Model selection using GAMM with MuMIn

Kamil Bartoń

September 12, 2011

1 Extending MuMIn's functionality to support gamm

The two principal functions in MuMIn, model.avg and dredge rely on availability of methods for a several generic function for the class of the given fitted model object. These generic functions include ones defined in package stats (logLik, formula, nobs, and optionally deviance which may simply return NULL), as well as ones defined in MuMIn itself (coeffs, getAllTerms and tTable). In some cases the default methods may work as well.

In case of gamm and gamm4, the returned object has no special class, it is a list with two items: lme or mer, and gam (with some information stripped from it). Therefore no specific methods can be applied.

The solution is to provide a wrapper function for gamm that evaluates the model and adds a class attribute onto it, e.g.:

```
> gamm <- function(...) structure(c(mgcv::gamm(...), list(call = match.call())),
+     class = c("gamm", "list"))
```

similarly for gamm4 (but assign the same class gamm):

```
> gamm4 <- function(...) structure(c(gamm4::gamm4(...), list(call = match.call())),
+     class = c("gamm", "list"))
```

As the wrappers have the same names as the actual functions, use of them is invisible for the user, and they mask the original functions on the level of .GlobalEnv.

In addition, these wrappers add a call element, containing the original call to the wrapper function. It is not necessary, but makes things easier later on for dredge.

Once we have an object of class gamm, it is possible to provide methods for it. First let us define the generic methods from stats.

```
> logLik.gamm <- function(object, ...) logLik(object[[if (is.null(object$lme)) "mer" else "lme"]],
+     ...)
> formula.gamm <- function(x, ...) formula(x$gam, ...)
> nobs.gamm <- function(object, ...) nobs(object$gam, ...)
```

It should be noted here that the issue of what the log-likelihood for GAMM should be is not entirely clear. The documentation for gamm states that the log-likelihood of lme is not the one of the fitted GAMM. However, comparing alternative models presents some evidence that it may be still appropriate for gamm. Namely both the log-likelihood of fitted lme, and one of the lme part of gamm (including only linear terms to make the comparison adequate) have identical values.

```
> dat <- gamSim(6, n = 100, scale = 0.2, dist = "gaussian")
```

```
4 term additive + random effectGu & Wahba 4 term additive model
```

```
> fm1 <- gamm(y ~ x0 + x1 + x2 + x3, data = dat, random = list(fac = ~1),
+     method = "ML")
> fm2 <- lme(y ~ x0 + x1 + x2 + x3, data = dat, random = list(fac = ~1),
+     method = "ML")
> logLik(fm1$lme)
```

```
'log Lik.' -214.5197 (df=7)
```

```
> logLik(fm2)
```

```
'log Lik.' -214.5197 (df=7)
```

Likewise is in the generalised case of `gamm4` and `lmer`:

```
> dat <- gamSim(6, n = 100, scale = 0.2, dist = "poisson")
```

```
4 term additive + random effectGu & Wahba 4 term additive model
```

```
> fmg1 <- gamm4(y ~ x0 + x1 + x2 + x3, family = poisson, data = dat,
+     random = ~(1 | fac))
> fmg2 <- lmer(y ~ x0 + x1 + x2 + x3 + (1 | fac), family = poisson,
+     data = dat)
> logLik(fmg1$mer)
```

```
'log Lik.' -460.5087 (df=6)
```

```
> logLik(fmg2)
```

```
'log Lik.' -460.5087 (df=6)
```

Similarly, comparison of `gamm4` with a smooth term, with fixed two degrees of freedom gives log-likelihood which is very close to that of `lmer` that includes a linear and quadratic term.

```
> fmgs1 <- gamm4(y ~ x0 + s(x1, k = 3, fx = TRUE) + x2 + x3, family = poisson,
+     data = dat, random = ~(1 | fac))
> fmgs2 <- lmer(y ~ x0 + x1 + I(x1^2) + x2 + x3 + (1 | fac), family = poisson,
+     data = dat)
> logLik(fmgs1$mer)
```

```
'log Lik.' -459.4854 (df=7)
```

```
> logLik(fmgs2)
```

```
'log Lik.' -460.3622 (df=7)
```

