lme4 extras

Ben Bolker

May 26, 2012

Contents

This vignette is intended to document some extra tricks that can be used with `lme4` models. Some of them are included here because they are statistically non-rigorous and we didn't want to build them into automatic functions that could be applied unthinkingly, but recipes are supplied here for use **at your own risk** and assuming that you know what you're doing ...

1 To do

In principle, we should be able to get confidence intervals on parameters *and* confidence intervals on predictions via (1) quadratic/Wald approximation (ignoring uncertainty of θ, and possibly of u, for CIs of prediction); (2) cheesy MCMC (in this case, for LMMs, we need a way to retrieve new values of sigma and the fixed effects conditional on θ); (3) parametric bootstrap.

The basic machinery for this is (1) functions for converting among parameterizations of the random effects, i.e. from a (standard deviation, correlation) vectors to θ (concatenated Cholesky-factor vector); (2) a way to extract a deviance function from a fit (i.e. `mkdevfun`) and (3) a way to simulate values from a fit (i.e. `simulate`), along with basic components (matrix inversion, etc.).

We'll see how far I get.

2 Fit basic models

In this section we simply fit a few basic models to use as examples later on.

```
library(lme4)
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
gm1 <- glmer(cbind(incidence, size - incidence) ~
             period + (1 | herd),
             data = cbpp, family = binomial)
```

3 Quadratic confidence intervals on random effects parameters

Extract the deviance function and the ML (or REML) parameters:

```
fm1Fun <- update(fm1,devFunOnly=TRUE)
fm1_thpar <- getME(fm1,"theta")
```

Extract internal functions for converting (standard deviation, correlation) vectors to θ (concatenated Cholesky-factor) parameterization, and vice versa (**this is temporary, until we finalize the definitions/names of these functions ...**)

```
Sv_to_Cv <- lme4:::Sv_to_Cv  ## standard dev vector to cholesky
vector
Cv_to_Sv <- lme4:::Cv_to_Sv  ## vice versa
```

Test the round-trip of these functions: does converting from standard deviation scale to Cholesky scale, and back, work?

```
fm1_spar <- Cv_to_Sv(fm1_thpar,s=sigma(fm1))
all(abs(Sv_to_Cv(fm1_spar,s=fm1_spar[4])-fm1_thpar)<1e-6)

## [1] TRUE
```

A wrapper around the deviance function that we extracted.

```
fm1FunS <- function(spar) {
    thpar <- Sv_to_Cv(c(spar,NA),s=fm1_spar[4])
    fm1Fun(thpar)
}
```

Use the `numDeriv` package to compute the Hessian (second derivative) matrix at the MLE:

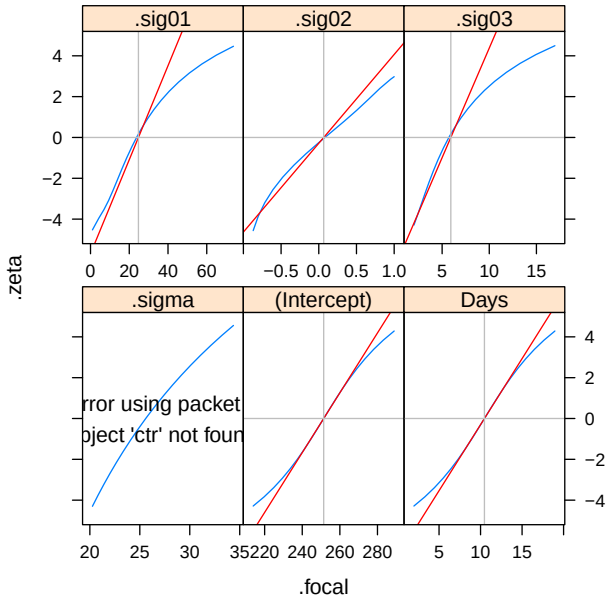
```
library(numDeriv)
h <- hessian(fm1FunS,fm1_spar[-4])
```

Variance-covariance matrix of the random-effects (standard deviation) parameters:

```
vcov_ran <- solve(h)
```

Compare profiles to their quadratic approximations, and profile confidence intervals to these approximate (Wald) confidence intervals:

```
pp <- profile(fm1)
```



FIXME: These slopes are not quite right. Why? (Are the profiles correct?) Should we be recomputing σ for each set of θ values? Think about this sometime when I have a brain ...

