Penalized Cox models and Frailty

Terry M Therneau Patricia M Grambsch

November 17, 1998

1   Introduction

A very general mechanism for penalized regression has been added to the coxph function in S-Plus. A user written S-Plus function can be supplied that gives additional term(s) to the partial-likelihood, along with the first and second derivatives of those terms. The variance and degrees of freedom for the extended model are then computed as outlined in Gray [9]. Several other arguments control optional aspects of the iteration. This setup allows for general shrinkage methods including ridge regression, the lasso, smoothing splines, and other techniques.

There is an interesting connection between penalized regression and random effects or *frailty* models. It happens that the gamma frailty model can be represented exactly as a penalized regression, and a Gaussian frailty can be represented approximately. Thus we can fit these models as well using the generalized program.

2   Shrinkage Estimation

2.1   Mathematical Basis

Consider a Cox model with both constrained and unconstrained effects

$$\lambda_i(t) = \lambda_0(t)e^{X_i\beta + Z_i\omega}$$

where X and Z are the covariates and β, ω are the unconstrained and constrained coefficients, respectively. The problem is solved by maximizing a penalized partial likelihood

$$PPL = PL(\beta, \omega; \text{data}) - f(\omega; \theta)$$

over both β and ω. Here PL is the usual Cox partial likelihood, treating ω as "just another parameter", and f is some constraint function which gives large values to

1

"bad" values of ω. For the moment assume that θ, a vector of tuning parameters, is known and constant.

Following Gray [9], let $\mathcal{I}$ be the usual Cox model information matrix, and

$$H = \mathcal{I} - \begin{pmatrix} 0 & 0 \\ 0 & f'' \end{pmatrix}$$

be the second derivative matrix for the penalized likelihood PPL. His suggested estimate of the variance is

$$V = H^{-1} \mathcal{I} H^{-1} \tag{1}$$

Let c be a column vector of constants, and (β', ω') be the combined vector of $p+q$ parameters. Then for a general test of hypothesis $z = (\beta', \omega')c = 0$ Gray recommends the Wald test $z'(c'H^{-1}c)^{-1}z$. Because of the shrinkage, this is not necessarily a chi-square statistic. Let e be the eigenvalues of the matrix $(c'H^{-1}c)^{-1}(c'Vc)$; then under H_0 the Wald test is distributed as as $\sum e_i X_i^2$ where the X are iid Gaussian random variables. Let $k = \sum e_i$. When the e_i are all 0 or 1, the case for non-penalized models, then the mean and variance of the test statistic are k and $2k$ respectively, and the distribution is chi-square on k degrees of freedom. In penalized models $e_i \le 1$ and the variance is $\sum 2e_i^2 < 2k$; so the distribution of the statistic is more compact than a standard chi-square test based on k degrees of freedom and the test will be conservative.

The generalized degrees of freedom for the test statistic can be written as

$$df = \text{trace}[(c'H^{-1}c)^{-1}(c'Vc)]$$

so computation of eigenvalues is not stricly necessary. For a particular term in the model this becomes $\text{trace}((H^{-1}[i,i])^{-1}V[i,i])$ where [ ] are S-Plus style subscripts and i indexes the columns corresponding to the term.

An alternate variance estimator is to use H^{-1} directly, the inverse of the second derivative matrix of the full log likelihood, which is the variance used in the Wald statistic. It has an interpretation as a posterior variance in a Bayes setting. It also tends to be larger than V and thus more conservative. Wahba [26] showed it had good frequentist coverage properties for pointwise intervals for the smoothing spline curve fit to noisy data. In the context of smoothing, Hastie and Tibshirani [11] (page 60) compare confidence intervals based on the analog of V with those based on the analog of H and show that H has a component of bias built into it. They further suggest that with small degreees of freedom for the smoother, the two are likely to be very similar but differ more as there are more degrees of freeedom. In Statistical Models in S [3] ,chapter 7 where they discuss the implementation of GAM, they indicate (p 303-4) that in computing standard errors for the smooth

they actually use the analog of H rather than V. Here they justify it on the grounds of computational simplicity.

The S-Plus function returns both `var2`=$H^{-1}\mathcal{I}H^{-1}$ and `var`=H^{-1}. The chi-square tests are based on `var`. Simulation experiments (see appendix xxx) suggest that this is the more reliable choice for tests.

2.2   S-Plus functions

Penalized likelihoods for the Cox model have been implemented in S-Plus in a very general way. The iteration depends on two user defined functions, a control function 'cfun' and a penalty function 'pfun'. If there are multiple penalized terms, e.g., smoothing splines on two distinct variables, then each term has it's own pair of functions, but for the moment assume only a single penalized term. The algorithm is

1. On the initial call (with `iteration=0`) the control function `cfun` returns an initial value for θ.

2. The penalized likelihood is solved, for fixed θ, using a Newton-Rhapson iteration. Repeated calls to the penalty function `pfun` are used to obtain necessary values of f and its first and second derivatives.

3. The control function `cfun` is called to obtain both the next value for θ and a flag indicating whether iteration is complete. If iteration is not complete, return to step 2.

The algorithm thus consists of an outer and an inner loop, and the returned value of `iter` is a vector of length 2 giving the number of outer and inner iterations, respectively. There are at least three distinct types of outer loop: θ fixed, in which the control function does nothing; calibration problems, where the parameter is fixed by the user but is on a different scale from the internal θ; and actual iteration, such as the use of generalized cross-validation (GCV) to choose an 'optimal' θ. The variance formula used by the routine assumes a fixed value of θ, and so is not correct for the third case. Nevertheless, it seems to be fairly accurate in several instances. For many of the problems considered here, the program is fast enough that more reliable variance estimates could be obtained via resampling techniques such as the bootstrap.

We will start with a simple example. Let $f(\omega, \theta) = (\theta/2) \sum \omega_j^2$, a penalty function which will tend to shrink the coefficients ω_j towards zero. The penalty and control functions are quite simple in this case:

```
pfun.ridge <- function(coef,theta) {
                       list(penalty= sum(coef^2)*theta/2,
                            first  = theta*coef,
                            second = rep(theta, length(coef)),
                            flag=F)
                            }


cfun.ridge <- function(parms, ...) list(theta=parms$theta, done=T)
```

This psuedo ridge-regression function is simplistic: no provision has been made for factor (classification) variables and there is no scaling of the penalty with respect to the scale of the covariates. We will improve on these aspects later. A third "packaging" function `ridge` is also needed, which passes through the data, adding attributes that identify the above as the penalty and control functions. Details of the packaging function are discussed in appendix C.

The penalty function is called with the coefficients *for the term*, e.g., those corresponding to `age` and `ecog.ps` in the example below, along with the tuning parmeter(s) θ. It needs to return the value of the penalty and its first and second derivatives. For some penalty functions and values of θ the penalty may be infinite, in which case the flag argument should be set to True. (We will see this in a later example.) In this example f'' is diagonal, and so pfun returns only a vector of second derivatives. In other cases, such as smoothing splines, pfun will need to return a second derivative matrix.

Here is an example of using the ridge functions. The data set is from Edmunson et. al [6], and gives the survival time of 26 women with advanced ovarian carcinoma, randomized to two treatments. Important covariates are the patient's age and performance score. The latter is a measure of physical debilitation with 0=normal and 4=bedridden. The value of $\theta = 1$ used for the shrinkage parameter was chosen arbitrarily.

```
> fit0 <- coxph(Surv(futime, fustat) ~ rx + age + ecog.ps, data=ovarian)
> fit1 <- coxph(Surv(futime, fustat) ~ rx + ridge(age, ecog.ps, theta=1),
                data=ovarian)

> fit0
           coef exp(coef) se(coef)     z      p
     rx -0.815     0.443   0.6342 -1.28 0.2000
    age  0.147     1.158   0.0463  3.17 0.0015
ecog.ps  0.103     1.109   0.6064  0.17 0.8600

> fit1
```

```
                  coef   se(coef)    se2   Chisq DF      p
          rx  -0.8124   0.6333    0.6327  1.65   1   0.2000
   ridge(age)  0.1468   0.0461    0.0461 10.11   1   0.0015
ridge(ecog.ps)  0.0756  0.5177    0.4429  0.02   1   0.8800


Iterations: 1 outer, 4 Newton-Raphson
Degrees of freedom for terms= 1.0 1.7
Likelihood ratio test=15.9  on 2.73 df, p=0.000875  n= 26
```

The likelihood ratio test that is printed is twice the difference in the PL between the null model ($\beta = \omega = 0$) and the final fitted model. The p-value is based on comparing this to a chisquare distribution with 2.73 degrees of freedom. As mentioned earlier this comparison is somewhat conservative (p too large). The eigenvalues for the problem, `eigen(solve(fit1$var, fit1$var2))`, are 1, 0.9156 and 0.8486. The respective quantiles of this weighted sum of squared normals and the chi-square distribution `qchisq(q, 2.73)` are

	80%	90%	95%	99%
Actual sum	4.183	5.580	7.027	10.248
$\chi^2_{2.73}$	4.264	5.818	7.337	10.789

from which we see that the actual distribution is somewhat more compact than the Chi-square approximation.

The shrinkage has had a much smaller effect on age than on the ecog score. Although the unpenalized coefficients for the two covariates are of about the same magnitude (`fit0`), the standard error for ecog score is much larger. The impact on overall fit (Cox PL) of shrinking the age coefficient will thus be larger than that for ecog score; the age coefficient is "harder to change".

One improvement to the function would be to scale the penalty for each variable by its variance, so that the function will be invariant to the units of the data; the above fit would change if age were given in days, for instance. This is taken up in appendix C.

2.3 Spline fits

We now explore a more complicated example, which is to fit a general spline term. The method we will use is P-splines [7]. Start by spanning the range of x with a b-spline basis, such that the basis functions are evenly spaced and identical in shape. This differs from the traditional b-spline basis for smoothing splines, which has an asymmetric basis function (knot) for each data point. An example fit for 11
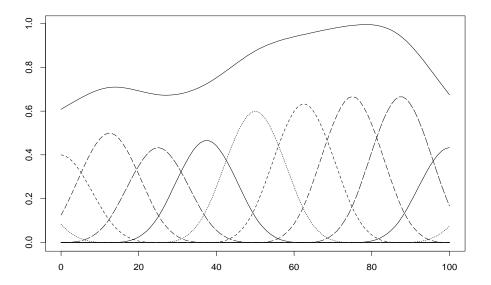
Figure 1: *A representative set of weighted p-splines*

'terms' is shown in figure 1; the heights of the basis functions are the coefficients, and the sum of the basis functions is the fit.

Several authors have noted that for moderate degrees of freedom, a smaller number of basis functions give a fit which is nearly identical to the standard smoothing spline. Gray [9] suggests that there is little advantage to using more than 10-20 knots and uses 10 knots for his 3 degree of freedom simulations and examples. Hastie [10] uses a more sophisticated eigenvector approach to find a nearly optimal approximating set of basis functions; for several examples with 4-5 degrees of freedom his basis set has 7-8 terms. The functions below use `round(2.5*df)` terms by default but the user can adjust this parameter.

P-splines have several attractive properties, one of which was the key reason for their inclusion here. Because of the symmetry of the basis functions, the usual spline penalty $\theta \int [f''(x)]^2 dx$ is very close to the the sum of second differences of the coefficients $\theta*$`sum((diff(diff(coef)))`$^$`2)`, and this last is very easy to program. Let T be the matrix of second differences, e.g., for 6 coefficients T is

$$\begin{pmatrix} 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \end{pmatrix}.$$

Then with $P \equiv T'T$ the p-spline penalty is $f(\omega, \theta) = \theta \omega' P \omega$. The first derivative of the penalty is $2\theta P \omega$ and the second derivative is $2\theta P$. This extends easily to a penalty based on third differences. Other properties of note are

- The penalty does not depend on the values of the data x, other than for establishing the range of the spline basis.

- If the coefficients are a linear series, then the fitted function is a line. Thus a linear trend test on the coefficients is a test for the significance of a linear model. This makes it relatively easy to testt for the significance of non-linearity.

- Since there are a small number of terms, ordinary methods of estimation can be used, i.e., the program can compute and return the variance matrix of $\hat{\beta}$. Contrast this to the classical smoothing spline basis, which has a term (knot) for each unique x value. For a large sample size storage of the n by n matrix H becomes infeasable.

Pat - should we just delete the following example. It no longer seems important to the flow?

The following function creates the spline basis suggested by Eilers et. al. [7], and draws the figure 1. It makes use of the internal S-Plus function `spline.des`.

```
pspl <- function(x, df, degree, nterm=round(2.5*df)) {
    lower <- min(x, na.rm=T)
    upper <- max(x, na.rm=T)
    if (nterm < 3) stop("Too few basis functions")
    dx <- (upper-lower)/nterm
    knots <- seq(lower - degree*dx, upper + degree*dx, by=dx)
    spline.des(knots, x, degree+1)$design
    }

xx <- 0:100
yy <- pspl(xx, degree=3, nterm=8)

coef <-  c(10,12,15,13,14,18,19,20,20,13,9)/20
yy <- cbind(yy %*% coef, yy %*% diag(coef))
matplot(xx, yy, type='l', col=1)
```

The S-Plus function to impliment P-spline fits, `pspline`, has 3 optional parameters:

- The degree of the spline, with cubic splines as the default.

- The desired degrees of freedom for the fit. Optionally, the user can specify θ directly.

- The number of basis functions or terms.

The actual penalty used by the function is $[\theta/(1 - \theta)]\omega'P\omega$ The first term was changed for user convenience: $\theta = 1$ now corresponds exactly to the straight line model (an infinite penalty for curvature).

