Package ‘buildmer’

October 20, 2022

Title Stepwise Elimination and Term Reordering for Mixed-Effects Regression

Version 2.7

Description Finds the largest possible regression model that will still converge for various types of regression analyses (including mixed models and generalized additive models) and then optionally performs stepwise elimination similar to the forward and backward effect-selection methods in SAS, based on the change in log-likelihood or its significance, Akaike's Information Criterion, the Bayesian Information Criterion, the explained deviance, or the F-test of the change in $R^2$.

Depends R (>= 3.2)

Imports graphics, lme4, methods, mgcv, nlme, stats, utils

Suggests GLMMadaptive, MASS, gamm4, glmerTree, glmmTMB, knitr,
        lmerTest, mnet, ordinal, parallel, partykit, pbkrtest,
        rmarkdown, testthat

License FreeBSD

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

BugReports https://github.com/cvoeten/buildmer/issues

VignetteBuilder knitr

NeedsCompilation no

Author Cesko C. Voeten [aut, cre] (<https://orcid.org/0000-0003-4687-9973>)

Maintainer Cesko C. Voeten <cvoeten@gmail.com>

Repository CRAN

Date/Publication 2022-10-20 15:45:07 UTC

R topics documented:

  buildmer-package .................................................. 2
  add.terms .......................................................... 3
construct and fit as complete a model as possible and perform step-wise elimination

Description

The buildmer package consists of a number of functions, each designed to fit specific types of models (e.g. buildmer for mixed-effects regression, buildgam for generalized additive models, buildmertree for mixed-effects-regression trees, and so forth). The common parameters shared by all (or most of) these functions are documented here. If you are looking for a more general description of what the various build... functions do, see under ‘Details’. For function-specific details, see the documentation for each individual function.
add.terms

Add terms to a formula

Description

Add terms to a formula

Usage

add.terms(formula, add)

Arguments

formula  The formula to add terms to.
add       A vector of terms to add. To add terms nested in random-effect groups, use
           ‘(term|group)’ syntax if you want to add an independent random effect (e.g.
           ‘(olderterm|group) + (term|group)’), or use ‘term|group’ syntax if you want to
           add a dependent random effect to a pre-existing term group (if no such group
           exists, it will be created at the end of the formula).

Value

The updated formula.

Examples

library(buildmer)
form <- Reaction ~ Days + (1|Subject)
add.terms(form, 'Days|Subject')
add.terms(form, '(0+Days|Subject)')
add.terms(form, c('many', 'more|terms', 'to|terms', '(be|added)', 'to|test'))

build.formula

Convert a buildmer term list into a proper model formula

Description

Convert a buildmer term list into a proper model formula

Usage

build.formula(dep, terms, env = parent.frame())
Arguments

- **dep**: The dependent variable.
- **terms**: The term list.
- **env**: The environment of the formula to return.

Value

A formula.

Examples

```
library(buildmer)
form1 <- Reaction ~ Days + (Days|Subject)
terms <- tabulate.formula(form1)
form2 <- build.formula(dep='Reaction',terms)

# check that the two formulas give the same results
library(lme4)
check <- function (f) resid(lmer(f,sleepstudy))
all.equal(check(form1),check(form2))

# can also do double bars now
form1 <- Reaction ~ Days + (Days||Subject)
terms <- tabulate.formula(form1)
form2 <- build.formula(dep='Reaction',terms)
all.equal(check(form1),check(form2))
```

---

**buildbam**

*Use buildmer to fit big generalized additive models using bam from package mgcv*

Description

Use buildmer to fit big generalized additive models using bam from package mgcv

Usage

```
buildbam(  
  formula,  
  data = NULL,  
  family = gaussian(),  
  buildmerControl = buildmerControl()  
)
```
**buildclmm**

Arguments

- **formula**
  See the general documentation under `buildmer-package`
- **data**
  See the general documentation under `buildmer-package`
- **family**
  See the general documentation under `buildmer-package`
- **buildmerControl**
  Control arguments for `buildmer` — see the general documentation under `buildmerControl`

Details

To work around an issue in `bam`, you must make sure that your data do not contain a variable named 'intercept'.

Lme4 random effects are supported: they will be automatically converted using `re2mgcv`.

As `bam` uses PQL, only `crit='F'` and `crit='deviance'` (note that the latter is not a formal test) are supported for non-Gaussian errors.

