

Model selection using GAMM with MuMIn

Kamil Bartoń

January 21, 2012

1 Extending MuMIn's functionality to support gamm

This document describes how to implement the interface between the routines performing model selection and averaging in MuMIn and a class of models not supported by default, using `gamm` from package `mgcv` as an example.

The two principal functions in MuMIn, `model.avg` and `dredge` rely on the availability of methods for several generic functions for the class of the given fitted model object. These generic functions include the 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 `coefTable`). In some cases the default methods may work as well.

In the case of `gamm` and `gamm4`, the returned object has no special class, and 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(mgcv::gamm))),  
+   class = c("gamm", "list"))
```

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

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

As these 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 = "normal")

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.' -224.2712 (df=7)

> logLik(fm2)

'log Lik.' -224.2712 (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.' -703.5312 (df=6)

> logLik(fmg2)

'log Lik.' -703.5312 (df=6)
```

A comparison of `gamm4` including a smooth term with fixed two degrees of freedom gives log-likelihood which is very close to that of `lmer` including 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.' -676.0842 (df=7)
```

```
> logLik(fmgs2)
```

```
'log Lik.' -661.7715 (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 the 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, ...)
```

```
> coefTable.gamm <- function(model, ...) coefTable(model$gam, ...)
```

Two columns are obligatory in the `data.frame` returned by `coefTable`: '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 = 5, dist = "normal")
```

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

```
> fmgs2 <- gamm(y ~ 1 + s(x0) + s(x3) + s(x2), family = gaussian,  
+ data = dat, random = list(fac = ~1))
```

This model fits poorly, but this is deliberate, to justify the model averaging.

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

Global model call: `gamm(formula = y ~ 1 + s(x0) + s(x3) + s(x2), random = list(fac = ~1), family = gaussian, data = dat)`

Model selection table

	(Intercept)	s(x0)	s(x2)	s(x3)	df	logLik	AICc	delta	weight
3	16.58		+		5	-325.830	662.3	0.00	0.480
7	16.58		+	+	7	-323.916	663.0	0.75	0.330
4	16.58	+	+		7	-325.016	665.2	2.95	0.110
8	16.58	+	+	+	9	-323.507	667.0	4.72	0.045
1	16.58				3	-331.043	668.3	6.04	0.023
5	16.58			+	5	-329.624	669.9	7.59	0.011

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

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

Call:

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

Component models:

	df	logLik	AICc	Delta	Weight
2	5	-325.83	662.30	0.00	0.52
23	7	-323.92	663.05	0.75	0.36
12	7	-325.02	665.25	2.95	0.12

Term codes:

s(x0)	s(x2)	s(x3)
1	2	3

Model-averaged coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	1.658e+01	1.737e+00	9.545	<2e-16 ***
s(x2).1	-4.393e+00	2.983e+00	1.472	0.1409
s(x2).2	-1.208e+01	7.979e+00	1.513	0.1302
s(x2).3	-1.157e+00	2.126e+00	0.544	0.5863
s(x2).4	-2.318e+00	5.211e+00	0.445	0.6564
s(x2).5	-1.129e+00	1.606e+00	0.703	0.4820
s(x2).6	-3.211e+00	4.609e+00	0.697	0.4860
s(x2).7	-1.581e+00	1.707e+00	0.926	0.3544
s(x2).8	1.426e+01	1.169e+01	1.219	0.2227
s(x2).9	3.243e+00	4.770e+00	0.680	0.4965
s(x3).1	-4.985e-09	1.669e-04	0.000	1.0000
s(x3).2	-6.610e-09	2.326e-04	0.000	1.0000
s(x3).3	1.033e-09	5.467e-05	0.000	1.0000
s(x3).4	-8.233e-09	1.359e-04	0.000	1.0000
s(x3).5	-1.724e-09	3.452e-05	0.000	1.0000
s(x3).6	-8.022e-09	1.284e-04	0.000	1.0000
s(x3).7	4.433e-09	6.862e-05	0.000	0.9999
s(x3).8	4.255e-08	4.185e-04	0.000	0.9999
s(x3).9	-1.149e+00	5.834e-01	1.969	0.0489 *
s(x0).1	1.278e-08	1.489e-04	0.000	0.9999
s(x0).2	-1.051e-08	2.333e-04	0.000	1.0000
s(x0).3	1.635e-09	5.412e-05	0.000	1.0000
s(x0).4	-7.340e-09	1.410e-04	0.000	1.0000
s(x0).5	1.691e-09	4.206e-05	0.000	1.0000
s(x0).6	-7.870e-09	1.285e-04	0.000	1.0000
s(x0).7	-3.062e-09	5.340e-05	0.000	1.0000
s(x0).8	3.200e-08	4.384e-04	0.000	0.9999
s(x0).9	7.461e-01	5.836e-01	1.278	0.2011

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Full model-averaged coefficients (with shrinkage):

(Intercept)	s(x2).1	s(x2).2	s(x2).3	s(x2).4	s(x2).5
1.6583e+01	-4.3927e+00	-1.2075e+01	-1.1568e+00	-2.3180e+00	-1.1290e+00
s(x2).6	s(x2).7	s(x2).8	s(x2).9	s(x3).1	s(x3).2
-3.2112e+00	-1.5809e+00	1.4258e+01	3.2433e+00	-1.7880e-09	-2.3708e-09
s(x3).3	s(x3).4	s(x3).5	s(x3).6	s(x3).7	s(x3).8
3.7056e-10	-2.9532e-09	-6.1827e-10	-2.8775e-09	1.5900e-09	1.5263e-08
s(x3).9	s(x0).1	s(x0).2	s(x0).3	s(x0).4	s(x0).5
-4.1208e-01	1.5258e-09	-1.2551e-09	1.9519e-10	-8.7648e-10	2.0188e-10
s(x0).6	s(x0).7	s(x0).8	s(x0).9		
-9.3974e-10	-3.6567e-10	3.8213e-09	8.9094e-02		

Relative variable importance:

(Intercept)	s(x0)	s(x2)	s(x3)
1.00	0.12	1.00	0.36