2.4 Example

Consider the ovarian data included with S-Plus, and fit 3 models.

```
> fit1 <- coxph(Surv(futime, fustat) ~ rx + age, ovarian)
> fit2 <- coxph(Surv(futime, fustat) ~ rx + pspline(age, df=2),
                data=ovarian)
> fit4 <- coxph(Surv(futime, fustat) ~ rx + pspline(age, df=4),
                data=ovarian)
> fit1
       coef exp(coef) se(coef)     z      p
 rx -0.804     0.448   0.6320 -1.27 0.2000
age  0.147     1.159   0.0461  3.19 0.0014

Likelihood ratio test=15.9  on 2 df, p=0.000355  n= 26



> fit2
        coef se(coef)   se2 Chisq DF      p
  rx -0.589 0.699     0.679 0.71  1  0.400
age2  0.790 1.379     0.620 0.33  1  0.570
age3  1.539 2.143     1.210 0.52  1  0.470
age4  2.206 2.401     1.630 0.84  1  0.360
age5  3.066 2.406     1.808 1.62  1  0.200
age6  4.463 2.394     1.863 3.48  1  0.062
age7  5.990 2.479     1.983 5.84  1  0.016
age8  7.475 2.911     2.275 6.59  1  0.010

Iterations: 2 outer, 7 Newton-Raphson
Degrees of freedom for terms= 0.9 1.9
Likelihood ratio test=17  on 2.87 df, p=6e-04  n= 26

> fit2$history[[1]]
$theta:  0.44688
```

```
$done:    T
$thetas: 0.000000 1.000000 0.600000 0.484520
$dfs:     5.000000 1.000000 1.734267 1.929305
```

The printout for the simple Cox model shows an increase in the log-hazard for death of .147 per year of age, with an overall chi-square for the model of 15.9. The p-spline basis functions sum to a constant, so the first one of them is deleted to remove the singularity. The actual values of the coefficients are not very useful, other than that one can see an overall linear trend. The spline fit with 2 degrees of freedom has not improved this significantly; the likelihood based test for non-linearity would be a chisquare of $17 - 15.9 = 1.1$ on '0.87' degrees of freedom. We have explicitly printed out the `history` component of the final model, which contains the last return value(s) of the control function for the problem. In the case of `cfun.ps`, the elements are `theta`, a flag indicating that iteration was completed, and the list of successive θ values tried by the routine in it's attempt to achieve a fit with the requested degrees of freedom. The routine assumes, without computation, that a penalty of 1 will give a linear fit (1 df), while a penalty of 0 gives a fit with 5 df. There is no good theoretical reason for the value of "5" (number of basis functions - degree of the spline), but it seems to work well as a generator of the initial guess of $\theta = 0.6$. Using these starting guesses, it took one more iteration to find the value of $\theta = .48$ leading to 1.93 df, which is within the default tolerance of 0.1 df used by the `pspline` function. The values of θ that were attempted required 4 and 3 Newton-Raphson steps, respectively, leading to 2 outer and 7 inner iterations. The *next* value of θ that would have been used in iteration was .44688; however, it was not attempted because the routine considered 1.93 sufficiently close to the target value of 2 degrees of freedom.

A plot of the new fit is easily obtained, and is shown in Figure 2. It was produced by the code below. The addition of confidence bands would be straightforward using the `se.fit` argument of `predict`. The picture verifies what we had seen in the tests, that there is not an important non-linear component to the age effect in this data set.

```
> xx <- ovarian$age
> yy <- cbind(predict(fit1, type='terms')[,2],
              predict(fit2, type='terms')[,2],
              predict(fit4, type='terms')[,2])
> temp <- order(xx)
> matplot(xx[temp], yy[temp,], type='l', xlab='Age', ylab='Risk Score')
```

The routine as distributed has a more refined printout than that shown above. Here is the actual printed result for the 4 degree of freedom fit.
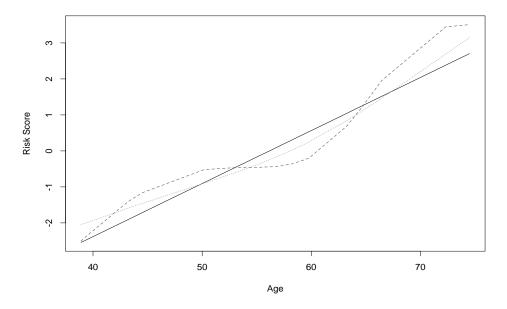
Figure 2: *Spline fits for the ovarian data*

```
> fit4
                              coef se(coef)     se2 Chisq    DF      p
                    rx -0.373 0.761       0.749 0.24  1.00 0.6200
pspline(age, df = 4), linear   0.139 0.044       0.044 9.98  1.00 0.0016
pspline(age, df = 4), nonlin                      2.59  2.93 0.4500


Iterations: 3 outer, 13 Newton-Raphson
         Theta= 0.26
Degrees of freedom for terms= 1.0 3.9
Likelihood ratio test=19.4  on 4.9 df, p=0.00149  n= 26
```

There are actually 13 coefficients associated with the 4 degree of freedom spline for age. These have been summarized in the printout as a linear and nonlinear effect. Because of the symmetry of the p-spline basis functions, the chi-square test for linearity is a test for zero slope in a regression of the spline coefficients on the centers of the basis functions, using `var` as the known variance matrix of the coefficients. The linear 'coefficient' that is printed is the slope of this regression. This computation of coefficient and p-value is equivalent to the approximate backwards elimination method of Lawless and Singhal [14], here removing all of the non-linear terms for age. (By fitting a non-spline model ∼ `rx + age` we find that the true linear coefficient is .147 and chisquare for non-linearity is 3.5.; the approximation

10

was reasonably good.) If the terms being dropped are important, i.e. a significant non-linearity, then the approximation for the linear coefficient is not as accurate.

Pat, the next paragraph needs help!

The degrees of freedom for the terms are the results of Gray's formula. We have applied this formula both to the penalized and to the unpenalized terms. This gives a value of 0.9 df for the `rx` term, when we know that the 'true' df for this term is 1 since it is unpenalized. (Or is it unpenalized, since it is not uncorrelated with the age term?) If nothing else, the devation of 0.1 can be viewed as a measure of accuracy for the degrees of freedom computation.

Using multiple spline terms, we are able to investigate models that are similar to the *Generalized Additive Models* [11] available for binary and other exponential family data using the `gam` function of S-Plus. As a more interesting example, we look at data from the multi-center post-infarction project (MPIP) [21]. This contains data on 866 patients, gathered after hospital admission for a myocardial infarction. The main goal of the study was to ascertain which factors, if any, were predictive of the future clinical course of the patients. Four variables will be used in the model of survival time:

- VED, ventricular etopic polarizations per hour, obtained from analysis of a 24 hour Holter monitor. A large number of these irregular heartbeats is indicative of high risk for fatal arrythmia.

- New York Heart Association class, a measure of the amount of activity that a subject is able to undertake without angina, ranging from 1 to 4.

- Presence of pulmonary rales on initial examination.

- Ejection fraction, the proportion of blood cleared from the heart on each contraction.

VED is very skewed; it has a mean value of 19.1, a median of .45, a maximum value of 733, and 14% of the subjects have a value of 0. The minimum non zero value is 0.042, so we use `lved = log(ved+.02)` as a derived covariate. It is still a skewed variable, but is not unmanagably so. A simple linear fit of the four variables shows all to be highly significant.

```
> fit1 <- coxph(Surv(futime, status) ~ lved + nyha + rales +ef, mpip)
> fit1
          coef exp(coef) se(coef)    z       p
 lved  0.1007     1.106  0.04266  2.36 1.8e-02
 nyha  0.3707     1.449  0.09379  3.95 7.7e-05
rales  0.4535     1.574  0.10528  4.31 1.7e-05
```

```
     ef -0.0265    0.974  0.00833 -3.18 1.5e-03

 Likelihood ratio test=79.4  on 4 df, p=2.22e-16  n=764
    (102 observations deleted due to missing)
```

Next, let us explore more complicated forms for the effect of the covariates. Since `rales` is a binary covariate it allows no further transformation, and `nyha`, with four levels, will be entered as a factor variable. The two continuous variables, `lved` and `ef`, are modeled as p-splines with the default (4) degrees of freedom.

```
> fit2 <- coxph(Surv(futime, status) ~ pspline(lved) + factor(nyha) +
                     rales + pspline(ef), mpip)
>fit2
                         coef se(coef)      se2 Chisq   DF       p
pspline(lved), linear  0.0982  0.04384  0.04359  5.02 1.00 0.02500
pspline(lved), nonlin                            2.59 3.06 0.47000
        factor(nyha)2 -0.0615  0.31835  0.31780  0.04 1.00 0.85000
        factor(nyha)3  0.6971  0.31853  0.31729  4.79 1.00 0.02900
        factor(nyha)4  1.0151  0.29218  0.29113 12.07 1.00 0.00051
                rales  0.4204  0.10816  0.10761 15.11 1.00 0.00010
   pspline(ef), linear -0.0256  0.00738  0.00737 12.03 1.00 0.00052
   pspline(ef), nonlin                           8.06 3.01 0.04500

Iterations: 4 outer, 11 Newton-Raphson
 Penalized terms:
        Theta= 0.767
        Theta= 0.658
Degrees of freedom for terms= 4.1 3.0 1.0 4.0
Likelihood ratio test=92.5  on 12.04 df, p=1.69e-14  n=764
    (102 observations deleted due to missing)
```

It would appear that NYHA classes 1 and 2 might be combined, that the nonlinear effect for VED is not significant, and that the non-linear effect of ejection fraction is important. Plots of the two spline terms are shown in figure 3 and are produced with the following commands.

```
> temp <- predict(fit2, type='terms', se.fit=T)
> tmat <- cbind(temp$fit[,1], temp$fit[,1] + 1.96*temp$se.fit[,1],
                         temp$fit[,1] - 1.96*temp$se.fit[,1])
> jj <- match(sort(unique(lved)), lved)
> matplot(lved[jj], tmat[jj,], type='l', lty=c(1,2,2), xaxt='n')
> xx <- c(0, 1, 50, 100, 500)
> axis(1, log(xx+.2), as.character(xx))
```

12
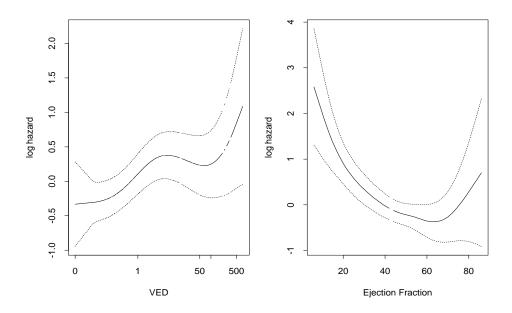
Figure 3: *Spline fits for the MPIP data*

```
> title(xlab='VED', ylab='log hazard')

> tmat <- cbind(temp$fit[,2], temp$fit[,2] + 1.96*temp$se.fit[,2],
                               temp$fit[,2] - 1.96*temp$se.fit[,2])
> jj <- match(sort(unique(mpip$ef)), mpip$ef)
> matplot(mpip$ef[jj], tmat[jj,], lty=c(1,2,2),
          xlab='Ejection Fraction', ylab='log hazard')
```

Some extra work was required to label the first graph in the original VED units, this is done with the `axis` command. The `match` function and the `jj` subscripts sort the plot from left to right, otherwise the line becomes a back and forth scribble. We see from the graph that there is an increase in risk with ejection fractions below 60%, sharply so below 20%. The rise after 70% is not significant based on the wide confidence intervals, this agrees with the conventional wisdom of the physicians that the instumentation is not able to reliably distinguish values above this level.

2.5 Automatic selection of the degrees of freedom

There are several methods of automatically choosing the amount of smoothing in a spline fit including cross-validation, generalized cross validation (GCV) and Bayesian approaches. One of the easiest in this programming context is to use the

13

Akaike informaion criteria

$$\text{AIC} = 2[\log(PL\{\hat{\beta} = \text{initial}\}) - \log(PL\{\hat{\beta} = \text{final}\})] - 2df\,. \qquad (2)$$

Notice that this uses the ordinary partial likelihood without the penalty term, with degrees of freedom as a correction rather than the penalty. Hurvich, Simonoff and Tsai [12] show that in rich non-parametric regression model the AIC can under-penalize, however, leading to models with an excess number of degrees of freedom. They suggest a corrected AIC, which uses $n(df+1)/(n-(df+2))$ as the correction term in place of df. In the case of a Cox model we replace n by the total number of events. Practically, AIC_c favors smaller models than AIC, with a bias that grows with df, i.e, it might choose 1.9 rather than 2 df and 10 rather than 25 df. The parameter `df=0` directs the `pspline` routine to select a control function that maximizes AIC, with `caic=T/F` as an optional argument to choose either the corrected or uncorrected form of the statistic.

For the lung cancer data either criteria chooses 1 degree of freedom for the age effect. For the MPIP data we get the following.

```
> coxph(Surv(futime, status) ~ lved + factor(nyha) +
        pspline(ef, df=0), data=mpip)


                           coef se(coef)     se2 Chisq   DF        p
                 lved    0.1021  0.04173  0.04169  5.99 1.00 1.4e-02
         factor(nyha)2  -0.0207  0.31674  0.31664  0.00 1.00 9.5e-01
         factor(nyha)3   0.7051  0.31239  0.31216  5.09 1.00 2.4e-02
         factor(nyha)4   1.0413  0.28822  0.28705 13.05 1.00 3.0e-04
pspline(ef, df=0, linear -0.0360  0.00739  0.00739 23.73 1.00 1.1e-06
pspline(ef, df=0, nonlin                           6.55 1.18 1.4e-02


Iterations: 15 outer, 39 Newton-Raphson
     Theta= 0.989
Degrees of freedom for terms= 1.0 3.0 2.2
Likelihood ratio test=72.1  on 6.17 df, p=1.93e-13
  n=764 (102 observations deleted due to missing)
```

The result is shown in figure 4. The AIC_C criterion has chosen 2.2 degrees of freedom for the spline term, which has essentially removed the upward jump of the right hand tail found in figure 3.

We had already concluded that lved had no non-linear effect. An attempt to use the AIC criteria to choose the smoothness of 2 terms at once however, `pspline(lvef, df=0) + pspline(ef, df=0)` is likely to be unsuccessful in general. Each term is being
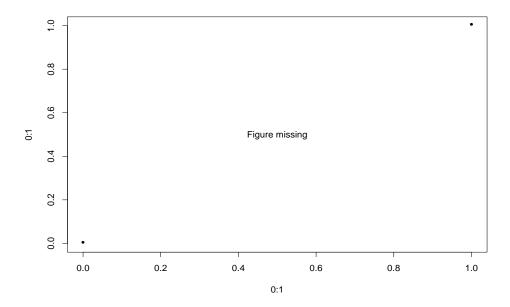
Figure 4: *Spline fit with degrees of freedom chosen by AIC_C*

controlled by a separate 1-paramteter minimizer, and unless the terms are nearly independent (and these two cardiac tests clearly are not) simultaneous univariate minimization is not a good strategy for finding the global maximum of a function, even though it is quite reliable for the bivariate *calibration* problem that arises when the df values are specified.

