Package: glmmTMB (via r-universe)

November 5, 2024

Title Generalized Linear Mixed Models using Template Model Builder

Version 1.1.10.9000

Description Fit linear and generalized linear mixed models with various extensions, including zero-inflation. The models are fitted using maximum likelihood estimation via 'TMB' (Template Model Builder). Random effects are assumed to be Gaussian on the scale of the linear predictor and are integrated out using the Laplace approximation. Gradients are calculated using automatic differentiation.

License AGPL-3

Depends R (>= 3.6.0)

Imports methods, TMB (>= 1.9.0), lme4 (>= 1.1-18.9000), Matrix, nlme, numDeriv, mgcv, reformulas (>= 0.2.0)

LinkingTo TMB, RcppEigen

Suggests knitr, rmarkdown, testthat, MASS, lattice, ggplot2 (>= 2.2.1), mlmRev, bbmle (>= 1.0.19), pscl, coda, reshape2, car (>= 3.0.6), emmeans (>= 1.4), estimability, DHARMa, multcomp, MuMIn, effects (>= 4.0-1), dotwhisker, broom, broom.mixed, plyr, png, boot, texreg, xtable, huxtable, parallel, blme, purrr, dplyr, ade4, ape, gsl

SystemRequirements GNU make

VignetteBuilder knitr, rmarkdown

URL https://github.com/glmmTMB/glmmTMB

LazyData TRUE

BugReports https://github.com/glmmTMB/glmmTMB/issues

NeedsCompilation yes

Encoding UTF-8

RoxygenNote 7.3.2

Config/pak/sysreqs cmake make

Repository https://glmmtmb.r-universe.dev

RemoteUrl https://github.com/glmmtmb/glmmtmb

RemoteRef HEAD

RemoteSha 70f24c7295eded8195ba68406772915741e49430

Contents

`Anova.glmmTMB` *Downstream methods*

Description

Methods have been written that allow `glmmTMB` objects to be used with several downstream packages that enable different forms of inference. For some methods (`Anova` and `emmeans`, but *not* `effects` at present), set the `component` argument to "cond" (conditional, the default), "zi" (zero-inflation) or "disp" (dispersion) in order to produce results for the corresponding part of a `glmmTMB` model. Support for **emmeans** also allows additional options `component = "response"` (response means taking both the `cond` and `zi` components into account), and `component = "cmean"` (mean of the [possibly truncated] conditional distribution).

In particular,

- `car::Anova` constructs type-II and type-III Anova tables for the fixed effect parameters of any component
- the `emmeans` package computes estimated marginal means (previously known as least-squares means) for the fixed effects of any component, or predictions with `type = "response"` or `type = "component"`. Note: In hurdle models, `component = "cmean"` produces means of the truncated conditional distribution, while `component = "cond", type = "response"` produces means of the *untruncated* conditional distribution.
- the `effects` package computes graphical tabular effect displays (only for the fixed effects of the conditional component)

Usage

```
Anova.glmmTMB(
  mod,
  type = c("II", "III", 2, 3),
  test.statistic = c("Chisq", "F"),
  component = "cond",
  vcov. = vcov(mod)[[component]],
  singular.ok,
  include.rankdef.cols = FALSE,
  ...
)

Effect.glmmTMB(focal.predictors, mod, ...)
```

Arguments

`mod`	a glmmTMB model
`type`	type of test, `"II"`, `"III"`, `2`, or `3`. Roman numerals are equivalent to the corresponding Arabic numerals. See `Anova` for details.
`test.statistic`	unused: only valid choice is "Chisq" (i.e., Wald chi-squared test)

component	which component of the model to test/analyze ("cond", "zi", or "disp") or, in **emmeans** only, "response" or "cmean" as described in Details.
vcov.	variance-covariance matrix (usually extracted automatically)
singular.ok	OK to do ANOVA with singular models (unused) ?
include.rankdef.cols	include all columns of a rank-deficient model matrix?
...	Additional parameters that may be supported by the method.
focal.predictors	a character vector of one or more predictors in the model in any order.

Details

While the examples below are disabled for earlier versions of R, they may still work; it may be necessary to refer to private versions of methods, e.g. `glmmTMB:::Anova.glmmTMB(model, ...)`.

Examples

```
warp.lm <- glmmTMB(breaks ~ wool * tension, data = warpbreaks)
salamander1 <- up2date(readRDS(system.file("example_files","salamander1.rds",package="glmmTMB")))
if (require(emmeans)) withAutoprint({
    emmeans(warp.lm, poly ~ tension | wool)
    emmeans(salamander1, ~ mined, type="response")  # conditional means
    emmeans(salamander1, ~ mined, component="cmean")     # same as above, but re-gridded
    emmeans(salamander1, ~ mined, component="zi", type="response")  # zero probabilities
   emmeans(salamander1, ~ mined, component="response")  # response means including both components
})
if (getRversion() >= "3.6.0") {
   if (require(car)) withAutoprint({
       Anova(warp.lm,type="III")
       Anova(salamander1)
       Anova(salamander1, component="zi")
   })
   if (require(effects)) withAutoprint({
       plot(allEffects(warp.lm))
       plot(allEffects(salamander1))
   })
}
```

---

as.theta.vcov *Get theta parameterisation of a covariance structure*

Description

Get theta parameterisation of a covariance structure

Usage

```
as.theta.vcov(Sigma, corrs.only = FALSE)
```

Arguments

Sigma	a covariance matrix
corrs.only	return only values corresponding to the correlation matrix parameters?

Value

the corresponding theta parameter vector

---

confint.glmmTMB *Calculate confidence intervals*

Description

Calculate confidence intervals

Usage

```
## S3 method for class 'glmmTMB'
confint(
  object,
  parm = NULL,
  level = 0.95,
  method = c("wald", "Wald", "profile", "uniroot"),
  component = c("all", "cond", "zi", "other"),
  estimate = TRUE,
  include_nonest = FALSE,
  parallel = c("no", "multicore", "snow"),
  ncpus = getOption("profile.ncpus", 1L),
  cl = NULL,
  full = FALSE,
  ...
)
```

Arguments

object	glmmTMB fitted object.
parm	which parameters to profile, specified • by index (position) [*after* component selection for confint, if any] • by name (matching the row/column names of vcov(object,full=TRUE)) • as "theta_" (random-effects variance-covariance parameters), "beta_" (conditional and zero-inflation parameters), or "disp_" or "sigma" (dispersion parameters) Parameter indexing by number may give unusual results when some parameters have been fixed using the map argument: please report surprises to the package maintainers.

level	Confidence level.
method	'wald', 'profile', or 'uniroot': see Details function)
component	Which of the three components 'cond', 'zi' or 'other' to select. Default is to select 'all'.
estimate	(logical) add a third column with estimate ?
include_nonest	include dummy rows for non-estimated (mapped, rank-deficient) parameters?
parallel	method (if any) for parallel computation
ncpus	number of CPUs/cores to use for parallel computation
cl	cluster to use for parallel computation
full	CIs for all parameters (including dispersion) ?
...	arguments may be passed to profile.glmmTMB (and possibly from there to tmbprofile) or tmbroot

Details

Available methods are

"wald" These intervals are based on the standard errors calculated for parameters on the scale of their internal parameterization depending on the family. Derived quantities such as standard deviation parameters and dispersion parameters are back-transformed. It follows that confidence intervals for these derived quantities are typically asymmetric.

"profile" This method computes a likelihood profile for the specified parameter(s) using profile.glmmTMB; fits a spline function to each half of the profile; and inverts the function to find the specified confidence interval.

"uniroot" This method uses the uniroot function to find critical values of one-dimensional profile functions for each specified parameter.

At present, "wald" returns confidence intervals for variance parameters on the standard deviation/correlation scale, while "profile" and "uniroot" report them on the underlying ("theta") scale: for each random effect, the first set of parameter values are standard deviations on the log scale, while remaining parameters represent correlations on the scaled Cholesky scale. For a random effects model with two elements (such as a random-slopes model, or a random effect of factor with two levels), there is a single correlation parameter θ; the correlation is equal to $\rho = \theta/\sqrt{1+\theta^2}$. For random-effects terms with more than two elements, the mapping is more complicated: see https://github.com/glmmTMB/glmmTMB/blob/master/misc/glmmTMB_corcalcs.ipynb

Examples

```
data(sleepstudy, package="lme4")
model <- glmmTMB(Reaction ~ Days + (1|Subject), sleepstudy)
model2 <- glmmTMB(Reaction ~ Days + (1|Subject), sleepstudy,
    dispformula= ~I(Days>8))
confint(model)  ## Wald/delta-method CIs
confint(model,parm="theta_")  ## Wald/delta-method CIs
confint(model,parm=1,method="profile")
```

diagnose *diagnose model problems*

Description

EXPERIMENTAL. For a given model, this function attempts to isolate potential causes of convergence problems. It checks (1) whether there are any unusually large coefficients; (2) whether there are any unusually scaled predictor variables; (3) if the Hessian (curvature of the negative log-likelihood surface at the MLE) is positive definite (i.e., whether the MLE really represents an optimum). For each case it tries to isolate the particular parameters that are problematic.

Usage

```
diagnose(
  fit,
  eval_eps = 1e-05,
  evec_eps = 0.01,
  big_coef = 10,
  big_sd_log10 = 3,
  big_zstat = 5,
  check_coefs = TRUE,
  check_zstats = TRUE,
  check_hessian = TRUE,
  check_scales = TRUE,
  explain = TRUE
)
```

Arguments

fit	a glmmTMB fit
eval_eps	numeric tolerance for 'bad' eigenvalues
evec_eps	numeric tolerance for 'bad' eigenvector elements
big_coef	numeric tolerance for large coefficients
big_sd_log10	numeric tolerance for badly scaled parameters (log10 scale), i.e. for default value of 3, predictor variables with sd less than 1e-3 or greater than 1e3 will be flagged)
big_zstat	numeric tolerance for Z-statistic
check_coefs	identify large-magnitude coefficients? (Only checks conditional-model parameters if a (log, logit, cloglog, probit) link is used. Always checks zero-inflation, dispersion, and random-effects parameters. May produce false positives if predictor variables have extremely large scales.)
check_zstats	identify parameters with unusually large Z-statistics (ratio of standard error to mean)? Identifies likely failures of Wald confidence intervals/p-values.
check_hessian	identify non-positive-definite Hessian components?
check_scales	identify predictors with unusually small or large scales?
explain	provide detailed explanation of each test?

