SAS comparisons

Terry Therneau

August 26, 2021

> "I found a bug in your software; it gives a different answer than SAS."
> Message from a user.

One of the perils of writing and maintaining a basic package like `survival` is the inevitable comparisons to SAS. This note talks about a few of these. It's primary message is that for most issues here is a good reason for the differences: they are not an oversight.

None of the material in this vignette is terribly important, and in fact for many of cases the change will be numerically small — of the $n-1$ vs n variety — and can be ignored for practical purposes. I'm hoping that it diverts at least some of the comments like the above. (By the way, in that particular email the real problem was the the *data sets* used for the SAS and R calls were slightly different, a simple user error had occured in setting up the comparison.)

1 Convergence

It should go without saying that `coxph` and SAS `phreg` will not give *exactly* the same answers. They are both working with finite precision floating point arithmetic, a world in which $(1.1 + 1.3) - 1.4 \neq 1.1 + (1.3 - 1.4)$. See the R FAQ, item 7.31 for a longer discussion and explanation of this.

In any case, it is a certainty that even if the two routines are evaluating the same formulas, and both are using Newton-Raphson iteration to arrive at a solution, there will be points at which certain arithmetic operations were done in a different order. Add onto this slightly different methods for matrix inversion, prescaling of the data or not, and different thresholds for our convergence criteria, then yes, the answers will certainly differ in the final digits. Neither is wrong in this case.

2 Efron approximation

The mathematics for a Cox model are all worked out for continuous time values, but in real data there will be ties, i.e., two or more events on the same day. There are several approximations that are used to adapt the method for tied times: the Breslow and Efron approximations, Prentice's marginal likelihood, or the exact partial likelihood option found in Cox's original paper. SAS defaults to the Breslow approximation and R to the Efron approximation.

There was a lot of interest and activity in this area in the early years after the introduction of the Cox model, but the consensus has now settled down to a few simple points.

- The Breslow approximation is both easy to program and computationally efficient. The Efron approximation is nearly as fast, and somewhat more accurate.
- The exact partial likelihood is computationally intensive: if there are d events at one time point out of n subjects at risk, the likelihood is a sum of all $\binom{n}{d}$ possible subsets. When d is between 2 and 10 this is manageable, particularly if an efficient enumeration due to Gail [1] is used, but beyond that point the computation can quickly become untenable. Prentice's marginal likelihood requires a numerical integration; it's computation time is intermediate between the Efron and exact methods.
- At any time point where there are no ties, all four approximations produce identical increments to the partial likelihood.
- If the number of ties is small to moderate, as it is in most data sets, then the *numerical* difference between the methods will be negligible.

It has now come to be appreciated that point 4 may be the most important one, and most people have now simply (and sensibly) stopped worrying about which approximation to use. Consider for instance the example below using recurrence time for the colon cancer data set, and then a second version of the data where time is rounded to the nearest month. In the original data most of the event times are unique; the largest count of ties is 5 events on the same day, which happened twice. The coefficients under the three approximations hardly differ for this data set: standard errors of each coefficient are about 0.1 (not shown) while the differences in estimates do not appear until the third decimal point. That is, there is less than .01 standard error of difference, which is essentially identical from a statistical point of view.

```
> rdata <- subset(colon, etype==1)
> table(table(colon$time[colon$status==1]))
  1   2   3   4   5
533 133  29   6   2
> lfit1 <- coxph(Surv(time, status) ~ rx + adhere + node4, rdata,
                 ties='breslow')
> lfit2 <- coxph(Surv(time, status) ~ rx + adhere + node4, rdata,
                 ties='efron')
> lfit3 <- coxph(Surv(time, status) ~ rx + adhere + node4, rdata,
                 ties='exact')
> rbind(breslow= coef(lfit1), efron=coef(lfit2), exact=coef(lfit3))
              rxLev  rxLev+5FU    adhere     node4
breslow -0.02191469 -0.5119040 0.2819693 0.8851118
efron   -0.02189126 -0.5121441 0.2819438 0.8855844
exact   -0.02193393 -0.5122693 0.2822260 0.8858689
```

Using the coarsened time scale there are only 67 unique event times for the 468 events which occur, leading to some time points with over 20 events. But even in then coefficients only disagree in the second digit.

```
> months <- floor(rdata$time/30.5)
> table(table(months[rdata$status==1]))
```

```
 1  2  3  4  5  6  7  8  9 11 13 14 15 16 19 20 22 23 26
16  9  8  4  3  3  1  3  2  4  1  3  1  1  1  2  1  1  3
> mfit1 <- coxph(Surv(months, status) ~ rx + adhere + node4, rdata,
                 ties='breslow')
> mfit2 <- coxph(Surv(months, status) ~ rx + adhere + node4, rdata,
                 ties='efron')
> mfit3 <- coxph(Surv(months, status) ~ rx + adhere + node4, rdata,
                 ties='exact')
> rbind(breslow2= coef(mfit1), efron2=coef(mfit2), exact2=coef(mfit3))
              rxLev  rxLev+5FU    adhere     node4
breslow2 -0.02220581 -0.5070205 0.2798748 0.8733094
efron2   -0.02199445 -0.5121841 0.2829676 0.8844781
exact2   -0.02283086 -0.5173065 0.2870346 0.8944991
```

Looking more carefully, the Efron approximation has been affected the least by the coarsening; it is also (less importantly) closer to the exact computation result. The survival package chose the Efron as the default from a basic "why not the best" logic. After all, if one had two approximations to the cosine function with similar compute cost, but one of them behaved better in certain edge cases, any sensible code for fundamental libraries would use the more stable one. However, in a statistical analysis context and given the usual size of standard errors, a Breslow default will be just as good. The Breslow approximation is the default in many statistical packages for a simple historical reason: since it is the easiest to program it was usually the first method to be implemented.

Once having chosen the Efron default it is important to carry through, however. The approximation turns out to have implications for how residuals and baseline hazard functions are computed, which in turn affects robust variance estimates. Details of this can be found in the validation vignette. This is one area where the phreg procedure is not quite correct, so its results do not agree with the validation suite. For all the examples that we have investigated in detail, however, the practical implications of the inaccuracy have been small. The largest practical issue vis-a-vis SAS/R is users who don't read the documentation, and then get worried when numbers don't exactly agree, since one run used Efron and the other Breslow.

3 Efficiency of the Cox model

A question of the relative speeds of coxph and phreg is one that arises fairly often; a particular data set caused us to look into this more formally.

