On mixed-effect Cox models, sparse matrices, and modeling data from large pedigrees

Terry M Therneau

December 31, 2007

Contents

1   Introduction

This technical report attempts to document many of the thoughts and computational issues behind the S-Plus/R *kinship* library, which contains the `coxme` and `lmekin` functions. Like many other projects, this really started with a data set and a problem. From this came statistical ideas for a solution, followed by some initial programming — which more than anything else helped to define the *real* computational and statistical issues — and then a more ambitious programming solution. The problem turned out to be harder than I thought; the first release-worthy code has taken over 3 years in gestation.

For several years I have been involved in an NIH funded program project grant of Dr. Tom Sellers; the goals of the grant are to further understand genetic and environmental risk factors for the development of breast cancer. To this end, Dr. Sellers has assembled a cohort of 426 extended families comprising over 26000 individuals. A little under half of these are females, and over 4000 of the females have married into the families as opposed to being blood relatives. The initial population was entirely from Minnesota and the large majority of the participants remain so; in this population is is reasonable to assume little or no ethnic stratification with respect to marriage, so that we can assume that the marry-ins form an unbiased control sample.

In analyzing this data, how should one best adjust for genetic associations when examining other covariates such as parity or early life diet? Both stratification or a single per-family random effect are unattractive, as they assume a consistency within family that need not be there. In particular, in such large pedigrees it is certainly possible that a genetic risk has followed one branch of the family tree and not another. Also, the marry-ins are genetically linked to the tree through their children, but are nevertheless not full blood members. One appealing solution to this is to use a correlated random effects model, where there is a per-patient random effect but correlated according to a matrix of relationships.

This in turn raises two immediate computational issues. The first is simple: with 26050 subjects the full kinship matrix must be avoided, as it would consume almost 4 terabytes of main memory. Luckily, the matrix is sparse in a simply patterned way, and substantial storage and computational savings can be achieved. The second issue is a somewhat more subtle one of design: although it would be desirable to copy the user-level syntax of `lme`, the linear mixed-effects models in S-Plus, we can do so only partially. A basic assumption of lme is that the random effects are the same for each subgroup, both in number and in correlation structure. This is of course not true for a kinship relation: each family is unique.

Details of these issues, examples, further directions, and computational side bars are all jumbled together in the rest of this note. I hope it is enlightening, perhaps enjoyable, but most of all at least comprehensible. Eventually much of this should find its way into a set of (more organized) papers.

2   Software

The *kinship* library is a set of routines designed for use in S-Plus. The centerpiece of the collection are `coxme` and `lmekin`. In terms of sheer volume of code, these are overshadowed by the support routines for sparse matrices (of a particular type), generalized cholesky decomposition, pedigree drawing, and kinship matrices.

The modeling routines are

- coxme: general mixed-effects Cox model. The calling syntax is patterned after `lme`, but with important differences due to the need to handle genetic problems.

- lmekin: a variant of `lme` that allows for genetic correlation matrices. In time, Jose Pinheiro and I have talked about merging its capabilities into `lme`, which would be a good thing since there is virtually no overlap between the type of problem handled by the two routines.

The main pedigree handling routines are

- familycheck: evaluates the consistency of a family id variable and pedigree structure.

- makekinship: create a sparse kinship matrix representing the genetic relation between individuals in a collection of families.

- pedigree and plot.pedigree: create a pedigree structure representing a single family, and plot it in a very compact way. There are many other packages that draw prettier or more general diagrams; the goal of this was to create a reasonably good, fast, automatic drawing that fits on the screen, as a debugging aid for the central programs. It has turned out to be more useful than anticipated.

Supporting matrix routines include

- bdsmatrix: the overall class structure for block-diagonal symmetric matrices.

- gchol: generalized cholesky decomposition, for both ordinary matrices and bdsmatrix objects.

- bdsmatrix.ibd: read in the data representing an identity-by-descent (IBD) matrix from a file, and store it as a bdsmatrix object. The kinship suite does not include routines to compute an ibd matrix directly.

There are also a large number of supporting routines which will rarely if ever be called directly by the user. Descriptions of these routines are found in the appendix.

We have done neither a port to S/Windows nor to R, although we expect others will. This is not an argument against either environment, they are just not the environment that we use, and there are only so many hours in a day. For an R port we are aware of 3 issues:

4

- Trivial: The bdsS.h and coxmeS.h files contain some S-specific definitions. The changes for R are well known.

- Simple: The coxme.fit routine uses the nlminb function of S-Plus for minimization; in R one would use the optim function.

- Moderate(?): The bdsmatrix routines are based on the "new" style class structure. These classes have been added to R, but appear to be in evolution.

3 Random Effects Cox Model

Notationally, we will stay quite close to the standards for linear mixed-effect models, avoiding some of the (unnecessary we believe) notational complexities of the literature on frailty models. The hazard function for the sample is defined as

$$
\begin{aligned}
\lambda(t) &= \lambda_0(t)e^{X\beta+Zb} \text{ or} &(1)\\
\lambda_i(t) &= \lambda_0(t)e^{X_i\beta+Z_ib} &(2)
\end{aligned}
$$

where λ_0 is an unspecified baseline hazard rate, i refers to a particular subject and X_i, Z_i refer to the ith rows of the X and Z matrices, respectively. We will use these two equations interchangeably, appropriate to the context. The X matrix and the parameter vector β represent the fixed effects of the model, and Z, b the random effects.

Because it is the only model that easily generalizes to arbitrary covariance matrices, we will assume the Gaussian random effects model set forth by Ripatti and Palmgren [6],

$$
b \sim N(0,\Sigma). \tag{3}
$$

Integrating the random effect out of the partial likelihood gives an integrated log-partial likelihood $\mathcal{L}$ of

$$
\begin{aligned}
e^{\mathcal{L}} &= \frac{1}{\sqrt{2\pi|\Sigma|}} \int e^{\mathrm{PL}(\beta,b)} e^{-b'\Sigma^{-1}b/2} db &(4)\\
&= \frac{1}{\sqrt{2\pi|\Sigma|}} \int e^{\mathrm{PPL}(\beta,b)} db \\
&= \frac{1}{\sqrt{2\pi|\Sigma|}} e^{\mathrm{PPL}(\beta,\hat{b})} \int e^{-(b-\hat{b})'H_{\hat{b}\hat{b}}(b-\hat{b})/2} db + err &(5)\\
&= e^{\mathrm{PPL}(\beta,\hat{b})} \frac{1}{\sqrt{|H_{\hat{b}\hat{b}}||\Sigma|}} \left[\sqrt{\frac{|H_{\hat{b}\hat{b}}|}{2\pi}} \int e^{-(b-\hat{b})'H_{\hat{b}\hat{b}}(b-\hat{b})/2} db \right] + err \\
\mathcal{L} &= \mathrm{PPL}(\beta,\hat{b}) - \frac{1}{2}\left(\log|H_{\hat{b}\hat{b}}| + \log|\Sigma|\right) + err^* &(6)
\end{aligned}
$$

Equation 4 is an intractable multi-dimensional integral—b has a dimension of order n for correlated frailty problems; the partial likelihood $\exp(\mathrm{PL})$ is a

product of ratios, which is not a "nice" integral with respect to direct solution, and so does not factor out as is sometimes the case with Bayes problems and a conjugate prior.

First, recognize the integrand as $\exp(\text{PPL}(\beta, b))$, where PPL is the penalized partial likelihood consisting of the usual Cox (log) partial likelihood minus a penalty. The Laplace approximation to the integral replaces the argument of exp with a second order Taylor series about $(\beta, \hat{b})$, $f(b) \approx f(\hat{b}) + (b - \hat{b})' f''(\hat{b})(b - \hat{b})/2$. Since $\hat{b}$ is by definition the value with first derivative of 0, the Taylor series has only the second order term. Pulling the constant term out of the integral leads to equation 5. We now recognize the term under the integral as the kernel of a multivariate Gaussian distribution, leading directly to 6. The code assumes that err is ignorable.

Rippatti and Palmgren do a somewhat more formal likelihood derivation, which worries about the implicit baseline hazard λ_0 and the fact that (4) is not actually a likelihood, but arrive at the same final equation. Essentially, the Laplace approximation has replaced a multi-dimensional integration with a multi-dimensional maximization. This is unlike linear mixed-effects models, where the integration over b can be done directly, leading to a maximization over β and the parameters of the variance matrix Σ alone.

The matrix H is the second derivative or Hessian of the PPL, which is easily seen to be $-\mathcal{I}_{bb} - \Sigma^{-1}$, where $\mathcal{I}_{bb}$ is the portion of the usual information matrix for the Cox PL corresponding to the random effects coefficients b. By the product rule for determinants, we can rewrite the second two terms of 6 as

$$-(1/2)\log|\Sigma\mathcal{I}_{bb} + \mathrm{I}|$$

where I is the identity matrix. As the variance of the random effect goes to zero, this converges to $\log|0 + \mathrm{I}| = 0$, showing that the PPL and the integrated likelihood $\mathcal{L}$ coincide at the no frailty case. Computation of the correction term, however, makes use of the form found in equation 6, since those components are readily at hand from the Newton-Raphson calculations that were used to compute $(\hat{\beta}, \hat{b})$ for the PPL maximization.

The `coxme` code returns a likelihood vector of three elements: the PPL for $\Sigma = 0$ and $(\beta, b) = $ initial values (usually 0), the integrated likelihood at the final solution, and the PPL at the final solution. Assume for the moment that $\Sigma = \sigma_1^2 A + \sigma_2^2 \mathrm{I}$ for some fixed matrix A, that $\text{rank}(X) = p$ and that Z has q columns. There are two possible likelihood ratio tests; that based on the integrated likelihood is $2(L_2 - L_1)$, and is approximately χ^2 on $p + 2$ degrees of freedom. That based on the PPL is $2(L_3 - L_1)$ and is only approximately χ^2, on $p + q - \text{trace}(\Sigma^{-1}[H^{-1}]_{bb})$ degrees of freedom, see equation 5.16 of Therneau and Grambsch [9]. As pointed out there, the current code uses an approximation for the p-value that is strictly conservative.

In equation 5 we could expand the quadratic form in terms of H_{bb} as was done, or $[(H^{-1})_{bb}]^{-1}$. The first corresponds to treating $\hat{\beta}$ as a fixed parameter, and yields the MLE estimate proposed by Ripatti and Palmgren. The second implicitly recognizes that b and $\hat{\beta}$ are linked, and is an expansion in terms of

the profile likelihood. It leads to the substitution

$$-\log|H_{bb}| = \log|(H_{bb})^{-1}| \implies \log|(H^{-1})_{bb}|$$

in equation 6. Yau and McGilchrist [10] point out that the Newton-Raphson iteration of a Cox model can be written in the form of a linear model, and that the resultant equation for the update is identical to that for a linear mixed effects model. On this heuristic grounds, the substitution above is called an REML estimate for a mixed-effects Cox model. Computationally, the REML estimate turns out to be more demanding due to algorithmic details of the sparse Cholesky decomposition.

Another important question, of course, is the adequacy of the Laplace approximation at all. It has long been known that the Cox PL is well approximated by a quadratic in the neighborhood of the maximum, if there are an adequate number of events and $X\hat{\beta}$ is modest size: 10 events/covariate and risks < 4 are reasonable values. Within this range the Cox partial likelihood is very quadratic, as evidenced by the fact that a Newton-Raphson procedure unfailingly converges using the simple starting estimates of 0. (For a counterexample with infinite β see [9]). Random effects pose a new case that requires investigation, and within that an exploration of ML vs. REML superiority.

Later code includes the ability to do a monte carlo refinement of the estimate. From equation 6, we have that

$$
\begin{aligned}
err &= \frac{1}{\sqrt{2\pi\Sigma}} \int \left(e^{PPL(\beta,b)} - e^{PPL(\beta,\hat{b})-(b-\hat{b})'H_{\hat{b}\hat{b}}(b-\hat{b})/2} \right) db \\
&= \frac{1}{\sqrt{2\pi\Sigma}} \int \left(e^{PL(\beta,b)} - e^{PPL(\beta,\hat{b})-(b-\hat{b})'H_{\hat{b}\hat{b}}(b-\hat{b})/2+b'\Sigma^{-1}b/2} \right) e^{-b'\Sigma^{-1}b/2} db \\
&\equiv \frac{1}{\sqrt{2\pi\Sigma}} \int G(b;\beta,\hat{b}) e^{-b'\Sigma^{-1}b/2} db
\end{aligned}
\tag{7}
$$

which can be estimated by the sum

$$\widehat{err} = (1/m) \sum_{j=1}^{m} G(b_i;\beta,\hat{b})$$

where b_i is a random vector drawn from a $N(0,\Sigma)$ distribution. Since the integrand G is close to a zero function, evaluation of $\widehat{err}$ should be quite accurate even for modest values of m.

The routine actually reports in terms of the relative error err^*. Rewrite the transition from equation 5 to 6 as

$$
\begin{aligned}
\log(L_a + err) &= \log[L_a(1 + err/L_a)] \\
&= \log(L_a) + log(1 + err/L_a) \\
&= \log(L_a) + err^*
\end{aligned}
$$

where L_a is the Laplace approximation to the integrated likelihood.

4 Sparse matrix computations

4.1 Generalized Cholesky Decomposition

The generalized Cholesky decomposition of a symmetric matrix A is

$$A = L'DL$$

where L is lower triangular with 1's on the diagonal and D is diagonal. This decomposition exists for any symmetric matrix A. L is always of full rank, but D may have zeros.

D will be strictly positive if and only if A is a symmetric positive definite matrix, and we can then convert to the usual Cholesky decomposition

$$A = [L\sqrt{D}][\sqrt{D}L'] = U'U$$

where U is upper triangular. If D is non-negative, then A is a symmetric non-negative definite matrix. However, the decomposition exists (with possibly negative elements of D) for any symmetric A.

