

Package ‘gamm4’

April 3, 2020

Version 0.2-6

Author Simon Wood, Fabian Scheipl

Maintainer Simon Wood <simon.wood@r-project.org>

Title Generalized Additive Mixed Models using 'mgcv' and 'lme4'

Description Estimate generalized additive mixed models via a version of function `gamm()` from 'mgcv', using 'lme4' for estimation.

Depends R (>= 2.9.0), methods, Matrix, lme4 (>= 1.0), mgcv (>= 1.7-23)

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2020-04-03 19:30:02 UTC

R topics documented:

gamm4	1
Index	10

gamm4	<i>Generalized Additive Mixed Models using lme4 and mgcv</i>
-------	--

Description

Fits the specified generalized additive mixed model (GAMM) to data, by making use of the modular fitting functions provided by `lme4` (new version). For earlier `lme4` versions modelling fitting is via a call to `lmer` in the normal errors identity link case, or by a call to `glmer` otherwise (see [lmer](#)). Smoothness selection is by REML in the Gaussian additive case and (Laplace approximate) ML otherwise.

`gamm4` is based on [gamm](#) from package `mgcv`, but uses `lme4` rather than `nlme` as the underlying fitting engine via a trick due to Fabian Scheipl. `gamm4` is more robust numerically than [gamm](#), and by avoiding PQL gives better performance for binary and low mean count data. Its main disadvantage

is that it can not handle most multi-penalty smooths (i.e. not `te` type tensor products or adaptive smooths) and there is no facility for `nlme` style correlation structures. Tensor product smoothing is available via `t2` terms (Wood, Scheipl and Faraway, 2013).

For fitting generalized additive models without random effects, `gamm4` is much slower than `gam` and has slightly worse MSE performance than `gam` with REML smoothness selection. For fitting GAMMs with modest numbers of i.i.d. random coefficients then `gamm4` is slower than `gam` (or `bam` for large data sets). `gamm4` is most useful when the random effects are not i.i.d., or when there are large numbers of random coefficients (more than several hundred), each applying to only a small proportion of the response data.

To use this function effectively it helps to be quite familiar with the use of `gam` and `lmer`.

Usage

```
gamm4(formula, random=NULL, family=gaussian(), data=list(), weights=NULL,
       subset=NULL, na.action, knots=NULL, drop.unused.levels=TRUE,
       REML=TRUE, control=NULL, start=NULL, verbose=0L, ...)
```

Arguments

<code>formula</code>	A GAM formula (see also <code>formula.gam</code> and <code>gam.models</code>). This is like the formula for a <code>glm</code> except that smooth terms (<code>s</code> and <code>t2</code> but not <code>te</code>) can be added to the right hand side of the formula. Note that <code>ids</code> for smooths and fixed smoothing parameters are not supported.
<code>random</code>	An optional formula specifying the random effects structure in <code>lmer</code> style. See example below.
<code>family</code>	A family as used in a call to <code>glm</code> or <code>gam</code> .
<code>data</code>	A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>gamm4</code> is called.
<code>weights</code>	a vector of prior weights on the observations. <code>NULL</code> is equivalent to a vector of 1s. Used, in particular, to supply the number-of-trials for binomial data, when the response is proportion of successes.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process.
<code>na.action</code>	a function which indicates what should happen when the data contain 'NA's. The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is unset. The "factory-fresh" default is 'na.omit'.
<code>knots</code>	this is an optional list containing user specified knot values to be used for basis construction. Different terms can use different numbers of knots, unless they share a covariate.
<code>drop.unused.levels</code>	by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing.
<code>REML</code>	passed on to <code>lmer</code> fitting routines (but not <code>glmer</code> fitting routines) to control whether REML or ML is used.

control	lmerControl or glmerControl list as appropriate (NULL means defaults are used).
start	starting value list as used by lmer or glmer .
verbose	passed on to fitting lme4 fitting routines.
...	further arguments for passing on to model setup routines.