3.1 Counting process data

The solution to the Cox estimating equations is found using Newton-Raphson iteration, which requires the computation of first and second derivatives with respect to to each coefficient. The basic quantities for these are a weighted mean and variance of X, at each event time, taken over the set of subjects who are at risk at that event time, using $\exp(X\hat{\beta})$ as the weights. Let p be the number of covariates, d the number of unique event times and n the number of observations. An ideal Cox model program would have a computational burden of $[O(d)+O(dp)+O(dp^2)]k$ for the

partial likelihood, the coefficients, and the variance matrix, respectively, where k is the number of iterations required; for almost all data sets k is between 3 and 5. This is simply the work of adding up the partial likelihood and its first and second derivatives. In a real program there will an additional $[O(n) + O(np) + O(np^2)]k$ time for creating the increments which should be added to the sums, where n is total number of observations in the data set. For a Cox model the increment to the second derivative, at each death time, is a weighted variance of the covariates X, over all those who were at risk for the event, while the increment to the first derivative involves the vector of weighted means. For the rest of this discussion assume that p and k are fixed, and we will refer the run time for a Cox model program as $O(n) + O(d)$. Since $n > d$, often by a large margin, the first term normally dominates the second. More importantly, real programs and algorithms will not match this ideal, since there are various other tasks to be done.

For ordinary `Surv(time, status)` survival data, SAS phreg, R coxph, and every other code that I am aware achieves the $O(n)$ ideal via a simple strategy: process the data from largest to smallest survival time. That way the sums needed for each of the risk sets occur naturally as we proceed; and one pass through the data will suffice to compute them using well known formulas for updating a mean and variance. Essenially no time is spent on the "who in each risk set" task. This makes the code quite efficient. If there are strata in the model, means and variances of X are taken *within* strata; this adds no extra work since we need only zero the relevant running sums at the start of each stratum. The computation starts by sorting the data by reverse time within strata, but sorting is a basic operation that all packages do efficiently so we haven't counted it in the total.

When the data is in counting process form, i.e., `Surv(time1, time2, status)`, the computation is more challenging. The risk set for a subject who dies at some time t is the set of all observations such that `time1` $< t \leq$ `time2`. A worst case algorithm is to compute the mean and variance de novo at each death time, iterating over all the observations to enumerate the at risk subset. This leads to a computation time of $O(nd)$. We hope to do better than that. A slightly better approach is to keep a running list of which observations are in the risk set, leading to an $O(rd)$ algorithm where r is the average number of subjects at risk. (Updating a list of indices is a well known problem in computer science, and like sorting it can be done very efficiently. See red/black trees for example.)

As a way of investigating the performance we will use a simulation data set. It is intended to mimic to some degree the Nurses Health Study, since analysis of that data set raised a question about the relative speed of coxph and phreg. The code for the data set is below; most readers can skip directly to the printout at the end.

```
> # Simulate a study with long follow-up
> # Base it roughly on the NHS, with follow-up every 2 years
> #
> set.seed(1960)
> n <- 121700    # number of nurses who enrolled
> #n <-  12170      # test run, with smaller data
> temp <- floor(c(30*365.25, 55*365.25))
> age <- sample(temp[1]:temp[2], n, replace=TRUE) # age in days, at enrollment
> iage <- floor(age/365.25)               # birthday age
> # up to 16 bi-annual follow-ups for each subject + enrollment visit
> #  covariates are rounded to fewer digits to make nicer printout
```

```
> #  of data set rows
> times <- matrix(sample(700:770, n*16, replace=TRUE), ncol=n)
> vage <- apply(rbind(age, times), 2, cumsum)
> temp <- data.frame(id=rep(1:n, each=17),
                     vage = c(vage),     # age at visit
                     x1 = round(runif(n* 17, 100, 200)),
                     x2 = round(rgamma(n* 17, 1,2), 2))
> # Every subject has a flottila of binary covariates,
> #  which I will make constant per subject (it's simpler and won't change
> #  the timing behavior) and somewhat correlated.
> # There are nice packages for this, but I'm constrained by the fact that
> #  survival is a recommended package
> cbin <- function(n, p, rho=.5) {
      gauss <- matrix(rnorm(n*p), ncol=p)
      cmat <- diag(p)
      cmat[col(cmat) < row(cmat)*.8] <- rho   # cholesky of my variance matrix
      ifelse(gauss %*% cmat >0, 1, 0)
  }
> xb <- cbin(n, 40)  # binary covariates
> # a random survival time for each subject.  Scaled Minnesota death rates
> #  give approx the same number of deaths as the NHS (31,000).
> truecoef <- c(rep(seq(-5,4), 2)/10, rep(0,20)) # some are important, more not
> eta <- xb %*% truecoef + rnorm(n, sd=.3)  # we never know all the risks
> sy <- survexp.mn[1:110, 'female', "1985"]*365.25 # yearly hazard rate
> sy <- sy * .8 / mean(exp(eta))  # emprical adjustment
> dtime <- double(n)
> for (i in 30:55) {   # some start at each age
      j <- which(iage == i)
      chaz <- rexp(length(j))  # chaz for each subject is exp(1)
      dtime[j] <- approx(cumsum(sy[i:109]), i:109, chaz/exp(eta[j]), rule=2)$y
  }
> truecoef <- c(0,0, truecoef)  # add the coefs for x1 and x2
> # change to days
> dtime <- pmax(30, ceiling(dtime*365.25))  # at least 1 month of survival
> dummy <- data.frame(id=1:n, dtime=dtime, agem = floor(age*12/365.25))
> nhs <- tmerge(dummy, dummy, id=id,  death=event(dtime),
                options= list(tstartname="age1", tstopname="age2"))
> nhs <- tmerge(nhs, temp, id=id, qnum= cumtdc(vage),
                x1=tdc(vage, x1), x2=tdc(vage, x2))
> latedeath <- with(nhs, death==1 & (age2-age1) > 800)
> nhs <- subset(nhs, age1>0 & !latedeath)
> # Adding xb last makes it remain a matrix within the dataframe.
> # This allows me to write
> #   + xb in the model rather than +xb.1 + xb.2 + ... + xb.40
> #  That is, same model, less typing.
> nhs$xb <- xb[nhs$id,]
```