There are two advantages of this over the Cholesky. The first, and a very minor one, is that the decomposition can be computed without any square root operations. The larger one is that matrices with redundant columns, as often arise in statistical problems with particular codings for dummy variables, become particularly transparent. If column i of A is redundant with columns 1 to i-1, then $D_{ii} = 0$ and $L_{ij} = 0$ for all $i \neq j$. This simple labeling of the redundant columns makes for easy code downstream of the `gchol` routines. The `tolerance` argument in the `gchol` function, and the corresponding `tolerance.chol` argument in `coxph.control` is used to distinguish redundant columns. The threshold is multiplied by the maximum diagonal element of A, if a diagonal element in the decomposition is less than this relative threshold it is set to 0.

Let E be a generalized inverse of D, that is, define E_{ii} to be zero if $D_{ii} = 0$ and as $E_{ii} = 1/D_{ii}$ otherwise. Because of it's structure L is always of full rank, and being triangular it is easy to invert. Then it is easy to show that

$$B = (L^{-1})'EL^{-1}$$

is a generalized inverse of A. That is: $ABA = A$ and $BAB = B$.

The `gchol` routines have been used internally to the `coxph` and `survreg` functions for many years. The new S routines just give an interface to that code. Internally, the return value of `gchol` is a matrix with L below the diagonal and D on the diagonal. Above the diagonal are zeros. At present, I decided not to bother with doing packed storage, although it would be easy to adapt the code called by the bdsmatrix routines.

If

```
      x <- gchol(a)
```

then

`as.matrix(x)`	returns L
`diag(x)`	returns D (as a vector)
`print(x)`	combines L and D, with D on the diagonal
`as.matrix(x, ones=F)`	is the matrix used by `print`
`x@rank`	is the rank of x

Note that `solve(x) = solve(gchol(x))`. This is in line with the current solve function in S, which always returns the result for the original matrix, when presented with a factorization. If `x` is not full rank, the returned value is a generalized inverse. To get the inverse of L, one can use the `full=F` option. One use of this is for transforming data. Suppose that y has correlation $\sigma^2 A$, where A is a general $n \times n$ matrix (kinship for instance). Then

```
> temp <- gchol(A)
> ystar<- solve(temp, y, full=F)
> xstar<- solve(temp, x, full=F)
> fit <- lm(ystar ~ xstar)
```

is a solution to the general regression problem. The vector `ystar` will have correlation σ^2 times the identity, since $L\sqrt{D}$ is a square root of A. The resulting fit corresponds to a linear regression of y on x accounting for the correlation.

4.2 Block Diagonal Symmetric matrices

A major part of the work in creating the kinship library was the formation of the `bdsmatrix` class of objects. A bdsmatrix object has the form

$$\begin{array}{cc} A & B' \\ B & C \end{array}$$

where A is block-diagonal, A and C are symmetric, and B, C may be dense.
Internally, the elements of the object are

- blocksize: vector of block sizes

- blocks: vector containing the blocks, strung together. Together blocksize and blocks represent the block-diagonal A matrix. The blocks component only keeps elements on or below the diagonal, and only those that are within one of the blocks. For instance `bdsmatrix(blocksize=rep(1,n), blocks=rep(1.0,n))` is a representation of the $n \times n$ identity matrix using n instead of n^2 elements of storage.

- rmat: the matrix $(BC)'$. This will have 0 rows if there is no dense portion to the matrix.

- offdiag: the value of off-diagonal elements. This usually arises when some-one has done `y <- exp(bdsmatrix)` or some such. Some of the methods for bdsmatrices, e.g. cholesky decomposition, will punt on these and use as.matrix to convert to a full matrix form.

- .Dim and .Dimnames are the same as they would be for an ordinary matrix.

There are a full compliment of methods for the class, including arithmetic operators, subscripting and matrix multiplication. To a user at the command line, an object of the class might appear to be an ordinary matrix. (At least, that is the intention). One non-obvious issue, however, occurs when the matrix is converted. Assume that `kmat` is a large bdsmatrix object.

```
> xx <- kmat[1:1000, 1:1000]
> xx <- kmat[1000:1, 1:1000]
Problem in kmat[1000:1, 1:1000]: Automatic conversion would create
too large a matrix
```

Why did the first request work and the second one fail? Reordering the rows and/or columns of a bdsmatrix can destroy the block-diagonal structure of the object. If the row and column subscripts are identical, and they are in sorted order, then the result of a subscripting operation will still be block diagonal. If this is not so, then the result will be an ordinary matrix object. To prevent the accidental creation of huge matrices that might exhaust system memory, yet still allow simple interrogation queries such as `kmat[15,21]`, there is a option `bdsmatrixsize` with a default value of 1000. Creation of a matrix larger than about 31 by 31 will fail with the message above. This can be overridden by the user, e.g., `options(bdsmatrixsize=5000)`, to allow the creation of bigger objects. An explicit call to `as.matrix` does not check the size, however, on the assumption that if the user is explicit then they must know what they are doing. (Which is of course just an assumption.)

The `gchol` functions also have methods for bdsmatrix objects, and make use of a key fact — this fact is the actual reason for creating the entire library in the first place — if the block-diagonal portions of the matrix precede the dense portions, as they do in a bdsmatrix object, then the generalized Cholesky decomposition of the matrix is also block-diagonal sparse, with the same blocksize structure as the original. Assume that X were a bdsmatrix of dimension $p + q$, with q sparse and p dense columns. The `rmat` component of the decomposition (which by definition contains the transpose of the lower border of the matrix) will have $p + q$ rows and p columns. The lower p by p portion of `rmat` has zeros below the diagonal, but as with the generalized cholesky of an ordinary matrix, we do not try to save extra space by using sparse storage for this component. Almost all of the savings in space has already been realized by keeping the q by q portion in block-diagonal form.

5 Kinship

We have mentioned using a correlation matrix A that accounts for how subjects are related. One natural choice is the kinship matrix K. Roughly speaking, the elements of K are the amount of genetic material that two subjects would be expected to have in common, by chance. Thus, there are 1's on the diagonal (and for identical twins), 0.5 for parent/child and sib/sib relations, 0.25 for grandparent/grandchild, uncle/niece, and etc. Coefficients other that $1/2^i$ occur if there is inbreeding.

Formally, the elements of K_{ij} are the probability that a gene selected randomly from case i and another selected from case j will be identical by descent (ibd) at some given locus. Thus, the usual diagonal element is 0.5 and the matrix described in the paragraph above is $2K$. See Lange [5] for a fuller explanation, along with a description of other possible relationship matrices.

One consequence of the formal definition is that $K_{ij} = 0$ whenever i and j are from different family trees. Thus K is a symmetric block diagonal matrix. There are 5 principle routines that deal with creation of the kinship matrix: `kinship`, `makekinship`, `makefamid`, `familycheck`, and `bdsmatrix.ibd`.

The `kinship` routine creates a kinship matrix K, using the algorithm of Lange [5]. The algorithm requires that the subjects first be sorted by generation, any partial ordering such that no offspring appears before his/her parents is sufficient. This is accomplished by an internal call to the `kindepth` routine, which assigns a depth of 0 to founders, and $1 + \max$(father's depth, mother's depth) to all others.

The modeling functions `coxme` and `lmekin` currently adjust any input variance matrices to have a diagonal of 1, so it is not necessary to explicitly replace K with $2K$. With inbreeding, the diagonal of $2K$ may become greater than 1. We currently don't know quite what to do in this case, and the routines will fail with a message to that effect.

To create the kinship matrix for a subset of the subjects, it is necessary to first create the full K matrix and then subset it. For instance, if only females were used to create K, then my daughter and my sister would appear to be unrelated. This actually turns out to be more help than hurt in the data processing; a single K can be created once and for all for a study and then used in all subsequent analyses. The routines choose appropriate rows/cols from K automatically based on the dimnames of the matrix, which are normally the individual subject identifiers.

The `kinship` function creates a full n by n matrix. When there are multiple families this matrix has a lot of zeros. The `makekinship` function creates a sparse kinship matrix (of class bdsmatrix) by calling the kinship routine once per family and "gluing" the results together. For the Seller's data, the full kinship matrix is 26050 by 26050 = 678,602,500 elements. But of these, less than 0.00076 are within family and possibly non-zero. (A marry-in with no children can be "in" a family but still have zeros off the diagonal).

The `makefamid` function creates a family-id variable, marking disjoint pedigrees within the data. Subjects who have neither children nor parents in the

pedigree (marry-in for instance) are marked with a 0. The `makekinship` function recognizes a 0 as unrelated, and they become separate 1 by 1 blocks in the matrix. Because `makefamid` uses only (id, father id, mother id), it will pick up on cross-linked families.

For the extended pedigrees of the breast cancer data set, there are a lot of marry-ins. Overall, `makefamid` identifies 8191 of the 26050 participants in the study as singletons, reduces the maximal family size from 286 to 180 and the mean family size from 61.1 to 41.9, and results in a K matrix that has about $1/2$ the number of non-sparse elements. This can result in substantial improvements in processing time. For most data sets, however, the main utility of the routine may be just as a data check. This use is formalized in the `familycheck` function. It's output is a dataframe with columns

- famid: the family id, as entered into the data set

- n : number of subjects in the family

- unrelated: number of them that appear to be unrelated to anyone else in the entire pedigree set. This is usually marry-ins with no children (in the pedigree), and if so are not a problem.

- split : number of unique "new" family ids.

  - if this is 0, it means that no one in this "family" is related to anyone else (not good)

  - 1 = all is well

  - 2+= the family appears to be a set of disjoint trees. Are you missing some of the people?

- join : number of other families that had a unique famid, but are actually joined to this one. One expects 0 of these.

If there are any joins, then an attribute `join` is attached to the dataframe. It will be a matrix with family id as row labels, new-family-id as the columns, and the number of subjects as entries. This allows one to diagnose which families have inter-married. The example below is from a pedigree that had several identifier errors.

```
> checkit<- familycheck(ids2$famid, ids2$gid, ids2$fatherid,
                                        ids2$motherid)
> table(checkit$split)    # should be all 1's
   0   1 2
 112 424 4
# Shows 112 of the "families" were actually isolated individuals,
#  and that 4 of the families actually split into 2.
#  In one case, a mistyped father id caused one child, along with his
#  spouse and children, to be "set adrift" from the connected pedigree.

> table(checkit$join)
```

```
     0 1 2
  531 6 3
#
# There are 6 families with 1 other joined to them (3 pairs), and 3
#  with 2 others added to them (one triplet).
# For instance, a single mistyped father id of someone in family 319,
#  which was by bad luck the id of someone else in family 339,
#  was sufficient to join two groups.
> attr(checkit, 'join')
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
  31   78    0    0    0    0    0    0
  32    3   15    0    0    0    0    0
  33    6    0   12    0    0    0    0
  63    0    0    0   63    0    0    0
  65    0    0    0   17   16    0    0
 122    0    0    0    0    0   16    0
 127    0    0    0    0    0   30    0
 319    0    0    0    0    0    0   20
 339    0    0    0    0    0    0   37
```

The other sparse matrix which is often used in the analysis is an identity by descent (ibd) matrix. Whereas K contains the expected value for an ibd comparison of a randomly selected locus, an ibd matrix contains the actual agreement for a particular locus based on the results of laboratory typing of the locus. In an ideal world, ibd matrices would contain only the 0/1 indicator value, shared or not shared, for each pair. But the results of genetic analysis of a pedigree are rarely so clear. Subjects who are homozygous at the locus, and of course those for whom a genetic sample was not available, give uncertainty to the results, and the matrix contains the expected value of each indicator, given the available data.

The kinship library does not contain any routines for calculating an ibd matrix B. However, most other routines which do so are able to output a data file of (i, j, value) triplets; each asserts that B_{ij}=value. Since B is symmetric and sparse, only the non-zero elements on or below the diagonal are output to the file. The bdsmatrix.ibd routine can read such a file, infer the familial clustering, and create B as a bdsmatrix object. One of the challenging "bookkeeping" tasks in the library was to match up multiple matrices. Routines that calculate ibd matrices, such as solar, often will reorder the subjects within a family; the K matrix from our makekinship function will not exactly match the ibd matrix, yet is in truth computable with it given some row/column shifts. The bdsmatrix.reconcile function, called by coxme and lmekin but NOT by users, accomplishes this. It makes use of the dimnames to accomplish the matching, so the use of a common subject identifier for all matrices is critical to the success of the process.

Here is a small example of using a kinship matrix.

```
> kmat <- makekinship(newfam, bdata$gid, bdata$dadid, bdata$momid)
> class(kmat)
```

```
[1] "bdsmatrix"
> kmat[8192:8197, 8192:8196]
           0041000002 0041000003 0041001700 0041003001 0041003400
0041000002      0.500      0.000      0.125       0.25      0.250
0041000003      0.000      0.500      0.125       0.25      0.250
0041001700      0.125      0.125      0.500       0.25      0.125
0041003001      0.250      0.250      0.250       0.50      0.250
0041003400      0.250      0.250      0.125       0.25      0.500
0041003401      0.250      0.250      0.125       0.25      0.250
```

Note that the row/column order of kmat is *NOT* the order of subjects in the
data set. The first 8191 rows/cols of kmat correspond to the unrelated subjects,
which are treated as families of size 1, then come the larger families one by one.
(The first portion of the matrix is equivalent to the .5 times the identity matrix,
and is not an interesting subset to print out.) The gid variable in this particular
study is a structured character string: the people shown are all from family 004.

Some of the routines above have explicit loops within them, and so cannot be
said to be optimized for the S-Plus environment. However, even on the n=26050
breast cancer set none of them took more than a minute, and they need to be
run only once for a given study.

