exp(Zgamma) * shared.

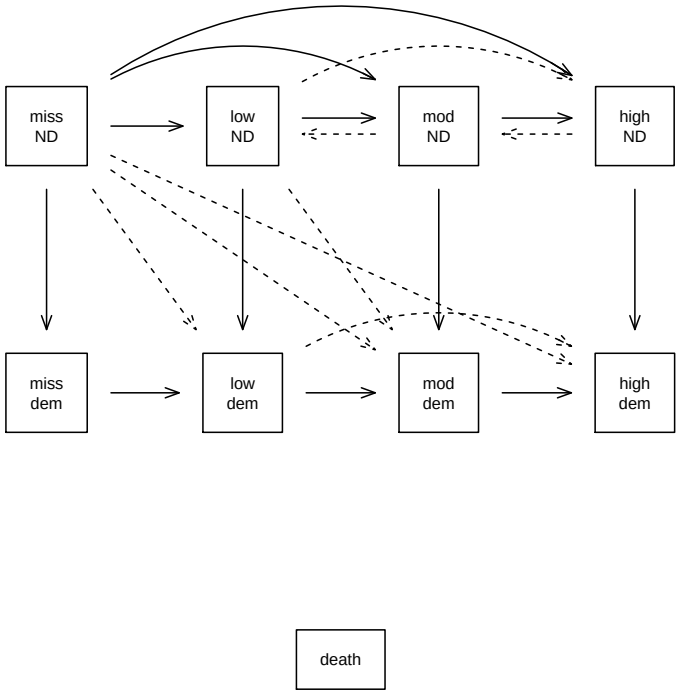
In the fit, there is no distinction between the γ coefficients of our equations, which are associated with proportional baseline hazards, and the β coefficients associated with per-subject covariates. This distinction is recovered at the end of the fit, by identifying components of XB that are constant within each sub-hazard of a shared hazard. Such terms are effectively independent of subject, and we assign them to the $X\gamma$ portion. This is important later, when computing absolute risks. If any are identified, a phZ matrix is included that duplicates relevant rows of cmap, and contains relevant value from the Z matrix. By definition, a time-fixed covariate is not eligible, and (IMHO) it makes no sense to have a TD covariate which is sometime bundled with the hazard and sometimes not, as I have no blueprint to create a proper curve in such a case.

Two cases have led to a lot of thinking. The first is one that I don't yet see any use for. Imagine a semi-competing risks problem, with arrows for A:B, A:death, and B:death; and further that the user asked for a single shared hazard using `+ shared`. The above code will lead to three substrata: A:B and A:death contain exactly the same data rows (all rows with initial state A), but one set has an extra variable ph(A:B); the third subgroup has all rows with initial state of B. The code will work, but the likelihood is a puzzle to me.

- In one sense, each subject in state A actually is a risk twice, once for an A:B and once for an A:death transition.
- This setup does generate martingal residuals that sum to 0, and curves that are properly proportional
- A robust variance should counter the dual appearance of the same subject twice in a risk set
- Yet, a risk set containing clones feels wrong.

The example that has caused the most thinking is from the dementia data set, which is included in the mshmm package. We need to treat amyloid level as a state, because it is time dependent. There are 4 levels of amyloid, along with outcomes of non-demented (ND), dementia, and death. This led to the state space below:

```
> astate <- c(outer( c("miss", "low", "mod", "high"),c("ND", "dem"),
                    paste, sep="\n"), "death")
> amat <- matrix(0, 9, 9, dimnames=list(astate, astate))
> for (i in 1:3) { amat[i, i+1] <- 1; amat[i+4, i+5] <- 1} # right arrows
> for (i in 1:4) {amat[i, i+4] <- 1} #down arrows
> amat[1:8,9] <- 1
> amat[1,3] <- amat[1,4] <- .6
> amat2 <- amat
> amat2[2,4] <- amat2[6,8] <- .6
> amat2[3,2] <- amat2[4,3] <- 1
> amat2[1,6] <- amat2[1,7] <- amat2[1,8] <- amat2[2,7] <- 1
> temp <- sum(amat2[,-9] >0)
> statefig(rbind(4,4,1), amat2, acol=rep(1:0, c(temp,8)),
           alty= (1 + 1*(amat2 != amat))[amat2 >0], offset=.01)
```



We have left out arrows for the 8 transitions to death to make the figure less busy. The solid lines are transitions we expect, i.e., amyloid accumulates, without jumps, and does not change at the exact same instant as cognition. The dashed lines are observed transitions that don't fit this model. But, the transition-based likelihood needs to account for *all* observed transitions. The two solid arcs from missing amyloid to moderate and high are consequences of the fact that the amyloid ligand for PET (Pittsburgh compound B) was not available until several years after the study started, so many imaged subjects start out with a series of non-imaged visits. Due to application of LVCF, no one reverts to the missing state.