Details

A generalized additive mixed model is a generalized linear mixed model in which the linear predictor depends linearly on unknown smooth functions of some of the covariates ('smooths' for short). `gamm4` follows the approach taken by package `mgcv` and represents the smooths using penalized regression spline type smoothers, of moderate rank. For estimation purposes the penalized component of each smooth is treated as a random effect term, while the unpenalized component is treated as fixed. The wiggleness penalty matrix for the smooth is in effect the precision matrix when the smooth is treated as a random effect. Estimating the degree of smoothness of the term amounts to estimating the variance parameter for the term.

`gamm4` uses the same reparameterization trick employed by `gamm` to allow any single quadratic penalty smoother to be used (see Wood, 2004, or 2006 for details). Given the reparameterization then the modular fitting approach employed in `lmer` can be used to fit a GAMM. Estimation is by Maximum Likelihood in the generalized case, and REML in the gaussian additive model case. `gamm4` allows the random effects specifiable with `lmer` to be combined with any number of any of the (single penalty) smooth terms available in `gam` from package `mgcv` as well as `t2` tensor product smooths. Note that the model comparison on the basis of the (Laplace approximate) log likelihood is possible with GAMMs fitted by `gamm4`.

As in `gamm` the smooth estimates are assumed to be of interest, and a covariance matrix is returned which enables Bayesian credible intervals for the smooths to be constructed, which treat all the terms in random as random.

For details on how to condition smooths on factors, set up varying coefficient models, do signal regression or set up terms involving linear functionals of smooths, see [gam.models](#), but note that the type tensor product and adaptive smooths are not available with `gamm4`.

Value

Returns a list with two items:

gam	an object of class <code>gam</code> . At present this contains enough information to use <code>predict</code> , <code>plot</code> , <code>summary</code> and <code>print</code> methods and <code>vis.gam</code> , from package <code>mgcv</code> but not to use e.g. the <code>anova</code> method function to compare models.
mer	the fitted model object returned by <code>lmer</code> or <code>glmer</code> . Extra random and fixed effect terms will appear relating to the estimation of the smooth terms. Note that unlike <code>lme</code> objects returned by <code>gamm</code> , everything in this object always relates to the fitted model itself, and never to a PQL working approximation: hence the usual methods of model comparison are entirely legitimate.

WARNINGS

If you don't need random effects in addition to the smooths, then `gam` is substantially faster, gives fewer convergence warnings, and slightly better MSE performance (based on simulations).

Models must contain at least one random effect: either a smooth with non-zero smoothing parameter, or a random effect specified in argument `random`.

Note that the `gam` object part of the returned object is not complete in the sense of having all the elements defined in `gamObject` and does not inherit from `glm`: hence e.g. multi-model anova calls will not work.

Linked smoothing parameters, adaptive smoothing and `te` terms are not supported.

This routine is obviously less well tested than `gamm`.

Author(s)

Simon N. Wood <simon.wood@r-project.org>

References

Bates D., M. Maechler, B. Bolker & S. Walker (2013). lme4: Linear mixed-effects models using Eigen and S4. <https://cran.r-project.org/package=lme4>

Wood S.N., Scheipl, F. and Faraway, J.J. (2013/2011 online) Straightforward intermediate rank tensor product smoothing in mixed models. *Statistics and Computing* 23(3): 341-360

Wood, S.N. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. *Journal of the American Statistical Association*. 99:673-686

Wood S.N. (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC Press.

For more GAMM references see `gamm`

<http://www.maths.bris.ac.uk/~sw15190/>

See Also

`gam`, `gamm`, `gam.models`, `lmer`, `predict.gam`, `plot.gam`, `summary.gam`, `s`, `vis.gam`

