Ben Bolker

Wed Jul 3 11:19:07 2013

mixed model lab

Packages/versions used:

```
##         coda    coefplot2     glmmADMB        lme4     MCMCglmm
##       0.16-1      0.1.3.3        0.7.6 0.99999911-5         2.17
##         nlme     pbkrtest         plyr       R2jags     reshape2
##      3.1-109        0.3-4          1.8      0.03-08        1.2.2
```

- Please install the **development** version of `lme4`:

```
install.packages("lme4",repos=c("http://lme4.r-forge.r-project.org/repos",getOption("repos"
```

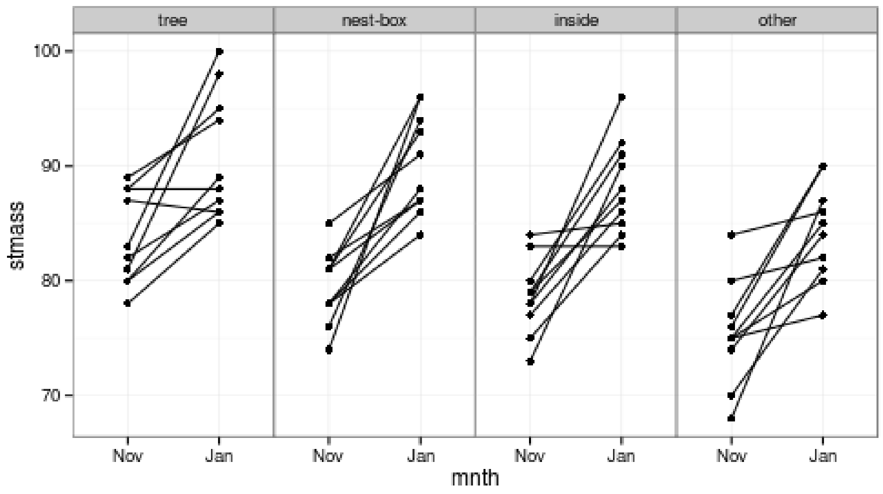
- `coefplot2` is still under development: try `install.packages("coefplot2",repos="http://www.math.m` *after* making sure that all the dependencies are installed (`coda`, `lme4`, `reshape`).

Linear mixed model: starling example

Data from Toby Marthews, r-sig-mixed-models mailing list:

```
library(ggplot2)
## cosmetic tweaks
theme_set(theme_bw())
library(grid)
zmargin <- theme(panel.margin=unit(0,"lines")) ## squash panels together
load("data/starling.RData")
```

```
ggplot(dataf,aes(mnth,stmass))+
  geom_point()+
  geom_line(aes(group=subject))+  ## connect subjects with a line
  facet_grid(.~roostsitu)+        ## 1 row of panels by roost
  zmargin                         ## squash together
```



You should also try

```
ggplot(dataf,aes(mnth,stmass,colour=roostsitu))+
  geom_point()+
  geom_line(aes(group=subject))
```

for comparison, but I prefer the former plot

It's pretty obvious that the starting (November) mass varies among roost situations (tree/nest-box/etc.), and that mass increases from November to January, but we might like to know if the slopes differ among situations. That means our fixed effects would be `~roostsitu*mnth`, with our attention focused on the `roostsitu:mnth` (interaction) term. For random effects, we can allow both intercept (obviously) and slope (maybe) to vary among individuals, via `~1+mnth|subject` or equivalent ... in this case, because measurements are only taken in two months, we **could** also write the random term as `~1|subject/mnth`. **However**, it turns out that we can't actually estimate random slopes for this model, because every individual is only measured twice. That means that the variance in the slopes would be completely confounded with the residual variance (I only figured this out when I went to run the model in `lme4` and it complained: `lme` gave me an answer, but (as we shall see) it didn't necessarily make sense ...

```r
library(nlme)
lme1 <- lme(stmass~mnth*roostsitu,
  random=~1|subject/mnth,data=dataf)
```

The variance components apparently make sense: look at `VarCorr(lme1)`. However, if we try to get the approximate confidence intervals via `intervals(lme1,which="var-cov")` we get an error: `cannot get confidence intervals on var-cov components: Non-positive definite approximate variance-covariance`. This is a **bad sign**.

```r
lme2 <- lme(stmass~mnth*roostsitu,random=~1|subject,data=dataf)
```

We can now get estimates, although the subject-level random effects are *very* uncertain: see `intervals(lme2,which="var-cov")`.

Walking through the output:

```
## Linear mixed-effects model fit by REML
##  Data: dataf
##     AIC   BIC logLik
##   449.6 472.4 -214.8
```

- This reminds us that the model was fitted by restricted maximum likelihood, and gives us the value of the AIC, BIC, and log-likelihood (only useful for comparing with other fitted models)

```
## Random effects:
##  Formula: ~1 | subject
##         (Intercept) Residual
## StdDev:      0.5869    4.165
```

- This tells us about the random effect – it reminds us of the formula (intercept variation among subjects) and gives us the standard deviation of the random effect (this is *not* an estimate of the uncertainty of the estimate – the estimate itself *is* a variance, or standard deviation), and the standard deviation of the residual variance. We can see that the standard deviation is pretty small relative to the residual variance (which is the appropriate comparison).

```
## Fixed effects: stmass ~ mnth * roostsitu
##                             Value Std.Error DF t-value p-value
## (Intercept)                  83.6     1.330 36   62.85  0.0000
## mnthJan                       7.2     1.863 36    3.87  0.0004
## roostsitunest-box            -4.2     1.881 36   -2.23  0.0319
## roostsituinside            -5.0     1.881 36   -2.66  0.0117
## roostsituother             -8.2     1.881 36   -4.36  0.0001
## mnthJan:roostsitunest-box   3.6     2.634 36    1.37  0.1802
## mnthJan:roostsituinside     2.4     2.634 36    0.91  0.3683
## mnthJan:roostsituother      1.6     2.634 36    0.61  0.5474
```

- the standard fixed-effect output. `lme` tells us the denominator degrees of freedom for the test (36), but please note **this value is often wrong in random-slopes models**. The df are fairly large here, so the p-values are pretty close to their Normal equivalent (the 95% critical value is 2.03).
- the next few lines (21-38) are the correlations among the fixed-effect parameters, which we're not usually very concerned with – we might be concerned if the correlations were very large (>0.95?), possibly indicating problems with the fit

```
## Standardized Within-Group Residuals:
##     Min      Q1     Med      Q3     Max
## -1.7555 -0.7687 -0.0864  0.7022  2.1693
```

