GLMM on symbiont effects on coral predation

Ben Bolker

June 12, 2009

1 Preliminaries

The purpose of this document is to explore/explain some of the nitty-gritty details of fitting, and making inferences from, GLMMs in R. It may turn into a "real paper" at some point; we'll see.

The data are from Adrian Stier and Sea McKeon, and represent trials of *Culcita* sea stars attacking coral that harbor differing combinations of protective symbionts (crabs and shrimp). The design is a randomized complete block design with some replication (2 replications per treatment per block, 4 treatments (no symbionts, crabs alone, shrimp alone, both crabs and shrimp), each of these units of 8 repeated in 10 blocks). The data represent a reasonably typical small-scale ecological field experiment.

Some issues that do *not* arise in this example:

- extremely large data sets (requiring concerns for computational efficiency and stability)
- extremely small (5–6 or less) numbers of random effect units that have to be dealt with in some special way (treatment as fixed effects or supply of a Bayesian prior (Gelman, 2006))
- crossed random effects
- overdispersion
- spatial or temporal correlations that decrease with separation in space or time
- other interesting correlation structures (phylogenetic?)
- lack of balance

The basic data can be reduced, for the purposes of this exercise, to a single treatment (`ttt`) [which consists of combinations of different symbionts: crab, shrimp, both or neither, but never mind]; a binary response (`predation`); and a blocking factor (`block`).

Data entry/summary:

```
> x=read.csv("culcitalogreg.csv",
+   colClasses=c(rep("factor",2),
+     "numeric",
+     rep("factor",6)))
> ## abbreviate slightly
> levels(x$ttt.1) <- c("none","crabs","shrimp","both")
```

Adjust contrasts for the treatment, to compare (1) no-symbiont vs symbiont cases, (2) crabs vs shrimp, (3) effects of a single pair/type of symbionts vs effects of having both:

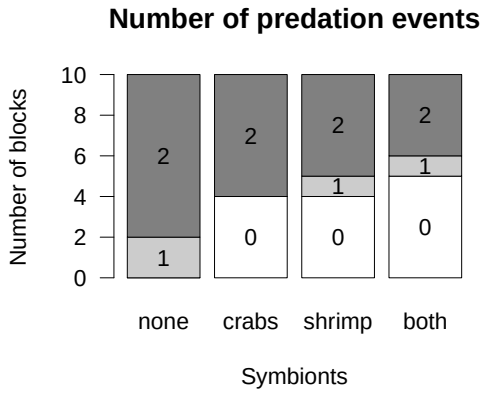
```
> contrasts(x$ttt)=
+   matrix(c(3,-1,-1,-1,
+            0,1,-1,0,
+            0,1,1,-2),
+          nrow=4,
+          dimnames=list(c("none","C","S","CS"),
+            c("symb","C.vs.S","twosymb")))
```

2 Plots

Two ways to plot the data:

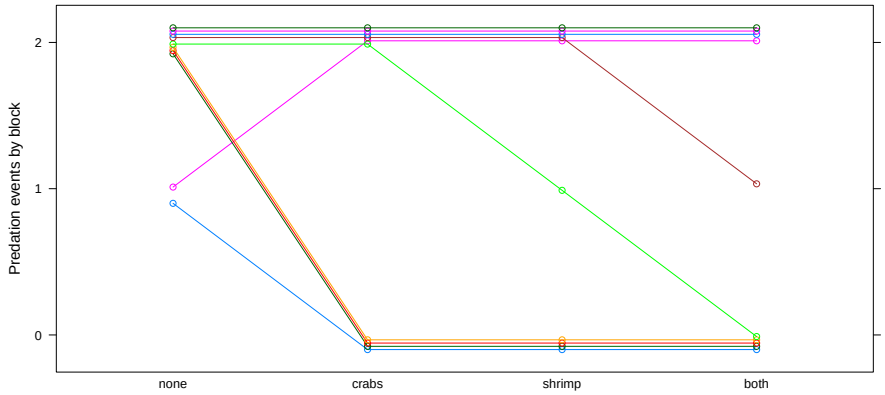
1. A basic barplot that loses information on blocks but gives the distribution of numbers of predation events per block:

```
> library(reshape)
> m  <- melt(x[,c(1,4,3)],id.vars=1:2)
> m3 <- recast(m,ttt.1+block~variable,fun.aggregate=sum)
> p  <- with(m3,table(predation,ttt.1))
> op <- par(las=1,cex=1.5,bty="l")
> bb <- barplot(p,ylab="Number of blocks",xlab="Symbionts",
+               main="Number of predation events",
+               col=gray(c(1,0.8,0.5)))
> ## the rest of this is fanciness to plot the values
> ##  within the bar segments (rather than just
> ##  a plain old legend)
> bloc <- apply(p,2,cumsum)
> midb <- rbind(bloc[1,]/2,
+               (bloc[1,]+bloc[2,])/2,
+               (bloc[2,]+bloc[3,])/2)
> text(bb[col(p)][p>0],midb[p>0],c(0,1,2)[row(p)][p>0])
> par(op)
```



2. A (too?) clever strip plot that shows the pattern for each block:

```
> library(lattice)
> ctab <- with(x,as.data.frame.table(tapply(predation,list(block,ttt.1),sum)))
> names(ctab) <- c("block","ttt","pred")
> ctab$rblock <- with(ctab,
+                     reorder(block,pred))
> ctab$jpred <- ctab$pred+seq(-0.1,0.1,length.out=10)[ctab$rblock]
> print(stripplot(jpred~ttt,groups=block,data=ctab,
+           type=c("p","l"),ylab="Predation events by block",
+                 scales=list(y=list(tick.number=3))))
```



A simpler plot, or one based on model fits?