If we declare that all transtions from ND to dementia have a shared baseline hazard based on amyloid group: `D(1):D(2)  agrp +1 / common`, this implies that all possible transitions from ND to dementia have the same hazard. That is, all 16 of them. The code really has no choice but to believe the user's declaration. The result in the stacker routine will be 16 subshare sets for that transition, with every at risk observation (current state of ND) appearing in 4 of them. The partial likelihood for each observed ND:dementia transition will have all of these, so 4 copies of each. We can factor this out, giving a constant term of -log(4) time the number of events in the LPL, thus the PL will maximize with exactly the same coefficients as before. Still, the "wrong" PL bothers me, and a lot of the excursions documented in stacker.R (comment block with Dec2024) are based on this, and an attempt to correct it.

We need all 4 copies to get a proper cumulative hazard, however, since the solution needs to obey sum(expected) = sum(observed), and the left hand side of this equation will have all 16 hazards. The real problem is that this is the correct solution to the wrong estimate. Say for instance that the ND:dementia rate is 1/20 person-years but the amyloid low:mod rate was 1/200. The above forces both of them to be about 11/200. What we actually want is an additive model. This is not supported by the coxph/survfit code; one needs to use the intermittent likelihood of mshmm.

The assign attribute

Consider a 3 state model with the following call

```
> fit <- coxph(list(Surv(time1, time2, state) ~ x1 + x2,
                    1:2 + 2:3~ x3/common,  1:3 ~ x4- x2), ...)
```

where x2 and x3 are factors with 3 and 4 levels, respectively. The assign attribute returned by model.matrix will be (1, 2,2, 3,3,3, 4), which correspond to the rows of cmap. The fit, however, will have 13 coefficients. The proper assign vector now is (1,2,2, 3,3,3) for coefficients 1 to 6, then the 2:3 transtion with (4, 5,5, 3,3,3), and (6,7) for the last two coefficients. The expandassign function works this out.

Chapter 5

Testing proportional hazards

The simplest test of proportional hazards is to use a time dependent coefficient $\beta(t) = a + bt$. Then $\beta(t)x = ax + b * (tx)$, and the extended coefficients a and b can be obtained from a Cox model with an extra 'fake' covariate tx. More generally, replace t with some function $g(t)$, which gives rise to an entire family of tests. An efficient assessment of this extended model can be done using a score test.

- Augment the original variables $x_1, \ldots x_k$ with k new ones $g(t)x_1, \ldots, g(t)x_k$
- Compute the first and second derivatives U and H of the Cox model at the starting estimate of $(\hat{\beta}, 0)$; prior covariates at their prior values, and the new covariates at 0. No iteration is done. This can be done efficiently with a modified version of the primary C routines for coxph.
- By design, the first k elements of U will be zero. Thus the first iteration of the new coefficients, and the score tests for them, are particularly easy.

The information or Hessian matrix for a Cox model is

$$\sum_{j \in deaths} V(t_j) = \sum_j V_j$$

where V_j is the variance matrix of the weighted covariate values, over all subjects at risk at time t_j. Then the expanded information matrix for the score test is

$$H = \begin{pmatrix} H_1 & H_2 \\ H_2' & H_3 \end{pmatrix}$$
$$H_1 = \sum V(t_j)$$
$$H_2 = \sum V(t_j) g(t_j)$$
$$H_3 = \sum V(t_j) g^2(t_j)$$

The inverse of the matrix will be more numerically stable if $g(t)$ is centered at zero, and this does not change the test statistic. In the usual case $V(t)$ is close to constant in time — the variance

of X does not change rapidly — and then H_2 is approximately zero. The original cox.zph used an approximation, which is to assume that $V(t)$ is exactly constant. In that case $H_2 = 0$ and $H_3 = \sum V(t_j) \sum g^2(t_j)$ and the test is particularly easy to compute. This assumption of constant components can fail badly for models with a covariate by strata interaction, and for some models with covariate dependent censoring. Multi-state models finally forced a change.