Details

Problems in one category (e.g. complete separation) will generally also appear in "downstream" categories (e.g. non-positive-definite Hessians). Therefore, it is generally advisable to try to deal with problems in order, e.g. address problems with complete separation first, then re-run the diagnostics to see whether Hessian problems persist.

Value

a logical value based on whether anything questionable was found

---

`dtruncated_nbinom2` *truncated distributions*

Description

Probability functions for k-truncated Poisson and negative binomial distributions.

Usage

```
dtruncated_nbinom2(x, size, mu, k = 0, log = FALSE)

dtruncated_poisson(x, lambda, k = 0, log = FALSE)

dtruncated_nbinom1(x, phi, mu, k = 0, log = FALSE)
```

Arguments

`x`	value
`size`	number of trials/overdispersion parameter
`mu`	mean parameter
`k`	truncation parameter
`log`	(logical) return log-probability?
`lambda`	mean parameter
`phi`	overdispersion parameter

epil2 *Seizure Counts for Epileptics - Extended*

Description

Extended version of the `epil` dataset of the **MASS** package. The three transformed variables `Visit`, `Base`, and `Age` used by Booth et al. (2003) have been added to `epil`.

Usage

```
epil2
```

Format

A data frame with 236 observations on the following 12 variables:

`y` an integer vector.

`trt` a factor with levels `"placebo"` and `"progabide"`.

`base` an integer vector.

`age` an integer vector.

`V4` an integer vector.

`subject` an integer vector.

`period` an integer vector.

`lbase` a numeric vector.

`lage` a numeric vector.

Visit `(rep(1:4,59) - 2.5) / 5`.

Base `log(base/4)`.

Age `log(age)`.

References

Booth, J.G., G. Casella, H. Friedl, and J.P. Hobert. (2003) Negative binomial loglinear mixed models. *Statistical Modelling* **3**, 179–191.

Examples

```
epil2$subject <- factor(epil2$subject)
op <- options(digits=3)
(fm <- glmmTMB(y ~ Base*trt + Age + Visit + (Visit|subject),
              data=epil2, family=nbinom2))
meths <- methods(class = class(fm))
if((Rv <- getRversion()) > "3.1.3") {
  funs <- attr(meths, "info")[, "generic"]
  funs <- setdiff(funs, "profile")  ## too slow! pkgdown is trying to run this??
  for(fun in funs[is.na(match(funs, "getME"))]) {
```

```
        cat(sprintf("%s:\n-----\n", fun))
        r <- tryCatch( get(fun)(fm), error=identity)
        if (inherits(r, "error")) cat("** Error:", r$message,"\n")
        else tryCatch( print(r) )
        cat(sprintf("---end{%s}--------------\n\n", fun))
  }
}
options(op)
```

`family_params` *Retrieve family-specific parameters*

Description

Most conditional distributions have only parameters governing their location (retrieved via `predict`) and scale (`sigma`). A few (e.g. Tweedie, Student t, ordered beta) are characterized by one or more additional parameters.

Usage

```
family_params(object)
```

Arguments

`object`	glmmTMB object

Value

a named numeric vector

`fitTMB` *Optimize TMB models and package results, modularly*

Description

These functions (called internally by `glmmTMB`) perform the actual model optimization, after all of the appropriate structures have been set up (`fitTMB`), and finalize the model after optimization (`finalizeTMB`). It can be useful to run `glmmTMB` with `doFit=FALSE`, adjust the components as required, and then finish the fitting process with `fitTMB` (however, it is the user's responsibility to make sure that any modifications create an internally consistent final fitted object).

Usage

```
fitTMB(TMBStruc, doOptim = TRUE)

finalizeTMB(TMBStruc, obj, fit, h = NULL, data.tmb.old = NULL)
```

Arguments

TMBStruc	a list containing lots of stuff ...
doOptim	logical; do optimization? If FALSE, return TMB object
obj	object created by `fitTMB(., doOptim = FALSE)`
fit	a fitted object returned from `nlminb`, or more generally a similar list (i.e. containing elements `par`, `objective`, `convergence`, `message`, `iterations`, `evaluations`)
h	Hessian matrix for fit, if computed in previous step
data.tmb.old	stored TMB data, if computed in previous step

Examples

```
## 1. regular (non-modular) model fit:
m0 <- glmmTMB(count ~ mined + (1|site),
             family=poisson, data=Salamanders)
## 2. the equivalent fit, done modularly:
##  a.
m1 <- glmmTMB(count ~ mined + (1|site),
             family=poisson, data=Salamanders,
             doFit = FALSE)
## result is a list of elements (data to be passed to TMB,
## random effects structures, etc.) needed to fit the model
names(m1)
## b. The next step calls TMB to set up the automatic differentiation
## machinery
m2 <- fitTMB(m1, doOptim = FALSE)
## The result includes initial parameter values, objective function
## (fn), gradient function (gr), etc.
names(m2)
## Optionally, one could choose to
## modify the components of m1$env$data at this point ...
## updating the TMB structure as follows may be necessary:
m2 <- with(m2$env,
              TMB::MakeADFun(data,
                             parameters,
                             map = map,
                             random = random,
                             silent = silent,
                             DLL = "glmmTMB"))
## c. Use the starting values, objective function, and gradient
## function set up in the previous step to do the nonlinear optimization
m3 <- with(m2, nlminb(par, objective = fn, gr = gr))
## the resulting object contains the fitted parameters, value of
## the objective function, information on convergence, etc.
names(m3)
## d. The last step is to combine the information from the previous
## three steps into a \code{glmmTMB} object that is equivalent to
## the original fit
m4 <- finalizeTMB(m1, m2, m3)
m4$call$doFit <- NULL ## adjust 'call' element to match
all.equal(m0, m4)
```

`fixef` *Extract fixed-effects estimates*

Description

Extract Fixed Effects

Usage

```
## S3 method for class 'glmmTMB'
fixef(object, ...)
```

Arguments

`object`	any fitted model object from which fixed effects estimates can be extracted.
`...`	optional additional arguments. Currently none are used in any methods.

Details

Extract fixed effects from a fitted `glmmTMB` model.

The print method for `fixef.glmmTMB` object *only displays non-trivial components*: in particular, the dispersion parameter estimate is not printed for models with a single (intercept) dispersion parameter (see examples)

Value

an object of class `fixef.glmmTMB` comprising a list of components (`cond`, `zi`, `disp`), each containing a (possibly zero-length) numeric vector of coefficients

Examples

```
data(sleepstudy, package = "lme4")
fm1 <- glmmTMB(Reaction ~ Days, sleepstudy)
(f1 <- fixef(fm1))
f1$cond
## show full coefficients, including empty z-i model and
## constant dispersion parameter
print(f1, print_trivials = TRUE)
```

formatVC *Format the 'VarCorr' Matrix of Random Effects*

Description

"format()" the 'VarCorr' matrix of the random effects – for print()ing and show()ing

Usage

```
formatVC(
  varcor,
  digits = max(3, getOption("digits") - 2),
  comp = "Std.Dev.",
  formatter = format,
  useScale = attr(varcor, "useSc"),
  ...
)
```

Arguments

varcor	a VarCorr (-like) matrix with attributes.
digits	the number of significant digits.
comp	character vector of length one or two indicating which columns out of "Variance" and "Std.Dev." should be shown in the formatted output.
formatter	the function to be used for formatting the standard deviations and or variances (but *not* the correlations which (currently) are always formatted as "0.nnn"
useScale	whether to report a scale parameter (e.g. residual standard deviation)
...	optional arguments for formatter(*) in addition to the first (numeric vector) and digits.

Value

a character matrix of formatted VarCorr entries from varc.

formula.glmmTMB *Extract the formula of a glmmTMB object*

Description

Extract the formula of a glmmTMB object

Usage

```
## S3 method for class 'glmmTMB'
formula(x, fixed.only = FALSE, component = c("cond", "zi", "disp"), ...)
```

Arguments

x	a glmmTMB object
fixed.only	(logical) drop random effects, returning only the fixed-effect component of the formula?
component	formula for which component of the model to return (conditional, zero-inflation, or dispersion)
...	unused, for generic consistency

---

getCapabilities *List model options that glmmTMB knows about*

Description

List model options that glmmTMB knows about

Usage

```
getCapabilities(what = "all", check = FALSE)
```

Arguments

what	(character) which type of model structure to report on ("all","family","link","covstruct")
check	(logical) do brute-force checking to test whether families are really implemented (only available for what="family")

Value

if check==FALSE, returns a vector of the names (or a list of name vectors) of allowable entries; if check==TRUE, returns a logical vector of working families

Note

these are all the options that are *defined* internally; they have not necessarily all been *implemented* (FIXME!)

`getME.glmmTMB` *Extract or Get Generalize Components from a Fitted Mixed Effects Model*

Description

Extract or Get Generalize Components from a Fitted Mixed Effects Model

Usage

```
## S3 method for class 'glmmTMB'
getME(
  object,
  name = c("X", "Xzi", "Z", "Zzi", "Xdisp", "theta", "beta", "b"),
  ...
)
```

Arguments

`object`	a fitted `glmmTMB` object
`name`	of the component to be retrieved
`...`	ignored, for method compatibility

See Also

`getME` Get generic and re-export:

`getReStruc` *Calculate random effect structure Calculates number of random effects, number of parameters, block size and number of blocks. Mostly for internal use.*

Description

Calculate random effect structure Calculates number of random effects, number of parameters, block size and number of blocks. Mostly for internal use.

