Creating Special Variance Structures for Coxme

Terry Therneau and Marianne Huebner
Mayo Clinic

December 28, 2011

1 Introduction

The `coxme` function allows the user to create their own variance handling routines. This allows the program to be extended in ways not visualized by the author. This short vignette gives an example of such a structure.

We would like to fit the following random effects model

$$\begin{aligned} \lambda(t) &= \lambda_0(t) e^{X\beta + Zb} \\ b &= N(0, V(\theta) \\ \theta &= (\sigma^2, \rho) \\ V_{ij} &= \begin{cases} \sigma^2 & \text{if } i = j \\ \sigma^2 \rho & \text{otherwise} \end{cases} \end{aligned}$$

The motivating example is the use of gene sets in a lung cancer study. Here X is a matrix of known covariates such as grade and stage of the tumor and Z is a matrix of gene expression levels from a microarray experiment; each column of Z is one gene from a specified gene set. There is a separate Z for each gene set, the number of columns of Z typically ranges from 10 to 100.

When $\rho = 1$ this model constrains all of the coefficients to be identical. This is equivalent to a shrinkage method found commonly in the utilization of questionnaires, where all of the questions in a particular domain are summed to get an overall score for that domain which is then used as a covariate in a model. When $\rho = 0$ it becomes simple L_2 shrinkage of the coefficients. The above model allows for a fit where the coefficients for a gene set are constrained to be close to each other, but not identical.

The coxme code for this model will be

```
> fit <- coxme(Surv(time, status) ~ stage + histology + (z|1),
               data=mydata, varlist=gexchange)
```

2 Variance functions

The variance function `gexchange` consists of a list with 3 elements: an initialize function, a generation function , and a closing function.

```
> gexchange <- function() {
     out <- list(initialize = geinit,
                 generate  = gegenerate,
                 wrapup    = gewrapup)
     class(out) <- "coxmevar"
     out
 }
```

2.1 Initialization function

The initialization function is often the most tedious to create. It is responsible for 4 tasks:

1. Verify that the input data is consistent with the variance function
2. Process any initial values given by the user
3. Transform parameters
4. Create the return value containing the starting estimates and the control parameters.

Input arguments are

- vinit: a list or vector of initial estimates for the parameters, as supplied by the user. This may be empty. An example would be `vinit=list(sigma=c(.1, .3), rho=c(.1, .8))`. This would evaluate the fit over a 2 by 2 grid of values, the best fit found becomes the starting estimate for the full maximization. The routine below requires that the starting estimates be named and that both are supplied, other routines could be more or less flexible.
- vfixed: a vector of fixed values for the parameters, as supplied by the user. For instance adding the clause `vfixed=list(rho=.4)` could be used to fit a model where ρ is fixed at 0.4.
- intercept: Signals that the term includes random intercepts per group. This would be true for a term like `(1+ trt | site)`, but is invalid for this variance function (there are no groups).
- G: A vector or matrix containing the grouping variables found on the right hand side of the vertical bar.
- X: A matrix of covariates found on the left of the vertical bar.
- sparse: User specified parameter which advises with respect to the use of sparse matrices. It is ignored in this function.