The newer version of the routine has two separate tracks: for the formal test and another for the residuals.

Chapter 6

Residuals

6.1 Survival

The residuals.survfit function will return the IJ values at selected times, and is the more useful interface for a user. One reason is that since the user specifies the times, the result can be returned as an array with (obervation, state, time) as the dimensions, even for a fit that has multiple curves. Full influence from the `survfit` routine, on the other hand, are returned as a list with one per curve. The reason is that each curve might have a different set of event times.

Because only a few times are reported, an important question is whether a more efficient algorithm can be created. For the cumulative hazard, each transition is a separte computation, so we can adapt the prior fast computation. We now have separate hazards for each observed transition jk, with numerator in the n.transitions matrix and denominator in the n.risk matrix. An observation in state j is at risk for all $j:k$ transitions for the entire (time1, time2) interval; the code can use a single set of yindex, tindex, sindex and dindex intervals. The largest change is to iterate over the transtions, and the 0/1 for each j:k transition replaces simple status.

The code for this section is a bit opaque, here is some further explanation. Any given observation is in a single state over the (time1, time2) interval for that transition and accumulates the $-\lambda_{jk}(t)/n_j(t)$ portion of the IJ for all $j:k$ transitions that it overlaps, but only those that occur before the chosen reporting time τ. Create a matrix hsum, one column for each observed jk transition, each column contains the cumulative sum of $-\lambda_{jk}(t)/n_j(t)$ for that transition, and one row for each event time in the survfit object.

To index the hsum matrix create a matrix ymin which has a row per observation and a column per reporting time (so is the same size as the desired output), whose i,j element contains the index of the largest event time which is $\leq \min(\text{time2[i]}, \tau_j)$, that is, the last row of hsum that would apply to this (observation time, reporting time) pair. Likewise let smin be a matrix which points to the largest event time which does *not* apply. The ymin and smin matrices apply to all the transitions; they are created using the outer and pmin functions. The desired $d\lambda$ portion of the residual for the jk transition will be `hsum[smin, jk] - hsum[ymin, jk]`, where jk is a shorthand for the column of hmat that encodes the $j:k$ transition. A similar sort of trickery is used for the dN_{jk} part of the residual. Matrix subscripts could be used to make it slightly faster, at the price of further impenetrability.

Efficient computation of residuals for the probability in state are more difficult. The key issue is that at every event time there is a matrix multiplication $U(t-)H(t)$ which visits all n by p elements of U. For illustration assume 3 event times at 1,2,3, and the user only wants to report the leverate at time 3. Let B be the additions at each time point. Then

$$\begin{aligned}
U(1) &= U(0)H(1) + B(1) \\
U(2) &= U(1)H(2) + B(2) \\
U(3) &= U(2)H(3) + B(3) \\
U(4) &= U(3)H(4) + B(4) \\
&= ([U(0)H(1) + B(1)]H(2) + B(2))\, H(3) + B(3) \\
&= U(0)[H(1)H(2)H(3)] + B(1)[H(2)H(3)] + B(2)H(3) + B(3)
\end{aligned}$$

The first four rows are the standard iteration. The U matrix will quickly become dense, since any $j : k$ transition adds to any at-risk row in state j. Reporting back only at only a few time point does not change the computational burden of the matrix multiplications. We cannot reuse the logarithm trick from the KM residuals, as $\log(AB) \neq \log(A) + \log(B)$ when A and B are matrices.

The expansion in the last row above shows another way to arrange the computation. For any time t with only a $j : k$ event, $B(t)$ is non-zero only in columns j and k, and only for rows with $Y_j(t) = 1$, those at risk and in state j at time t. For example, assume that d_{50} were the largest death time less than or equal to the first reporting time. Accumulate the product in reverse order as $B(50) + B(49)H(50) + B(48)[H(49)H(50)], \ldots$, updating the product of H terms at each step. Most updates to H involve a sparse matrix (only one event) so are $O(s^2)$ where s is the number of states. Each B multiplication is also sparse. We can then step ahead to the next reporting time using the same algorithm, along with a final matrix update to carry forward $U(50)$. At the end there will have been d=number of event times matrix multipications of a sparse update $H(t)$ times the dense product of later H matrices, and each row i of U will have been 'touched' once for every death time k such that $Y_i(d_k) = 1$ (i.e., each B term that it is part of), and once more for each prior reporting time.

