

# Post-model-fitting procedures with `glmmTMB` models: diagnostics, inference, and model output

October 14, 2023

The purpose of this vignette is to describe (and test) the functions in various downstream packages that are available for summarizing and otherwise interpreting `glmmTMB` fits. Some of the packages/functions discussed below may not be suitable for inference on parameters of the zero-inflation or dispersion models, but will be restricted to the conditional-mean model.

```
library(glmmTMB)
library(car)
library(emmeans)
library(effects)
library(multcomp)
library(MuMIn)
require(DHARMA, quietly = TRUE) ## may be missing ...
library(broom)
library(broom.mixed)
require(dotwhisker, quietly = TRUE)
library(ggplot2); theme_set(theme_bw())
library(texreg)
library(xtable)
if (huxtable_OK) library(huxtable)
## retrieve slow stuff
L <- gt_load("vignette_data/model_evaluation.rda")
```

A couple of example models:

```
owls_nb1 <- glmmTMB(SiblingNegotiation ~ FoodTreatment*SexParent +
                    (1|Nest)+offset(log(BroodSize)),
                    contrasts=list(FoodTreatment="contr.sum",
                                   SexParent="contr.sum"),
                    family = nbinom1,
                    zi = ~1, data=owls)
```

```
data("cbpp",package="lme4")
cbpp_b1 <- glmmTMB(incidence/size~period+(1|herd),
                  weights=size,family=binomial,
                  data=cbpp)
## simulated three-term Beta example
set.seed(1001)
dd <- data.frame(z=rbeta(1000,shape1=2,shape2=3),
                 a=rnorm(1000),b=rnorm(1000),c=rnorm(1000))
simex_b1 <- glmmTMB(z~a*b*c,family=beta_family,data=dd)
```

# 1 model checking and diagnostics

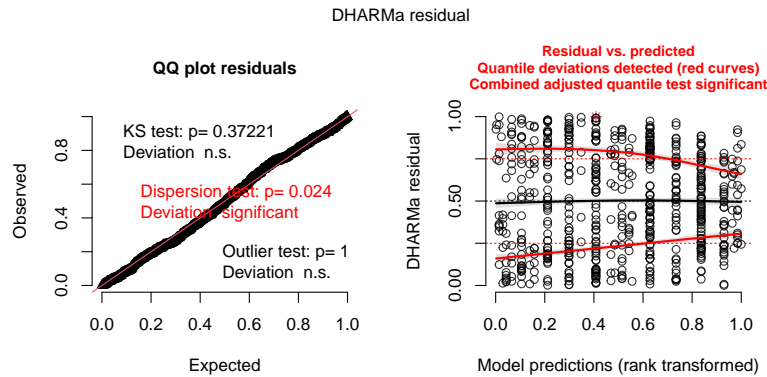
## 1.1 DHARMA

The DHARMA package provides diagnostics for hierarchical models. After running

```
owls_nb1_simres <- simulateResiduals(owls_nb1)
```

you can plot the results:

```
plot(owls_nb1_simres)
```



DHARMa provides lots of other methods based on the simulated residuals:  
 see `vignette("DHARMa", package="DHARMa")`

### 1.1.1 issues

- DHARMa will only work for models using families for which a simulate method has been implemented (in TMB, and appropriately reflected in glmmTMB)

## 2 Inference

### 2.1 `car::Anova`

We can use `car::Anova()` to get traditional ANOVA-style tables from glmmTMB fits. A few limitations/reminders:

- these tables use Wald  $\chi^2$  statistics for comparisons (neither likelihood ratio tests nor  $F$  tests)
- they apply to the fixed effects of the conditional component of the model only (other components *might* work, but haven't been tested at all)
- as always, if you want to do type 3 tests, you should probably set sum-to-zero contrasts on factors and center numerical covariates (see `contrasts` argument above)

```

if (requireNamespace("car") && getRversion() >= "3.6.0") {
  Anova(owls_nb1) ## default type II
  Anova(owls_nb1, type="III")
}