Profile confidence intervals:

```
(ci_prof <- confint(pp))
```

```
##                2.5 %  97.5 %
## .sig01       14.3815  37.716
## .sig02       -0.4815   0.685
## .sig03        3.8012   8.753
## .sigma       22.8983  28.858
## (Intercept) 237.6807 265.130
## Days          7.3587  13.576
```

Wald confidence intervals:

```
c(fm1_spar,fixef(fm1))+
    1.96*outer(c(sqrt(diag(vcov_ran)),
                 NA,
                 sqrt(diag(as.matrix(vcov(fm1))))),
               c(-1,1))

##                  [,1]    [,2]
##               16.2243  33.257
##               -0.3849   0.516
##                4.0969   7.747
##                    NA      NA
## (Intercept) 238.0289 264.781
## Days          7.4375  13.497
```

4 Approximate confidence intervals on predictions

5 Poor man's MCMC

```
library(MCMCpack)
```

`MCMCpack` expects a function that gives a value proportional to the log posterior density for any specified set of parameters. We can get `lme4` to give us a function for the deviance (by using `devFunOnly=TRUE`. If we assume all-improper priors (i.e. flat on the scale on which we have defined the parameters), then the log posterior density is $-D/2$:

```
fm1_metropfun <- function(x) {
  ## getME(.,"lower")?
  if (any(x<fm1@lower)) -Inf else -fm1Fun(x)/2
}
```

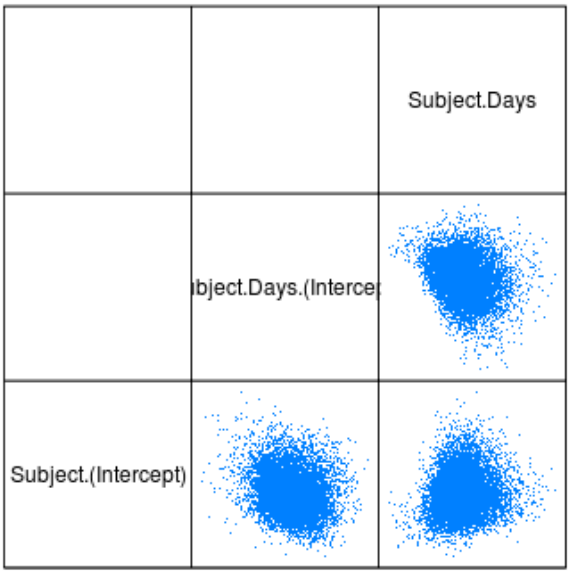
```
fm1_mcmc_out <- MCMCmetrop1R(fm1_metropfun,fm1_thpar,seed=12345)

##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.49678
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

colnames(fm1_mcmc_out) <- names(fm1_thpar)
```

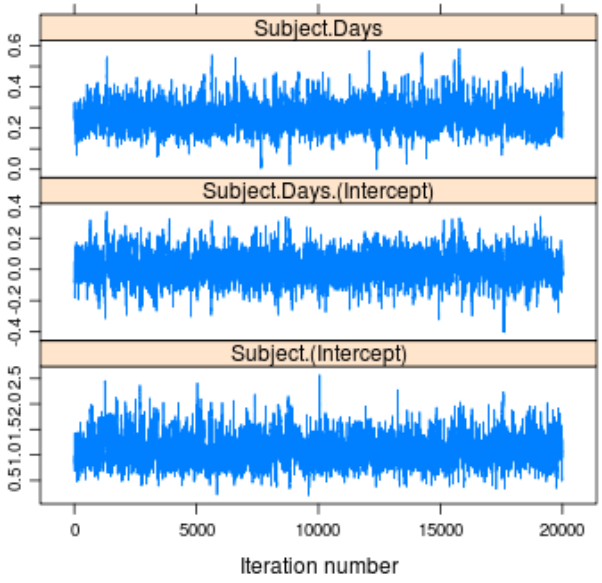
```
library(coda)
```

```
splom(fm1_mcmc_out)
```