I have not explored whether this idea will work in practice, nor do I see how to extend it to the AUC computation.

As a first example, consider the mgus2 data set, set, a case with 2 competing risks of death and plasma cell malignancy (PCM), and use a reporting times of 10, 20, and 30 years. There are $n =$‘r nrow(m2)‘ observations, ‘r etot‘ events and ‘r sum(etime)‘ unique event times before 30 years, with an average of $r =$ ‘r round(nisk,1)‘ subjects at risk at each of these event times. As part of data blinding follow up times in the MGUS data set were rounded to months, and as a consequence there are very few singleton event times.

Both the H matrix and products of H remain sparse for any row corresponding to an absorbing state (a single 1 on the diagonal), so for a competing risk problem the UH multiplication is $O(3n)$ multiplications, those for the nested algorithm will be $O(3r)$ where r is the average number at risk. In this case the improvement for the nested algorithm is modest, and at an additional price of $9d$ multiplications to accumulate H.

```
> ndata <- tmerge(nafld1[,1:8], nafld1, id=id, death= event(futime, status))
> ndata <- tmerge(ndata, subset(nafld3, event=="nafld"), id,
                  nafld= tdc(days))
```

```
> ndata <- tmerge(ndata, subset(nafld3, event=="diabetes"), id = id,
                  diabetes = tdc(days), e1= cumevent(days))
> ndata <- tmerge(ndata, subset(nafld3, event=="htn"),  id = id,
                  htn = tdc(days), e2 = cumevent(days))
> ndata <- tmerge(ndata, subset(nafld3, event=="dyslipidemia"), id=id,
                  lipid = tdc(days), e3= cumevent(days))
> ndata <- tmerge(ndata, subset(nafld3, event %in% c("diabetes", "htn",
                                                     "dyslipidemia")),
                  id=id, comorbid= cumevent(days))
> ndata$cstate <- with(ndata, factor(diabetes + htn + lipid, 0:3,
                                     c("0mc", "1mc", "2mc", "3mc")))
> temp <- with(ndata, ifelse(death, 4, comorbid))
> ndata$event <- factor(temp, 0:4,
          c("censored", "1mc", "2mc", "3mc", "death"))
> ndata$age1 <- ndata$age + ndata$tstart/365.25   # analysis on age scale
> ndata$age2 <- ndata$age + ndata$tstop/365.25
> ndata2 <- subset(ndata, age2 > 50 & age1 < 90)
> nfit <- survfit(Surv(age1, age2, event) ~1, ndata, id=id, start.time=50,
                  p0=c(1,0,0,0,0), istate=cstate, se.fit = FALSE)
> netime <- (nfit$time <=90 & rowSums(nfit$n.event) > 0)
> # the number at risk at any time is all those in the intial state of a transition
> #  at that time
> from <- as.numeric(sub(":[0-9]*", "", colnames(nfit$n.transition)))
> fmat <- model.matrix(~ factor(from, 1:5) -1)
> temp <- (nfit$n.transition %*% fmat) >0 # TRUE for any transition 'from' state
> frisk <- (nfit$n.risk * ifelse(temp,1, 0))
> nrisk <- rowSums(frisk[netime,])
> maxrisk <- apply(frisk[netime,],2,max)
```

As a second case consider the NAFLD data used as an example in the survival vignette, a 5 state model with death and 0, 1, 2, or 3 metabolic comorbidities as the states. For a survival curve on age scale, using age 50 as a starting point and computing the influence at age 90, the data set has 16584 rows that overlap the age 50-90 interval, while the average risk set size is $r = 703$, $< 12\%$ of the data rows. There are 4392 unique event times between 50 and 90 of which 4194 have only a single event type and the remainder 2.

In this case the first algorithm can take advantage of sparse H matrices and the second of sparse B matrices. The big gain is rows of B that are 0. To take maximal advantage of this we can keep an updated vector of the indices of the rows that are at risk, see the section on skiplists. Note: this has not yet been implemented.