Note that this is currently called from the main routine using positional unnamed arguments. Thus you can name the arguments otherwise, but not reorder them.

```
> geinit <- function(vinit, vfixed, intercept, G, X, sparse) {
     if (length(G) >0)
         stop("This function does not handle groupings")
```

```
    if (length(X)==0)
        stop("No covariates given!")
    if (ncol(X) <2)
        stop("Only one random slope, correlation is irrelevant")

    if (length(vinit) >0) {
        if (length(vinit) !=2) stop("Wrong length for vinit")
        indx <- match(names(vinit), c("sigma", "rho"))
        if (any(is.na(indx)))
            stop("Unrecognized parameter name in vinit values")

        theta <- list(vinit[[indx[1]]], vinit[[indx[2]]])
    }
    else theta <- list(c(.05, .2, .6)^2, c(.01, .5, .9))

    which.fixed <- c(FALSE, FALSE)
    if (length(vfixed) >0) {
        indx <- match(names(vfixed), c("sigma", "rho"))
        if (any(is.na(indx)))
            stop("Unrecognized parameter name in vfixed values")
        if (is.list(vfixed)) {
            if (any(sapply(vfixed, length) !=1))
                stop("Any fixed parameter must have a single value")
            vfixed <- unlist(vfixed)
        }
        which.fixed[indx] <- TRUE
        theta[which.fixed] <- vfixed[which.fixed]
    }

    if (any(theta[[1]] <=0)) stop("Variance must be >0")
    if (any(theta[[2]] <0) || any(theta[[2]] >=1))
        stop("Correlation must be between 0 and 1")

    theta[[2]] <- theta[[2]]/(1-theta[[2]])

    # In the shape of X, first column=1, second =2, etc
    xmap <- matrix(rep(1:ncol(X), each=nrow(X)), nrow(X))

    list(theta=lapply(theta, log)[!which.fixed], imap=NULL,
         X=X, xmap=xmap,
         parms=list(theta=sapply(theta, function(x) x[1]),
                    fixed=which.fixed,
                    xname=dimnames(X)[[2]], nvar=ncol(X)))
}
```

The first six lines check that this variance structure is relevant for the random term as specified. The next block of statements processes any initial values for the parameters. The parameter vector as a whole is $\theta = (\sigma^2, \rho)$ ="theta". If no initial values were given then a default grid of values for the initial search is $\sigma = .05, .2, .6$ and $\rho = .01, .5, .9$. Note that the final θ must be formatted as a list, even if σ and ρ were each a single value.

The next block of lines checks for any fixed parameters. If any are found the corresponding elements of `which.fixed` are set and the initial value is reset. For iteration we will use transformed parameters:

$$\tilde{\theta}_1 = \log(\sigma)$$
$$\tilde{\theta}_2 = \log\left(\frac{\rho}{1-\rho}\right)$$

The actual optimization is done by the `optim` routine, and we need it to operate in an unconstrained space. No transform is required for the fixed parameters, which are never made visible to the optim routine.

Finally we create the returned value, which is a list with components:

- theta: A list with one element for each parameter value which is to be optimized, each element is a vector of potential starting values, on the transformed scale. The list can be NULL, for instance when all the parameters are fixed.
- imap: mapping between intercept terms and coefficients
- X: the covariate matrix, possibly transformed, an example might be prescaling
- xmap: mapping between slopes and coefficients
- parms: an arbitrary list which will be passed forward to the generate and wrapup routines, the main coxme code does not examine it.

The `imap` and `xmap` vectors are important for some calculations but not here. For instance, in familial studies the per-subject random intercepts are correlated within familial blocks. For storage and computational efficiency the relevant variance function uses a bsdmatrix object for the variance. This requires that all the coefficients for a family are adjacent in the parameter vector, but the subject identifier variable might not be in this order. A value for `imap` of 1, 7, 4, ... would map subject 1 to coefficient 1, subject 7 to coffined 2, subject 3 to coefficient 4, etc. A valid imap value will be a matrix with one column per grouping variable (the term (1| school/teacher) would have 2 columns), each row contains the coefficient number for that subject. The xmap matrix is the same shape as Z, and shows for each subject the coefficient number which multiplies the relevant covariate.

2.2 Generation function

The input arguments to this function are the current vector of iterated parameters, along with the list of parameters supplied by the init function. The result of the routine is the covariance matrix V of the random effects for the term.