3 Frailty

In the last several years there has been significant and active research on the addition of random effects to survival. The random effect is usually viewed as a categorical variable which describes excess risk or *frailty* for an individual or family. The idea is that individuals have different frailties, and that those who are most frail will die earlier than the others. Aalen [1] discusses the impact of such heterogeneity on analyses, and includes several examples along with an overview of the early literature. He gives a good discussion of why such models are both of interest and practical utility.

Computationally, the frailty is usually viewed as an unobserved covariate. This has led naturally to the use of the EM algorithm as an estimation tool. However, the algorithm is slow, the proper variance estimate is uncertain, and no implimentation

has appeared in any of the more widely available packages.

Assume a proportional hazards model with random effects or *frailties*, with hazard function

$$\lambda_i(t) = \lambda_0(t)e^{X_i\beta + Z_i\omega} \, .$$

Here β is a vector of p fixed effects and ω a vector of q random effects, where the individual elements ω_j are iid realizations from some distribution $W(\theta)$. The matrix X will normally contain measured covariate values, and Z will be a design matrix that describes how the random effects apply to individual subjects. Both X and Z might contain time dependent effects, but we will ignore this complication for the moment. The baseline hazard may contain other parameters ξ; these will also be ignored. Note that if X contains an intercept term (which is implicit for the proportional hazards model), we can constrain ω to have mean 0.

We can treat the random effects as unobserved data and apply the EM algorithm. The 'x' of the formal EM argument is the entire observed data (time, status, covariates) *plus* the frailties, and 'y' is the data without the frailties. The full log-likelihood, if we had observed ω, is

$$
\begin{aligned}
L_f &= \sum_{j=1}^{q} \log(W(\omega;\theta)) \\
&+ \sum_{i=1}^{n} \delta_i[\log(\lambda_0(t_i)) + X_i\beta + Z_i\omega] - \Lambda_0(t_i)e^{X_i\beta + Z_i\omega}
\end{aligned}
$$

Here $\delta_i = 0$ for censored observations, and 1 for events. X will be an n by p matrix and Z will be n by q.

This way of setting up the problem is similar in notation to random effects models in linear regression. Another notation, which is more common in the survival literature, is to define $\varpi = \exp(Z_i\omega)$ as the frailty parameter for each subject. Then

$$\lambda_i(t) = \varpi_j \lambda_0(t)e^{X_i\beta},$$

subject i being a member of the jth family. The imposed constraint is usually $\mathrm{E}(\varpi) = 1$ rather than $\mathrm{E}(\omega) = 0$.

The most popluar choice for the random distribution is the gamma frailty model, where ϖ is from a Gamma distribution with variance mean 1 and variance $\theta = \sigma^2$. The details of the EM approach for survival data and for the gamma frailty model in general are found in appendix B. This shows that the marginal likelihood L_g after integrating out the frailty is

$$L_g = \text{Cox PL} + \sum_j \nu \log\left(\frac{\nu}{\nu + E_j^*}\right) + \log\left(\frac{\Gamma(\nu + D_j)}{\Gamma(\nu)}\right) - D_j \log(\nu + D_j). \quad (3)$$

16

By "Cox PL" in the above we mean the numerical value returned as the partial likelihood by a standard Cox model program for the given values of β and ω, ω having been entered as an offset term. This derivation only applies to the simple frailty problem where each subject i is a member of exactly 1 family j, with a random effect per family. Then D_j is the number of events in the family, and $E_j^* = E_j/\exp(\omega_j)$ where E_j is the expected number of events for the family, using the final model.

3.1 Computation

Some care must be taken with computing the log-likelihood as ν goes to infinity, in which case the frailty goes to zero and we should converge to the results of an ordinary, non-frailty Cox model. Equation (3) converges in this particular case to Cox PL $-\sum D_j$. For this reason, we have *added* $\sum D_j$ to the value of L_g in the S-Plus code, to make the results of frailty and non-frailty models appear comparable on the printout.

The computation of the correction terms in (3) must be done with care for them to converge to $\sum D_j$ as desired. The first correction term, $\nu \log[\nu/(\nu+E_j)]$ becomes numerically problematic on a Sun workstation for $\nu > 10^9$ or so. Such a value is unlikely to occur during iteration, but if so the term could be replaced at that point by it's Taylor series expansion $-E_j/(1+E_j/\nu)$. Because the martingale residuals $D-E$ sum to zero, for $\nu = \infty$ this converges to the sum of the events.

The more problematic terms, numerically, are

$$\log\left(\frac{\Gamma(\nu+D_j)}{\Gamma(\nu)}\right) - D_j \log(\nu+D_j).$$

There are three possible ways to compute this; as the expression above, as the difference in $\log\Gamma$ functions, which is an elementary function in S, or using a recurrence formula for the Γ function to show that

$$\frac{\Gamma(\nu+D_j)}{\Gamma(\nu)} = \prod_{k=1}^{D_j} \nu + D_j - k$$

On a Sun workstation the first method is unstable for $\nu > 100$, the second for $\nu > 10^7$.

Finally, we can combine the $\log\Gamma$ and logarithmic terms together giving

$$\log\left(\prod_{k=1}^{D_j} \frac{\nu+D_j-k}{\nu+D_j}\right). \tag{4}$$

For large ν this form will be well behaved, and clearly goes to zero in the limit.

4 Frailty and penalized likelihoods

4.1 Derivation

There is an interesting connection between the frailty models and penalized like-
lihoods. In particular, let the penalty function for a constrained solution be the
log-gamma density

$$-f(w; \nu) = \nu(w - e^w) + \nu \log(\nu) - \log \Gamma(\nu),$$

with $\theta = 1/\nu$ as the variance of the random effect and with Z defined as in the
frailty model. The first and second derivatives are $\nu(1 - \exp(\omega))$ and $-\nu \exp(\omega)$,
respectively.

Surprisingly, for any fixed value of ν the EM algorithm and this contrained
minimization have the same solution. This was discovered by accident in some
tests, but can easily be verified by rewriting the formula for L_g. Combine the
definition of the PLL

$$PLL = \text{Cox PL} + \sum_{j=1}^{q} \nu(w_j - e^{w_j}) + \nu \log(\nu) - \log \Gamma(\nu)$$

with equation 18, using equation 16 to eliminate E_j^*. Per the prior section, we want
to maximize $L_g + D$ where D is the total number of events. Then

$$
\begin{aligned}
L_g + D &= \text{PLL} + \sum_{j=1}^{q} [\nu e^{\omega_j} + D_j + \log \Gamma(\nu + D_j) - (\nu + D_j) \log(\nu + D_j)] \\
&= \text{PLL} + \sum_{j=1}^{q} \nu + D_j + \log \Gamma(\nu + D_j) - (\nu + D_j) \log(\nu + D_j) \quad (5)
\end{aligned}
$$

The second step above can be made because of a constraint: throughout the inter-
ation the solution values will satisfy $\text{E}(\varpi_i) = 1$. Since the correction terms involve
neither β nor ω, we see that L_g and the PLL must have the same maximum.

This connection between the two methods has several interesting consequences.

- Since penalized likelihood methods are well understood numerically, this leads
  to more stable computational methods. In particular, it fits in nicely to the
  new `coxph` function.

- Equation (5) can also be viewed as the objective function from an empirical
  Bayes model, with a gamma prior on ω and a hyperprior on ν that is a product
  of q densities each of the form $e^x \Gamma(x)/x^x$, where $x = \nu + D_j$. This density is
  extremely long tailed.

18

- There is a connection to the "degrees of freedom" for a fit.

- It suggests a heuristic approach for other frailty distributions and/or frailty terms, e.g. nested models, for which the EM mathematics is not tenable.

An example of the last point is found in McGilchrist and Aisbet [18, 17], who use a Gaussian density with variance σ^2 as the penalty function f. So

$$PLL = \text{Cox PL} - (1/2\sigma^2)\sum_{j=1}^{q}\omega_j^2 \tag{6}$$

There remains the question of how to choose the variance or "shrinkage" paramter σ. There is no exact connection to frailty models as there was with the gamma distribution. However, the authors note the similarity of the Cox model's Newton-Raphson step to an interatively reweighted least-squares calculation, and using this as a basis they propose using standard estimators from Gaussian problems. External to the modified Cox program, σ is chosen to satisfy

$$\sigma^2 = \frac{\sum_{j=1}^{q}\omega_j^2}{q-r}$$

where r is

- BLUP estimate: r=1

- ML estimate: $\text{trace}[(H_{22})^{-1}]/\sigma^2$

- REML estimate: $\text{trace}[(H^{-1})_{22}]/\sigma^2$.

It should be possible to make this derivation more precise using a Laplace approximation, as in Breslow and Clayton [2], but the task is a little beyond me. McGilchrist and Yau [19, 27] generalize the REML method to a more general case. Assume that the variance of the random effect is $\sigma^2 A$ for a known matrix A. Then the penalty is $\omega'A^{-1}\omega/2\sigma^2$, and the REML estimate satisfies

$$\sigma^2 = \frac{\omega'A^{-1}\omega + \text{trace}[A^{-1}(H^{-1})_{22}]}{q},$$

which is equivalent to the above formula when A is the identity matrix. (The ML method would use $(H_{22})^{-1}$.) However, in simulations they find the REML to be less biased than the ML method.

4.2 Sparse computation

One important computational addition has been made to the underlying code specifically to support the frailty computations, represented by the `sparse` argument (although it may be useful in other contexts as well). Assume a single frailty term of the usual type, $Z_{ij} = 1$ if subject i is a member of group j, and zero otherwise. If we partition the Cox model's information matrix according to the rows of X and Z

$$\mathcal{I} = \left(\begin{array}{cc} \mathcal{I}_{XX} & \mathcal{I}_{XZ} \\ \mathcal{I}_{ZX} & \mathcal{I}_{ZZ} \end{array} \right)$$

then the lower right corner will be a diagonally dominant matrix. (It has almost the form of the variance matrix for a multinomial). The added penalty is also diagonal.

If sparse computation is elected, then the underlying programs retain only the diagonal of $\mathcal{I}_{ZZ}$.

- The savings in space can be considerable, particularly in the case of a frailty term per subject. Assume that there were 300 subjects and 4 other variables (age, sex, ...). Then the full matrix will have $304^2 = 92416$ elements, but the sparse version retains only the left hand "slice" of $304 * 4 = 1216$ elements.

- Because the score vector and likelihood are not changed, the solution point is identical. The Newton-Raphson iteration may undergo a slight loss of efficiency so that 1-2 more iterations are required. However, because each NR iteration requires the Cholesky decomposition of the information matrix, the sparse problem is much faster per-iteration than the full matrix version. (We sweep out the sparse rows first in the decompositon, which makes them particularly simple to process). The final solution may differ trivially from the non-sparse one because of a different iteration path.

- In a small number of examples, the effect on estimates of degrees of freedom and standard error have been slight.

- The output of the program is slightly changed. Under the assumption that the sparse terms coefficients should not be printed by default, they are returned as a component `frail`, and the usual `coef` and `var` components contain only the non-sparse terms.

- The Schoenfeld and dfbeta residuals, both of which are a matrix with one column per variable, are computed with the frailty treated as a fixed offset.

Whether or not sparse computation is selected, the program does one other 'trick' that is worth noting. We want the shrinkage term to be symmetric with

respect to the families. However, the standard S model matrix would be formed per the global `contrast` option, which leads to *nfamily-1* coefficients that are not symmetric, e.g., for treatment style contrasts the reference group is left out. Although this recoding would not change the predicted value of a given subject in an ordinary Cox model, it does change both the coefficients and the prediction in a penalized model, because of the explicit shrinkage. In order to prevent this transformation a special "don't do contrasts" attribute is added to the frailty term. Alternatively, the penalty function could be made more sophisticated, with the type of contrast passed to it as an argument.

Exactly the same issue must be considered to extend the `ridge` function to categorical variables.

The computation of the degrees of freedom and variance matrices are also specialized to avoid any intermediate steps that would give an q by q result, where q is the number of sparse coefficients and p is the number of other variables in the problem. The details of this are shown in appendix A.

The martingale and deviance residuals are unchanged by the use of a sparse computation. To compute the score and Schoenfeld residuals, the code treats the final fitted values of the sparse term as fixed values, the returned matrix of residuals does *not* include columns for the dummy variables that represent the frailty.

The function for predicted values has not yet been updated to accomodate sparse terms. For those values that depend only on the linear predictor, e.g., the estimated per-subject risk score, the current routines will work as expected. The default baseline survival curve corresponds to a fictional subject with 0 for the random effect and means for the other covariates. Results that depend on a new data set are more problematical; should we allow a 'value' to be set for the random effect, or always force it to be zero? What should the variance of the prediction be?

4.3 Examples

4.3.1 Rat data

A data set on the effect of treatment on survival for 150 female rats, where the rats come from 50 litters, has been used by several authors. The data set can be found in Mantel, Bohidar and Ciminera [16]. This example concentrates on the female litters.

The data set has 3 rats per litter, one of which recieved a potentially tumorigenic treatment. Forty rats developed a tumor during follow-up. In order to match prior analyses we need to use the Breslow approximation for tied times.

```
> rfit  <- coxph(Surv(time, status) ~ rx + frailty(litter),
```

```
                                data=rats, method='breslow')
> rfit
                   coef se(coef)   se2 Chisq   DF      p
             rx 0.906 0.323     0.319  7.88  1.0 0.005
frailty(litter)                        16.89 13.8 0.250


Iterations: 6 outer, 19 Newton-Raphson
     Variance of random effect= 0.474   EM likelihood = -181.1
Degrees of freedom for terms=  1.0 13.9
Likelihood ratio test=36.3  on 14.83 df, p=0.001  n= 150



> rfit0 <- coxph(Surv(time, status) ~ rx, rats, method='breslow')
> rfit0
    coef exp(coef) se(coef)    z       p
rx 0.898      2.46     0.317 2.83 0.0047


Likelihood ratio test=7.87  on 1 df, p=0.00503  n= 150

> rfit1 <- coxph(Surv(time, status) ~ rx + frailty(litter, theta=1),
                 data=rats, method='breslow')
> rfit1
                             coef se(coef)   se2 Chisq   DF      p
                       rx 0.918 0.327     0.321  7.85  1.0 0.0051
frailty(litter, theta = 1)                       27.25 22.7 0.2300


Iterations: 1 outer, 5 Newton-Raphson
     Variance of random effect= 1   EM likelihood = -181.5
Degrees of freedom for terms=  1.0 22.7
Likelihood ratio test=50.7  on 23.67 df, p=0.001  n= 150
```

The main thing to notice about the result is how little the treatment coefficient is changed by the inclusion of a random effect term. This is likely a consequence of the balanced model; each litter recieved both the active and control treatments.