For the AUC we can use a rearrangment. Below we write out the leverage on the AUC at time d_4, the fourth event, with $w_i = d_{i+1} - d_i$ the width of the adjactent rectagles that make up the area under the curve.

$$
\begin{aligned}
A(d_4) = U(0)\,[w_0 + H(d_1)w_1 + H(d_1)H(d_2)w_2 + H(d_1)H(d_2)H(d_3)w_3] + \\
B(d_1)\,[w_1 + H(d_2)w_2 + H(d_2)H(d_3)w_3] + \\
B(d_2)\,[w_2 + H(d_3)w_3] + B(d_3)w_3
\end{aligned}
$$

For U the matrix sequence was I, H_3I, H_2H_3I, ..., it is now starts with w_3I, at the next step we multiply the prior matrix by H_3 and add w_2I, at the third multiply the prior matrix by H_2 and add w_1, etc.

6.2 Mulistate hazard models

For multistate models, or for simple models with multiple events per subject, we will use the IJ estimate of variance. As a preliminar write down the martingale process for subject i

$$
\begin{aligned}
M_{mjk}(t) &= \int_0^t dN_{mjk}(s) - Y_{mj}(s)e^{r_m(z)}d\Lambda_{jk}(t;z) \\
&= \int_0^t dN_{mjk}(s) - Y_{mj}(s)e^{r_m(z)}\frac{\sum_i w_i dN_{ijk}(s)}{\sum_i Y_{ij}(s)e^{r_i(z)}}
\end{aligned}
$$

Because of cancellation of $\exp(z)$ in the numerator and denominator the second expression is independent of the target value z. Computationally, we will always use a centering constant c = the `mean` component of the fit object to forestall any round off issues in the exp function.

$$
\begin{aligned}
\frac{\partial \sum_i w_i Y_{ij}(t) r_i(t;z)}{\partial w_m} &= Y_{mj}(t)r_m(t;z) + \sum_i w_i Y_{ij}(t) r_i(t;z)\left[\sum_p (x_{ip} - z_p)\frac{\partial \beta_p}{\partial w_m}\right] && (6.1) \\
&= Y_m(t)r_m(t;z) + \sum_i w_i Y_i(t) r_i(t;z)(x_i - z)D'_m && (6.2) \\
\frac{\partial \lambda_{jk}(t;z)}{\partial w_m} &= \frac{dN_{mjk}(t) - dM_m(t)}{\sum_i w_i r_i(z) Y_{ij}(t)} - \frac{[\sum_i w_i dN_{ijk}(t)]\,[\sum_i w_i Y_i(t) r_i(t;z)(x_i - z)D'_m]}{[\sum_i w_i r_i(z) Y_{ij}(t)]^2} \\
&= a - b && (6.3) \\
a &= \exp(z-c)\frac{dN_{mjk}(t) - dM_m(t)}{\sum_i w_i r_i(c) Y_{ij}(t)} \\
b &= \exp(z-c)\lambda_{jk}(t;c)(\overline{x}(t) - z)D'_m && (6.4)
\end{aligned}
$$

D_m is row m of the dfbeta matrix for the Cox model's coefficent vector β; D is an n by p matrix where n is the number of observations and p the number of coefficients; treat $x_i - z$ as a row vector, the ith row of the X matrix, recentered. The first term of equation (6.3) is essentially the leverage of the ordinary cumulative hazard, with $w_i r_i$ replacing w_i. The second term is the the impact of w_m as mediated through its effect on the model coefficients. Much like confidence intervals for the fitted line in a linear regression, the impact is greater when z is far from the mean of the covariates.

Any row of the data will be at risk in exactly one state for the (time1, time2) interval of that data row, so $Y_{ij}(t)$ is non-zero for only one j. The second term is the the impact of w_m as mediated through its effect on the model coefficients. In the AJ estimate this meant that any row of the data only has an effect on hazards that originate from that state. For the MSH model, a coefficient that is common to two transitions means that all observations at risk for either transition will also have an IJ term for both.