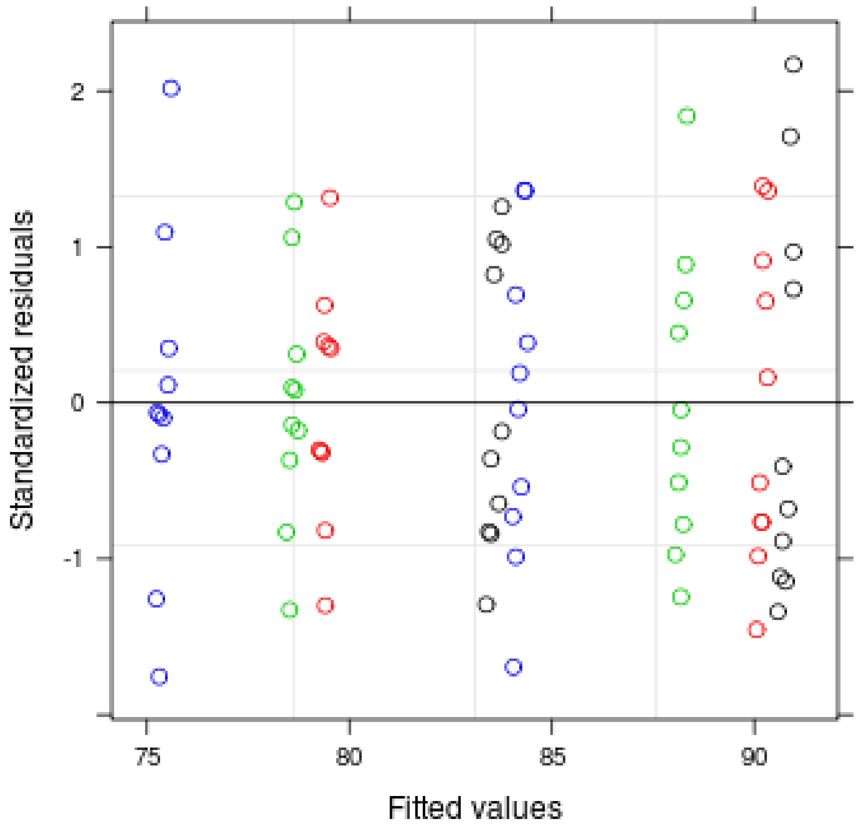
- the distribution of standardized residuals should be reasonably consistent with a sample from the standard normal of the appropriate size (median near zero, quartiles around ± 0.67, min/max appropriate for the sample size)

```
## Number of Observations: 80
## Number of Groups: 40
```

- the listed number of observations and groups is *very* useful for double-checking that the random effects grouping specification is OK
- Notice that the new residual variance is the same as the old subject-by-month variance plus the old residual variance, and the subject-level (intercept) variance is (very nearly) identical.

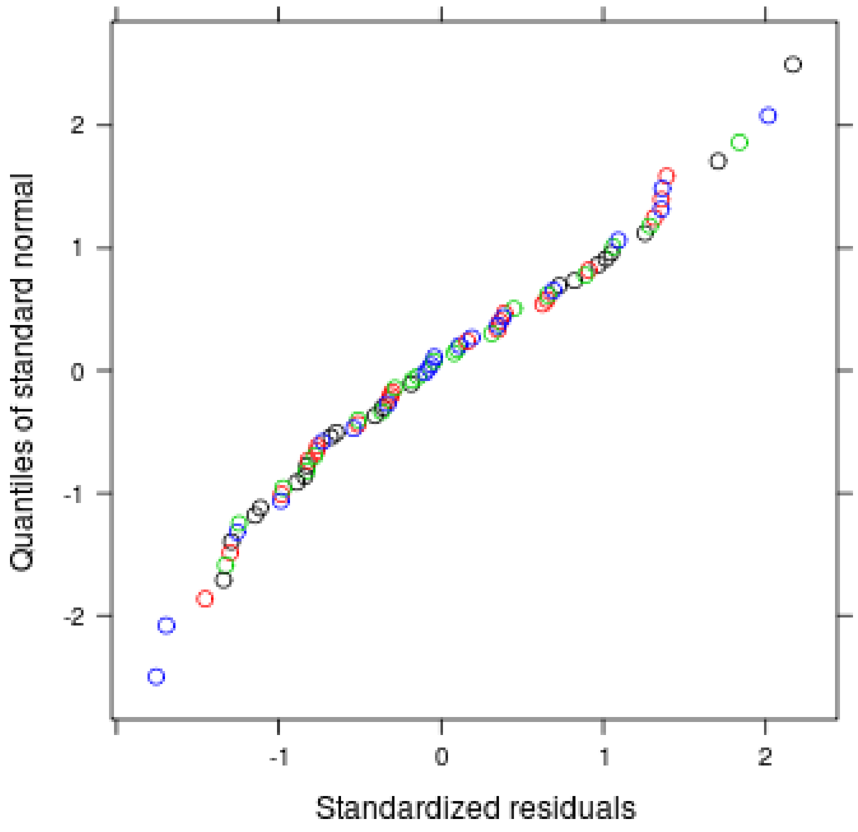
Diagnostic plots: fitted vs. residuals, coloured according to the `roostsitu` variable:

```
plot(lme2,col=dataf$roostsitu)
```



Q-Q plot (a straight line indicates normality)

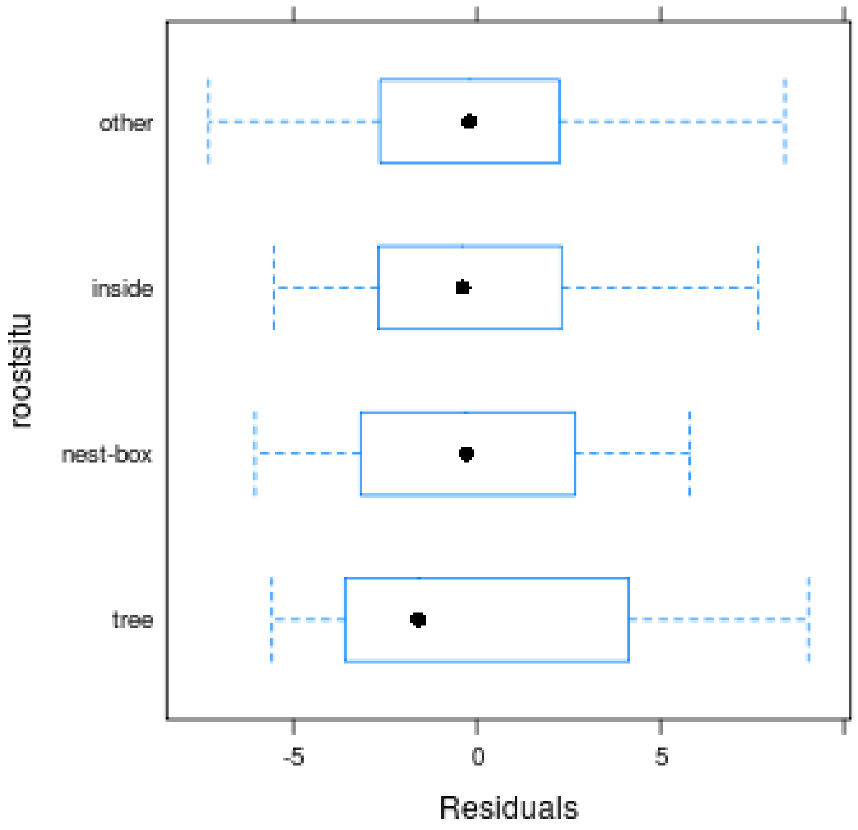
```
qqnorm(lme2,col=dataf$roostsitu)
```



(There are some deviations here, but not enough that I would worry very much.)

