Package 'GLMMadaptive'

October 13, 2020

Title Generalized Linear Mixed Models using Adaptive Gaussian Quadrature

Version 0.7-15

Date 2020-10-13

Maintainer Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

BugReports https://github.com/drizopoulos/GLMMadaptive/issues

Description Fits generalized linear mixed models for a single grouping factor under maximum likelihood approximating the integrals over the random effects with an adaptive Gaussian quadrature rule; Jose C. Pinheiro and Douglas M. Bates (1995) <doi:10.1080/10618600.1995.10474663>.

Imports MASS, nlme, parallel, matrixStats

Suggests lattice, knitr, rmarkdown, pkgdown, multcomp, emmeans, estimability, effects, DHARMa, optimParallel

Encoding UTF-8

LazyLoad yes

LazyData yes

License GPL (>= 3)

URL https://drizopoulos.github.io/GLMMadaptive/, https://github.com/drizopoulos/GLMMadaptive

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Dimitris Rizopoulos [aut, cre] (<https://orcid.org/0000-0001-9397-0900>)

Repository CRAN

Date/Publication 2020-10-13 15:20:09 UTC

R topics documented:

`Continuation Ratio Set-Up`
Functions to Set-Up Data for a Continuation Ratio Mixed Model

---

Description

Data set-up and calculation of marginal probabilities from a continuation ratio model

Usage

```
cr_setup(y, direction = c("forward", "backward"))

cr_marg_probs(eta, direction = c("forward", "backward"))
```

Arguments

`y`	a numeric vector denoting the ordinal response variable.
`direction`	character string specifying the direction of the continuation ratio model; `"forward"` corresponds to a discrete hazard function.
`eta`	a numeric matrix of the linear predictor, with columns corresponding to the different levels of the ordinal response.

Note

Function `cr_setup()` is based on the `cr.setup()` function from package **rms**.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

Frank Harrell

Examples

```
n <- 300 # number of subjects
K <- 8 # number of measurements per subject
t_max <- 15 # maximum follow-up time

# we constuct a data frame with the design:
# everyone has a baseline measurment, and then measurements at random follow-up times
DF <- data.frame(id = rep(seq_len(n), each = K),
                 time = c(replicate(n, c(0, sort(runif(K - 1, 0, t_max))))),
                 sex = rep(gl(2, n/2, labels = c("male", "female")), each = K))

# design matrices for the fixed and random effects
X <- model.matrix(~ sex * time, data = DF)[, -1]
Z <- model.matrix(~ 1, data = DF)

thrs <- c(-1.5, 0, 0.9) # thresholds for the different ordinal categories
betas <- c(-0.25, 0.24, -0.05) # fixed effects coefficients
D11 <- 0.48 # variance of random intercepts
D22 <- 0.1 # variance of random slopes

# we simulate random effects
b <- cbind(rnorm(n, sd = sqrt(D11)), rnorm(n, sd = sqrt(D22)))[, 1, drop = FALSE]
# linear predictor
eta_y <- drop(X %*% betas + rowSums(Z * b[DF$id, , drop = FALSE]))
# linear predictor for each category
eta_y <- outer(eta_y, thrs, "+")
# marginal probabilities per category
mprobs <- cr_marg_probs(eta_y)
# we simulate ordinal longitudinal data
DF$y <- unname(apply(mprobs, 1, sample, x = ncol(mprobs), size = 1, replace = TRUE))

# If you want to simulate from the backward formulation of the CR model, you need to
# change `eta_y <- outer(eta_y, thrs, "+")` to `eta_y <- outer(eta_y, rev(thrs), "+")`,
# and `mprobs <- cr_marg_probs(eta_y)` to `mprobs <- cr_marg_probs(eta_y, "backward")`

################################################

# prepare the data
# If you want to fit the CR model under the backward formulation, you need to change
# `cr_vals <- cr_setup(DF$y)` to `cr_vals <- cr_setup(DF$y, "backward")`
cr_vals <- cr_setup(DF$y)
cr_data <- DF[cr_vals$subs, ]
cr_data$y_new <- cr_vals$y
cr_data$cohort <- cr_vals$cohort

# fit the model
fm <- mixed_model(y_new ~ cohort + sex * time, random = ~ 1 | id,
                  data = cr_data, family = binomial())

summary(fm)
```

effectPlotData *Predicted Values for Effects Plots*

Description

Creates predicted values and their corresponding confidence interval for constructing an effects plot.

Usage

```
effectPlotData(object, newdata, level, ...)

## S3 method for class 'MixMod'
effectPlotData(object, newdata,
   level = 0.95, marginal = FALSE, CR_cohort_varname = NULL,
   direction = NULL, K = 200, seed = 1, sandwich = FALSE,
   ...)
```

Arguments

object	an object inheriting from class "MixMod".
newdata	a data frame base on which predictions will be calculated.
level	a numeric scalar denoting the level of the confidence interval.
marginal	logical; if FALSE predicted values are calculated for the "mean" subject (i.e., the one with random effects values equal to 0). When TRUE marginal predicted values are calculated using function marginal_coefs.
CR_cohort_varname	a character string denoting the name of the cohort variable when a continuation ratio model is fitted.
direction	the direction argument of cr_marg_probs needs to be provided when CR_cohort_varname is not NULL.
K	numeric scalar denoting the number of Monte Carlo samples from the approximate posterior of the parameters; applicable only for zero-inflated models.
seed	numerical scalar giving the seed to be used in the Monte Carlo scheme.
sandwich	logical; if TRUE robust/sandwich standard errors are used in the calculations.
...	additional arguments passed to marginal_coefs, e.g., cores.