Usage

```
getReStruc(reTrms, ss = NULL, aa = NULL, reXterms = NULL, fr = NULL)
```

Arguments

reTrms	random-effects terms list
ss	a vector of character strings indicating a valid covariance structure (one for each RE term). Must be one of names(glmmTMB:::.valid_covstruct); default is to use an unstructured variance-covariance matrix ("us") for all blocks).
aa	additional arguments (i.e. rank, or var-cov matrix)
reXterms	terms objects corresponding to each RE term
fr	model frame

Value

a list

blockNumTheta	number of variance covariance parameters per term
blockSize	size (dimension) of one block
blockReps	number of times the blocks are repeated (levels)
covCode	structure code
simCode	simulation code; should we "zero" (set to zero/ignore), "fix" (set to existing parameter values), "random" (draw new random deviations)?

Examples

```
data(sleepstudy, package="lme4")
rt <- lme4::lFormula(Reaction~Days+(1|Subject)+(0+Days|Subject),
                     sleepstudy)$reTrms
rt2 <- lme4::lFormula(Reaction~Days+(Days|Subject),
                     sleepstudy)$reTrms
getReStruc(rt)
getReStruc(rt2)
```

---

getXReTrms *Create X and random effect terms from formula*

Description

Create X and random effect terms from formula

Usage

```
getXReTrms(
  formula,
  mf,
  fr,
  ranOK = TRUE,
  type = "",
```

```
  contrasts,
  sparse = FALSE,
  old_smooths = NULL
)
```

Arguments

formula	current formula, containing both fixed & random effects
mf	matched call
fr	full model frame
ranOK	random effects allowed here?
type	label for model type
contrasts	a list of contrasts (see ?glmmTMB)
sparse	(logical) return sparse model matrix?
old_smooths	smooth information from a prior model fit (for prediction)

Value

a list composed of

X	design matrix for fixed effects
Z	design matrix for random effects
reTrms	output from mkReTrms, possibly augmented with information about mgcv-style smooth terms
ss	splitform of the formula
aa	additional arguments, used to obtain rank
terms	terms for the fixed effects
offset	offset vector, or vector of zeros if offset not specified
reXterms	terms for the model matrix in each RE term

---

get_cor *translate vector of correlation parameters to correlation values*

Description

translate vector of correlation parameters to correlation values

Usage

```
get_cor(theta, return_val = c("vec", "mat"))

put_cor(C, input_val = c("mat", "vec"))
```

Arguments

theta	vector of internal correlation parameters (elements of scaled Cholesky factor, in *row-major* order)
return_val	return a vector of correlation values from the lower triangle ("vec"), or the full correlation matrix ("mat")?
C	a correlation matrix
input_val	input a vector of correlation values from the lower triangle ("vec"), or the full correlation matrix ("mat")?

Details

These functions follow the definition at http://kaskr.github.io/adcomp/classdensity_1_1UNSTRUCTURED__CORR__t.html: if L is the lower-triangular matrix with 1 on the diagonal and the correlation parameters in the lower triangle, then the correlation matrix is defined as $\Sigma = D^{-1/2}LL^\top D^{-1/2}$, where $D = \text{diag}(LL^\top)$. For a single correlation parameter θ_0, this works out to $\rho = \theta_0/\sqrt{1+\theta_0^2}$. The get_cor function returns the elements of the lower triangle of the correlation matrix, in column-major order.

Value

a vector of correlation values (get_cor) or glmmTMB scaled-correlation parameters (put_cor)

Examples

```
th0 <- 0.5
stopifnot(all.equal(get_cor(th0),th0/sqrt(1+th0^2)))
set.seed(101)
C <- get_cor(rnorm(21), return_val = "mat")
## test: round-trip
stopifnot(all.equal(get_cor(put_cor(C), return_val = "mat"), C))
```

---

glmmTMB *Fit Models with TMB*

Description

Fit a generalized linear mixed model (GLMM) using Template Model Builder (TMB).

Usage

```
glmmTMB(
  formula,
  data = NULL,
  family = gaussian(),
  ziformula = ~0,
  dispformula = ~1,
  weights = NULL,
```

```
  offset = NULL,
  contrasts = NULL,
  na.action,
  se = TRUE,
  verbose = FALSE,
  doFit = TRUE,
  control = glmmTMBControl(),
  REML = FALSE,
  start = NULL,
  map = NULL,
  sparseX = NULL,
  priors = NULL
)
```

Arguments

formula	combined fixed and random effects formula, following lme4 syntax.
data	data frame (tibbles are OK) containing model variables. Not required, but strongly recommended; if data is not specified, downstream methods such as prediction with new data (predict(fitted_model, newdata = ...)) will fail. If it is necessary to call glmmTMB with model variables taken from the environment rather than from a data frame, specifying data=NULL will suppress the warning message.
family	a family function, a character string naming a family function, or the result of a call to a family function (variance/link function) information. See family for a generic discussion of families or family_glmmTMB for details of glmmTMB-specific families.
ziformula	a *one-sided* (i.e., no response variable) formula for zero-inflation combining fixed and random effects: the default ~0 specifies no zero-inflation. Specifying ~. sets the zero-inflation formula identical to the right-hand side of formula (i.e., the conditional effects formula); terms can also be added or subtracted. **When using ~. as the zero-inflation formula in models where the conditional effects formula contains an offset term, the offset term will automatically be dropped**. The zero-inflation model uses a logit link.
dispformula	a *one-sided* formula for dispersion combining fixed and random effects: the default ~1 specifies the standard dispersion given any family. The argument is ignored for families that do not have a dispersion parameter. For an explanation of the dispersion parameter for each family, see sigma. The dispersion model uses a log link. In Gaussian mixed models, dispformula=~0 fixes the residual variance to be 0 (actually a small non-zero value), forcing variance into the random effects. The precise value can be controlled via control=glmmTMBControl(zero_dispval=...); the default value is sqrt(.Machine$double.eps).
weights	weights, as in glm. Not automatically scaled to have sum 1.
offset	offset for conditional model (only).
contrasts	an optional list, e.g., list(fac1="contr.sum"). See the contrasts.arg of model.matrix.default.

na.action	a function that specifies how to handle observations containing NAs. The default action (na.omit, inherited from the 'factory fresh' value of getOption("na.action")) strips any observations with any missing values in any variables. Using na.action = na.exclude will similarly drop observations with missing values while fitting the model, but will fill in NA values for the predicted and residual values for cases that were excluded during the fitting process because of missingness.
se	whether to return standard errors.
verbose	whether progress indication should be printed to the console.
doFit	whether to fit the full model, or (if FALSE) return the preprocessed data and parameter objects, without fitting the model.
control	control parameters, see glmmTMBControl.
REML	whether to use REML estimation rather than maximum likelihood.
start	starting values, expressed as a list with possible components beta, betazi, betadisp (fixed-effect parameters for conditional, zero-inflation, dispersion models); b, bzi, bdisp (conditional modes for conditional, zero-inflation, and dispersion models); theta, thetazi, thetadisp (random-effect parameters, on the standard deviation/Cholesky scale, for conditional, z-i, and disp models); psi (extra family parameters, e.g., shape for Tweedie models).
map	a list specifying which parameter values should be fixed to a constant value rather than estimated. map should be a named list containing factors corresponding to a subset of the internal parameter names (see start parameter). Distinct factor values are fitted as separate parameter values, NA values are held fixed: e.g., map=list(beta=factor(c(1,2,3,NA))) would fit the first three fixed-effect parameters of the conditional model and fix the fourth parameter to its starting value. In general, users will probably want to use start to specify non-default starting values for fixed parameters. See MakeADFun for more details.
sparseX	a named logical vector containing (possibly) elements named "cond", "zi", "disp" to indicate whether fixed-effect model matrices for particular model components should be generated as sparse matrices, e.g. c(cond=TRUE). Default is all FALSE
priors	a data frame of priors, in a similar format to that accepted by the brms package; see priors

Details

- Binomial models with more than one trial (i.e., not binary/Bernoulli) can either be specified in the form prob ~ ..., weights = N, or in the more typical two-column matrix cbind(successes,failures)~... form.
- Behavior of REML=TRUE for Gaussian responses matches lme4::lmer. It may also be useful in some cases with non-Gaussian responses (Millar 2011). Simulations should be done first to verify.
- Because the df.residual method for glmmTMB currently counts the dispersion parameter, users should multiply this value by sqrt(nobs(fit) / (1+df.residual(fit))) when comparing with lm.
- Although models can be fitted without specifying a data argument, its use is strongly recommended; drawing model components from the global environment, or using df$var notation within model formulae, can lead to confusing (and sometimes hard-to-detect) errors.

- By default, vector-valued random effects are fitted with unstructured (general symmetric positive definite) variance-covariance matrices. Structured variance-covariance matrices can be specified in the form `struc(terms|group)`, where `struc` is one of
  - `diag` (diagonal, heterogeneous variance)
  - `ar1` (autoregressive order-1, homogeneous variance)
  - `cs` (compound symmetric, heterogeneous variance)
  - `ou` (* Ornstein-Uhlenbeck, homogeneous variance)
  - `exp` (* exponential autocorrelation)
  - `gau` (* Gaussian autocorrelation)
  - `mat` (* Matérn process correlation)
  - `toep` (* Toeplitz)
  - `rr` (reduced-rank/factor-analytic model)
  - `homdiag` (diagonal, homogeneous variance)
  - `propto` (* proportional to user-specified variance-covariance matrix)

  Structures marked with * are experimental/untested. See `vignette("covstruct", package = "glmmTMB")` for more information.
- For backward compatibility, the `family` argument can also be specified as a list comprising the name of the distribution and the link function (e.g. `list(family="binomial", link="logit")`). However, **this alternative is now deprecated**; it produces a warning and will be removed at some point in the future. Furthermore, certain capabilities such as Pearson residuals or predictions on the data scale will only be possible if components such as `variance` and `linkfun` are present, see `family`.
- Smooths taken from the `mgcv` package can be included in `glmmTMB` formulas using `s`; these terms will appear as additional components in both the fixed and the random-effects terms. This functionality is *experimental* for now. We recommend using `REML=TRUE`. See `s` for details of specifying smooths (and `smooth2random` and the appendix of Wood (2004) for technical details).