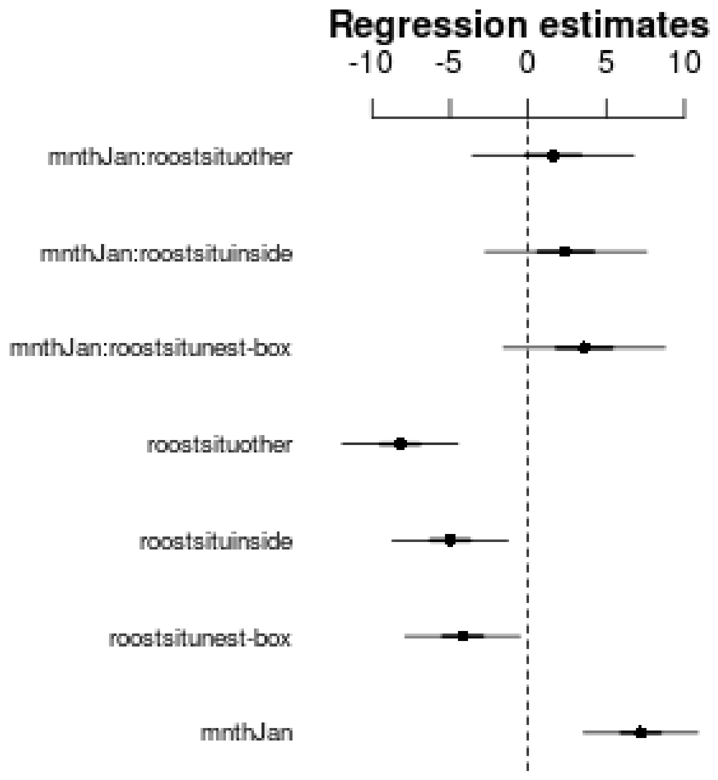
Boxplots of residuals subdivided by `roostsitu` (it's a quirk that you have to put the grouping variable on the *left* side of the formula here):

```
plot(lme2,roostsitu~resid(.))
```



One good way to assess the results of a model fit is to look at a coefficient plot:

```
library(coefplot2)
coefplot2(lme2)
```



Stop and explain to yourself what these parameters mean. If you're not sure, try resetting the base level of the `roostsitu` factor: `dataf2 <- transform(dataf,roostsitu=relevel(roostsitu,ref="other"))`, predict what will happen to the results, and re-run the analysis.)

Exercise: This is actually a slightly trivial example, because there are only two measurements for each individual: thus we can actually get the same answers using a *t* test (as long as we only care about the changes within subjects).

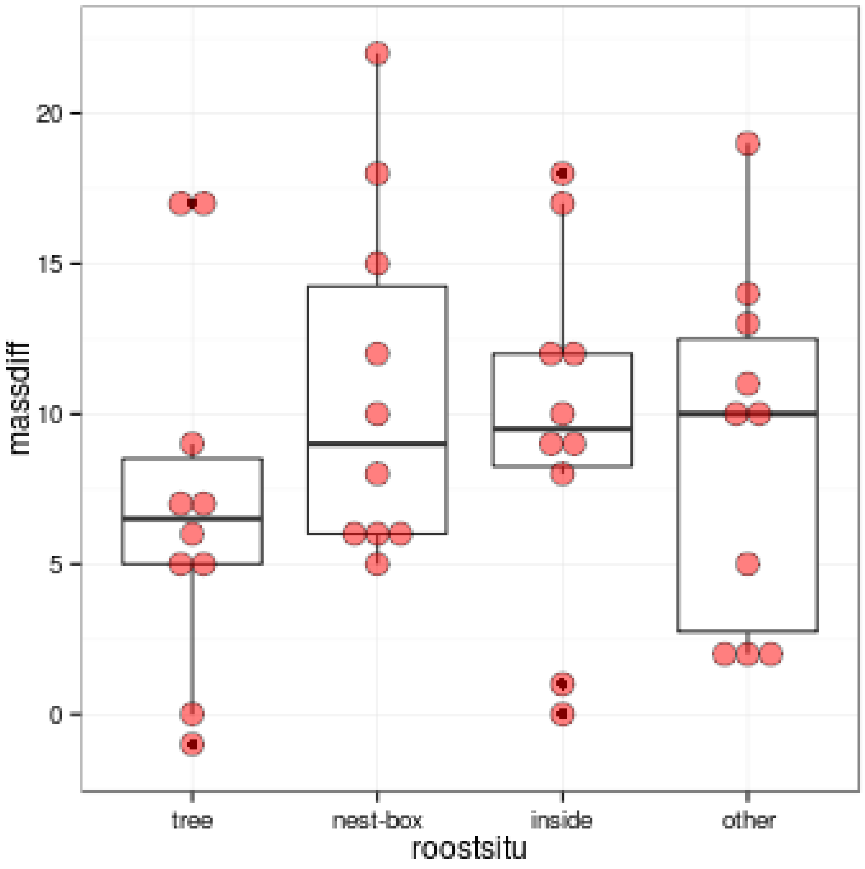
Rearrange the data to get differences by subject:

```
library(plyr)
dataf2 <- ddply(dataf,"subject", summarise,
             roostsitu=roostsitu[1],massdiff=diff(stmass))
```

This says to split the data frame `dataf` by the `subject` variable; for each value of `subject`, put together a data frame with a `roostsitu` variable equal to the first element of the `rootsitu` vector for that subject and a `massdiff` variable equal to the difference between the masses.

Draw a picture (boxplot+beeswarm plot):

```
ggplot(dataf2,aes(x=roostsitu,y=massdiff))+geom_boxplot()+
   geom_dotplot(binaxis="y",stackdir="center",fill="red",alpha=0.5)
```



As you can see, `geom_dotplot()` adds a horizontal dot-plot for each group (see the documentation for examples and more detail).

Analyze the data with `lm` and convince yourself that the estimates (fixed-effect coefficients, residual variance, etc.) are equivalent to those found from the previous analysis.

analyze with `lme4`

The `lmer` syntax is almost identical, except that the random effects (in parentheses) are added to the fixed effects to make a single formula rather than being expressed separately.

```
library(lme4)
```

First try out the original (bad) model specification and see what happens:

```
lmer(stmass~mnth*roostsitu+(1|subject/mnth),data=dataf)
```

```
lmer1 <- lmer(stmass~mnth*roostsitu+(1|subject),data=dataf)
```

Compare the results (you can use `coefplot2(list(lmer1,lme2))` to compare the fixed effects graphically).

See what other options you have for exploring the results.

analyze with `MCMCglmm`

```
library(MCMCglmm)
mcmcglmm1 <- MCMCglmm(stmass~mnth*roostsitu,
         random=~subject,data=dataf,
         verbose=FALSE)
```

We use `verbose=FALSE` to turn off the progress messages, which would be ugly in this document but are generally useful.

- Compare the results (use `summary()`: printing out the a raw `MCMCglmm` model is ugly).

For MCMC approaches, it is your responsibility to check that the chain(s) are well-behaved.

Try this:

```
library(coda)
xyplot(as.mcmc(mcmcglmm1$Sol))
```

analyze with JAGS/R2jags

```
jagsmodel <- function() {
  for (i in 1:ntot) {
   eta[i]  <- inprod(X[i,],beta)    ## fixed effects
   eta2[i] <- eta[i] + u1[subject[i]] ## add random effect
   stmass[i] ~ dnorm(eta2[i],tau.res)
  }
  for (i in 1:nindiv) {
    u1[i] ~ dnorm(0,tau.indiv)
  }
  ## priors
  for (i in 1:ncoef) {
    beta[i] ~ dnorm(0,0.001)
  }
  ## traditional but sometimes dangerous -- discuss in class
  tau.indiv ~ dgamma(0.01,0.01)
  tau.res ~ dgamma(0.01,0.01)
  ## for convenience, translate precisions to std devs
  sd.indiv <- pow(tau.indiv,-0.5)
  sd.res <- pow(tau.res,-0.5)
}
source("R/writeModel.R")
write.model(jagsmodel,"starling.bug")
```

