Ben Bolker

Mon Oct 29 23:20:23 2012

mixed model lab 1

Linear mixed model: starling example

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

```
load("starling.RData")
library(ggplot2)
library(grid)
theme_set(theme_bw())
## squash panels together (need grid package loaded)
zmargin <- theme(panel.margin=unit(0,"lines"))
ggplot(dataf,aes(mnth,stmass))+
  geom_point()+
  geom_line(aes(group=subject))+
  facet_grid(.~roostsitu)+
  zmargin
```

You could also try

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

but I find the former more pleasant to look at.

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,

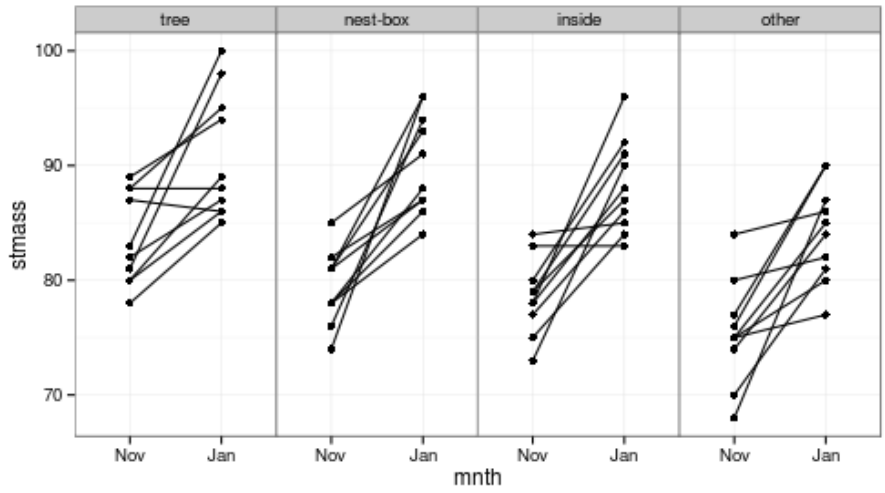

Figure 1: plot of chunk starling1

but we might like to know if the slopes differ among situations. That means our fixed effects would be `~roostsitu*month`, with our attention focused on the `roostsitu:month` (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 ...

```
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.

```
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")` (we may investigate this

further in a future lab).

- 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.
- Print out `summary(lme2)` and try to interpret all the pieces.

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

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

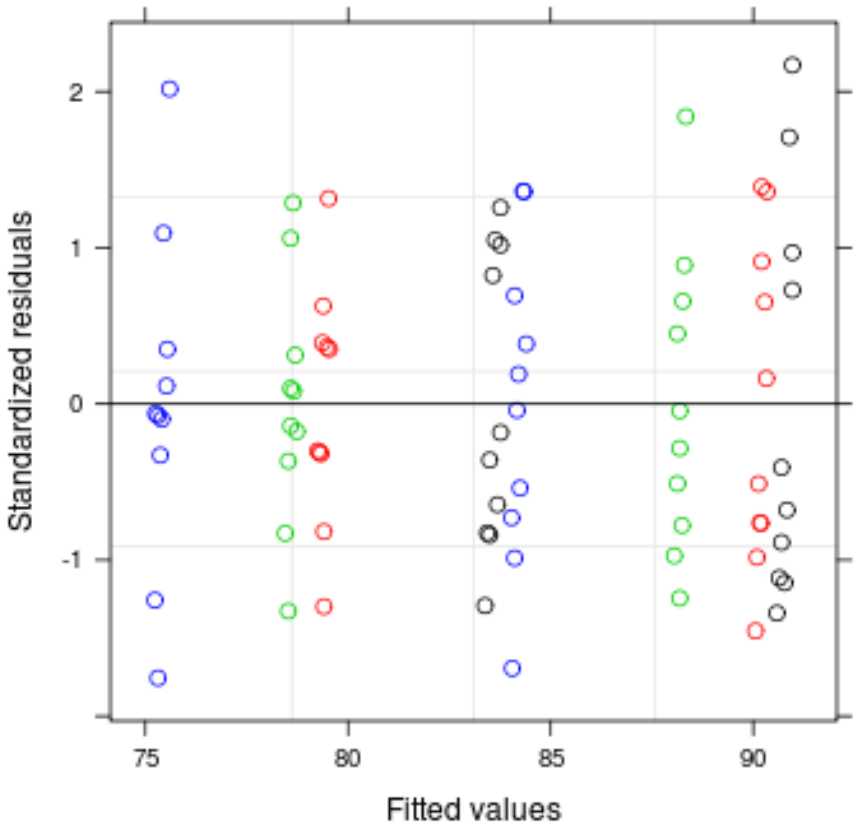

Figure 2: plot of chunk lmediag

Q-Q plot (a straight line indicates normality)