See Also

- `buildmer-package`

Examples

```r
library(buildmer)
model <- buildbam(f1 ~ s(timepoint,by=following) + s(participant,by=following,bs='re') + s(participant,timepoint,by=following,bs='fs'),data=vowels)
```

**Description**

Use `buildmer` to fit cumulative link mixed models using `clmm` from package `ordinal`

Usage

```r
buildclmm(formula, data = NULL, buildmerControl = buildmerControl())
```

Arguments

- **formula**
  A formula specifying both fixed and random effects using `lme4` syntax
- **data**
  See the general documentation under `buildmer-package`
- **buildmerControl**
  Control arguments for `buildmer` — see the general documentation under `buildmerControl`
See Also

buildmer-package

Examples

```r
if (requireNamespace('ordinal')) {
  model <- buildclmm(SURENESS ~ PROD + (1|RESP), data=ordinal::soup,
                      buildmerControl=list(args=list(link='probit', threshold='equidistant')))
}
```

buildcustom

Use buildmer to perform stepwise elimination using a custom fitting function

Usage

```r
buildcustom(
  formula,
  data = NULL,
  fit = function(p, formula) stop("'fit' not specified"),
  crit = function(p, ref, alt) stop("'crit' not specified"),
  elim = function(x) stop("'elim' not specified"),
  REML = FALSE,
  buildmerControl = buildmerControl()
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>formula</code></td>
<td>See the general documentation under buildmer-package</td>
</tr>
<tr>
<td><code>data</code></td>
<td>See the general documentation under buildmer-package</td>
</tr>
<tr>
<td><code>fit</code></td>
<td>A function taking two arguments, of which the first is the buildmer parameter list <code>p</code> and the second one is a formula. The function must return a single object, which is treated as a model object fitted via the provided formula. The function must return an error (<code>stop()</code>) if the model does not converge.</td>
</tr>
<tr>
<td><code>crit</code></td>
<td>A function taking one argument and returning a single value. The argument is the return value of the function passed in <code>fit</code>, and the returned value must be a numeric indicating the goodness of fit, where smaller is better (like AIC or BIC).</td>
</tr>
<tr>
<td><code>elim</code></td>
<td>A function taking one argument and returning a single value. The argument is the return value of the function passed in <code>crit</code>, and the returned value must be a logical indicating if the small model must be selected (return <code>TRUE</code>) or the large model (return <code>FALSE</code>).</td>
</tr>
</tbody>
</table>
REML

A logical indicating if the fitting function wishes to distinguish between fits differing in fixed effects (for which \(p\$\text{reml} \) will be set to FALSE) and fits differing only in the random part (for which \(p\$\text{reml} \) will be TRUE). Note that this ignores the usual semantics of buildmer's optional \texttt{REML} argument, because they are redundant: if you wish to force REML on or off, simply code it so in your custom fitting function.

\texttt{buildmerControl}

Control arguments for buildmer — see the general documentation under \texttt{buildmerControl}

\textbf{See Also}

\texttt{buildmer-package}

\textbf{Examples}

```r
## Use \code{buildmer} to do stepwise linear discriminant analysis
library(buildmer)
migrant[,,-1] <- scale(migrant[,,-1])
flipfit <- function (p,formula) {
  # The predictors must be entered as dependent variables in a MANOVA
  # (i.e. the predictors must be flipped with the dependent variable)
  Y <- model.matrix(formula,migrant)
  m <- lm(Y ~ 0+migrant$changed)
  # the model may error out when asking for the MANOVA
  test <- try(anova(m))
  if (inherits(test,'try-error')) test else m
}
crit.F <- function (p,a,b) { # use whole-model F
  pvals <- anova(b)$\Pr(>F)$ # not valid for backward!
  pvals[length(pvals)-1]
}
crit.Wilks <- function (p,a,b) {
  if (is.null(a)) return(crit.F(p,a,b)) #not completely correct, but close as F approximates X2
  Lambda <- anova(b,test='Wilks')$Wilks[1]
  p <- length(coef(b))
  n <- 1
  m <- nrow(migrant)
  Bartlett <- ((p-n+1)/2-m)*log(Lambda)
  pchisq(Bartlett,n*p,lower.tail=FALSE)
}

# First, order the terms based on Wilks' Lambda
model <- buildcustom(changed ~ friends.nl+friends.be+multilingual+standard+hearing+reading+
  attention+sleep+gender+handedness+diglossic+age+years,buildmerControl=list(  
    fit=flipfit,crit=crit.Wilks,direction='order'))