Details

The confidence interval is calculated based on a normal approximation.

Value

The data frame newdata with extra columns pred, low and upp.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

See Also

mixed_model, marginal_coefs

Examples

```
# simulate some data
set.seed(123L)
n <- 500
K <- 15
t.max <- 25

betas <- c(-2.13, -0.25, 0.24, -0.05)
D <- matrix(0, 2, 2)
D[1:2, 1:2] <- c(0.48, -0.08, -0.08, 0.18)

times <- c(replicate(n, c(0, sort(runif(K-1, 0, t.max)))))
group <- sample(rep(0:1, each = n/2))
DF <- data.frame(year = times, group = factor(rep(group, each = K)))
X <- model.matrix(~ group * year, data = DF)
Z <- model.matrix(~ year, data = DF)

b <- cbind(rnorm(n, sd = sqrt(D[1, 1])), rnorm(n, sd = sqrt(D[2, 2])))
id <- rep(1:n, each = K)
eta.y <- as.vector(X %*% betas + rowSums(Z * b[id, ]))
DF$y <- rbinom(n * K, 1, plogis(eta.y))
DF$id <- factor(id)

################################################

# Fit a model
fm1 <- mixed_model(fixed = y ~ year * group, random = ~ year | id, data = DF,
                   family = binomial())

# An effects plot for the mean subject (i.e., with random effects equal to 0)
nDF <- with(DF, expand.grid(year = seq(min(year), max(year), length.out = 15),
    group = levels(group)))

plot_data <- effectPlotData(fm1, nDF)

require("lattice")
xyplot(pred + low + upp ~ year | group, data = plot_data,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
       xlab = "Follow-up time", ylab = "log odds")

expit <- function (x) exp(x) / (1 + exp(x))
xyplot(expit(pred) + expit(low) + expit(upp) ~ year | group, data = plot_data,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
```

```
       xlab = "Follow-up time", ylab = "Probabilities")

# we put the two groups in the same panel
my.panel.bands <- function(x, y, upper, lower, fill, col, subscripts, ..., font,
                           fontface) {
    upper <- upper[subscripts]
    lower <- lower[subscripts]
    panel.polygon(c(x, rev(x)), c(upper, rev(lower)), col = fill, border = FALSE, ...)
}

xyplot(expit(pred) ~ year, group = group, data = plot_data, upper = expit(plot_data$upp),
       low = expit(plot_data$low), type = "l", col = c("blue", "red"),
       fill = c("#0000FF80", "#FF000080"),
       panel = function (x, y, ...) {
           panel.superpose(x, y, panel.groups = my.panel.bands, ...)
           panel.xyplot(x, y, lwd = 2,  ...)
}, xlab = "Follow-up time", ylab = "Probabilities")

# An effects plots for the marginal probabilities
plot_data_m <- effectPlotData(fm1, nDF, marginal = TRUE, cores = 1L)

expit <- function (x) exp(x) / (1 + exp(x))
xyplot(expit(pred) + expit(low) + expit(upp) ~ year | group, data = plot_data_m,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
       xlab = "Follow-up time", ylab = "Probabilities")
```

`Extra Family Objects` *Family functions for Student's-t, Beta, Zero-Inflated and Hurdle Poisson and Negative Binomial, Hurdle Log-Normal, and Hurdle Beta Mixed Models*

Description

Specifies the information required to fit a Beta, zero-inflated and hurdle Poisson, zero-inflated and hurdle Negative Binomial, a hurdle normal and a hurdle Beta mixed-effects model, using `mixed_model()`.

Usage

```
students.t(df = stop("'df' must be specified"), link = "identity")
beta.fam()
zi.poisson()
zi.negative.binomial()
hurdle.poisson()
hurdle.negative.binomial()
hurdle.lognormal()
hurdle.beta.fam()
unit.lindley()
```

```
beta.binomial(link = "logit")
Gamma.fam()
```

Arguments

link	name of the link function.
df	the degrees of freedom of the Student's t distribution.

Note

Currently only the log-link is implemented for the Poisson, negative binomial and Gamma models, the logit link for the beta and hurdle beta models and the identity link for the log-normal model.

Examples

```
# simulate some data from a negative binomial model
set.seed(102)
dd <- expand.grid(f1 = factor(1:3), f2 = LETTERS[1:2], g = 1:30, rep = 1:15,
                  KEEP.OUT.ATTRS = FALSE)
mu <- 5*(-4 + with(dd, as.integer(f1) + 4 * as.numeric(f2)))
dd$y <- rnbinom(nrow(dd), mu = mu, size = 0.5)

# Fit a zero-inflated Poisson model, with only fixed effects in the
# zero-inflated part
fm1 <- mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                   family = zi.poisson(), zi_fixed = ~ 1)

summary(fm1)


# We extend the previous model allowing also for a random intercept in the
# zero-inflated part
fm2 <- mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                   family = zi.poisson(), zi_fixed = ~ 1, zi_random = ~ 1 | g)

# We do a likelihood ratio test between the two models
anova(fm1, fm2)

################################################################################
################################################################################

# The same as above but with a negative binomial model
gm1 <- mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                   family = zi.negative.binomial(), zi_fixed = ~ 1)

summary(gm1)

# We do a likelihood ratio test between the Poisson and negative binomial models
anova(fm1, gm1)
```

GLMMadaptive *Generalized Linear Mixed Models using Adaptive Gaussian Quadrature*

Description

This package fits generalized linear mixed models for a single grouping factor under maximum likelihood approximating the integrals over the random effects with an adaptive Gaussian quadrature rule.