6   Linear Mixed Effects model

The lmekin function is a mimic of lme that allows the user to input correlation
matrices. As an example, consider a simulated data set dervied from GAW.

```
> dim(adata)
[1] 1497   13
> names(adata)
 [1] "famid"  "id"       "father" "mother" "sex"      "age"      "ef1"      "ef2"
 [9] "q1"      "q2"       "q3"       "q4"       "q5"
>
> kmat <- makekinship(adata$famid, adata$id, adata$father, adata$mother)
> dim(kmat)
[1] 1497 1497
```

To this we will add 2 ibd matrices for the data, generated as output data files
from solar. The variable pedindex is a 2-column array containing the subject
label used by solar in column 1 and the original id in column 2.

```
> temp <-matrix( scan("pedindex.out"), ncol=9, byrow=T)
> pedindex <- temp[,c(1,9)]

> temp <- read.table('ibd.d06g030', row.names=NULL,
                      col.names=c("id1", "id2", "x", "dummy"))
> ibd6.30 <- bdsmatrix.ibd(temp$id1, temp$id2, temp$x,
                           idmap=pedindex)
> temp <- read.table('ibd.d06g090', row.names=NULL,
                      col.names=c("id1", "id2", "x", "dummy"))
```

14

```
> ibd6.90 <- bdsmatrix.ibd(temp$id1, temp$id2, temp$x,
                           idmap=pedindex)
> fit1 <- lmekin(age ~ 1, data=adata, random = ~1|id,
                 varlist=kmat)
> fit1
  Log-likelihood = -4252.912
  n=1000 (497 observations deleted due to missing values)
Fixed effects: age ~ 1
                Value Std. Error  t value Pr(>|t|)
(Intercept) 45.51726  0.6348451 71.69821        0

Random effects:  ~ 1 | id
 Variance list: kmat
                      id      resid
Standard Dev: 5.8600853 16.0306836
  % Variance: 0.1178779  0.8821221

> fit2 <- lmekin(age ~ q4, data=adata, random = ~1|id,
                 varlist=list(kmat, ibd6.90))
> fit2
  Log-likelihood = -4073.227
  n=1000 (497 observations deleted due to missing values)
Fixed effects: age ~ q4
      Value Std. Error    t value   Pr(>|t|)
(Intercept) -3.634481  2.4610657  -1.476792 0.1400469
         q4  2.367681  0.1131428  20.926475 0.0000000

Random effects:  ~ 1 | id
 Variance list: list(kmat, ibd6.90)
                     id1        id2      resid
Standard Dev: 5.9110099 3.8240786 12.5529729
  % Variance: 0.1686778 0.0705973  0.7607249
```

In both cases, the results agree with the same run of solar, with the exception of the log-likelihood, which differs by a constant, and the value of the intercept term in the second model, for which solar gives a value of -3.634 + 2.367* mean(q4). This implies that solar subtracts the mean from each covariate before doing the fit. For the log-likelihood, we have made lmekin consistent with the results of lme.

Even more intesting is a fit with multiple loci

```
> fit3 <- lmekin(age ~ q4, adata, random= ~1|id,
                 varlist=list(kmat, ibd6.30, ibd6.90))

  Log-likelihood = -4073.242
  n=1000 (497 observations deleted due to missing values)
Fixed effects: age ~ q4
      Value Std. Error   t value   Pr(>|t|)
1 -3.635242  2.4611164  -1.47707 0.1399723
2  2.367719  0.1131451  20.92640 0.0000000
```

```
Random effects:  ~ 1 | id
 Variance list: list(kmat, ibd6.30, ibd6.90)
                     id1            id2           id3       resid
Standard Dev: 5.8975380 0.3969544649 3.82594576 12.5528024
  % Variance: 0.1679029 0.0007606731 0.07066336  0.7606731
```

We see that the 6.30 locus adds very little to the fit. (In fact, to avoid numeric issues, the opimizer is internally constrained to have no component smaller than .001 times the residual standard error, so we see that this was on the boundary). Accurate confidence intervals for a parameter can be obtained by profiling the likelihood:

```
theta <- seq(.001, .3, length=20)
ltheta <- theta
for (i in 1:20) {
    tfit <- lmekin(age ~1, adata, random= ~1|id, varlist=list(kmat, ibd6.90),
                   variance=c(0, theta[i]))
    ltheta[i] <- tfit$loglik
    }
plot(theta, ltheta, ylab="Profile likelihood", xlab='Variance')
abline(h=fit2$loglik - qchisq(.95,1)/2)
```

The optional argument `variance=c(0,.02)` will fix the variance for ibd6.90 at 0.02 while optimizing over the remaining parameter.

The `lmekin` function, nevertheless, is very restricted as compared to `lme`, allowing for only a single random effect. It's purpose is not to create a viable competitor to `lme`, and certainly not to challenge broad packages such as `solar`. But since it shares almost all of its data-processing code with `coxme`, it acts as a very comprehensive test for the correctness of the kinship and ibd matrices and our manipulations of them, and greatly increases our confidence in the latter function. A second, but perhaps even more important role is to help make the mental connections for how `coxme` output might be interpreted.

7 Breast cancer data set

7.1 Minnesota breast cancer family study

The aggregation of breast cancer within families has been the focus of investigation for at least a century [2]. There is clear evidence that both genetic and environmental factors are important, but despite literally hundreds of studies, it is estimated that less than 50% of the cancer cases can be accounted for by known risk factors.

The Minnesota Breast Cancer Family Resource is a unique collection of families, first identified by V. Elving Anderson and colleagues at the Dight Institute for Human Genetics [1]. They collected data on 544 sequential breast cancer cases seen at the University of Minnesota between 1944 and 1952 Baseline information on probands, relatives (parents, aunts and uncles, sisters and brothers,

16

sons and daughters) was obtained by interviews, followup letters, and telephone calls, to investigate the issues of parity, genetics and other life factors on breast cancer risk.

This data then sat unused until 1991, when the study was revived by Dr. Tom Sellers. The revised study excluded 58 subjects who were prevalent rather than incident cases, and another 19 who had only 0 or 1 living relative at the time of the first study. Of the remaining 426 families, 10 had no living members, 23 were lost to follow-up, and only 8 refused, leaving a cohort of 426 participating. Sellers et al. [8] have recently extended the followup of all pedigrees through 1995 as part of an ongoing research program, 98.2% of the families agreed to further participation. There are currently 26,050 subjects in the registry, of which 12,699 are female and 13,351 are male. Among the females, the data set has over 435000 person-years of follow up. There a a total of 1063 incident breast cancers: 426/426 of the probands, 376/7090 blood relatives of the probands, and 188/5183 of the females who have married into the families.

There is a wide range of family sizes:

family size	1	4–20	21–50	51-100	> 100
count	8191	72	228	115	11

The 8191 "families" of size 1 in the above table is the count of people, both males and females, that have married into one of the 426 family trees but have not had any offspring. For genetic purposes, these can each be treated as a disconnnected subject, and we do so for efficiency reasons. (If studying shared environmental factors, this simplification would not be possible).

Age is the natural time scale for the baseline risk. The followup for most of the familial members starts at age 18, that for marry-ins to the families begins at the age that they married into the family tree. A portion of this data set was already seen in the section on kinship. The model below includes only `parity`, which is defined to be 0 for women who have not had a child and 1 otherwise. It is a powerful risk variables, conferring an approximately 30% risk reduction. Because they became a part of the data set due to their endpoint, simple inclusion of the probands would bias the analysis, and they are deleted from the computation.

```
> fit1 <- coxph(Surv(startage, endage, cancer) ~ parity,  breast,
        subset=(sex=='F' & proband==0) )
> fit1
         coef exp(coef) se(coef)    z      p
parity0 -0.303    0.739    0.116 -2.62 0.0088

Likelihood ratio test=6.35  on 1 df, p=0.0117
  n=9399 (2876 observations deleted due to missing values)
>fit1$loglik
 -5186.994 -5183.817
```

Several subjects are missing information on either parity (1452), follow-up time/status (2705) or both (1276). Many of these were advanced in age when the study be-

gan, and were not available for the follow-up in 1991. Of the 426 families, 423 are represented in the fit after deletions.

```
> coxme(Surv(startage, endage, cancer) ~ parity, breast,
        random= ~1|famid, subset=(sex=='F'& proband==0))

  n=9399 (2876 observations deleted due to missing values)
  Iterations= 6 63
                    NULL Integrated Penalized
Log-likelihood -5186.994  -5174.865 -5121.984

  Penalized loglik: chisq= 130.02 on 90.59 degrees of freedom, p= 0.0042
 Integrated loglik: chisq= 24.26 on 2 degrees of freedom, p= 5.4e-06

Fixed effects: Surv(startage, endage, cancer) ~ parity0
             coef exp(coef)  se(coef)     z      p
parity0 -0.3021981 0.7391916 0.1172174 -2.58 0.0099

Random effects:  ~ 1 | famid
              famid
Variance: 0.2090332
```

The random effects model based on family shows a moderate familial variance of 0.21, or a standard error of b of about .46. Since $\exp(.46) \approx 1.6$, this says that individual families commonly have a breast cancer risk that is 60% larger or smaller than the norm. One interesting aspect of the random-effects Cox model is that the variances are directly interpretable in this way, without reference to a baseline variance.

However, this is probably not the best model to fit, since it attempts to assign the same "extra" risk to both blood relatives and grafted family members alike. Figure 1 shows the results of the fit for one particular family in the study. (We chose a family with a large enough pedigree to be interesting, but small enough to fit easily on a page). Darkened circles correspond to breast cancer in the females, or prostate cancer in the males. Beneath each subject is the age of event or last follow-up, followed by the $\exp(\hat{b}_i)$, the estimated excess risk for the subject. The female in the upper left corner, diagnosed with breast cancer at age 36, is the proband. Because she was not included in the fit of the random effects model, no coefficient b_i is present. The proband's mother had breast cancer at age 56, four sisters are breast cancer free at ages 88, 60, 78 and 74, but a daughter and a niece are also affected at a fairly young age. Females in this high risk pedigree are all assigned a common risk of 1.47, including a daughter-in-law.

A more reasonable model would assign a separate family id to each marry-in subject (family of size 1). Other options are a single id for all the marry-ins, in which case the frailty level for that single group might be looked upon as the "background Minnesota" level, or to apply either of these options only to those marry-ins with no offspring. The variable `tempid1` below corresponds to the
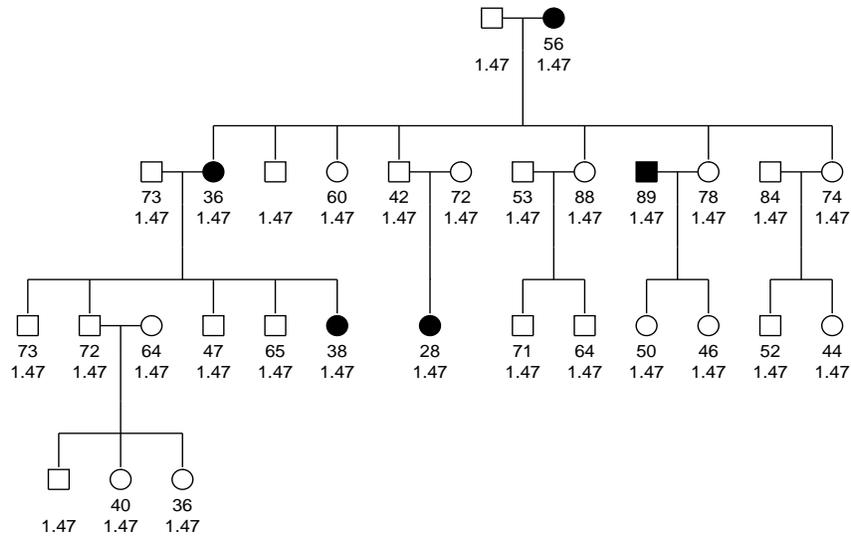
Figure 1: *Shared frailty model, for family 8*

first case: a blood relative receives their family id, and each marry in a unique number > 1000. The variable `tempid2` assigns all marry-ins to family 1000.

```
> tempid1 <- ifelse(breast$bloodrel, breast$famid, 1000 + 1:nrow(breast))
> tempid2 <- ifelse(breast$bloodrel, breast$famid, 1000)
> fit2 <- coxme(Surv(startage, endage, cancer) ~ parity , breast,
                random= ~1|tempid1, subset=(sex=='F' & proband==0))
> fit3 <- coxme(Surv(startage, endage, cancer) ~ parity , breast,
                random= ~1|tempid2, subset=(sex=='F' & proband==0))
> fit2

  n=9399 (2876 observations deleted due to missing values)
  Iterations= 4 50
                    NULL Integrated Penalized
Log-likelihood -5186.994  -5163.226 -5031.745

  Penalized loglik: chisq= 310.5 on 232.65 degrees of freedom, p= 0.00048
 Integrated loglik: chisq= 47.54 on 2 degrees of freedom, p= 4.8e-11

Fixed effects: Surv(startage, endage, cancer) ~ parity0
              coef exp(coef) se(coef)     z     p
parity0 -0.2935372 0.7456215 0.119258 -2.46 0.014

Random effects:  ~ 1 | tempid1
           tempid1
Variance: 0.5063702
```

19

The results using `tempid2` have very similar coefficients: -0.23 for parity and 0.51 for the variance of the random effect, but has a far shorter vector of random effects b — 424 vs 4527 — and a more significant likelihood ratio test, 79.5 versus 47.5. A survey of the random effects associated with the marry-in subjects verifies that the estimated random effects from `fit2` are indeed quite tight.

```
> group <- breast$bloodrel[match(names(fit2$frail), tempid2)]
> table(group)
  Marry-in  Blood
     4104    423
> tapply(fit2$frail, group, quantile)
               0%    25%    50%    75%  100%
Marry-in -0.096 -0.045 -0.026 -0.009 0.506
   Blood -0.802 -0.199 -0.045  0.260 2.039
```

The returned vector of random effects (frailties) will not necessarily be ordered by subject id, and so it is necessary to retrieve them by matching on coefficient names. (They are, in fact, ordered so as to take maximum advantange of the sparseness of the kinship matrix, i.e., an order determined solely by computational considerations). The quantiles for the 4104 marry-ins in the final model range from -.05 to -.01, and those for the 423 blood families in the model range from -0.2 to 0.26. The sum of all 4527 coefficients is constrained to sum to zero, and the random effects structure shrinks all the individual effects towards zero, particularly those for the individual subjects. The amount of shrinkage for a particular frailty coefficient is dependent on the total amount of information in the group (essentially the expected number of events in the group), so large families are shrunk less than small ones, and the individuals most of all. For `fit3`, the random effect for all marry-ins together is estimated at -0.50 or about 40% less than the average for the study as a whole.