```

Chisq	Df	Pr(>Chisq)
21.4	1	3.66e-06
46.1	1	1.1e-11
0.512	1	0.474
2.29	1	0.13

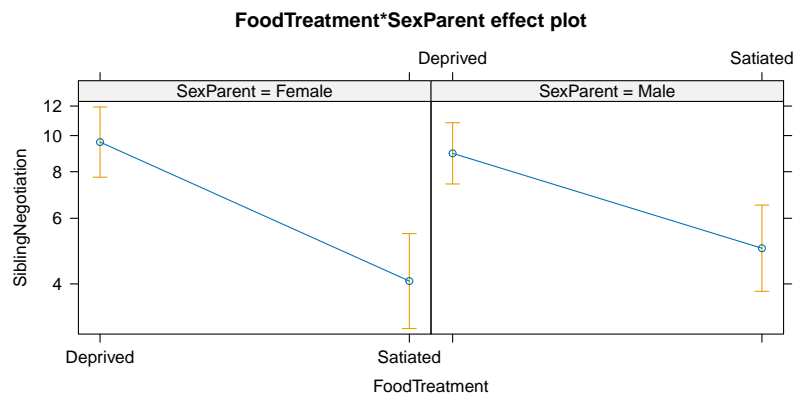
## 2.2 effects

```

effects_ok <- (requireNamespace("effects") && getRversion() >= "3.6.0")
if (effects_ok) {
  (ae <- allEffects(owls_nb1))
  plot(ae)
}

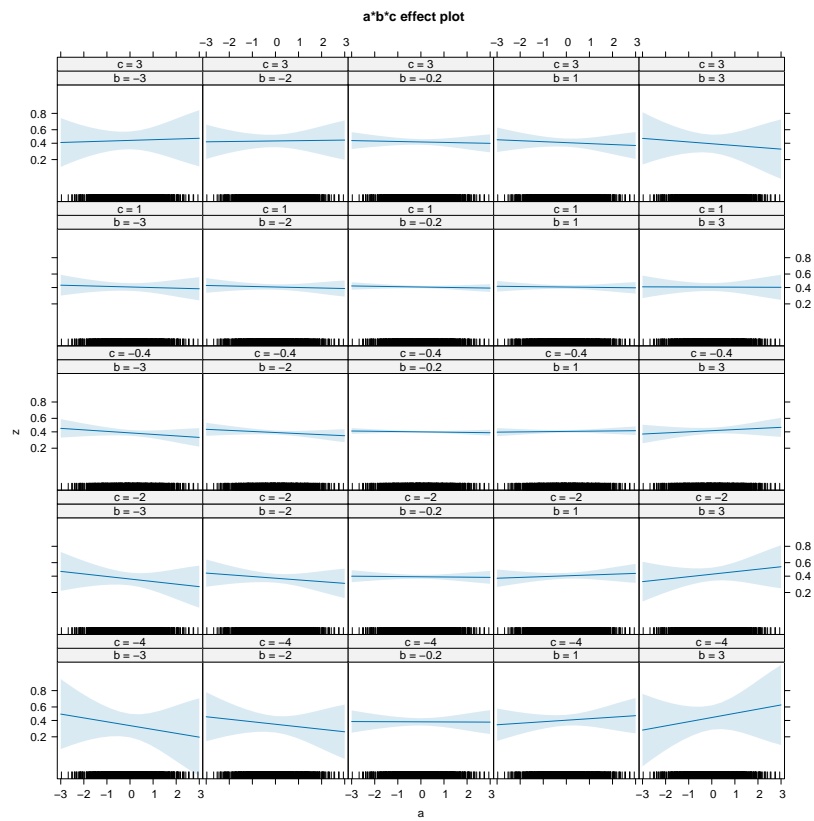
## Warning in Effect.glmmTMB(predictors, mod, vcov. = vcov., ...):
## overriding variance function for effects/dev.resids: computed variances
## may be incorrect

```



(the error can probably be ignored)

```
if (effects_ok) {
  plot(allEffects(simex_b1))
}
```