Details

Package:	GLMMadaptive
Type:	Package
Version:	0.7-15
Date:	2020-10-13
License:	GPL (>=3)

This package fits mixed effects models for grouped / repeated measurements data for which the integral over the random effects in the definition of the marginal likelihood cannot be solved analytically. The package approximates these integrals using the adaptive Gauss-Hermite quadrature rule.

Multiple random effects terms can be included for the grouping factor (e.g., random intercepts, random linear slopes, random quadratic slopes), but currently only a single grouping factor is allowed.

The package also offers several utility functions that can extract useful information from fitted mixed effects models. The most important of those are included in the **See also** Section below.

Author(s)

Dimitris Rizopoulos

Maintainer: Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

References

Fitzmaurice, G., Laird, N. and Ware J. (2011). *Applied Longitudinal Analysis*, 2nd Ed. New York: John Wiley & Sons.

Molenberghs, G. and Verbeke, G. (2005). *Models for Discrete Longitudinal Data*. New York: Springer-Verlag.

See Also

`mixed_model`, `methods.MixMod`, `effectPlotData`, `marginal_coefs`

marginal_coefs *Marginal Coefficients from Generalized Linear Mixed Models*

Description

Calculates marginal coefficients and their standard errors from fitted generalized linear mixed models.

Usage

```
marginal_coefs(object, ...)

## S3 method for class 'MixMod'
marginal_coefs(object, std_errors = FALSE,
   link_fun = NULL, M = 3000, K = 100, seed = 1,
   cores = max(parallel::detectCores() - 1, 1),
   sandwich = FALSE, ...)
```

Arguments

object	an object inheriting from class "MixMod".
std_errors	logical indicating whether standard errors are to be computed.
link_fun	a function transforming the mean of the repeated measurements outcome to the linear predictor scale. Typically, this derived from the family argument of mixed_model.
M	numeric scalar denoting the number of Monte Carlo samples.
K	numeric scalar denoting the number of samples from the sampling distribution of the maximum likelihood estimates.
seed	integer denoting the seed for the random number generation.
cores	integer giving the number of cores to use; applicable only when std_errors = TRUE.
sandwich	logical; if TRUE robust/sandwich standard errors are used in the calculations.
...	extra arguments; currently none is used.

Details

It uses the approach of Hedeker et al. (2017) to calculate marginal coefficients from mixed models with nonlinear link functions. The marginal probabilities are calculated using Monte Carlo integration over the random effects with M samples, by sampling from the estimated prior distribution, i.e., a multivariate normal distribution with mean 0 and covariance matrix $\hat{D}$, where $\hat{D}$ denotes the estimated covariance matrix of the random effects.

To calculate the standard errors, the Monte Carlo integration procedure is repeated K times, where each time instead of the maximum likelihood estimates of the fixed effects and the covariance matrix of the random effects, a realization is used from the sampling distribution of the maximum likelihood estimates. To speed-up this process, package **parallel** is used.

Value

A list of class "m_coefs" with components betas the marginal coefficients, and when std_errors = TRUE, the extra components var_betas the estimated covariance matrix of the marginal coefficients, and coef_table a numeric matrix with the estimated marginal coefficients, their standard errors and corresponding p-values using the normal approximation.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

References

Hedeker, D., du Toit, S. H., Demirtas, H. and Gibbons, R. D. (2018), A note on marginalization of regression parameters from mixed models of binary outcomes. *Biometrics* **74**, 354–361. doi:10.1111/biom.12707

See Also

mixed_model

Examples

```
# simulate some data
set.seed(123L)
n <- 500
K <- 15
t.max <- 25

betas <- c(-2.13, -0.25, 0.24, -0.05)
D <- matrix(0, 2, 2)
D[1:2, 1:2] <- c(0.48, -0.08, -0.08, 0.18)

times <- c(replicate(n, c(0, sort(runif(K-1, 0, t.max)))))
group <- sample(rep(0:1, each = n/2))
DF <- data.frame(year = times, group = factor(rep(group, each = K)))
X <- model.matrix(~ group * year, data = DF)
Z <- model.matrix(~ year, data = DF)

b <- cbind(rnorm(n, sd = sqrt(D[1, 1])), rnorm(n, sd = sqrt(D[2, 2])))
id <- rep(1:n, each = K)
eta.y <- as.vector(X %*% betas + rowSums(Z * b[id, ]))
DF$y <- rbinom(n * K, 1, plogis(eta.y))
DF$id <- factor(id)

################################################

fm1 <- mixed_model(fixed = y ~ year * group, random = ~ 1 | id, data = DF,
                   family = binomial())

fixef(fm1)
```

```
marginal_coefs(fm1)
marginal_coefs(fm1, std_errors = TRUE, cores = 1L)
```

`mixed_model` *Generalized Linear Mixed Effects Models*

Description

Fits generalized linear mixed effects models under maximum likelihood using adaptive Gaussian quadrature.