# Now, use the six most important terms (arbitrary choice) in the LDA
if (require('MASS')) {
  model <- lda(changed ~ diglossic + age + reading + friends.be + years +
    multilingual,data=migrant)
}
```
Use buildmer to fit generalized additive models using gam from package mgcv

Usage

buildgam(
  formula,  
  data = NULL,  
  family = gaussian(),  
  quickstart = 0,  
  buildmerControl = buildmerControl()
)

Arguments

- **formula**: See the general documentation under `buildmer-package`
- **data**: See the general documentation under `buildmer-package`
- **family**: See the general documentation under `buildmer-package`
- **quickstart**: A numeric with values from 0 to 5. If set to 1, will use bam to obtain starting values for gam’s outer iteration, potentially resulting in a much faster fit for each model. If set to 2, will disregard ML/REML and always use bam’s fREML for the quickstart fit. 3 also sets discrete=TRUE. Values between 3 and 4 fit the quickstart model to a subset of that value (e.g. quickstart=3.1 fits the quickstart model to 10% of the data, which is also the default if quickstart=3. Values between 4 and 5 do the same, but also set a very sloppy convergence tolerance of 0.2.
- **buildmerControl**: Control arguments for buildmer — see the general documentation under `buildmerControl`

Details

To work around an issue in gam, you must make sure that your data do not contain a variable named ’intercept’.

lme4 random effects are supported: they will be automatically converted using `re2mgcv`.

If gam’s optimizer argument is not set to use outer iteration, gam fits using PQL. In this scenario, only crit=’F’ and crit=’deviance’ (note that the latter is not a formal test) are legitimate in the generalized case.

General families implemented in mgcv are supported, provided that they use normal formulas. Currently, this is only true of the cox.ph family. Because this family can only be fitted using REML,
buildgamm automatically sets gam's select argument to TRUE and prevents removal of parametric terms.

**buildmerControl**'s quickstart function may be used here. If you desire more control (e.g. discrete=FALSE but use.chol=TRUE), additional options can be provided as extra arguments and will be passed on to `bam` as they are applicable. Note that quickstart needs to be larger than 0 to trigger the quickstart path at all.

If scaled-t errors are used (family=scat), the quickstart path will also provide initial values for the two theta parameters (corresponding to the degrees of freedom and the scale parameter), but only if your installation of package mgcv is at least at version 1.8-32.

**See Also**

`buildmer-package`

**Examples**

```r
library(buildmer)
model <- buildgam(f1 ~ s(timepoint,by=following) + s(participant,by=following,bs='re') +
    s(participant,timepoint,by=following,bs='fs'),data=vowels)
```

**Description**

Use `buildmer` to fit big generalized additive models using `gamm` from package mgcv

**Usage**

```r
buildgamm(
    formula,
    data = NULL,
    family = gaussian(),
    buildmerControl = buildmerControl()
)
```

**Arguments**

- `formula` See the general documentation under `buildmer-package`
- `data` See the general documentation under `buildmer-package`
- `family` See the general documentation under `buildmer-package`
- `buildmerControl` Control arguments for buildmer — see the general documentation under `buildmerControl`
Details

The fixed and random effects are to be passed as a single formula in lme4 format. This is internally split up into the appropriate fixed and random parts. Only a single grouping factor is allowed. The random-effect covariance matrix is always unstructured. If you want to use pdMat covariance structures, you must (a) not specify any lme4 random-effects term in the formula, and (b) specify your own custom random argument in the args list in buildmerControl. Note that buildgamm will merely pass this through; no term reordering or stepwise elimination is done on a user-provided random argument.

See Also

buildmer-package

Examples

library(buildmer)
model <- buildgamm(f1 ~ s(timepoint,by=following) + (following|participant) +
s(participant,timepoint,by=following,bs='fs'),data=vowels)

Description

Use buildmer to fit generalized additive models using package gamm4

Usage

buildgamm4(
  formula,
  data = NULL,
  family = gaussian(),
  buildmerControl = buildmerControl()
)

Arguments

formula See the general documentation under buildmer-package
data See the general documentation under buildmer-package
family See the general documentation under buildmer-package
buildmerControl Control arguments for buildmer — see the general documentation under buildmerControl
The fixed and random effects are to be passed as a single formula in \texttt{lme4} format. This is internally split up into the appropriate fixed and random parts.

See Also

\texttt{buildmer-package}

Examples

```r
library(buildmer)
if (requireNamespace('gamm4')) {model <- buildgamm4(f1 ~ s(timepoint, by=following) + s(participant, timepoint, by=following, bs='fs'), data=vowels)
}
```

---

\texttt{buildGLMMadaptive} \hspace{1cm} \textit{Use \texttt{buildmer} to fit generalized linear mixed models using \texttt{mixed_model} from package \texttt{GLMMadaptive}}

Description

Use \texttt{buildmer} to fit generalized linear mixed models using \texttt{mixed_model} from package \texttt{GLMMadaptive}

Usage

\begin{verbatim}
buildGLMMadaptive(
  formula,
  data = NULL,
  family,
  buildmerControl = buildmerControl()
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{formula} \hspace{1cm} A formula specifying both fixed and random effects using \texttt{lme4} syntax. (Unlike \texttt{mixed_model}, \texttt{buildGLMMadaptive} does not use a separate random argument!)
  \item \texttt{data} \hspace{1cm} See the general documentation under \texttt{buildmer-package}
  \item \texttt{family} \hspace{1cm} See the general documentation under \texttt{buildmer-package}
  \item \texttt{buildmerControl} \hspace{1cm} Control arguments for \texttt{buildmer} — see the general documentation under \texttt{buildmerControl}
\end{itemize}
Details

The fixed and random effects are to be passed as a single formula in lme4 format. This is internally split up into the appropriate fixed and random parts.

As GLMMadaptive can only fit models with a single random-effect grouping factor, having multiple different grouping factors will raise an error.

If multiple identical random-effect grouping factors are provided, they will be concatenated into a single grouping factor using the double-bar syntax, causing GLMMadaptive to assume a diagonal random-effects covariance matrix. In other words, (1|g) + (0+x|g) will correctly be treated as diagonal, but note the caveat: (a|g) + (b|g) will also be treated as fully diagonal, even if a and b are factors which might still have had correlations between their individual levels! This is a limitation of both GLMMadaptive and buildmer’s approach to handling double bars.

See Also

buildmer-package

Examples