## 2.3 emmeans

```
emmeans(owls_nb1, poly ~ FoodTreatment | SexParent)

## $emmeans
## SexParent = Female:
##   FoodTreatment emmean      SE   df asymp.LCL asymp.UCL
##   Deprived      2.30 0.1104 Inf      2.09      2.52
##   Satiated      1.44 0.1493 Inf      1.15      1.74
##
## SexParent = Male:
##   FoodTreatment emmean      SE   df asymp.LCL asymp.UCL
##   Deprived      2.23 0.0964 Inf      2.04      2.42
##   Satiated      1.65 0.1357 Inf      1.38      1.91
##
## Results are given on the log (not the response) scale.
## Confidence level used: 0.95
##
## $contrasts
## SexParent = Female:
##   contrast estimate      SE   df z.ratio p.value
##   linear      -0.859 0.149 Inf   -5.776  <.0001
##
## SexParent = Male:
##   contrast estimate      SE   df z.ratio p.value
##   linear      -0.586 0.129 Inf   -4.531  <.0001
##
## Results are given on the log (not the response) scale.
```

Let us also consider a corresponding hurdle model:

```
owls_hnb1 <- update(owls_nb1, family = truncated_nbinom1, ziformula = ~.)
```

On the response scale, this model estimates the means of the component distribution as follows:

```

emmeans(owls_hnb1, ~ FoodTreatment * SexParent, component = "cond", type = "response")

## FoodTreatment SexParent response SE df asymp.LCL asymp.UCL
## Deprived Female 10.04 0.932 Inf 8.37 12.05
## Satiated Female 7.08 0.830 Inf 5.63 8.91
## Deprived Male 9.31 0.716 Inf 8.01 10.83
## Satiated Male 7.37 0.726 Inf 6.08 8.94
##
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale

# --- or ---
emmeans(owls_hnb1, ~ FoodTreatment * SexParent, component = "cmean")

## FoodTreatment SexParent emmean SE df asymp.LCL asymp.UCL
## Deprived Female 10.19 0.888 Inf 8.45 11.93
## Satiated Female 7.46 0.738 Inf 6.02 8.91
## Deprived Male 9.50 0.677 Inf 8.17 10.83
## Satiated Male 7.72 0.653 Inf 6.44 9.00
##
## Confidence level used: 0.95

```

These estimates differ because the first ones are back-transformed from the linear predictor, which is based on the *un-truncated* component distribution, while the second ones are estimates of the means of the *truncated* distribution (with zero omitted). This discrepancy occurs only with hurdle models.

The response means combine both the conditional and the zero-inflation model:

```

emmeans(owls_hnb1, ~ FoodTreatment * SexParent, component = "response")

## FoodTreatment SexParent emmean SE df asymp.LCL asymp.UCL
## Deprived Female 8.86 0.874 Inf 7.14 10.57
## Satiated Female 3.99 0.692 Inf 2.63 5.35
## Deprived Male 8.72 0.668 Inf 7.41 10.03
## Satiated Male 4.74 0.662 Inf 3.44 6.03
##
## Confidence level used: 0.95

```

## 2.4 drop1

`stats::drop1` is a built-in R function that refits the model with various terms dropped. In its default mode it respects marginality (i.e., it will only drop the top-level interactions, not the main effects):

```
system.time(owls_nb1_d1 <- drop1(owls_nb1, test="Chisq"))
```

```
##      user  system elapsed  
##    0.741    0.002    0.742
```

```
print(owls_nb1_d1)
```

```
## Single term deletions  
##  
## Model:  
## SiblingNegotiation ~ FoodTreatment * SexParent + (1 | Nest) +  
##      offset(log(BroodSize))  
##  
##              Df      AIC      LRT Pr(>Chi)  
## <none>              3383.6  
## FoodTreatment:SexParent  1 3383.9 2.2766  0.1313
```

In principle, using `scope = . ~ . - (1|Nest)` should work to execute a “type-3-like” series of tests, dropping the main effects one at a time while leaving the interaction in (we have to use `-(1|Nest)` to exclude the random effects because `drop1` can’t handle them). However, due to the way that R handles formulas, dropping main effects from an interaction of *\*factors\** has no effect on the overall model. (It would work if we were testing the interaction of continuous variables.)

### 2.4.1 issues

The `mixed` package implements a true “type-3-like” parameter-dropping mechanism for `[g]lmer` models. Something like that could in principle be applied here.