Usage

```
mixed_model(fixed, random, data, family, weights = NULL,
  na.action = na.exclude, zi_fixed = NULL, zi_random = NULL,
  penalized = FALSE, n_phis = NULL, initial_values = NULL,
  control = list(), ...)
```

Arguments

`fixed`	a formula for the fixed-effects part of the model, including the outcome.
`random`	a formula for the random-effects part of the model. This should only contain the right-hand side part, e.g., `~ time | id`, where `time` is a variable, and `id` the grouping factor. When the symbol `||` is used in the definition of this argument (instead of `|`), then the covariance matrix of the random effects is assumed to be diagonal.
`data`	a data.frame containing the variables required in `fixed` and `random`.
`family`	a `family` object specifying the type of the repeatedly measured response variable, e.g., `binomial()` or `poisson()`. The function also allows for user-defined family objects, but with specific extra components; see the example is `negative.binomial` for more details. Contrary to the standard practice in model fitting R functions with a `family` argument (e.g., `glm`) in which the default family is `gaussian()`, in `mixed_model()` no default is provided. If the users wish to fit a mixed model for a Gaussian outcome, this could be done with function `lme()` from the **nlme** package or function `lmer()` from the **lme4** package.
`weights`	a numeric vector of weights. These are simple multipliers on the log-likelihood contributions of each group/cluster, i.e., we presume that there are multiple replicates of each group/cluster denoted by the weights. The length of 'weights' need to be equal to the number of independent groups/clusters in the data.
`na.action`	what to do with missing values in `data`.
`zi_fixed, zi_random`	formulas for the fixed and random effects of the zero inflated part.

`penalized` logical or a list. If logical and equal to `FALSE`, then no penalty is used. If logical and equal to `TRUE`, for the fixed effects a Student's-t penalty/prior with mean 0, scale equal to 1 and 3 degrees of freedom is used. If a list, then it is expected to have the components `pen_mu`, `pen_sigma` and `pen_df`, denoting the mean, scale and degrees of freedom of the Student's-t penalty/prior for the fixed effects.

`n_phis` a numeric scalar; in case the family corresponds to a distribution that has extra (dispersion/shape) parameters, you need to specify how many extra parameters are needed.

`initial_values` a list of initial values. This can have up to three components, namely,

betas a numeric vector of fixed effects. This can also be `family` object. In this case initial values for the fixed effects will be calculated by using `glm` to the data ignoring the correlations in the repeated measurements. For example, for a negative binomial response outcome, we could set `betas = poisson()`.

D a numeric matrix denoting the covariance matrix of the random effects.

phis a numeric vector for the extra (dispersion/shape) parameters.

`control` a list with the following components:

iter_EM numeric scalar denoting the number of EM iterations; default is 30.

iter_qN_outer numeric scalar denoting the number of outer iterations during the quasi-Newton phase; default is 15. In each outer iteration the locations of the quadrature points are updated.

iter_qN numeric scalar denoting the starting number of iterations for the quasi-Newton; default is 10.

iter_qN_incr numeric scalar denoting the increment in `iter_qN` for each outer iteration; default is 10.

optimizer character string denoting the optimizer to be used; available options are `"optim"` (default), `"nlminb"` and `"optimParallel"`, the last option implemented in the **optimParallel** package.

optim_method character string denoting the type of `optim` algorithm to be used when `optimizer = "optim"`; default is the BFGS algorithm.

parscale_betas the control argument `parscale` of `optim` for the fixed-effects; default is 0.1.

parscale_D the control argument `parscale` of `optim` for the unique element of the covariance matrix of the random effects; default is 0.01.

parscale_phis the control argument `parscale` of `optim` for the extra (dispersion/shape) parameters; default is 0.01.

tol1, tol2, tol3 numeric scalars controlling tolerances for declaring convergence; `tol1` and `tol2` are for checking convergence in successive parameter values; `tol3` is similar to `reltop` of `optim`; default values are `1e-03`, `1e-04`, and `1e-08`, respectively.

numeric_deriv character string denoting the type of numerical derivatives to be used. Options are `"fd"` for forward differences, and `cd` for central difference; default is `"fd"`.

nAGQ numeric scalar denoting the number of quadrature points; default is 11 when the number of random effects is one or two, and 7 otherwise.

update_GH_every numeric scalar denoting every how many iterations to update the quadrature points during the EM-phase; default is 10.

max_coef_value numeric scalar denoting the maximum allowable value for the fixed effects coefficients during the optimization; default is 10.

max_phis_value numeric scalar denoting the maximum allowable value for the shape/dispersion parameter of the negative binomial distribution during the optimization; default is `exp(10)`.

verbose logical; print information during the optimization phase; default is `FALSE`.

`...` arguments passed to `control`.

Details

General: The `mixed_model()` function fits mixed effects models in which the integrals over the random effects in the definition of the marginal log-likelihood cannot be solved analytically and need to be approximated. The function works under the assumption of normally distributed random effects with mean zero and variance-covariance matrix D. These integrals are approximated numerically using an adaptive Gauss-Hermite quadrature rule. Using the control argument `nAGQ`, the user can specify the number of quadrature points used in the approximation.

User-defined family: The user can define its own family object; for an example, see the help page of `negative.binomial`.

Optimization: A hybrid approach is used, starting with `iter_EM` iterations and unless convergence was achieved it continuous with a direct optimization of the log-likelihood using function `optim` and the algorithm specified by `optim_method`. For stability and speed, the derivative of the log-likelihood with respect to the parameters are internally programmed.