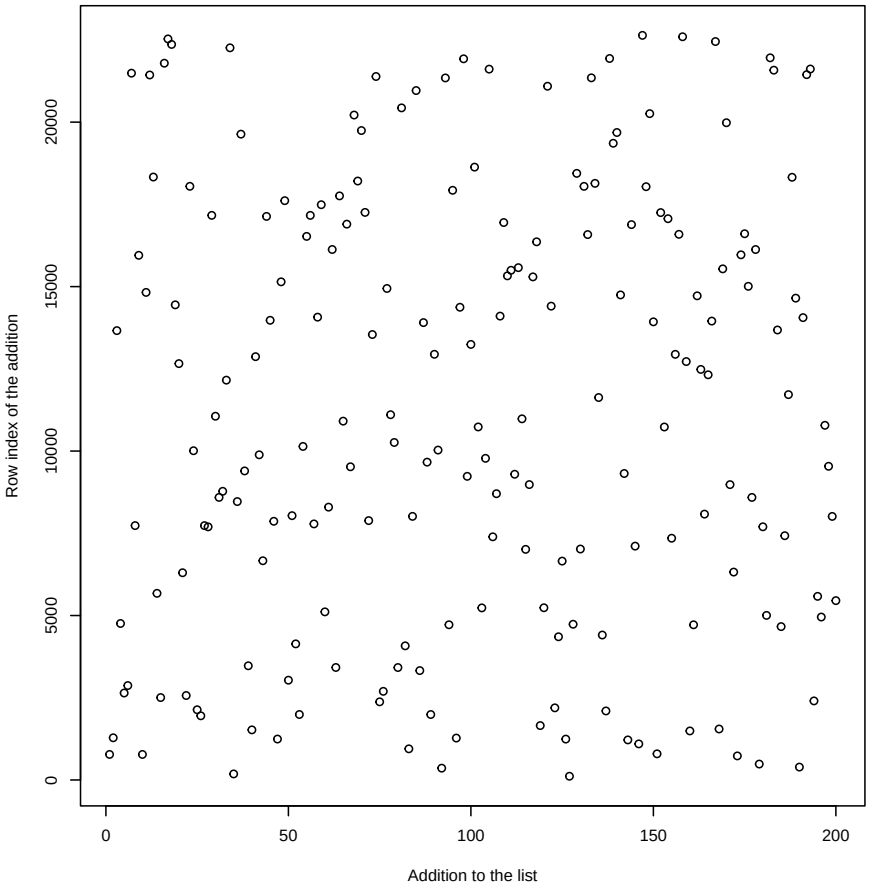
When the same covariate affects two transitions but with disjoint coefficients, the `stacker` routine will have created an expanded X matrix which has separate columns for the two coefficients. Likewise, the D matrix will have a separate column for each coefficient and will be semi-sparse, in that it has zeros for data rows that do not contribute to a give coefficient. Over all rows, the right hand portion of equation (6.4) is `rowSums((X- rep(z, each=nrow(X)))* D)`.

6.3 Skiplists

An efficient structure for a continually updated index to the rows currently at risk appears to be a skiplist, see table II of Pugh [1]. Each observation is added (at time2) and deleted (at time 1) just once from the risk set so we have n additions and n deletions, which are faster for skip lists than binary trees. Reading out the entire list, which is done at each event time, has the same cost for each structure. Searching is faster for trees, but we don't need that operation. If there are k states we keep a separate skiplist containing those at risk for each state.

When used in the residuals function, we can modify the usual skiplist algorithm to take advantage of two things: we know the maximum size of the list from the n.risk component, and that the additions are nearly in random order. The last follows from the fact that most data sets are in subject id order, so that ordering by observation's ending time skips wildly across data rows. For the nafld data example above, the next figure shows the row number of the first 200 additions to the risk set, when going from largest time to smallest. For the example given above, the maximum number at risk in each of the 5 states is (1421, 1280, 845, 514, 0).

```
> sort2 <- order(ndata$age2)
> plot(1:200, rev(sort2)[1:200], xlab="Addition to the list",
       ylab="Row index of the addition")
```



Using $p = 4$ as recommended by Pugh [1] leads to $\log_4(1420) = 5$ linked lists; 1420 is the maximum number of the n =22683 rows at risk in any one state at any one time. The first is a forward linked list containing all those currently at risk, the second has 1/4 the elements, the third 1/16, 1/64, 1/256. More precisely, 3/4 of the nodes have a single forward pointer, 3/16 have 2 forward pointers corresponding to levels 1 and 2, 3/64 have 3 forward pointer, etc. All told there are 4/3 as many pointer as a singly linked list, a modest memory cost over a linked list. To add a new element to the list first find the point of insertion in list 4, list 3, list 2, and list 1 sequentially; each of which requires on average 2 forward steps. Randomly choose whether the new entry should participate in 1, 2, 3 or 4 of the levels, and insert it.

