Getting started with the `glmmADMB` package

Ben Bolker, Hans Skaug, Arni Magnusson, Anders Nielsen

March 27, 2012

1 Introduction/quick start

`glmmADMB` is a package, built on the open source AD Model Builder nonlinear fitting engine, for fitting generalized linear mixed models and extensions.

- response distributions: Poisson, binomial, negative binomial (NB1 and NB2 parameterizations), Gamma, Beta, truncated Poisson and negative binomial; Gaussian; logistic
- link functions: log, logit, probit, complementary log-log (`"cloglog"`), inverse, identity
- zero-inflation (models with a constant zero-inflation value only); hurdle models via truncated Poisson/NB
- single or multiple (nested or crossed) random effects
- offsets
- post-fit MCMC chain for characterizing uncertainty

As of version 0.6.5, the package has been greatly revised to allow a wider range of response and link functions and to allow models with multiple random effects. For now, the resulting package is slower than the old (single-random-effect version), but we hope to increase its speed in the future.

In order to use `glmmADMB` effectively you should already be reasonably familiar with generalized linear mixed models (GLMMs), which in turn requires familiarity with (i) generalized linear models (e.g. the special cases of logistic, binomial, and Poisson regression) and (ii) 'modern' mixed models (those working via maximization of the marginal likelihood rather than by manipulating sums of squares).

In order to fit a model in `glmmADMB` you need to:

- specify a model for the fixed effects, in the standard R (Wilkinson-Rogers) formula notation (see `?formula` or Section 11.1 of the Introduction to R. Formulae can also include *offsets*.

- specify a model for the random effects, in the notation that is common to the `nlme` and `lme4` packages. Random effects are specified as `e|g`, where `e` is an effect and `g` is a grouping factor (which must be a factor variable, or a nesting of/interaction among factor variables). For example, the formula would be `1|block` for a random-intercept model or `time|block` for a model with random variation in slopes through time across groups specified by `block`. A model of nested random effects (block within site) would be `1|site/block`; a model of crossed random effects (block and year) would be `(1|block)+(1|year)`.

  Random effects can be specified either in a separate `random` argument (as in `nlme`) or as part of the model formula (as in `lme4`).

- choose the error distribution by specifying the family (as a string: e.g. `"poisson"` or `"binomial"`)

- specify a link function (as a string: e.g. `"logit"` or `"log"`.

- optionally specify that zero-inflation is present `zeroInflation=TRUE`. In the current version, zero-inflation can only be specified as a single constant term across the entire model — i.e. it cannot vary across groups or with covariates.

This document was generated using R Under development (unstable) (2012-03-04 r58577) and package versions:

```
       bbmle          coda     coefplot2       ggplot2      glmmADMB
     1.0.4.2        0.14-6         0.1.3         0.9.0     0.7.2.10
        lme4          MASS     scapeMCMC
0.999902344-0       7.3-17         1.1-3
```

```
> citation("glmmADMB")

To cite package 'glmmADMB' in publications use:

Fournier DA, Skaug HJ, Ancheta J, Ianelli J, Magnusson A, Maunder M,
Nielsen A and Sibert J (2012). ■AD Model Builder: using automatic
differentiation for statistical inference of highly parameterized
complex nonlinear models.■ _Optim. Methods Softw._, *27*, pp. 233-249.

Skaug H, Fournier D, Nielsen A, Magnusson A and Bolker B ($ Date $).
_Generalized Linear Mixed Models Using AD Model Builder_. R package
version 0.7.2.10.
```

2 Owls data

These data, taken from [3] and ultimately from [2], quantify the number of negotiations among owlets (owl chicks) in different nests *prior* to the arrival of a provisioning parent as a function of food treatment (deprived or satiated), the sex of the parent, and arrival time. The total number of calls from the nest is recorded, along with the total brood size, which is used as an offset to allow the use of a Poisson response.

Since the same nests are measured repeatedly, the nest is used as a random effect. The model can be expressed as a zero-inflated generalized linear mixed model (ZIGLMM).

First we draw some pictures (Figures 1, 2).

Load the `glmmADMB` package to get access to the `Owls` data set; load the `ggplot2` graphics package.

```
> library("glmmADMB")
> library("ggplot2")
```

Various small manipulations of the data set: (1) reorder nests by mean negotiations per chick, for plotting purposes; (2) add log brood size variable (for offset); (3) rename response variable.

```
> Owls <- transform(Owls,
                    Nest=reorder(Nest,NegPerChick),
                    NCalls=SiblingNegotiation)
```

(If you were really using this data set you should start with `summary(Owls)` to see what columns are there and what their characteristics are.)