```r
if (requireNamespace('GLMMadaptive')) {
  # nonsensical model given these data
  model <- buildGLMMadaptive(stress ~ vowel + (vowel|participant),
                              family=binomial, data=vowels, buildmerControl=list(args=list(nAGQ=1)))
  # or with double-bar syntax for a diagonal r.e. cov. matrix
  model <- buildGLMMadaptive(stress ~ vowel + (vowel||participant),
                              family=binomial, data=vowels, buildmerControl=list(args=list(nAGQ=1)))
}
```

buildglmmTMB

Use buildmer to perform stepwise elimination on glmmTMB models

Description

Use buildmer to perform stepwise elimination on glmmTMB models

Usage

```r
buildglmmTMB(
  formula,
  data = NULL,
  family = gaussian(),
  buildmerControl = buildmerControl()
)
```
**buildgls**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>See the general documentation under <code>buildmer-package</code></td>
</tr>
<tr>
<td>data</td>
<td>See the general documentation under <code>buildmer-package</code></td>
</tr>
<tr>
<td>family</td>
<td>See the general documentation under <code>buildmer-package</code></td>
</tr>
<tr>
<td>buildmerControl</td>
<td>Control arguments for <code>buildmer</code> — see the general documentation under <code>buildmerControl</code></td>
</tr>
</tbody>
</table>

**See Also**

`buildmer-package`

**Examples**

```r
library(buildmer)
if (requireNamespace('glmmTMB')) {
  model <- buildglimmTMB(Reaction ~ Days + (Days|Subject), data=lme4::sleepstudy)
}
```

---

**Description**

Use `buildmer` to fit generalized-least-squares models using `gls` from `nlme`

**Usage**

```r
buildgls(formula, data = NULL, buildmerControl = buildmerControl())
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>See the general documentation under <code>buildmer-package</code></td>
</tr>
<tr>
<td>data</td>
<td>See the general documentation under <code>buildmer-package</code></td>
</tr>
<tr>
<td>buildmerControl</td>
<td>Control arguments for <code>buildmer</code> — see the general documentation under <code>buildmerControl</code></td>
</tr>
</tbody>
</table>

**Details**

A workaround is included to prevent an error when the model matrix is of less than full rank. The summary output of such a model will look a bit strange!

**See Also**

`buildmer-package`
Examples

```r
library(buildmer)
library(nlme)
vowels$event <- with(vowels, interaction(participant, word))
model <- buildgls(f1 ~ timepoint*following, data=vowels,
        buildmerControl=list(args=list(correlation=corAR1(form=~1|event))))
```

---

buildlme

Use buildmer to perform stepwise elimination of mixed-effects models fit via lme from nlme

Description

Use `buildmer` to perform stepwise elimination of mixed-effects models fit via `lme` from `nlme`

Usage

```r
buildlme(formula, data = NULL, buildmerControl = buildmerControl())
```

Arguments

- `formula`: A formula specifying both fixed and random effects using `lme4` syntax. (Unlike `lme`, `buildlme` does not use a separate random argument!)
- `data`: See the general documentation under `buildmer-package`
- `buildmerControl`: Control arguments for `buildmer` — see the general documentation under `buildmerControl`

Details

The fixed and random effects are to be passed as a single formula in `lme4` format. This is internally split up into the appropriate fixed and random parts. Only a single grouping factor is allowed. The random-effect covariance matrix is always unstructured. If you want to use `pdMat` covariance structures, you must (a) not specify any `lme4` random-effects term in the formula, and (b) specify your own custom random argument in the `args` list in `buildmerControl`. Note that `buildlme` will merely pass this through; no term reordering or stepwise elimination is done on a user-provided random argument.

See Also

- `buildmer-package`

Examples

```r
library(buildmer)
model <- buildlme(Reaction ~ Days + (Days|Subject), data=lme4::sleepstudy)
```
Use `buildmer` to fit mixed-effects models using `lmer/glmer` from `lme4`

### Description

Use `buildmer` to fit mixed-effects models using `lmer/glmer` from `lme4`

### Usage

```r
buildmer(
  formula,
  data = NULL,
  family = gaussian(),
  buildmerControl = buildmerControl()
)
```

### Arguments

- `formula` See the general documentation under `buildmer-package`
- `data` See the general documentation under `buildmer-package`
- `family` See the general documentation under `buildmer-package`
- `buildmerControl` Control arguments for `buildmer` — see the general documentation under `buildmerControl`