3 Basic fits

should I try to test the `ttt:block` *interaction? Probably will be a mess since within-block/treatment replication is so limited (two binary samples), but worth*

it for completeness?

3.1 glm (base R)

Fit without blocking (i.e., no random effects):

```
> mod1 = glm(predation ~ ttt, binomial, data = x)
```

3.2 glmer (lme4)

Fitting with the default Laplace approximation and with adaptive Gauss-Hermite quadrature (abbreviated sometimes as AGQ and sometimes as (A)GHQ):

```
> library(lme4)
> ## Laplace
> mod2 <- glmer(predation~ttt+(1|block),family=binomial,data=x)
> ## AGQ
> mod2B <- glmer(predation~ttt+(1|block),family=binomial,
+                nAGQ=8,data=x)
```

3.3 glmmML (glmmML)

The `glmmML` function is in a separate package (the `glmmML` package, surprisingly enough). It handles only random intercept models with a single blocking factor (I chose `nAGQ=8` above to match the default number of AGQ points (8) for `glmmML` with `method="ghq"`).

```
> library(glmmML)
> ## Laplace (default)
> mod3=glmmML(predation~ttt,family=binomial,data=x,cluster=block)
> ## AGQ (=GHQ) with 8 points
> mod3B=glmmML(predation~ttt,family=binomial,data=x,cluster=block,
+   method="ghq")
```

Fitting with Laplace gives a warning: `Hessian non-positive definite. No variance!` This means it can fit the model but fails to get (approximate) standard errors for the parameters. We'll decide later whether we want to worry about this.

3.4 glmm (repeated)

Jim Lindsey's `repeated` package (available not from CRAN, but from `http://www.commanster.eu/rcode.html`) contains the `glmm` function:

```
> library(repeated)
> mod7=glmm(cbind(predation,1-predation)~ttt,family=binomial,
+   data=x,nest=block,points=8)
> mod7.ci = confint(mod7) ## not reliable!
```

Have to specify predation as $(k, N-k)$, `glmm` apparently doesn't recognize binary data as such. `glmm` has a `confint` method, which is supposed to compute profile confidence intervals, but it gives ridiculously optimistic results, so we'll fall back on the estimated standard errors for CIs below.

3.5 glmmADMB

```
> library(glmmADMB)
```

```
================================================================================

Welcome to glmmADMB

================================================================================
```

```
> mod4 = glmm.admb(predation ~ ttt, random = ~1, group = "block",
+     data = x, family = "binomial", link = "logit")
```

Can supposedly improve the Laplace fit with importance sampling, but I didn't really understand the meaning of the `impSamp` parameter from the documentation, and got errors when I tried setting it > 0 (after step 2: `The function maximizer failed`). (This is mentioned, but not discussed much in, the `glmm.admb` help page: probably need to see ref therein: Skaug and Fournier (2004), which has a line about importance sampling, further referencing Skaug (2002).

3.6 MCMCglmm

Would have tried `MCMCglmm`, but it is impossible to run `MCMCglmm` models *without* individual-level random effects — which in this case (with binary data) causes convergence problems.

3.7 glmmBUGS

(Have to do this next bit carefully: both of the next two approaches load the `nlme` package (used by `glmmPQL`). Since `nlme` and `lme4` conflict, we have to unload/reload the packages.)

```
> detach("package:lme4")
> library(MASS)
> library(nlme)
```

The `glmmBUGS` package is supposed to translate a mixed-model statement into BUGS code which can then be run via `R2WinBUGS`.

This doesn't quite work — first because the binomial-family specification doesn't get translated correctly (a `dbin(x)` specification should read `dbin(x,1)`), and then from an unknown BUGS problem ...

```
> library(glmmBUGS)
> gg1 <- glmmBUGS(predation ~ ttt, data = x, effects = "block",
+     family = "binomial")
> startingValues <- gg1$startingValues
> source("getInits.R")
> library(R2WinBUGS)
> gg2 <- bugs(gg1$ragged, getInits, parameters.to.save = names(getInits()),
+     working.directory = getwd(), model.file = "model.bug")
```

3.8 glmmPQL

Also worth trying with `glmmPQL` (which uses a less reliable algorithm, but is more flexible etc.):

```
> mod5 = glmmPQL(predation ~ ttt, random = ~1 | block, family = binomial,
+     data = x)
> pqlfix <- fixef(mod5)
> pqlfix.se <- summary(mod5)$tTable[, "Std.Error"]
> pqlran <- as.numeric(VarCorr(mod5)["(Intercept)", "StdDev"])
```

```
> detach("package:nlme")
> library(lme4)
```