Note

For more information about the **glmmTMB** package, see Brooks et al. (2017) and the `vignette(package="glmmTMB")` collection. For the underlying **TMB** package that performs the model estimation, see Kristensen et al. (2016).

References

Brooks, M. E., Kristensen, K., van Benthem, K. J., Magnusson, A., Berg, C. W., Nielsen, A., Skaug, H. J., Mächler, M. and Bolker, B. M. (2017). glmmTMB balances speed and flexibility among packages for zero-inflated generalized linear mixed modeling. *The R Journal*, **9**(2), 378–400.

Kristensen, K., Nielsen, A., Berg, C. W., Skaug, H. and Bell, B. (2016). TMB: Automatic differentiation and Laplace approximation. *Journal of Statistical Software*, **70**, 1–21.

Millar, R. B. (2011). *Maximum Likelihood Estimation and Inference: With Examples in R, SAS and ADMB*. Wiley, New York. Wood, S. N. (2004) Stable and Efficient Multiple Smoothing Parameter Estimation for Generalized Additive Models. *Journal of the American Statistical Association* **99**(467): 673–86. doi:10.1198/016214504000000980

Examples

```
(m1 <- glmmTMB(count ~ mined + (1|site),
  zi=~mined,
  family=poisson, data=Salamanders))
summary(m1)
##' ## Zero-inflated negative binomial model
(m2 <- glmmTMB(count ~ spp + mined + (1|site),
  zi=~spp + mined,
  family=nbinom2, data=Salamanders))

## Hurdle Poisson model
(m3 <- glmmTMB(count ~ spp + mined + (1|site),
  zi=~spp + mined,
  family=truncated_poisson, data=Salamanders))

## Binomial model
data(cbpp, package="lme4")
(bovine <- glmmTMB(cbind(incidence, size-incidence) ~ period + (1|herd),
  family=binomial, data=cbpp))

## Dispersion model
sim1 <- function(nfac=40, nt=100, facsd=0.1, tsd=0.15, mu=0, residsd=1)
{
  dat <- expand.grid(fac=factor(letters[1:nfac]), t=1:nt)
  n <- nrow(dat)
  dat$REfac <- rnorm(nfac, sd=facsd)[dat$fac]
  dat$REt <- rnorm(nt, sd=tsd)[dat$t]
  dat$x <- rnorm(n, mean=mu, sd=residsd) + dat$REfac + dat$REt
  dat
}
set.seed(101)
d1 <- sim1(mu=100, residsd=10)
d2 <- sim1(mu=200, residsd=5)
d1$sd <- "ten"
d2$sd <- "five"
dat <- rbind(d1, d2)
m0 <- glmmTMB(x ~ sd + (1|t), dispformula=~sd, data=dat)
fixef(m0)$disp
c(log(5), log(10)-log(5)) # expected dispersion model coefficients


## Using 'map' to fix random-effects SD to 10
m1_map <- update(m1, map=list(theta=factor(NA)),
                 start=list(theta=log(10)))
VarCorr(m1_map)

## smooth terms
data("Nile")
ndat <- data.frame(time = c(time(Nile)), val = c(Nile))
sm1 <- glmmTMB(val ~ s(time), data = ndat,
               REML = TRUE, start = list(theta = 5))
plot(val ~ time, data = ndat)
```

```
lines(ndat$time, predict(sm1))

## reduced-rank model
m1_rr <- glmmTMB(abund ~ Species + rr(Species + 0|id, d = 1),
                              data = spider_long)
```

---

glmmTMBControl *Control parameters for glmmTMB optimization*

Description

Control parameters for glmmTMB optimization

Usage

```
glmmTMBControl(
  optCtrl = NULL,
  optArgs = list(),
  optimizer = nlminb,
  profile = FALSE,
  collect = FALSE,
  parallel = list(n = getOption("glmmTMB.cores", 1L), autopar =
    getOption("glmmTMB.autopar", NULL)),
  eigval_check = TRUE,
  zerodisp_val = log(.Machine$double.eps)/4,
  start_method = list(method = NULL, jitter.sd = 0),
  rank_check = c("adjust", "warning", "stop", "skip"),
  conv_check = c("warning", "skip")
)
```

Arguments

optCtrl	Passed as argument `control` to optimizer. Default value (if default `nlminb` optimizer is used): `list(iter.max=300, eval.max=400)`
optArgs	additional arguments to be passed to optimizer function (e.g.: `list(method="BFGS")` when `optimizer=optim`)
optimizer	Function to use in model fitting. See `Details` for required properties of this function.
profile	(logical) Experimental option to improve speed and robustness when a model has many fixed effects
collect	(logical) Experimental option to improve speed by recognizing duplicated observations.
parallel	(named list with an integer value `n` and a logical value `autopar`, e.g. `list(n=4L, autopar=TRUE)`) Set number of OpenMP threads to evaluate the negative log-likelihood in parallel, and determine whether to use auto-parallelization (see

openmp). The default is to evaluate models serially (i.e. single-threaded); users can set default values for an R session via `options(glmmTMB.cores=<value>, glmmTMB.autopar=<value>)`. An integer number of cores (only) can be passed instead of a list, in which case the default or previously set value of `autopar` will be used. At present reduced-rank models (i.e., a covariance structure using `rr(...)`) cannot be fitted in parallel unless `autopar=TRUE`; the number of threads will be automatically set to 1, with a warning if this overrides the user-specified value. To trace OpenMP settings, use `options(glmmTMB_openmp_debug = TRUE)`.

`eigval_check` Check eigenvalues of variance-covariance matrix? (This test may be very slow for models with large numbers of fixed-effect parameters.)

`zerodisp_val` value of the dispersion parameter when `dispformula=~0` is specified

`start_method` (list) Options to initialize the starting values when fitting models with reduced-rank (`rr`) covariance structures; `jitter.sd` adds variation to the starting values of latent variables when `method = "res"`.

`rank_check` Check whether all parameters in fixed-effects models are identifiable? This test may be slow for models with large numbers of fixed-effect parameters, therefore default value is 'warning'. Alternatives include 'skip' (no check), 'stop' (throw an error), and 'adjust' (drop redundant columns from the fixed-effect model matrix).

`conv_check` Do basic checks of convergence (check for non-positive definite Hessian and non-zero convergence code from optimizer). Default is 'warning'; 'skip' ignores these tests (not recommended for general use!)

Details

By default, glmmTMB uses the nonlinear optimizer nlminb for parameter estimation. Users may sometimes need to adjust optimizer settings in order to get models to converge. For instance, the warning 'iteration limit reached without convergence' may be fixed by increasing the number of iterations using (e.g.)

`glmmTMBControl(optCtrl=list(iter.max=1e3,eval.max=1e3))`.

Setting `profile=TRUE` allows `glmmTMB` to use some special properties of the optimization problem in order to speed up estimation in cases with many fixed effects.

Control parameters may depend on the model specification. The value of the controls is evaluated inside an R object that is derived from the output of the mkTMBStruc function. For example, to specify that `profile` should be enabled if the model has more than 5 fixed-effect parameters, specify

`profile=quote(length(parameters$beta)>=5)`

The `optimizer` argument can be any optimization (minimizing) function, provided that:

- the first three arguments, in order, are the starting values, objective function, and gradient function;
- the function also takes a `control` argument;
- the function returns a list with elements (at least) `par`, `objective`, `convergence` (0 if convergence is successful) and `message` (`glmmTMB` automatically handles output from `optim()`, by renaming the `value` component to `objective`)

Examples

```
## fit with default (nlminb) and alternative (optim/BFGS) optimizer
m1 <- glmmTMB(count~ mined, family=poisson, data=Salamanders)
m1B <- update(m1, control=glmmTMBControl(optimizer=optim,
               optArgs=list(method="BFGS")))
## estimates are *nearly* identical:
all.equal(fixef(m1), fixef(m1B))
```

---

isLMM.glmmTMB *support methods for parametric bootstrapping*

Description

see refit and isLMM for details

Usage

```
## S3 method for class 'glmmTMB'
isLMM(x, ...)

## S3 method for class 'glmmTMB'
refit(object, newresp, ...)
```

Arguments

x	a fitted glmmTMB object
...	additional arguments (for generic consistency; ignored)
object	a fitted glmmTMB object
newresp	a new response vector

Details

These methods are still somewhat experimental (check your results carefully!), but they should allow parametric bootstrapping. They work by copying and replacing the original response column in the data frame passed to glmmTMB, so they will only work properly if (1) the data frame is still available in the environment and (2) the response variable is specified as a single symbol (e.g. proportion or a two-column matrix constructed on the fly with cbind(). Untested with binomial models where the response is specified as a factor.