### Examples

```r
library(buildmer)
model <- buildmer(Reaction ~ Days + (Days|Subject), lme4::sleepstudy)

# Tests from github issue #2, that also show the use of the 'direction' and 'crit' parameters:
bm.test <- buildmer(cbind(incidence,size - incidence) ~ period + (1 | herd),
  family=binomial, data=lme4::cbpp)
bm.test <- buildmer(cbind(incidence,size - incidence) ~ period + (1 | herd),
  family=binomial, data=lme4::cbpp, buildmerControl=buildmerControl(direction="forward")
bm.test <- buildmer(cbind(incidence,size - incidence) ~ period + (1 | herd),
  family=binomial, data=lme4::cbpp, buildmerControl=buildmerControl(crit="AIC")
bm.test <- buildmer(cbind(incidence,size - incidence) ~ period + (1 | herd),
  family=binomial, data=lme4::cbpp, buildmerControl=buildmerControl(direction="forward", crit="AIC")

# Example showing use of the 'include' parameter to force a particular term into the model
m1 <- buildmer(Reaction ~ Days, data=lme4::sleepstudy, buildmerControl=list(include=~(1|Subject))
# the below are equivalent
m2 <- buildmer(Reaction ~ Days, data=lme4::sleepstudy, buildmerControl=list(include='(1|Subject)'))
m3 <- buildmer(Reaction ~ Days + (1|Subject), data=lme4::sleepstudy, buildmerControl=list( include=~(1|Subject))
m4 <- buildmer(Reaction ~ Days + (1|Subject), data=lme4::sleepstudy, buildmerControl=list( include='(1|Subject)'))
```
buildmer-class  

The buildmer class

Description

This is a simple convenience class that allows ‘anova’ and ‘summary’ calls to fall through to the underlying model object, while retaining buildmer’s iteration history. If you need to use the final model for other things, such as prediction, access it through the ‘model’ slot of the buildmer class object.

Slots

model  The final model containing only the terms that survived elimination
p  Parameters used during the fitting process
anova  The model’s ANOVA, if the model was built with ‘anova=TRUE’
summary  The model’s summary, if the model was built with ‘summary=TRUE’

See Also

buildmer

Examples

# Manually create a bare-bones buildmer object:
model <- lm(Sepal.Length ~ Petal.Length, iris)
p <- list(in.buildmer=FALSE)
library(buildmer)
bm <- mkBuildmer(model=model, p=p, anova=NULL, summary=NULL)
summary(bm)

buildmer.nb  

Use buildmer to fit negative-binomial models using glm.nb and glmer.nb

Description

Use buildmer to fit negative-binomial models using glm.nb and glmer.nb

Usage

buildmer.nb(formula, data = NULL, buildmerControl = buildmerControl())
buildmerControl

Arguments

- **formula**: See the general documentation under `buildmer-package`
- **data**: See the general documentation under `buildmer-package`

**buildmerControl**
Control arguments for `buildmer` — see the general documentation under `buildmerControl`

See Also

- `buildmer-package`

Examples

```r
library(buildmer)
if (requireNamespace("MASS")) {
  model <- buildmer.nb(Days ~ Sex*Age*Eth*Lrn, MASS::quine)
}
```

**Description**

`buildmerControl` provides all the knobs and levers that can be manipulated during the `buildmer` fitting and summary/anova process. Some of these are part of `buildmer`'s core functionality—for instance, **crit** allows to specify different elimination criteria, a core `buildmer` feature—whereas some are only meant for internal usage, e.g. **I_KNOW_WHAT_I_AM DOING** is only used to turn off the PQL safeguards in `buildbam/buildgam`, which you really should only do if you have a very good reason to believe that the PQL check is being triggered erroneously for your problem.