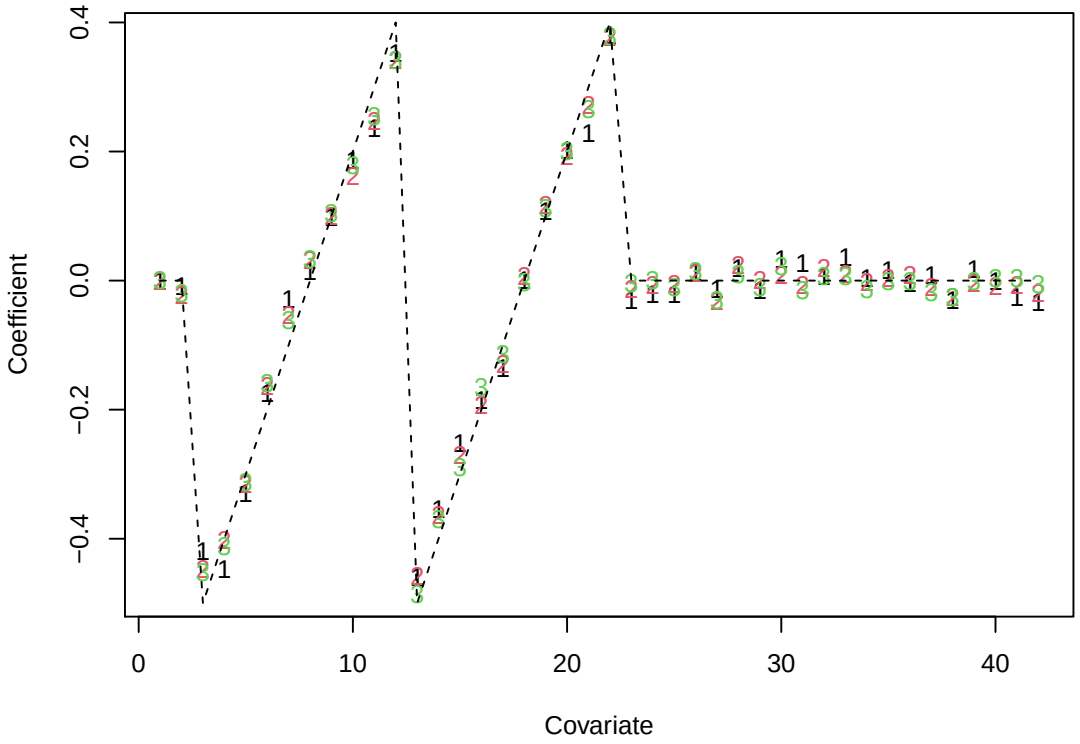
```
> # Look at statistics for the first 1/3, 2/3, and 3/3 of the subjects
> nsubject <- max(nhs$id)
> stats <- matrix(0, 5, 3)
> fit1 <- vector("list", 3)
> for (i in 1:3) {
      temp <- subset(nhs, id <= (nsubject* i/3))
      stats[1,i] <- nrow(temp)

      km <- survfit(Surv(age1, age2, death) ~ 1, temp)
      stats[2,i] <- sum(km$n.event)
      stats[3,i] <- sum(km$n.event>0)
      stats[4,i] <- mean(km$n.risk[km$n.event >0])

      time1 <- system.time(fit1[[i]] <-coxph(Surv(age1, age2, death) ~ x1+ x2 + xb,
                           ties="breslow", temp))
      stats[5,i] <- sum(time1[-3])   # ignore elapsed time
  }
> dimnames(stats) <- list(c("number of observations", "number of events",
                            "unique event times", "mean number at risk",
                            "coxph cpu time"),
                          c("first 1/3", "2/3", "all"))
> round(stats)
                       first 1/3     2/3     all
number of observations    610216 1221132 1831069
number of events           10287   20432   30677
unique event times          7255   10811   12733
mean number at risk        22366   45328   68526
coxph cpu time                50     104     159
```

The resulting data set has 1.8 million records for 122 thousand subjects, reflecting 24 years of biennial follow up in the NHS. Subjects have a new observation at each visit. In the actual data the covariates are time dependent (though most covariates don't change, for most of the visits, for most subjects.) There are a lot of events, so coefficient estimates are quite precise as shown in the next graph, whether one uses all the patients or only 1/3 of them. Evaluation required 4 iterations of the `coxph` function.

```
> y <- cbind(coef(fit1[[1]]), coef(fit1[[2]]), coef(fit1[[3]]))
> matplot(1:42, y, pch="123",
          xlab="Covariate", ylab="Coefficient")
> lines(1:42, truecoef, lty=2) # add the true coefficient to the plot
```



The run time for `coxph` is nearly linear in n, the number of observations. The approach used by coxph is to use running sums for (time1, time2) data, just as it does for simple survival, but in this case subjects will be added *and* subtracted from the totals. The code marches backwards in time; an observation of (100, 200, 1) would be added to the totals when the code's internal time value crosses 200, and then removed when the time crosses 100. This requires some care in the internal routines to avoid catastrophic cancellation; e.g., in finite precision arithmetic (1e18 + 123) - (1e18) =0 so a large outlier that is added and then later removed can be deadly. (Mitigation for this is discussed in the noweb source code.) Said algorithm is expected to have $O(2n)$ run time since it touchs each observation twice. Since $d < n/60$ the $O(d)$ term hardly matters in this data.

Running the same data set using SAS phreg, the execution times were 85, 303, and 867 seconds for 1/9, 2/9, and 3/9 of the subjects, a quadratic growth rate. (We used ties='breslow' in R to match the phreg default.) The phreg algorithm appears to use the simpler approach, which is to calculate the variance anew at each death time. In this case the dominating term in the compute time will be $O(nd)$, and d grows with n; the full data set fit is anticipated to take approximatedly 9*867 seconds or about 2.2 hours. (We ran out of patience and did not actually run this case.)

I was asked if the iteration count would increase in the real data set, due to high correlation of the covariates. Something that is often not appreciated is that one can replace the matrix of covariates X with a transformed version AX, and the iteration path for the Newton-Raphson iteration of a Cox model remains unchanged, giving the exact same series of likelihood values. Practically, this means that the expected iteration count does not depend on whether

the covariates are correlated, unless X has become so close to singularity that round off errors intervene.

3.2 Data expansion

The NHS example started with a complaint that R was 8 fold *slower* than SAS on the actual NHS data set, which puzzled me. In what situation could this happen?

One way is to make the R code slower by artificially increasing n. This can be done by *dicing* the nhs data set, i.e., cut each observation into multiple small pieces (in the same sense as dicing vegetables in the kitchen). The very first observation of the simulation data set, for instance, is a censored time span from age 40 to age 42.06 = 14615 to 15361 days of age. What if we were to replace this observation by a whole set of rows with (age1, age2, death) values of (14610, 14611, 0), (14611, 14612, 0), (14612, 14613, 0), ... (15360, 15361, 0)? Since NHS subjects are contacted every two years, this will increase the data set size by approximately 730 fold, leading to a corresponding 730 fold increase in the compute time for `coxph`.

This kind of expansion is sometimes done to create dummy covariates x*log(t) as a step towards testing proportional hazards. Said PH test does not require expansion for every single day, however, only for the set of unique death times, i.e., we can use the set of death times as the cut points when creating an expanded data set. (Any intervals that don't overlap a death time turn out to play no role in the Cox partial likelihood.) In the survival package an expanded data set is never needed for this particular purpose, since the `cox.zph` function directly computes the relevant score test in $O(n)$ time, without data expansion.