Examples

```
## NOTE: most examples are flagged as 'do not run' simply to
## save time in package checking on CRAN.

#####
## A simple additive mixed model...
#####
library(gam4)

set.seed(0)
dat <- gamSim(1,n=400,scale=2) ## simulate 4 term additive truth
## Now add 20 level random effect `fac'...
dat$fac <- fac <- as.factor(sample(1:20,400,replace=TRUE))
dat$y <- dat$y + model.matrix(~fac-1)%*%rnorm(20)*.5

br <- gamm4(y~s(x0)+x1+s(x2),data=dat,random=~(1|fac))
plot(br$gam,pages=1)
```

```

summary(br$gam) ## summary of gam
summary(br$mer) ## underlying mixed model
anova(br$gam)

## compare gam fit of the same
bg <- gam(y~s(x0)+x1+s(x2)+s(fac,bs="re"),
          data=dat,method="REML")
plot(bg,pages=1)
gam.vcomp(bg)

#####
## Poisson example GAMM...
#####
## simulate data...
x <- runif(100)
fac <- sample(1:20,100,replace=TRUE)
eta <- x^2*3 + fac/20; fac <- as.factor(fac)
y <- rpois(100,exp(eta))

## fit model and examine it...
bp <- gamm4(y~s(x),family=poisson,random=~(1|fac))
plot(bp$gam)
bp$mer

## Not run:
#####
## Add a factor to the linear predictor, to be modelled as random
## and make response Poisson. Again compare `gamm` and `gamm4`
#####
set.seed(6)
dat <- gamSim(1,n=400,scale=2) ## simulate 4 term additive truth
## add random effect...
g <- as.factor(sample(1:20,400,replace=TRUE))
dat$f <- dat$f + model.matrix(~ g-1)%*%rnorm(20)*2
dat$y <- rpois(400,exp(dat$f/7+1))

b2<-gamm(y~s(x0)+s(x1)+s(x2)+s(x3),family=poisson,
         data=dat,random=list(g=~1))
plot(b2$gam,pages=1)

b2r<-gamm4(y~s(x0)+s(x1)+s(x2)+s(x3),family=poisson,
           data=dat,random = ~ (1|g))

plot(b2r$gam,pages=1)

rm(dat)
vis.gam(b2r$gam,theta=35)

#####
# Multivariate varying coefficient
# With crossed and nested random
# effects.

```

```
#####

## Start by simulating data...

f0 <- function(x, z, sx = 0.3, sz = 0.4) {
  (pi*sx * sz) * (1.2 * exp(-(x - 0.2)^2/sx^2 - (z -
    0.3)^2/sz^2) + 0.8 * exp(-(x - 0.7)^2/sx^2 -
    (z - 0.8)^2/sz^2))
}
f1 <- function(x2) 2 * sin(pi * x2)
f2 <- function(x2) exp(2 * x2) - 3.75887
f3 <- function(x2) 0.2 * x2^11 * (10 * (1 - x2))^6 + 10 * (10 * x2)^3 *
  (1 - x2)^10

n <- 1000

## first set up a continuous-within-group effect...

g <- factor(sample(1:50,n,replace=TRUE)) ## grouping factor
x <- runif(n) ## continuous covariate
X <- model.matrix(~g-1)
mu <- X%*%rnorm(50)*.5 + (x*X)%*%rnorm(50)

## now add nested factors...
a <- factor(rep(1:20,rep(50,20)))
b <- factor(rep(rep(1:25,rep(2,25)),rep(20,50)))
Xa <- model.matrix(~a-1)
Xb <- model.matrix(~a/b-a-1)
mu <- mu + Xa%*%rnorm(20) + Xb%*%rnorm(500)*.5

## finally simulate the smooth terms
v <- runif(n);w <- runif(n);z <- runif(n)
r <- runif(n)
mu <- mu + f0(v,w)*z*10 + f3(r)

y <- mu + rnorm(n)*2 ## response data

## First compare gamm and gamm4 on a reduced model

br <- gamm4(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),random = ~ (1|a/b))

ba <- gamm(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),random = list(a=~1,b=~1),method="REML")

par(mfrow=c(2,2))
plot(br$gam)

plot(ba$gam)

## now fit the full model

br <- gamm4(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),random = ~ (x+0|g) + (1|g) + (1|a/b))
```

```

br$mer
br$gam
plot(br$gam)

## try a Poisson example, based on the same linear predictor...

lp <- mu/5
y <- rpois(exp(lp),exp(lp)) ## simulated response

## again compare gamm and gamm4 on reduced model

br <- gamm4(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),family=poisson,random = ~ (1|a/b))

ba <- gamm(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),family=poisson,random = list(a=~1,b=~1))

par(mfrow=c(2,2))
plot(br$gam)
plot(ba$gam)

## and now fit full version (very slow)...

br <- gamm4(y ~ s(v,w,by=z) + s(r,k=20,bs="cr"),family=poisson,random = ~ (x|g) + (1|a/b))
br$mer
br$gam
plot(br$gam)

#####
# Different smooths of x2 depending
# on factor `fac'...
#####
dat <- gamSim(4)

br <- gamm4(y ~ fac+s(x2,by=fac)+s(x0),data=dat)
plot(br$gam,pages=1)
summary(br$gam)

#####
# Timing comparison with `gam'... #
#####

dat <- gamSim(1,n=600,dist="binary",scale=.33)

system.time(lr.fit0 <- gam(y~s(x0)+s(x1)+s(x2),
  family=binomial,data=dat,method="ML"))

system.time(lr.fit <- gamm4(y~s(x0)+s(x1)+s(x2),
  family=binomial,data=dat))

lr.fit0;lr.fit$gam
cor(fitted(lr.fit0),fitted(lr.fit$gam))