Figure 6.1 shows a simplified list with 40 elements and 3 levels, along with the search path for adding a new entry with value 100. The search starts at the head (shown here at x= 0), takes one step at level 3 to find the largest element < 100 at level 3, then 3 steps at level 2, then one more at level 1 to find the insertion point. This is compared to 20 steps using only level 1. Since the height of a inserted node is chosen randomly there is always the possiblity of a long final search on level 1, but even a string of 10 is unlikely: $(3/4)^{10} < .06$.

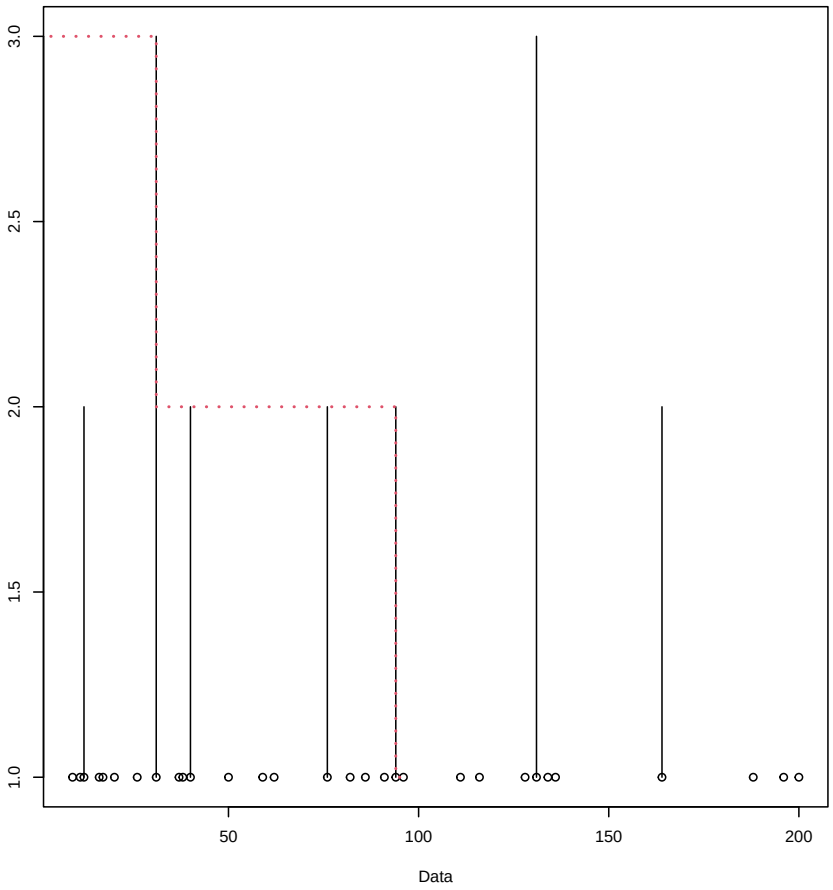


Figure 6.1: A simplified skiplist with 40 elements and 3 levels, showing the path used to insert a new element with value 100.

Chapter 7

Misc numerics

7.1 Matrix exponentials and transition matrices

For multi-state models, we need to compute the exponential of the transition matrix, sometimes many times. The matrix exponential is formally defined as

$$\exp(A) = I + \sum_{j=1}^{\infty} A^i/i!$$

The computation is nicely solved by the expm package *if* we didn't need derivatives and/or high speed. We want both.

We take advantage of three special cases to speed things up. Each depends on the structure of A, which has the transition rates from state j to state k in off diagonal element A_{jk} (non negative), and diagonal elements such that each row sums to zero. $P = \exp(A)$ is the transition probabilty matrix for this time point, each row of P sums to 1 and all elements are non-negative.

1. If all the non-zero rates fall into a single row, or they all fall into a single column, then there are simple closed form formulas shown below.

2. If the rate matrix R is upper triangular and the (non-zero) diagonal elements are distinct, there is a fast matrix decomposition algorithm. If the transition matrix is acylic then it can be rearranged to be in upper triangular form. The decomposition also gives a simple expression for the derivative.

3. In the general case we use a Pade-Laplace algorithm: the same found in the matexp package, but also supply derivatives if requested.