Another use of such data sets is when a user desires to control the risk sets exactly. An example is nested case-control sampling, where at each death time a subsample of the subjects who were at risk at that time is selected. Say there were 52 events and 10 controls were selected for each. A special data set is constructed which has 52*11 observations: the first event + the 10 controls for it, then the second event + its 10 controls, etc. The data set will contain a `group` variable that divides these into 52 groups and a status variable of 1=event/ 0=control. A standard Cox model program can now be used to fit the data. A `time` value, which used by the `coxph` routine to determine risk set membership, is not needed since the risk sets are already set, and one can use the call

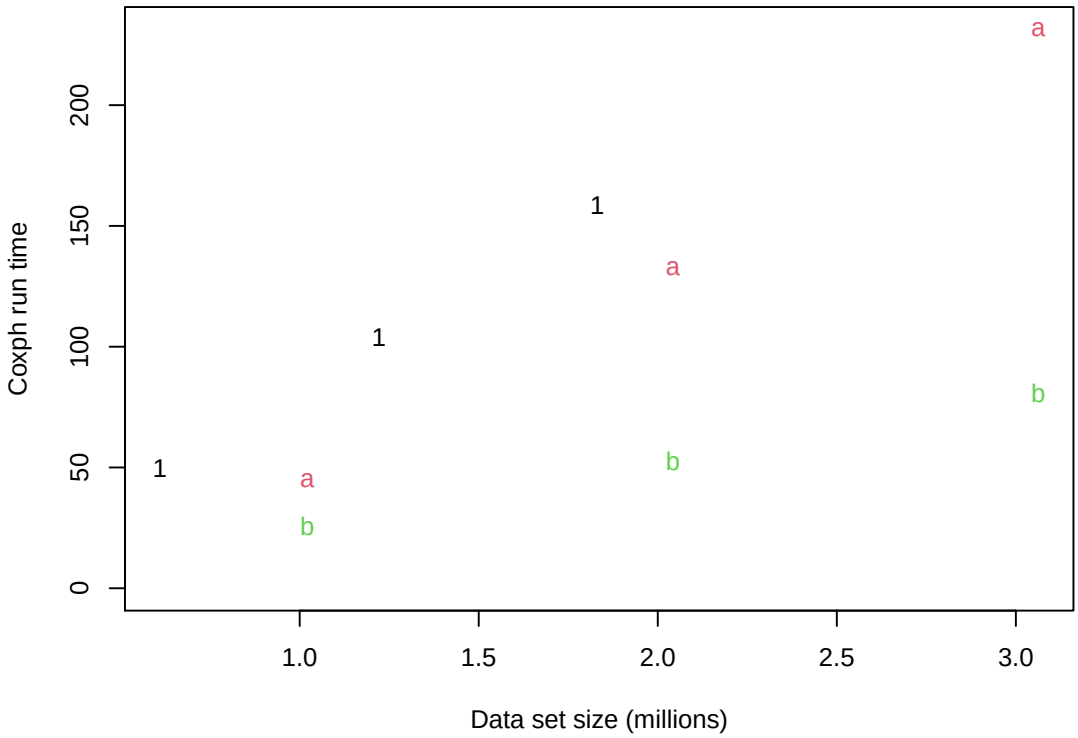
```
fit <- coxph(Surv(dummy, status) ~ strata(group) + x1 + x2 + ...)
```

where `dummy` is a dummy variable set to a constant value (often 0 or 1).

Such an approach has attraction for phreg, since we are no longer using a (time1, time2) form. Perhaps trading a larger data set size for $O(n)$ behavior will be a win? To test this out, and yet keep the running time of this vignette sane, we create an expanded data set using 1/12 of the subjects, breaking each subject into monthly intervals. This will create a data set with approximately 24/12 = 2 times as many rows as the starting `nhs` data set. Within the data set, use age in months as the time scale.

```
> temp <-  subset(nhs, id <= 10140)    # 1/12 of 121700 subjects
> temp$dummy <- 1
> temp$month1 <- floor(temp$age1 * 12/365.25)
> temp$month2 <- floor(temp$age2 * 12/365.25)
> temp <- subset(temp, month1 < month2)   # avoid 0 length intervals
```

```
> dtime <- sort(unique(temp$month2[temp$death == 1]))
> nhs2 <- survSplit(Surv(month1, month2, death) ~., data=temp, cut=dtime)
> nrow(nhs2)  # 3 million rows
[1] 3062743
> stat2 <- matrix(0, 6, 3)
> maxid <- max(nhs2$id)
> fit3a <- fit3b <- vector("list", 3)
> for (i in 1:3) {
      temp <- subset(nhs2, id <= (maxid*i/3))  # 1/3, 2/3, 3/3
      km <- survfit(Surv(age1, age2, death) ~ 1, data=temp)
      time1 <- system.time(fit3a[[i]] <- coxph(Surv(month1, month2, death) ~ x1 +
                                               x2 + xb,
                                 ties = "breslow", data=temp))
      time2 <- system.time(fit3b[[i]] <- coxph(Surv(dummy, death) ~ x1 + x2 + xb +
                                    strata(month2), ties="breslow", data=temp))
      stat2[1, i] <- nrow(temp)
      stat2[2, i] <- sum(temp$death)
      stat2[3, i] <- length(unique(temp$age2[temp$death==1]))
      stat2[4, i] <- mean(km$n.risk[km$n.event > 0])
      stat2[5, i] <- sum(time1[-3])
      stat2[6, i] <- sum(time2[-3])
  }
> all.equal(coef(fit3a[[3]]), coef(fit3b[[3]]))
[1] TRUE
> dimnames(stat2) <- list(c("observations", "death", "unique deaths",
                            "mean risk set",
                            "coxph time 1", "coxph time 2"),
                          c("first 1/3", "2/3", "all"))
> round(stat2)
              first 1/3     2/3     all
observations    1021010 2042415 3062743
death               831    1680    2533
unique deaths       808    1581    2310
mean risk set     41407   84651  125407
coxph time 1         45     132     231
coxph time 2         26      53      81
> ytemp <- cbind(stats[5,], stat2[5,], stat2[6,])
> matplot(cbind(stats[1,], stat2[1,], stat2[1,])/1e6,  ytemp,
          ylim=c(0, max(ytemp)), pch="1ab",
          xlab="Data set size (millions)", ylab="Coxph run time")
```



The figure shows that the run times from `fit1` on the original time scale and `fit3a` and `fit3b` on the coarsened and expanded data. For the (time1, time2) forms of the data the compute time remains linear in the number of rows, coarsening the time scale has had only a small impact. The compute time for a model that takes advantage of the strata and a dummy time value is indeed smaller, about 1/3 of the (time1, time2) form. However, the overall effect is a net loss: 81 seconds for to fit 1/12 of the data in the expanded form versus 159 seconds for the entire data set in its original form.

The row count grows faster than the number of subjects, since the number of cut points per subject also grows with n, though that will eventually slow (there are only so many possible death times). The two fits in the loop, one using (age1, age2) and the other stratified on the age at death, give exactly the same likelihood since they have exactly the same risk sets at each event time.