```

```

## plot model components with truth overlaid in red
op <- par(mfrow=c(2,2))
fn <- c("f0", "f1", "f2", "f3"); xn <- c("x0", "x1", "x2", "x3")
for (k in 1:3) {
  plot(lr.fit$gam, select=k)
  ff <- dat[[fn[k]]]; xx <- dat[[xn[k]]]
  ind <- sort.int(xx, index.return=TRUE)$ix
  lines(xx[ind], (ff-mean(ff))[ind]*.33, col=2)
}
par(op)

## End(Not run)

#####
## A "signal" regression example, in
## which a univariate response depends
## on functional predictors.
#####

## simulate data first...

rf <- function(x=seq(0,1,length=100)) {
  ## generates random functions...
  m <- ceiling(runif(1)*5) ## number of components
  f <- x*0;
  mu <- runif(m, min(x), max(x)); sig <- (runif(m)+.5)*(max(x)-min(x))/10
  for (i in 1:m) f <- f+ dnorm(x, mu[i], sig[i])
  f
}

x <- seq(0,1,length=100) ## evaluation points

## example functional predictors...
par(mfrow=c(3,3)); for (i in 1:9) plot(x, rf(x), type="l", xlab="x")

## simulate 200 functions and store in rows of L...
L <- matrix(NA, 200, 100)
for (i in 1:200) L[i,] <- rf() ## simulate the functional predictors

f2 <- function(x) { ## the coefficient function
  (0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10)/10
}

f <- f2(x) ## the true coefficient function

y <- L*f + rnorm(200)*20 ## simulated response data

## Now fit the model  $E(y) = L*f(x)$  where  $f$  is a smooth function.
## The summation convention is used to evaluate smooth at each value
## in matrix  $X$  to get matrix  $F$ , say. Then  $\text{rowSum}(L*F)$  gives  $E(y)$ .

## create matrix of eval points for each function. Note that
## `smoothCon` is smart and will recognize the duplication...

```



```
X <- matrix(x,200,100,byrow=TRUE)

## compare `gam` and `gamm4` this time

b <- gam(y~s(X,by=L,k=20),method="REML")
br <- gamm4(y~s(X,by=L,k=20))
par(mfrow=c(2,1))
plot(b,shade=TRUE);lines(x,f,col=2)
plot(br$gam,shade=TRUE);lines(x,f,col=2)
```

Index

*Topic **models**

gamm4, 1

*Topic **regression**

gamm4, 1

*Topic **smooth**

gamm4, 1

bam, 2

formula.gam, 2

gam, 2–4

gam.models, 2–4

gamm, 1, 3, 4

gamm4, 1

gamObject, 4

glm, 2

glmer, 2, 3

glmerControl, 3

lmer, 1–4

lmerControl, 3

plot.gam, 4

predict.gam, 4

s, 2, 4

summary.gam, 4

t2, 2, 3

te, 2

vis.gam, 4