JAGS machinery: specify data list, starting values, run model:

```
library(R2jags)
```

```
## Warning: the specification for S3 class "bugs" in package 'R2jags' seems
## equivalent to one from package 'coefplot2' and is not turning on duplicate
## class definitions for this class
```

```
modelMat <- model.matrix(~mnth * roostsitu,data=dataf)
jagsData <- list(X=modelMat,
              stmass=dataf$stmass,
               subject=as.numeric(dataf$subject),
                 ntot=nrow(dataf),ncoef=ncol(modelMat),
                 nindiv=length(unique(dataf$subject)))
jagsInits <- list(list(beta=rep(0,8),tau.indiv=0.1,tau.res=0.1))
jags1 <- jags(data=jagsData,
     inits=jagsInits,
     model="starling.bug",
    parameters=c("beta","sd.indiv","sd.res"),n.chains=1)
```

Notes:

- for simple models it's easier to write out (e.g. `y <- a + b*x`, but for more complex models it rapidly becomes worthwhile to construct a model matrix in R and pass it to `JAGS`, so that all you need is to multiply (`inprod` essentially does a single row of the $X\beta$ calculation at a time)
- it's good practice to avoid hard-coding values as much as possible (e.g. don't set `ncoef` to 8 even if you know that's what it is for this particular example) – instead, use the dimensions of your data to set them
- `write.model` is a handy function from `R2WinBUGS` – since we don't need the rest of it, I copied the code and made it available
- initial values are a *list of lists*. I've been lazy here and used a single chain. Below I suggest that you try it with multiple chains.
- should set random-number seed!

In order to do more fun stuff with the results, it's a good idea to (1) convert to an `mcmc` object (for `coda` quantitative and graphical diagnostics) and (2) rename the `beta` variables more meaningfully. JAGS arranges output variables **alphabetically**: in this case we know the `beta` values are the first 8 columns of the output, but we will again try to avoid hard-coding:

```
jagsmm1 <- as.mcmc(jags1)
betacols <- grep("^beta",colnames(jagsmm1)) ## find beta columns
colnames(jagsmm1)[betacols] <- colnames(modelMat)
```

Trace plots, density plots, and coefficient plots:

```
xyplot(jagsmm1) ## trace plot: default single-column layout
xyplot(jagsmm1,layout=c(3,4),asp="fill")
densityplot(jagsmm1,layout=c(3,4),asp="fill")
coefplot2(jags1)  ## coefficient plot
```

Run the Raftery-Lewis diagnostic (suitable for a single chain):

```
raftery.diag(jagsmm1)

##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
```

Exercise: * Try re-running with `list(list(beta=rep(0,8),tau.indiv=0.1,tau.res=0.1),`
`list(beta=rep(1,8),tau.indiv=0.1,tau.res=0.1),beta=rep(0,8),tau.indiv=1,tau.res=1)),n.chains`
examine the output; run `gelman.diag` from the `coda` package on the `mcmc` results (may need `lump.mcmc.list` from the `emdbook` package ... ?)

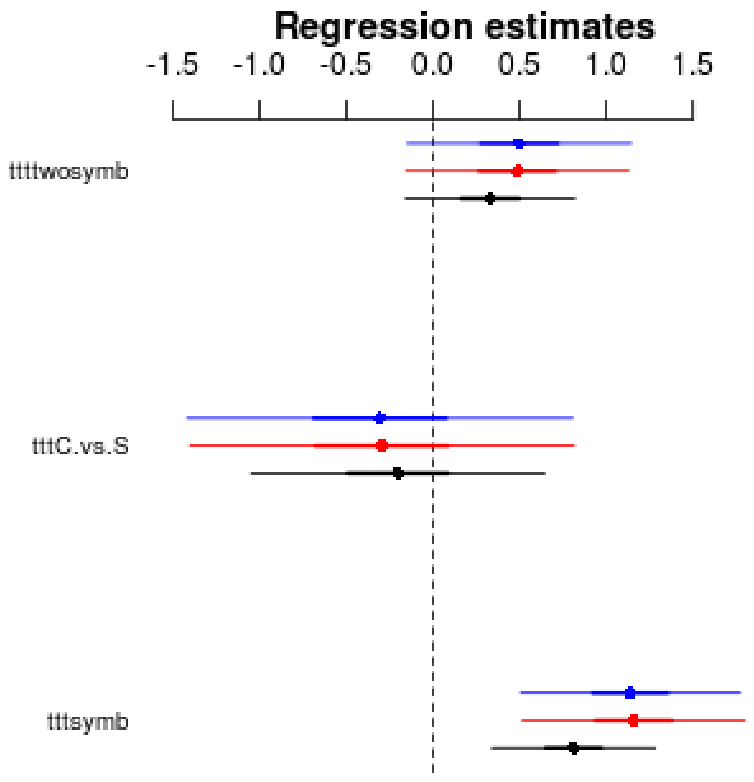
Generalized linear mixed model: *Culcita* example

```
culcdat <- read.csv("data/culcitalogreg.csv",
  colClasses=c(rep("factor",2),
    "numeric",
    rep("factor",6)))
## abbreviate slightly
levels(culcdat$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(culcdat$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")))
```

```
library(lme4)
library(MASS)
culcmod0 <- glmmPQL(predation~ttt,random=~1|block,family=binomial,data=culcdat,
                      verbose=FALSE)
culcmod1 <- glmer(predation~ttt+(1|block),family=binomial,data=culcdat)
culcmod2 <- glmer(predation~ttt+(1|block),family=binomial,data=culcdat,nAGQ=8)
coefplot2(list(glmmPQL=culcmod0,Laplace=culcmod1,
            GHQ8=culcmod2),col=c(1,2,4),legend.x="right")
```



Try it with `glmmADMB` and `MCMCglmm`:

```
library(MCMCglmm)
library(glmmADMB)

## Warning: the specification for S3 class "glmmadmb" in package 'glmmADMB'
## seems equivalent to one from package 'coefplot2' and is not turning on
## duplicate class definitions for this class

culcmod3 <- glmmadmb(predation~ttt,random=~1|block,family="binomial",
                     data=culcdat)
culcdat$nopred <- 1-culcdat$predation
culcmod4 <- MCMCglmm(cbind(predation,nopred)~ttt,random=~block,family="multinomial2",
                     data=culcdat,verbose=FALSE)
```

Check out the results. `MCMCglmm` doesn't seem to be doing well: need stronger priors?

JAGS

```
culcmodel <- function() {
 for (i in 1:ncoef) {
     beta[i] ~ dnorm(0,0.001)  ## fixed-effect parameters: priors
  }
  sd.b ~ dunif(0,maxsdprior)             ## prior for block variance
  tau.b <- 1/sd.b^2
   for (i in 1:nblock) {
     u[i] ~ dnorm(0,tau.b)   ## priors for block random effects
  }
  for (i in 1:nobs) {
    ## linear predictor: design matrix*coeffs + random effects
    eta[i] <- inprod(X[i,],beta)+u[block[i]]
    p[i] <-  1/(1+exp(-eta[i]))            ## convert to probabilities
    obs[i] ~ dbern(p[i])          ## Bernoulli response
  }
}
write.model(culcmodel,"culcita.bug")

cModelMat <- model.matrix(~ttt,data=culcdat)
cJagsDat <- list(nblock=length(unique(culcdat$block)),
                 ncoef=ncol(cModelMat),
                 nobs=nrow(culcdat),
                 maxsdprior=5,
                 obs=culcdat$predation,
                 block=culcdat$block,
                 X=cModelMat)
cJagsInit <- with(cJagsDat,list(list(beta=rep(0,ncoef),
                       sd.b=1,u=rep(0,nblock))))
load.module("glm")
cjags <- jags(data=cJagsDat,
          inits=cJagsInit,
          n.chains=1,
          model.file="culcita.bug",
          parameters=c("beta","sd.b"))

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
##    Graph Size: 790
##
## Initializing model
```

- in the actual analysis, we wanted to use a log link rather than the standard logit link, which turned out be quite difficult . . .