3.9 BUGS

Or, we can simply write our own `BUGS` file:

```
model {
  for (i in 1:N) {
    lprob[i] <- b[1]*mm[i,1]+b[2]*mm[i,2]+
                        b[3]*mm[i,3]+b[4]*mm[i,4]+eps[block[i]]
    prob[i] <- 1/(1+exp(-lprob[i]))
    ## for JAGS compatibility, don't use logit(prob[i]) <-
    ## also appears to improve stability!
    predation[i] ~ dbin(prob[i],1)
  }
  for (i in 1:nblock) {
     eps[i] ~ dnorm(0,tau.block)
  }
  ## priors
  for (i in 1:4) {
     b[i] ~ dnorm(0,0.01)
  }
  tau.block ~ dgamma(0.01,0.01)
  sd.block <- 1/sqrt(tau.block)
}
```

```
> library(R2WinBUGS)
```

```
> mm <- model.matrix(predation ~ ttt, data = x)
> bugsdata <- list(mm = mm, N = nrow(x), nblock = length(levels(x$block)),
+     block = as.numeric(x$block), predation = x$predation)
> bugsinit <- list(list(b = c(0.5, rep(0, 3)), eps = rep(0, 10),
+     tau.block = 1), list(b = c(0.5, rep(-0.5, 3)), eps = rep(0,
+     10), tau.block = 1), list(b = c(0.5, rep(0.5, 3)), eps = rep(0,
+     10), tau.block = 1))
> b1 <- bugs(data = bugsdata, inits = bugsinit, parameters = c("b",
+     "eps", "tau.block", "sd.block"), model.file = "culcita_bugs.txt",
+     n.chains = 3, n.iter = 4000, working.directory = getwd())
```

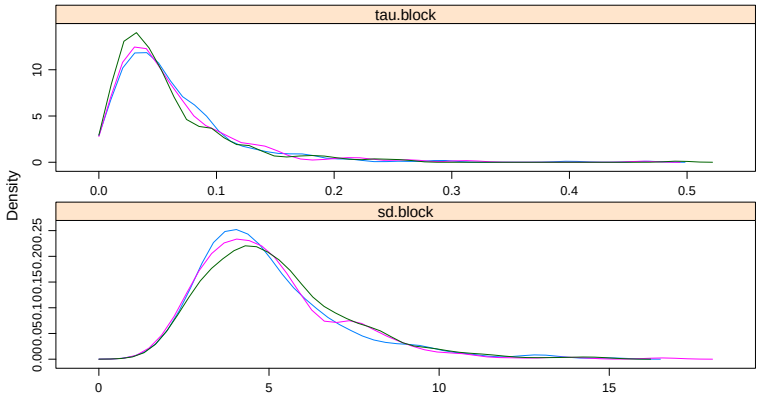
We can get 95% credible intervals ...

```
> library(emdbook)
> b2 <- as.mcmc.list(b1)
> b3 <- lump.mcmc.list(b2)
> head(coda::HPDinterval(b3), 4)
```

```
       lower  upper
b[1] -1.9910 5.3590
b[2]  0.5678 2.1790
b[3] -1.6150 0.9084
b[4] -0.2228 1.3590
```

... or examine the posterior densities (of block effect expressed as precision (τ) or standard deviation (σ) ...

```
> b4 <- lapply(b2, function(x) {
+     x[, c("sd.block", "tau.block")]
+ })
> class(b4) <- "mcmc.list"
> print(densityplot(b4, from = 0))
```



3.10 glmmAK

The `glmmAK` package is a lesser-known alternative. Its range of allowed models is small (logistic and Poisson regression, plus multinomial/ordinal logistic regression), and its interface is a bit quirky ... I think the specification below gives a model that is equivalent to the BUGS model above ...

```
> library(glmmAK)
> mod6 = cumlogitRE(y = x$predation, x = mm[, -1], intcpt.random = TRUE,
+     cluster = x$block, prior.fixed = list(mean = 0, var = 100),
+     prior.random = list(Ddistrib = "gamma", Dshape = 0.01, DinvScale = 100),
+     nsimul = list(niter = 6000, nthin = 40, nburn = 2000))
```

```
> mod6fixchain <- read.table("betaF.sim", header = TRUE)
> mod6rechain <- read.table("varR.sim", header = TRUE)
> b6 = as.mcmc(cbind(mod6fixchain, sqrt(mod6rechain[, 1])))
```

SAS, ASREML/Genstat, others?

3.11 Comparisons

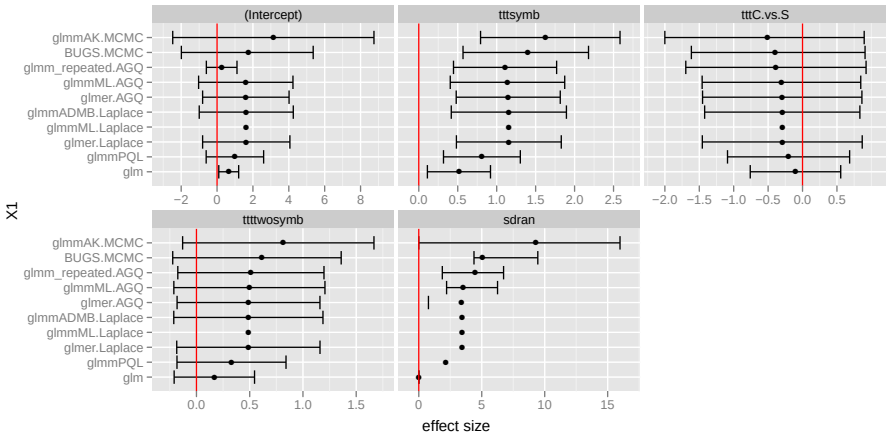
(Ugly code suppressed here — look at the original Sweave file if you want the gory details on extracting coefficients, standard errors, etc. from the model fits.)