Examples

```
if (requireNamespace("lme4")) {
## Not run:
   fm1 <- glmmTMB(count~mined+(1|spp),
                  ziformula=~mined,
                  data=Salamanders,
                  family=nbinom1)
```

```
   ## single parametric bootstrap step: refit with data simulated from original model
   fm1R <- refit(fm1, simulate(fm1)[[1]])
   ## the bootMer function from lme4 provides a wrapper for doing multiple refits
   ##   with a specified summary function
   b1 <- lme4::bootMer(fm1, FUN=function(x) fixef(x)$zi, nsim=20, .progress="txt")
   if (requireNamespace("boot")) {
      boot.ci(b1,type="perc")
    }
   ## can run in parallel: may need to set up cluster explicitly,
   ## use clusterEvalQ() to load packages on workers
   if (requireNamespace("parallel")) {
      cl <- parallel::makeCluster(2)
      parallel::clusterEvalQ(cl, library("lme4"))
      parallel::clusterEvalQ(cl, library("glmmTMB"))
      b2 <- lme4::bootMer(fm1, FUN = function(x) fixef(x)$cond,
              nsim = 10, ncpus = 2, cl = cl, parallel = "snow")
   }

## End(Not run)
}
```

`map.theta.propto` *Set map values for theta to be fixed (NA) for propto*

Description

Set map values for theta to be fixed (NA) for propto

Usage

```
map.theta.propto(ReStruc, map)
```

Arguments

`ReStruc`	a random effects structure
`map`	a list of mapped elements

Value

the corresponding `theta` parameter vector

nbinom2 *Family functions for glmmTMB*

Description

Family functions for glmmTMB

Usage

```
nbinom2(link = "log")

nbinom1(link = "log")

nbinom12(link = "log")

compois(link = "log")

truncated_compois(link = "log")

genpois(link = "log")

truncated_genpois(link = "log")

truncated_poisson(link = "log")

truncated_nbinom2(link = "log")

truncated_nbinom1(link = "log")

beta_family(link = "logit")

betabinomial(link = "logit")

tweedie(link = "log")

skewnormal(link = "identity")

lognormal(link = "log")

ziGamma(link = "inverse")

t_family(link = "identity")

ordbeta(link = "logit")

bell(link = "log")
```

Arguments

`link`	(character) link function for the conditional mean ("log", "logit", "probit", "inverse", "cloglog", "identity", or "sqrt")

Details

If specified, the dispersion model uses a log link. Denoting the variance as V, the dispersion parameter as $\phi = \exp(\eta)$ (where η is the linear predictor from the dispersion model), and the predicted mean as μ:

gaussian (from base R): constant $V = \phi^2$

Gamma (from base R) phi is the shape parameter. $V = \mu\phi$

ziGamma a modified version of `Gamma` that skips checks for zero values, allowing it to be used to fit hurdle-Gamma models

nbinom2 Negative binomial distribution: quadratic parameterization (Hardin & Hilbe 2007). $V = \mu(1 + \mu/\phi) = \mu + \mu^2/\phi$.

nbinom1 Negative binomial distribution: linear parameterization (Hardin & Hilbe 2007). $V = \mu(1+\phi)$. *Note* that the phi parameter has opposite meanings in the `nbinom1` and `nbinom2` families. In `nbinom1` overdispersion increases with increasing `phi` (the Poisson limit is `phi=0`); in `nbinom2` overdispersion decreases with increasing `phi` (the Poisson limit is reached as `phi` goes to infinity).

nbinom12 Negative binomial distribution: mixed linear/quadratic, as in the `DESeq2` package or as described by Lindén and Mäntyniemi (2011). $V = \mu(1 + \phi + \mu/psi)$. (In Lindén and Mäntyniemi's parameterization, $\omega = \phi$ and $\theta = 1/\psi$.) If a dispersion model is specified, it applies only to the linear (`phi`) term.

truncated_nbinom2 Zero-truncated version of nbinom2: variance expression from Shonkwiler 2016. Simulation code (for this and the other truncated count distributions) is taken from C. Geyer's functions in the `aster` package; the algorithms are described in this vignette.

compois Conway-Maxwell Poisson distribution: parameterized with the exact mean (Huang 2017), which differs from the parameterization used in the **COMPoissonReg** package (Sellers & Shmueli 2010, Sellers & Lotze 2015). $V = \mu\phi$.

genpois Generalized Poisson distribution (Consul & Famoye 1992). $V = \mu\exp(\eta)$. (Note that Consul & Famoye (1992) define ϕ differently.) Our implementation is taken from the `HMMpa` package, based on Joe and Zhu (2005) and implemented by Vitali Witowski.

beta Beta distribution: parameterization of Ferrari and Cribari-Neto (2004) and the **betareg** package (Cribari-Neto and Zeileis 2010); $V = \mu(1-\mu)/(\phi+1)$

betabinomial Beta-binomial distribution: parameterized according to Morris (1997). $V = \mu(1-\mu)(n(\phi+n)/(\phi+1))$

tweedie Tweedie distribution: $V = \phi\mu^power$. The power parameter is restricted to the interval $1 < power < 2$, i.e. the compound Poisson-gamma distribution. Code taken from the `tweedie` package, written by Peter Dunn. The power parameter (designated `psi` in the list of parameters) uses the link function `qlogis(psi-1.0)`; thus one can fix the power parameter to a specified value using `start = list(psi = qlogis(fixed_power-1.0)), map = list(psi = factor(NA))`.

t_family Student-t distribution with adjustable scale and location parameters (also called a Pearson type VII distribution). The shape (degrees of freedom parameter) is fitted with a log link; it may be often be useful to fix the shape parameter using `start = list(psi = log(fixed_df))`, `map = list(psi = factor(NA))`.

ordbeta Ordered beta regression from Kubinec (2022); fits continuous (e.g. proportion) data in the *closed* interval [0,1]. Unlike the implementation in the `ordbeta` package, this family will not automatically scale the data. If your response variable is defined on the closed interval [a,b], transform it to [0,1] via `y_scaled <- (y-a)/(b-a)`.

lognormal Log-normal, parameterized by the mean and standard deviation *on the data scale*

skewnormal Skew-normal, parameterized by the mean, standard deviation, and shape (Azzalini & Capitanio, 2014); constant $V = \phi^2$

bell Bell distribution (see Castellares et al 2018).

Value

returns a list with (at least) components

`family`	length-1 character vector giving the family name
`link`	length-1 character vector specifying the link function
`variance`	a function of either 1 (mean) or 2 (mean and dispersion parameter) arguments giving a value proportional to the predicted variance (scaled by `sigma(.)`)

References

- Azzalini A & Capitanio A (2014). "The skew-normal and related families." Cambridge: Cambridge University Press.
- Castellares F, Ferrari SLP, & Lemonte AJ (2018) "On the Bell Distribution and Its Associated Regression Model for Count Data" Applied Mathematical Modelling 56: 172–85. doi:10.1016/j.apm.2017.12.014
- Consul PC & Famoye F (1992). "Generalized Poisson regression model." Communications in Statistics: Theory and Methods 21:89–109.
- Ferrari SLP, Cribari-Neto F (2004). "Beta Regression for Modelling Rates and Proportions." *J. Appl. Stat.* 31(7), 799-815.
- Hardin JW & Hilbe JM (2007). "Generalized linear models and extensions." Stata Press.
- Huang A (2017). "Mean-parametrized Conway–Maxwell–Poisson regression models for dispersed counts." *Statistical Modelling* 17(6), 1-22.
- Joe H & Zhu R (2005). "Generalized Poisson Distribution: The Property of Mixture of Poisson and Comparison with Negative Binomial Distribution." *Biometrical Journal* 47(2): 219–29. doi:10.1002/bimj.200410102.
- Lindén, A & Mäntyniemi S. (2011). "Using the Negative Binomial Distribution to Model Overdispersion in Ecological Count Data." *Ecology* 92 (7): 1414–21. doi:10.1890/101831.1.
- Morris W (1997). "Disentangling Effects of Induced Plant Defenses and Food Quantity on Herbivores by Fitting Nonlinear Models." *American Naturalist* 150:299-327.
- Kubinec R (2022). "Ordered Beta Regression: A Parsimonious, Well-Fitting Model for Continuous Data with Lower and Upper Bounds." *Political Analysis*. doi:10.1017/pan.2022.20.

- Sellers K & Lotze T (2015). "COMPoissonReg: Conway-Maxwell Poisson (COM-Poisson) Regression". R package version 0.3.5. https://CRAN.R-project.org/package=COMPoissonReg
- Sellers K & Shmueli G (2010) "A Flexible Regression Model for Count Data." *Annals of Applied Statistics* 4(2), 943–61. doi:10.1214/09AOAS306.
- Shonkwiler, J. S. (2016). "Variance of the truncated negative binomial distribution." *Journal of Econometrics* 195(2), 209–210. doi:10.1016/j.jeconom.2016.09.002.

`numFactor` *Factor with numeric interpretable levels.*

Description

Create a factor with numeric interpretable factor levels.

Usage

```
numFactor(x, ...)

parseNumLevels(levels)
```

Arguments

`x`	Vector, matrix or data.frame that constitute the coordinates.
`...`	Additional vectors, matrices or data.frames that constitute the coordinates.
`levels`	Character vector to parse into numeric values.

Details

Some `glmmTMB` covariance structures require extra information, such as temporal or spatial coordinates. `numFactor` allows to associate such extra information as part of a factor via the factor levels. The original numeric coordinates are recoverable without loss of precision using the function `parseNumLevels`. Factor levels are sorted coordinate wise from left to right: first coordinate is fastest running.

Value

Factor with specialized coding of levels.

Examples

```
## 1D example
numFactor(sample(1:5,20,TRUE))
## 2D example
coords <- cbind( sample(1:5,20,TRUE), sample(1:5,20,TRUE) )
(f <- numFactor(coords))
parseNumLevels(levels(f)) ## Sorted
## Used as part of a model.matrix
```

```
model.matrix( ~f )
## parseNumLevels( colnames(model.matrix( ~f )) )
## Error: 'Failed to parse numeric levels: (Intercept)'
parseNumLevels( colnames(model.matrix( ~ f-1 )) )
```

---

omp_check *Check OpenMP status*

Description

Checks whether OpenMP has been successfully enabled for this installation of the package. (Use the `parallel` argument to `glmmTMBControl`, or set `options(glmmTMB.cores=[value])`, to specify that computations should be done in parallel.) To further trace OpenMP settings, use `options(glmmTMB_openmp_debug = TRUE)`.