Value

An object of class `"MixMod"` with components:

`coefficients` a numeric vector with the estimated fixed effects.

`phis` a numeric vector with the estimated extra parameters.

`D` a numeric matrix denoting the estimated covariance matrix of the random effects.

`post_modes` a numeric matrix with the empirical Bayes estimates of the random effects.

`post_vars` a list of numeric matrices with the posterior variances of the random effects.

`logLik` a numeric scalar denoting the log-likelihood value at the end of the optimization procedure.

`Hessian` a numeric matrix denoting the Hessian matrix at the end of the optimization procedure.

`converged` a logical indicating whether convergence was attained.

`data` a copy of the `data` argument.

`id` a copy of the grouping variable from `data`.

`id_name` a character string with the name of the grouping variable.

Terms	a list with two terms components, termsX derived from the fixed argument, and termsZ derived from the random argument.
model_frames	a list with two model.frame components, mfX derived from the fixed argument, and mfZ derived from the random argument.
control	a copy of the (user-specific) control argument.
Funs	a list of functions used in the optimization procedure.
family	a copy of the family argument.
call	the matched call.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

See Also

methods.MixMod, effectPlotData, marginal_coefs

Examples

```
# simulate some data
set.seed(123L)
n <- 500
K <- 15
t.max <- 25

betas <- c(-2.13, -0.25, 0.24, -0.05)
D <- matrix(0, 2, 2)
D[1:2, 1:2] <- c(0.48, -0.08, -0.08, 0.18)

times <- c(replicate(n, c(0, sort(runif(K-1, 0, t.max)))))
group <- sample(rep(0:1, each = n/2))
DF <- data.frame(year = times, group = factor(rep(group, each = K)))
X <- model.matrix(~ group * year, data = DF)
Z <- model.matrix(~ year, data = DF)

b <- cbind(rnorm(n, sd = sqrt(D[1, 1])), rnorm(n, sd = sqrt(D[2, 2])))
id <- rep(1:n, each = K)
eta.y <- as.vector(X %*% betas + rowSums(Z * b[id, ]))
DF$y <- rbinom(n * K, 1, plogis(eta.y))
DF$id <- factor(id)

################################################

fm1 <- mixed_model(fixed = y ~ year * group, random = ~ 1 | id, data = DF,
                   family = binomial())

# fixed effects
fixef(fm1)

# random effects
```

```
head(ranef(fm1))

# detailed output
summary(fm1)

# fitted values for the 'mean subject', i.e., with
# random effects values equal to 0
head(fitted(fm1, type = "mean_subject"))

# fitted values for the conditioning on the estimated random effects
head(fitted(fm1, type = "subject_specific"))

###############

fm2 <- mixed_model(fixed = y ~ year, random = ~ 1 | id, data = DF,
                   family = binomial())

# likelihood ratio test between the two models
anova(fm2, fm1)

# the same hypothesis but with a Wald test
anova(fm1, L = rbind(c(0, 0, 1, 0), c(0, 0, 0, 1)))

###############

# An effects plot for the mean subject (i.e., with random effects equal to 0)
nDF <- with(DF, expand.grid(year = seq(min(year), max(year), length.out = 15),
    group = levels(group)))

plot_data <- effectPlotData(fm2, nDF)

require("lattice")
xyplot(pred + low + upp ~ year | group, data = plot_data,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
       xlab = "Follow-up time", ylab = "log odds")

expit <- function (x) exp(x) / (1 + exp(x))
xyplot(expit(pred) + expit(low) + expit(upp) ~ year | group, data = plot_data,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
       xlab = "Follow-up time", ylab = "Probabilities")

# An effects plots for the marginal probabilities
plot_data_m <- effectPlotData(fm2, nDF, marginal = TRUE, cores = 1L)

expit <- function (x) exp(x) / (1 + exp(x))
xyplot(expit(pred) + expit(low) + expit(upp) ~ year | group, data = plot_data_m,
       type = "l", lty = c(1, 2, 2), col = c(2, 1, 1), lwd = 2,
       xlab = "Follow-up time", ylab = "Probabilities")

###############

# include random slopes
fm1_slp <- update(fm1, random = ~ year | id)
```

```
# increase the number of quadrature points to 15
fm1_slp_q15 <- update(fm1_slp, nAGQ = 15)

# a diagonal covariance matrix for the random effects
fm1_slp_diag <- update(fm1, random = ~ year || id)

anova(fm1_slp_diag, fm1_slp)
```

MixMod Methods *Various Methods for Standard Generics*

Description

Methods for object of class "MixMod" for standard generic functions.