	(Intercept)	tttsymb	tttC.vs.S	ttttwosymb	sdran
glm	0.651	0.516	-0.102	0.168	0.000
glmmPQL	0.995	0.810	-0.203	0.329	2.154
glmer.Laplace	1.628	1.156	-0.294	0.487	3.437
glmmML.Laplace	1.628	1.156	-0.294	0.487	3.437
glmmADMB.Laplace	1.628	1.156	-0.294	0.488	3.448
glmer.AGQ	1.605	1.148	-0.295	0.489	3.401
glmmML.AGQ	1.602	1.139	-0.306	0.497	3.511
glmm_repeated.AGQ	0.257	1.108	-0.386	0.511	4.471
BUGS.MCMC	1.748	1.400	-0.400	0.613	5.066
glmmAK.MCMC	3.148	1.627	-0.508	0.813	9.287

Conclusions: ignoring the random effect underestimates all of the fixed effect sizes (this is more or less as expected); PQL underestimates the standard deviation of the random effect by about 50% relative to Laplace/AGQ. Laplace and AGQ give very similar answers: the Laplace results are identical across all 3 packages tried. AGQ makes a small difference, but even the direction of change depends on which implementation you use. (I wouldn't say that any of these differences are large enough to concern us in practice in this case). The two MCMC-based methods, `BUGS` and `glmmAK`, give substantially higher estimates of the fixed and random effects (this is not due to asymmetry of the posterior distributions/differences between the mode and the mean of the posterior).

By analogy with randomized complete block designs for LMMs, the "denominator df" for this problem should be $27 = (10 - 1) \times (4 - 1)$: see for example `http://www.tfrec.wsu.edu/ANOVA/RCBsub.html`. Add confidence intervals, crudely based on Wald intervals with 27 df (except for BUGS/`glmmAK` results, which are credible intervals):

(`glmmADMB` gives a standard deviation of the variance estimate, but $\hat{\sigma}^2 = 11.9$ and its standard deviation is supposed to be 8.8, so I can't really use it to construct sensible confidence intervals.)



4 Inference

4.1 Wald Z tests

These are given (for fixed effects) by both packages as `Pr(>|z|)` in the standard model print-out (they're missing for glmmML/Laplace because there was a computational problem) and suggest that there is an effect of having any symbionts at all (`ttt1`), but that crabs are not different from shrimp (`ttt2`) and that having two symbionts isn't any better than having one (`ttt3`). Replicating the results for the glmmML code here (this appears in `summary(mod3B)`, but it doesn't look like this function call returns anything useful — it just prints the object in the usual way).

```
> csd <- mod3B$coef.sd
> z.table <- cbind(coef(mod3B), csd, coef(mod3B)/csd, pnorm(abs(coef(mod3B))/csd,
+     lower.tail = FALSE) * 2)
> colnames(z.table) <- c("coef", "se(coef)", "Z", "Pr(>|Z|)")
> round(z.table, 4)
```

```
              coef se(coef)       Z Pr(>|Z|)
(Intercept)  1.6017   1.2818  1.2496   0.2115
tttsymb      1.1387   0.3581  3.1801   0.0015
tttC.vs.S   -0.3062   0.5615 -0.5454   0.5855
ttttwosymb   0.4971   0.3458  1.4378   0.1505
```

You can get equivalent fits from the `glmer` fit (`summary(mod2B)@coefs`) or the `glmm` fit (`coef(summary(mod7))`).

The main problems with this analysis are (1) it doesn't take account of the residual degrees of freedom/uncertainty in standard error estimates (asymptotic resullts) and (2) it's a fairly crude, Wald-based estimate — not clear whether the null distribution of the parameter estimates divided by their standard errors is really normal Z (or t) in this case (3) it tests parameters (contrasts) one at a time. (I've written a `waldF()` function to compute Wald F tests — it can be found in `glmmfuns.R`).

4.2 Wald t tests

One of the harder parts of GLMMs (in the traditional Wald- or F-testing approach) is figuring out the appropriate degrees of freedom. As mentioned above, I will use $(10-1)\times(4-1)=27$ df as the denominator df (may check later that this actually fits the null Wald distribution ...). (For the same model, `nlme`'s "inner-outer" approach (Pinheiro and Bates, 2000) gives 37 df instead — but this is only the difference between a critical value of 2.052 for 27 df and 2.026 for 37 df, vs. the magic 1.96 for a Z test.)

None of the tests we did cross the magic $p < 0.05$ line in either direction under these circumstances (results are similar for `glmer`):

```
> round(Z.to.t(z.table, 27), 3)
```

```
              coef se(coef) t value Pr(>|t|)
(Intercept)  1.602    1.282   1.250    0.222
tttsymb      1.139    0.358   3.180    0.004
tttC.vs.S   -0.306    0.562  -0.545    0.590
ttttwosymb   0.497    0.346   1.438    0.162
```

`Z.to.t()` is a small function to hack Z tables (provided e.g. by `glmer` and `glmm` into t tables, *provided that you can specify the appropriate "denominator" degrees of freedom*.

This is of course not using any of the fancy technology (which Doug Bates distrusts) for computing corrected degrees of freedom (Satterthwaite or Kenward-Roger) — but which may not be necessary here in any case.

