Making general deviance functions for mixed/factor models in lme4

Steve Walker

Contents

1 Introduction

The generalized linear mixed model (GLMM) in `lme4` takes the form,

$$\boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{b} \tag{1}$$

$$\boldsymbol{b} = \boldsymbol{\Lambda}_{\boldsymbol{\theta}}\boldsymbol{u} \tag{2}$$

$$\boldsymbol{u} \sim \mathcal{N}(0, \boldsymbol{I}) \tag{3}$$

$$\boldsymbol{y} \sim \mathcal{D}(\boldsymbol{\eta}, \boldsymbol{\phi}) \tag{4}$$

where $\mathcal{D}$ is an exponential family distribution. This GLMM is suitable for modelling a wide variety of data. However, in community ecology the response variables are often species abundances or species presence-absence, and such data are often characterized by correlations between species even after the effects of environmental, phylogenetic, and space are accounted for. These correlations among species are typically due to either unmeasured site and species characteristics and species interactions. These correlations can be accounted for by allowing

$\boldsymbol{Z}$ to depend on a vector of parameters, $\boldsymbol{\psi}$. Therefore the model that we will be using is given by,

$$\boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}_{\psi}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\boldsymbol{u} \tag{5}$$

We now have three parameter vectors:

- covariance parameters, $\boldsymbol{\theta}$, `covar`
- fixed effect parameters, $\boldsymbol{\beta}$, `fixef`
- general factor loadings, $\boldsymbol{\psi}$, `loads`

Here I illustrate a function, `mkGeneralGlmerDevfun`, in the `lme4ord` package, which can be used to fit such models.

2 Simple simulation example

I simulate a small data set with the following dimensions.

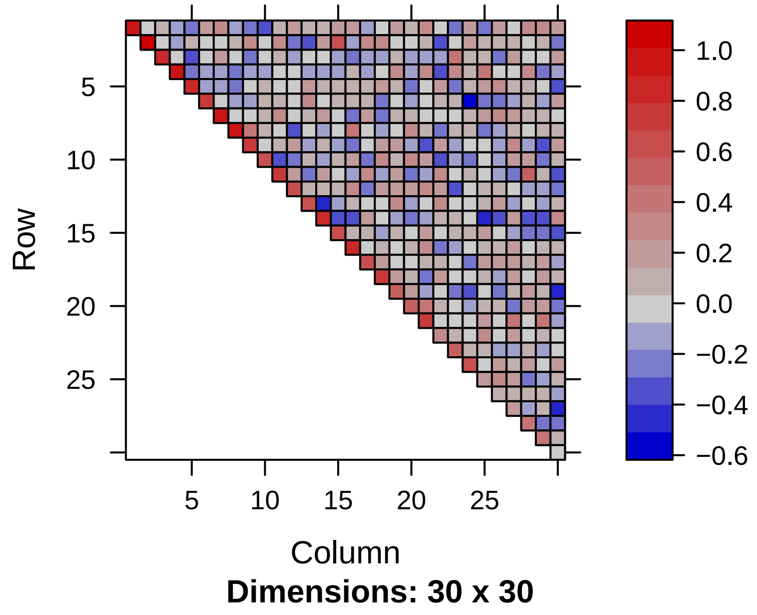
```
n <- 60 # samples
p <- 2  # fixed effects
q <- 30 # random effects
```

The covariance factor, $\Lambda_{\boldsymbol{\theta}}^{\top}$, is dense (although we must specify it with a sparse structure).

```
covTemplate <- as(chol(cov(matrix(rnorm((q+1)*q), q + 1, q))),
                  "sparseMatrix")
Lambdat <- covTemplate
```

This structure could represent a phylogenetic covariance matrix for example.

```
image(Lambdat)
```