**Usage**

```r
buildmerControl(
  formula = quote(stop("No formula specified")),
  data = NULL,
  family = gaussian(),
  args = list(),
  direction = c("order", "backward"),
  cl = NULL,
  crit = NULL,
  elim = NULL,
  fit = function(...) stop("No fitting function specified"),
  include = NULL,
  quiet = FALSE,
  calc.anova = FALSE,
  calc.summary = TRUE,
  ddf = "Wald",
  quickstart = 0,
)```

Arguments

formula The model formula for the maximal model you would like to fit. Alternatively, a buildmer term list as obtained from `tabulate.formula`. In the latter formulation, you also need to specify a `dep=`... argument specifying the dependent variable to go along with the term list. See `tabulate.formula` for an example of where this is useful.

data The data to fit the model(s) to.

family The error distribution to use.

args Extra arguments passed to the fitting function.

direction Character string or vector indicating the direction for stepwise elimination; possible options are 'order' (order terms by their contribution to the model), 'backward' (backward elimination), 'forward' (forward elimination, implies order). The default is the combination `c('order', 'backward')`, to first make sure that the model converges and to then perform backward elimination; other such combinations are perfectly allowed.

c1 Specifies a cluster to use for parallelizing the evaluation of terms. This can be an object as returned by function `makeCluster` from package `parallel`, or a whole number to let buildmer create, manage, and destroy a cluster for you with the specified number of parallel processes.

crit Character string or vector determining the criterion used to test terms for their contribution to the model fit in the ordering step. Possible options are 'LRT' (likelihood-ratio test based on chi-square mixtures per Stram & Lee 1994 for random effects; this is the default), 'LL' (use the raw \(-2\) log likelihood), 'AIC' (Akaike Information Criterion), 'BIC' (Bayesian Information Criterion), and 'deviance' (explained deviance — note that this is not a formal test). If left at its default value of NULL, the same value is used as in the `elim` argument; if that is also NULL, both are set to 'LRT'.

elim Character string or vector determining the criterion used to test terms for elimination in the elimination step. Possible options are 'LRT' (likelihood-ratio test based on chi-square mixtures per Stram & Lee 1994 for random effects; this is the default), 'LL' (use the raw \(-2\) log likelihood), 'AIC' (Akaike Information Criterion), 'BIC' (Bayesian Information Criterion), and 'deviance' (explained deviance — note that this is not a formal test). If left at its default value of NULL, the same value is used as in the `crit` argument; if that is also NULL, both are set to 'LRT'.

buildmerControl

dep = NULL,
REML = NA,
can.use.reml = TRUE,
force.reml = FALSE,
singular.ok = FALSE,
grad.tol = formals(buildmer::converged)$grad.tol,
hess.tol = formals(buildmer::converged)$hess.tol,
I_KNOW_WHAT_I_AM_DOING = FALSE
}
**buildmerControl**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fit</strong></td>
<td>Internal parameter — do not modify.</td>
</tr>
<tr>
<td><strong>include</strong></td>
<td>A one-sided formula or character vector of terms that will be included in the model at all times and are not subject to testing for elimination. These do not need to be specified separately in the <code>formula</code> argument. Useful for e.g. passing correlation structures in glmTMB models.</td>
</tr>
<tr>
<td><strong>quiet</strong></td>
<td>A logical indicating whether to suppress progress messages.</td>
</tr>
<tr>
<td><strong>calc.anova</strong></td>
<td>Logical indicating whether to also calculate the ANOVA table for the final model after term elimination.</td>
</tr>
<tr>
<td><strong>calc.summary</strong></td>
<td>Logical indicating whether to also calculate the summary table for the final model after term elimination.</td>
</tr>
<tr>
<td><strong>ddf</strong></td>
<td>The method used for calculating ( p )-values for lme4 models and <code>calc.anova=TRUE</code> or <code>calc.summary=TRUE</code>. Options are 'Wald' (default), 'Satterthwaite' (if package <code>lmerTest</code> is available), 'Kenward-Roger' (if packages <code>lmerTest</code> and <code>pbkrtest</code> are available), and 'lme4' (no ( p )-values).</td>
</tr>
<tr>
<td><strong>quickstart</strong></td>
<td>For gam models only: a numeric with values from 0 to 5. If set to 1, will use bam to obtain starting values for gam's outer iteration, potentially resulting in a much faster fit for each model. If set to 2, will disregard ML/REML and always use bam's fREML for the quickstart fit. 3 also sets <code>discrete=TRUE</code>. Values between 3 and 4 fit the quickstart model to a subset of that value (e.g., <code>quickstart=3.1</code> fits the quickstart model to 10% of the data, which is also the default if <code>quickstart=3</code>). Values between 4 and 5 do the same, but also set a very sloppy convergence tolerance of 0.2.</td>
</tr>
<tr>
<td><strong>dep</strong></td>
<td>A character string specifying the name of the dependent variable. Only used if <code>formula</code> is a buildmer terms list.</td>
</tr>
<tr>
<td><strong>REML</strong></td>
<td>In some situations, the user may want to force REML on or off, rather than using buildmer's autodetection. If <code>REML=TRUE</code> (or more precisely, if <code>isTRUE(REML)</code> evaluates to true), then buildmer will always use REML. This results in invalid results if formal model-comparison criteria are used with models differing in fixed effects (and the user is not guarded against this), but is useful with the 'deviance-explained' criterion, where it is actually the default (you can disable this and use the 'normal' REML/ML-differentiating behavior by passing <code>REML=NA</code>).</td>
</tr>
<tr>
<td><strong>can.use.reml</strong></td>
<td>Internal option specifying whether the fitting engine should distinguish between fixed-effects and random-effects model comparisons. Do not set this option yourself unless you are programming a new fitting function for buildcustom.</td>
</tr>
<tr>
<td><strong>force.reml</strong></td>
<td>Internal option specifying whether, if not differentiating between fixed-effects and random-effects model comparisons, these comparisons should be based on ML or on REML (if possible). Do not set this option yourself unless you are programming a new fitting function for buildcustom. Enabling this option only makes sense for criteria that do not compare likelihoods, in which case this is an optimization; it is applied automatically for the 'deviance-explained' criterion.</td>
</tr>
<tr>
<td><strong>singular.ok</strong></td>
<td>Logical indicating whether singular fits are acceptable. Only for lme4 models.</td>
</tr>
<tr>
<td><strong>grad.tol</strong></td>
<td>Tolerance for declaring gradient convergence. For buildbam, the default value is multiplied by 100.</td>
</tr>
</tbody>
</table>
hess.tol  Tolerance for declaring Hessian convergence. For buildbam, the default value is multiplied by 100.

I_KNOW_WHAT_I_AM_DOING
An internal option that you should not modify unless you know what you are doing.

Details

With the default options, all buildmer functions will do two things:

1. Determine the order of the effects in your model, based on their importance as measured by the likelihood-ratio test statistic. This identifies the ‘maximal model’, which is the model containing either all effects specified by the user, or subset of those effects that still allow the model to converge, ordered such that the most information-rich effects have made it in.

2. Perform backward stepwise elimination based on the significance of the change in log-likelihood.

The final model is returned in the model slot of the returned buildmer object. All functions in the buildmer package are aware of the distinction between (f)REML and ML, and know to divide chi-square p-values by 2 when comparing models differing only in random effects (see Pinheiro & Bates 2000). The steps executed above can be changed using the direction argument, allowing for arbitrary chains of, for instance, forward-backward-forward stepwise elimination (although using more than one elimination method on the same data is not recommended). The criterion for determining the importance of terms in the ordering stage and the elimination of terms in the elimination stage can also be changed, using the crit argument.

---

buildmertree  Use buildmer to perform stepwise elimination for lmertree and glmertree models from package glmertree

Description

Use buildmer to perform stepwise elimination for lmertree and glmertree models from package glmertree

Usage

buildmertree(
  formula,  
  data = NULL,  
  family = gaussian(),  
  buildmerControl = buildmerControl(crit = "AIC")
)
**Arguments**

- **formula**
  Either a glmertree formula, looking like dep ~ left | middle | right where the middle part is an lme4-style random-effects specification, or an ordinary formula (or buildmer term list thereof) specifying only the dependent variable and the fixed and random effects for the regression part. In the latter case, the additional argument partitioning must be specified as a one-sided formula containing the partitioning part of the model.

- **data**
  See the general documentation under `buildmer-package`

- **family**
  See the general documentation under `buildmer-package`

- **buildmerControl**
  Control arguments for buildmer — see the general documentation under `buildmerControl`

**Details**

Note that the likelihood-ratio test is not available for glmertree models, as it cannot be assured that the models being compared are nested. The default is thus to use AIC. In the generalized case or when testing many partitioning variables, it is recommended to pass `joint=FALSE`, as this results in a dramatic speed gain and reduces the odds of the final glmer model failing to converge or converging singularly.

**See Also**

- `buildmer-package`

**Examples**