4.3 LRT

It is possible, but strongly advised against, to use Likelihood Ratio tests to make inferences about fixed effects (Pinheiro and Bates, 2000) — it might be the recommended way to make inferences about random effects. (Fabian Scheipl's `RLRSim` package is probably the best way to make inferences on significance of random effects in simple LMMs, but it handles neither GLMMs nor more complex (crossed, multi-random-factor, etc.) LMMs.)

One problem with testing the random effects here is that `glmer` can't fit models with no random effects at all. In principle we can compare the log-likelihood from glmer (with block effects) to that from glm (without block effects). In practice we have to be *very careful* when comparing log-likelihoods calculated by different functions, because they can incorporate (or drop) different additive constants (which makes no difference when we compare results computed by the same function, but can screw us up when we compare results computed by different functions).

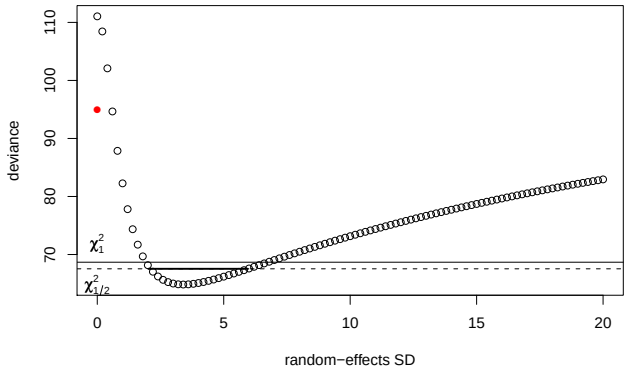
Ways to check/test this: (1) look at the code for both functions (ugh); (2) simulate data with a very small or zero block effect, fit it with glmer and glm (presumably getting a very small/zero estimate of the random effect and hence similar log-likelihoods), and compare; (3) set the variance parameter in the glmer fit to zero and re-evaluate (this is what I do below).

Using the `varprof` function (too ugly to show here — defined in `glmmfuns.R`) which computes a deviance profile by varying the value of the random-effects standard deviation, we can show the profile.

```
> profres <- varprof(mod2B)
```

since I'm not that interested in inference on random effects in this case, this section is a little dusty/could use some work

In the figure below, the solid line is the critical level for the LRT based on a χ^2 distribution with 1 df, the dashed line is for $\chi^2_{1/2}$ (see "boundary effects" in the GLMM paper for why $\chi^2_{1/2}$ may be more appropriate). The p value based on this distribution is 0.



The deviance calculated by glmer for $\sigma = 0$ is 111.05; the deviance calculated by glm is 94.97 [shown in red in the figure], so the computed log-likelihoods do *not* match — we were right to be careful. (That, or I made some other mistake here.)

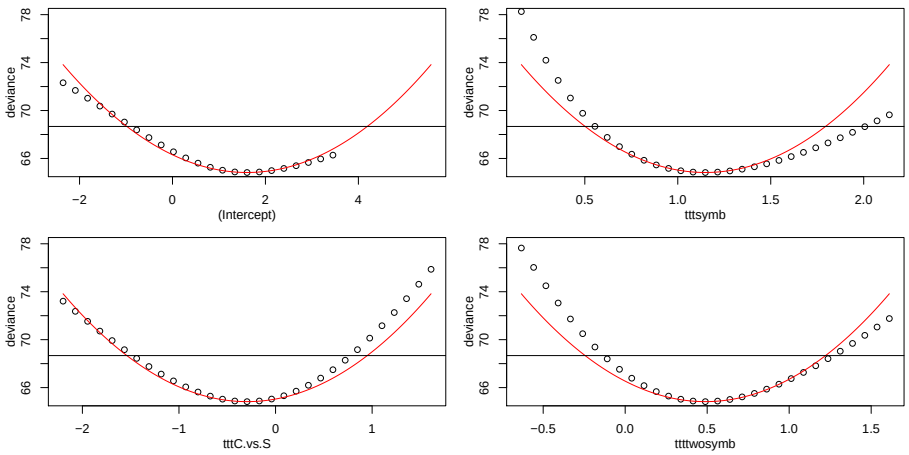
The 95% profile confidence limits on the random-effects sd, based on $\chi^2_{1/2}$ (see below for why I think this is the best answer) are {2.02,5.96} (see ugly invisible code above, taken out of one of the `mle` profiling functions, for how to get these numbers). (This value takes boundary effects into account, but not finite sample size — from that point of view, it may be a little bit conservative.)

We can also get likelihood profiles for the fixed effects. The basic recipe is to leave one fixed effect parameter (the one being profiled) out of the model; use the `offset` argument to `glmer` to set it to a particular (fixed) value; and re-fit the model. For factors this is a little bit trickier because it's hard to leave one parameter out of the model. The way I ended up doing this was a bit of a hack — treating the model as a regression analysis on the individual columns of the model matrix (it would be easier if one could pass a model matrix directly). Right now the code is fairly ugly and fairly specific to this problem, it could certainly be cleaned up and generalized ...

Plots of the deviance profiles for each fixed-effect parameter. Red lines are quadratic approximations based on the local curvature, as used in the Wald

Z tests; horizontal lines are the LRT cutoff — for quadratic approximations, equivalent to the Wald Z 95% critical value.

```
> vals <- fixprof(mod2B)
```



The profile for the intercept ran into some numerical difficulties beyond about 3.5 — these are simply not shown on the plot.

compare likelihood profile confidence intervals (based on asymptotic χ^2_1, or on a scaled $F_{1,27}$ or $F_{1,??}$?) with Wald Z/t

4.4 AIC

Finish this!