For SAS however, the two constructions differ substantially in compute time. A portion of the SAS code is shown below. Using (time1, time2) form on the expanded data set the run time was a little over 22 minutes, continuing the quadratic behavior we saw earlier, while the stratified form with a dummy time value required 10, 18, and 25 seconds for 1/3, 2/3 and 3/3 of the subjects, the expected linear growth. Extrapolating to the full data set we would predict 12*25 seconds or about 6 minutes, versus 867*9 seconds or 130 minutes for the original counting process form. Expansion is a clear win.

```
proc phreg data= nhs2;
   model (month1 month2) * death(0) = x1 x2 xb_1 xb_2 xb_3 xb_4 xb_5 xb_6 xb_7
        xb_8 xb_9 xb_10 xb_11 xb_12 xb_13 xb_14 xb_15 xb_16 xb_17 xb_18 xb_19
```

```
        xb_20 xb_21 xb_22 xb_23 xb_24 xb_25 xb_26 xb_27
        xb_28 xb_29 xb_30 xb_31 xb_32 xb_33 xb_34 xb_35 xb_36
        xb_37 xb_38 xb_39 xb_40;
        strata month2;
proc phreg data=nhs2;
   model dummy*death(0) =x1 x2 xb_1 xb_2 xb_3 xb_4 xb_5 xb_6 xb_7
        xb_8 xb_9 xb_10 xb_11 xb_12 xb_13 xb_14 xb_15 xb_16 xb_17 xb_18 xb_19
        xb_20 xb_21 xb_22 xb_23 xb_24 xb_25 xb_26 xb_27
        xb_28 xb_29 xb_30 xb_31 xb_32 xb_33 xb_34 xb_35 xb_36
        xb_37 xb_38 xb_39 xb_40;
   strata age2;
```

The main take home message for R is "don't dice your data set!" That step is essential to achieve good performance in SAS, but what makes things faster for one statistical package does not necessarily work for another. This is especially true for something like the NHS simulation, where per subject covariate updates occur at a low frequency.

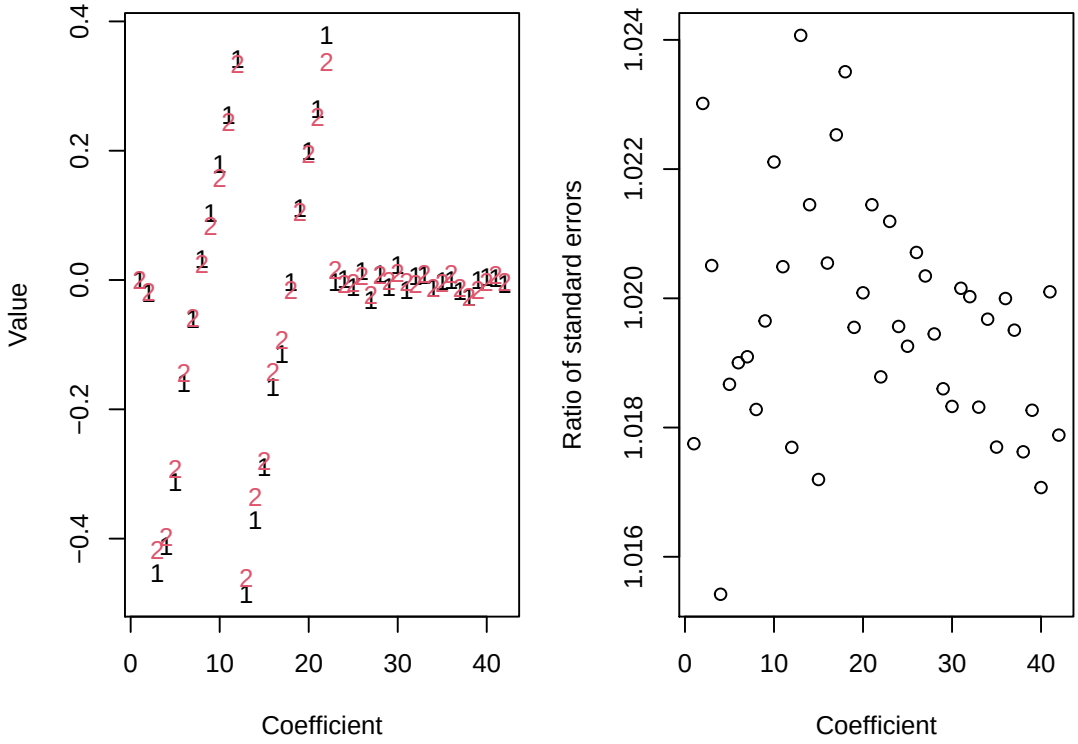
3.3 Stratification

The average number of subjects at risk, at each death, in the full Cox model, is over 65 thousand:

```
> kmfit <- survfit(Surv(age1, age2, death) ~1, nhs)
> mean(kmfit$n.risk[kmfit$n.event > 0])
[1] 68525.59
```

This is statistical overkill: 10-20 conrols per case will work just as well. One simple way to thin this down is to stratify the analysis. As an example we will break the subjects into sets based on their age at enrollment (in months); which is the variable `agem` in our data set above. This leads to 300 strata, corresponding to the 25 year span in ages at enrollment. What does it do to the compute time?

```
> system.time(fit2 <- coxph(Surv(age1, age2, death) ~ x1 + x2 + xb +
                                    strata(agem), ties='breslow', data=nhs))
   user  system elapsed
125.744   1.423 127.169
> par(mfrow=c(1,2))
> matplot(1:42, cbind(coef(fit1[[3]]), coef(fit2)),
                       xlab="Coefficient", ylab= "Value")
> s1 <- sqrt(diag(vcov(fit1[[3]])))  # std of coefficients
> s2 <- sqrt(diag(vcov(fit2)))
> plot(1:42, s2/s1, xlab="Coefficient", ylab="Ratio of standard errors")
```