If time permits, check out the `Contraception` data set from the `mlmRev` package . . . % Ben Bolker % Wed Jul 3 11:19:42 2013

Starling example: inference

Wald tests:

```
printCoefmat(summary(lme2)$tTable,digits=3,has.Pval=TRUE)
```

```
##                            Value Std.Error    DF t-value p-value
## (Intercept)                83.60      1.33 36.00   62.85 < 2e-16 ***
## mnthJan                     7.20      1.86 36.00    3.87 0.00045 ***
## roostsitunest-box          -4.20      1.88 36.00   -2.23 0.03188 *
## roostsituinside            -5.00      1.88 36.00   -2.66 0.01165 *
## roostsituother             -8.20      1.88 36.00   -4.36 0.00010 ***
## mnthJan:roostsitunest-box  3.60      2.63 36.00    1.37 0.18024
## mnthJan:roostsituinside    2.40      2.63 36.00    0.91 0.36834
## mnthJan:roostsituother     1.60      2.63 36.00    0.61 0.54743
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

(you can see this in context with `summary(lme2)`; I used the more complicated command here to isolate just the coefficent matrix).

We conclude that the interactions are not doing much, but there's definitely an effect of the roosts. This agrees with the picture from `coefplot2` (above).

However, we probably want to test the overall effect of the interactions, not the individual levels. Here are the type II (sequential) ANOVA results:

```
anova(lme2)
```

```
##                  numDF denDF F-value p-value
## (Intercept)          1    36   31144  <.0001
## mnth                 1    36      95  <.0001
## roostsitu            3    36      11  <.0001
## mnth:roostsitu       3    36       1  0.5838
```

Because of the design of this particular study, the denominator degrees of freedom (`denDF` column) is identical for all effects.

If we want to evaluate the marginal sums of squares, i.e. dropping one term at a time from the model, we usually want to change the model to use sum-to-zero contrasts:

```
lme2B <- update(lme2,
    contrasts=list(mnth="contr.sum",roostsitu="contr.sum"))
```

The alternative approach is to use `options(contrasts=c("contr.sum","contr.poly"))`, then refit the model, but I prefer to use the `contrasts` argument because it is more explicit.

Type III (marginal) results:

```
anova(lme2B,type="marginal")
```

```
##                 numDF denDF F-value p-value
## (Intercept)         1    36   31144  <.0001
## mnth                1    36      95  <.0001
## roostsitu           3    36      11  <.0001
## mnth:roostsitu      3    36       1  0.5838
```

In this case the results are identical ***because the original design is balanced (hence, orthogonal)***. Not true if the data are (1) unbalanced (which is often true of ANOVA [categorical-predictor] designs, and almost always true of regression designs) or (2) GLMM or nonlinear.

The explicit model-comparison approach uses a likelihood ratio test rather than an F test (i.e., it does not correct for the fact that the denominator sums of squares is estimated with error). In this case it hardly matters.

```
lme2C <- update(lme2B,method="ML")
lme2D <- update(lme2C,. ~ . - mnth:roostsitu)
anova(lme2C,lme2D)
```

```
##       Model df   AIC   BIC logLik   Test L.Ratio p-value
## lme2C     1 10 468.4 492.3 -224.2
## lme2D     2  7 464.6 481.3 -225.3 1 vs 2   2.134   0.545
```

If we now want to use the model-comparison approach on the reduced (no-interaction) model to test the significance of `roostsitu`, we can use `update` to drop the `roostsitu` effect, but we also have to make sure to update the `contrasts` argument so that it only refers to predictors that remain in the reduced model (otherwise, we get an error).

```
lme2E <- update(lme2D,.~.-roostsitu,
                 contrasts=list(mnth="contr.sum"))
anova(lme2D,lme2E)
```

```
##       Model df   AIC   BIC logLik   Test L.Ratio p-value
## lme2D     1  7 464.6 481.3 -225.3
## lme2E     2  4 483.9 493.5 -238.0 1 vs 2   25.34  <.0001
```

If we want to test the random effect, we would in principle remove the random effect and test with `anova`, but this is a bit problematic here: `lme` can't fit a model without any random effects.

Let's try this with `gls`:

```
gls1 <- gls(stmass~mnth*roostsitu,
            data=dataf,method="ML")
(a1 <- anova(lme2C,gls1))
```

```
##       Model df   AIC   BIC logLik   Test L.Ratio p-value
## lme2C     1 10 468.4 492.3 -224.2
## gls1      2  9 466.5 487.9 -224.2 1 vs 2 0.01516   0.902
```

This seems reasonable: let's see if we can confirm this result with `RLRsim`.

```
library(RLRsim)
exactRLRT(m=lme2B)
```

```
##
##  simulated finite sample distribution of RLRT.  (p-value based on
##  10000 simulated values)
##
## data:
## RLRT = 0, p-value = 1
```

(This reports a p value of exactly zero: this is "consistent with" a p value of 0.902, but worries me.)

For `lmer` we get the same anova table, but (1) without the `Intercept` term included (2) with sum-of-squares and mean-squares columns included (3) *without* denominator df or p-values (4) with slightly different precision (1 more significant figure):

```
(a2 <- anova(lmer1))
```

```
## Analysis of Variance Table
##                Df Sum Sq Mean Sq F value
## mnth            1   1656    1656   95.46
## roostsitu       3    552     184   10.61
## mnth:roostsitu  3     34      11    0.66
```

We have to deduce the number of degrees of freedom from standard rules:

```
with(dataf,table(mnth,roostsitu))
```

```
##      roostsitu
## mnth  tree nest-box inside other
##   Nov   10       10     10    10
##   Jan   10       10     10    10
```

If we think about this in terms of the paired *t*-test, there are 40 comparisons and 4 parameters (one for each roost site)= 36 df.

If you wanted to compute the *p*-values by hand, you could:

```
fvals <- a2[,"F value"]
numdf <- a2[,"Df"]
dendf <- 36
pf(fvals,numdf,dendf,lower.tail=FALSE)
```