Usage

```
coef(object, ...)

## S3 method for class 'MixMod'
coef(object, sub_model = c("main", "zero_part"),
    ...)

fixef(object, ...)

## S3 method for class 'MixMod'
fixef(object, sub_model = c("main", "zero_part"), ...)

ranef(object, ...)

## S3 method for class 'MixMod'
ranef(object, post_vars = FALSE, ...)

confint(object, parm, level = 0.95, ...)

## S3 method for class 'MixMod'
confint(object,
  parm = c("fixed-effects", "var-cov","extra", "zero_part"),
  level = 0.95, sandwich = FALSE, ...)

anova(object, ...)

## S3 method for class 'MixMod'
anova(object, object2, test = TRUE,
  L = NULL, sandwich = FALSE, ...)
```

```
fitted(object, ...)

## S3 method for class 'MixMod'
fitted(object,
  type = c("mean_subject", "subject_specific", "marginal"),
  link_fun = NULL, ...)

residuals(object, ...)

## S3 method for class 'MixMod'
residuals(object,
  type = c("mean_subject", "subject_specific", "marginal"),
  link_fun = NULL, tasnf_y = function (x) x, ...)

predict(object, ...)

## S3 method for class 'MixMod'
predict(object, newdata, newdata2 = NULL,
    type_pred = c("response", "link"),
    type = c("mean_subject", "subject_specific", "marginal", "zero_part"),
    se.fit = FALSE, M = 300, df = 10, scale = 0.3, level = 0.95,
    seed = 1, return_newdata = FALSE, sandwich = FALSE, ...)

simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'MixMod'
simulate(object, nsim = 1, seed = NULL,
    type = c("subject_specific", "mean_subject"), new_RE = FALSE,
    acount_MLEs_var = FALSE, sim_fun = NULL,
    sandwich = FALSE, ...)

terms(x, ...)

## S3 method for class 'MixMod'
terms(x, type = c("fixed", "random", "zi_fixed", "zi_random"), ...)

formula(x, ...)

## S3 method for class 'MixMod'
formula(x, type = c("fixed", "random", "zi_fixed", "zi_random"), ...)


model.frame(formula, ...)

## S3 method for class 'MixMod'
model.frame(formula, type = c("fixed", "random", "zi_fixed", "zi_random"), ...)
```

```
model.matrix(object, ...)

## S3 method for class 'MixMod'
model.matrix(object, type = c("fixed", "random", "zi_fixed", "zi_random"), ...)

nobs(object, ...)

## S3 method for class 'MixMod'
nobs(object, level = 1, ...)

VIF(object, ...)

## S3 method for class 'MixMod'
VIF(object, type = c("fixed", "zi_fixed"), ...)

cooks.distance(model, ...)

## S3 method for class 'MixMod'
cooks.distance(model, cores = max(parallel::detectCores() - 1, 1), ...)
```

Arguments

`object`, `object2`, `x`, `formula`, `model`	objects inheriting from class `"MixMod"`. When `object2` is also provided, then the model behind `object` must be nested within the model behind `object2`.
`sub_model`	character string indicating for which sub-model to extract the estimated coefficients; it is only relevant for zero-inflated models.
`post_vars`	logical; if `TRUE` the posterior variances of the random effects are returned as an extra attribute of the numeric matrix produced by `ranef()`.
`parm`	character string; for which type of parameters to calculate confidence intervals. Option `"var-cov"` corresponds to the variance-covariance matrix of the random effects. Option `extra` corresponds to extra (shape/dispersion) parameters in the distribution of the outcome (e.g., the θ parameter in the negative binomial family). Option `zero_inflated` corresponds to the coefficients of the zero-inflated sub-model.
`level`	numeric scalar between 0 and 1 denoting the level of the confidence interval. In the `nobs()` method it denotes the level at which the number of observations is counted. The value 0 corresponds to the number of independent sample units determined by the number of levels of the grouping variable. If set to a value greater than zero, it returns the total number of observations.
`test`	logical; should a p-value be calculated.
`L`	a numeric matrix representing a contrasts matrix. This is only used when in `anova()` only `object` is provided, and it can only be specified for the fixed effects. When `L` is used, a Wald test is performed.
`sandwich`	logical; if `TRUE` the sandwich estimator is used in the calculation of standard errors.

`type` character string indicating the type of fitted values / residuals / predictions / variance inflation factors to calculate. Option `"mean_subject"` corresponds to only using the fixed-effects part; option `"subject_specific"` corresponds to using both the fixed- and random-effects parts; option `"marginal"` is based in multiplying the fixed effects design matrix with the marginal coefficients obtained by `marginal_coefs`.

`link_fun` the `link_fun` of `marginal_coefs`.

`tasnf_y` a function to transform the grouped / repeated measurements outcome before calculating the residuals; for example, relevant in two-part models for semi-continuous data, in which it is assumed that the log outcome follows a normal distribution.

`newdata, newdata2` a data frame based on which predictions are to be calculated. `newdata2` is only relevant when `level = "subject_specific"`; see **Details** for more information.

`type_pred` character string indicating at which scale to calculate predictions. Options are `"link"` indicating to calculate predictions at the link function / linear predictor scale, and `"response"` indicating to calculate predictions at the scale of the response variable.

`se.fit` logical, if `TRUE` standard errors of predictions are returned.

`M` numeric scalar denoting the number of Monte Carlo samples; see **Details** for more information.

`df` numeric scalar denoting the degrees of freedom for the Student's t proposal distribution; see **Details** for more information.

`scale` numeric scalar or vector denoting the scaling applied to the subject-specific covariance matrices of the random effects; see **Details** for more information.

`seed` numerical scalar giving the seed to be used in the Monte Carlo scheme.

`return_newdata` logical; if `TRUE` the `predict()` method returns a copy of the `newdata` and of `newdata2` if the corresponding argument was not `NULL`, with extra columns the predictions, and the lower and upper limits of the cofidence intervals when `type = "subject_specific"`.

`nsim` numeric scalar giving the number of times to simulate the response variable.

`new_RE` logical; if `TRUE`, new random effects will be simulated, and new outcome data will be simulated by `simulate()` using these new random effect. Otherwise, the empirical Bayes estimates of the random effects from the fitted model will be used.