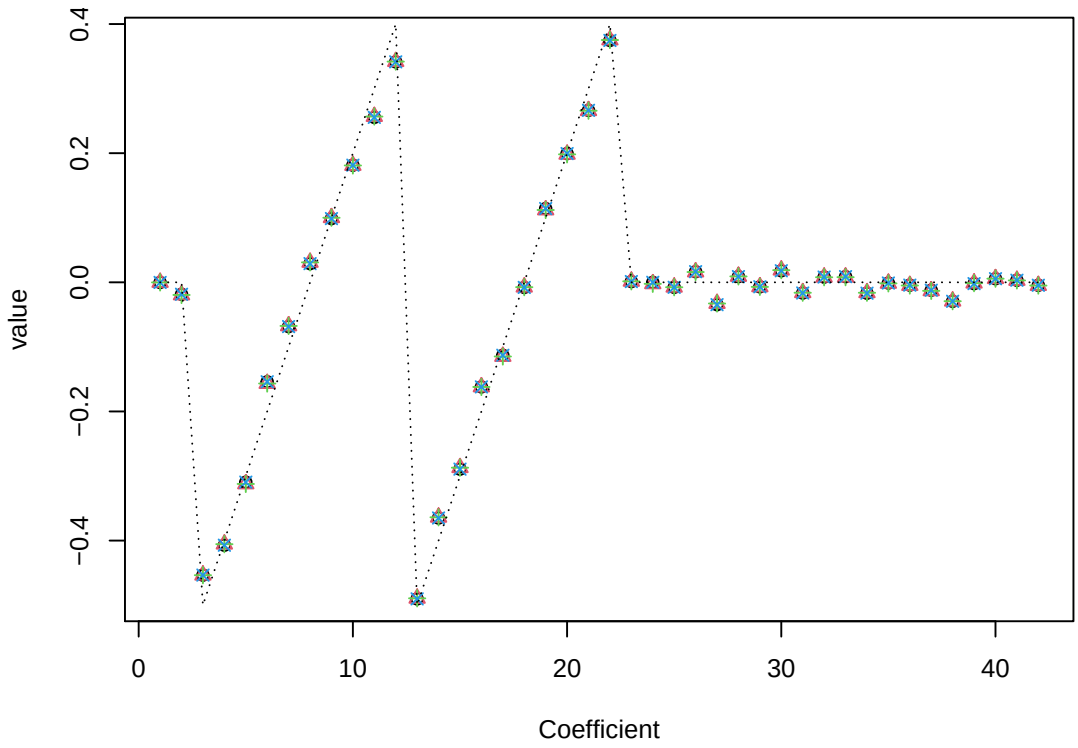
Reducing the average number at risk by 300 fold has reduced the compute time by about 30%, without any appreciable change in the estimates or their standard errors. The overall size of the data set has not changed and size dominates the `coxph` compute time. A small gain comes from observations that occur after the last event in any stratum, as (time1, time2) intervals that do not overlap an event time play no role in the partial likelihood and are ignored by coxph. The effect of stratification on phreg, however, was profound: the run times for 1/3, 2/3, and 3/3 of the data were now 8, 21, and 40 seconds; still quadratic but with a 120 fold decrease.

3.4 Speeding up phreg

What is the best that we can do with phreg? The obvious target would be to avoid the (time1, time2) form at all costs, while at the same time not having to expand the data set; and secondly to employ strata to reduce the size of the risk sets. Further correspondence with the NHS staff revealed that they had done exactly this. The key trick is to stratify observations based on each subject's age at their *prior* visit (in months). Here are four versions in R.

```
> nhs$month1 <- floor(nhs$age1 * 12/365.25)
> nhs$month2 <- floor(nhs$age2 * 12/365.25)
> tdata <- subset(nhs, month1 < month2)  # remove any ties
> fit4a <- coxph(Surv(age1, age2, death) ~ x1 + x2 + xb + strata(month1),
                 data=tdata, ties='breslow')
> fit4b <- coxph(Surv(month1, month2, death) ~ x1 + x2 + xb + strata(month1),
                 data=tdata, ties='breslow')
```

```
> fit4c <- coxph(Surv(month2, death) ~ x1 + x2 + xb + strata(month1),
                 data=tdata, ties='breslow')
> fit4d <- coxph(Surv(age2, death) ~ x1 + x2 + xb + strata(month1),
                 data=tdata, ties='breslow')
> matplot(1:42, cbind(coef(fit4a), coef(fit4b), coef(fit4c), coef(fit4d)),
          pch=1:4,
          xlab="Coefficient", ylab="value")
> lines(1:42, truecoef, lty=3)
> all.equal(coef(fit4a), coef(fit4b))
[1] "Mean relative difference: 0.00780723"
> all.equal(coef(fit4b), coef(fit4c))
[1] TRUE
```



The plot shows the coefficients from the 4 fits, results of the four are almost identical. This is not surprising: over the 50 year age range of the NHS, coarsening the time scale to months is a small relative change to any subject. A closer look reveals that the fit4a and fit4b differ by less than 1%, and the coefficients for fit4b and fit4c are completely identical. The reason for the latter is that everyone in a given stratum has exactly the same `month1` value, so we can dispense with the (time1, time2) form of the call.

In the above example we removed observations where month1=month2 to demonstrate the equivalence of fit4b and fit4c; such observations are automatically removed when fitting (time1, time2) data. However, this is not necessary when using the (time2, death) form of the model.

The coxph compute time for fit3 above, without the removals, was 24 seconds.

The phreg compute times for three models above with (age1 age2), age2, and month2 as the response, respectively, and each stratified by month1, was 289, 28, and 28 seconds. Clearly stratification and avoiding the counting process form are important for phreg, but the final step of coarsening did not create any further gain. Why is the (age1, age2) fit for phreg so much longer than an identical model stratified by month of entry? This was a complete surprise and I do not have any hypothesis.

3.5 Data set sizes

The 'xz' compression option of the `save` command is particularly successful for data sets like the NHS simulation that contain a lot of whole numbers, an rda file containing nhs and nhs2 consumed just under 10 MB of disk space, while the size using default `save` options was 38 MB. The save operation itself takes longer using xz compression, but reloading the save data sets requires similar time. The SAS data set for nhs used 290 MB and that for nhs2 505 MB using default options, more knowledgeable SAS coders perhaps could to do better.

3.6 The FAST option

Our 2019 version of phreg now has a *FAST* option. The phreg manual is completely vague about the actual approach:

> "FAST uses an alternative algorithm to speed up the fitting of the Cox regression for a large data set that has the counting process style of input. Simonsen (2014) has demonstrated the efficiency of this algorithm when the data set contains a large number of observations and many distinct event times. The algorithm requires only one pass through the data to compute the Breslow or Efron partial log-likelihood function and the corresponding gradient and Hessian. PROC PHREG ignores the FAST option if you specify a TIES= option value other than BRESLOW or EFRON, or if you specify programming statements for time-varying covariates. You might not see much improvement in the optimization time if your data set has only a moderate number of observations."

The reference in the manual is "Simonsen, J. (2014). Statens Serum Institut, Copenhagen. Unpublished SAS macro", which is not helpful at all. However, someone else pointed us to a web reference which contains a macro that is the purported prototype for the phreg option; it can be found using a web search for the article title "A method for speeding up PROC PHREG when doing a Cox regression".

A closer look at the macro brought on a strong case of deja vu. Early in the author's computing career an analysis data set would sometimes be too big to fit into memory, and a common work around for a binary outcome was to

1. Reduce any predictors to categorical variables of 2–4 levels.
2. Create a new data set with one row for each unique combination of y and the (new) predictors, giving each a case weight equal to the number of observations in the original data set that fell into that bin.

3. Analyze the new data set, using these case weights.

Reductions in the data set size of 20-100 fold were not uncommon.

The FAST option appears to use essentially the same approach. Say that there are k unique combinations of the categorized covariates. For a Cox model, create an intermediate data set with 2k observations at each unique event time, one with status=0 and one with status=1, and weights equal to the number of censorings and events in that bin*time combination. Observations with a weight of 0, if any, may be omitted. Then fit an ordinary Cox model to this data set, stratified by event time and using Surv(dummy, status) as the response, `dummy` a dummy vector zeros. The compute time will be $O(kd)$, which is potentially much smaller than the $O(nd)$ compuation for the original data set. Creating the collapsed data set is itself a task that potentially requires $O(nd)$ time, since for each event time the tally needs to check all n observations to see if their (time1, time2) interval includes the given event. However, this is essentially an indexing problem for which there are fast algorithms.