A rather simple model, but with surprising results, is to fit a random effect per subject.

```
fit4 <- coxme(Surv(startage, endage, cancer) ~ parity, breast,
         random= ~1|gid, subset=(sex==F & proband==0))
fit4
  n=9399 (2876 observations deleted due to missing values)
  Iterations= 7 68
                     NULL Integrated Penalized
Log-likelihood -5186.994  -5183.815 -5163.111

  Penalized loglik: chisq= 47.77 on 42.26 degrees of freedom, p= 0.26
 Integrated loglik: chisq= 6.36 on 2 degrees of freedom, p= 0.042

Fixed effects: Surv(startage, endage, cancer) ~ parity0
               coef exp(coef) se(coef)     z      p
parity0 -0.3039211 0.7379191 0.116263 -2.61 0.0089

Random effects:   ~ 1 | gid
                 gid
```

```
Variance: 0.06780249
```

It gives a very small random effect of 0.07, and is almost indistinguishable from a model with no random effect at all: compare the integrated log-likelihood of 5183.815 to the value of 5183.817 from fit1! With only one observation per random effect, these models are essentially not identifiable. Technically, it has been shown that with even a single covariate, the models with one frailty term per observation are identifiable in the sense of converging to the correct solution as $n \to \infty$, but in this case it appears that n really does need to be almost infinite. Cox nodels with one independent random effect per observation are not useful in practice.

7.2   Correlated Frailty

The most interesting models for the data involve correlated frailty.

```
> coxme(Surv(startage, endage, cancer) ~ parity0, breast,
              random= ~1|gid, varlist=kmat,
              subset=(sex=='F' & proband==0))

  n=9399 (2876 observations deleted due to missing values)
  Iterations= 4 49
                   NULL Integrated Penalized
Log-likelihood -5187.746   -5172.056 -4922.084

  Penalized loglik: chisq= 531.32 on 471.63 degrees of freedom, p= 0.029
 Integrated loglik: chisq= 31.38 on 2 degrees of freedom, p= 1.5e-07

Fixed effects: Surv(startage, endage, cancer) ~ parity0
              coef exp(coef)  se(coef)     z       p
parity0 -0.3201102  0.726069 0.1221572 -2.62 0.0088

Random effects:   ~ 1 | gid
 Variance list: kmat
                gid
Variance: 0.8714414
```

When we adjust for the structure of the random effect, then the estimated variance of the random effect is quite large: individual risks of 2.5 fold are reasonably common. This model has 9399 random effects, one per subject, and one fixed effect for the parity. The `nlminb` routine is responsible for maximizing the profile likelihood, which is a function only of σ^2, the variance of the random effect. It required 3 iterations, in it's way of counting, but actually required 9 evaluations for different test values of σ (this number is not shown). Each evaluation of the profile likelihood for a fixed σ requires iterative solution of the Cox PPL likelihood equations for $\hat{\beta}$ and $\hat{b}$ as shown in equation (6); a total of 41 Newton-Raphson iterations for the PPL were used "behind the scenes" in this way.
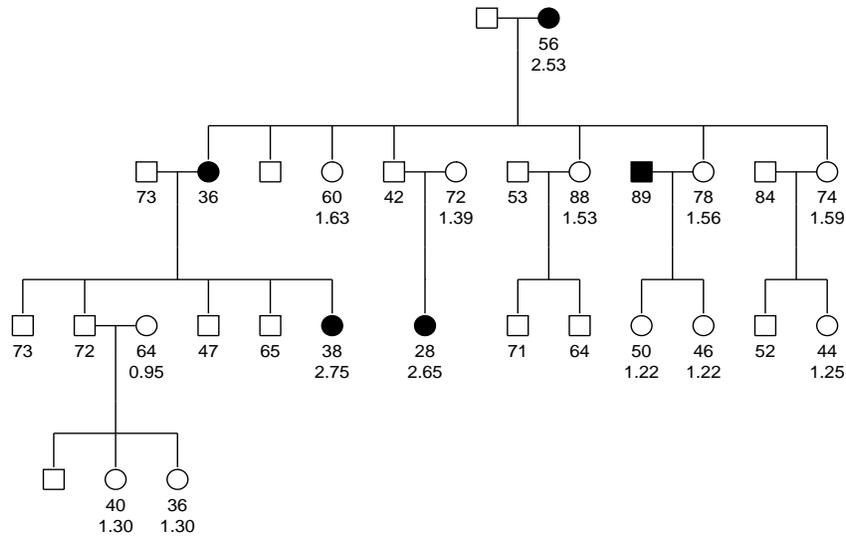
Figure 2: *Correlated random effects fit for family 8*

Figure 2 displays the fit for family 8. The 88 year old sister has a smaller estimated genetic random effect than the 60 year old sister; with more years of follow-up there is stronger evidence that she did not inherit as large a portion of the genetic risk. The unaffected neice at age 44 is genetically further from the affecteds and has a lower estimated risk. Note also that the two females who married into the family, one with and one without an affected daughter, have very different risks than the blood relatives.

The figure was drawn by the following code

```
> fam8 <- breast[famid=='008', ]
> ped8 <- pedigree(fam8$gid, fam8$dadid, fam8$momid, sex=fam8$sex,
                   affected=(!is.na(fam8$cancer) & fam8$cancer==1))
> ped8$hints[10,1] <- 1.5
> risk8 <- fit4$frail[match(fam8$gid, names(fit4$frail))]
> risk8 <- ifelse(is.na(risk8), '', format(round(exp(risk8),2)))
> age8  <- ifelse(is.na(fam8$endage), "", round(fam8$endage))
> plot(ped8, id= paste(age8, risk8, sep="\n"))
```

The hints are used to adjust the order of siblings in line 2 of the plot, and was not strictly necessary. (More on this in a later section). The order of the coefficients in `fit4$frail` is determined by the `coxme` program itself, using the ordering that is simplest for indexing the bdsmatrix `kmat`, so it is necessary to retrieve coefficients by name. The coefficients and the age are then formatted in a nice way, and pasted together to form the label for each node of the genetic tree. (The word 'backslash' in the above should be the character

, but latex and I have not yet agreed on how to get it to print what I want within an example).

We can also fit a model with a more general random effect per subject:

```
> coxme(Surv(startage, endage, cancer) ~ parity2, breast,
                random= ~1|gid, varlist=list(kmat, bdsI),
                subset=(sex=='F' & proband==0))

  n=9399 (2876 observations deleted due to missing values)
  Iterations= 4 65
                    NULL Integrated Penalized
Log-likelihood -5186.994  -5170.824 -4896.216

  Penalized loglik: chisq= 581.56 on 516.1 degrees of freedom, p= 0.024
 Integrated loglik: chisq= 32.34 on 3 degrees of freedom, p= 4.4e-07

Fixed effects: Surv(startage, endage, cancer) ~ parity0
              coef exp(coef)  se(coef)     z       p
parity0 -0.3219566 0.7247296 0.1228136 -2.62 0.0088

Random effects:  ~ 1 | gid
  Variance list: list(kmat, bdsI)
              gid1       gid2
Variance: 0.909301 0.0520675
```

This again fits a model with one random effect per subject, but a covariance matrix $b \sim N(0, \sigma_1^2 K + \sigma_2^2 I)$. This is equivalent to the sum of two independent random effects, one correlated according to the kinship matrix and the other an independent effect per subject. Again, the addition of an unconstrained per subject random effect does not add much; the likelihood increases by only 1.2 for 1 extra degree of freedom. This is in contrast to the linear model, where a residual variance term is expected and important.

7.3 Connections between breast and prostate cancer

Within the MBRFS, a substudy was conducted to examine the question of possible common genetic factors between breast and prostate cancer. For 60 high risk families (4 or more breast cancers) and a sample of 81 of the 138 lowest risk families (no breast cancers beyond the original proband), all male relatives over the age of 40 were assessed for prostate cancer using a questionairre.

Three models were considered:

1. Common genes: each person's risk of cancer depends on that of both male and female relatives. This makes sense if general defect-repair mechanisms are responsible for both cancers, mechanisms that would effect both genders.

2. Separate genes: a female's risk of cancer is linked to the risk of her female relatives, a male's is linked to that of his male relatives, but there is no

	Variance			
	M/F	F/F	M/M	$\mathcal{L}$
Common	0.68	0.68	0.68	46.4
Separate	–	0.98	0.71	51.9
Combined	0.20	0.92	0.70	52.5

Table 1: *Results for the breast-prostate models. Shown are the variances of the random effects, along with the likelihood ratio $\mathcal{L}$ for each model with the null.*

interdependency. This makes sense if the cancers are primarily hormone driven, with different genes at risk for estrogen and androgen damage.

3. Combined: some risk of each type exists.

To examine these, consider a partitioned kinship matrix where the variance strucure is $\sigma_1^2 k_{ij}$ for i, j both female, $\sigma_2^2 k_{ij}$ for i, j both male, and $\sigma_3^2 k_{ij}$ when i, j differ in gender, where k are the usual elements of the kinship matrix. The common gene model corresponds to $\sigma_1 = \sigma_2 = \sigma_3$, the separate gene model to $\sigma_3 = 0$, and the combined model to unconstrained variances. In practice, the code requires the creation of three variant matrices: `kmat.f` is a version of the kinship matrix where only female-female elements are non-zero, `kmat.m` similarly for male-male and `kmat.mf` for male-female intersections. The code below fits model 2:

```
> coxme(Surv(startage, endage, cancer) ~ parity + strata(sex),
                  subset=(proband==0),
            random=~1|id, varlist=list(kmat.f, kmat.m))
```

Table 1 shows results for the 3 models. The variance coefficients for model 2 are identical to doing separate fits of the males and the females; a joint fit also gives the overall partial likelihood. We see that the separate gene model is significantly better than a shared gene hypothesis, that the familial effects for both breast and prostate cancer are quite large, and that the combined model is somewhat, but not significantly, better than a separate genes hypothesis. For a woman, knowing her sister's status is much more useful than knowing that of male relatives.

8   Random treatment effect

The development of `coxme` was focused on genetic models, where each subject has their own random effect. It can also be used for simpler cases, albeit with more work involved than the usual `lme` call. The data illustrated below is from an cancer trial in the EORTC; the example is courtesy of Jose Cortinas.

There are 37 enrolling centers, with a single treatment variable. Fitting a random effect per center is easy:

```
> fit0 <- coxph(Surv(y, uncens) ~x, data1)    # No random effect
> fit1 <- coxme(Surv(y, uncens) ~ x, data1, random= ~1|centers)
> fit1
Cox mixed-effects model fit by maximum likelihood
  Data: data1
  n= 2323
  Iterations= 11 137
                   NULL Integrated Penalized
Log-likelihood -10638.71   -10521.44 -10489.41

  Penalized loglik: chisq= 298.6 on 31.47 degrees of freedom, p= 0
 Integrated loglik: chisq= 234.55 on 2 degrees of freedom, p= 0

Fixed effects: Surv(y, uncens) ~ x
       coef exp(coef)    se(coef)       z p
x 0.7115935  2.037235 0.06428942 11.07 0

Random effects:  ~ 1 | centers
             centers
Variance: 0.1404855

> fit0$log
-10638.71 -10585.88
```

By comparing the fit with and without the random effect, we get a test statistic of $2(10585.8 - 10521.44) = 129$ on 1 degree of freedom. The center effect is highly significant.

Now, we would also like to add a random treatment effect, nested within center. Eventually this also will be simple to fit using $\sim$ x|centers as the random effect formula. We can also fit this with the current offering by constructing a single random effect with the appropriate covariance structure. The model has two random effects, a group effect b_{j1} and a treatment effect xb_{j2} where j is the enrollment center

$$
\begin{aligned}
b_{.1} &\sim N(0, \sigma_1^2) \\
b_{.2} &\sim N(0, \sigma_2^2)
\end{aligned}
$$

The combined random effect $c \equiv b_{j1} + xb_{j2}$ has a variance matrix of the following form

$$
A = \begin{pmatrix}
\sigma_1^2 & \sigma_1^2 & 0 & 0 & \cdots \\
\sigma_1^2 & \sigma_1^2 + \sigma_2^2 & 0 & 0 & \cdots \\
0 & 0 & \sigma_1^2 & \sigma_1^2 & \cdots \\
0 & 0 & \sigma_1^2 & \sigma_1^2 + \sigma_2^2 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
$$

The rows/columns correspond to group 1/treatment 0, group 1/treatment 1, 2/0, 2/1, etc. Essentially, since treatment is a 0/1 variable we are able to view treatment as a factor nested within center. To fit the model we need to construct two variance matrices such that $A = \sigma_1^2 V_1 + \sigma_2^2 V_2$.

25

```
> ugroup <- paste(rep(1:37, each=2), rep(0:1, 37), sep='/') #unique groups
> mat1 <- bdsmatrix(rep(c(1,1,1,1), 37), blocksize=rep(2,37),
                    dimnames=list(ugroup,ugroup))

> mat2 <- bdsmatrix(rep(c(0,0,0,1), 37), blocksize=rep(2,37),
                    dimnames=list(ugroup,ugroup))

> group <- paste(data1$centers, data1$x, sep='/')
> fit2 <- coxme(Surv(y, uncens) ~x, data1,
                random= ~1|group, varlist=list(mat1, mat2),
                rescale=F, pdcheck=F)
> fit2
Cox mixed-effects model fit by maximum likelihood
  Data: data1
  n= 2323
  Iterations= 10 165
                     NULL Integrated Penalized
Log-likelihood -10638.71      -10516 -10484.44

  Penalized loglik: chisq= 308.54 on 35.62 degrees of freedom, p= 0
  Integrated loglik: chisq= 245.42 on 3 degrees of freedom, p= 0

Fixed effects: Surv(y, uncens) ~ x
        coef exp(coef)   se(coef)    z p
x 0.7346435  2.084739 0.07511273 9.78 0

Random effects:   ~ 1 | group
 Variance list: list(mat1, mat2)
              group1      group2
Variance: 0.04925338 0.07654925
```

Comparing this to fit1, we have an significantly better fit, $(245.2 - 234.6) = 10.6$ on 1 degree of freedom. We needed to set `pdcheck=F` to bypass the internal test that both `mat1` and `mat2` are positive definite, since the second matrix is not. The `rescale=F` argument stops an annoying warning message that `mat2` does not have a constant diagonal.