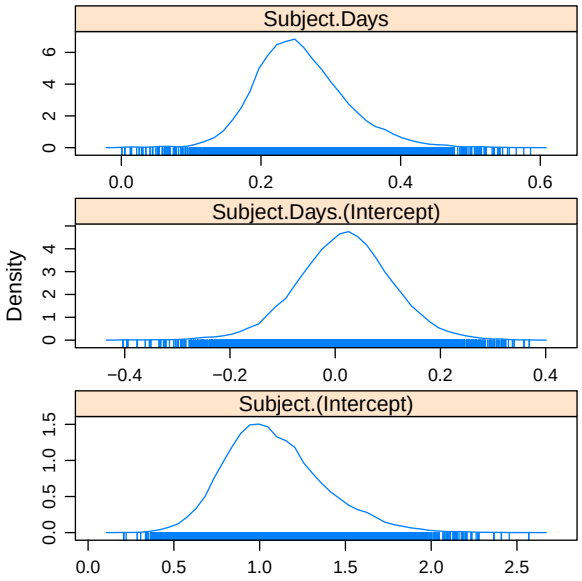


Scatter Plot Matrix

```
xyplot(fm1_mcmc_out)
```




```
densityplot(fm1_mcmc_out,layout=c(1,3))
```



```
HPDinterval(fm1_mcmc_out)
```

```
##                             lower  upper
## Subject.(Intercept)       0.5646 1.6712
## Subject.Days.(Intercept) -0.1693 0.1903
## Subject.Days              0.1416 0.3966
## attr(,"Probability")
## [1] 0.95
```

FIXME: if we want to get this on the sd/corr scale we have to figure out how to recalculate sigma for each set of θ values ... for now, just use a fixed sigma

```
sdmat <- as.mcmc(t(apply(fm1_mcmc_out,1,
                         function(x)
c(Cv_to_Sv(x,s=sigma(fm1))))))
```

Highest posterior density intervals:

```
HPDinterval(sdmat)
```

```
##      lower   upper
## 11 14.4493 42.7685
## 12 -0.5457  0.6567
## 13  4.0532 10.5283
##    25.5918 25.5918
## attr(,"Probability")
## [1] 0.95
```

Or quantile-based estimates:

```
t(apply(sdmat,2,quantile,c(0.025,0.975)))
```

```
##       2.5%   97.5%
## 11 15.4823 44.2549
## 12 -0.5194  0.6922
## 13  4.2457 10.8785
##    25.5918 25.5918
```

The latter *should* be translation-invariant, and hence (???) the same as:

```
qq <- t(apply(fm1_mcmc_out,2,quantile,c(0.025,0.975)))
apply(qq,2,function(x) VC(fm1,theta=x,format="sdcorvec"))
```

```
##                             2.5%   97.5%
## Subject.(Intercept)      15.4823 44.2549
## Subject.Days.(Intercept) -0.7561  0.4393
## Subject.Days              5.6209 11.4225
```

(Only true for variable 1, although not terribly different: think about this (i.e. the effect of $\theta\theta^T$) some more ...)

If we have the Cholesky form

$$\begin{pmatrix} c_1 & 0 \\ c_2 & c_3 \end{pmatrix}$$

and take the cross-product, we get

$$\begin{pmatrix} c_1^2 & c_1 c_2 \\ c_1 c_2 & c_2^2 + c_3^2 \end{pmatrix}$$

so it's natural that only element 1 scales as we would expect: all the other terms are not just scale translations of a single element, but combinations of multiple elements.

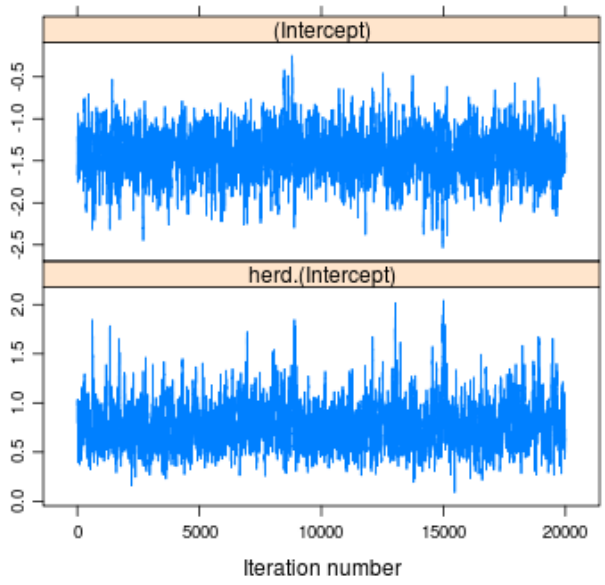
Should work for GLMMs as well:

```
gm1Fun <- update(gm1,devFunOnly=TRUE)
gm1_par <- c(getME(gm1,"theta"),fixef(gm1))
nt <- length(getME(gm1,"theta"))
gm1_metropfun <- function(x) {
  if (any(x[seq(nt)]<gm1@lower)) return(-Inf)
  r <- try(gm1Fun(x),silent=TRUE)
  if (inherits(r,"try-error")) return(-Inf)
  -r/2
}
```