As someone who often preaches against categorization of predictors, the author does not find this approach attractive.

3.7 Summary

Statistical theory tells us that the variance of the coefficients from a Cox model will be of order $V \sum_j (r_j - 1)/r_j$, where V is the variance of the predictors X and r_j is the size of the risk set associated with the jth death. In this data set with an average risk set size of > 65 thousand, it should be no surprise that replacing the risk set with a smaller subsample has no measurable effect on the estimates; whether that were to be done with a random subsample, stratification on entry date (1/300) or stratificatin on the 650 "prior visit" dates. A primary lesson for this statistician is that when analysis is slow due to a large data set, perhaps our first reaction should be the *think* instead of reaching for a bigger hammer.

For the naive $O(nd)$ algorithm used by phreg, the effect of subsampling is a staggering decrease in the compute time; stratification essentially replaced n with $n/300$. A clever stratification that allows one to omit time1 gave further improvement.

Stratification can give a statistical benefit of less biased results as well, not just speed, if the stratification variable is correlated with any unmeasured covariates that are associated with outcome. This is a justification for stratifying on enrolling institution, for instance. In the present case it is hard to argue that either month of enrollment or month of the prior response is anything more than a random partition of the original risk set {all subjects of the same age as the death}. In every statistical aspect it should perform the same as random subsampling of the same size. (But in terms of terms of speeding up SAS phreg it is a *very* clever idea.)

I have not been able to replicate any case where phreg is significanly faster than coxph, and in that sense the report this behaviour from the NHS staff remains a mystery.

4 Type 3 tests

Another user query has been for "type 3" tests of survival models. Up until the latest release (at our institution) the phreg procedure would automatically the type 3 test — you couldn't turn it off in fact. They have now been relabeled as "joint tests" but appear to be the exact same values

as before. The primary problem with these is that they depend on the way in which covariates are coded.

```
data test;
input time status x1 $ x2 $;
cards4;
 1 a  A
 2 b  B
10 c  C
50 a  D
 5 b  A
 4 c  B
 8 a  C
40 b  D
60 c  A
20 a  B
21 b  C
22 c  D
 3 a  A
 5 b  B
12 c  C
52 a  D
 7 b  A
 8 c  B
16 a  C
88 a  D
58 a  A
28 a  B
20 a  C
 5 a  B
;;;;

data test2; set test;
   status =1;  * add a dummy status variable;
   if (x2= 'A') then x2a="xA"; else x2a=x2; * A is the reference;
   if (x2= 'B') then x2b="xB"; else x2b=x2; * make B the reference;
   if (x2= 'C') then x2c="xC"; else x2c=x2; * make C the reference;

proc phreg data = test2;
   class x1 x2a;
   model time * status(0) = x1 x2a x1*x2a/ type1;

proc phreg data = test2;
   class x1 x2b;
   model time * status(0) = x1 x2b x1*x2b/ type1;
```

```
proc phreg data = test2;
   class x1 x2c;
   model time * status(0) = x1 x2c x1*x2c / type1;

proc phreg data = test2;
   class x1 x2d;
   model time * status(0) = x1 x2c x1*x2c / type1;

proc phreg data= test2;
    class x1 x2/ param=effect;
    model time * status(0) = x1 x2 x1*x2 / type1;
```

The resulting 'joint tests' for covariate `x1` were 7.9, 5.4, 0.7, 2.4, and 5.3: coding x2 differently changes the results for x1! What are we to make of this? The sequential or "type 1" tests are identical for all 5 runs.

Linear Models

Type III tests and sums of squares can be traced back to a 1934 paper by F. Yates [4]. For a linear model that was fit to unbalanced data, he proposed using a population estimate for the main effects, where the "population" is an ideal experiment, e.g., a completely balanced factorial study. One can create his proposed estimates by a simple 3 step process.

1. Fit a linear model to the data at hand using ordinary least squares.
2. For each treatment arm, create the set of predicted values for that treatment, for all combinations of the other factors.
3. The average of these values is the estimated overall effect for the treatment.

```
> with(data1, table(x1, x2))
   x2
x1  A B C D
  a 3 3 3 3
  b 2 2 1 1
  c 1 2 2 1
```

In the data set used above there are four levels for `x2` for instance, so the effect for `x1='a'` will involve the average of 4 predicted values. The estimates are a prediction of what the average result for x1 *would have been* had the original data been balanced. SAS calls these averages *least squares means*. Modern causal modelers would call this a g-estimation method, though they, in turn, would almost never choose a factorial experiment as the underlying population.

The next logical step is to compute standard errors for each of these averages, along with a global test of whether the averages for x1=a, x1=b and x1=c are the same. The simplest method is to start by creating a matrix with all of the coefficients for the predicted values as rows, and then use matrix operations. This was not feasible in the computing environment of Yates' day, however, which led to a small industry of shortcut methods for producing a "Yates'

sum of squares". The SAS GLM algorithm is based on a particular one of these, namely that for a balanced factorial design, the sums-of-squares for all of the effects are orthagonal. Therefore, choose contrasts that are orthagonal to the variance matrix we *would have had* had the data been of this type. It can be reproduced using the following steps:

- Form a $Z'Z$ matrix from a *balanced subset* of the observations in the data set,
- create a set of contrasts that are orthogonal to $(Z'Z)^{-1}$
- apply those contrasts to the original data and fit, to get sums-of-squares for each term.

Here is an example of Yates' tests using a subset of the `solder` data set, based on the SAS algorithm. The `ctest` function contains standard linear models manipulations to compute the SS and test for a contrast.

```
> ctest <- function(contr, fit) {
      estimate <- drop(contr %*% coef(fit))
      var.estimate <- contr %*% vcov(fit) %*% t(contr)
      test <-  solve(var.estimate, estimate)%*% estimate
      list(estimate= estimate, variance= var.estimate, chisq = drop(test))
  }
> test <- subset(solder, Mask != "A6")
> table(test$Opening, test$Mask)   # the data is unbalanced
    A1.5  A3  A6  B3  B6
  L   60 120   0  60  60
  M   60  60   0  60  60
  S   60  90   0  60  60
> fit <- lm(skips ~ Opening*Mask, data=test, x=TRUE)  # ordinary linear model
> z.balance <- unique(fit$x)    # 12 unique combinations of Opening and Mask
> contr <- chol(crossprod(z.balance))   # cholesky decomposition of Z'Z
> ctest(contr[2:3,], fit)$chisq         # the test for Opening
[1] 542.4632
> ctest(contr[5:8,], fit)$chisq         # the test for Mask
[1] 340.7837
> ctest(contr[9:12,], fit)$chisq        # the test for Mask:Opening
[1] 153.7459
```