Finally, we might wish to have a correlation between the random effects for intercept and slope. In this case the off-diagonal elements of each block of the variance matrix are $\mathrm{cov}(b_{j1}, b_{j1} + b_{j2}) = \sigma_1^2 + \sigma_{12}$ and the lower right element is $\mathrm{var}(b_{j1} + b_{j2}) = \sigma_1^2 + \sigma_2^2 + \sigma_{12}$. We need a third matrix to carry the covariance term, giving

```
> mat3 <-  bdsmatrix(rep(c(0,1,1,1), 37), blocksize=rep(2,37),
                     dimnames=list(ugroup,ugroup))
> fit3 <- coxme(Surv(y, uncens) ~x, data1,
                random= ~1|group, varlist=list(mat1, mat2, mat3),
                rescale=F, pdcheck=F, vinit=c(.04, .12, .02))
> fit3
Cox mixed-effects model fit by maximum likelihood
  Data: data1
```

```
   n= 2323
   Iterations= 7 169
                      NULL Integrated Penalized
Log-likelihood -10638.71  -10515.54 -10486.59

  Penalized loglik: chisq= 304.23 on 30.09 degrees of freedom, p= 0
 Integrated loglik: chisq= 246.33 on 4 degrees of freedom, p= 0

Fixed effects: Surv(y, uncens) ~ x
       coef exp(coef)   se(coef)     z p
x 0.7142582  2.042671 0.06660611 10.72 0

Random effects:  ~ 1 | group
 Variance list: list(mat1, mat2, mat3)
             group1      group2      group3
Variance: 0.02482083 0.07976837 0.03000053
```

By default the routine uses `mat1 + mat2 + mat3` as a starting estimate for iteration. However, in this case that particular combination is a singular matrix, so the routine needs a little more help as supplied by the `vinit` argument.

Because treatment is a 0/1 variable, one should also be able to fit this as a simple nested model.

```
> fit4 <- coxme(Surv(y, uncens) ~x, data=data1, random= ~1|centers/x)
> fit4
Cox mixed-effects model fit by maximum likelihood
  Data: data1
  n= 2323
  Iterations= 11 165
                      NULL Integrated Penalized
Log-likelihood -10638.71  -10517.57 -10483.22

  Penalized loglik: chisq= 310.99 on 39.77 degrees of freedom, p= 0
 Integrated loglik: chisq= 242.29 on 3 degrees of freedom, p= 0

Fixed effects: Surv(y, uncens) ~ x
       coef exp(coef)   se(coef)    z p
x 0.7434213  2.103119 0.08381642 8.87 0

Random effects:  ~ 1 | group
 Variance list: list(mat1, mat2b)
             group1      group2
Variance: 0.06842845 0.04462965
```

This differs substantially from fit2. Why? Let c, d, e, f be random effects, and consider two subjects from center 3. The predicted risk score for the subjects is

	Fit 2	Fit 4
x=0	$0 + c_3$	$0 + e_3 + f_{30}$
x=1	$\beta + c_3 + d_3$	$\beta + e_3 + f_{31}$

Here c and e are the random center effects and d and f are the random treatment effects under the two models, respectively. Fit 2 can be written in terms of the coefficients of fit 4: $c = e + f_{.0}$, $d = f_{.1} - f_{.0}$. To be equivalent to fit 4, then, we would have $\sigma_c^2 = \sigma_e^2 + \sigma_f^2$, $\sigma_d^2 = 2\sigma_f^2$ and $\sigma_{cd} = -\sigma_f^2$. An uncorrelated fit on one scale is not the same as an uncorrelated one on the other scale. This fit can be verified

```
> temp <- fit4$coef$random
> fit3c <- coxme(Surv(y, uncens) ~x, data1,
            random= ~1|group, varlist=list(mat1, mat2, mat3),
            rescale=F, pdcheck=F, lower=c(0,0,-100),
            variance=c(temp[1]+temp[2], 2*temp[2], -temp[2]))
> fit3c$log
      NULL Integrated Penalized
 -10638.71    -10517.9 -10477.83
```

The main point of this section is that nearly any model can be fit "by hand" if need by by constructing the appropriate variance matrix list for a combined random effect.

9 Questions and Conclusion

The methods presented above have been very useful in our assessment of breast cancer risk, factors affecting breast density, and other aspects of the research study. A random effects Cox model, with Gaussian effects, has some clear advantages:

- The Cox model is very familiar, and investigators feel comfortable in interpreting the results

- The counting process (start, stop] notation available in the model allows us to use time-dependent covariates, alternate time scales, and multiple events/subject data in a well understood way, at least from the view of setting up the data and running the model.

- Gaussian random effects allow for efficient analysis of large genetic correlations

Nevertheless, there are a large number of unanswered questions. A primary one is biological: the Cox model implicitly assumes that what one inherits, as the unmeasured genetic effect, is a *rate*. Subject x has 1.3 times the risk of cancer as subject y after controlling for covariates, every day, unchanging, forever. Other models can easily be argued for.

Statistically, our own largest question relates to the required amount of information. How much data is needed to reliably estimate the variances? We have already commented that 1 obs/effect is far too little for an unstructured model. For the correlated frailty model on the breast cancer data, the profile likelihood for σ has about the same relative width as the one for the fixed parity

effect, so in this case we clearly have enough. Unfortunately, we have more examples of the first kind than the second, but this represents fairly limited experience.

Only time and experience may answer some of these.

A Sparse terms and factors

The main efforts at efficiency in the `coxme` routine have focused on random effects that are discrete, that is, the grouping variables. In the kinship models, in particular, b is of length n, the number of observations, or sometimes even longer if there are multiple random terms.

The first step with such variables is to maintain their simplicity.

1. They are coded in the Z matrix as having one coefficient for each level of the variable. The usual contrast issues (Helmert vs treatment vs ordered) are completely ignored. This is also the correct thing to do mathematically; because of the penalty the natural constraint on these terms is the sum constraint $b'Ab = 0$, where A is the inverse of the variance matrix of b, similar to the old $\sum \alpha_i = 0$ constraint of one-way ANOVA.

2. The dummy variables are not centered and scaled, as is done for the ordinary X variables.

3. Thus, the matrix of dummy variables Z never needs to be formed at all. A single vector containing the group number of each observation is passed to the C code. If there are multiple random effects that are grouping variables, then this a matrix with one column per random effect.

A second speedup takes place in the code. The program needs to keep a running estimate of two terms, $E(Z)$ and $E(Z^2)$, which are used in updating the score vector and Hessian matrix at each death. The vector of sums $a_j \equiv \sum w_i Z_{ij}$ and the denominator $\sum w_i$ are kept separately. As each subject enters or leaves the current risk set, we know that their set of indicator variables Z_i is 1 for the group they are a part of, group k say, and 0 for all other columns. Thus only a_k needs to be updated. Secondly, we know that $\sum w_i Z_{ij}^2 = \sum w_i Z_{ij}$, since these are 0/1 variables, and that $\sum w_i Z_{ij} Z_{ik} = 0$ for $j \neq k$. Thus, for these variables, a is sufficient.

For the X variables, we keep both the vector of sums a and the matrix of cross-products, the latter as a matrix C. If there are p fixed effects and q random effects, then C has dimensions p by $q + p$; the cross product terms between X and Z are retained. But again, this cross-product matrix can be updated very quickly, since only one of the q columns corresponding to the random effects will change for any subject.

The overall score vector u for the random effects is equal to the penalty term plus a sum over the deaths of

$$Z_i - E[Z(t)],$$

the Z vector of the subject who experienced the event minus the average over those at risk at that time. The first term is again fast to compute; add 1 to u_k, where k is the group to which the subject who had the event belongs. For the second term, the entire length q vector of means must be subtracted from u, which is q divisions (a[j]/denominator) and q subtractions. As discussed further below, this simple update is, surprisingly, the critical computational bottleneck in the code.

The computation of the overall Hessian matrix is a little more subtle. For b there is again a penalty term plus a Cox PL contribution at each death. But although the penalty term may be block diagonal, the PL part is not. The contribution of the PL to the overall Hessian consists, at each event, of a multinomial variance matrix with diagonal elements of $p_j(1 - p_j)$ and off-diagonals $-p_j p_k$ where p_j is the weighted proportion of subjects in each level of the factor, at the time of the event.

If the penalty matrix is block-diagonal, then the `coxme` code retains as an *approximate* Hessian only the block-diagonal portion of the full H, the other parts of H are thrown away (and not computed). This is equivalent to setting them to zero. How big an impact does this have? Assume that the random effect has q levels, with approximately the same number of subjects in each group. Then the diagonal elements of H are $O(1/q) + $ penalty, the ignored off-diagonal ones are $O(1/q^2)$, and the retained off-diagonal elements are $O(1/q^2) + $ penalty. We see that if q is large, the approximation is likely to be good. In fact it works best when we need it most. The approximation is also likely to work well when the penalty is large, i.e., when the estimated variance of the random effect is small.

The approximation can still fail when q is large, however, if one or more of the groups contains a large fraction of the data. An example of this was the breast cancer analysis where we treated all of the marry-in subjects to be from one large family "Minnesota"; $q = 4126$ but over half the observations were in a single one of the levels. The off-diagonal elements with respect to this column are of a similar order to the diagonal ones, and the error in the approximate Hessian is such that the Newton-Raphson updates do not converge.

The `sparse` argument of the program is intended to deal with this. If one or more levels of one of the grouping variable are considered "not sparse", then the indicator variable for that level is assigned to the dense part of the penalty matrix, that is, the `rmat` portion of the bdsmatrix. The calculation of the a vector and C matrix are still fast, but all covariance information for this level is retained. (If the variance matrix for a grouping variable is supplied by the user through the `varlist` argument, then deciding what is and is not sparse is their problem; encapsulated in the structure of the supplied bdsmatrix).

The vector of random effects is then laid out in the following order: grouping variables for which the variance matrix is kept in block-diagonal form (sparse), non-sparse grouping variables, and then other penalized terms (such as random slopes). The computer code has to keep track of both the cutoff between factor/non-factor random terms (which controls the fast method for updating a), and sparse/non-sparse locations in the penalty matrix. This ordering is ev-

ident in the returned vector of coefficients for the random effects. Users who want to make use of the coefficients b will usually have to explicitly look at their names; the final order cannot be inferred as the sorted order of their levels as with an ordinary factor variable.

The returned variance matrix for the coefficients is in the order (b, β), with the random coefficients in the order described above followed by the fixed effects coefficients. It will also be a bdsmatrix object. Now, the inverse of a block-diagonal matrix is itself block diagonal, but the inverse of a bdsmatrix object that contains an `rmat` component, as the `coxme` models will if there are any fixed effects(so that β is not null), is not block diagonal. What is returned by the function is the block-diagonal portion of this inverse. The covariance elements between elements of b that are omitted are not equal to zero, so the result is incorrect in this aspect, but those off-diagonal element that are included are computed correctly.

B Computation time

Given the values of the variance terms, the solution for (β, b) involves minimization of the the Cox PPL. This "inner loop" of the program is programmed in C, and is very like the PL computation for an ordinary Cox model, with the addition of extra penalty terms to the integrated likelihood $\mathcal{L}$, the score vector, and H.

The coxme function uses nlminb to find the minimum of $\mathcal{L}$ as a function of the variances, with one call to the PPL computation for each trial value of nlminb. Because there may be many dozens of evaluations of the Cox PPL, every attempt has been made to make this inner routine fast. The coxfit6a routine is called once at the beginning to upload the data into main memory and do any computations that are common to all calls, such as subtracting the mean from all covariates and computing various indices. At the last, the coxfit6c routine is called to return various ancillary aspects of the computation, those which are needed in the output structure but not directly for computation of $\mathcal{L}$. The coxfit6b routine, which does the actual computation of $\mathcal{L}$ for a given penalty matrix, thus has as few arguments as possible.

Nevertheless, it is in this routine that all the time is spent. Let n be the number of observations, b_1 the total number of elements in the sparse representation of Σ, b_2 the average number of elements in each column of that representation, p the number of elements of β, q the number of sparse terms (usually 1), f the number of frailties (the length of b), and d be the number of deaths. For a simple random effect such as `1|famid`, both q and b_2 will be 1 and b_1 and f would be the number of families.

The inner loop currently uses a fixed number of iterations, set to 4, rather than iteration to convergence, and a single fixed starting estimate. This turns out to be very important, as otherwise the likelihood surface appears to have discontinuities (in the eyes of nlminb) simply due to variable stopping points. I am almost certain that this number could be reduced to 3 or perhaps even

2, but that has yet to be tested. The default number of inner iterations is the `iter.inner` parameter of `coxme.control`.

For any given call to coxfit6b, each of the following is done once per iteration

- calculate the risk score: $O(n[p + q])$

- update mean and variance: $O(n[p + q + p(p + q)])$

- update score vector: $O(d[p + f])$

- update the Hessian: $O(d[p(p + f) + b_1])$

- update parameters: $O(p^2(p + f) + b_1 b_2)$

This is for the Efron approximation, for the Breslow approximation replace d with the number of unique death times. This is the only routine in the survival suite for which the Breslow and Efron approximations differ in compute time, and interestingly enough the Breslow is faster in exactly the situation where one would be statistically uncomfortable in using it. However, it may be a useful thing to do in the preliminary analyses of a data set.