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

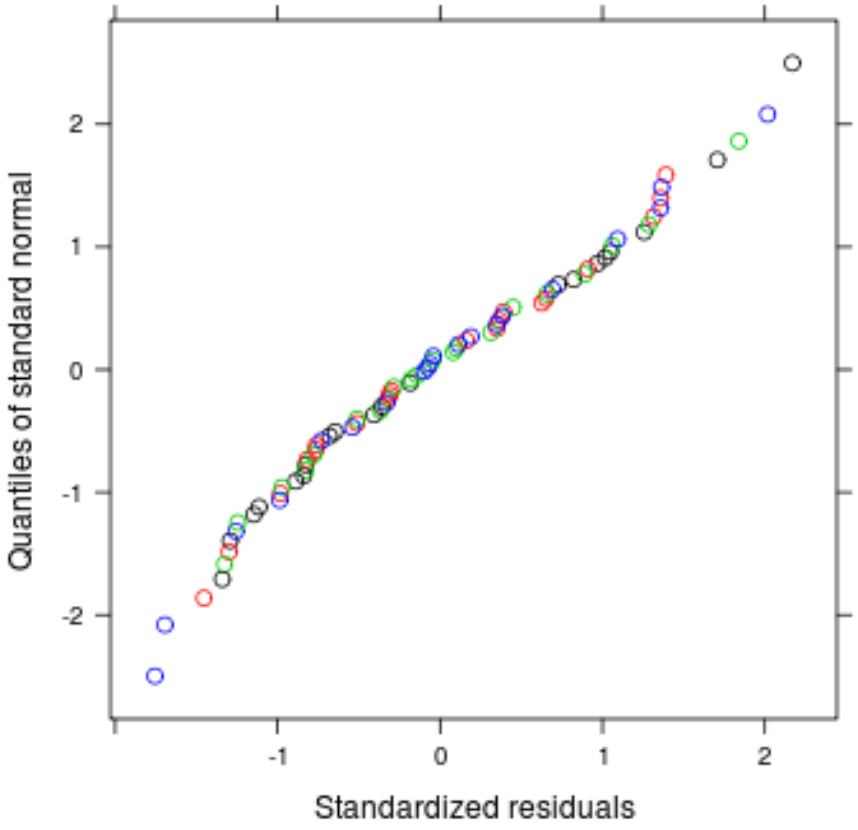

Figure 3: plot of chunk qqnorm

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)
```

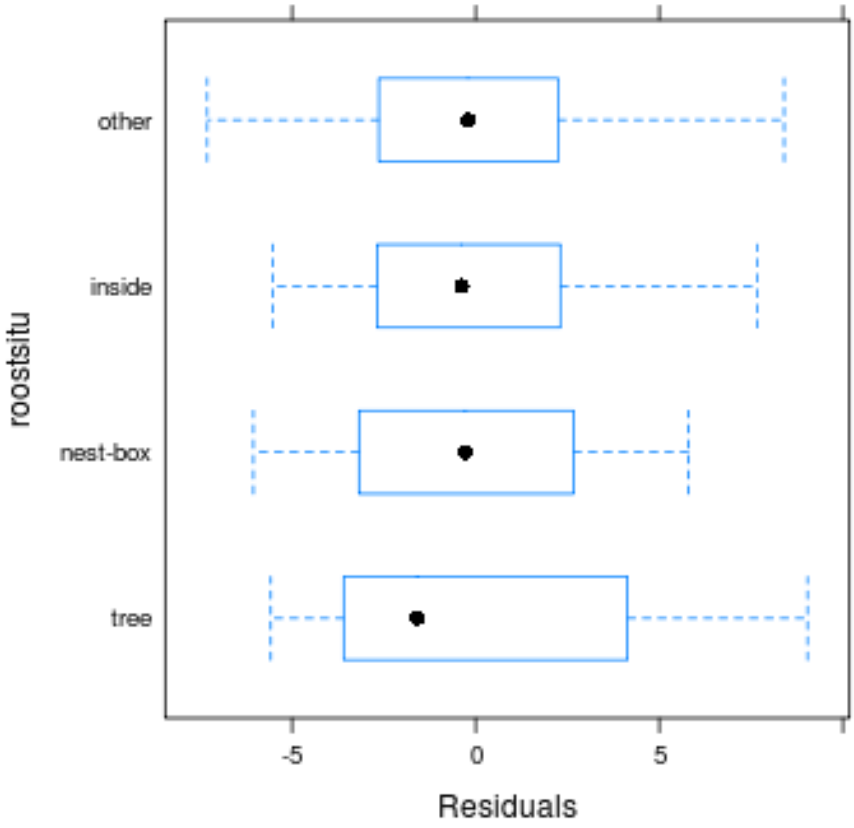

Figure 4: plot of chunk diagbox

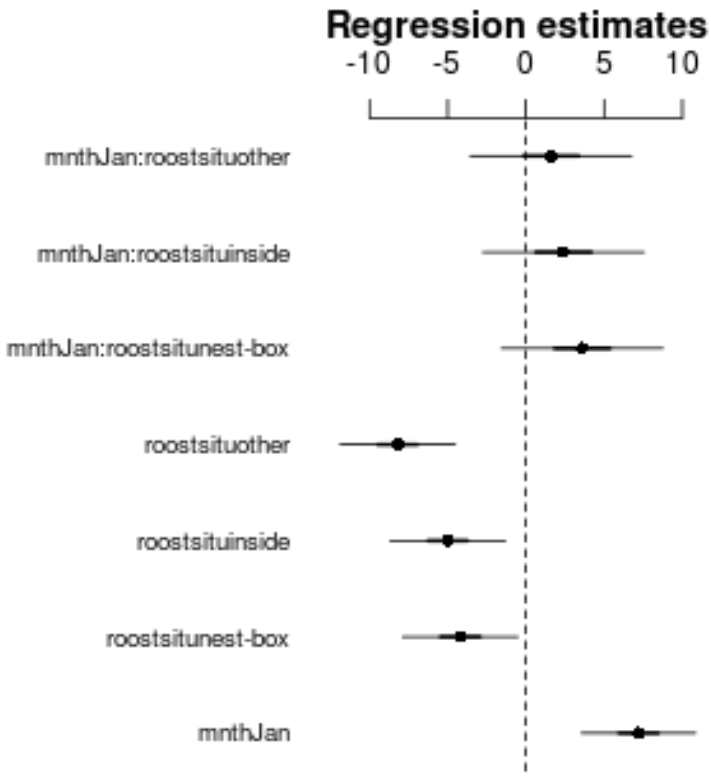

Figure 5: plot of chunk coefplot

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 git differences by subject:

```
library(plyr)
dataf2 <- ddply(dataf,.(subject), function(x) {
    with(x,data.frame(roostsitu,massdiff=diff(stmass)))})
```

Draw a picture:

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

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 formula rather than being expressed separately:

```
detach("package:nlme",unload=TRUE) ## nlme and lme4 don,t like each other
```

```
## Warning: 'nlme' namespace cannot be unloaded: namespace 'nlme' is imported
## by 'lme4.0' so cannot be unloaded
```

```
library(lme4)
```

```
## Warning: the specification for S3 class "family" in package 'lme4' seems
## equivalent to one from package 'lme4.0' and is not turning on duplicate
## class definitions for this class
```