```r
if (requireNamespace('glmertree')) {
  model <- buildmertree(Reaction ~ 1 | (Days|Subject) | Days,
                         buildmerControl=buildmerControl(crit='LL',
                                                       direction='order',
                                                       args=list(joint=FALSE)),
                         data=lme4::sleepstudy)

  model <- buildmertree(Reaction ~ 1 | (Days|Subject) | Days,
                         buildmerControl=buildmerControl(crit='LL',
                                                       direction='order',
                                                       args=list(joint=FALSE)),
                         data=lme4::sleepstudy,family=Gamma(link=identity))
}
```

---

`buildmultinom`  
*Use buildmer to perform stepwise elimination for multinom models from package nnet*

**Description**

Use buildmer to perform stepwise elimination for multinom models from package nnet

**Usage**

```r
buildmultinom(formula, data = NULL, buildmerControl = buildmerControl())
```
Test a model for convergence

**Description**

Test a model for convergence

**Usage**

```r
converged(model, singular.ok = FALSE, grad.tol = 0.1, hess.tol = 0.01)
```

**Arguments**

- `model`: The model object to test.
- `singular.ok`: A logical indicating whether singular fits are accepted as ‘converged’ or not. Relevant only for lme4 models.
- `grad.tol`: The tolerance to use for checking the gradient. This is currently only used by mgcv, glmmTMB, and clm(m) models.
- `hess.tol`: The tolerance to use for checking the Hessian for negative eigenvalues. This is currently only used by mgcv, glmmTMB, and clm(m) models.

**Value**

Logical indicating whether the model converged.
Examples

```r
library(buildmer)
library(lme4)
good1 <- lm(Reaction ~ Days, sleepstudy)
good2 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
bad <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy, control=lmerControl(
  optimizer='bobyqa', optCtrl=list(maxfun=1)))
sapply(list(good1, good2, bad), converged)
```

Description

Diagonalize the random-effect covariance structure, possibly assisting convergence

Usage

```r
## S4 method for signature 'formula'
diag(x)
```

Arguments

- `x` A model formula.

Value

The formula with all random-effect correlations forced to zero, per Pinheiro & Bates (2000)

Examples