Now fit some models:

The basic `glmmadmb` fit — a zero-inflated Poisson model.

```
> fit_zipoiss <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                                   offset(log(BroodSize))+(1|Nest),
                                   data=Owls,
                                   zeroInflation=TRUE,
                                   family="poisson")

> summary(fit_zipoiss)

Call:
glmmadmb(formula = NCalls ~ (FoodTreatment + ArrivalTime) * SexParent +
    offset(log(BroodSize)) + (1 | Nest), data = Owls, family = "poisson",
    zeroInflation = TRUE)


Coefficients:
                                    Estimate Std. Error z value Pr(>|z|)
(Intercept)                           2.8562     0.3871    7.38  1.6e-13 ***
FoodTreatmentSatiated                -0.3314     0.0635   -5.22  1.8e-07 ***
ArrivalTime                          -0.0807     0.0156   -5.18  2.3e-07 ***
SexParentMale                         0.2882     0.3575    0.81     0.42
FoodTreatmentSatiated:SexParentMale   0.0740     0.0761    0.97     0.33
```

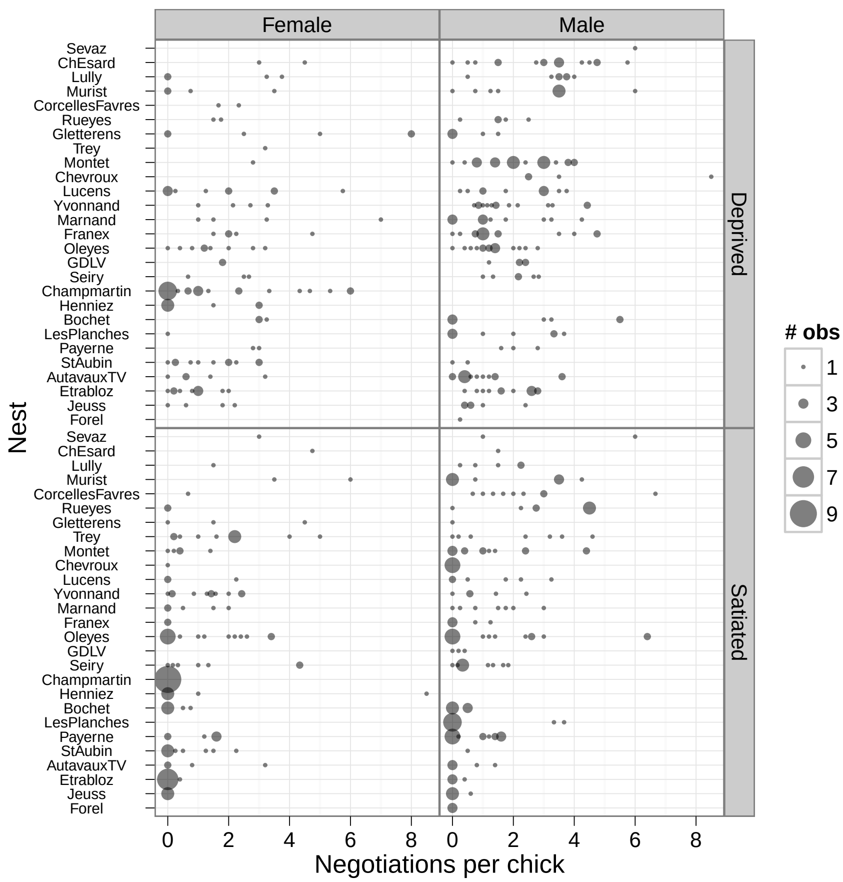


Figure 1: Basic view of owl data (arrival time not shown).