Usage

```
omp_check()
```

Value

`TRUE` or `FALSE` depending on availability of OpenMP,

See Also

`benchmark`, `glmmTMBControl`

Owls *Begging by Owl Nestlings*

Description

Begging by owl nestlings

Usage

```
data(Owls)
```

Format

The `Owls` data set is a data frame with 599 observations on the following variables:

`Nest` a factor describing individual nest locations

`FoodTreatment` (factor) food treatment: `Deprived` or `Satiated`

`SexParent` (factor) sex of provisioning parent: `Female` or `Male`

`ArrivalTime` a numeric vector

`SiblingNegotiation` a numeric vector

`BroodSize` brood size

`NegPerChick` number of negotations per chick

Note

Access to data kindly provided by Alain Zuur

Source

Roulin, A. and L. Bersier (2007) Nestling barn owls beg more intensely in the presence of their mother than in the presence of their father. *Animal Behaviour* **74** 1099–1106. doi:10.1016/j.anbehav.2007.01.027; http://www.highstat.com/Books/Book2/ZuurDataMixedModelling.zip

References

Zuur, A. F., E. N. Ieno, N. J. Walker, A. A. Saveliev, and G. M. Smith (2009) *Mixed Effects Models and Extensions in Ecology with R*; Springer.

Examples

```
data(Owls, package = "glmmTMB")
require("lattice")
bwplot(reorder(Nest,NegPerChick) ~ NegPerChick | FoodTreatment:SexParent,
       data=Owls)
dotplot(reorder(Nest,NegPerChick) ~ NegPerChick| FoodTreatment:SexParent,
        data=Owls)
## Not run:
## Fit negative binomial model with "constant" Zero Inflation :
owls_nb1 <- glmmTMB(SiblingNegotiation ~ FoodTreatment*SexParent +
                                    (1|Nest)+offset(log(BroodSize)),
              family = nbinom1(), zi = ~1, data=Owls)
owls_nb1_bs <- update(owls_nb1,
                      . ~ . - offset(log(BroodSize)) + log(BroodSize))
fixef(owls_nb1_bs)

## End(Not run)
```

`predict.glmmTMB` *prediction*

Description

prediction

Usage

```
## S3 method for class 'glmmTMB'
predict(
  object,
  newdata = NULL,
  newparams = NULL,
  se.fit = FALSE,
```

```
  cov.fit = FALSE,
  re.form = NULL,
  allow.new.levels = FALSE,
  type = c("link", "response", "conditional", "zprob", "zlink", "disp", "latent"),
  zitype = NULL,
  na.action = na.pass,
  fast = NULL,
  debug = FALSE,
  ...
)
```

Arguments

object	a glmmTMB object
newdata	new data for prediction
newparams	new parameters for prediction
se.fit	return the standard errors of the predicted values?
cov.fit	return the covariance matrix of the predicted values?
re.form	NULL to specify individual-level predictions; ~0 or NA to specify population-level predictions (i.e., setting all random effects to zero)
allow.new.levels	allow previously unobserved levels in random-effects variables? see details.
type	Denoting mu as the mean of the conditional distribution and p as the zero-inflation probability, the possible choices are: **"link"** the linear predictor of the conditional model, or equivalently the conditional mean on the scale of the link function (this equivalence does not hold for truncated distributions, where the link-scaled value is not adjusted for the effect of truncation on the mean; to get the corrected value of the conditional mean on the linear predictor scale, use family(m)$linkfun(predict(m, type = "conditional"))) **"response"** expected value; this is $mu * (1 - p)$ for zero-inflated models and mu otherwise **"conditional"** mean of the conditional response; mu for all models (i.e., synonymous with "response" in the absence of zero-inflation **"zprob"** the probability of a structural zero (returns 0 for non-zero-inflated models) **"zlink"** predicted zero-inflation probability on the scale of the logit link function (returns -Inf for non-zero-inflated models) **"disp"** dispersion parameter, however it is defined for that particular family (as described in sigma.glmmTMB) **"latent"** return latent variables
zitype	deprecated: formerly used to specify type of zero-inflation probability. Now synonymous with type
na.action	how to handle missing values in newdata (see na.action); the default (na.pass) is to predict NA

fast	predict without expanding memory (default is TRUE if newdata and newparams are NULL and population-level prediction is not being done)
debug	(logical) return the TMBStruc object that will be used internally for debugging?
...	unused - for method compatibility

Details

- To compute population-level predictions for a given grouping variable (i.e., setting all random effects for that grouping variable to zero), set the grouping variable values to NA. Finer-scale control of conditioning (e.g. allowing variation among groups in intercepts but not slopes when predicting from a random-slopes model) is not currently possible.
- Prediction of new random effect levels is possible as long as the model specification (fixed effects and parameters) is kept constant. However, to ensure intentional usage, a warning is triggered if allow.new.levels=FALSE (the default).
- Prediction using "data-dependent bases" (variables whose scaling or transformation depends on the original data, e.g. poly, ns, or poly) should work properly; however, users are advised to check results extra-carefully when using such variables. Models with different versions of the same data-dependent basis type in different components (e.g. formula= y ~ poly(x,3), dispformula= ~poly(x,2)) will probably *not* produce correct predictions.

Examples

```
data(sleepstudy,package="lme4")
g0 <- glmmTMB(Reaction~Days+(Days|Subject),sleepstudy)
predict(g0, sleepstudy)
## Predict new Subject
nd <- sleepstudy[1,]
nd$Subject <- "new"
predict(g0, newdata=nd, allow.new.levels=TRUE)
## population-level prediction
nd_pop <- data.frame(Days=unique(sleepstudy$Days),
                     Subject=NA)
predict(g0, newdata=nd_pop)
## return latent variables (BLUPs/conditional modes/etc. ) with standard errors
##  (actually conditional standard deviations)
predict(g0, type = "latent", se.fit = TRUE)
```

---

print.VarCorr.glmmTMB *Printing The Variance and Correlation Parameters of a* glmmTMB

Description

Printing The Variance and Correlation Parameters of a glmmTMB

Usage

```
## S3 method for class 'VarCorr.glmmTMB'
print(
  x,
  digits = max(3, getOption("digits") - 2),
  comp = "Std.Dev.",
  formatter = format,
  ...
)
```

Arguments

x	a result of VarCorr(<glmmTMB>).
digits	number of significant digits to use.
comp	a string specifying the component to format and print.
formatter	a function.
...	optional further arguments, passed the next print method.

---

priors *use of priors in glmmTMB*

Description

(EXPERIMENTAL/subject to change)

Details

glmmTMB can accept prior specifications, for doing maximum *a posteriori* (MAP) estimation (or Hamiltonian MC with the tmbstan package), or (outside of a Bayesian framework) for the purposes of regularizing parameter estimates

The priors argument to glmmTMB must (if not NULL) be a data frame with columns

prior character; the prior specification, e.g. "normal(0,2)"

class the name of the underlying parameter vector on which to impose the prior ("fixef", "fixef_zi", "fixef_disp", "ranef", "ranef_zi", "psi")

coef (optional) a string (if present) specifying the particular elements of the parameter vector to apply the prior to. coef should specify an integer parameter index, a column name from the fixed effect model matrix or a grouping variable for a random effect (the behaviour is currently undefined if there is more one than random effect term with the same grouping variable in a model ...); one can also append "_cor" or "_sd" to a random-effects class specification to denote the correlation parameters, or all of the standard deviation parameters, corresponding to a particular random effect term. If the class element is missing, or a particular element is blank, then all of the elements of the specified parameter vector use independent priors with the given specification. The exception is for the fixed-effect parameter vectors ("fixef", "fixef_zi", "fixef_disp"), where the intercept (if present) is not included; the prior on the intercept must be set explicitly.

'The available prior distributions are:

- "normal" (mean/sd parameterization)
- "t" (mean/sd/df)
- "cauchy" (location/scale)
- "gamma" (mean/shape); applied on the SD (*not* the log-SD) scale
- "lkj" (correlation) [WARNING, maybe buggy at present!]

The first three are typically used for fixed effect parameters; the fourth for standard deviation parameters; and the last for correlation structures. See the "priors" vignette for examples and further information.

Examples

```
data("sleepstudy", package = "lme4")
prior1 <- data.frame(prior = c("normal(250,3)","t(0,3,3)","gamma(10,1)"),
                     class = c("fixef", "fixef", "ranef_sd"),
                     coef = c("(Intercept)", "Days", "Subject"))
g1 <- glmmTMB(Reaction ~ 1 + Days + (1 + Days |Subject), sleepstudy)
update(g1, priors = prior1)
prior2 <- data.frame(prior = c("t(0,3,3)","gamma(10,1)"),
                     class = c("fixef", "ranef_sd"),
                     coef = c("", "Subject"))
update(g1, priors = prior2)
## no prior is set for the intercept in this case - see Details above
prior3 <- data.frame(prior = "t(0, 3, 3)",
                     class = "fixef")
update(g1, priors = prior3)
```

`profile.glmmTMB` *Compute likelihood profiles for a fitted model*

Description

Compute likelihood profiles for a fitted model

Usage

```
## S3 method for class 'glmmTMB'
profile(
  fitted,
  parm = NULL,
  level_max = 0.99,
  npts = 8,
  stepfac = 1/4,
  stderr = NULL,
  trace = FALSE,
```

```
  parallel = c("no", "multicore", "snow"),
  ncpus = getOption("profile.ncpus", 1L),
  cl = NULL,
  ...
)

## S3 method for class 'profile.glmmTMB'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

`fitted`	a fitted `glmmTMB` object
`parm`	which parameters to profile, specified • by index (position) • by name (matching the row/column names of `vcov(object,full=TRUE)`) • as `"theta_"` (random-effects variance-covariance parameters) or `"beta_"` (conditional and zero-inflation parameters)
`level_max`	maximum confidence interval target for profile
`npts`	target number of points in (each half of) the profile (*approximate*)
`stepfac`	initial step factor (fraction of estimated standard deviation)
`stderr`	standard errors to use as a scaling factor when picking step sizes to compute the profile; by default (if `stderr` is `NULL`, or `NA` for a particular element), uses the estimated (Wald) standard errors of the parameters
`trace`	print tracing information? If `trace=FALSE` or 0, no tracing; if `trace=1`, print names of parameters currently being profiled; if `trace>1`, turn on tracing for the underlying `tmbprofile` function
`parallel`	method (if any) for parallel computation
`ncpus`	number of CPUs/cores to use for parallel computation
`cl`	cluster to use for parallel computation
`...`	additional arguments passed to `tmbprofile`
`object`	a fitted profile (`profile.glmmTMB`) object
`level`	confidence level

Details

Fits natural splines separately to the points from each half of the profile for each specified parameter (i.e., values above and below the MLE), then finds the inverse functions to estimate the endpoints of the confidence interval

Value

An object of class `profile.glmmTMB`, which is also a data frame, with columns `.par` (parameter being profiled), `.focal` (value of focal parameter), value (negative log-likelihood).

Examples