```
## [1] 1.152e-11 3.833e-05 5.838e-01
```

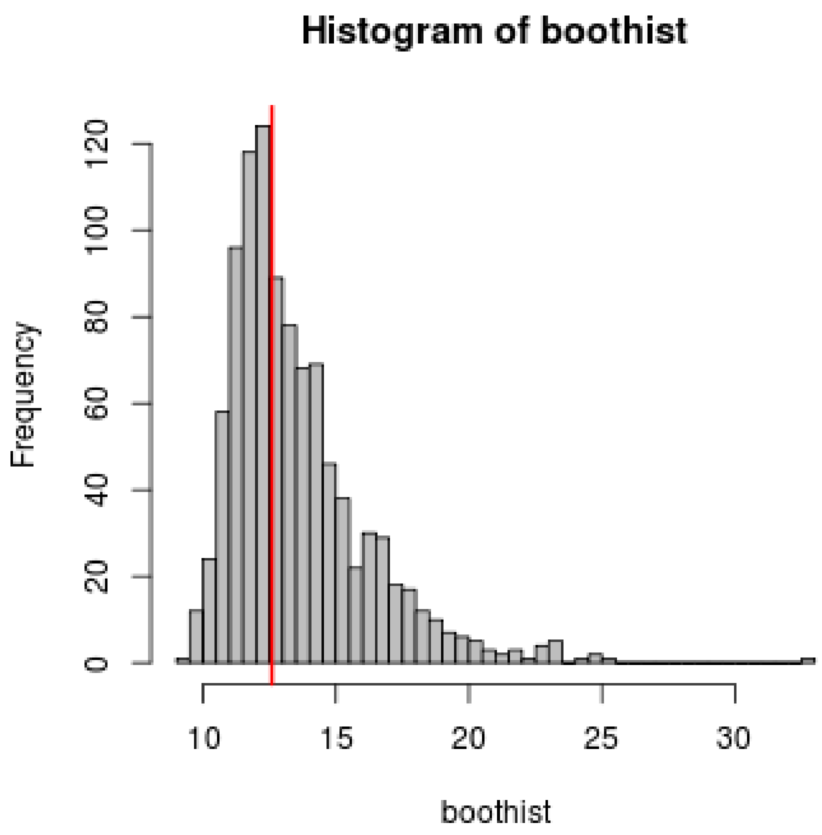
Alternatively, we can try a parametric bootstrap: the `pbkrtest` package can do this, or we can just set it up by hand:

```
lmer3 <- lmer(stmass~mnth*roostsitu+(1|subject),data=dataf)
lmer4 <- lmer(stmass~mnth+roostsitu+(1|subject),data=dataf)

pboot <- function(m0,m1) {
  s <- simulate(m0)
  L1 <- logLik(refit(m1,s[[1]]))
  L0 <- logLik(refit(m0,s[[1]]))
  2*(L1-L0)
}
pboot(lmer4,lmer3)
```

```
## 'log Lik.' 13.98 (df=10)
```

```
boothist <- replicate(1000,pboot(lmer4,lmer3))
library(plyr)
boothist <- rlply(1000,pboot(lmer4,lmer3))
## can use .progress="text" to get a progress bar ...
boothist <- unlist(boothist)
hist(boothist,breaks=50,col="gray")
obsval <- 2*(logLik(lmer3)-logLik(lmer4))
abline(v=obsval,col=2,lwd=2)
```



```
mean(boothist>obsval)
```

```
## [1] 0.548
```

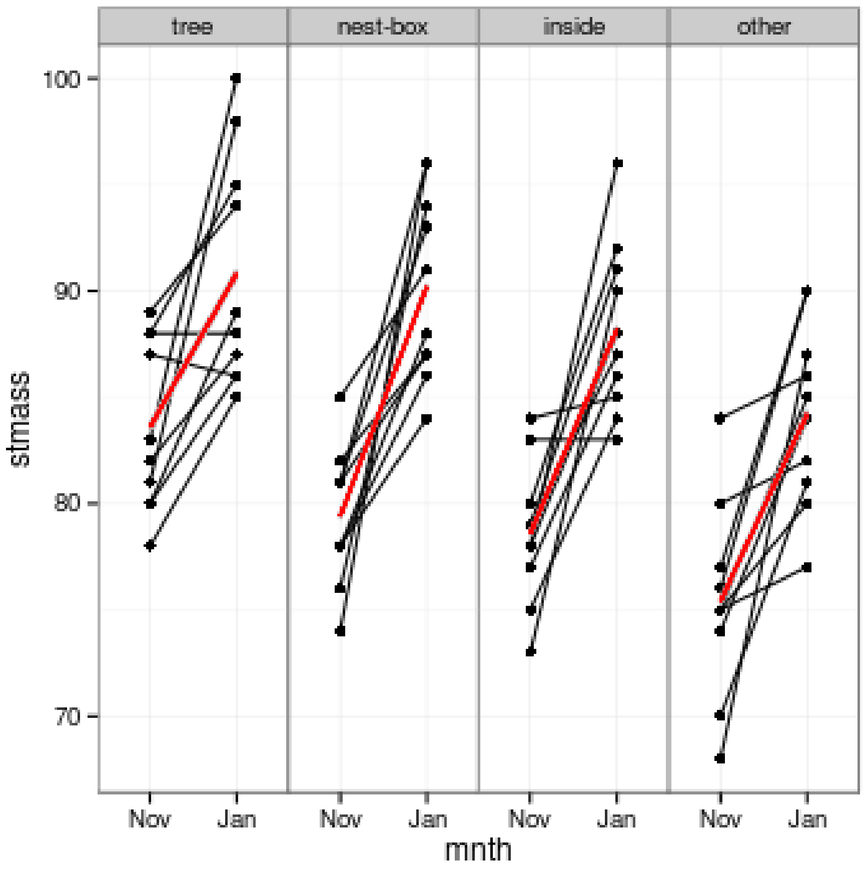
```
library(pbkrtest)
KRmodcomp(lmer3,lmer4)
```

```
## F-test with Kenward-Roger approximation; computing time: 0.26 sec.
## large : stmass ~ mnth * roostsitu + (1 | subject)
## small : stmass ~ mnth + roostsitu + (1 | subject)
##        stat   ndf   ddf F.scaling p.value
## Ftest  0.66  3.00 36.00         1    0.58
```

In this case, the Kenward-Roger correction appropriately does nothing different – we have a classical balanced design and no correction is actually necessary. But it does give a denominator df and a *p*-value for this lmer model, which is handy . . .

Computing predicted values and superimposing them on data plots. In `lme`, `predict` has a `level` argument that specifies which levels of the random effects should be included (0=none, population level; 1=prediction at the subject level; more complex models, might have additional nested levels). The stable version of `lmer` doesn't have a `predict` method: you can try the development version, or look at prediction code at the GLMM faq.

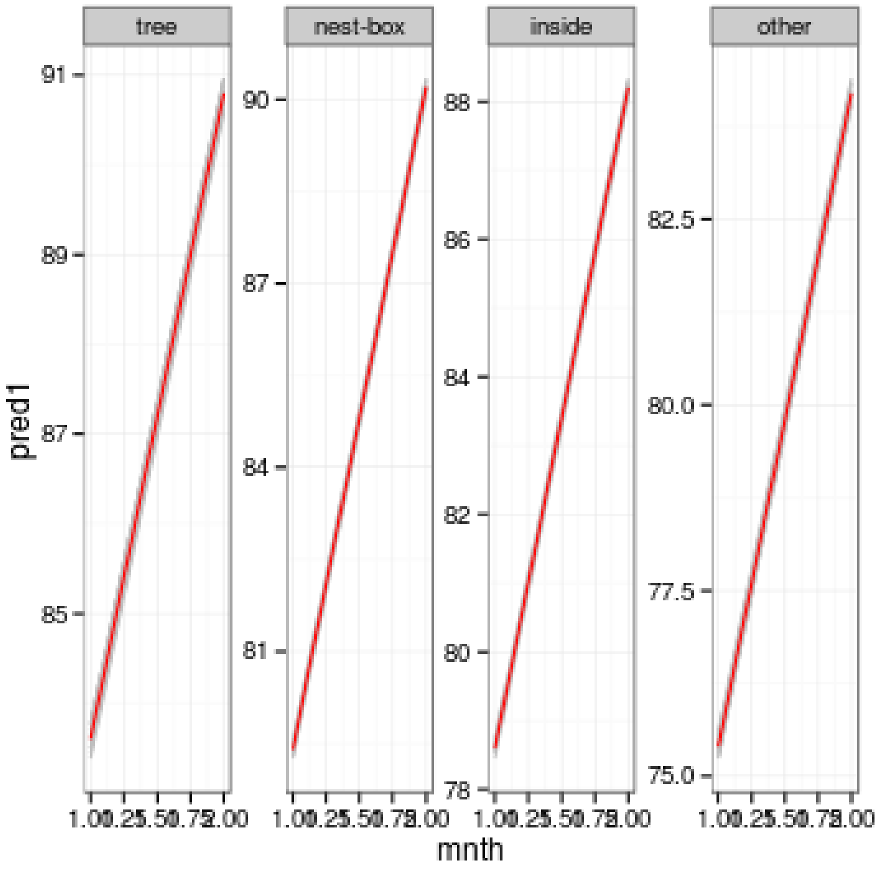
```
library(ggplot2)
library(grid)
theme_set(theme_bw())
## squash panels together (need grid package loaded)
zmargin <- theme(panel.margin=unit(0,"lines"))
dataf$pred <- predict(lme2,level=0)  ## population level
dataf$pred1 <- predict(lme2,level=1) ## individual level
g0 <- ggplot(dataf,aes(mnth,stmass))+
  geom_point()+
  geom_line(aes(group=subject))+
  facet_grid(.~roostsitu)+
  zmargin
g0 +   geom_line(colour="gray",aes(y=pred1,group=subject)) +
  geom_line(colour="red",aes(y=pred,group=subject))
```



There is so much shrinkage (the among-individual variance is very small) that we can barely see the individual-level predictions (gray lines) behind the population-level predictions (red lines).

Unfortunately computing confidence intervals for the predictions is a little tricky: again, there is some code on the GLMM faq for this (also see below under `MCMCglmm`).