Normally, the object returned by `gam` inherits also from glm, so the `nobs` method for `glm` is called, but in case of `gamm` the `gam` element has only class `gam`, so we need to define method directly (it just calls `nobs.glm`):

```
> nobs.gam <- function(object, ...) stats:::nobs.glm(object, ...)
```

Methods for generic functions defined in `MuMIn`:

```
> coeffs.gamm <- function(model) coef(model$gam)
> getAllTerms.gamm <- function(x, ...) getAllTerms(x$gam)
> tTable.gamm <- function(model, ...) tTable(model$gam)
```

(the name `tTable` is somewhat misleading, as the `data.frame` returned does not need to contain t-values, two columns are obligatory: 'Estimate' and 'Std. Error')

2 Model selection

Now we have all the prerequisites to proceed with the model selection:

```
> set.seed(0)
> dat <- gamSim(6, n = 100, scale = 0.5, dist = "normal")

4 term additive + random effectGu & Wahba 4 term additive model

> fmgs2 <- gamm(y ~ s(x0) + s(x3) + f0, family = gaussian, data = dat,
+     random = list(fac = ~1))
```

This model fits quite poor. This is deliberate, to justify the model averaging.

```
> head(dd2 <- dredge(fmgs2))

Global model: gamm(y ~ s(x0) + s(x3) + f0, family = gaussian, data = dat, random = list(fac = ~1))
---
Model selection table
  (Int) f0     s(x0) s(x3) k AICc  delta weight
1 16.28                    3 558.1 0.000 0.577
2 15.81 0.3445             4 560.0 1.896 0.224
5 16.28              +     5 562.1 4.002 0.078
3 16.28        +           5 562.4 4.311 0.067
6 15.77 0.3785       +     6 564.0 5.935 0.030
4 15.83 0.3310 +           6 564.4 6.324 0.024
```

(Note that we get quite different results using `gamm4`)

```
> summary(model.avg(dd2, subset = cumsum(weight) <= 0.95))
```

```
Call:  model.avg(object = dd2, subset = cumsum(weight) <= 0.95)


Model summary:
  Deviance   AICc Delta Weight
           558.08  0.00   0.61
1          559.98  1.90   0.24
3          562.08  4.00   0.08
2          562.39  4.31   0.07

Variables:
    1     2     3
   f0 s(x0) s(x3)

Model-averaged coefficients:
            Coefficient         SE z value Pr(>|z|)
(Intercept)   1.617e+01  1.676e+00   9.647   <2e-16 ***
f0            8.148e-02  3.539e-01   0.230    0.818
s(x0).1       1.968e-10  2.044e-05   0.000    1.000
s(x0).2      -3.431e-10  3.203e-05   0.000    1.000
s(x0).3       4.556e-11  7.431e-06   0.000    1.000
s(x0).4      -2.157e-10  1.937e-05   0.000    1.000
s(x0).5       1.860e-11  5.775e-06   0.000    1.000
s(x0).6      -2.047e-10  1.765e-05   0.000    1.000
s(x0).7      -8.549e-11  7.333e-06   0.000    1.000
s(x0).8       8.673e-10  6.020e-05   0.000    1.000
s(x0).9      -7.150e-03  1.006e-01   0.071    0.943
s(x3).1       1.543e-10  2.566e-05   0.000    1.000
s(x3).2      -1.090e-10  3.578e-05   0.000    1.000
s(x3).3       1.660e-11  8.408e-06   0.000    1.000
s(x3).4      -9.882e-11  2.090e-05   0.000    1.000
s(x3).5      -2.364e-11  5.310e-06   0.000    1.000
s(x3).6      -9.143e-11  1.975e-05   0.000    1.000
s(x3).7       4.571e-11  1.055e-05   0.000    1.000
s(x3).8       3.603e-10  6.437e-05   0.000    1.000
s(x3).9      -1.847e-02  1.208e-01   0.153    0.878
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Non-present predictors taken to be zero

Relative variable importance:
   f0 s(x3) s(x0)
 0.24  0.08  0.07
```