We see that for a fixed value of the frailty the iteration is nearly as efficient as for a normal Cox model, which usually requires 3-4 iterations. The generalized fit required 6 guesses to maximize the profile likelihood, and about 3 internal iterations per ν value.

The "likelihood ratio test" is always the difference in partial likelihood between the initial and final fit, ignoring penalty terms and corrections. The default for the initial fit is $(\beta, \omega) = 0$, which is a fit with no covariates or random effect. The

degrees of freedom is computed as described earlier. The random effect has added little.

The `history` component of the returned fit contains the final return values of the control function(s). For an iterative method such as this, this shows the history of the iteration as well as the final value for θ.

```
>rfit$history[[1]]
$theta: 0.4742854

$done: T

$loglik: -181.0777

$history:
          theta     loglik   c.loglik
[1,] 0.0000000 -181.8451 -181.8451
[2,] 1.0000000 -218.3683 -181.5458
[3,] 0.5000000 -273.3117 -181.0788
[4,] 0.3090061 -337.7537 -181.1490
[5,] 0.4645720 -281.3849 -181.0775
[6,] 0.4736212 -279.2127 -181.0773
```

The component `history$history` has columns that give successive values of θ, the (maximal) penalized likelihood for that value of θ, and the corrected likelihood L_g. We see that in this example that the profile likelihood L_g is very flat as a function of θ. The first element of the list, 0.4742854, is the value that would have been used for the *next* iteration.

The solution using a Gaussian frailty is not much different.

```
> rfit2 <- coxph(Surv(time, status) ~ rx +
                        frailty(litter, dist='gauss'), rats)
> rfit2
                         coef se(coef)   se2 Chisq  DF      p
                   rx 0.913 0.323     0.319  8.01  1.0 0.0046
frailty(litter, dist=ga                     15.57 11.9 0.2100

Iterations: 6 outer, 16 Newton-Raphson
Penalized terms:
     Variance of random effect= 0.412
Degrees of freedom for terms=  1.0 11.9
Likelihood ratio test=35.3  on 12.87 df, p=0.000712  n= 150
```

4.4 Survival of kidney catheters

The following data set is presented in McGilchrist and Aisbett [18]. Each observation is the time to infection, at the point of insertion of the catheter, for kidney patients using portable dialysis equipment. Catheters may be removed for reasons other than infection in which case the observation is censored. There are 38 patients, each of which has exactly 2 observations. Variables are the subject id, age, sex (1=male, 2=female), disease type (glomerulo nephritis, acute nephritis, polycystic kidney disease, and other), and the time to infection or censoring for each insertion.

```
> kfit <- coxph(Surv(time, status) ~ age + sex + disease + frailty(id), kidney)
> temp <-  coxph(Surv(time, status) ~ age + sex + disease +
                                   frailty(id, sparse=F), kidney)
>kfit
              coef  se(coef)      se2 Chisq DF          p
       age  0.00318 0.0111    0.0111  0.08 1  7.8e-01
       sex -1.48314 0.3582    0.3582 17.14 1  3.5e-05
 diseaseGN  0.08796 0.4064    0.4064  0.05 1  8.3e-01
 diseaseAN  0.35079 0.3997    0.3997  0.77 1  3.8e-01
diseasePKD -1.43111 0.6311    0.6311  5.14 1  2.3e-02
frailty(id)                          0.00 0  9.5e-01


Iterations: 6 outer, 29 Newton-Raphson
Penalized terms:
     Variance of random effect= 1.47e-07   EM likelihood = -179.1
Degrees of freedom for terms= 1 1 3 0
Likelihood ratio test=17.6  on 5 df, p=0.00342  n= 76


>temp
              coef exp(coef) se(coef)     se2      z        p
       age  0.00318     1.003   0.0111 0.0111  0.285 7.8e-01
       sex -1.48314     0.227   0.3582 0.3582 -4.140 3.5e-05
 diseaseGN  0.08796     1.092   0.4064 0.4064  0.216 8.3e-01
 diseaseAN  0.35079     1.420   0.3997 0.3997  0.878 3.8e-01
diseasePKD -1.43111     0.239   0.6311 0.6311 -2.268 2.3e-02
(38 lines of other coefs)


Iterations: 6 outer, 19 Newton-Raphson
Penalized terms:
     Variance of random effect= 1.47e-07   EM likelihood = -179.1
Degrees of freedom for terms= 1 1 3 0
```
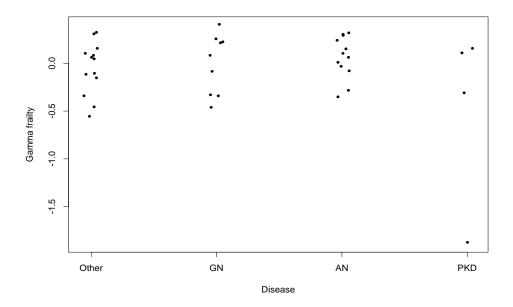
Figure 5: *Realized gamma frailties for the kidney data*

```
Likelihood ratio test=17.6  on 5 df, p=0.00342  n= 76
```

The non-sparse solution has given the same answer as the sparse algorithm in 10 fewer iterations, but it took somewhat over 3 times as long to run. Both programs have decided that the optimal value for a gamma frailty is essentially 0. The "p-value" for the frailty term is almost random, both the test statistic and the degrees of freedom are $< 10^{-5}$, and round off error in both the computation and the chi-square probability function begin to dominate.

A somewhat different result ensues when the disease variable is left out of the model, however.

```
> kfit2 <- coxph(Surv(time, status) ~ age + sex + frailty(id),
        data=kidney)
> kfit2
            coef se(coef)     se2 Chisq   DF       p
     age  0.00522 0.0119   0.0088  0.19  1.0 0.66000
     sex -1.58335 0.4594   0.3515 11.88  1.0 0.00057
frailty(id)                       22.97 12.9 0.04100

Iterations: 7 outer, 49 Newton-Raphson
     Variance of random effect= 0.408    EM likelihood = -181.6
Degrees of freedom for terms=  0.6  0.6 12.9
```

```
     Likelihood ratio test=46.6  on 14.06 df, p=2.36e-05  n= 76
```

A plot of the realized frailty coefficients shows that the gamma frailty has "picked up" a large outlier; the same subject is an influential point for the fourth disease group. In a model with only age and gender, this subject has a martingale residual of -7.4; the next smallest is -1.8! This subject, number 21, is a 46 year old male (median age=45.5). There are 10 males in the study and most had early failures, 16/18 of the remaining kidneys had an infection at a median of 19 (days?), 2 were censored at 4 and 8 days respectively. Subject 21 however had failures at 154 and 562 days, making him quite an extreme outlier.

Using the approximate Gaussian frailty method of McGilchrist [17] with REML gives a non-zero estimate of the random effect.

```
> mfit1 <- coxph(Surv(time, status) ~ age + sex + disease +
                 frailty(id, dist='gauss'), data=kidney)
mfit1
                   coef se(coef)    se2 Chisq   DF        p
        age     0.00489 0.015    0.0106  0.11  1.0 0.74000
        sex    -1.69727 0.461    0.3617 13.56  1.0 0.00023
  diseaseGN     0.17985 0.545    0.3927  0.11  1.0 0.74000
  diseaseAN     0.39294 0.545    0.3982  0.52  1.0 0.47000
 diseasePKD    -1.13633 0.825    0.6173  1.90  1.0 0.17000
 frailgy(id)                           17.89 12.1 0.12000


Iterations: 8 outer, 32 Newton-Raphson
Penalized terms:
     Variance of random effect= 0.493
Degrees of freedom for terms=  0.5  0.6  1.7 12.1
Likelihood ratio test=47.5  on 14.9 df, p=2.83e-05  n= 76


> mfit2 <- coxph(Surv(time, status) ~ age + sex + disease +
                 frailty(id, dist='gaus', sparse=F), kidney)
> mfit2
                        coef se(coef)    se2 Chisq   DF        p
                age  0.00492 0.0149   0.0108  0.11  1.0 0.74000
                sex -1.70204 0.4631   0.3613 13.51  1.0 0.00024
          diseaseGN  0.18173 0.5413   0.4017  0.11  1.0 0.74000
          diseaseAN  0.39442 0.5428   0.4052  0.53  1.0 0.47000
         diseasePKD -1.13160 0.8175   0.6298  1.92  1.0 0.17000
frailty(id, dist='gaus                      18.13 12.3 0.12000


Iterations: 6 outer, 17 Newton-Raphson
```
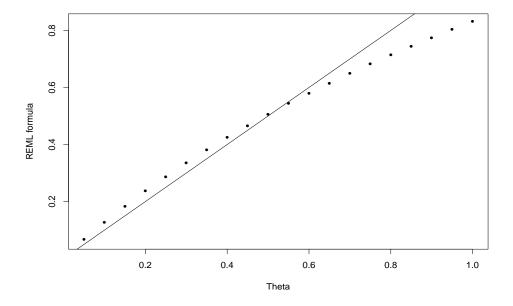
Figure 6: *REML solution for the kidney data*

```
   Penalized terms:
            Variance of random effect= 0.509
   Degrees of freedom for terms=   0.5   0.6   1.7  12.3
   Likelihood ratio test=118   on 15.14 df, p=0   n= 76
```

In this case the sparse routines have some impact on the solution itself. The REML estimate of θ satisfies the following equation

$$\hat{\theta} = \frac{\sum \omega_i + \text{sum}(\text{diag}(\text{var}[6:43]))}{38} \, . \tag{7}$$

Moderate changes in the estimated H matrix can have a large effect on the final value of $\hat{\theta}$. The sparse method uses a diagonal approximation to the 'frailty' portion of H, and this effects the diagonal of H^{-1}=var. The overall Wald test of 18.3 is still not significant. The Gaussian frailty model without the disease variable also has a single large negative frailty.

We should point out as well that these answers differ slightly from the author's [17] results. Their paper presents formulas that are completely valid only for untied data, and this data set has a 5 tied pairs and one quadruple. This is certainly not clinically significant, and in a standard Cox model would barely perturb the answers. Unfortunately, the REML solution for σ is very touchy. Figure 6 shows the left and right-hand sides of equation 7 for a range of values along with the line

$y = x$. (Remember that H is implicitly a function of the current θ). The solution point lies at the intersection of these curves.

4.5   CGD Data

Chronic granulotomous disease (CGD) is a heterogeneous group of uncommon inherited disorders characterized by recurrent pyogenic infections that usually begin early in life and may lead to death in childhood. Interferon gamma is a principal macrophage-activating factor shown to partially correct the metabolic defect in phagocytes, and for this reason it was hypothesised that it would reduce the frequency of serious infections in patients with CGD. In 1986, Genentech, Inc. conducted a randomized, double-blind, placebo-controlled trial in 128 CGD patients who received Genentech's humanized interferon gamma (rIFN-g) or placebo three times daily for a year. The resultant data set can be found in appendix D of Fleming and Harrington [8]. The primary endpoint of the study was the time to the first serious infection. However, data was collected on all serious infections until loss-to-followup, which occurred before day 400 for most patients. Thirty of the 65 patients in the placebo group and 14 of the 63 patients in the rIFN-g group had at least one serious infection. The total number of infections was 56 and 20 in the placebo and treatment groups, respectively.

```
> coxph(Surv(tstart, tstop, status) ~ rx + cluster(id), cgd)
    coef exp(coef) se(coef) robust se     z       p
rx -1.1     0.334    0.261     0.312 -3.51 0.00045


> coxph(Surv(tstart, tstop, status) ~ rx + cluster(id) + strata(enum), cgd)
     coef exp(coef) se(coef) robust se     z       p
rx -0.86     0.423     0.28     0.292 -2.95 0.0032


> coxph(Surv(tstart, tstop, status) ~ rx +frailty(id) + strata(enum), cgd)
             coef se(coef)  se2 Chisq   DF      p
        rx -0.863 0.281    0.28 9.45 1.00 0.0021
frailty(id)                    0.73 0.66 0.2600


Iterations: 5 outer, 17 Newton-Raphson
Penalized terms:
     Variance of random effect= 0.01  EM likelihood = -247.1
Degrees of freedom for terms= 1.0 0.7
Likelihood ratio test=10.2  on 1.66 df, p=0.00394  n= 203


> coxph(Surv(tstart, tstop, status) ~ rx +mfrail(id) +strata(enum), cgd)
```

```
           coef se(coef)   se2 Chisq   DF       p
      rx -0.983 0.312    0.281  9.9  1.0 0.0017
mfrail(id)                    34.5  23.7 0.0700


Iterations: 8 outer, 36 Newton-Raphson
Penalized terms:
     Variance of random effect= 0.483
Degrees of freedom for terms=  0.8 23.7
Likelihood ratio test=62.8  on 24.49 df, p=3.34e-05  n= 203
```

The example above shows partial results of Andersen-Gill, conditional, gamma frailty and normal frailty models. The conditional model is known to be biased downwards, and one question is whether the addition of a frailty term can compensate.

For the gamma frailty model, the final value of θ was essentially zero − the fit is maximized at the no frailty model. For the REML normal model (sparse computation), the value is maximized at a value of $\theta = .483$. With this value of θ, the coefficient is significantly increased.

A gamma model with the same degrees of freedom agrees with the above in all important respects; it has almost an identical treatment coefficient and random effect variance.

```
> coxph(Surv(tstart, tstop, status) ~ rx + frailty(id, df=23.7)
                   + strata(enum),  cgd)


                      coef se(coef)   se2 Chisq   DF       p
              rx -0.981 0.309    0.281 10.1   1.0 0.0015
frailty(id, df = 23.7)              31.2  23.7 0.1400


Iterations: 3 outer, 11 Newton-Raphson
Penalized terms:
     Variance of random effect= 0.471   EM likelihood = -247.5
Degrees of freedom for terms=  0.8 23.7
Likelihood ratio test=55.1  on 24.53 df, p=0.000392  n= 203
```

4.6 Colon Cancer study

This data is from a study by Moertel, et. al. [20] of three regimens, placebo, Levamisole, and Levamisole + 5-FU, in the treatment of resected colon cancer. The data is used in Lin's paper on marginal Cox models [15] (he uses only 2 of the arms). For each patient we have both the time to survival and the time to progression, and

would like to use both concurrently in an assessment of treatment. There are 929
patients distributed as follows

	N umber of events		
	0	1	2
Placebo	125	35	155
Levamisole	128	31	151
Lev+5FU	170	26	108

As we can see, most patients have either both outcomes or neither.