```
ggplot(dataf,aes(mnth,pred1))+
  geom_line(aes(group=subject,x=as.numeric(mnth)),colour="gray")+
  facet_wrap(~roostsitu,scale="free_y",nrow=1)+
  geom_line(aes(y=pred,x=as.numeric(mnth)),colour="red")
```



For most cases you will want to set up a new data frame to do prediction rather than just using the covariates from the original data (e.g. if the data are sampled irregularly, or sparsely), and use the `newdata` argument of `predict`. The `expand.grid` function is handy in this context too.

analyze with `MCMCglmm`

```
summary(mcmcglmm1)
```

```
##
##  Iterations = 3001:12991
##  Thinning interval  = 10
##  Sample size  = 1000
##
##  DIC: 466.8
```

```
##
##  G-structure:  ~subject
##
##         post.mean l-95% CI u-95% CI eff.samp
## subject  2.18e-10 1.23e-16 7.18e-10        0
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units      18.2     12.8     24.4      835
##
##  Location effects: stmass ~ mnth * roostsitu
##
##                            post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)                   83.627   80.853   86.170      878 <0.001 ***
## mnthJan                        7.185    3.309   10.688     1000 <0.001 ***
## roostsitunest-box             -4.172   -7.776   -0.583      870  0.026 *
## roostsituinside               -5.020   -8.751   -1.367     1000  0.006 **
## roostsituother                -8.188  -11.767   -4.399     1000  0.002 **
## mnthJan:roostsitunest-box      3.547   -1.420    8.813      802  0.188
## mnthJan:roostsituinside        2.407   -2.327    7.640     1000  0.340
## mnthJan:roostsituother         1.567   -4.125    6.590     1000  0.560
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

MCMCglmm makes it a little bit easier to get confidence intervals on the predictions. The `marginal` argument specifies which random effects we want the predictions to "marginalise" (i.e., average over). The default is to use all of the random effects in the original model (i.e. `~subject` in this case), i.e. to predict the average population-level outcome. The approach taken below to get the subject-level predictions (i.e. marginalise over nothing, since `subject` is the only random effect) is a bit of a hack: this may be easier in future versions.

In order to get all the predictions we want, we need to refit the model using `pr=TRUE` to store the random effects (which are sometimes quite large, hence not stored by default):

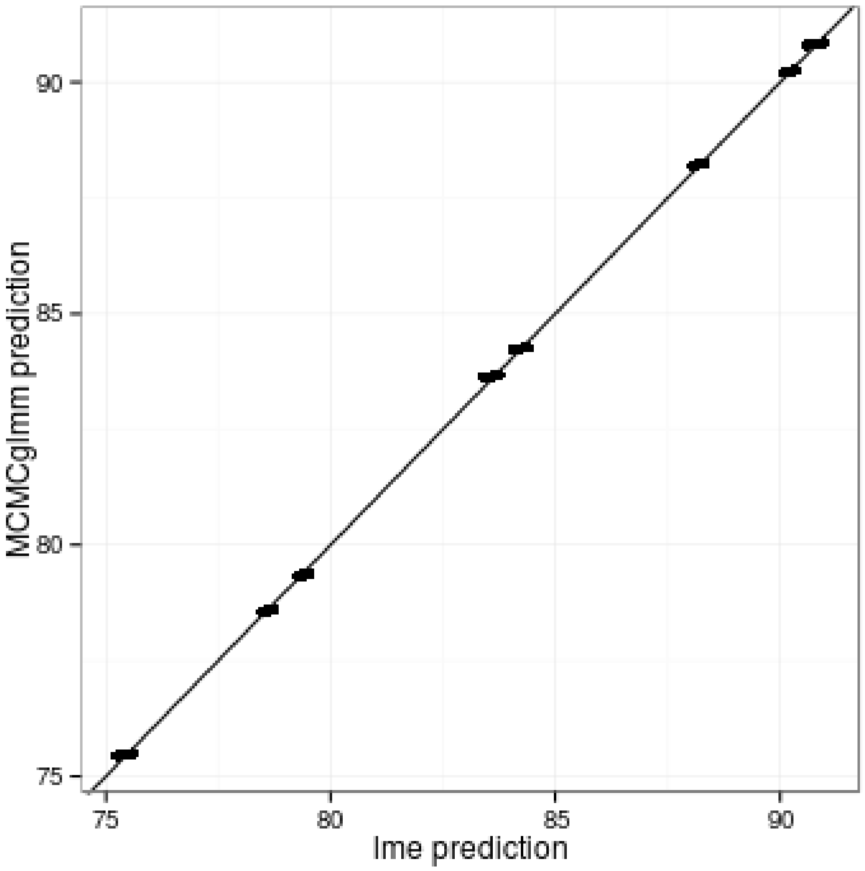
```
mcmcglmm1R <- MCMCglmm(stmass~mnth*roostsitu,
         random=~subject,data=dataf,
         verbose=FALSE,pr=TRUE)
```

```
mpred0 <- predict(mcmcglmm1R,interval="confidence")
colnames(mpred0) <- paste("mpred0",c("fit","lwr","upr"),sep=".")
mpred1 <- predict(mcmcglmm1R,interval="confidence",
                  marginal=c("",""))  ## HACK
```

```
colnames(mpred1) <- paste("mpred1",c("fit","lwr","upr"),sep=".")
dataf <- data.frame(dataf,mpred0,mpred1)
```