The difference between q and f in these expressions is not trivial. For the breast data, we profiled the computation of coxfit6b for a single variance value of 0.1, and a per-subject random effect `1|id`. In this case $p = q = 1$, $f = 9820$, and over 99% of the routine's time was spent on the score vector and the Hessian, split about 1:2 between them. The data set has 1034 events but only 125 unique event times — due to the use of surrogate information for many of the events only whole ages were used. The run time for coxme with a fixed variance was 30 seconds using the Efron approximation and 5.6 seconds with the Breslow. For an iterated solution, where comparatively less of the time is spent in the initial setup, the ratio would be higher. It is not uncommon for the `nlminb` code to make 40–50 calls to the inner routine.

C Faster code

While discussion the above issues with a colleague, by way of explaining why the code could not possibly be made any faster, a realization occurred that the algorithm could indeed be improved. Let us start with the score statistic, noting that

1. For a factor variable, $a_i(t)$ changes only rarely as the program loops over time. For a correlated frailty model with 1 random effect per subject, it will change only once – when that subject enters the risk set. (The program sums from latest event time to earliest).

2. The computationally expensive part of the score statistic is a sum over the deaths $a_i(t)/d(t)$, where $d(t)$ is the weighted number at risk at the death time. This is organized as an outer loop over the death times, and an inner one over i.

3. To speed this up, factor out common values of $a_i(t)$. Assume that $t_1, t_2, \ldots, t_d$ are the d death times and that $a_3(t)$ changes only at t_5. We can write this as $a_3(t_1)(1/d(t_1) + \ldots + 1/d(t_4))$ plus $a_3(t_5)(1/d(t_5) + \ldots + 1/d(t_d))$. Keep a vector of lagged denominators `dlag`, of the same length as a and initialized to zero, along with a running total `dtot`$= \sum 1/d(t)$. Whenever a_i changes

   - subtract (old a_i) (`dtot` - `dlag`$_i$) from the ith element of the score vector

   - update a_i and set `dlag`$_i = $ `dtot`.

4. The loop over all f factor elements of u is done only twice — when setting to zero at the head of the loop, and a final summation at the end of the loop — as opposed to d evaluations, one per death.

The score vector calculation is now $O(dp + 2f + n)$, compared to the earlier $O(dp + df])$. This is a huge advantage for something like the breast study, where $d > 1000$ and $f = n \approx 10000$. For studies where $f << n$, such as a random institutional effect with only a few enrolling centers, it would be a disadvantage to calculate things this way. This is exactly the situation, however, where sparse matrix routines would not be used for the Hessian, so it will suffice to use the "new" trick only for the sparse factor terms.

What about the sparse part of the Hessian matrix H? The diagonal elements of H are a sum over the deaths of $a_i(t)/d(t) - [a_i(t)/d(t)]^2$. The same lagged computation as before can be used, with a second vector containing the lagged sum of squared denominators. Off the diagonal H is a sum of $-a_i(t)a_j(t)/d(t)^2$. This requires a sparse matrix of lagged sums of the same size as the sparse representation of H. The computation time for this will be $O(nb_2)$, where b_2 is the average number of rows in a block of the sparse matrix; the usual computation approach had $O(dfb_2)$ for this portion.

Third, we need to think about the sparse/dense portion of H, the interactions between a sparse factor term and a non-sparse term. The elements of the sum are $C_{ij}(t)/d(t) - (a_i(t)a_j(t))/d(t)^2$, where i is one of the sparse variables and j a dense one. As noted earlier, the ith column of C changes only when a_i changes, so the update step is similar to that for the score vector u. For the second term, we re-write it as $a_i[a_j(t)/d(t)^2]$, thinking of the parenthesized term again as just a weight! These lagged weights require scratch space of the same size as the sparse/dense portion of H.

The last issue to deal with is the computations for the Efron approximation. If there are k tied deaths at a given time, then the columns for those factor variables that had a change *at that death time* (there must be $\leq k$ of them) are computed in the same way as the non-sparse variables. That is, the update for that particular death time is computed for those rows/columns, and then the lagged denominators are updated.

D Determinants and trace

For reference on this, see Searle [7].

- For any matrix $|AB| = |A||B|$, where $|A|$ is the determinant of A. Thus $|A^{-1}| = 1/|A|$.

- For orthonormal A, $|A| = \pm 1$.

- For a triangular matrix $|A| = \prod A_{ii}$.

- For idempotent A, $|A| = 0$ or 1 (since $A^2 = A$).

Now, if A is symmetric positive-definite, then $A = LDL'$ where L is lower triangular with 1s on the diagonal and D is diagonal. In this case $\log|A| = \sum_i \log(D_{ii})$, and $\log|A^{-1}| = -\sum_i \log(D_{ii})$.

Note that the product rule for matrices is

$$\frac{\partial AB}{\partial \theta} = \frac{\partial A}{\partial \theta}B + A\frac{\partial B}{\partial \theta} \ .$$

(Hint, look at each element of the matrix product individually). Hence, using $AA^{-1} = I$ we can derive that

$$\frac{\partial A^{-1}}{\partial \theta} = A^{-1}\frac{\partial A}{\partial \theta}A^{-1}.$$

The derivative of the determinant is

$$\frac{\partial |A|}{\partial \theta} = |A| \operatorname{trace}\left(A^{-1}\frac{\partial A}{\partial \theta}\right) \ .$$

This can be used to create an estimating equation for the frailty model. Ripatti and Palmgren ([6]) use this to estimate the variance of the random effects. However, because the profile likelihood function for these parameters is often very skewed, we do not consider such standard error estimates to be very useful.

We do need the determinant of the Hessian to compute the integrated likelihood, however. Consider the partitioned Hessian matrix

$$H = \left(\begin{array}{cc} H_1 & H_2 \\ H_2' & H_3 \end{array}\right)$$

with the upper block being $q \times q$ (the number of random terms) and the lower right block $p \times p$ (the fixed effects). We can similarly partition the Cholesky decomposition as

$$L = \left(\begin{array}{cc} L_1 & 0 \\ S & L_2 \end{array}\right), \ D = \left(\begin{array}{cc} D_1 & 0 \\ 0 & D_1 \end{array}\right)$$

where D_1 and D_2 are diagonal, L_1 and L_2 are lower triangular with 1's on the diagonal, and S will be a dense $p \times q$ matrix. Then $H_1 = L_1 D_1 L_1'$ and $\log|H_1| =$

$\sum_{i=1}^{q} \log(D_{ii})$. Thus the determinant $|H_1|$, needed for the ML calculation, can be read off of the Cholesky decomposition of H, which at hand already in hand as part of the Newton-Raphson iteration step.

For an 'REML' esimate we need $|(H^{-1})_1|$, the upper right corner of the inverse. But getting the corner of the inverse is not as simple as the inverse of the corner. The inverse of L can be written as

$$L^{-1} = \begin{pmatrix} U_1 & T \\ 0 & U_2 \end{pmatrix}'$$

where U_1 and U_2 are upper triangular matrices, $U_1' = L_1^{-1}$, $U_2' = L_2^{-1}$, and $T = -U_1 S' U_2$. This can be verified by simply multiplying out the product. Similarly

$$\begin{aligned} H^{-1} &= \begin{pmatrix} U_1 & T \\ 0 & U_2 \end{pmatrix} \begin{pmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{pmatrix} \begin{pmatrix} U_1' & 0 \\ T' & U_2' \end{pmatrix} \\ &= \begin{pmatrix} U_1 D_1^{-1} U_1' + T D_2^{-1} T' & T D_2^{-1} U_2' \\ U_2 D_2^{-1} T' & U_2 D_2^{-1} U_2' \end{pmatrix} \end{aligned}$$

We see that $(H^{-1})_1 = [L_1 D_1 L_1']^{-1} + T D_2^{-1} T'$, so the determinant of $(H^{-1})_1$ cannot be simply read off of the Cholesky decomposition of H. This last is what makes the REML estimate more work to compute.

But, with a little clever matrix work, there is a computationally reasonable way out. The first equation below is from Henderson and Searle [3], the second is a standard result for partitioned matrices.

$$\begin{aligned} \begin{vmatrix} A & B \\ B' & C \end{vmatrix} &= |C||A - B'C^{-1}B| \\ \begin{pmatrix} A & B \\ B' & C \end{pmatrix}^{-1} &= \begin{pmatrix} D & -DBc^{-1} \\ -C^{-1}B'D & C^{-1} + C^{-1}B'DBC^{-1} \end{pmatrix} \\ \text{where } D &\equiv \left(A - B'C^{-1}B \right)^{-1} \end{aligned}$$

Comparing the upper corner of the inverse to the terms in the partitioned determinant, we see that $|(H^{-1})_1| = |H_3|/|H|$. This requires a second Cholesky decomposition, but of a fairly small $p \times p$ matrix H_3.

E   Manual pages

E.1   align.pedigree

Given a pedigree, this function creates helper matrices that describe the layout of a plot of the pedigree.

```
align.pedigree(ped, packed=T, hints=ped$hints, width=6, align=T)
```

Required arguments

ped a pedigree object

Optional arguments

packed should the pedigree be compressed, i.e., to allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.

hints two column hints matrix. The first column determines the relative order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The second column contains spouse information, e.g., if hints[2,6] = 17, then subject number 17 of the pedigree is a spouse of number 2, and is preferentially plotted to the right of number 2. Negative numbers plot the spouse preferentially to the left.

width for a packed output, the minimum width

align should iterations of the 'springs' algorithm be used to improve the plotted output. If True, a default number of iterations is used. If numeric, this specifies the number of iterations.

Return value: a structure with components:

n a vector giving the number of subjects on each horizonal level of the plot

nid a matrix with one row for each level, giving the numeric id of each subject plotted. (An value of 17 means the 17th subject in the pedigree).

pos a matrix giving the horizontal position of each plot point

fam a matrix giving the family id of each plot point. A value of "3" would mean that the two subjects in positions 3 and 4, in the row above, are this subject's parents.

spouse a matrix with values 1= subject plotted to the immediate right is a spouse, 2= subject plotted to the immediate right is an inbred spouse, 0 = not a spouse

twins optional matrix which will only be present if the pedigree contains twins. It has values 1= sibling to the right is a monozygotic twin, 2= sibling to the right is a dizygotic twin, 3= sibling to the right is a twin of unknown zygosity, 0 = not a twin

This is an internal routine, used almost exclusively by *plot.pedigree*. The subservient functions *alignped1*, *alignped2*, *alignped3*, and *alignped4* contain the bulk of the computation.

E.2   autohint

A pedigree structure can contain a *hints* matrix which helps to reorder the pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine is called by the *pedigree* function to create an intial hints matrix.

```
autohint(ped)
```

Required arguments

ped a pedigree structure

Return value

 a two column hints matrix

This routine would not normally be called by a user. It moves children within families, so that marriages are on the "edge" of a set children, closest to the spouse. For pedigrees that have only a single connection between two families this simple-minded approach works surprisingly well. For more complex structures either hand-tuning of the hints matrix, or use of the *besthint* routine will usually be required.

E.3   bdsmatrix.ibd

Routines that create identity-by-descent (ibd) coefficients often output their results as a list of values (i, j, x[i,j]), with unlisted values of the x matrix assumed to be zero. This routine recasts such a list into *bdsmatrix* form.

```
bdsmatrix.ibd(id1, id2, x, idmap, diagonal=1)
```

Required arguments

id1 row identifier for the value, in the final matrix. Optionally, *id1* can be a 3 column matrix or data.frame, in which case it is assumed to contain the first 3 arguments, in order.

id2 column identifier for the value, in the final matrix.

x the value to place in the matrix

Optional arguments

idmap a two column matrix or data frame. Sometimes routines create output with integer values for *id1* and *id2*, and then this argument is the mapping from this internal label to the "real" name).

diagonal If diagonal elements are not preserved in the list, this value will be used for the diagonal of the result. If the argument appears, then the output matrix will contain an entry for each value in *idlist*. Otherwise only those with an explicit entry appear.

Return value

a *bdsmatrix* object representing a block-diagonal sparse matrix.

The routine first checks for non-symmetric or otherwise inconsistent input. It then groups observations together into 'families' of related subjects, which determines the structure of the final matrix. As with the *makekinship* function, singletons with no relationships are first in the output matrix, and then families appear one by one.

E.4   bdsmatrix

Sparse block diagonal matrices are used in the the large parameter matrices that can arise in random-effects coxph and survReg models. This routine creates such a matrix. Methods for these matrices allow them to be manipulated much like an ordinary matrix, but the total memory use can be much smaller.

```
bdsmatrix(blocksize, blocks, rmat, dimnames)
```

Required arguments

blocksize vector of sizes for the matrices on the diagonal

blocks contents of the diagonal blocks, strung out as a vector

Optional arguments

rmat the dense portion of the matrix, forming a right and lower border

dimnames a list of dimension names for the matrix

Return value

an object of type bdsmatrix

Consider the following matrix, which has been divided into 4 parts.

```
1  2  0  0  0 | 4  5
2  1  0  0  0 | 6  7
0  0  3  1  2 | 8  8
0  0  1  4  3 | 1  1
0  0  2  3  5 | 2  2
-------------+-----
4  6  8  1  2 | 7  6
5  7  8  1  2 | 6  9
```

The upper left is block diagonal, and can be stored in a compressed form without the zeros. With a large number of blocks, the zeros can actually account for over 99% of a matrix; this commonly happens with the kinship matrix for a large collection of families (one block/family). The arguments to this routine would be block sizes of 2 and 3, along with a 2 by 7 "right hand" matrix. Since the matrix is symmetrical, the bottom slice is not needed.

```
# The matrix shown above is created by
tmat <- bdsmatrix(c(2,3), c(1,2,1, 3,1,2, 4,3, 5),
                  rmat=matrix(c(4,6,8,1,2,7,6, 5,7,8,1,2,6,9), ncol=2))

# Note that only the lower part of the blocks is needed, however, the
#  entire block set is also allowed, i.e., c(1,2,2,1, 3,1,2,1,4,3,2,3,5)
```

E.5   besthint

A pedigree structure can contain a *hints* matrix which helps to reorder the pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine trys out a large number of configurations, finding the best by brute force.

```
besthint(ped, wt=c(1000, 10, 1), tolerance=0)
```

Required arguments

ped a pedigree object

Optional arguments

wt relative weights for three types of "distortion" in a plotted pedigree. The final score for a pedigree is the weighted sum of these; the lowest score is considered the best. The three components are 1: the number of dotted lines, connecting two instances of the same person; 2: the lengths of those dotted lines; and 3: the horizontal offsets between parent/child pairs.

tolerance the threshold for acceptance. If any of the orderings that are attempted have a score that is less than or equal to this value, the routine ceases searching for a better one.