If $L'L = Z'Z$ is the Cholesky decomposition of $Z'Z$, then $L(Z'Z)^{-1}L' = I$, i.e., the Cholesky decomposition was simply a convenient way to create a set of contrasts that are orthogonal with respect to $(Z'Z)^{-1}$. Any full rank contrast matrix C such that $T = C(Z'Z)^{-}C'$ is block diagonal will give the same sums of squares, i.e., any matrix C such that $T_{ij} = 0$ whenever rows i and j correspond to different terms. The equivalence is a consequence of Cochan's theorem. Goodnight [2] states that any matrix which satisfies this condition defines a set of type 3 tests. The SAS technical report given as the usual reference for type 3 tests [3] describes a particular algorithm for creating such an orthogonal decomposition; it unfortunately depends on using an X matrix that is in exactly the same internal form as is used by the SAS GLM procedure. (The report is also remarkably opaque about exactly *what* the algorithm is computing.) Both the Cholesky

approach and the SAS GLM are invariant to how categorical factors are represented: first level as reference, last as reference, Helmert coding, etc.; all yield the same values.

If the linear model has both continuous and categorical predictors, then the `z.balance` line above needs to be modified to first omit any columns of X corresponding to continuous variables or interactions with continuous variables (but should retain the intercept): SAS "type 3" tests for continuous variables have no connection to Yates' method or population averages.

The bugaboo comes when the resulting $Z'Z$ matrix is singular. For example, if we use the entire solder data set, there are 0 observations in the (Mask=A6, Opening=S) cell, the coefficient vector from the `lm` fit will have an NA in that position, and an attempted Cholesky decomposition of $Z'Z$ will give an error message. In this case a Yates' population average for mask A6 is also undefined, since one of the predicted values in the average depends on the NA coefficient; SAS GLM for instance will report a missing value for that element of the least squares means. The Yates' sum of squares for comparing the 5 mask types is also undefined; nevertheless SAS GLM reports a type 3 sums of squares for the Mask effect! What appears to be happening is that SAS GLM is creating contrasts that are orthogonal with respect to a generalized inverse. The problem is that the resulting test statistics depend on exactly which generalized inverse is used; different g-inverse matrices lead to different values. There are an infinite number of generalized inverses for such a problem, the documentation in [3] does not give enough detail to exactly replicate the GLM algorithm for this case, and exact mimicry is the only way to exactly reproduce GLM type 3 values for an incomplete design.

An alternate "type 3" algorithm found outside of SAS is the following.

1. Create a design matrix X for the regression in standard order, i.e., from left to right are the intercept, columns for main effects, then 2-way interactions, then 3-way interactions, etc.

2. Proceeding from left to right, eliminate any columns which can be expressed as a linear combination of prior columns.

3. Define the RSS for any term as the difference between a fit using the full X matrix, and a fit eliminating those (remaining) columns that correspond to the term.

If the columns of X for the categorical variables in the model are coded using the summation constraint, and the Yates' SS is well defined (no missing cells), then, rather remarkably, half-baked approach will re-create the Yates sum-of-squares for each term. If any other method is used to generate the 0/1 columns for a categorical variable the results are meaningless, and in fact will often change by an order of magnitude across different codings. I refer to this as the not-safe type three (NSTT) algorithm. The `car` package in R uses this approach, for instance, and the package's documentation clearly states that models need to be fit with R's `contr.sum` option in effect.

Many of us have not seen the sum constraint since graduate school. As a reminder here is the two way form where ϵ is the residual error.

$$
\begin{aligned}
y_{ijk} &= \mu + \alpha_i + \beta_j + \gamma_{ij} + \epsilon_{ijk} \\
0 = \sum_i \alpha_i 0 \qquad & = \sum_j \beta_j 0 = \sum_i \gamma_{ij} \qquad = \sum_j \gamma_{ij}
\end{aligned}
$$

Assume a simple 2 factor model y = trt + x2 + interaction, where treatment has two levels A/B and x2 has k levels. In this simple case one can quickly verify that the the NSTT test for treatment is actually a comparison of A vs. B in the *reference* level for x2, ignoring all others, leading to k different results as we rotate the choice of reference. This is very different than the type 3 narrative of that it is a global test. When sum constraints are used the 'reference' for x2 will be an average and the NSTT matches the Yates' estimate. We suspect that the same general pattern extends to more complex models.

The flawed NSTT algorithm is, I believe, a primary source of a common critique that type 3 tests not marginal, i.e., that they are a test for main effects in the presence of interactions and are thus invalid. This has muddied the discussion of population contrasts terribly, and is in addition to the harm caused by misapplication of the algorithm: sum constraints are not the default in most packages and I daresay that many or even most computations have been wrong. (How many people read directions?)

Type 3 and phreg

The NSTT is precisely the algorithm used by the SAS phreg procedure, though without any warning about how to choose the coding for categorical predictors. This can be verified for the test example above by creating the appropriate `coxph` fits and performing Wald tests. For example, the first two fits below reproduce phreg results that used 'A' and 'D' as the reference levels. The `yates` function in the survival package produces Yates' population contrast; it's result is identical whether fit1 or fit2 is used as the input. The final phreg fit that used a sum constraint, referred to as "effects" coding in SAS, agreed with the Yates' result.

```
> contr.x1 <- matrix(0, nrow=2, ncol=11)
> contr.x1[1,1] <- contr.x1[2,2] <- 1
> fit1 <- coxph(Surv(y) ~ x1*x2, data=data1, ties='breslow')
> ctest(contr.x1, fit1)$chisq
[1] 7.93572
> options(contrasts=c('contr.SAS', 'contr.poly'))
> fit2 <- coxph(Surv(y) ~ x1*x2, data=data1, ties='breslow')
> ctest(contr.x1, fit2)
$estimate
[1] -2.1617601 -0.6966789

$variance
         [,1]     [,2]
[1,] 2.106360 1.246472
[2,] 1.246472 2.247885

$chisq
[1] 2.443343
> yates(fit1, ~x1, population="factorial")
 x1     pmm     std        test chisq df      Pr
  a 0.27782 0.65627      global 5.293  2 0.07089
```

```
b 1.85188 1.06082
c 0.95198 0.97138
```

References

[1] M. H. Gail, J. H. Lubin, and L. V. Rubinstein. Likelihood calculations for matched case-control studies and survival studies with tied death times. *Biometrika*, 68:703–707, 1981.

[2] J. Goodnight. Tests of hypotheses in fixed-effects linear models. Technical Report Technical Report R-101, SAS Institute, Inc., 1978. author formally changed to "SAS Institute" at a later date.

[3] SAS Institute Inc. *The four types of estimable functions.* SAS Institute, Inc., Cary, N.C., 2008. SAS/STAT 9.2 User's Guide: Chapter 15.

[4] F. Yates. The analysis of multiple classifications with unequal numbers in the different classes. *J. Amer. Stat. Assoc.*, 29:51–66, 1934.