```
# 1. Create explicit columns for factor variables
library(buildmer)
vowels <- cbind(vowels, model.matrix(~vowel, vowels))
# 2. Create formula with diagonal covariance structure
form <- diag(f1 ~ (vowel1+vowel2+vowel3+vowel4)*timepoint*following +
  ((vowel1+vowel2+vowel3+vowel4)*timepoint*following | participant) +
  (timepoint | word))
# 3. Convert formula to buildmer terms list, grouping terms starting with 'vowel'
terms <- tabulate.formula(form, group='vowel[^:]')
# 4. Directly pass the terms object to buildmer, using the 'dep' argument to specify the
# dependent variable
model <- buildmer(terms, data=vowels, buildmerControl=list(dep='f1'))
```
LRTalpha \hspace{1cm} \textit{Generate an LRT elimination function with custom alpha level}

Description

The \texttt{elim} argument in \texttt{buildmerControl} can take any user-specified elimination function. \texttt{LRTalpha} generates such a function that uses the likelihood-ratio test, based on a user-specified alpha level. (For the default alpha of .05, one can also simply specify the string 'LRT' or the function \texttt{buildmer:::elim.LRT}).

Usage

\texttt{LRTalpha(alpha)}

Arguments

\begin{itemize}
  \item \texttt{alpha} \hspace{1cm} The alpha level for the likelihood-ratio test.
\end{itemize}

See Also

\texttt{buildmerControl}

migrant \hspace{1cm} \textit{A very small dataset from a pilot study on sound change.}

Description

A very small dataset from a pilot study on sound change.

Usage

\texttt{data(migrant)}

Format

A standard data frame.
**re2mgcv**

**Convert lme4 random-effect terms to mgcv 're' smooths**

**Description**

Convert lme4 random-effect terms to mgcv 're' smooths

**Usage**

`re2mgcv(formula, data)`

**Arguments**

- `formula`: The lme4 formula.
- `data`: The data.

**Examples**

```r
library(buildmer)
re <- re2mgcv(temp ~ angle + (1|replicate) + (1|recipe), lme4::cake)
model <- buildgam(re$formula, re$data, family = mgcv::scat)

# note: the below does NOT work, as the dependent variable is looked up in the data by name!
re <- re2mgcv(log(Reaction) ~ Days + (Days|Subject), lme4::sleepstudy)
```

---

**remove.terms**

**Remove terms from a formula**

**Description**

Remove terms from a formula

**Usage**

`remove.terms(formula, remove, check = TRUE)`

**Arguments**

- `formula`: The formula.
- `remove`: A vector of terms to remove. To remove terms nested inside random-effect groups, use `(term|group)` syntax. Note that marginality is respected, i.e. no effects will be removed if they participate in a higher-order interaction, and no fixed effects will be removed if a random slope is included over that fixed effect.
- `check`: A logical indicating whether effects should be checked for marginality. If TRUE (default), effects will not be removed if doing so would violate marginality. Setting check to FALSE will remove terms unconditionally.
Examples

library(buildmer)
remove.terms(Reaction ~ Days + (Days|Subject),'(Days|Subject)')
# illustration of the marginality checking mechanism:
# this refuses to remove the term:
remove.terms(Reaction ~ Days + (Days|Subject),'(1|Subject)')
# so does this, because marginality is checked before removal:
remove.terms(Reaction ~ Days + (Days|Subject),c('(Days|Subject)','(1|Subject)'))
# but it works with check=FALSE
remove.terms(Reaction ~ Days + (Days|Subject),'(1|Subject)'),check=FALSE)

---
tabulate.formula
Parse a formula into a buildmer terms list

Description
Parse a formula into a buildmer terms list

Usage

```r
tabulate.formula(formula, group = NULL)
```

Arguments

- `formula`: A formula.
- `group`: A character vector of regular expressions. Terms matching the same regular expression are assigned the same block, and will be evaluated together in buildmer functions.

Value

A buildmer terms list, which is just a normal data frame.

See Also

buildmer-package

Examples

```r
form <- diag(f1 ~ (vowel1+vowel2+vowel3+vowel4)*timepoint*following +
           ((vowel1+vowel2+vowel3+vowel4)*timepoint*following|participant) + (timepoint|word))
tabulate.formula(form)
tabulate.formula(form, group='vowel[1-4]')
```
vowels

<table>
<thead>
<tr>
<th>vowels</th>
<th>Vowel data from a pilot study.</th>
</tr>
</thead>
</table>

**Description**

Vowel data from a pilot study.

**Usage**

```r
data(vowels)
```

**Format**

A standard data frame.
Index

* datasets
  migrant, 24
  vowels, 27

add.terms, 3

build.formula, 3
buildbam, 4
buildclmm, 5
buildcustom, 6
buildgam, 2, 8
buildgamm, 9
buildgamm4, 10
buildGLMMadaptive, 11
buildglmmTMB, 12
buildgls, 13
buildlme, 14
buildmer, 2, 15, 16
buildmer-class, 16
buildmer-package, 2
buildmer.nb, 16
buildmerControl, 5, 7–11, 13–15, 17, 17, 21, 22, 24
buildmertree, 2, 20
buildmultinom, 21

collapsed, 22

diag.formula-method, 23
LRTalpha, 24

migrant, 24
mkBuildmer (buildmer-class), 16

re2mgcv, 5, 8, 25
remove.terms, 25

tabulate.formula, 18, 26

vowels, 27