Return value

a hints matrix

Assume that a pedigree has k founding couples, i.e., husband-wife pairs for which neither has a parent in the pedigree. The routine tries all $k!/2$ possible left to right orderings of the founders (in random order), uses the *autohint* function to optimize the order of children within each family, and computes a score. The hints matrix for the first pedigree to match the tolerance level is returned, or that for the best score found if none match the tolerance.

```
# Find a good plot, only trying to avoid dotted connectors
myped$hints <- besthint(myped, wt=c(1000,100,0))
```

E.6   coxme.control

Set various control parameters for the *coxme* function.

```
coxme.control(eps=0.00001, toler.chol=.Machine$double.eps^0.75,
              toler.ms=.01, inner.iter=4, iter.max=10, sparse.calc=NULL)
```

Optional arguments

eps convergence criteria. Iteration ceases when the relative change in the log-likelihood is less than *eps*.

toler.chol tolerance that is used to detect singularity, i.e., redundant predictor variables in the model, in the underlying Cholesky decompostion routines.

toler.ms convergence criteria for the minimization of the integrated loglikelihood over the variance parameters. Since this "outer" iteration uses the Cox iteration as an inner loop, and the Cox iteration in turn uses the cholesky decomposition as an inner look, each of these treating the computations below it as if they were exact, the cholesky tolerance should be tighter than the Cox tolerance, which in turn should be tighter than that for the variance estimates. Also keep in mind that for any but enormous data sets, the standard errors of the variance terms are often of the order of 10-20% of their value. It does not make much sense to iterate to a "precision" of .0001 on a value with statistical uncertainty of 0.1.

inner.iter the number of iterations for the inner iteration loop.

iter.max maximum number of iterations for solution of a Cox partial likelihood, given the values of the random effect variances. Calls with *iter=0* are useful to evaluate the likelihood for a prespecified parameter vector, such as in the computation of a profile likelihood.

sparse.calc style of computation for the inner likelihood code. The results of the two computations are identical, but can differ in total compute time. The optional calculation (calc=1) uses somewhat more memory, but can be substantially faster when the total number of random effects is of order n, the total sample size. The standard calculation (calc=0) is faster when the number of random effects is small. By default, the *coxme.fit* function chooses the method dynamically. It may not always do so optimally.

Return value

a list containing values for each option.

The central computation of *coxme* consists of an outer maximization to determine the variances of the random effects, performed by the *nlmin* function. Each evaluation for nlmin, however, itself requires the solution of a minimization problem; this is the inner loop. It is important that the inner loop use a fixed number of iterations, but it is not yet clear what is the minimal sufficient number for that inner loop. Making this number smaller will make the routine faster, but perhaps at the expense of accuracy.

E.7   coxme

Returns an object of class *coxme* representing the fitted model.

```
coxme(fixed, data, random,
   weights, subset, na.action, init, control,
   ties=c("efron", "breslow", "exact"), singular.ok=T,
   varlist, variance, vinit=.2, sparse=c(50, .02), rescale=T, x=F, y=T, ...)
```

Required arguments

fixed formula decribing the fixed effects part of the model.

data a data frame containing the variables.

random a one-sided formula describing the random effects part of the model.

Optional arguments

weights case weights for each observation

subset an expression describing the subset of the data that should be used in the fit.

na.action a function giving the default action on encountering missing values. It is more usual to use the global na.action system option to control this.

init initial values for the coefficients for the fixed portion of the model, or the frailties followed by the fixed effect coefficients.

control the result of a call to *coxph.control*

ties the approximation to be used for tied death times: either "efron" or "breslow"

singular.ok if TRUE, then redundant coefficients among the fixed effects are set to NA, if FALSE the program will fail with an error message if there are redundant variables.

varlist variance specifications, often of class *bdsmatrix*, decsribing the variance/covariance structure of one or more of the random effects.

variance fixed values for the variances of selected random effects. Values of 0 are placeholders and do not specify a fixed value.

vinit the initial value to be used for the variance estimates. This only applies to those parameters that were not given a fixed value. The default value reflects two things: first that final results are in the range $[0, .5]$ much of the time, and second that the inner Cox model iteration can sometimes become unstable for variance parameters larger than 1–2.

sparse determines which levels of the random effects factor variables, if any, for which the program will use sparse matrix techniques. If a grouping variable has less than sparse[1] levels, then sparse methods are not used for that variable. If it has greater than or equal to sparse[1] unique levels, sparse methods will be used for those values which represent less than sparse[2] as a proportion of the data. For instance, if a grouping variable has 4000 levels, but 40% of the subjects are in group 1, 10% in group 2 and the rest distributed evenly, then 3998 of the levels will be represented sparsely in the variance matrix. A single logical value of F is equivalent to setting sparse[1] to infinity.

rescale scale any user supplied variance matrices so as to have a diagonal of 1.0.

pdcheck verify that any user-supplied variance matrix is positive definite (SPD). It has been observed that IBD matrices produced by some software are not strictly SPD. Sometimes models with these matrices still work (throughout the iteration path, the weighted sum of variance matrices was always SPD) and sometimes they don't. In the latter case, the occurence of non-spd matrices will effectively constrain some variance parameters away from 0.

x retain the X matrix in the output.

y retain the dependent variable (a Surv object) in the output.

Return value

an object of class coxme

E.8   familycheck

Compute the familial grouping structure directly from (id, mother, father) information, and compare the results to the supplied family id variable.

```
familycheck(famid, id, father.id, mother.id, newfam)
```

Required arguments

famid a vector of family identifiers

id a vector of unique subject identifiers

father.id vector containing the id of the biological father

mother.id vector containing the id of the biological mother

Optional arguments

newfam the result of a call to *makefamid*. If this has allready been computed by the user, adding it as an argument shortens the running time somewhat.

Return value: a data frame with one row for each unique family id in the *famid* argument.

famid the family id, as entered into the data set

n number of subjects in the family

unrelated number of them that appear to be unrelated to anyone else in the entire pedigree set. This is usually marry-ins with no children (in the pedigree), and if so are not a problem.

split number of unique "new" family ids. If this is 0, it means that no one in this "family" is related to anyone else (not good); 1 = everythings is fine; 2+= the family appears to be a set of disjoint trees. Are you missing some of the people?

join number of other families that had a unique famid, but are actually joined to this one. 0 is the hope. If there are any joins, then an attribute "join" is attached. It will be a matrix with famid as row labels, new-family-id as the columns, and the number of subjects as entries.

The *makefamid* function is used to create a de novo family id from the parentage data, and this is compared to the family id given in the data. makefamid, makekinship

E.9   gchol

Perform the generalized Cholesky decompostion of a real symmetric matrix.

```
gchol(x, tolerance=1e-10)
```

Required arguments

x the symmetric matrix to be factored

Optional arguments

tolerance the numeric tolerance for detection of singular columns in x.

Return value

an object of class *gchol* containing the generalized Cholesky decompostion. It has the appearance of a lower triangular matrix.

The *solve* has a method for gchol decompostions, and there are gchol methods for block diagonal symmetric (*bdsmatrix*) matrices as well.

```
# Create a matrix that is symmetric, but not positive definite
#    The matrix temp has column 6 redundant with cols 1-5
smat <- matrix(1:64, ncol=8)
smat <- smat + t(smat) + diag(rep(20,8))  #smat is 8 by 8 symmetric
temp <-  smat[c(1:5, 5:8), c(1:5, 5:8)]
```

```
ch1  <- gchol(temp)

print(as.matrix(ch1))   # print out L
print(diag(ch1))        # print out D
aeq <- function(x,y) all.equal(as.vector(x), as.vector(y))
aeq(diag(ch1)[6], 0)    # Check that it has a zero in the proper place

ginv <- solve(ch1)    # see if I get a generalized inverse
aeq(temp %*% ginv %*% temp, temp)
aeq(ginv %*% temp %*% ginv, ginv)
```

E.10   kinship

Computes the n by n kinship matrix for a set of n related subjects

```
kinship(id, father.id, mother.id)
```

Required arguments

id a vector of subject identifiers. It may be either numeric or character.

father.id for each subject, the identifier of the biological father.

mother.id for each subject, the identifier of the biological mother.

Return value

a matrix of kinship coefficients.

Two genes G1 and G2 are identical by descent (ibd) if they are both physical copies of the same ancestral gene; two genes are identical by state if they represent the same allele. So the brown eye gene that I inherited from my mother is ibd with hers; the same gene in an unrelated individual is not.

The kinship coefficient between two subjects is the probability that a randomly selected allele will be ibd between them. It is obviously 0 between unrelated individuals. If there is no inbreeding in the pedigree, it will be .5 for an individual with themselves (we could choose the same allele twice), .25 between mother and child, etc [4].

The computation is based on a recursive algorithm described in Lange. It is unfortunately not vectorizable, so the S code is slow. For studies with multiple disjoint families see the *makekinship* routine.

```
test1 <- data.frame(id  =c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14),
                    mom =c(0, 0, 0, 0, 2, 2, 4, 4, 6,  2,  0,  0, 12, 13),
                    dad =c(0, 0, 0, 0, 1, 1, 3, 3, 3,  7,  0,  0, 11, 10),
                    sex =c(0, 1, 0, 1, 0, 1, 0, 1, 0,  0,  0,  1,  1,  1))
round(8*kinship(test1$id, test1$dad, test1$mom))

   1 2 3 4 5 6 7 8 9 10 11 12 13 14
 1 4 0 0 0 2 2 0 0 1  0  0  0  0  0
```

```
 2 0 4 0 0 2 2 0 0 1  2  0  0  0  1
 3 0 0 4 0 0 0 2 2 2  1  0  0  0  0
 4 0 0 0 4 0 0 2 2 0  1  0  0  0  0
 5 2 2 0 0 4 2 0 0 1  1  0  0  0  0
 6 2 2 0 0 2 4 0 0 2  1  0  0  0  0
 7 0 0 2 2 0 0 4 2 1  2  0  0  0  1
 8 0 0 2 2 0 0 2 4 1  1  0  0  0  0
 9 1 1 2 0 1 2 1 1 4  1  0  0  0  0
10 0 2 1 1 1 1 2 1 1  4  0  0  0  2
11 0 0 0 0 0 0 0 0 0  0  4  0  2  1
12 0 0 0 0 0 0 0 0 0  0  0  4  2  1
13 0 0 0 0 0 0 0 0 0  0  2  2  4  2
14 0 1 0 0 0 0 1 0 0  2  1  1  2  4
```

genetics

E.11   lmekin

A similar function to lme, but allowing for a complete specification of the co-variance matrix for the random effects.

```
lmekin(fixed, data=sys.parent, random,
    varlist, variance, sparse=c(20, .05),
    rescale=T, pdcheck=T,
    subset, weight, na.action)
```

Required arguments

fixed model statement for the fixed effects

random model statement for the random effects

Optional arguments

data data frame containing the variables

varlist variance specifications, often of class *bdsmatrix*, describing the variance/covariance structure of one or more of the random effects.

variance fixed values for the variances of selected random effects. Values of 0 indicate that the final value should be solved for.

sparse determines which levels of random effects factor variables, if any, for which the program will use sparse matrix techniques. If a grouping variable has less than sparse[1] levels, then sparse methods are not used for that variable. If it has greater than or equal to sparse[1] unique levels, sparse methods will be used for those values which represent less than sparse[2] as a proportion of the data. For instance, if a grouping variable has 4000 levels, but 40subjects are in group 1 then 3999 of the levels will be represented sparsely in the variance matrix. A single logical value of F is equivalent to setting sparse[1] to infinity.

rescale scale any user supplied variance matrices so as to have a diagonal of 1.0.

pdcheck verify that any user-supplied variance matrix is positive definite (SPD). It has been observed that IBD matrices produced by some software are not strictly SPD. Sometimes models with these matrices still work (throughout the iteration path, the weighted sum of variance matrices was always SPD) and sometimes they don't. In the latter case, messages about taking the log of negative numbers will occur, and the results of the fit are not necessarily trustworthy.

subset selection of a subset of data

weight optional case weights

na.action the action for missing data values

Return value

an object of class *lmekin*, sharing similarities with both lm and lme objects.

The lme function is designed to accept a prototype for the variance matrix of the random effects, with the same prototype applying to all of the groups in the data. For familial genetic random effects, however, each family has a different covariance pattern, necessitating the input of the entire set of covariance matrices. In return, at present lmekin does not have the prototype abilities of lme.

```
#
# Make a kinship matrix for the entire study
# These two functions are NOT fast, the makekinship one in particular
#
cfam <- makefamid(main$gid, main$momid, main$dadid)
kmat <- makekinship(cfam, main$gid, main$momid, main$dadid)

# The kinship matrix for the females only: quite a bit smaller
#
kid <- dimnames(kmat)[[1]]
temp <- main$sex[match(kid, main$gid)] == 'F'
fkmat <- kmat[temp,temp]

# The dimnames on kmat are the gid value, which are necessary to match
#  the appropriate row/col of kmat to the analysis data set
# A look at %dense tissue on a mammogram, with age at mammogram and
#  weight as covariates, and a familial random effect
#
fit <- lmekin(percdens ~ mammage + weight, data=anal1,
                    random = ~1|gid, kmat=fkmat)

Linear mixed-effects kinship model fit by maximum likelihood
```

```
  Data: anal1
  Log-likelihood = -6093.917
  n= 1535

Fixed effects: percdens ~ mammage + weight
 (Intercept)    mammage     weight
     87.1593 -0.5333198 -0.1948871


Random effects:  ~ 1 | gid
         Kinship Residual
StdDev: 7.801603 10.26612
```

E.12   makefamid

Given a set of parentage relationships, this subdivides a set of subjects into families.

```
makefamid(id, father.id, mother.id)
```

Required arguments

id a vector of unique subject identifiers

father.id for each subject, the identifier of their biological father

mother.id for each subject, the identifier of their biological mother

Return value

a vector of family identifiers. Individuals who are not blood relatives of anyone else in the data set as assigned a family id of 0.