Both `glmmML` and `glmer` fits will give AIC values, although it is a bit tricky to get them — I had to write the extractor methods for `glmmML` myself.

```
> library(stats4)
> sapply(list(mod2, mod3, mod2B, mod3B), AIC)
```

```
      ML                ML
70.70584 70.70584 74.82970 70.30988
```

```
> sapply(list(mod2, mod3, mod2B, mod3B), logLik)
```

```
       ML                  ML
-30.35292 -30.35292 -32.41485 -30.15494
```

Same issues about comparing log-likelihoods from different functions apply here, too, but with additional complication – parameter counting (as well as likelihood constants) can differ among functions, e.g. whether one includes an implicitly estimated variance term or not. So you have to be careful. Once you have the log-likelihoods, though, it is fairly easy to figure out how a given function is counting — easier than working out the log-likelihood comparisons (see above).

5 Randomization/simulation approaches

The idea here is to simulate from the null model (zero block effect, or zero for some treatment effect or combination of effects), refit the full model, get some summary statistics – the log-likelihood or deviance (or Z or t statistic) – and repeat. For many (1000 or so) simulations these statistics will give the null distribution, which you can then compare against the observed values.

In general, this is fairly simple (not necessarily even worth packing into/hiding behind a function): fit the reduced model; use `simulate` to generate new realizations from the reduced model; for each realization, use `refit` to refit the full model and (possibly) the reduced model to the data simulated from the reduced model; and compute some sort of summary statistics (deviance/log-likelihood difference, Z/t statistic, ?).

5.1 Random effects

Also a little dusty — redo along the lines of the fixed effect example below?

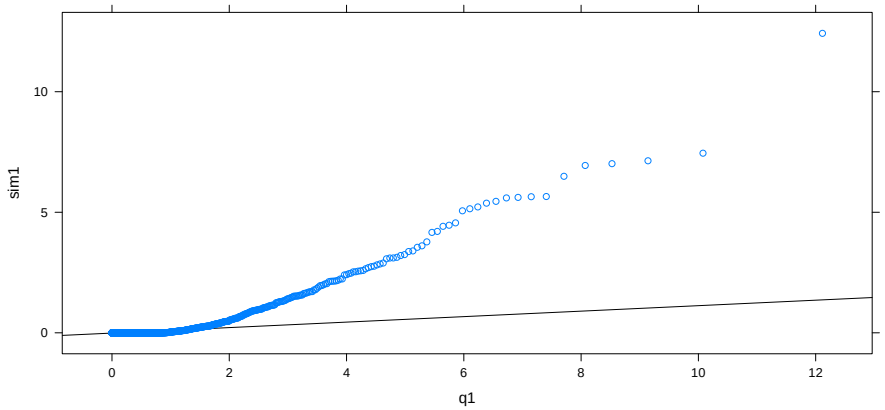
For dropping the only random effect from the model we can do almost the same thing, but we have a little problem — we need to simulate from a `glm` fit instead of a reduced `glmer` fit, because `glmer` can't fit models with no random effects. (In this case I had to write `simulate.glm` since it didn't exist: it's in `glmmprof.R`.)

As promised, the code is actually pretty simple. The extra complexities are:

- a little bit of code to load previously computed results from a file/save the results to a file (since this is computationally intensive);
- because `glm` computes likelihood differently from `glmer`, we use the `zerodev` function (defined above) to return the value of the deviance when a model is refitted with the random-effects standard deviation forced equal to zero — rather than what we would usually do, refitting a `glm` model to the new realization and computing the deviance that way

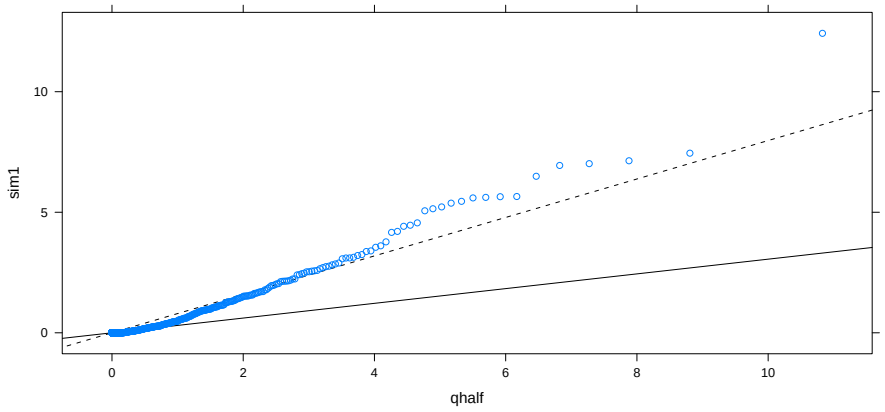
```
> if (file.exists("glmersim1.RData")) {
+     load("glmersim1.RData")
+ } else {
+     svals <- simulate(mod1, nsim = 1000)
+     t0 <- system.time(sim1 <- apply(svals, 2, function(x) {
+         r <- refit(mod2, x)
+         zerodev(r) - deviance(r)
+     }))
+     save("svals", "t0", "sim1", file = "glmersim1.RData")
+ }
```

(this took 26 seconds on my laptop). The following figure compares the χ^2_1 distribution with the actual distribution of deviance differences:



The next figure compares the observed null distribution to the $\chi^2_{1/2}$ distribution. (The quantiles of this distribution are equivalent to $\chi^2_1(1-2\alpha)$, e.g. the 95% quantile is equal to the 90% quantile for χ^2_1. The `qchibarsq` function from the `emdbook` package is a convenience function that implements this distribution.)