```
## Not run:
m1 <- glmmTMB(count~ mined + (1|site),
       zi=~mined, family=poisson, data=Salamanders)
salamander_prof1 <- profile(m1, parallel="multicore",
                            ncpus=2, trace=1)
## testing
salamander_prof1 <- profile(m1, trace=1,parm=1)
salamander_prof1M <- profile(m1, trace=1,parm=1, npts = 4)
salamander_prof2 <- profile(m1, parm="theta_")


## End(Not run)
salamander_prof1 <- readRDS(system.file("example_files","salamander_prof1.rds",package="glmmTMB"))
if (require("ggplot2")) {
    ggplot(salamander_prof1,aes(.focal,sqrt(value))) +
        geom_point() + geom_line()+
        facet_wrap(~.par,scale="free_x")+
    geom_hline(yintercept=1.96,linetype=2)
}
salamander_prof1 <- readRDS(system.file("example_files","salamander_prof1.rds",package="glmmTMB"))
confint(salamander_prof1)
confint(salamander_prof1,level=0.99)
```

`ranef.glmmTMB` *Extract Random Effects*

Description

Extract random effects from a fitted `glmmTMB` model, both for the conditional model and zero inflation.

Usage

```
## S3 method for class 'glmmTMB'
ranef(object, condVar = TRUE, ...)

## S3 method for class 'ranef.glmmTMB'
as.data.frame(x, ...)

## S3 method for class 'glmmTMB'
coef(object, condVar = FALSE, ...)
```

Arguments

`object`	a `glmmTMB` model.
`condVar`	whether to include conditional variances in result.

... some methods for this generic function require additional arguments (they are unused here and will trigger an error)

x a `ranef.glmmTMB` object (i.e., the result of running `ranef` on a fitted `glmmTMB` model)

Value

- For `ranef`, an object of class `ranef.glmmTMB` with two components:

  cond a list of data frames, containing random effects for the conditional model.

  zi a list of data frames, containing random effects for the zero inflation.

  disp a list of data frames, containing random effects for the dispersion model.

  If `condVar=TRUE`, the individual list elements within the `cond`, `zi`, and `disp` components (corresponding to individual random effects terms) will have associated `condVar` attributes giving the conditional variances of the random effects values. These are in the form of three-dimensional arrays: see `ranef.merMod` for details. The only difference between the packages is that the attributes are called 'postVar' in **lme4**, vs. 'condVar' in **glmmTMB**.

- For `coef.glmmTMB`: a similar list, but containing the overall coefficient value for each level, i.e., the sum of the fixed effect estimate and the random effect value for that level. *Conditional variances are not yet available as an option for* `coef.glmmTMB`.

- For `as.data.frame`: a data frame with components

  component part of the model to which the random effects apply (conditional or zero-inflation)

  grpvar grouping variable

  term random-effects term (e.g., intercept or slope)

  grp group, or level of the grouping variable

  condval value of the conditional mode

  condsd conditional standard deviation

Note

When a model has no zero inflation, the `ranef` and `coef` print methods simplify the structure shown, by default. To show the full list structure, use `print(ranef(model),simplify=FALSE)` or the analogous code for `coef`. In all cases, the full list structure is used to access the data frames, see example.

See Also

`fixef.glmmTMB`.

Examples

```
if (requireNamespace("lme4")) {
   data(sleepstudy, package="lme4")
   model <- glmmTMB(Reaction ~ Days + (1|Subject), sleepstudy)
   rr <- ranef(model)
   print(rr, simplify=FALSE)
   ## extract Subject conditional modes for conditional model
   rr$cond$Subject
```

```
    as.data.frame(rr)
}
```

---

reinstalling *Reinstalling binary dependencies*

Description

The glmmTMB package depends on several upstream packages, which it uses in a way that depends heavily on their internal (binary) structure. Sometimes, therefore, installing an update to one of these packages will require that you re-install a *binary-compatible* version of glmmTMB, i.e. a version that has been compiled with the updated version of the upstream package.

- If you have development tools (compilers etc.) installed, you should be able to re-install a binary-compatible version of the package by running `install.packages("glmmTMB", type="source")`. If you want to install the development version of glmmTMB instead, you can use `remotes::install_github("glmmTMB/` (On Windows, you can install development tools following the instructions at https://cran.r-project.org/bin/windows/Rtools/; on MacOS, see https://mac.r-project.org/tools/.)
- If you do *not* have development tools and can't/don't want to install them (and so can't install packages with compiled code from source), you have two choices:
  - revert the upstream package(s) to their previous binary version. For example, using the checkpoint package:

    ```
    ## load (installing if necessary) the checkpoint package
    while (!require("checkpoint")) install.packages("checkpoint")
    ## retrieve build date of installed version of glmmTMB
    bd <- as.character(asDateBuilt(
          packageDescription("glmmTMB",fields="Built")))
    oldrepo <- getOption("repos")
    use_mran_snapshot(bd) ## was setSnapshot() pre-checkpoint v1.0.0
    install.packages("TMB")
    options(repos=oldrepo) ## restore original repo
    ```

    A similar recipe (substituting Matrix for TMB and TMB for glmmTMB) can be used if you get warnings about an incompatibility between TMB and Matrix.
  - hope that the glmmTMB maintainers have posted a binary version of the package that works with your system; try installing it via `install.packages("glmmTMB",repos="https://glmmTMB.github.` If this doesn't work, please file an issue (with full details about your operating system and R version) asking the maintainers to build and post an appropriate binary version of the package.

`residuals.glmmTMB` *Compute residuals for a glmmTMB object*

Description

Compute residuals for a glmmTMB object

Usage

```
## S3 method for class 'glmmTMB'
residuals(
  object,
  type = c("response", "pearson", "working", "deviance", "dunn-smyth"),
  re.form = NULL,
  ...
)

## S3 method for class 'glmmTMB'
deviance(object, ...)
```

Arguments

`object`	a "glmmTMB" object
`type`	(character) residual type
`re.form`	`NULL` to specify individual-level predictions; `~0` or `NA` to specify population-level predictions (i.e., setting all random effects to zero)
`...`	for method compatibility (unused arguments will throw an error)

Details

- Residuals are computed based on predictions of type "response", i.e. equal to the conditional mean for non-zero-inflated models and to `mu*(1-p)` for zero-inflated models
- Computing deviance residuals depends on the implementation of the `dev.resids` function from the object's `family` component; at present this returns `NA` for most "exotic" families (i.e. deviance residuals are currently only implemented for families built into base R plus `nbinom1`, `nbinom2`). Deviance residuals are based on the conditional distributions only, i.e. ignoring zero-inflation components.
- Deviance is computed as the sum of squared deviance residuals, so is available only for the families listed in the bullet point above. See deviance.merMod for more details on the definition of the deviance for GLMMs.

`Salamanders` *Repeated counts of salamanders in streams*

Description

A data set containing counts of salamanders with site covariates and sampling covariates. Each of 23 sites was sampled 4 times. When using this data set, please cite Price et al. (2016) as well as the Dryad data package (Price et al. 2015).

Usage

```
data(Salamanders)
```

Format

A data frame with 644 observations on the following 10 variables:

site name of a location where repeated samples were taken

mined factor indicating whether the site was affected by mountain top removal coal mining

cover amount of cover objects in the stream (scaled)

sample repeated sample

DOP Days since precipitation (scaled)

Wtemp water temperature (scaled)

DOY day of year (scaled)

spp abbreviated species name, possibly also life stage

count number of salamanders observed

References

Price SJ, Muncy BL, Bonner SJ, Drayer AN, Barton CD (2016) Effects of mountaintop removal mining and valley filling on the occupancy and abundance of stream salamanders. *Journal of Applied Ecology* **53** 459–468. doi:10.1111/13652664.12585

Price SJ, Muncy BL, Bonner SJ, Drayer AN, Barton CD (2015) Data from: Effects of mountaintop removal mining and valley filling on the occupancy and abundance of stream salamanders. *Dryad Digital Repository*. doi:10.5061/dryad.5m8f6

Examples

```
require("glmmTMB")
data(Salamanders)

zipm3 = glmmTMB(count~spp * mined + (1|site), zi=~spp * mined, Salamanders, family="poisson")
```

set_simcodes *helper function to modify simulation settings for random effects*

Description

This modifies the TMB object *in place* (beware!) Ultimately this will allow `terms` to be a vector of term names, with a matching `val` vector to specify the behaviour for each term

Usage

```
set_simcodes(g, val = "zero", terms = "ALL")
```

Arguments

`g`	a TMB object
`val`	a legal setting for sim codes ("zero", "random", or "fix")
`terms`	which terms to apply this to

sigma.glmmTMB *Extract residual standard deviation or dispersion parameter*

Description

For Gaussian models, `sigma` returns the value of the residual standard deviation; for other families, it returns the dispersion parameter, *however it is defined for that particular family*. See details for each family below.

Usage