One of the greatest challenges for this routine is to guarantee that the resulting matrix is positive definite to a sufficient degree that the creation of V^{-1} will give a full rank result. In

simulations, we found that under the null hypothesis of no association between the gene set and survival that somewhat more than 1/2 of the solutions had $\hat{\sigma} = 0$, i.e., this is the actual maximizing value of the likelihood. Thus, for about half the no-association cases the code is iterating towards 0 as a solution. In this case a race condition ensues. As $\log(\sigma^2)$ goes to $-\infty$ in the maximizer, the variance matrix V approaches singularity while the random coefficients b approach zero. Since the partial log-likelihood (PL) depends only on the linear predictor $X\beta + Zb$ it quickly asymptotes to a its final value which is the PL for the fixed effects model $X\beta$. Convergence of the PL to within the final tolerance gives successful termination; a singular matrix leads to program failure.

In the `coxme.control` routine we see that the default convergence criteria for the partial likelihood is 1e-8, whereas the singularity test for the variance matrix is `.Machine$double.eps^.75` $\approx$ 2e-12. Since the default convergence criteria is larger than the singularity tolerance normally all proceeds well: the likelihood is declared to converge before singularity occurs. However, occasionally the maximizer makes a very large step in the negative direction leading to $\log(\sigma^2)$ values that are extremely small. We try to protect against this. The Affymetrix gene-set for which this routine was developed had values between 6 and 15, which in turn means that any coefficient less than about .0001 has a negligible effect on the linear predictor. Variances of about $\exp(-20)$ are large enough to avoid issues with the matrix inversion, but small enough that the linear predictor will have effectively converged: b is are approximately Gaussian with a standard deviation of .00004 so $Zb < .00008 * 15$. The second limit was in response to a single data set where an intermediate guess in the maximization gave too large a variance, far beyond what is biologically feasible, which in turn led to overflow of the exp function in calculation of $\exp(X\beta + Zb)$.

```
> gegenerate <- function(newtheta, parms) {
     safe.exp <- function(x, emax=20) {
         exp(pmax(-emax, pmin(emax, x)))
     }

     theta <- parms$theta
     if (!all(parms$fixed))
         theta[!parms$fixed] <- safe.exp(newtheta)

     correlation <- theta[2]/(1+theta[2])
     variance    <- min(theta[1],20)   #keep it out of trouble
     varmat <- matrix(variance*correlation, nrow=parms$nvar,
                      ncol=parms$nvar)
     diag(varmat) <- variance
     varmat
 }
```

2.3 Wrapping up

The wrapup routine has two tasks. The first is simple: to return any transformed parameters back to their original scale. The second is to add names and structure to the results so that they will print with reasonable names.

The return value is the vector of final θ values, and a list containing the random coefficient vector b.

```
> gewrapup <- function(newtheta, b, parms) {
     theta <- parms$theta
     theta[!parms$fixed] <- exp(newtheta)
     correlation <- theta[2]/(1+theta[2])

     rtheta <- list('(Shrinkage)' = c(variance=theta[1],
                                      correlation=correlation))
     names(b) <- parms$xname
     list(theta=rtheta, b=list(X=list(b)))
 }
```