When derivatives are needed, the routine is also passed an array `dR` whose (j, k, i) element is the derivative of A_{jk} with respect to parameter θ_i. In the case of infinitesimal jackknife compuations, which is what is most used in the survival package, θ will be the per-subject leverage on each of the underlying hazards at that time. (Or a group of subjects, for a grouped jackknife). Since the rows of P must sum to one, note that the rows of any derivative of P will sum to zero, i.e., $dR_{jji} = -\sum_{k!=j} dR_{jki}$. Since zero rates only occur for transitions that are

not possible ($\exp(X\beta)\lambda(t)$ is 0 only if $\lambda(t)$ is 0) the derivative will be non-zero only if the rate is non-zero.

Let $P(t) = \exp(A(t))$. If there is only one non-zero diagonal element, A_{jj} say, then

$$\begin{aligned} P_{jj} &= e^{-A_{jj}} \\ P_{jk} &= \left(1 - e^{-A_{jj}}\right) \frac{A_{jk}}{/} \sum_{l \neq j} A_{jl} \\ P_{kk} &= 1; k \neq j \end{aligned}$$

and all other elements of P are zero. If there is only a single event at this time, which turns out to be the most common case with continuous time data, then P and its derivative have only two non-zero elements, corresponding to k= current state of the observation with an event. It is not true that the derivative is zero for other subjects than the i who had an event, as everyone has an impact on $\hat{\beta}$ and thus on the estimated rate at that time.

$$\begin{aligned} \frac{\partial P_{jj}}{\partial \theta_{jki}} &= -P_{jj} \partial A_{jk} \partial \theta_{jki} \\ \frac{\partial P_{jk}}{\partial \theta_{jki}} &= P_{jj} \partial A_{jk} \partial \theta_{jki} \end{aligned}$$

If there are multiple events at this time point but all have the same starting state, the The derivative of P with respect to θ_{jki} will be 0 for all rows except row j.

$$\begin{aligned} \frac{\partial P_{jk}}{\partial \theta_{jki}} &= eta_{jk} A_{jj} \text{ single event type} \\ \frac{\partial A_{jk}}{\partial \eta_{jm}} &= A_{jj} eta_{jm} \frac{A_{jm}}{\sum_{l \neq j} A_{jl}} + (A_{jj} - 1) \frac{\eta_{jm}(1 - \sum_{l \neq j} A_{jl})}{(\sum_{l \neq j} A_{jl})^2} \end{aligned}$$

If all non-zero rates lie in a single column k, then

$$\begin{aligned} P_{jj} = exp(-A_{jk}) \quad j \neq k \\ P_{jk} = 1 - P_{jj} \quad j \neq k \\ P_{kk} = 1 \end{aligned}$$

If time is continuous then most events will be at a unique event time, and the first formula will thus be the most common case. This formula also holds for competing risks. If there is a shared hazard for a set of transitions to a single state k (often death), then for a unique event time (or tied events of the same type) the second formula will hold.

The matrix decomposition is use when the state space is acylic, the case for many survival problems. The states can be reordered so that A is always upper triangular. In that case, the diagonal elements of A are the eigenvalues. If these are unique (ignoring the zeros), then an algorithm of Kalbfleisch and Lawless gives both A and the derivatives of A in terms of a matrix decomposition. For the remaining cases use the Pade' approximation as found in the matexp package.

The overall stategy is the following:

1. Call `survexpmsetup` once, which will decide if the matrix is acyclic, and return a reorder vector if so or a flag if it is not. This determination is based on the possible transitions, e.g., on the transitions matrix from survcheck.

2. Call `survexpm` for each individual transition matrix. In that routine

   - First check for the simple cases, otherwise
   - Do not need derivatives: call survexpm
   - Do need derivatives
     - If upper triangular and no tied values, use the deriv routine
     - Otherwise use the Pade routine

Index

Aalen-Johansen
    residuals, 38
Andersen-Gill
    influence, 18

cox:survival, 28
coxph:survival, 23

Kaplan-Meir, 4

multi-state
    influence, 18
multistate missing, 23

skiplist, 42
surfit
    number at risk, 15
survfit, 3
    AUC, 10
survfitkm
    robust variance, 6
survival:Kalbfleisch-Prentice estimate, 24
survival:time-dependent coefficients, 26

tied times, 3

Bibliography

[1] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications ACM*, 33:668–676, 1990.

[2] S. M. Snapinn, Q. Jiang, and B. Iglewicz. Illustrating the impact of a time-varying covariate with an extended Kaplan-Meier estimator. *Amstat News*, 59:301–7, 2005.