The solid line, which looks really bad, intersects the (0.25,0.75) quantiles — this distribution is extremely skewed, so most of the points actually fall in the lower left-hand corner. The dashed line (admittedly cherry-picked) intersects the (0.25,0.95) quantiles and seems OK (*this varies by realization — it looked better for a previous set of simulation results ...*)



The critical values/quantiles from the simulated and the $\chi^2_{1/2}$ distribution:

```
> cbind(sim = quantile(sim1, c(0.9, 0.95, 0.99)), qhalf = qchibarsq(c(0.9,
+     0.95, 0.99)))

         sim    qhalf
90% 1.121128 1.642374
```

```
95% 2.157363 2.705543
99% 5.454854 5.411894
```

```
> obsdev <- c(zerodev(mod2B) - (-2 * logLik(mod2B)))
```

Estimated p values for the random effect from simulated and $\chi^2_{1/2}$ (both very very small):

```
> pchibarsq(obsdev, lower.tail = FALSE)
```

```
          ML
5.295299e-12
```

```
> mean(c(sim1, obsdev) >= obsdev)
```

```
[1] 0.000999001
```

This is essentially 1/1001 — it is general practice (ref???) to include the observed value in the set of null-hypothesis values, which means that the minimum p-value achieved in this way is $1/(N+1)$.

5.2 Fixed effects

Permutation

A basic function (maybe there is a better way?) to permute predation events within blocks:

```
> permfun <- function(x) {
+   r <- do.call("rbind",
+                lapply(split(x,x$block),
+                       function(z) {
+                         z$predation <- sample(z$predation)
+                         z
+                       }))
+   ## to use refit(), we have to make sure that
+   ## we reassemble into the proper order matching the original
+   ## data -- better/cleaner way to do this?
+   r[order(r$ttt,as.numeric(as.character(r$block))),]
+ }
```

Check that it preserves the structures we want:

```
> ## experimental design --- ttt * block
> with(permfun(x),table(block,ttt))
```

```
     ttt
block 1 2 3 4
   1  2 2 2 2
   10 2 2 2 2
   2  2 2 2 2
   3  2 2 2 2
```

```
  4  2 2 2 2
  5  2 2 2 2
  6  2 2 2 2
  7  2 2 2 2
  8  2 2 2 2
  9  2 2 2 2
```

```
> ## total predation events
> c(sum(x$predation),sum(permfun(x)$predation))
```

```
[1] 50 50
```

```
> ## predation events per block
> rbind(tapply(x$predation,list(x$block),sum),
+       tapply(permfun(x)$predation,list(x$block),sum))
```

```
     1 10 2 3 4 5 6 7 8 9
[1,] 1  7 2 2 2 5 7 8 8 8
[2,] 1  7 2 2 2 5 7 8 8 8
```

A function to fit the full and reduced model to a new predation vector and return the difference in deviance and the Wald statistics:

```
> statfun <- function(x, mod0, mod1) {
+     mod1 = try(refit2(mod1, x))
+     mod0 = try(refit2(mod0, x))
+     if (inherits(mod0, "try-error")) {
+         if (inherits(mod1, "try-error")) {
+             return(rep(NA, 5))
+         }
+         else return(c(NA, summary(mod1)@coefs[, "z value"]))
+     }
+     likratio <- c(-2 * (logLik(mod1) - logLik(mod0)))
+     wald <- summary(mod1)@coefs[, "z value"]
+     c(likratio, wald)
+ }
```

A function to do Monte Carlo simulation of the predation events from the null model:

```
> mcfun <- function(x) {
+     my.mer.sim(mod0)
+ }
```

Run permutations:

```
> source("~/projects/glmm/glmmfuns.R")
> permvals <- as.data.frame(t(replicate(2000, statfun(permfun(x)$predation,
+     mod0, mod1))))
```

Run MC simulations:

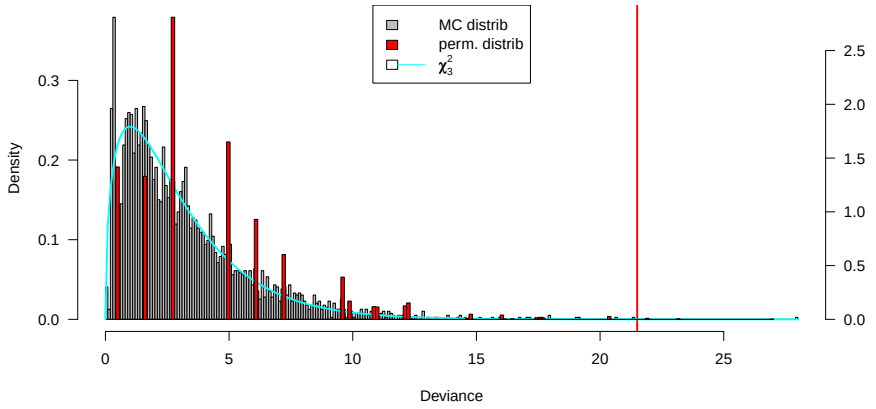
```
> mcvals <- as.data.frame(t(replicate(4000, statfun(mcfun(x, mod0,
+     mod1)))))
```

```
> load("culcita_mcvals_1.RData")
> load("culcita_permvals_1.RData")
```