This function may be useful to create a family identifier if none exists in the data (rare), to check for anomalies in a given family identifier (see the *family-check* function), or to create a more space and time efficient kinship matrix by separating out marry-ins without children as 'unrelated'. makefamid, kinship, makekinship

```
> newid <- makefamid(cdata$gid, cdata$dadid, cdata$momid)
> table(newid==0)
 FALSE TRUE
 17859 8191
# So nearly 1/3 of the individuals are not blood relatives.

> kin1 <- makekinship(cdata$famid, cdata$gid, cdata$dadid, cdata$momid)
> kin2 <- makekinship(newid, cdata$gid, cdata$dadid, cdata$momid, unique=0)
> dim(kin2)
[1] 26050 26050
> dim(kin1)
[1] 26050 26050
```

```
> length(kin2@blocks)/length(kin1@blocks)
[1] 0.542462
# Basing kin1 on newid rather than cdata$famid (where marry-ins were each
#   labeled as members of one of the 426 families) reduced its size by just
#   less than half.
```

E.13   makekinship

Compute the overall kinship matrix for a collection of families, and store it efficiently.

```
makekinship(famid, id, father.id, mother.id, father.id, unrelated=0)
```

Required arguments

famid a vector of family identifiers

id a vector of unique subject identifiers

father.id for each subject, the identifier of their biolgical father

mother.id for each subject, the identifier of thier biological mother

Optional arguments

unrelated subjects with this family id are considered to be unrelated single-tons, i.e., not related to each other or to anyone else.

Return value

a sparse kinship matrix of class *bdsmatrix*

For each family of more than one member, the *kinship* function is called to calculate a per-family kinship matrix. These are stored in an efficient way into a single block-diagaonal sparse matrix object, taking advantage of the fact that between family entries in the full matrix are all 0. Unrelated individuals are considered to be families of size 0, and are placed first in the matrix. ¡br¿¡br¿ The final order of the rows within this matrix will not necessarily be the same as in the origianl data, since each family must be contiguous. The dimnames of the matrix contain the id variable for each row/column. Also note that to create the kinship matrix for a subset of the data it is necessary to create the full kinship matrix first and then subset it. One cannot first subset the data and then call the function. For instance, a call using only the female data would not detect that a particular man's sister and his daughter are related.

```
# Data set from a large family study of breast cancer
#  there are 26050 subjects in the file, from 426 families
> table(cdata$sex)
     F     M
 12699 13351
```

48

```
> length(unique(cdata$famid))
[1] 426

> kin1 <- makekinship(cdata$famid, cdata$gid, cdata$dadid, cdata$momid)
> dim(kin1)
[1] 26050 26050
> class(kin1)
[1] "bdsmatrix"
# The next line shows that few of the elements of the full matrix are >0
> length(kin1@blocks)/ prod(dim(kin1))
[1] 0.00164925

# kinship matrix for the females only
 femid <- cdata$gid[cdata$sex=='F']
 femindex <- !is.na(match(dimnames(kin1)[[1]], femid))
 kin2 <- kin1[femindex, femindex]
#
# Note that "femindex <- match(femid, dimnames(kin1)[[1]])" is wrong, since
#  then kin1[femindex, femindex] might improperly reorder the rows/cols
#  (if families were not contiguous in cdata).
# However sort(match(femid, dimnames(kin1)[[1]])) would be okay.
```

E.14   pedigree

Create pedigree structure in format needed for plotting function.

```
pedigree(id, momid, dadid, sex, affected, status, ...)
```

Required arguments

id Identification variable for individual

momid Identification variable for mother

dadid Identification variable for father

sex Gender of individual noted in 'id'. Character("male","female","unknown",
"terminated") or numeric (1="male", 2="female", 3="unknown", 4="ter-
minated") allowed.

Optional arguments

affected One variable, or a matrix, indicating affection status. Assumed that
1="unaffected", 2="affected", NA or 0 = "unknown".

status Status (0="censored", 1="dead")

... Additional variables to be carried along with the pedigree.

Return value

An object of class pedigree.

E.15 plot.pedigree

plot objects created with the function pedigree

```
plot.pedigree(x, id=x$id, sex=x$sex, status=x$status,
        affected=as.matrix(x$affected), cex=1,
        col=rep(1, length(x$id)), symbolsize=1,
        branch=0.6, packed=T, align=packed, width=8,
        density=c(-1, 50, 70, 90), angle=c(90, 70, 50, 0))
```

Required arguments

x object created by the function pedigree.

Optional arguments

id id variable - used for labeling.

sex sex variable - used to determine which symbols are plotted.

status can be missing. If it exists, 0=alive/missing and 1=death.

affected variable, or matrix, of up to 4 columns representing 4 different affected statuses.

cex controls text size. Default=1.

col color for each id. Default assigns the same color to everyone.

symbolsize controls symbolsize. Default=1.

branch defines how much angle is used to connect various levels of nuclear families.

packed default=T. If T, uniform distance between all individuals at a given level.

align

width

density defines density used in the symbols. Takes up to 4 different values.

angle defines angle used in the symbols. Takes up to 4 different values.

Return value: returns points for each plot plus original pedigree.

E.16   solve.bdsmatrix

This function solves the equation Ax=b for x, when A is a block diagonal sparse matrix (an object of class *bdsmatrix*).

```
solve.bdsmatrix(a, b, tolerance=1e-10, full=T)
```

Required arguments

a a block diagonal sparse matrix object

Optional arguments

b a numeric vector or matrix, that forms the right-hand side of the equation.

tolerance the tolerance for detecting singularity in the a matrix

full if true, return the full inverse matrix; if false return only that portion corresponding to the blocks. This argument is ignored if *b* is present. If the bdsmatrix *a* has a non-sparse portion, i.e., if the *rmat* component is present, then the inverse of *a* will not be block-diagonal sparse. In this case setting full=F returns only a portion of the inverse. The elements that are returned are those of the full inverse, but the off-diagonal elements that are not returned would not have been zero.

Return value: if argument *b* is not present, the inverse of *a* is returned, otherwise the solution to matrix equation. The equation is solved using a generalized Cholesky decomposition.

The matrix *a* consists of a block diagonal sparse portion with an optional dense border. The inverse of *a*, which is to be computed if *y* is not provided, will have the same block diagonal structure as *a* only if there is no dense border, otherwise the resulting matrix will not be sparse.

However, these matrices may often be very large, and a non sparse version of one of them will require gigabytes of even terabytes of space. For one of the common computations (degrees of freedom in a penalized model) only those elements of the inverse that correspond to the non-zero part of *a* are required; the *full=F* option returns only that portion of the (block diagonal portion of) the inverse matrix.

```
> tmat <- bdsmatrix(c(3,2,2,4),
             c(22,1,2,21,3,20,19,4,18,17,5,16,15,6,7, 8,14,9,10,13,11,12),
             matrix(c(1,0,1,1,0,0,1,1,0,1,0,10,0,
                       0,1,1,0,1,1,0,1,1,0,1,0,10), ncol=2))
> dim(tmat)
[1] 13 13
> solve(tmat, cbind(1:13, rep(1,13)))
```

E.17 solve.gchol

This function solves the equation Ax=b for x, given b and the generalized Cholesky decomposition of A. If only the first argument is given, then a G-inverse of A is returned.

```
solve.gchol(a, b, full=T)
```

Required arguments

a a generalized Cholesky decomposition of a matrix, as returned by the *gchol* function.

Optional arguments

b a numeric vector or matrix, that forms the right-hand side of the equation.

full solve the problem for the full (original) matrix, or for the Cholesky matrix.

Return value

if argument *b* is not present, the inverse of *a* is returned, otherwise the solution to matrix equation.

A symmetric matrix A can be decomposed as LDL', where L is a lower triangular matrix with 1's on the diagonal, L' is the transpose of L, and D is diagonal. This routine solves either the original problem Ay=b (*full* argument) or the subproblem sqrt(D)L'y=b. If *b* is missing it returns the inverse of A or L, respectively.

```
# Create a matrix that is symmetric, but not positive definite
#   The matrix temp has column 6 redundant with columns 1-5
> smat <- matrix(1:64, ncol=8)
> smat <- smat + t(smat) + diag(rep(20,8))  #smat is 8 by 8 symmetric
> temp <-  smat[c(1:5, 5:8), c(1:5, 5:8)]
> ch1  <- gchol(temp)

> print(as.matrix(ch1))   # print out L
> print(diag(ch1))        # print out D
> aeq <- function(x,y) all.equal(as.vector(x), as.vector(y))
> aeq(diag(ch1)[6], 0)     # Check that it has a zero in the proper place

> ginv <- solve(ch1)    # see if I get a generalized inverse
> aeq(temp %*% ginv %*% temp, temp)
> aeq(ginv %*% temp %*% ginv, ginv)
```

F   Model statements

The `coxme` and `lmekin` agree with the `lme` function in how they process simple random effects formulas. The simplest model is of the form `covariate | group`. Two routines from the `lme` suite break this formula apart nicely:

```
> test <- ~ (age + weight) | inst/sex
> getGroupsFormula(test)
~ inst/sex
> getCovariateFormula(test)
 ~ (age + weight)
```

The functions, however, do not properly extend to multiple terms,

```
> getGroupsFormula(~1|inst + age|sex)
 ~ sex
> getCovariateFormula(~1|inst + age|sex)
 ~ 1 | inst + age
```

Further exploration shows that a formula object is stored as a recursive parse tree, with vertical bar binding least tightly. (Operator precedence is first the parenthesis, then the caret, then */, then +-, then |). Operators of the same precedence are processed from right to left. Consider the following potential formula, consisting of a random institutional effect along with random slopes within gender:

```
> test <- ~ 1|inst + age|sex
> class(test)
[1] "formula"
> length(test)
[1] 2
> test[[1]]
 ~
> test[[2]]
1 | inst + age | sex
> class(test[[2]])
[1] "call"
> length(test[[2]])
[1] 3
```

The full tree is shown below, giving each object followed by its class in parenthesis. The left hand column shows the top level list of length 2; it's second element is itself a list of length 3 shown in the next column, etc.

$\sim$ (name)

$$
1|inst+age|sex(\text{call}) \begin{cases} |(\text{name}) \\ \\ 1|inst+age\ (\text{call}) \begin{cases} |\ (\text{name}) \\ 1\ (\text{integer}) \\ inst+age\ (\text{call}) \begin{cases} +(\text{name}) \\ inst(\text{name}) \\ age\ (\text{name}) \end{cases} \end{cases} \\ \\ sex(\text{name}) \end{cases}
$$
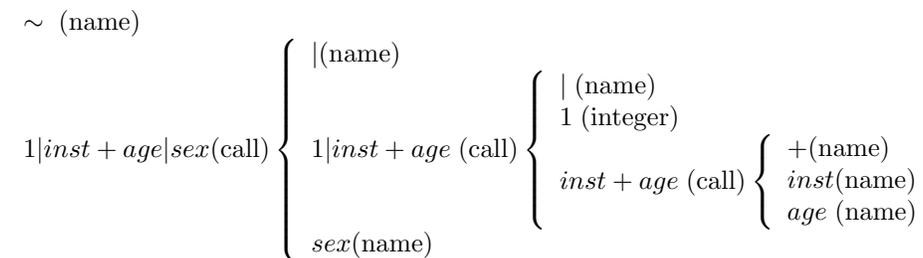
So `test[[2]][[2]][[3]][[3]]` is age.

The `lme` functions simply assume that the parse tree will have a particular structure; they fail for instance on a parenthesized expression `random=`

(1|group). We have added getCrossedTerms, which breaks the formula into distinct crossed terms, e.g. the formula $\sim$ 1|inst + age|sex ends up as a list with two components $\sim$ 1|inst and $\sim$ age|sex, each of which is appropriate for further processing. The getGroupsFormula and getCovariatesFormula routines can then be called on the simpler objects.

In extending to more complex structures lme uses pdMat structures, for which we essentially could never figure out the computer code; coxme moves forward with a varlist. Much of the reason for this is actually computational, as the elegant decomposition methods used by lme for speed are not available in more general likelihoods, negating much of the advantage of pdMat.

References

[1] V. E Anderson, H. O. Goodman, and S. Reed. *Variables Related to Human Breast Cancer*. University of Minnesota Press, Minneapolis, 1958.

[2] P. P. Broca. *Traites de Tumerus, volumes 1 and 2*. Asselin, Paris, 1866.

[3] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23:53–60, 1981.

[4] K. Lange. *Mathematical and Statistical Methods for Genetic Analysis*. Springer-Verlag, New York, 1997.

[5] K. Lange. *Mathematical and Statistical Methods for Genetic Analysis*. Springer-Verlag, New York, 2002.

[6] S. Ripatti and J. Palmgren. Estimation of multivariate frailty models using penalized partial likelihood. *Biometrics*, 56:1016–1022, 2000.

[7] S. R. Searle. *Matrix Algebra Useful for Statistics*. Wiley, New York, 1982.

[8] T. A Sellers, V. E. Anderson, J. D. Potter, S. A. Bartow, P. L. Chen, L. Everson, R. A. King, C. C. Kuni, L. H. Kushi, P. G. McGovern, S. S. Rich, J. F. Whitbeck, and G. L. Wiesner. Epidemiologic and genetic follow-up study of 544 minnesota breast cancer families: Design and methods. *Genetic Epidemiology*, 12:417–429, 1995.

[9] T.M. Therneau and P.M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York, 2000.

[10] 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.