The code below fits the marginal model (recommended by Lin), along with the
gamma and Gaussian frailty models.

```
> fitc1 <- coxph(Surv(time, status) ~ rx + extent + node4 + cluster(id)
                 + strata(etype), colon)
> fitc1
              coef exp(coef) se(coef) robust se      z        p
     rxLev -0.0362     0.964   0.0768    0.1056 -0.343 7.3e-01
rxLev+5FU -0.4488     0.638   0.0840    0.1168 -3.842 1.2e-04
    extent  0.5155     1.674   0.0796    0.1097  4.701 2.6e-06
     node4  0.8799     2.411   0.0681    0.0961  9.160 0.0e+00


Likelihood ratio test=248  on 4 df, p=0  n= 1858



> fitc2 <- coxph(Surv(time, status) ~ rx + extent + node4 + mfrail(id)
                 + strata(etype), colon)
> fitc2
               coef se(coef)     se2  Chisq DF        p
     rxLev -0.0267 0.241   0.0824    0.01   1 9.1e-01
 rxLev+5FU -0.7880 0.243   0.1071   10.50   1 1.2e-03
    extent  1.1305 0.218   0.1068   26.81   1 2.2e-07
     node4  2.1266 0.210   0.0984  102.56   1 0.0e+00
mfrail(id)                         5464.64 730 0.0e+00

Iterations: 10 outer, 77 Newton-Raphson
Penalized terms:
     Variance of random effect= 7.05
Degrees of freedom for terms=   0.3   0.2   0.2 729.7
Likelihood ratio test=3544  on 730.49 df, p=0  n= 1858


> fitc3 <- coxph(Surv(time, status) ~ rx + extent + node4 + frailty(id)
```

```
                    + strata(etype), colon)
> fitc3
              coef se(coef)   se2   Chisq DF        p
      rxLev   0.0438 0.305    0.140    0.02  1 8.9e-01
 rxLev+5FU  -0.5111 0.310    0.171    2.71  1 1.0e-01
    extent   1.3382 0.251    0.137   28.38  1 1.0e-07
     node4   2.3397 0.233    0.156  100.63  1 0.0e+00
frailty(id)                        5956.44 867 0.0e+00


Iterations: 8 outer, 103 Newton-Raphson
Penalized terms:
     Variance of random effect= 8.06   EM likelihood = -5347.4
Degrees of freedom for terms=   0.5    0.3   0.4 867.0
Likelihood ratio test=3789  on 868.25 df, p=0  n= 1858


> fitc4 <-  coxph(Surv(time, status) ~ rx + extent + node4 +
                      frailty(id, df=30) + strata(etype), colon)
> fitc4
                      coef se(coef)    se2   Chisq DF        p
            rxLev  -0.0374 0.0789   0.0769    0.22  1 6.4e-01
        rxLev+5FU  -0.4565 0.0859   0.0840   28.27  1 1.1e-07
           extent   0.5289 0.0815   0.0798   42.13  1 8.5e-11
            node4   0.9078 0.0701   0.0681  167.85  1 0.0e+00
frailty(id, df = 30)                          58.53 30 1.4e-03


Iterations: 3 outer, 9 Newton-Raphson
 Penalized terms:
         Variance of random effect= 0.0337   EM likelihood = -5832.4
Degrees of freedom for terms=   1.9  1.0  0.9 30.0
Likelihood ratio test=276  on 33.8 df, p=0  n= 1858
```

This is the first data set where the final frailty solution is large: the variances are 8.1 for the gamma and 6.9 for the REML gaussian models. The fitting is also quite slow, as compared to a standard Cox model. The number of Newton-Raphson iterations necessary for a particular value of θ increases markedly when θ is larger than about 3, for this data set; the penalized information matrix is not so clearly diagonally dominant in this case, and the sparse solution not as efficient an approximation.

The unconstrained frailty fits are worrysome. For subjects with no events, the realized values of ω for the gamma frailty model range from -3 to -10 (median of -6), versus a median value of .-5 for those with 2 events. With such a weight, exp(-5.5)=
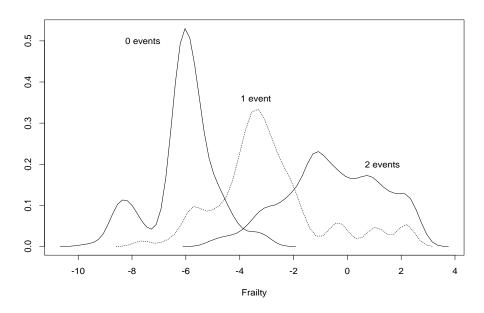
Figure 7: *Density of the realized frailties, colon cancer data*

.004, the model we obtain is almost equivalent to removing all of the subjects with no events from the data set. That is, the frailty model declares them to be "risk free".

Figure 7 shows the densities of the realized frailties from the gamma frailty model. There is almost no overlap in the densities of subjects with both death and progression, and those with neither.

4.7 Further notes

In the code as actually distributed, the penalty functions are incorporated into the wrapper, rather than being separate as shown in some of the examples. This avoids cluttering up the S-Plus space with too many function names.

At this point the underlying C programs and the control structures of the `coxph` program are very solid, and I don't anticipate any changes. The functions to describe the penalties are quite simple, so they also shouldn't change much. The biggest place for algorithmic improvement appears to be in the control functions. The heart of these are simple one-dimensional minimizers, either Brent's method (`frailty.brent`) or exponential interpolation (`frailty.controldf`). The starting estimates, bracketing rules, and step sizes in each of these has not been examined in any systematic way, although they do work for all of the current examples.

The important auxillary functions predict.coxph and survfit.coxph have not been modified as yet; they will work as is for non-sparse models. However, both the default "predicted survival curve" and the type=terms result from predict cannot be of much use for a random effects model. Plans are to treat the sparse terms as a fixed offset.

In conjunction with Silke Schmidt (her thesis) we have tried extending this code to a more complicated frailty problem involving family data. This is a nested problem with both a per-subject effect and a familial effect that has a genetic basis. The result is that the penalty matrix for the subject random effects has a block diagonal structure. The sparse Newton-Raphson didn't work. (Blast! And I had such high hopes.) Ignoring the off-diagonal information causes the algorithm to take poor steps, sometimes in the opposite direction to the actual solution. A more sophisticated change does appear to work, and may become part of the distributed routines at a later date.

And last, this document is still a working draft. The authors would appreciate any feedback on its structure or deficiencies.

A   Computing degrees of freedom

Let H denote the negative Hessian (minus the matrix of second partial derivatives) of the log penalized partial Cox likelihood. A Cholesky decomposition gives $H = U'DU$, where U is upper triangular with $U_{ii} = 1$ for all i and D is diagonal. The underlying code places the sparse terms first in the model, so that the Cholesky decomposition actually has the form

$$U = \begin{pmatrix} I & U_1 \\ 0 & U_2 \end{pmatrix}$$

with I being a q by q identity matrix and 0 a matrix of zeros. Then, we have

$$
\begin{aligned}
H^{-1} &= U^{-1}D^{-1}U'^{-1} \\
&\equiv TD^{-1}T'
\end{aligned}
$$

where $T = U^{-1}$ is upper triangular with $1's$ on the diagonal, like U, and can be partitioned analogously into I, 0, T_1 and T_2.

Note that $T_2 = U_2^{-1}$. So:

$$H^{-1} = \begin{pmatrix} I & T_1 \\ 0 & T_2 \end{pmatrix} \begin{pmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ T_1' & T_2' \end{pmatrix}$$

$$= \begin{pmatrix} D_1^{-1} + T_1 D_2^{-1} T_1' & T_1 D_2^{-1} T_2' \\ T_2 D_2^{-1} T_1' & T_2 D_2^{-1} T_2' \end{pmatrix}$$

$$\equiv \begin{pmatrix} A & B \\ B' & C \end{pmatrix}$$

A is a q by q matrix, and our goal is to compute further information using only the diagonal of A. Let $V = H^{-1}(H - P)H^{-1}$, where P is the matrix of second partial derivatives of the penalty terms for the log of the Cox penalized log likelihood, so $H = \mathcal{I} + P$. The lower right p by p submatrix of V is the second estimate of variance of the nonfrailty terms (`var2` on the printout), the lower right p by p submatrix of H^{-1} being the first estimate of variance (`var`) If P is partitioned:

$$P = \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix}$$

where P_1 is a q by q submatrix with the penalty terms for the frailty part of the model and P_2 is a p by p submatrix with penalty terms for the rest of the model, including any ridge or penalized splines, then we note that P_1 is diagonal as a consequence of the choice of sparse computation, and that P_2 will be zero if there are no other penalized terms in the model, which is a common situation when doing frailty models.

The `coxpenal.df` program, which does the essential computations, is called with arguments `hmat`, the $q + p$ by p matrix containing the righmost columns of U, `hinv` which contains the p rightmost columns of T, `fdiag` containing the diagonal of D^{-1}, along with the penalty matrices and some bookkeeping information. It returns the lower right corners of V and H^{-1}, the vector of degrees of freedom, and the diagonal of the upper left corner of H^{-1}.

The lower right corner of H^{-1} is simply $T_2 D_2^{-1} T_2'$, a p by p matrix. Some further efficiency could be gained, in terms of the total number of multiplications and additions, by noting that T_2 is upper triangular but this has not been persued. The lower right of V is

$$V_{22} = H_{22}^{-1} - (B'P_1 B + CP_2 C).$$

The degrees of freedom for the sparse term is

$$\begin{aligned} \text{trace}\left[(H_{11}^{-1})^{-1}V_{11}\right] &= \text{trace}[A^{-1}(A - AP_1 A - BP_2 B')] \\ &= q - \text{trace}[P_1 A] - \text{trace}[A^{-1}BP_2 B'] \\ &= q - \text{trace}[P_1 A] - \text{trace}[B'A^{-1}BP_2]. \end{aligned}$$

34

If P_2 is zero then since P_1 is diagonal, only the diagonal of A is needed. This simplification occurs for any penalized term, sparse or not, when it is the only penalized term in the model and P is diagonal.

If P_2 is non-zero, then note that the standard formula for partitioned matrices gives that the lower right corner of H must be

$$H_{22} = [C - B'A^{-1}B]^{-1} .$$

We can compute the p by p matrix H_{22} directly from T and then subtract from C to get the matrix $B'A^{-1}B$, without having to form or invert the q by q matrix A.

For individual terms that are part of the non-sparse portion of the matrix, we have to use the usual formula since this may involve sub-portions of the matrix. However, since the two quantities of interest are already computed this is not a large issue. (Again, we assume that p is relatively small).

The S-Plus code also makes frequent use of the identity `trace(x %*% y) = sum(t(x) * y)`, where `%*%` is matrix multipltication, `t()` is the transpose function, and `*` is elementwise multiplication.

B   EM Algorithm

B.1   Derivation

Let x be the full data with density $f(x;\theta)$, y the observed, or partial, data with density $g(y;\theta)$, and let

$$k(x|y;\theta) = \frac{f(x;\theta)}{g(y;\theta)}$$

be the conditional density of f given g. For simplicity abbreviate $\log[g(y;\theta)]$ as $L_g(y;\theta)$, and similarly for f and k.

The log-likelihood can be broken up according to the simple identity

$$
\begin{aligned}
\log[g(y;\theta)] &= \log[f(x;\theta)] - \{\log[f(x;\theta) - \log[g(y;\theta)]\} \\
L_g(\theta) &= L_f(\theta) - L_k(\theta)
\end{aligned}
\tag{8}
$$

Now, let ϕ be a provisional guess for the value of θ, and take the conditional expectation of equation (8) with respect to y and ϕ:

$$
\begin{aligned}
\mathrm{E}\{L_g(y;\theta)|y,\phi\} &= \mathrm{E}\{L_f(x;\theta)|y,\phi\} - \mathrm{E}\{L_k(x|y;\theta)|y,\phi\} \\
L_g(y;\theta) &\equiv Q(\theta,\phi) - H(\theta,\phi) .
\end{aligned}
\tag{9}
$$

The term on the left hand side of the equation is unchanged, since the conditional expectation of y with respect to any parameter(s) *and* y must be y itself. The right

hand side defines the terms Q and H in the original paper by Dempster, Laird and Rubin [5]. This equation is the key. In particular, the terms on the right are often computable, where the original term on the left is not. L_f, the full data likelihood may be easy to write down and maximize. H is not necessary to the maximization process, but is needed for the evaluation of the likelihood itself (if desired).

The difference between θ and ϕ is crucial. The EM algorithm consists of two alternating update steps.

E step: Compute the algebraic form of Q. This will involve replacement of x and functions of x in L_f with their expected values given y and ϕ.

M step: Maximize this new expression, now only a function of θ, with respect to θ. Let $\hat{\theta}$ be that maximizing value for $Q(\theta, \phi)$. Return to the E step, replacing ϕ with $\hat{\theta}$.

The convergence of the EM rests on the following lemma: For any value of θ, $H(\theta, \phi) \leq H(\phi, \phi)$, with equality occuring only if $L_f(\theta) = L_f(\phi)$ a.e. The left hand side is the expected value of a log-likelihood at parameter θ when the true density has parameter ϕ, and the lemma states that the expected value of a log-likelihood is maximized at the true parameter. I will follow the lead of other authors and state that the proof is a "standard result due to Jensen's inequality".

From this we can easily show that

$$L_g(\hat{\theta}) - L_g(\phi) = [Q(\hat{\theta}, \phi) - Q(\phi, \phi)] + [H(\phi, \phi) - H(\hat{\theta}, \phi)] > 0 \,.$$

The first term on the right hand side is greater than zero by definition of $\hat{\theta}$, and the second from the lemma. The algorithm may converge arbitrarily slowly, however. For any given component of θ, the error will eventually decrease by a multiplicative constant.