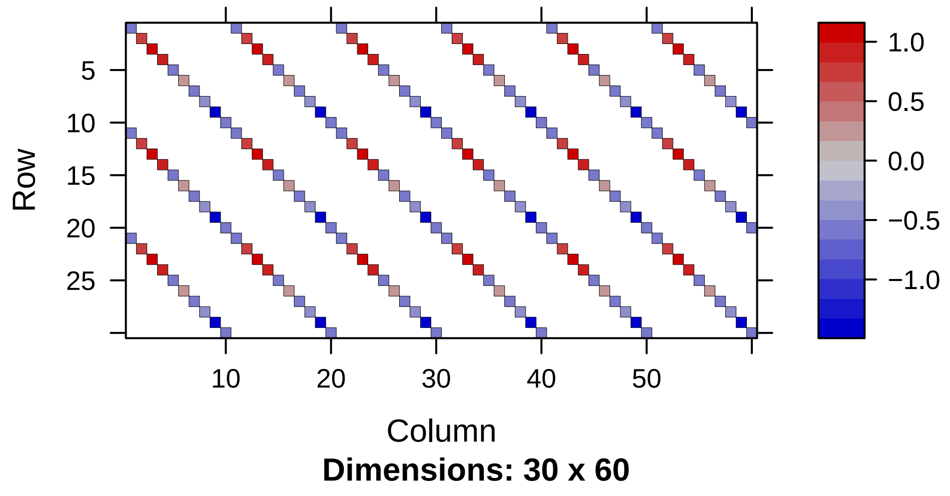


The transposed random effects model matrix and fixed effects model matrix are given by the following.

```
Zt <- sparseMatrix(i = rep(1:q, 6),
                   j = as.vector(outer(rep(1:10, 3), seq(0, 50, 10), "+")),
                   x = rep(rnorm(10), 18))
X <- matrix(rnorm(n * p), n, p)
```

The matrix has the following pattern. I don't know what this might represent, but I just wanted to show that essentially any structure will be fine. For real examples, one may freely compute Kronecker products and prewhiten, etc.

```
image(Zt)
```



Then we simulate the response vector, weights, and offset.

```
eta <- as.numeric(X %*% c(1, 1) + t(Zt) %*% t(Lambdat) %*% rnorm(q))
y <- rbinom(n, 1, plogis(eta))
weights <- rep(1, n); offset <- rep(0, n)
```

2.1 Organize the parameter vector

To fit this model we need to use a general nonlinear optimizer, which usually require a single parameter vector. Therefore, I put all three types of parameters in a single vector.

```
initPars <- c(covar = 1,
              fixef = c(0, 0),
              loads = rnorm(10))
```

Importantly, the deviance function needs to know how to find the different types of parameters, and one specifies this with a list of indices.

```
parInds <- list(covar = 1,
                fixef = 2:3,
                loads = 4:13)
```

Because this is a general approach, we must specify functions that take the parameter vectors and update the various objects. In particular, we need a function to map the `covar` parameters into the nonzero values of `Lambdat` (stored in `Lambdat@x`). Similarly, we need to map the factor loadings, `loads`, into the nonzero values of `Zt`.

```r
mapToCovFact <- function(covar) covar * covTemplate@x
mapToModMat <- function(loads) rep(loads, 18)
```

Note that the covariance factor is updated as though it were a Brownian motion model with parameter `covar` controlling the rate of evolution. Here I use the covariance factor for introducing phylogenetic information, but the random effects model matrix could also be used. In this case however, I wanted to illustrate the possibility of including factor loadings, which should be in the model matrix. In this illustrative example, the parameters have an effect on every nonzero element of the matrices. However, often (usually) there are no factor loadings and in this case, `mapToModMat` should just return the same nonzero values for any value of the loadings. Here is an example of using the mapping functions to update the two sparse matrices.

```r
Lambdat@x <- mapToCovFact(initPars[parInds$covar])
Zt@x <- mapToModMat(initPars[parInds$loads])
```

2.2 Construct the deviance function

The interesting thing to report here is that once all of these structures and mappings are produced, the computation of the deviance function is now quite straightforward.

```r
devfun <- mkGeneralGlmerDevfun(y, X, Zt, Lambdat,
                               weights, offset,
                               initPars, parInds,
                               mapToCovFact, mapToModMat)
```

Here is an example of evaluating it.

```r
devfun(initPars)

## [1] 76.4517
```

2.3 Optimize the deviance function

We may now use any nonlinear optimizer.

```r
opt <- minqa:::bobyqa(initPars, devfun)
```

And here are the optimum parameter values.

```
setNames(opt$par, names(initPars))

##      covar     fixef1     fixef2     loads1     loads2     loads3
##  1.8716033  1.7418559  2.2685221 -0.4423496  0.4861872  0.2376486
##     loads4     loads5     loads6     loads7     loads8     loads9
##  1.8161956  2.3022474  1.6111034  0.6569749 -0.1876849 -0.6591299
##    loads10
##  0.1047986
```

3 A latent factor model

```
library(reo)
library(multitable)
```

Here is an example with Don Jackson's Masters thesis.

```
data(fish)
data(limn)
Y <- as.matrix(fish)
n <- nrow(Y)
m <- ncol(Y)
x <- as.vector(scale(limn$pH))
dl <- data.list(Y = Y, x = x,
                dimids = c("sites", "species"))
dl <- dims_to_vars(dl)
summary(dl)

##            Y     x sites species
## sites   TRUE  TRUE  TRUE   FALSE
## species TRUE FALSE FALSE    TRUE
```