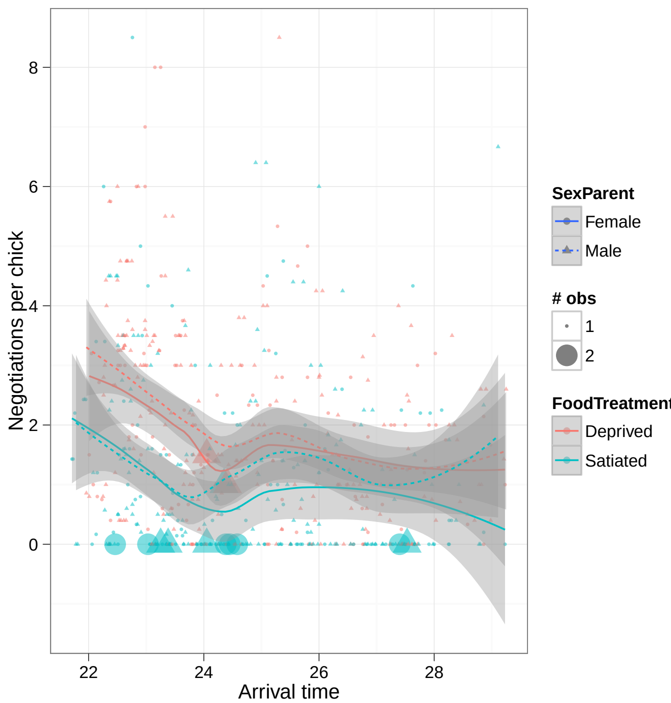


Figure 2: Basic view of owl data, #2 (nest identity not shown)

```
ArrivalTime:SexParentMale           -0.0150     0.0143   -1.05     0.29
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Number of observations: total=599, Nest=27
Random effect variance(s):
Group=Nest
            Variance StdDev
(Intercept)     0.14 0.3742
Zero-inflation: 0.25833  (std. err.:  0.018107 )

Log-likelihood: -1985.3
```

The `coefplot2` package knows about `glmmadmb` fits:

```
> library("coefplot2")
> coefplot2(fit_zipoiss)
```

We can also try a standard zero-inflated negative binomial model; the default is the "NB2" parameterization (variance = $\mu(1 + \mu/k)$).

```
> fit_zinbinom <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                    offset(log(BroodSize))+(1|Nest),
                    data=Owls,
                    zeroInflation=TRUE,
                    family="nbinom")
```

Alternatively, use an "NB1" fit (variance = $\phi\mu$).

```
> fit_zinbinom1 <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                                     offset(log(BroodSize))+(1|Nest),
                                     data=Owls,
                                     zeroInflation=TRUE,
                                     family="nbinom1")
```

Relax the assumption that total number of calls is strictly proportional to brood size (i.e. using log(brood size) as an offset):

```
> fit_zinbinom1_bs <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                                     BroodSize+(1|Nest),
                                     data=Owls,
                                     zeroInflation=TRUE,
                                     family="nbinom1")
```

Every change we have made so far improves the fit — changing distributions improves it enormously, while changing the role of brood size makes only a modest (-1 AIC unit) difference:

```
> library("bbmle")
> AICtab(fit_zipoiss,fit_zinbinom,fit_zinbinom1,fit_zinbinom1_bs)
```

```
                dAIC  df
fit_zinbinom1_bs   0.0 10
fit_zinbinom1      1.2 9
fit_zinbinom      68.7 9
fit_zipoiss      637.0 8
```

Compare the parameter estimates:

```
> vn <- c("food","arrivaltime","sex","food:sex","arrival:sex","broodsize")
> coefplot2(list(ZIP=fit_zipoiss,
                 ZINB=fit_zinbinom,
                 ZINB1=fit_zinbinom1,
                 ZINB1_brood=fit_zinbinom1_bs),
           varnames=vn,
           legend=TRUE)
```

2.1 Hurdle models

In contrast to zero-inflated models, hurdle models treat zero-count and non-zero outcomes as two completely separate categories, rather than treating the zero-count outcomes as a mixture of structural and sampling zeros.

As of version 0.6.7.1, `glmmADMB` includes truncated Poisson and negative binomial familes and hence can fit hurdle models. The two parts of the model have to be fitted separately, however. First we fit a truncated distribution to the non-zero outcomes:

```
> fit_hnbinom1 <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                               BroodSize+(1|Nest),
                               data=subset(Owls,NCalls>0),
                               family="truncnbinom1")
```