```
## S3 method for class 'glmmTMB'
sigma(object, ...)
```

Arguments

`object`	a "glmmTMB" fitted object
`...`	(ignored; for method compatibility)

Details

The value returned varies by family:

gaussian returns the *maximum likelihood* estimate of the standard deviation (i.e., smaller than the results of `sigma(lm(...))` by a factor of (n-1)/n)

nbinom1 returns a dispersion parameter (usually denoted α as in Hardin and Hilbe (2007)): such that the variance equals $\mu(1+\alpha)$.

nbinom2 returns a dispersion parameter (usually denoted θ or k); in contrast to most other families, larger θ corresponds to a *lower* variance which is $\mu(1+\mu/\theta)$.

Gamma Internally, glmmTMB fits Gamma responses by fitting a mean and a shape parameter; sigma is estimated as (1/sqrt(shape)), which will typically be close (but not identical to) that estimated by `stats:::sigma.default`, which uses sqrt(deviance/df.residual)

beta returns the value of ϕ, where the conditional variance is $\mu(1-\mu)/(1+\phi)$ (i.e., increasing ϕ decreases the variance.) This parameterization follows Ferrari and Cribari-Neto (2004) (and the `betareg` package):

betabinomial This family uses the same parameterization (governing the Beta distribution that underlies the binomial probabilities) as `beta`.

genpois returns the index of dispersion ϕ^2, where the variance is $\mu\phi^2$ (Consul & Famoye 1992)

compois returns the value of $1/\nu$; when $\nu = 1$, compois is equivalent to the Poisson distribution. There is no closed form equation for the variance, but it is approximately underdispersed when $1/\nu < 1$ and approximately overdispersed when $1/\nu > 1$. In this implementation, μ is exactly equal to the mean (Huang 2017), which differs from the COMPoissonReg package (Sellers & Lotze 2015).

tweedie returns the value of ϕ, where the variance is $\phi\mu^p$. The value of p can be extracted using `family_params`

ordbeta see details for `beta`

The most commonly used GLM families (`binomial`, `poisson`) have fixed dispersion parameters which are internally ignored.

References

- Consul PC, and Famoye F (1992). "Generalized Poisson regression model. Communications in Statistics: Theory and Methods" 21:89–109.
- Ferrari SLP, Cribari-Neto F (2004). "Beta Regression for Modelling Rates and Proportions." *J. Appl. Stat.* 31(7), 799-815.
- Hardin JW & Hilbe JM (2007). "Generalized linear models and extensions." Stata press.
- Huang A (2017). "Mean-parametrized Conway–Maxwell–Poisson regression models for dispersed counts. " *Statistical Modelling* 17(6), 1-22.
- Sellers K & Lotze T (2015). "COMPoissonReg: Conway-Maxwell Poisson (COM-Poisson) Regression". R package version 0.3.5. https://CRAN.R-project.org/package=COMPoissonReg

simulate.glmmTMB *Simulate from a glmmTMB fitted model*

Description

Simulate from a glmmTMB fitted model

Usage

```
## S3 method for class 'glmmTMB'
simulate(object, nsim = 1, seed = NULL, re.form = NULL, ...)
```

Arguments

object	glmmTMB fitted model
nsim	number of response lists to simulate. Defaults to 1.
seed	random number seed
re.form	(Not yet implemented)
...	extra arguments

Details

Random effects are also simulated from their estimated distribution. Currently, it is not possible to condition on estimated random effects.

Value

returns a list of vectors. The list has length nsim. Each simulated vector of observations is the same size as the vector of response variables in the original data set. In the binomial family case each simulation is a two-column matrix with success/failure.

simulate_new *Simulate from covariate/metadata in the absence of a real data set (EXPERIMENTAL)*

Description

See vignette("sim", package = "glmmTMB") for more details and examples, and vignette("covstruct", package = "glmmTMB") for more information on the parameterization of different covariance structures.

Usage

```
simulate_new(
  object,
  nsim = 1,
  seed = NULL,
  family = gaussian,
  newdata,
  newparams,
  ...,
  return_val = c("sim", "pars", "object")
)
```

Arguments

object	a *one-sided* model formula (e.g. ~ a + b + c (peculiar naming is for consistency with the generic function, which typically takes a fitted model object)
nsim	number of simulations
seed	random-number seed
family	a family function, a character string naming a family function, or the result of a call to a family function (variance/link function) information. See family for a generic discussion of families or family_glmmTMB for details of glmmTMB-specific families.
newdata	a data frame containing all variables listed in the formula, *including* the response variable (which needs to fall within the domain of the conditional distribution, and should probably not be all zeros, but whose value is otherwise irrelevant)
newparams	a list of parameters containing sub-vectors (beta, betazi, betadisp, theta, etc.) to be used in the model. If b is specified in this list, then the conditional modes/BLUPs will be set to these values; otherwise they will be drawn from the appropriate Normal distribution
...	other arguments to glmmTMB (e.g. family)
return_val	what information to return: "sim" (the default) returns a list of vectors of simulated outcomes; "pars" returns the default parameter vector (this variant does not require newparams to be specified, and is useful for figuring out the appropriate dimensions of the different parameter vectors); "object" returns a fake glmmTMB object (useful, e.g., for retrieving the Z matrix (getME(simulate_new(...), "Z")) or covariance matrices (VarCorr(simulate_new(...))) implied by a particular set of input data and parameter values)

Examples

```
## use Salamanders data for structure/covariates
sim_count <- simulate_new(~ mined + (1|site),
            newdata = Salamanders,
            zi = ~ mined,
            family = nbinom2,
            newparams = list(beta = c(2, 1),
                        betazi = c(-0.5, 0.5), ## logit-linear model for zi
```

```
                        betadisp = log(2), ## log(NB dispersion)
                        theta = log(1)) ## log(among-site SD)
)
sim_obj <- simulate_new(~ mined + (1|site),
            return_val = "object",
             newdata = Salamanders,
             zi = ~ mined,
             family = nbinom2,
             newparams = list(beta = c(2, 1),
                        betazi = c(-0.5, 0.5), ## logit-linear model for zi
                        betad = log(2), ## log(NB dispersion)
                        theta = log(1)) ## log(among-site SD)
)
data("sleepstudy", package = "lme4")
sim_obj <- simulate_new(~ 1 + (1|Subject) + ar1(0 + factor(Days)|Subject),
             return_val = "pars",
             newdata = sleepstudy,
             family = gaussian,
             newparams = list(beta = c(280, 1),
                        betad = log(2), ## log(SD)
                        theta = log(c(2, 2, 1))),
)
```

---

spider_long *Spider data from CANOCO, long format*

Description

data from spider2 directory, CANOCO FORTRAN package, with trait variables added; taken from the mvabund package and converted to long form. Variables:

- soil.dry
- bare.sand
- fallen.leaves
- moss
- herb.layer
- reflection
- id
- Species
- abund

Usage

```
spider_long
```

Format

An object of class `data.frame` with 336 rows and 9 columns.

References

- ter Braak, C. J. F. and Smilauer, P. (1998) CANOCO reference manual and user's guide to CANOCO for Windows: software for canonical community ordination (version 4). Microcomputer Power, New York, New York, USA
- van der Aart, P. J. M., and Smeenk-Enserink, N. (1975) Correlations between distributions of hunting spiders (Lycosidae, Ctenidae) and environmental characteristics in a dune area. Netherlands Journal of Zoology 25, 1-45.

---

terms.glmmTMB *Methods for extracting developer-level information from* `glmmTMB` *models*

Description

Methods for extracting developer-level information from `glmmTMB` models

Usage

```
## S3 method for class 'glmmTMB'
terms(x, component = "cond", part = "fixed", ...)

## S3 method for class 'glmmTMB'
model.matrix(
  object,
  component = "cond",
  part = "fixed",
  include_rankdef = FALSE,
  ...
)
```

Arguments

`x`	a fitted `glmmTMB` object
`component`	model component ("cond", "zi", or "disp"; not all models contain all components)
`part`	whether to return results for the fixed or random effect part of the model (at present only `part="fixed"` is implemented for most methods)
`...`	additional arguments (ignored or passed to `model.frame`)
`object`	a fitted `glmmTMB` object
`include_rankdef`	include all columns of a rank-deficient model matrix?

up2date *conditionally update glmmTMB object fitted with an old TMB version*

Description

conditionally update glmmTMB object fitted with an old TMB version

Load data from system file, updating glmmTMB objects

Usage

```
up2date(oldfit, update_gauss_disp = FALSE)

gt_load(fn, verbose = FALSE, mustWork = FALSE, ...)
```

Arguments

oldfit	a fitted glmmTMB object
update_gauss_disp	update betadisp from variance to SD parameterization?
fn	partial path to system file (e.g. test_data/foo.rda)
verbose	print names of updated objects?
mustWork	fail if file not found?
...	values passed through to up2date

vcov.glmmTMB *Calculate Variance-Covariance Matrix for a Fitted glmmTMB model*

Description

Calculate Variance-Covariance Matrix for a Fitted glmmTMB model

Usage

```
## S3 method for class 'glmmTMB'
vcov(object, full = FALSE, include_nonest = TRUE, ...)
```

Arguments

object	a "glmmTMB" fit
full	return a full variance-covariance matrix?
include_nonest	include variables that are mapped *or* dropped due to rank-deficiency? (these will be given variances and covariances of NA)
...	ignored, for method compatibility

Value

By default (`full==FALSE`), a list of separate variance-covariance matrices for each model component (conditional, zero-inflation, dispersion). If `full==TRUE`, a single square variance-covariance matrix for *all* top-level model parameters (conditional, dispersion, and variance-covariance parameters)

`weights.glmmTMB` *Extract weights from a glmmTMB object*

Description

Extract weights from a glmmTMB object

Usage

```
## S3 method for class 'glmmTMB'
weights(object, type = "prior", ...)
```

Arguments

`object`	a fitted `glmmTMB` object
`type`	weights type
`...`	additional arguments (not used; for methods compatibility)

Details

At present only explicitly specified *prior weights* (i.e., weights specified in the `weights` argument) can be extracted from a fitted model.

- Unlike other GLM-type models such as `glm` or `glmer`, `weights()` does not currently return the total number of trials when binomial responses are specified as a two-column matrix.
- Since `glmmTMB` does not fit models via iteratively weighted least squares, `working weights` (see `weights.glm`) are unavailable.

Index