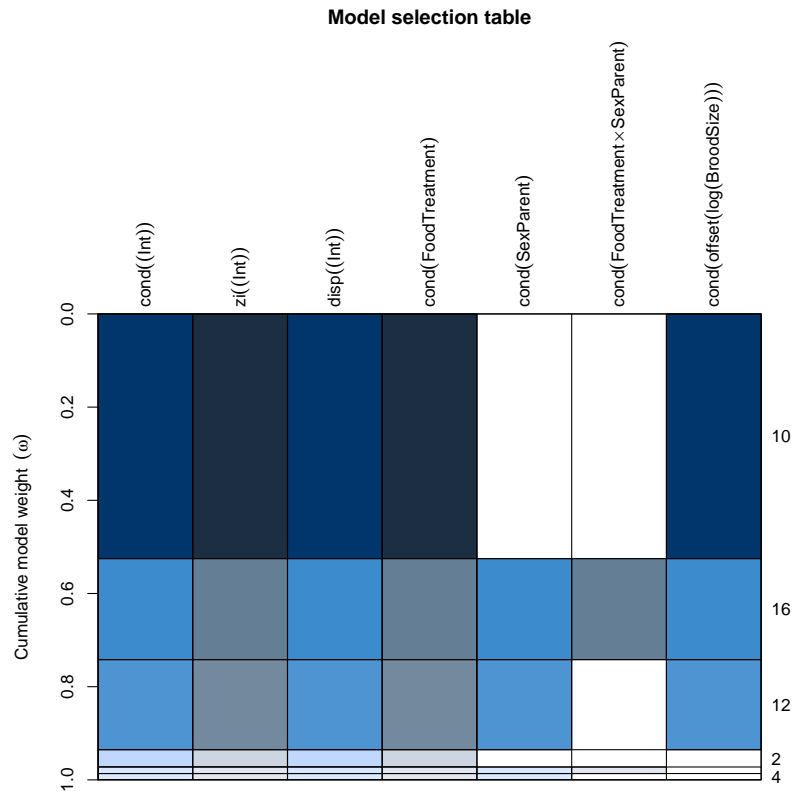
## 2.5 Model selection and averaging with MuMin

We can run `MuMin::dredge(owls_nb1)` on the model to fit all possible submodels. Since this takes a little while (45 seconds or so), we've instead loaded some previously computed results:

```
print(owls_nb1_dredge)

## Global model call: glmmTMB(formula = SiblingNegotiation ~ FoodTreatment * SexPa
##      (1 | Nest) + offset(log(BroodSize)), data = Owls, family = nbinom1,
##      ziformula = ~1, contrasts = list(FoodTreatment = "contr.sum",
##      SexParent = "contr.sum"), na.action = na.fail, dispformula = ~1)
## ---
## Model selection table
##      cnd((Int)) zi((Int)) dsp((Int)) cnd(FdT) cnd(SxP) cnd(FdT:SxP)
## 10      0.4284    -2.094           +         +
## 16      0.4275    -2.055           +         +         +         +
## 12      0.4257    -2.100           +         +         +
## 2       1.8290    -1.990           +         +
## 8       1.8280    -1.955           +         +         +         +
## 4       1.8260    -1.996           +         +         +
## 9       0.6295    -1.373           +
## 1       2.0980    -1.232           +
## 11      0.6220    -1.381           +         +
## 3       2.0920    -1.236           +         +
##      cnd(off(log(BrS))) df      logLik    AICc delta weight
## 10              + 5 -1685.978 3382.1 0.00 0.525
## 16              + 7 -1684.819 3383.8 1.77 0.217
## 12              + 6 -1685.957 3384.1 2.00 0.193
## 2                5 -1688.628 3387.4 5.30 0.037
## 8                7 -1687.556 3389.3 7.24 0.014
## 4                6 -1688.610 3389.4 7.30 0.014
## 9                + 4 -1708.573 3425.2 43.15 0.000
## 1                4 -1708.672 3425.4 43.35 0.000
## 11              + 5 -1708.420 3426.9 44.88 0.000
## 3                5 -1708.509 3427.1 45.06 0.000
## Models ranked by AICc(x)
## Random terms (all models):
##      cond(1 | Nest)
```

```
op <- par(mar=c(2,5,14,3))
plot(owls_nb1_dredge)
```



```
par(op) ## restore graphics parameters
```

Model averaging:

```
model.avg(owls_nb1_dredge)
```

```
##
## Call:
## model.avg(object = owls_nb1_dredge)
##
## Component models:
## '14'      '1234'    '124'      '1'        '123'      '12'      '4'        '(Null)'
```

```
## '24'      '2'
##
## Coefficients:
##          cond((Int)) cond(FoodTreatment1) zi((Int)) cond(SexParent1)
## full      0.5183099          0.353877 -2.079432      -0.009556203
## subset    0.5183099          0.353877 -2.079432      -0.021827791
##          cond(FoodTreatment1:SexParent1)
## full              0.01569108
## subset            0.06797533
```

### 2.5.1 issues

- may not work for Beta models because the `family` component (“beta”) is not identical to the name of the family function (`beta_family()`)? (Kamil Bartoń, pers. comm.)