Here then is an example using gexchange and a null gene set which has no relevance. The `coxme` routine accepts either a coxmevar object or a function which generates such an object as the argument to the `varlist` option.

```
> gexchange <- function() {
     out <- list(initialize = geinit,
                 generate  = gegenerate,
                 wrapup    = gewrapup)
     class(out) <- "coxmevar"
     out
 }
> require(coxme)
> set.seed(1960)
> n <- nrow(lung)
> dgene <- matrix(rnorm(n*12, mean=8, sd=2), ncol=12)
> fit <- coxme(Surv(time, status) ~ age + ph.ecog + (dgene |1),
              data=lung, varlist=gexchange)
> print(fit)

Cox mixed-effects model fit by maximum likelihood
  Data: lung
  events, n = 164, 227 (1 observation deleted due to missingness)
  Iterations= 12 50
                     NULL Integrated    Fitted
Log-likelihood -744.4805  -734.9533 -734.9522

                  Chisq df          p   AIC   BIC
Integrated loglik 19.05  4 7.6688e-04 11.05 -1.35
 Penalized loglik 19.06  2 7.2962e-05 15.05  8.85

Model:  Surv(time, status) ~ age + ph.ecog + (dgene | 1)
Fixed coefficients
              coef exp(coef)   se(coef)    z       p
```

```
age     0.01128166  1.011346 0.00931941 1.21 0.23000
ph.ecog 0.44348947  1.558135 0.11583153 3.83 0.00013

Random effects
 Group       Variable    Std Dev      Variance
 (Shrinkage) variance    5.191567e-04 2.695237e-07
             correlation 9.511232e-01 9.046354e-01

> coxme(Surv(time, status) ~ age + ph.ecog + (dgene |1),
              data=lung, varlist=gexchange,
              vfixed=c(sigma=.01))

Cox mixed-effects model fit by maximum likelihood
  Data: lung
  events, n = 164, 227 (1 observation deleted due to missingness)
  Iterations= 11 46
                    NULL Integrated    Fitted
Log-likelihood -744.4805  -737.0109 -734.7571

                  Chisq   df          p   AIC   BIC
Integrated loglik 14.94 3.00 0.00186950  8.94 -0.36
 Penalized loglik 19.45 3.11 0.00025196 13.22  3.57

Model:  Surv(time, status) ~ age + ph.ecog + (dgene | 1)
Fixed coefficients
              coef exp(coef)    se(coef)    z       p
age     0.01151817  1.011585 0.009334737 1.23 0.22000
ph.ecog 0.44600678  1.562062 0.116043415 3.84 0.00012

Random effects
 Group       Variable    Std Dev   Variance
 (Shrinkage) variance    0.1000000 0.0100000
             correlation 0.9990895 0.9981798
```

3 Alternate arguments

Arguments to a variance function do not need to be passed via the `vinit` and `vfixed` arguments of `coxme`. Here is a very simple variance function which was used when testing out a model with random treatment effects, i.e.

```
> coxme(Surv(time, status) ~ stage + trt + (1+trt | site), data=mydata,
       varlist=myvar(c(.1, .2, .4)))
```

The model has a random intercept and slope for each site, and during testing we wanted to evaluate this for various fixed values of the three parameters: variance of the random intercepts, variance of the random slopes, and correlation.

```
> myvar <- function(var) {
     initialize <- function(vinit, fixed, intercept, G, X, sparse) {
         imap <- as.matrix(as.numeric(G[[1]]))
         ngroup <- max(imap)
         v2 <- var
         v2[3] <- v2[3]*sqrt(v2[1]*v2[2])  #covariance
         list(theta=NULL, imap=imap, X=X, xmap=imap+ngroup,
              parms=list(v2=v2, theta=var, ngroup=ngroup))
     }
     generate <- function(newtheta, parms) {
         theta <- parms$v2
         varmat <- diag(rep(theta[1:2], each=parms$ngroup))
         for (i in 1:4) varmat[i,i+ngroup] <- varmat[i+ngroup,i] <- theta[3]
         varmat
     }

     wrapup <- function(newtheta, b, parms) {
         theta <- parms$theta
         rtheta <- list(site=c(var1=theta[1], var2=theta[2], cor=theta[3]))
         b <- matrix(b, ncol=2)
         dimnames(b) <- list(NULL, c("Intercept", "Slope"))
         list(theta=rtheta, b=list(site=b))
     }
     out <- list(initialize=initialize, generate=generate, wrapup=wrapup)
     class(out) <- 'coxmevar'
     out
     }
```

This routine ignores any vinit or vfixed arguments and does absolutely no error checking; it assumes for instance that Z has exactly one column and that there is exactly one grouping variable; the input argument is assumed to be length three with valid values. The routine's advantage was that it took only a few minutes to write. (I don't generally recommend using no error checks, by the way.) The random effects b are arranged as the four intercepts followed by the four slopes, we display them as a matrix for the user's convenience.