Even after we throw out fits with warning messages there are 2 or 3 really awful outliers, but we will just ignore these (doesn't make too much difference to p-values anyway):

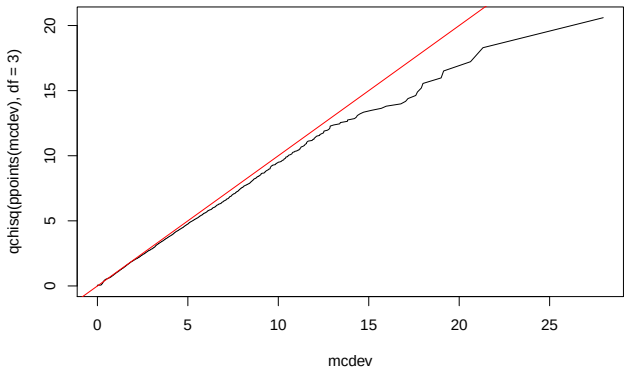
Analyze MC results:

```
> mod1 = glmer(predation ~ ttt2 + (1 | block), data = x, family = binomial)
> mod0 = update(mod1, . ~ . - ttt2)
> mcdev <- -mcvals$ML[abs(mcvals$ML) < 50]
> permdev <- na.omit(-permvals$ML)
> obsdev <- c(2 * (logLik(mod1) - logLik(mod0)))
```



The gray bars are the observed Monte Carlo distribution for simulations of the null hypothesis —

```
> qqplot(mcdev, qchisq(ppoints(mcdev), df = 3), type = "l")
> abline(0, 1, col = 2)
```

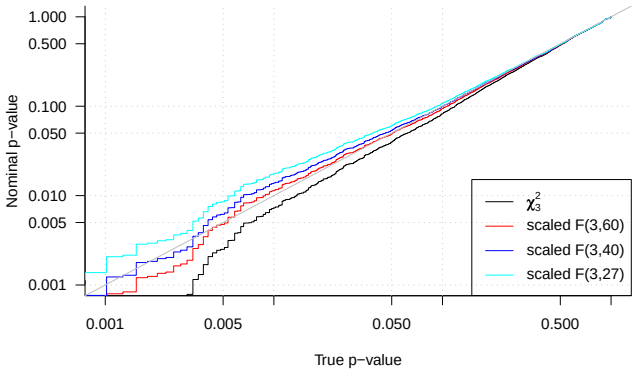


Permutation distribution only has 40 unique values (once we account for numeric errors on the order of 1e-9 from fitting models with data in different orders . . .)

```
> mean(obsdev <= pchisq(obsdev, 3, lower = FALSE))
```

```
[1] 0
```

It turns out that the LRT p-value is anticonservative in this finite-sample case, as previously reported for typical small-sample LMMs by Pinheiro and Bates (2000). That is, the reported or nominal p value (y axis) is generally below the Monte Carlo p value (x axis) (or in other words, the (black) χ^2_3 line is always below the (gray) 1:1 line in the figure below.



Also drawn in this figure are the F distributions (scaled to match the χ^2_3 distribution in the large-denominator-df limit) for varying denominator df. The line based on $df_2 = 27$ (expectation from classic ANOVA tables). The lines for $df_2 = 40$ and $df_2 = 60$ are closer, but none of them seem to match precisely —

as Bates would say, we don't necessarily have a reason to believe that the null distribution of deviance differences is F distributed in any case.

do the same kinds of checks for Wald statistics and for likelihood ratio tests for dropping individual contrasts

6 To do

- more AIC stuff, model averaging?
- incorporate profile confidence limits (backspline) in varprof code? profiling code could be tested and polished quite a bit: (1) better ways to manipulate formulas and model matrices; (2) test on other models, more robust, work out what to do about zero-intercept models, binomial responses with multiple columns, etc. (3) make profile objects match structure of glm/mle profile objects, so we can use the code for plotting, confidence intervals etc.
- structured bootstrapping (by block) example? (Can easily bootstrap on blocks: however, this will be a little slower than resampling just the response vector; we will need to refit every time. Sampling design will be such that we will have unbalanced data — fewer blocks, some blocks overrepresented.) Start by splitting data by block, sample blocks with replacement, replicate as necessary, squash back together, fit ...
- ASReML, SAS results ??? (MLWin, Stata, ???)
- make glmmBUGS work?
- can trace of hat matrix be resurrected??
- work with simulated data of a similar size?

References

Gelman, A. 2006. Prior distributions for variance parameters in hierarchical models. Bayesian Analysis **1**:515–533. URL `http://ba.stat.cmu.edu/journal/2006/vol01/issue03/gelman.pdf`.

Pinheiro, J. C. and D. M. Bates. 2000. Mixed-effects models in S and S-PLUS. Springer, New York.

Skaug, H. and D. Fournier. 2004. Automatic approximation of the marginal likelihood in nonlinear hierarchical models. URL `http://bemata.imr.no/laplace.pdf`.

Skaug, H. J. 2002. Automatic differentiation to facilitate maximum likelihood estimation in nonlinear random effects models. Journal of Computational and Graphical Statistics **11**:458–470. URL `http://pubs.amstat.org/doi/abs/10.1198/106186002760180617`.