Suppose that L_f has the expansion

$$L_f(x; \theta) = a(x) + b(x, \theta) + c(\theta)$$

for some functions a, b and c. Technically, the E step of the algorithm involves calculation of $\mathrm{E}\{a(x)|y, \phi\}$. However, since this term will not be used in the M step it can be ignored during iteration. For an exponential family, e.g., Gaussian with known variance,

$$b(x, \theta) = \theta' t(x)$$

where $t(x)$ is the vector of sufficient statistics. This leads to the "classic" EM, which simply replaces sufficient statistics by their expectation.

B.2 Example

Consider a censored data example found in Cox and Oakes [4]. Let x_i be the true survival time and (y_i, δ_i) be the observed time along with a censoring indicator.

$$
\begin{aligned}
x_i &= y_i, & \text{if } \delta_i = 1 \\
x_i &> y_i, & \text{if } \delta_i = 0
\end{aligned}
$$

a. Assume that X is exponential. For this problem we can easily write down the true likelihood

$$
L_g = \sum_{i=1}^{n} [\delta_i \log(\theta) - \theta y_i].
$$

b. Assume that the actual likelihood L_g was not known, and that we want instead to compute our solution using the EM method. The full data likelihood is easy to state, since it does not involve censoring.

$$
L_f = \sum [\log(\theta) - \theta x_i].
$$

We also need to compute the form of Q. In this case, it is simply a replacement of each x_i with $\mathrm{E}(x_i|y_i, \phi)$. This is equal to y_i for the uncensored observations. For the censored observations, the memoryless property of the exponential distribution means that the time remaining is also exponential, with expected value of $1/\phi$. Thus

$$
\begin{aligned}
Q(\theta, \phi) &= \sum [\log(\theta) + \theta \mathrm{E}(x_i|y_i, \delta_i, \phi)] \\
&= \sum [\log(\theta) + \theta(y_i + (1 - \delta_i)/\phi)]
\end{aligned}
$$

The E step is thus a replacement of each censored y by its expectation given the current value of $\hat{\theta}$. The M step consists of maximizing L_f, using these replaced data values.

c. Using the same conditional distribution argument, we see that

$$
L_k = \begin{cases} 0 & \delta = 1 \\ \log(\theta) - \theta(x - y) & \delta = 0 \end{cases}
$$

so that

$$
\begin{aligned}
H(\theta, \phi) &= \mathrm{E}(L_k|y, \delta, \phi) \\
&= \sum (1 - \delta_i)[\log(\theta) - \theta(y_i + 1/\phi - y_i)]
\end{aligned}
$$

The value of H is not needed for the iteration. Note, however, that the final loglikelihood at $\hat{\theta}$ can be computed as $Q(\hat{\theta}, \hat{\theta}) - H(\hat{\theta}, \hat{\theta})$. This may be important in other problems, where L_g is difficult to write down.

B.3 Gamma frailties and PH models

B.3.1 Preliminaries

We will make use of two densities in the derivation. Since they are used mostly in log-likelihoods, I give the log of the density. The gamma density has formula

$$\log[f(x; \eta, \nu)] = (\nu - 1)\log(x) - \eta x + \nu \log(\eta) - - \log \Gamma(\nu)$$

(This f has nothing to do with L_f − here it's just a dummy symbol for the distribution). Two quantities that will be needed are $E(X) = \nu/\eta$ and $E(\log(X)) = \psi(\nu) - \log(\eta)$, where ψ is the digamma function.

Let $\varpi = \exp(\omega)$ follow a gamma distribution with parameters ν and η, so that the distribution W of ω is:

$$\log(W(w; \eta, \nu) = (\nu w - \eta e^w) + \nu \log(\eta) - \log \Gamma(\nu). \tag{10}$$

The mean and variance of w are $\psi(\nu) - \log(\eta)$ and $\psi'(\nu)$, where ψ and ψ' are the digamma and trigamma functions, respectively. (The distribution of $w + \log(\eta)$ follows the log-gamma distribution of Kalbfleisch and Prentice [13] equation (2.3); see there for derivation of the result).

B.3.2 General setup

Assume a proportional hazards model with random effects or *frailties*, with hazard function

$$\lambda_i(t) = \lambda_0(t)e^{X_i\beta + Z_i\omega}. \tag{11}$$

Here β is a vector of p fixed effects and ω a vector of q random effects, where the individual elements ω_j are iid realizations from some distribution $W(\theta)$. The matrix X will normally contain measured covariate values, and Z will be a design matrix that describes how the random effects apply to individual subjects. Both X and Z might contain time dependent effects, but we will ignore this complication for the moment. The baseline hazard may contain other parameters ξ; these will also be ignored. Note that if X contains an intercept term (which is implicit for the proportional hazards model), we can constrain ω to have mean 0.

We can treat the random effects as unobserved data and apply the EM algorithm. The 'x' of the formal EM argument is the entire observed data (time, status, covariates) *plus* the frailties, and 'y' is the data without the frailties. The full log-likelihood, if we had observed ω, is

$$L_f \quad = \quad \sum_{j=1}^{q} \log(W(\omega; \theta))$$

$$+\sum_{i=1}^{n}\delta_i[\log(\lambda_0(t_i)) + X_i\beta + Z_i\omega] - \Lambda_0(t_i)e^{X_i\beta + Z_i\omega} \tag{12}$$

Here $\delta_i = 0$ for censored observations, and 1 for events. X will be an n by p matrix and Z will be n by q.

This way of setting up the problem is similar in notation to random effects models in linear regression. Another notation, which is more common in the survival literature, is to define $\varpi_i = \exp(Z_i\omega)$ as the frailty parameter for each subject. Then

$$\lambda_i(t) = \varpi_i\lambda_0(t)e^{X_i\beta},$$

and the imposed constraint is usually $E(\varpi) = 1$ rather than $E(\omega) = 0$. We will prefer the linear models like formulation.

B.3.3 Simpler case

The above equation (12) is not particularly useful unless we can determine the conditional distribution of ω given the rest of the information. This is possible in one particular case. We will largely follow the derivation of Neilson et. al. [22].

Let $\varpi = \exp(\omega)$ follow a gamma distribution with parameters ν and η. Without loss of generality, we will assume that $\nu = \eta$, i.e., that the distribution ϖ has mean 1 and variance $1/\nu$. (This is, in this case, algebraicly simpler than imposing the constraint that $E(\omega) = 0$.)

Furthermore, assume that the random effect consists of independent clusters of observations, i.e., $Z_{ij} = 1$ iff subject i belongs to cluster j, with no subject in two clusters. Then if we define

$$
\begin{aligned}
D_j &= \sum_{i=1}^{n} Z_{ij}\delta_i \\
E_j^* &= \sum_{i=1}^{n} Z_{ij}\Lambda_0(t_i)e^{X_i\beta}
\end{aligned}
\tag{13}
$$

we see that the likelihood (12) can be re-arranged so that the individual ω_j terms separate:

$$
\begin{aligned}
L_f &= \sum_{j=1}^{q}\left[(\nu\omega_j - \nu e^{\omega_j}) + \nu\log(\nu) - \log\Gamma(\nu)\right] \\
&\quad + \sum_{j=1}^{q}\left[D_j\omega_j - E_j^* e^{\omega_j}\right] \\
&\quad + \sum_{i=1}^{n}\delta_i[\log(\lambda_0(t_i)) + X_i\beta].
\end{aligned}
\tag{14}
$$

(Because each subject is in exactly one cluster, $X_i \omega = \omega_{j_i}$, with j_i being the cluster to which subject i belongs). D_j is the number of events in the jth cluster, and $E_j = E_j^* \exp(\omega_j)$ is the expected number of events in the cluster based on the covariates and the model. E^* is, roughly, the expected number of events for the cluster if their frailty were set equal to 1, but with all the frailties used to compute the baseline hazard $\hat{\Lambda}$.

As a function of any single ω_j, the likelihood (14) is proportional to

$$(\nu + D_j)\omega_j - (\nu + E_j^*)e^{\omega_j} .$$

Comparing this to the gamma distribution defined earlier, we see by inspection that conditional on the data the ϖ_j are distributed as gamma variates with shape and scale parameters of $\nu + D_j$ and $\nu + E_j^*$, respectively.

Now, given a provisional guess $\phi = (\tilde{\beta}, \tilde{\nu})$ and the observed data $y = (t, \delta)$, the E step of the algorithm will involve replacing ω_j and $\exp(\omega_j)$ terms in L_f by

$$
\begin{aligned}
\mathrm{E}(\omega_j|\phi,y) &= \psi(D_j + \tilde{\nu}) - \log(E_j^* + \tilde{\nu}) \equiv w_j^* \\
&\text{and} \hspace{6cm} (15) \\
\mathrm{E}(e^{\omega_j}|\phi,y) &= \frac{D_j + \tilde{\nu}}{E_j^* + \tilde{\nu}} \equiv e^{\widehat{w}_j} \hspace{3cm} (16)
\end{aligned}
$$

Since the prior variance of the random effect is $1/\nu$, we see that as the variance goes to infinity the estimated random effect $\varpi = \exp(\omega)$ converges to D/E^*, the unconstrained estimate of excess risk in the cluster *ignoring the frailty*. As the variance goes to zero, all of the frailties ω become 0. The EM algorithm is then:

initialize: An obvious starting guess is $\tilde{\nu} = \infty$, $\tilde{\beta} = \hat{\beta}$, the MLE from a no-frailty model. This is not particularly useful, however, since the frailties then remain fixed at zero in the update formula. We have found $(1, \hat{\beta})$ to be a reasonable starting estimate.

E step: From the provisional value of $\tilde{\beta}$ compute E_j for each group, and from this the function $Q(\theta, \phi) = \mathrm{E}(L_f|\tilde{\nu}, \tilde{\beta})$. If we then rearrange the terms (again) the function is

$$
\begin{aligned}
Q &= \sum_{j=1}^{q} [\nu \log(\nu) - \log \Gamma(\nu) \\
&\qquad\qquad + D_j(w_j^* - \widehat{w}_j) + \nu(w_j^* - e^{\widehat{w}_j}) \\
&+ \sum_{i=1}^{n} \delta_i[\log(\lambda_0(t_i)) + X_i\beta + Z_i\widehat{w}] - \Lambda_0(t_i)e^{X_i\beta + Z_i\widehat{w}} \qquad (17)
\end{aligned}
$$

(Note again that $\sum_j D_j \widehat{w}^j = \sum_i \delta_i Z_i \widehat{w}$).

M step: Choose new values of ν and β to maximize Q. Notice that the last line of the above expression, the only one to involve β, is precisely the log-likelihood for a full data model that has $Z_i w$ as an offset term. In fact, this is the very reason to rearrange the equation in this way. It shows that the next iterate of $\hat{\beta}$ can be obtained using a standard Cox model program. The partial likelihood returned by the Cox program will be the value of the last line in (17) above. Maximization with respect to ν can be done using any one of a number of methods since it is a single parameter.

In fact, the solution can be obtained using only the quantities returned by an ordinary Cox model program. Start with the case of a fixed variance $\theta = 1/\nu$. The quantities D_j and $M_j = D - E$ can be obtained by summing over the input data and the returned martingale residuals, respectively. E^* is obtained from E and the *current* estimates of $\hat{\omega}$, equation 16 is used to get the next estimate of $\hat{\omega}$, and finally, $Z\hat{\omega}$ is used as an `offset` in the next invocation of the Cox model. For fixed θ, this process will often converge in 4-6 iterations.

At the end of the iteration, we will want to compute the value of the actual log-likelihood $L_g = Q - H$ for the final parameter values. Let $a(\nu, \eta)$ be the log density shown in equation (10). We saw that the conditional distribution of the unknown parameters ω was $a(\nu + D_j, \eta + E_j)$. H is the expected value of this expression itself, with respect to the density $a(\nu + D_j, \eta + E_j)$ (circular sounding isn't it). Then

$$H = \sum_{j=1}^{q} \left[(\nu + D_j)w_j^* - (\eta + E_j^*)e^{\widehat{w_j}} \right] + (\nu + D_j)\log(\eta + E_j^*) - \log\Gamma(\nu + D_j)$$

so that

$$
\begin{aligned}
L_g = Q - H \quad &= \quad \text{Cox PL} + \sum_j \nu \log\left(\frac{\nu}{\nu + E_j^*}\right) + \log\left(\frac{\Gamma(\nu + D_j)}{\Gamma(\nu)}\right) \\
&\quad - D_j \log(\nu + D_j).
\end{aligned}
\tag{18}
$$

By "Cox PL" in the above we mean the numerical value returned as the partial likelihood by a standard Cox model program for the given values of β and ω, ω having been entered as an offset term. Again, we see that this can be computed in a standard package such as SAS, after the Cox model has been fit. For the no-covariate case $\beta = 0$ this formula gives, after cancellation, equation (10) of Nielsen et. al. [22]. (Note that for a Cox model the last term of equation 17, $\sum \Lambda_0(t_i)r_i$, is identically equal to the number of deaths, by definition of $\hat{\Lambda}$).

A heuristic approximation helps us get a feel for the probable size of θ. At its solution, the EM algorithm must satisfy $\tilde{\nu} = \hat{\nu}$, i.e., the next step of the algorithm

is equal to the current estimate. Then

$$
\begin{aligned}
e^{\hat{\omega}_j} &= \frac{D_j + \nu}{E_j e^{-\hat{\omega}_j} + \nu} \\
\nu e^{\hat{\omega}_j} + E_j &= \nu + D_j \\
e^{\hat{\omega}_j} &= 1 + \theta M_j
\end{aligned}
\tag{19}
$$

where $M_j = D_j - E_j$ is the sum of the martingale residuals for all subjects in group j. Since $e^x \approx 1 + x$, we see that the final values of $\hat{\omega}$ are approximately θ times the martingale residuals. Now, note that an approximate estimate of "functional form" for the group of subjects with $Z = j$ would be M_j, under an unpenalized model with Z as a factor variable [25]. This implies that the solution with $\theta = 1$ may not be too far from that with $\theta = \infty$.

B.4   Other frailties

Let $L(t; \theta)$ be the Laplace transform of the frailty distribution, and $L^{(n)}$ be the nth derivative of L. Let Z be restricted to the same form as in the last section, i.e., a 'one-way anova' layout where each subject is a member of exactly one frailty group. Then Parner [23] has shown that the EM step (for fixed θ) is

$$
e^{\hat{\omega}_j} = -\frac{L^{(D_j+1)}(E_j^*; \theta)}{L^{(D_j)}(E_j^*; \theta)} .
$$

For the gamma frailty this works out particularly well, since $L(t; \theta) = (1 - \theta t)^{-1/\theta}$, leading to formula (16).