I use the `multitable` package to organize multivariate data (could put traits in there too). Convert it to long format,

```
head(df <- as.data.frame(dl))

##                 Y          x       sites species
## 3 Island.PS     0  0.1843841    3 Island      PS
## Austin.PS       0 -0.8325221      Austin      PS
## Bear.PS         1  0.8277329        Bear      PS
## Bentshoe.PS     1 -0.8532753    Bentshoe      PS
## Big East.PS     1 -0.6249903    Big East      PS
## Big Orillia.PS 1  0.2051373 Big Orillia      PS
```

The response vector.

```
y <- df$Y
weights <- rep(1, length(y)); offset <- rep(0, length(y))
```

The fixed effects design matrix.

```
X <- model.matrix(Y ~ x, df)[,]
```

The random effects design matrix.

```
Jspecies <- t(as(as.factor(df$species), Class = "sparseMatrix"))
Zt <- KhatriRao(t(Jspecies), t(X))
```

Now this design matrix only contains 'traditional' random effects, not factor loadings.

```
nFreeLoadings <- m
U <- matrix(1:nFreeLoadings, nrow = m, ncol = 1)
latentVarNames <- "latent"
U <- setNames(as.data.frame(U), latentVarNames)
latentData <- data.list(U, drop = FALSE, dimids = "species")
df <- as.data.frame(dl + latentData)
Jsites <- with(df, t(as(as.factor(sites), Class = "sparseMatrix")))
Zt <- rBind(KhatriRao(t(Jsites), t(model.matrix(Y ~ 0 + latent, df))), Zt)
```

```
templateFact <- sparseMatrix(i = 1:n, j = 1:n, x = rep(1, n))
ii <- rep(1:2, 1:2); jj <- sequence(1:2)
templateRanef <- sparseMatrix(i = ii, j = jj, x = 1 * (ii == jj))
Lambdat <- .bdiag(c(list(templateFact),
                    rep(list(templateRanef), m)))
```

```
mapToModMat <- local({
    Ztx <- Zt@x
    Zwhich <- Zt@i %in% (seq_len(n) - 1)
    Zind <- Zt@x[Zwhich]
    loadInd <- 1:nFreeLoadings
    function(loads) {
        Ztx[Zwhich] <- loads[Zind]
        return(Ztx)
    }
})
```

```
mapToCovFact <- local({
    Lambdatx <- Lambdat@x
    LambdaWhich <- (n+1):length(Lambdatx)
    LindTemplate <- ii + 2 * (jj - 1) - choose(jj, 2)
    Lind <- rep(LindTemplate, m)
    function(covar) {
        Lambdatx[LambdaWhich] <- covar[Lind]
        return(Lambdatx)
    }
})
```

```
initPars <- c(covar = c(1, 0, 1),
              fixef = c(0, 0),
              loads = rep(0, m))
parInds <- list(covar = 1:3,
                fixef = 4:5,
                loads = (1:m)+5)
```

```
devfun <- mkGeneralGlmerDevfun(y, X, Zt, Lambdat,
                               weights, offset,
                               initPars, parInds,
                               mapToCovFact, mapToModMat)
```