## 2.6 multcomp for multiple comparisons and *post hoc* tests

```
g1 <- glht(cbpp_b1, linfct = mcp(period = "Tukey"))
summary(g1)

##
## Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: glmmTMB(formula = incidence/size ~ period + (1 | herd), data = cbpp,
## family = binomial, weights = size, ziformula = ~0, dispformula = ~1)
##
## Linear Hypotheses:
##          Estimate Std. Error z value Pr(>|z|)
## 2 - 1 == 0  -0.9923     0.3066  -3.236  0.00638 **
## 3 - 1 == 0  -1.1287     0.3266  -3.455  0.00283 **
## 4 - 1 == 0  -1.5803     0.4274  -3.697  0.00111 **
```

```
## 3 - 2 == 0 -0.1363      0.3807 -0.358  0.98368
## 4 - 2 == 0 -0.5880      0.4703 -1.250  0.58569
## 4 - 3 == 0 -0.4516      0.4843 -0.933  0.78117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

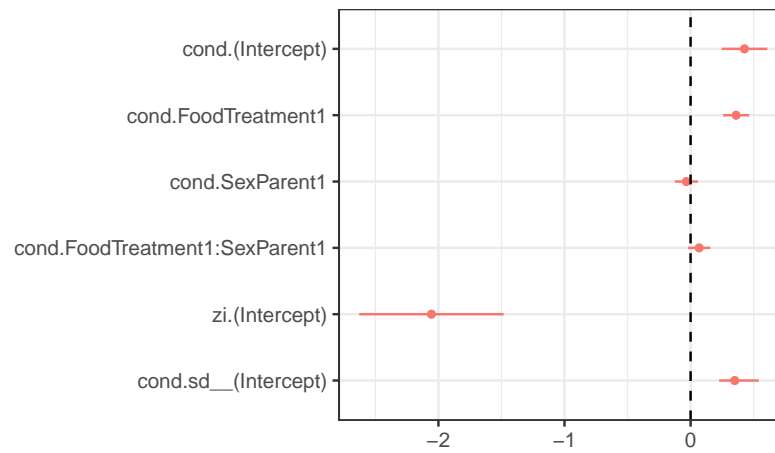
## 3 Extracting coefficients, coefficient plots and tables

### 3.1 broom and friends

The `broom` and `broom.mixed` packages are designed to extract information from a broad range of models in a convenient (tidy) format; the `dotwhisker` package builds on this platform to draw elegant coefficient plots.

```
if (requireNamespace("broom.mixed") && requireNamespace("dotwhisker")) {
  t1 <- broom.mixed::tidy(owls_nb1, conf.int = TRUE)
  t1 <- transform(t1,
                  term=sprintf("%s.%s", component, term))

  if (packageVersion("dotwhisker") > "0.4.1") {
    dw <- dwplot(t1)
  } else {
    owls_nb1$coefficients <- TRUE ## hack!
    dw <- dwplot(owls_nb1, by_2sd=FALSE)
  }
  print(dw+geom_vline(xintercept=0, lty=2))
}
```



### 3.1.1 issues

(these are more general `dwplot` issues)

- use black rather than `color(1)` when there's only a single model, i.e. only add `aes(colour=model)` conditionally? - draw points even if std err / confint are NA (draw `geom_point()` as well as `geom_pointrange()` ? need to apply all aesthetics, dodging, etc. to both ...)
- for glmmTMB models, allow labeling by component? or should this be done by manipulating the tidied frame first? (i.e.: `tidy(.) %>% tidyr::unite(term,c(co`

## 3.2 coefficient tables with `xtable`

The `xtable` package can output data frames as  $\text{\LaTeX}$  tables; this isn't quite as elegant as `stargazer` etc., but is not a bad start. I've sprinkled lots of hard line-breaks, spaces, and newlines in below: someone who was better at  $\text{\TeX}$  could certainly do a better job. (`xtable` can also produce HTML output.)