`acount_MLEs_var` logical; if `TRUE` it accounts for the variability of the maximum likelihood estimates (MLEs) by simulating a new value for the parameters from a multivariate normal distribution with mean the MLEs and covariance matrix the covariance matrix of the MLEs.

`sim_fun` a function based on which to simulate the response variable. This is relevant for non-standard models. The `simulate()` function also tries to extract this function from the `family` component of `object`. The function should have the

following four arguments: `n` a numeric scalar denoting the number of observations to simulate, `mu` a numeric vector of means, `phis` a numeric vector of extra dispersion/scale parameters, and `eta_zi` a numeric vector for the zero-part of the model, if this is relevant.

`cores` the number of cores to use in the computation.

`...` further arguments; currently none is used.

Details

In generic terms, we assume that the mean of the outcome y_i ($i = 1, ..., n$ denotes the subjects) conditional on the random effects is given by the equation:

$$gE(y_i|b_i) = \eta_i = X_i\beta + Z_ib_i,$$

where $g(.)$ denotes the link function, b_i the vector of random effects, β the vector of fixed effects, and X_i and Z_i the design matrices for the fixed and random effects, respectively.

Argument `type_pred` of `predict()` specifies whether predictions will be calculated in the link / linear predictor scale, i.e., η_i or in the response scale, i.e., $gE(y_i|b_i)$.

When `type = "mean_subject"`, predictions are calculated using only the fixed effects, i.e., the $X_i\beta$ part, where X_i is evaluated in `newdata`. This corresponds to predictions for the 'mean' subjects, i.e., subjects who have random effects value equal to 0. Note, that in the case of nonlinear link functions this does not correspond to the averaged over the population predictions (i.e., marginal predictions).

When `type = "marginal"`, predictions are calculated using only the fixed effects, i.e., the $X_i\beta$ part, where X_i is evaluated in `newdata`, but with β coefficients the marginalized coefficients obtain from `marginal_coefs`. These predictions will be marginal / population averaged predictions.

When `type = "zero_part"`, predictions are calculated for the logistic regression of the extra zero-part of the model (i.e., applicable for zero-inflated and hurdle models).

When `type = "subject_specific"`, predictions are calculated using both the fixed- and random-effects parts, i.e., $X_i\beta + Z_ib_i$, where X_i and Z_i are evaluated in `newdata`. Estimates for the random effects of each subject are obtained as modes from the posterior distribution $[b_i|y_i;\theta]$ evaluated in `newdata` and with $theta$ (denoting the parameters of the model, fixed effects and variance components) replaced by their maximum likelihood estimates.

Notes: (i) When `se.fit = TRUE` and `type_pred = "response"`, the standard errors returned are on the linear predictor scale, not the response scale. (ii) When `se.fit = TRUE` and the model contains an extra zero-part, no standard errors are computed when `type = "mean_subject"`. (iii) When the model contains an extra zero-part, `type = "marginal"` predictions are not yet implemented.

When `se.fit = TRUE` and `type = "subject_specific"`, standard errors and confidence intervals for the subject-specific predictions are obtained by a Monte Carlo scheme entailing three steps repeated `M` times, namely

Step I Account for the variability of maximum likelihood estimates (MLES) by simulating a new value θ^* for the parameters θ from a multivariate normal distribution with mean the MLEs and covariance matrix the covariance matrix of the MLEs.

Step II Account for the variability in the random effects estimates by simulating a new value b_i^* for the random effects b_i from the posterior distribution $[b_i|y_i;\theta^*]$. Because the posterior

distribution does not have a closed-form, a Metropolis-Hastings algorithm is used to sample the new value b_i^* using as proposal distribution a multivariate Student's-t distribution with degrees of freedom `df`, centered at the mode of the posterior distribution $[b_i|y_i;\theta]$ with θ the MLEs, and scale matrix the inverse Hessian matrix of the log density of $[b_i|y_i;\theta]$ evaluated at the modes, but multiplied by `scale`. The `scale` and `df` parameters can be used to adjust the acceptance rate.

Step III The predictions are calculated using $X_i\beta^* + Z_i b_i^*$.

Argument `newdata2` can be used to calculate dynamic subject-specific predictions. I.e., using the observed responses y_i in `newdata`, estimates of the random effects of each subject are obtained. For the same subjects we want to obtain predictions in new covariates settings for which no response data are yet available. For example, in a longitudinal study, for a subject we have responses up to a follow-up t (`newdata`) and we want the prediction at $t + \Delta t$ (`newdata2`).

Value

The estimated fixed and random effects, coefficients (this is similar as in package **nlme**), confidence intervals fitted values (on the scale on the response) and residuals.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