Other distributions do not simplify as neatly. The positive stable distribution, which has often been suggested as a candidate for the frailty, has $L(t) = \exp(-t^\theta)$, leading to

$$
\begin{aligned}
L^{(1)} &= e^{-t^\theta}(-\theta t^{\theta-1}) \\
L^{(2)} &= e^{-t^\theta}\left[(-\theta t^{\theta-1})^2 - \theta(\theta-1)/t^{\theta-2}\right] \\
L^{(3)} &= e^{-t^\theta}\left[(-\theta t^{\theta-1})^3 - \theta(\theta-1)(\theta-2)/t^{\theta-3} + 2\theta^2(\theta-1)t^{2\theta-3}\right] \\
&\vdots
\end{aligned}
$$

Parner derives a recursion formula for computing these derivatives, but the ratios do not have a simple algebraic form. Does any distribution except the gamma simplify?

Far more serious than any difficulty in extending to other distributions, however, is the fact that this approach does not easily generalize to other types of *problems*.

The entire derivation depends on the segregation of ω into a separate term as in equation 14, which in turn is only possible for these simple "one-way anova" layouts. Crossed effects, nested effects, random coefficients for continuous variables, and other interesting models are much more difficult.

C   Writing S-Plus functions

This section describes the details of adding an arbitrary penalty function to the S-Plus code. It can be skipped by most readers.

C.1   Ridge regression

The original control and penalty functions for the ridge regression example were:

```
pfun.ridge <- function(coef,theta, nevent) {
                     list(penalty= sum(coef^2)*theta/2,
                          first  = theta*coef,
                          second = rep(theta, length(coef)),
                          flag=F)
                          }

cfun.ridge <- function(parms, ...) list(theta=parms$theta, done=T)
```

This psuedo ridge-regression function is simplistic: no provision has been made for factor variables and there is no scaling of the penalty with respect to the scale of the covariates. In order to use these functions within `coxph` we need to 'package' them in the following way:

```
ridge <- function(..., theta=1) {
    x <- cbind(...)
    class(x) <- 'coxph.penalty'
    temp <- list( pfun= pfun.ridge,
                  cfun= cfun.ridge,
                  diag=T,
                  cparm=list(theta=theta))
    attributes(x) <- c(attributes(x), temp)
    x
    }

fit0 <- coxph(Surv(futime, fustat) ~ rx + age + ecog.ps, data=ovarian)
fit4 <- coxph(Surv(futime, fustat) ~ rx + ridge(age, ecog.ps, theta=.4),
              data=ovarian)
```

The ridge function 'passes through' the data given to it, with a small number of added attributes:

- the data has class 'coxph.penalty'. This is how the `coxph` routine recognizes penalty terms.

- pfun: the penalty function.

- cfun: the control function. In this case the control function does nothing except echo back the initial θ value and signal completion.

- diag: a flag indicating whether f'' is a diagonal matrix.

- cparm: the paramer vector for cfun.

The penalty function will be called with the coefficients *for the term*, e.g., those corresponding to `age` and `ecog.ps` in the above example, along with the tuning parmeter(s) θ and the number of events in the data set. It needs to return the value of the penalty and it's first and second derivatives. For some penalty functions and values of θ the penalty may be infinite, in which case the flag argument should be set to True. The iteration routines force coefficients to zero when the penalty is infinite (which is not always the mathematically correct solution). The main function of `flag` is to avoid numeric exceptions such as division by zero in the C language code.

In this example f'' is diagonal, and so pfun returns only a vector of second derivatives. This is indicated by `diag=T`. In other cases, such as smoothing splines, pfun will need to return a matrix.

Here is the output of fit0 and fit1 (some lines removed for clarity).

```
>fit0
          coef exp(coef) se(coef)     z      p
    rx -0.815      0.443   0.6342 -1.28 0.2000
   age  0.147      1.158   0.0463  3.17 0.0015
ecog.ps  0.103      1.109   0.6064  0.17 0.8600


>fit1
                                   coef exp(coef) se(coef)    se2       z
                             rx -0.8124     0.444   0.6327 0.6333  -1.284
ridge(age, ecog.ps, theta = 1)..1  0.1468     1.158   0.0461 0.0461   3.184
ridge(age, ecog.ps, theta = 1)..2  0.0756     1.079   0.4429 0.5177   0.171
                                      P
                             rx 0.2000
ridge(age, ecog.ps, theta = 1)..1 0.0015
ridge(age, ecog.ps, theta = 1)..2 0.8600
```

44

One improvement to our function would be better variable names, The printout is so wide that the column for p-values has wrapped onto a new line. (In the earlier part of the paper row labels were edited). Another improvement is to scale the penalty for each variable by it's variance, so that the function will be invariant to the units of the data; the current fit would change if age were given in days, for instance. To accomplish the scaling, we need to pass through the variances of the terms as another argument to the penalty function.

```
pfun.ridge <- function(coef, theta, nevent, vars) {
                      list(penalty= sum(coef^2*vars) * theta/2,
                           first  = theta*coef*vars,
                           second = theta*vars,
                           flag=F)
                           }
```

The changes to the control function are only a bit more complicated:

```
ridge <- function(..., theta=1) {
    x <- cbind(...)
    xname <- as.character(parse(text=substitute(cbind(...))))[-1]
    vars <- apply(x, 2, function(z) var(z[!is.na(z)]))
    class(x) <- 'coxph.penalty'
    temp <- list( pfun= pfun.ridge,
                  cfun= cfun.ridge,
                  diag= T,
                  cparm= list(theta=theta),
                  pparm= vars,
                  varname=paste('ridge(', xname, ')', sep=''))

    attributes(x) <- c(attributes(x), temp)
    x
    }
```

There are three basic changes: an improved variable name is returned as the `varname` attribute, the variances of the columns are passed as the `pparm` vector, and then those variances are used to scale the penalty function. The code to create the variable names is somewhat tricky (the idea is borrowed from the `data.frame` function). Normally, the "input" representation of a function argument x is obtained with `deparse(substitue(x))`, but in this case the expression only returns "..1". We use one more level of indirection; "..." is passed to `cbind` as the expanded list, `substitute` returns the single character string "cbind(age, ecog.ps)", and `parse` is used to break this expression into its 3 parts.

The result is both more useful and more pleasant to read.

```
> coxph(Surv(futime, fustat) ~ rx + ridge(age, ecog.ps, theta=.4),
                data=ovarian)

                    coef exp(coef) se(coef)    se2      z       p
          rx -0.836      0.433   0.6245 0.6250 -1.339 0.18000
   ridge(age)  0.136      1.145   0.0408 0.0424  3.325 0.00089
ridge(ecog.ps)  0.106      1.112   0.5811 0.5919  0.182 0.86000


Iterations: 1 outer, 4 Newton-Raphson
Degrees of freedom for terms= 1.0 1.9
Likelihood ratio test=15.1  on 2.89 df, p=0.00154  n= 26
```

C.2 Psplines

As in the ridge regression example, we need to pass "extra" information to the penalty function, in this case the penalty matrix P.

```
pfun.ps <- function(coef, theta, nevent, P) {
    if (theta >=1) list(penalty=(1-theta), flag=T)
    else {
        if (theta <=0) lambda =0
        else lambda <- (nevent/length(theta))* theta[1] / (1-theta[1])
        list(penalty= c(coef %*% P %*% coef) * lambda /2,
            first  = c(P %*% coef) * lambda,
            second = c(P*lambda),
            flag=F)
        }
    }
```

The penalty function is quite simple. The first new feature is special handling for $\theta = 1$, which leads to an infinite penalty and could cause an arithmetic exception in the underlying C code if not flagged. The first and second derivatives are never referenced by the underlying code in this case, so no values are needed. Secondly, we added some safety checks for illegal values of the tuning parameter θ. (The control function should, of course, never suggest such a value). Last, we have in this case decided to scale the penalty. The partial likelihood in a Cox model is proportional to the number of deaths, and the penalty function is roughly proportional to the number of terms; the initial constant will make fits with the same number of degrees of freedom, but different sample sizes or number of terms in the p-spline basis, have similar sized θ values. Of course the data plays a large role as well: for a fixed θ

46

a truely curvilinear variable will optimize the PPL at a larger degrees of freedom than one with a near linear relationship.

The function which appears in the user's model statment is

```
pspline <- function(x, theta, df=4, nterm=2.5*df, degree=3,
                          eps=.1) {
    if (!missing(theta)) {
        if (theta <=0 || theta >=1) stop("Invalid value for theta")
        }
    else if (df ==0) {
        method <- 'aic'
        nterm <- 15    #will be ok for up to 6-8 df
        if (missing(eps)) eps <- 1e-5
        }
    else {
        method <- 'df'
        if (df <=1) stop ('Too few degrees of freedom')
        }
    xname <- deparse(substitute(x))

    keepx <- !is.na(x)
    rx <- range(x[keepx])
    nterm <- round(nterm)
    if (nterm < 3) stop("Too few basis functions")
    if (df <= differ-1) stop("Too few degrees of freedom")
    dx <- (rx[2] - rx[1])/nterm
    knots <- c(rx[1] + dx*((-degree):(nterm-1)), rx[2]+ dx*(0:degree))
    if (all(keepx)) newx <- spline.des(knots, x, degree+1)$design
    else
        temp <- spline.des(knots, x[keepx], degree+1)$design
        newx <- matrix(NA, length(x), ncol(temp))
        newx[keepx,] <- temp

    newx <- newx[,-1]                #redundant coefficient
    class(newx) <- 'coxph.penalty'
    nvar <- 1+ ncol(newx)    #should be nterm + degree
    dmat <- diag(nvar)
    dmat <- apply(dmat, 2, diff, 1, 2)
    P <- t(dmat) %*% dmat
    P <- P[-1,-1]                    # rows corresponding to the 0 coef
    xnames <-paste('ps(', xname, ')', 2:nvar, sep='')
```

```
        if (method=='fixed') {
            temp <- list(pfun=pfun.ps,
                         printfun=printfun.ps,
                         pparm=dmat,
                         diag =F,
                         cparm=list(theta=theta),
                         varname=xnames,
                         cfun = function(parms, iter, old)
                                     list(theta=parms$theta, done=T))
            }
        else if (method=='df') {
            temp <- list(pfun=pfun.ps,
                         printfun=printfun.ps,
                         diag =F,
                         cargs=('df'),
                         cparm=list(df=df, eps=eps, thetas=c(1,0),
                                     dfs=c(1, nterm), guess=1 - df/nterm, ...),
                         pparm= dmat,
                         varname=xnames,
                         cfun = frailty.controldf)
            }
        else { # use AIC
            temp <- list(pfun=pfun.ps,
                         printfun=printfun.ps,
                         pparm=dmat,
                         diag =F,
                         cargs = c('status', 'df', 'plik'),
                         cparm=list(eps=eps, init=c(.5, .95),
                                     lower=0, upper=1, ...),
                         varname=xnames,
                         cfun = frailty.controlaic)
            }

    attributes(newx) <- c(attributes(newx), temp)
    newx
    }
```

- The first few lines do some error checking.

- The next lines create the matrix of basis functions. This is essentially a reprise of the pspl function shown earlier, with significant nuisance value added by the possibility of missing values. The pspline function is called *before* the

removal of missing values from the data frame by the `na.omit` function. Thus it must allow for, and propogate, missings.

- The next three lines create the penalty matrix P.

- If θ is fixed, then the control function needs only to set the initial value and signal completion; similarly to the ridge regression example above. If the user has set the degrees of freedom, then a calibration function is required, if AIC then the AIC control function. In the latter 2 cases some initial values for the functions are supplied in `cparms`.

- Some special control for printing has been added.

The first three arguments to the control function are always the parameter vector given by `cparms`, the iteration number, and the list returned by the control function from its last call (this allows storage of local variables without using fancy frame=0 tricks). There may be further arguments; the `cargs` parameter is used above to signal that the estimated degrees of freedom for the current term, at the current values of the fitted parameters, should be added as a fourth argument by the parent routine when `frailty.controldf` is called. Allowable "extra" arguments are

- coef: The coefficients for the term

- df: The degrees of freedom for the term

- x: The columns of the X matrix for this term

- status: The vector of status (censoring) values

- plik: The current value of the partial likelihood PL

- loglik: The penalized partial likelihood PPL

- trH: trace$[(H^{-1})_{22}]$, used for the REML Gaussian.

The last 4 of these are used by the frailty functions. (Others may be added to this list in later versions of the program.)

Here are the first lines of the calibration function

```
frailty.controldf <- function(parms, iter, old, df) {
    if (iter==0) {
        theta <- parms$guess
        theta[1] <- (parms$nterm -parms$df)/(parms$nterm-1)
        return(list(theta=theta, done=F,
                    thetas=parms$thetas, dfs=parms$dfs)
```

49

```
        }

    done <- (iter>1 &&
            (abs(df-parms$df) < parms$eps))

    thetas <- c(old$thetas, old$theta)
    dfs    <- c(old$dfs, df)
    newtheta <- newguess(thetas, dfs, parms$df)
    list(theta=theta, done=done, thetas=thetas, dfs=dfs)
    }
```

The control function has 4 steps. First, it maintains two vectors `thetas` and `dfs` that contain the history of guesses so far. The return values of the function are the next guess at θ, a flag, and these history lists. The final return value is included in the `coxph` output as the `history` component in the final model object, should the user want to examine them.

Second, at iteration 0, it returns a first guess for θ along with initial values for the `thetas` and `dfs` vectors based on input parameters. In the `pspline` function, these were set based on the known fact that $\theta = 0, 1$ correspond to nterms and 1 degree of freedom, respectively, and a linear interpolation between these two for the first guess at a solution.

Third, if this is not iteration 0, then the function checks to see whether it has finished iteration, by comparing `df` (the value resulting from this function's last guess at θ) to the target value `parms$df`.

Last, whether the fit has completed or not, the routine obtains a next guess at θ. The reason for this is that there may be multiple penalized terms in the model, and iteration may need to continue even though this particular term has converged successfully. If that is the case, a new guess at θ is *required* by the parent routine, and it might as well be a good one. The `newguess` function is a simple interpolation method and will not be listed.