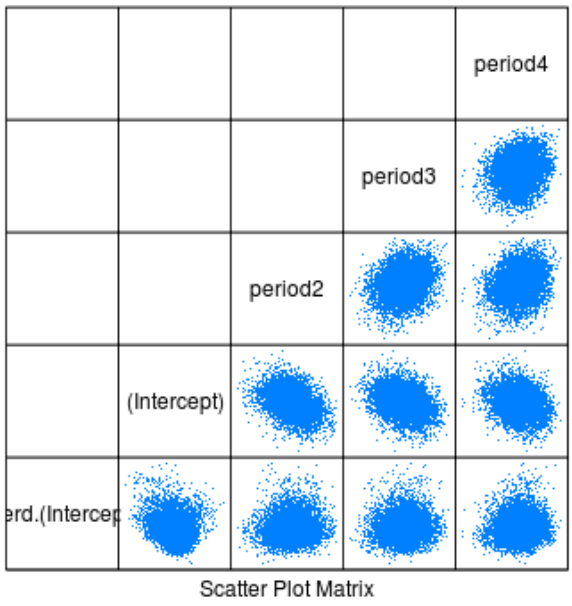
FIXME: PIRLS failure, restore when this is fixed

```
set.seed(101)
gm1_mcmc_out <- MCMCmetrop1R(gm1_metropfun,gm1_par)
```

```
colnames(gm1_mcmc_out) <- names(gm1_par)
xyplot(gm1_mcmc_out[,1:2])
```




```
splom(gm1_mcmc_out)
```



```
HPDinterval(gm1_mcmc_out)
```

```
##                     lower   upper
## herd.(Intercept)  0.3505  1.2576
## (Intercept)      -2.0225 -0.9204
## period2          -1.6026 -0.3795
## period3          -1.7782 -0.5182
## period4          -2.5010 -0.7952
## attr(,"Probability")
## [1] 0.95
```

6 Confidence intervals on predictions etc. via parametric bootstrap

FIXME: do we want a `as.data.frame.boot` function to retrieve stuff from `bootMer` output?

7 Zero-inflation via the EM algorithm

The `zipme` function, adapted from code by Mihoko Minami and Cleridy Lennert, is available . . . its form is

```
zipme <- function(cformula, zformula, cfamily=poisson,
                  data, maxitr=20, tol=1e-6, verbose=TRUE) {
    ...
}
```

Where:

cformula the conditional formula, i.e. the `glmer` formula for the GLMM part of the model

zformula the zero-inflation formula, i.e. the `glm` formula for the zero-inflation probability. The response variable *must* be specified as `..z`

cfamily the `family` variable for the conditional part of the model

data a data frame

maxitr the maximum number of EM iterations

tol convergence tolerance

verbose print out information on EM iterations?

An example of EM use:

```
set.seed(101)
zprob <- 0.2
nblock <- 20
ntot <- 500
nperblock <- 25
d <-
data.frame(x=runif(ntot),f=factor(rep(1:nblock,each=nperblock)))
u <- rnorm(nblock,sd=0.75)
d$eta <- with(d,1+2*x+u[as.numeric(f)])
d$resp <- ifelse(runif(ntot)<zprob,0,rpois(ntot,exp(d$eta)))
```

```
zfit1 <- zipme(resp~x+(1|f),..z~1,data=d,verbose=FALSE)
```

```
plogis(coef(zfit1$zfit))  ## cf true value of 0.2

## (Intercept)
##      0.2229

plogis(confint(zfit1$zfit))

## Waiting for profiling to be done...
##  2.5 % 97.5 %
## 0.1879 0.2607

fixef(zfit1$cfit)  ## true value: (1,2)

## (Intercept)           x
##      0.9523      2.0065
```

FIXME: suppress `non-integer #successes in a binomial glm!` warnings ... preferably without suppressing other possible warnings. Will it be possible to adapt this for zero-inflated binomial GLMMs, or will the dual use of `weights` get in the way?