Then we fit a model to the binary part of the data (zero vs. non-zero). In this case, I started by fitting a simple (intercept-only) model with intercept-level random effects only. This comes a bit closer to matching the previous (zero-inflation) models, which treated zero-inflation as a single constant level across the entire data set (in fact, leaving out the random effects and just using `glmmADMB(nz~1,data=Owls,family="binomial")`, or `glm(nz~1,data=Owls,family="binomial")`, would be an even closer match). I then fitted a more complex binary model — this is all a matter of judgment about how complex a model it's worth trying to fit to a given data set — but it does look as though the zero-inflation varies with arrival time and satiation.

```
> Owls$nz <- as.numeric(Owls$NCalls>0)
> fit_count <- glmmadmb(nz~1+(1|Nest),
                               data=Owls,
                               family="binomial")
> fit_ccount <- glmmadmb(nz~(FoodTreatment+ArrivalTime)*SexParent+(1|Nest),
```

```
                          data=Owls,
                          family="binomial")
> AICtab(fit_count,fit_ccount)

           dAIC df
fit_ccount  0.0 7
fit_count  84.1 2

> summary(fit_ccount)

Call:
glmmadmb(formula = nz ~ (FoodTreatment + ArrivalTime) * SexParent +
    (1 | Nest), data = Owls, family = "binomial")


Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)
(Intercept)                          7.3108     2.1577    3.39   0.0007 ***
FoodTreatmentSatiated               -1.8250     0.3479   -5.25  1.6e-07 ***
ArrivalTime                         -0.2171     0.0846   -2.57   0.0102 *
SexParentMale                        2.7699     3.0248    0.92   0.3598
FoodTreatmentSatiated:SexParentMale -0.3646     0.4740   -0.77   0.4418
ArrivalTime:SexParentMale           -0.0821     0.1175   -0.70   0.4849
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Number of observations: total=599, Nest=27
Random effect variance(s):
Group=Nest
            Variance StdDev
(Intercept)    1.423  1.193

Log-likelihood: -283.095
```

2.2 Testing and inference

(Sketchy: to be expanded.)

There are many challenging statistical issues surrounding tests of terms in GLMMs. Most often people use minor variations of existing approaches (Wald tests, likelihood ratio tests, etc.), either accounting for or sweeping under the rug some of the differences that should be accounted for when moving either from linear mixed models (LMMs) or generalized linear models (GLMs) to GLMMs. See `http://glmm.wikidot.com/faq`, or a good book on GLMMs (!!), for a discussion of these issues.

In the meantime, keeping the limitations in mind, you can

- use AIC to select models or generated weighted predictions (see example above).

- use anova to perform a likelihood ratio test:

```
> anova(fit_zipoiss,fit_zinbinom)

Analysis of Deviance Table

Model 1: NCalls ~ (FoodTreatment + ArrivalTime) * SexParent + offset(log(BroodSize))
Model 2: NCalls ~ (FoodTreatment + ArrivalTime) * SexParent + offset(log(BroodSize))
  NoPar  LogLik Df Deviance  Pr(>Chi)
1     8 -1985.3
2     9 -1700.1  1   570.32 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- use Anova from the car package to generate Wald tests:

```
> car::Anova(fit_zinbinom)

Analysis of Deviance Table (Type II tests)

Response: NCalls
                         Df  Chisq Pr(>Chisq)
FoodTreatment             1 8.8407   0.002946 **
ArrivalTime               1 8.5290   0.003495 **
SexParent                 1 0.1450   0.703340
FoodTreatment:SexParent   1 0.9953   0.318442
ArrivalTime:SexParent     1 0.1112   0.738736
Residuals               590
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- coefplot2
- drop1??,
- simulate??,
- parametric bootstrapping ??

2.3 Integration with lme4

```
> library("lme4")
> gm1_lme4 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
                    data = cbpp, family = binomial)
> gm1_glmmADMB <- glmmadmb(cbind(incidence, size - incidence) ~ period + (1 | herd),
                    data = cbpp, family = "binomial")
> ## sessionInfo()
> fixef(gm1_lme4)
```

```
(Intercept)     period2     period3     period4
  -1.375833   -1.057875   -1.196262   -1.638262
```

```
> fixef(gm1_glmmADMB) ## or coef()
```

```
(Intercept)     period2     period3     period4
   -1.39850    -0.99233    -1.12870    -1.58030
```

```
> unlist(ranef(gm1_lme4))
```

```
 herd.(Intercept)1  herd.(Intercept)2  herd.(Intercept)3  herd.(Intercept)4
        0.67444277        -0.43998913         0.57431817         0.07101472
 herd.(Intercept)5  herd.(Intercept)6  herd.(Intercept)7  herd.(Intercept)8
       -0.56297767        -0.59995895         1.23827285         0.76489735
 herd.(Intercept)9 herd.(Intercept)10 herd.(Intercept)11 herd.(Intercept)12
       -0.36259804        -0.79481945        -0.09280176        -0.04105403
herd.(Intercept)13 herd.(Intercept)14 herd.(Intercept)15
       -1.05820835         1.61319820        -0.94152085
```

```
> unlist(ranef(gm1_glmmADMB))
```

```
      herd1       herd2       herd3       herd4       herd5       herd6
 0.59113330 -0.29951639  0.40681651  0.03938333 -0.19037992 -0.40116575
      herd7       herd8       herd9      herd10      herd11      herd12
 0.89084943  0.60033432 -0.23844043 -0.54209341 -0.08476149 -0.06501281
     herd13      herd14      herd15
-0.69149400  0.97261479 -0.53178414
```

```
> VarCorr(gm1_lme4)
```

```
 Groups Name        Variance Std.Dev.
 herd   (Intercept) 0.60406  0.77721
```

```
> detach("package:lme4")  ## kluge!
> VarCorr(gm1_glmmADMB)
```

```
Group=herd
            Variance StdDev
(Intercept)   0.4152 0.6443
```