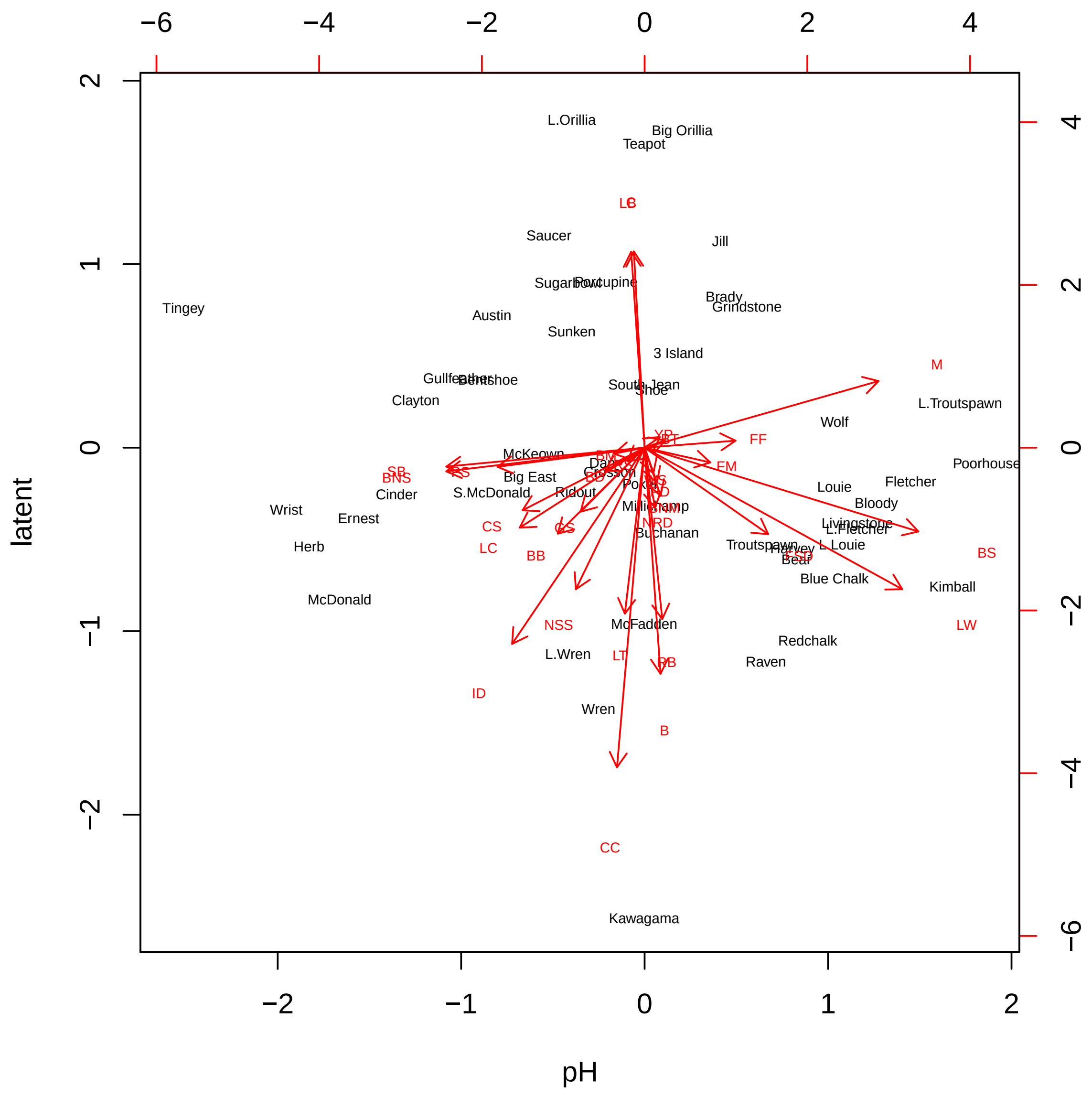
```
opt <- minqa:::bobyqa(initPars, devfun)

## Warning in commonArgs(par, fn, control, environment()):  maxfun < 10 * length(par)^2
is not recommended.

setNames(opt$par, names(initPars))

##      covar1      covar2      covar3      fixef1      fixef2      loads1
##  1.96405248 -1.47457951  0.56376024 -2.26248775  0.81707347 -0.29383447
##      loads2      loads3      loads4      loads5      loads6      loads7
##  0.16413041 -0.39519344 -4.90990508 -1.32109526 -0.91997268 -0.98342366
##      loads8      loads9     loads10     loads11     loads12     loads13
##  0.11042682 -0.29120583 -0.73886605 -0.96269062 -0.53656420 -1.32888318
##     loads14     loads15     loads16     loads17     loads18     loads19
##  3.00980086 -0.36394912 -3.47314050 -0.22687037 -2.54996759 -1.28776228
##     loads20     loads21     loads22     loads23     loads24     loads25
## -0.35752948 -3.01398890  3.01361676 -1.22801409 -2.63378068  0.10737366
##     loads26     loads27     loads28     loads29     loads30
## -0.09197036  1.02311855 -0.20881364 -2.17348057 -2.17346329
```

```
siteScores <- cbind(pH = x,
                    latent = environment(devfun)$pp$b(1)[1:n])
speciesScores <- cbind(pH = environment(devfun)$pp$b(1)[-(1:(n+m))],
                       latent = opt$par[parInds$loads])
rownames(siteScores) <- rownames(Y)
rownames(speciesScores) <- colnames(Y)
biplot(siteScores, speciesScores, cex = 0.5)
```



4 A simple phylogenetic example

Working on example in the ‘pglmer‘ helpfile. Going well ...

5 Discussion

An important point is that `mkGeneralGlmerDevfun` deals with model fitting only. However, in any application a model specification module and model presentation/inference module must also be provided. At this point, I have spent way way way too too too long thinking about how to do this in a general way. I need to get on with community ecology, and so the way forward is to have this general approach to model fitting and then treat specification and presentation separately.

As always, lot's to worry about.

- I'm expecting lots of bugs at this point
- Starting values?
- Control parameters? Can `glmerControl` be leveraged? Probably.