See Also

`mixed_model`, `marginal_coefs`

Examples

```
# simulate some data
set.seed(123L)
n <- 500
K <- 15
t.max <- 25

betas <- c(-2.13, -0.25, 0.24, -0.05)
D <- matrix(0, 2, 2)
D[1:2, 1:2] <- c(0.48, -0.08, -0.08, 0.18)

times <- c(replicate(n, c(0, sort(runif(K-1, 0, t.max)))))
group <- sample(rep(0:1, each = n/2))
DF <- data.frame(year = times, group = factor(rep(group, each = K)))
X <- model.matrix(~ group * year, data = DF)
Z <- model.matrix(~ year, data = DF)

b <- cbind(rnorm(n, sd = sqrt(D[1, 1])), rnorm(n, sd = sqrt(D[2, 2])))
id <- rep(1:n, each = K)
eta.y <- as.vector(X %*% betas + rowSums(Z * b[id, ]))
DF$y <- rbinom(n * K, 1, plogis(eta.y))
DF$id <- factor(id)
```

```
################################################

fm1 <- mixed_model(fixed = y ~ year + group, random = ~ year | id, data = DF,
                   family = binomial())

head(coef(fm1))
fixef(fm1)
head(ranef(fm1))


confint(fm1)
confint(fm1, "var-cov")

head(fitted(fm1, "subject_specific"))
head(residuals(fm1, "marginal"))

fm2 <- mixed_model(fixed = y ~ year * group, random = ~ year | id, data = DF,
                   family = binomial())

# likelihood ratio test between fm1 and fm2
anova(fm1, fm2)

# the same but with a Wald test
anova(fm2, L = rbind(c(0, 0, 0, 1)))
```

`negative.binomial` *Family function for Negative Binomial Mixed Models*

Description

Specifies the information required to fit a Negative Binomial generalized linear mixed model, using `mixed_model()`.

Usage

```
negative.binomial()
```

Note

Currently only the log-link is implemented.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

Examples

```
# simulate some data
set.seed(102)
dd <- expand.grid(f1 = factor(1:3), f2 = LETTERS[1:2], g = 1:30, rep = 1:15,
                  KEEP.OUT.ATTRS = FALSE)
mu <- 5 * (-4 + with(dd, as.integer(f1) + 4 * as.numeric(f2)))
dd$y <- rnbinom(nrow(dd), mu = mu, size = 0.5)

gm1 <-  mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                    family = negative.binomial())

summary(gm1)

# We do a likelihood ratio test with the Poisson mixed model
gm0 <- mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                    family = poisson())

anova(gm0, gm1)

# Define a cutom-made family function to be used with mixed_model()
# the required components are 'family', 'link', 'linkfun', 'linkinv' and 'log_dens';
# the extra functions 'score_eta_fun' and 'score_phis_fun' can be skipped and will
# internally approximated using numeric derivatives (though it is better that you provide
# them).
my_negBinom <- function (link = "log") {
    stats <- make.link(link)
    log_dens <- function (y, eta, mu_fun, phis, eta_zi) {
        # the log density function
        phis <- exp(phis)
        mu <- mu_fun(eta)
        log_mu_phis <- log(mu + phis)
        comp1 <- lgamma(y + phis) - lgamma(phis) - lgamma(y + 1)
        comp2 <- phis * log(phis) - phis * log_mu_phis
        comp3 <- y * log(mu) - y * log_mu_phis
        out <- comp1 + comp2 + comp3
        attr(out, "mu_y") <- mu
        out
    }
    score_eta_fun <- function (y, mu, phis, eta_zi) {
        # the derivative of the log density w.r.t. mu
        phis <- exp(phis)
        mu_phis <- mu + phis
        comp2 <- - phis / mu_phis
        comp3 <- y / mu - y / mu_phis
        # the derivative of mu w.r.t. eta (this depends on the chosen link function)
        mu.eta <- mu
        (comp2 + comp3) * mu.eta
    }
    score_phis_fun <- function (y, mu, phis, eta_zi) {
        # the derivative of the log density w.r.t. phis
        phis <- exp(phis)
```

```
        mu_phis <- mu + phis
        comp1 <- digamma(y + phis) - digamma(phis)
        comp2 <- log(phis) + 1 - log(mu_phis) - phis / mu_phis
        comp3 <- - y / mu_phis
        comp1 + comp2 + comp3
    }
    structure(list(family = "user Neg Binom", link = stats$name, linkfun = stats$linkfun,
                   linkinv = stats$linkinv, log_dens = log_dens,
                   score_eta_fun = score_eta_fun,
                   score_phis_fun = score_phis_fun),
              class = "family")
}

fm <-  mixed_model(fixed = y ~ f1 * f2, random = ~ 1 | g, data = dd,
                   family = my_negBinom(), n_phis = 1,
                   initial_values = list("betas" = poisson()))

summary(fm)
```

`scoring_rules` *Proper Scoring Rules for Categorical Data*

Description

Calculates the logarithmic, quadratic/Brier and spherical based on a fitted mixed model for categorical data.

Usage

```
scoring_rules(object, newdata, newdata2 = NULL, max_count = 2000,
    return_newdata = FALSE)
```

Arguments

`object`	an object inheriting from class `"MixMod"`.
`newdata`	a data.frame based on which to estimate the random effect and calculate predictions. It should contain the response variable.
`newdata2`	a data.frame based on which to estimate the random effect and calculate predictions. It should contain the response variable.
`max_count`	numeric scalar denoting the maximum count up to which to calculate probabilities; this is relevant for count response data.
`return_newdata`	logical; if `TRUE` the values of the scoring rules are ruturned as extra columns of the `newdata` or `newdata2` data.frame.

Value

A data.frame with (extra) columns the values of the logarithmic, quadratic and spherical scoring rules calculated based on the fitted model and the observed responses in `newdata` or `newdata2`.

Author(s)

Dimitris Rizopoulos <d.rizopoulos@erasmusmc.nl>

References

Carvalho, A. (2016). An overview of applications of proper scoring rules. *Decision Analysis* **13**, 223–242. doi:10.1287/deca.2016.0337

See Also

mixed_model, predict.MixMod

Examples

```
NA
NA
NA
```

Index