```
## Warning: the specification for class "lmList" in package 'lme4' seems
## equivalent to one from package 'lme4.0' and is not turning on duplicate
## class definitions for this class
```

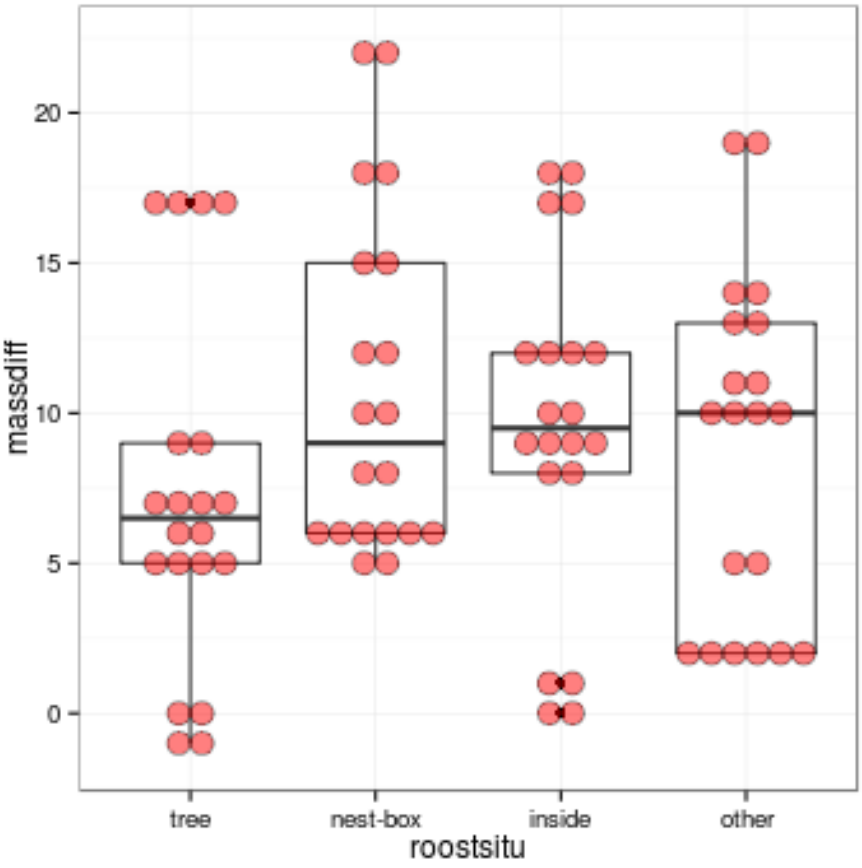

Figure 6: plot of chunk plotdiffs

```
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)
```

(you can leave `verbose` at its default value of `TRUE`).

- 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("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("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) {
```

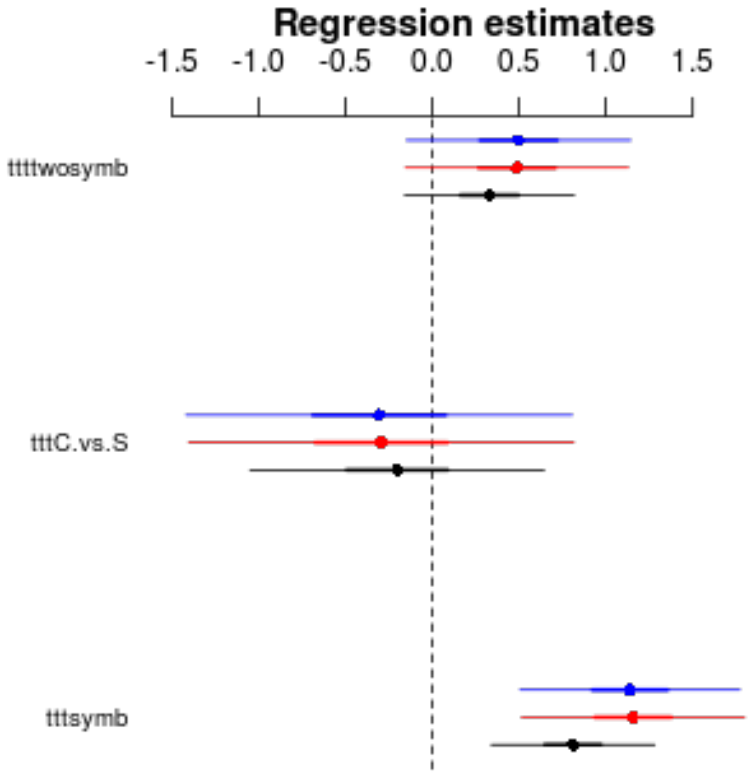

Figure 7: plot of chunk glmmfits

```
    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 ...