2.3.1 Convert glmmADMB parameters to lme4

this section is UNDER CONSTRUCTION

```
> library("lme4")  ## requires DEVELOPMENT version of lme4 ...
> new_lme4 <- packageVersion("lme4")>"0.999375.42"
> if (new_lme4) {
     lme4fun <- update(gm1_lme4,devFunOnly=TRUE)
     deviance(gm1_lme4)
```

```
    lme4fun(c(0,fixef(gm1_lme4)))  ## variance set to zero
    v1 <- getME(gm1_lme4,"theta")  ## log-Cholesky factor: equal in this case to standard d
    lme4fun(c(v1,fixef(gm1_lme4))) ## observed variance (*almost* identical)
    v2 <- sqrt(gm1_glmmADMB$S[[1]])
    lme4fun(c(v2,fixef(gm1_lme4))) ## ???
}

[1] 228.0689

> ## FIXME: figure out why this crashes knitr in non-interactive mode!!
> ## (then turn evaluation back on)
> glmmadmbfun <- function(pars,minval=exp(-9.5),verbose=FALSE) {
    if (pars[1]==0) {
        warning(sprintf("variance parameter set to min val (%f)",minval))
        pars[1] <- minval
    }
    g0 <- glmmadmb(cbind(incidence, size - incidence) ~ period + (1 | herd),
                   data = cbpp, family = "binomial",
                   extra.args=c("-maxfn 1 -phase 6"),
                   verbose=verbose,
                   start=list(RE_sd=log(pars[1]), fixed=pars[-1]))
    -logLik(g0)
}
> ## glmmadmbfun(c(0,fixef(gm1_lme4)))  ## fails
> v2B <- glmmadmbfun(c(v2,fixef(gm1_lme4)))
> v1B <- glmmadmbfun(c(v1,fixef(gm1_lme4)))
> all.equal(v1B,v2B,-logLik(gm1_glmmADMB))
```

2.4 MCMC fitting

AD Model Builder has the capability to run a *post hoc* Markov chain to assess variability — that is, it uses the MLE as a starting point and the estimated sampling distribution (variance-covariance matrix) of the parameters as a candidate distribution, and "jumps around" the parameter space in a consistent way (Metropolis-Hastings?) to generate a series of samples from a posterior distribution of the parameter distribution (assuming flat priors: please see the ADMB documentation, or [1], for more details).

This is very convenient, but tends to be a bit slow. In the example below, I ran a chain of 50,000 MCMC iterations — on examination, the default chain of 1000 iterations was much too short — which took about 1.04 hours on a modern (2012) desktop.

```
> OwlModel_nb1_bs_mcmc <- glmmadmb(NCalls~(FoodTreatment+ArrivalTime)*SexParent+
                                      BroodSize+(1|Nest),
                                      data=Owls,
                                      zeroInflation=TRUE,
                                      family="nbinom1",
```

```
                                    mcmc=TRUE,
                                    mcmc.opts=mcmcControl(mcmc=50000))
```

Convert the MCMC chain to an `mcmc` object which the `coda` package can handle:

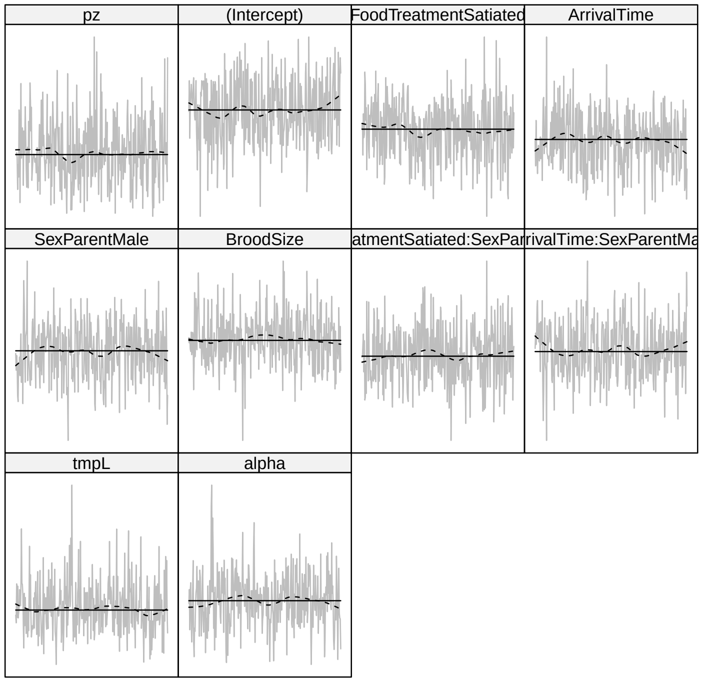
```
> library("coda")
> m <- as.mcmc(OwlModel_nb1_bs_mcmc$mcmc)
```

```
> mcmc_transform <- function(m,fit) {
   if (missing(fit)) {
     fit0 <- fit
     m <- fit$mcmc
     fit <- fit0
   }
   if (!is(m,"mcmc")) stop("m must be an 'mcmc' object")
   if (!is(fit,"glmmadmb")) stop("fit must a 'glmmadmb' object")
   ## zero-inflation
   pz <- m[,"pz",drop=FALSE]
   t_pz <- pz  ## (not transformed)
   ## fixed effects
   fixed <- m[,grep("^beta",colnames(m)),drop=FALSE]
   t_fixed <- as.mcmc(fixed %*% fit$phi)
   colnames(t_fixed) <- names(fixef(fit))
   ## variance parameters: log std dev
   theta <- m[,grep("^tmpL",colnames(m)),drop=FALSE]
   t_theta <- exp(theta)
   ## corr parameters ("offdiagonal elements of cholesky-factor of correlation matrix")
   corr <- m[,grep("^tmpL1",colnames(m)),drop=FALSE]
   t_corr <- corr
   ## scale/overdispersion parameter
   logalpha <- m[,grep("^log_alpha",colnames(m)),drop=FALSE]
   t_alpha <- matrix(exp(logalpha),dimnames=list(NULL,"alpha"))
   ## random effects
   re <- m[,grep("^u\\\\.[0-9]+",colnames(m)),drop=FALSE]
   t_re <- re
   mcmc(cbind(t_pz,t_fixed,t_theta,t_corr,t_alpha,t_re),
        start=start(m),end=end(m),thin=frequency(m))
 }
```

Look at the trace plots. (Something a bit odd happens at the end of the chain, so we drop the last few values . . . there may be a bug in the import-handling for MCMC for very long chains . . .)

```
> tm <- window(mcmc_transform(m,OwlModel_nb1_bs),1,320)
```

```
> library("scapeMCMC")
> plotTrace(tm)
```



The Geweke diagnostic gives Z scores for each variable for a comparison between (by default) the first 10% and last 50% of the chain

```
> (gg <- geweke.diag(tm))

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

                                   pz                           (Intercept)
                               0.7191                               -0.0986
                FoodTreatmentSatiated                           ArrivalTime
                               1.6912                                0.0370
                        SexParentMale                             BroodSize
                              -0.3944                                0.1543
FoodTreatmentSatiated:SexParentMale             ArrivalTime:SexParentMale
                              -0.6721                                0.3673
                                 tmpL                                 alpha
                               0.7926                               -0.4671

> summary(2*pnorm(abs(gg$z),lower.tail=FALSE))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0908  0.4795  0.6669  0.6309  0.8364  0.9705 
```

The most frequently used diagnostic, Gelman-Rubin (gelman.diag), requires multiple chains. The full set of diagnostic functions available in coda is:

```
[1] autocorr.diag gelman.diag   geweke.diag   heidel.diag   raftery.diag
```

effectiveSize gives the effective length of the chain for each variable, i.e. the number of samples corrected for autocorrelation:

```
> range(effectiveSize(tm))
```

```
[1] 233.0753 417.1391
```