The print function is shown below:

```
printfun <- function(coef, var, var2, df, history, cbase)
    test1 <- coxph.wtest(var, coef)$test
    # cbase contains the centers of the basis functions
    #   do a weighted regression of these on the coefs to get a slope
    xmat <- cbind(1, cbase)
    xsig <- coxph.wtest(var, xmat)$solve    # V X , where V = g-inverse(var)
    cmat <- coxph.wtest(t(xmat)%*% xsig, t(xsig))$solve[2,] #[X'VX]^-1 X'V
    linear <- sum(cmat * coef)
    lvar1  <- c(cmat %*% var %*% cmat)
```

```
lvar2  <- c(cmat %*% var2%*% cmat)
test2 <- linear^2 / lvar1
cmat <- rbind(c(linear, sqrt(lvar1), sqrt(lvar2),
                 test2, 1, 1-pchisq(test2, 1)),
              c(NA, NA, NA, test1-test2, df-1,
                1-pchisq(test1-test2, df-1)))
dimnames(cmat) <- list(c("linear", "nonlin"), NULL)
theta <- history$thetas[length(history$thetas)]
list(coef=cmat, history=paste("Theta=", format(theta)))


printfun[[6]] <- knots[2:nvar] + (rx[1] - knots[1])
```

This function will be passed along as part of the output structure, and then invoked by the coxph print routine at the time that a result is displayed. It is called with the coefficients that correspond to the penalized term, along with the appropriate *portions* of the H^{-1} and V matrices, the degrees of freedom for the term, and the history structure for the term (the last return values of the control function).

The routine returns two peices of output: one or more lines to be inserted into the printed table of coeffcients, and an optional line of further information that is printed just below the iteration count for the model. The first object must be a matrix or a vector, and fit into the table. It therefore has 6 elements or columns, which will list under the headings of "coef", "std(coef)", "std2", "Chisq", "DF", and "p". Missing values will be printed as blanks.

For the second part of the printout, this function lists the final value of θ. Note that this is *not* the value of `history$theta`, which contains the next value of θ that would have been tried, had iteration continued.

For the 'coefficient' printout, we have decided to print tests of the linear and nonlinear portions of the fit. Because of the nature of our pspline basis functions, any evenly spaced contrast vector c would give the same χ^2 statistic and p-value for the test of linearity. The test of non-linearity is defined as the difference between the overall test for non-zero coefficients and the linear portion (test2 - test1). As a coefficient for the linear test, we have printed an approximation to the fit of a simple linear term. Consider the least-squares line through the fitted coefficients. The base of the line is the x-coordinates of the centers of the basis functions, which are evenly spaced over the range of x, the vector `cbase` above. The coefficients are not independent, i.e., `var` is not diagonal, so the formula for the fitted line is a weighted linear regression. The `coxph.wtest` function is similar to the S-Plus `solve` function. Because it assumes that its first argument is symmetric it can use the relatively fast Cholesky decomposition to compute the result (exactly the same

routines as used by `coxph`), but more importantly, the routine does not fail for singular matrices, rather it produces a generalized inverse solution.

The last line is more S-Plus magic. At the time the print function is invoked the range of x will no longer be available, only the (static) function definition. This line changes the function so that the range of x, known at the time of function definition, becomes the default value for argument 6.

C.3 Frailty

Here is the penalty function for gamma frailties.

```
pfun.gfrail <- function(coef, theta, nevent) {
        if (theta==0) list(recenter=0, penalty=0, flag=T)
          else {
              recenter <- log(mean(exp(coef)))
              coef <- coef - recenter
              nu   <- 1/theta
              list(recenter = recenter,
                    first=   (exp(coef) -1) *nu,
                    second=  exp(coef) * nu,
                    penalty= -sum(coef) *nu,
                    flag=F)
                    }
          }
```

The first thing you might notice is that the penalty function is missing a portion, $(\nu \sum \exp(\nu)) + \nu \log(\nu) - \log(\Gamma(\nu))$. In line with the concerns found in section 3.1, it is better to combine these terms with the L_g corrections. The function below computes the overall correction term along with the missing PLL term from `pfun`

```
frailty.gammacon <- function(d, nu) {
    nfrail <- length(d)
    maxd <- max(d)
    if (nu > 1e7*maxd) term1 <- sum(d*d)/nu   #second order Taylor series
    else              term1 <- sum(d + nu*log(nu/(nu+d)))

    tbl <- table(factor(d[d>0], levels=1:maxd))
    ctbl<- rev(cumsum(rev(tbl)))
    dlev<- 1:maxd
    term2.numerator <- nu + rep(dlev-1, ctbl)
    term2.denom     <- nu + rep(dlev, tbl*dlev)
    term2 <- sum(log(term2.numerator/term2.denom))
```

```
    term1 + term2
    }
```

`Term2` is formula 4, and the term 1 approximation is based on a Taylor series for the logarithm.

The penalty function has one important addition over previous examples. A frailty terms adds a set of indicator variables to the model, one indicator per 'family'. Since each subject has one and only one of these indicators equal to 1, the set of coefficients γ for the indicators could be replaced by $\gamma + c$ for any constant c without changing the value of the Cox partial likelihood PL. The second term in the penalty $\nu \sum [\gamma_i + c - \exp(gamma_i + c)]$ is minimized when $\exp(c) = \text{mean} \exp(gamma)$. This is true trivially for the starting estimate $\gamma = 0$, and under full Newton-Raphson iteration it remains true at each iteration (within numerical precision). We stated earlier that the Newton-Raphson iteration preserves the identity $E(\varpi_i) = 1$, which is the same statement. When using routines based on a sparse approximation this is no longer algebraicly true. One function of the penalty routine is to recenter the coefficient vector so that the identity does hold. Doing so significantly speeds the convergence of the algorithm.

```
    frailty.gamma <- function(x, sparse=T, theta, df, eps=1e-5,
                        method=c('em', 'aic', 'df', 'fixed', ...) {
        if (sparse){
            x <-as.numeric(as.factor(x))
            class(x) <- "coxph.penalty"
            }
        else {
            x <- as.factor(x)
            class(x) <- c("coxph.penalty", "factor")
            attr(x,'contrasts') <- function(n,...) contr.treatment(n,F)
            }
        if (missing(method))
            if (!missing(theta))
                method <- 'fixed'
                if (!missing(df))
                        stop("Cannot give both a df and theta argument")

            else if (!missing(df)) method <- 'df'

        method <- match.arg(method)
```

```
            if (method=='em')
                temp <- list(pfun=pfun,
                                printfun=printfun,
                                diag =T,
                                sparse= sparse,
                                cargs = c("x", "status", "loglik"),
                                cfun = frailty.controlgam,
                                cparm= c(list(eps=eps), ...))


            else if (method=='aic')
                temp <- list(pfun=pfun,
                                printfun=printfun,
                                diag =T,
                                sparse= sparse,
                                cargs = c("x", "status", "loglik", "df", "plik"),
                                cparm=list(eps=eps, lower=0, init=c(.1, 1), ...),
                                cfun =function(opt, iter, old, x, status, loglik,...)
                                    temp <- frailty.controlaic(opt, iter, old, status,...)
                                    if (iter >0)
                                        #compute correction to the loglik
                                        if (old$theta==0) correct <- 0
                                        else
                                            if (is.matrix(x))
                                                    x <-c(x %*% 1:ncol(x))
                                            d <- tapply(status,x,sum)
                                            correct <- frailty.gammacon(d, 1/old$theta)

                                        temp$c.loglik <- loglik + correct

                                    temp
                                    )
                else  etc...

        attributes(x) <- c(attributes(x), temp)
        x
        }
```

The routine accomodates 4 options. The AIC solution uses the same control
function as for splines, but adds one more piece of information to the returned list,
namely the marginal likelihood L_g for a gamma model. If sparse=F then x will be
the matrix of indicator variables, otherwise a vector containing the grouping code;

the function to compute the correction term needs a vector containing the number of events in each group.

The solution for `method='df'` and `fixed` are similar to those for splines, and are not shown.

If method=em then the actual frailty problem is solved, using a profile likelihood search for θ. The frailty.controlgam program has only a few variations on the prior code that has been listed. Since almost all problems seen to date have a final value in (0,1), so 0 and 1 are used as the first two guesses for the frailty. Guesses at 2, 4, 8, ... are then tried until the solution has been bracketed. Once that has occured, Brent's algorithm [24] is used to perform the search for a maximizing value of θ.

D   Internal calls

For certain applications, it may be more useful to call the fitting function `coxpenal.fit` directly, for instance if a single penalization is to be applied across several terms of the model. This routine solves penalized models for both right censored and (start, stop] survival data. The arguments to the function are the same as those to coxph.fit, with three additions: `pcols`, `pattr` and `assign`.

The third argument `assign` may be omitted, in which case the assign attribute of the X matrix is assumed. It is simply a list which groups the columns of the X matrix into an arbitrary collection of 'terms', and gives names to each term. The `df` component of the returned fit has one value per term, containing the degrees of freedom for that term.

The `pcols` argument is similar to `assign`, and must be a strict subset of it. Each of the terms given in `pcols` will be penalized with a separate penalty function.

The `pattr` list contains most of the essential information. This list has one element for each term identified in pcols, and that element is itself a list with the following components:

- pfun: the penalty function

- cfun: the control function

- sparse: whether the term is to be solved using sparse methods

- diag: whether the penalty function is diagonal

- cargs, pparm, cparm: argument lists

- printfun: an optional print function

Each of these has been described in the earlier section.

If a term is sparse, then it must correspond to a single column of the X matrix which contains the group indices 1 – number of groups. No other contraints are made on X. The numerical optimizer can deal with only 1 sparse term.

The return value of the function contains the same components as that for `coxph.fit`, with these additions.

- var2: The alternate variance estimator $H^{-1}\mathcal{I}H^{-1}$.

- iter: The iter vector is of length 2, containing the number of outer iterations used by the control function, and the total number of Newton-Raphson steps.

- frail: If there was a sparse term, its coefficients are here rather than in the coefficient vector.

- fvar: Approximate variances for the sparse term.

- df: Degrees of freedom for each term.

- penalty: The value of the penalty function at the initial and final iterations.

- pterms: A vector of the same length as `assign`, with the code 0=ordinary term, 1=penalized term, 2=sparse penalized term.

- assign2: The assign component, minus the response and sparse terms. It is useful for printing because of it maps the coefficient vector.

- history: A list with one component per penalized term. Each component is a list, containing the last returned value of the control function for that term.

- printfun: The optional list of print functions.

References

[1] O.O Aalen. Heterogeneity in survival analysis. *Statistics in Medicine*, 7:1121–1137, 1988.

[2] N.E. Breslow and D.G. Clayton. Approximate inference in generalized linear mixed models. *J Amer Stat Assoc*, 88:9–25, 1993.

[3] J.M. Chambers and T.J. Haste. *Statistical Models in S*. Wasdworth, Belmont, California, 1990.

[4] D.R. Cox and D.O. Oakes. *Analysis of survival data*. Chapman and Hall, London, 1984.

[5] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J Royal Stat Soc B*, 39:1–38, 1977.

[6] J.H. Edmunson, T.R. Fleming, D.G. Decker, E.O. Jorgensen, G.D. Malkasian, J.A. Jeffries, M.J. Webb, and L.K. Kvols. Chemotherapeutic sensitivity of minimal residual disease following surgical excision of ovarian carcinoma. *Cancer Treatment Reports*, 63:241–247, 1979.

[7] P.H. Eilers and B.D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11:89–121, 1996.

[8] T.R. Fleming and D.P. Harrington. *Counting Processes and Survival Analysis*. Wiley, New York, 1991.

[9] R.J. Gray. Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. *J Amer Stat Assoc*, 87:942–951, 1992.

[10] T.J. Hastie. Pseudosplines. *J Royal Stat Soc B*, 58:379–396, 1996.

[11] T.J. Hastie and R.J. Tibshirani. *Generalized Additive Models*. Chapman and Hall, London, 1990.

[12] C.M. Hurvich, J.S. Simonoff, and Chih-Ling Tsai. Smoothing parameter selection in nonparametric regression using an improved akaike information criterion. *J Royal Stat Soc B*, 60:271–293, 1998.

[13] J.D. Kalbfleisch and R.L. Prentice. *The Statistical Analysis of Failure Time Data*. Wiley, New York, 1980.

[14] S.F. Lawless and K. Singhall. Efficient screening of nonnormal regression models. *Biometrics*, 34:318–32, 1978.

[15] D.Y. Lin. Cox regression analysis of multivariate failure time data, the marginal approach. *Statistics in Medicine*, 13:2233–2247, 1994.

[16] N. Mantel, N.R. Bohidar, and J.L. Ciminera. Mantel-Haenszel analysis of liter-matched time-to-response data with modifications for recovery of interlitter information. *Cancer Research*, 37:3863–3868, 1977.

[17] C.A. McGilchrist. REML estimation for survival models with frailty. *Biometrics*, 49:221–225, 1993.

[18] C.A. McGilchrist and C.W. Aisbett. Regression with frailty in survival analysis. *Biometrics*, 47:461–466, 1991.

[19] C.A. McGilchrist and K.K.W. Yau. The derivation of BLUP, ML and REML estimation methods for generalised linear mixed models. *Comm Stat Theory and Methods*, 24:2963–2980, 1995.

[20] C.G. Moertel, T.R. Fleming, J.S. McDonald, D.G. Haller, J.A. Laurie, P.J. Goodman, J.S. Ungerleider, W.A. Emerson, D.C. Tormey, J.H. Glick, M.H. Veeder, and J.A. Mailliard. Levamisole and fluorouracil for adjucant therapy of resected colon carcinoma. *New England J Medicine*, 332:352–358, 1990.

[21] A.J. Moss and the Multicenter Post-Infarction Research Group. Risk stratification and survival after myocarial infarction. *New England J. of Medicine*, 309:331–336, 1983.

[22] G.G. Nielsen, R.D. Gill, P.K. Andersen, and T.I. Sørensen. A counting process approach to maximum likelihood estimation of frailty models. *Scandinavian J of Statistics*, 19:25–43, 1992.

[23] E. Parner. A note on the EM-algorithm for frailty models. personal communication, 1996.

[24] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1988.

[25] T.M. Therneau, P.M. Grambsch, and T.R. Fleming. Martingale based residuals for survival models. *Biometrika*, 77:147–160, 1990.

[26] G. Wahba. Bayesian confidence intervals for the cross-validated moothing spline. *J Royal Stat Soc B*, 45:143–150, 1983.

[27] K.K.W. Yau and C.A. McGilchrist. Use of generalised linear mixed models for the analysis of clustered survival data. *Biometrical Journal*, 39:3–11, 1997.