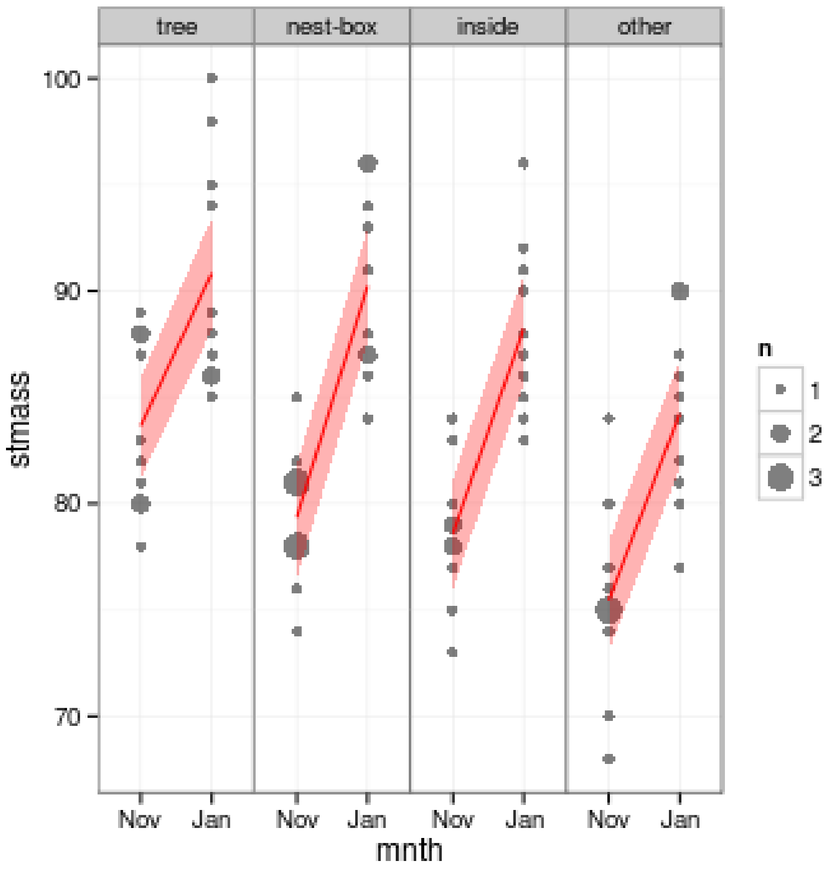
Testing that we get the same results for the level-1 predictions:

```
ggplot(dataf,aes(x=pred1,y=mpred1.fit))+geom_point()+
  geom_abline(slope=1,intercept=0)+
  labs(x="lme prediction",y="MCMCglmm prediction")
```



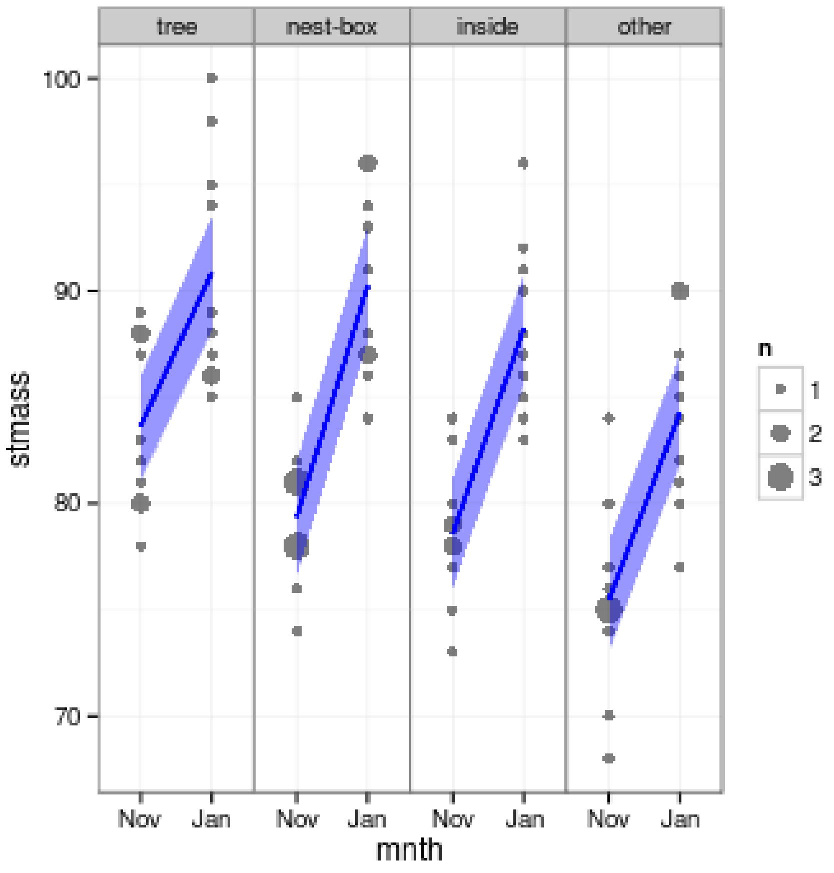
Now we can plot confidence intervals

```
g0 <- ggplot(dataf,aes(x=mnth,y=stmass))+
  stat_sum(aes(size=factor(..n..)),alpha=0.5)+
  facet_grid(~roostsitu)+
  scale_size_discrete(name="n",range=c(2,5))+
  zmargin
g0 + geom_line(aes(x=as.numeric(mnth),y=mpred0.fit),colour="red")+
  geom_ribbon(aes(x=as.numeric(mnth),y=mpred0.fit,
                ymin=mpred0.lwr,ymax=mpred0.upr),fill="red",
              alpha=0.3)
```



We can plot individual predictions and confidence intervals as well, although in this case the plot is almost identical:

```
g0 + geom_line(aes(x=as.numeric(mnth),y=mpred1.fit,group=subject),
               colour="blue")+
   geom_ribbon(aes(x=as.numeric(mnth),y=mpred1.fit,group=subject,
                ymin=mpred1.lwr,ymax=mpred1.upr),fill="blue",
              alpha=0.05)
```



analyze with JAGS/R2jags

Look at the summary:

```
summary(jagsmm1)
```

```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
```

```
##                                Mean     SD Naive SE Time-series SE
## (Intercept)                  83.398  1.657   0.0524         0.0524
## mnthJan                       7.346  1.973   0.0624         0.0624
## roostsitunest-box            -4.012  1.997   0.0632         0.0605
## roostsituinside              -4.791  2.122   0.0671         0.0671
## roostsituother               -7.959  2.374   0.0751         0.0751
## mnthJan:roostsitunest-box     3.509  2.608   0.0825         0.0825
## mnthJan:roostsituinside       2.310  2.811   0.0889         0.0889
## mnthJan:roostsituother        1.394  2.691   0.0851         0.0830
## deviance                    456.152 15.818   0.5002         0.5494
## sd.indiv                      0.573  0.531   0.0168         0.0168
## sd.res                        4.294  2.668   0.0844         0.0844
##
## 2. Quantiles for each variable:
##
##                                 2.5%     25%     50%     75%   97.5%
## (Intercept)                  80.8512  82.477  83.406  84.355  86.037
## mnthJan                       3.6345   6.036   7.337   8.601  11.133
## roostsitunest-box            -7.6632  -5.301  -3.998  -2.745  -0.548
## roostsituinside              -8.3313  -6.100  -4.883  -3.510  -1.005
## roostsituother              -11.5847  -9.323  -8.096  -6.702  -4.275
## mnthJan:roostsitunest-box    -1.5017   1.670   3.500   5.290   8.561
## mnthJan:roostsituinside      -2.8318   0.454   2.438   4.117   7.410
## mnthJan:roostsituother       -3.8512  -0.441   1.494   3.280   6.454
## deviance                    439.0435 452.628 455.758 459.511 467.286
## sd.indiv                      0.0653   0.166   0.364   0.811   1.984
## sd.res                        3.4634   3.937   4.180   4.439   4.983
```