HPDinterval gives the highest posterior density (credible interval):

```
> detach("package:lme4") ## kluge!!
> head(HPDinterval(tm))
```

```
                            lower       upper
pz                     0.04298075  0.14200879
(Intercept)            2.64801231  5.94767059
FoodTreatmentSatiated -1.19929429 -0.68661971
ArrivalTime           -0.18632185 -0.06141591
SexParentMale         -1.71678380  2.19172558
BroodSize              0.04600739  0.32411652
```

You might prefer inferences based on the quantiles instead:

```
> head(t(apply(tm,2,quantile,c(0.025,0.975))))
```

```
                             2.5%      97.5%
pz                     0.04603193  0.1493516
(Intercept)            2.57678904  5.8787228
FoodTreatmentSatiated -1.19407645 -0.6697261
ArrivalTime           -0.18398412 -0.0580883
SexParentMale         -1.51901806  2.5242395
BroodSize              0.05040258  0.3493775
```

You can also look at density plots or pairwise scatterplots ("splom" in lattice and scapeMCMC, for **S**catter**plo**t **m**atrices), although these are not particularly useful for this large a set of parameters:

```
> plotDens(tm)
> plotSplom(tm,pch=".")
```

The MCMC output in glmmADMB is currently in a very raw form — in particular, the internal names and scales of the parameters are used:

pz zero-inflation parameter (raw)

beta fixed-effect parameter estimates: **note** that these are the versions of the parameters fitted internally, using an orthogonalized version of the original design matrix, not the original coefficients. These can be converted to the original using the `phi` matrix as noted in the "Details" section of `?glmmadmb`

tmpL variance-covariance parameters (log-standard-deviation scale)

tmpL1 correlation/off-diagonal elements of variance-covariance matrices ("off-diagonal elements of the Cholesky factor of the correlation matrix'). (If you need to transform these to correlations, you will need to construct the relevant matrices with 1 on the diagonal and compute the cross-product, CC^T (see `tcrossprod`); if this makes no sense to you, contact the maintainers ...)

log_alpha log of overdispersion/scale parameter

u random effects (unscaled: these can be scaled using the estimated random-effects standard deviations from `VarCorr()`)

If you need to use the MCMC output and can't figure out how, please contact the maintainers and encourage them to work on them some more (!)

3 Other information

The standard set of accessors is available:

`coef` extract (fixed-effect) coefficients

`fixef` a synonym for `coef`, for consistency with `nlme`/`lme4`

`ranef` extract random effect coefficients ("BLUPs" or "conditional modes")

`residuals` extract (Pearson) residuals

`fitted` fitted values

`predict` predicted values (*based only on fixed effects, not on random effects*), possibly with standard errors (*based only on uncertainty of fixed effects*), possibly for new data

`logLik` extract log-likelihood

`AIC` extract AIC

`summary` print summary

`stdEr` extract standard errors of coefficients

`vcov` extract estimated variance-covariance matrix of coefficients

`VarCorr` extract variance-covariance matrices of random effects `confint` extract confidence intervals of fixed-effect coefficients

In case this list is out of date, you can try `methods(class="glmmadmb")` to tell you what methods are currently available.

4 To do/road map

4.1 Vignette

- More examples
- Show how to specify starting values
- fix MCMC! Apply phi, std dev
- General troubleshooting (extra arguments, running outside R)
- basic intro to `R2admb`?
- (appendix?) document details of TPL file – robustness hacks, etc.

4.2 Code

- Speed improvement by identifying special cases?
- Spatial models?
- Additional flexibility:
  - Allow model specification for zero-inflation
  - Allow model specification for shape parameter
  - More complex variance models (see AS-REML/MCMCglmm for interface/syntax ideas)
- Improve `predict` method: allow prediction based on REs
- `simulate` method

References

[1] Benjamin M. Bolker. *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ, 2008.

[2] A. Roulin and L. Bersier. Nestling barn owls beg more intensely in the presence of their mother than in the presence of their father. *Animal Behaviour*, 74:1099–1106, 2007.

[3] Alain F. Zuur, Elena N. Ieno, Neil J. Walker, Anatoly A. Saveliev, and Graham M. Smith. *Mixed Effects Models and Extensions in Ecology with R*. Springer, 1 edition, March 2009.