```
ss <- summary(owls_nb1)
## print table; add space,
pxt <- function(x,title) {
  cat(sprintf("{\n\n\\textbf{%s}\n\\ \\vspace{2pt}\\ \\n",title))
```



	Model 1
(Intercept)	0.43*** (0.09)
FoodTreatment1	0.36*** (0.05)
SexParent1	-0.03 (0.05)
FoodTreatment1:SexParent1	0.07 (0.05)
zi_(Intercept)	-2.06*** (0.29)

\*\*\* $p < 0.001$ ; \*\* $p < 0.01$ ; \* $p < 0.05$

Table 1: Owls model

```
source(system.file("other_methods", "extract.R", package="glmmTMB"))
texreg(owls_nb1, caption="Owls model", label="tab:owls")
```

See output in Table 1.

### 3.4 coefficient tables with huxtable

The `huxtable` package allows output in either  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  or HTML: this example is tuned for  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .

```
if (!huxtable_OK) {
  cat("Sorry, huxtable+LaTeX is unreliable on this platform; skipping\n")
} else {
  cc <- c("intercept (mean)"="(Intercept)",
        "food treatment (starvation)"="FoodTreatment1",
        "parental sex (M)"="SexParent1",
        "food  $\times$  sex"="FoodTreatment1:SexParent1")
  h0 <- huxreg(" " = owls_nb1, # give model blank name so we don't get '(1)'
    tidy_args = list(effects="fixed"),
    coefs = cc,
    error_pos = "right",
```

```

        statistics = "nobs" # don't include logLik and AIC
      )
names(h0)[2:3] <- c("estimate", "std. err.")
## allow use of math notation in name
h1 <- set_cell_properties(h0, row=5, col=1, escape_contents=FALSE)
cat(to_latex(h1, tabular_only=TRUE))
}

```

---

intercept (mean)	0.427 ***	(0.092)
food treatment (starvation)	0.361 ***	(0.053)
parental sex (M)	-0.033	(0.047)
food × sex	0.068	(0.045)
nobs	599	

---

\*\*\*  $p < 0.001$ ; \*\*  $p < 0.01$ ; \*  $p < 0.05$ .

### 3.4.1 issues

- `huxtable` needs quite a few additional L<sup>A</sup>T<sub>E</sub>X packages: use `report_latex_dependencies()` to see what they are.

## 4 influence measures

*Influence measures* quantify the effects of particular observations, or groups of observations, on the results of a statistical model; *leverage* and *Cook's distance* are the two most common formats for influence measures. If a projection matrix (or “hat matrix”) is available, influence measures can be computed efficiently; otherwise, the same quantities can be estimated by brute-force methods, refitting the model with each group or observation successively left out.

We’ve adapted the `car::influence.merMod` function to handle `glmmTMB` models; because it uses brute force, it can be slow, especially if evaluating the influence of individual observations. For now, it is included as a separate



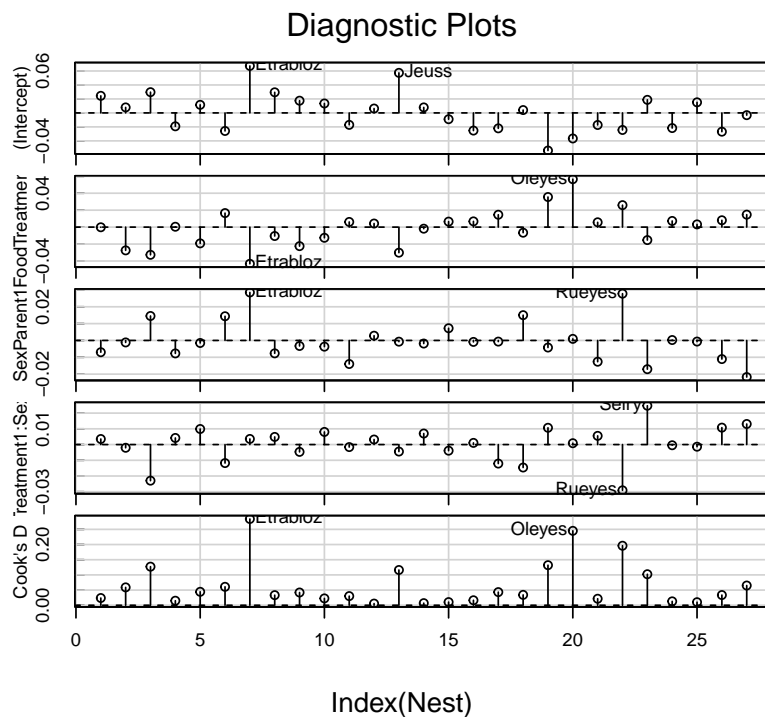
source file rather than exported as a method (see below), although it may be included in the package (or incorporated in the `car` package) in the future.

```
source(system.file("other_methods", "influence_mixed.R", package="glmmTMB"))
```

```
owls_nb1_influence_time <- system.time(  
  owls_nb1_influence <- influence_mixed(owls_nb1, groups="Nest")  
)
```

Re-fitting the model with each of the 27 nests excluded takes 14 seconds (on an old Macbook Pro). The `car::infIndexPlot()` function is one way of displaying the results:

```
car::infIndexPlot(owls_nb1_influence)
```

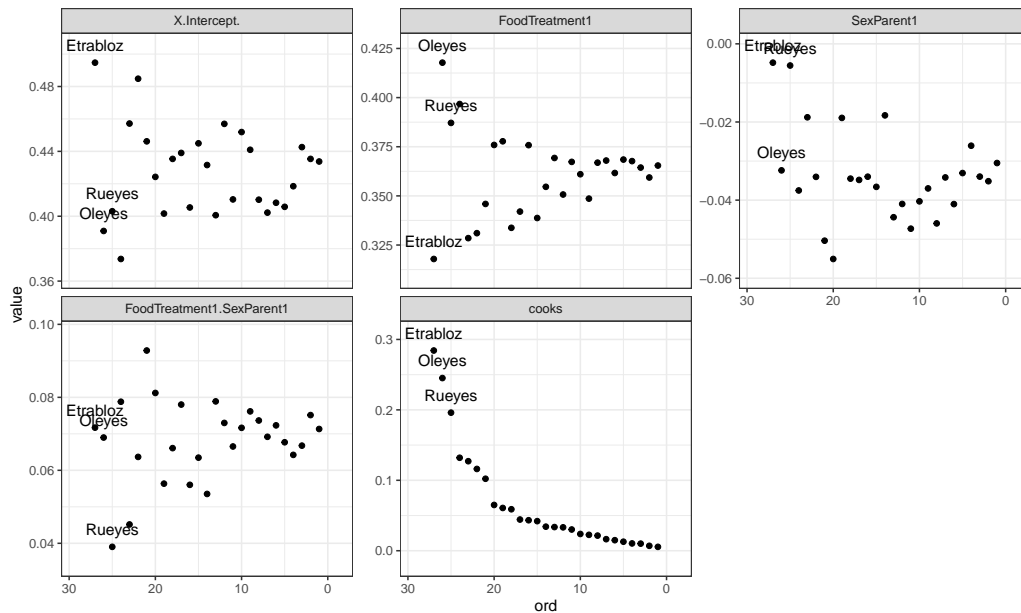


Or, you can transform the results and plot them however you like:

```

inf <- as.data.frame(owls_nb1_influence[["fixed.effects[-Nest]"]])
inf <- transform(inf,
                 nest=rownames(inf),
                 cooks=cooks.distance(owls_nb1_influence))
inf$ord <- rank(inf$cooks)
if (require(reshape2)) {
  inf_long <- melt(inf, id.vars=c("ord","nest"))
  gg_infl <- (ggplot(inf_long,aes(ord,value))
    + geom_point()
    + facet_wrap(~variable, scale="free_y")
    ## n.b. may need expand_scale() in older ggplot versions ?
    + scale_x_reverse(expand=expansion(mult=0.15))
    + scale_y_continuous(expand=expansion(mult=0.15))
    + geom_text(data=subset(inf_long,ord>24),
                aes(label=nest),vjust=-1.05)
  )
  print(gg_infl)
}

```



## 5 to do

- more plotting methods ( `sjplot` )
- output with `memisc`
- AUC etc. with `ModelMetrics`