Package 'emmeans'

February 24, 2018

Type Package

Title Estimated Marginal Means, aka Least-Squares Means

Version 1.1.2

Date 2018-02-24

Depends R (>= 3.2)

Imports estimability (>= 1.3), ggplot2, graphics, methods, stats, utils, nlme, coda (>= 0.17), multcomp, plyr, mvtnorm, xtable (>= 1.8-2)

Suggests bayesplot, car, lattice, mediation, multcompView, ordinal, pbkrtest (>= 0.4-1), lme4, lmerTest (>= 2.0.32), MASS, rsm, knitr, rmarkdown, testthat

Enhances CARBayes, coxme, gee, geepack, glmmADMB, MCMCglmm, MCMCpack, nnet, pscl, rstanarm, survival

Additional_repositories http://glmmadmb.r-forge.r-project.org/repos

URL https://github.com/rvlenth/emmeans

BugReports https://github.com/rvlenth/emmeans/issues

LazyData yes

ByteCompile yes

Description Obtain estimated marginal means (EMMs) for many linear, generalized linear, and mixed models. Compute contrasts or linear functions of EMMs, trends, and comparisons of slopes. Plots and compact letter displays. Least-squares means are discussed, and the term ``estimated marginal means'' is suggested, in Searle, Speed, and Milliken (1980) Population marginal means in the linear model: An alternative to least squares means, The American Statistician 34(4), 216-221 <doi:10.1080/00031305.1980.10483031>.

License GPL-2 | GPL-3

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Russell Lenth [aut, cre, cph],
Jonathon Love [ctb],
Maxime Herve [ctb]

Maintainer Russell Lenth <russell-lenth@uiowa.edu>

Repository CRAN

Date/Publication 2018-02-24 21:32:29 UTC

R topics documented:

`emmeans-package` *Estimated marginal means (aka Least-squares means)*

Description

This package provides methods for obtaining estimated marginal means (EMMs, also known as least-squares means) for factor combinations in a variety of models. Supported models include [generalized linear] models, models for counts, multivariate, multinomial and ordinal responses, survival models, GEEs, and Bayesian models. For the latter, posterior samples of EMMs are provided. The package can compute contrasts or linear combinations of these marginal means with various multiplicity adjustments. One can also estimate and contrast slopes of trend lines. Some graphical displays of these results are provided.

Overview

Vignettes A number of vignettes are provided to help the user get acquainted with the **emmeans** package and see some examples. See the vignette index.

Concept Estimated marginal means (see Searle *et al.* 1980 are popular for summarizing linear models that include factors. For balanced experimental designs, they are just the marginal means. For unbalanced data, they in essence estimate the marginal means you *would* have observed that the data arisen from a balanced experiment. Earlier developments regarding these techniques were developed in a least-squares context and are sometimes referred to as "least-squares means". Since its early development, the concept has expanded far beyond least-squares settings.

Reference grids The implementation in **emmeans** relies on our own concept of a *reference grid*, which is an array of factor and predictor levels. Predictions are made on this grid, and estimated marginal means (or EMMs) are defined as averages of these predictions over zero or more dimensions of the grid. The function `ref_grid` explicitly creates a reference grid that can subsequently be used to obtain least-squares means. The object returned by `ref_grid` is of class `"emmGrid"`, the same class as is used for estimated marginal means (see below).

Our reference-grid framework expands slightly upon Searle *et al.*'s definitions of EMMs, in that it is possible to include multiple levels of covariates in the grid.

Models supported As is mentioned in the package description, many types of models are supported by the package. See vignette("models", "emmeans") for full details. Some models may require other packages be installed in order to access all of the available features.

Estimated marginal means The `emmeans` function computes EMMs given a fitted model (or a previously constructed `emmGrid` object), using a specification indicating what factors to include. The `emtrends` function creates the same sort of results for estimating and comparing slopes of fitted lines. Both return an `emmGrid` object.

Summaries and analysis The `summary.emmGrid` method may be used to display an `emmGrid` object. Special-purpose summaries are available via `confint.emmGrid` and `test.emmGrid`, the latter of which can also do a joint test of several estimates. The user may specify by variables, multiplicity-adjustment methods, confidence levels, etc., and if a transformation or link function is involved, may reverse-transform the results to the response scale.

Contrasts and comparisons The `contrast` method for `emmGrid` objects is used to obtain contrasts among the estimates; several standard contrast families are available such as deviations from the mean, polynomial contrasts, and comparisons with one or more controls. Another `emmGrid` object is returned, which can be summarized or further analyzed. For convenience, a `pairs.emmGrid` method is provided for the case of pairwise comparisons. Related to this is the `cld.emmGrid` method, which provides a compact letter display for grouping pairs of means that are not significantly different. `cld` requires the **multcompView** package.

Graphs The `plot.emmGrid` method will display side-by-side confidence intervals for the estimates, and/or "comparison arrows" whereby the significance of pairwise differences can be judged by how much they overlap. The `emmip` function displays estimates like an interaction plot, multi-paneled if there are by variables. These graphics capabilities require the **lattice** package be installed.

MCMC support When a model is fitted using MCMC methods, the posterior chains(s) of parameter estimates are retained and converted into posterior samples of EMMs or contrasts thereof. These may then be summarized or plotted like any other MCMC results, using tools in, say **coda** or **bayesplot**.

multcomp interface The `as.glht` function and `glht` method for `emmGrid`s provide an interface to the `glht` function in the **multcomp** package, thus providing for more exacting simultaneous estimation or testing. The package also provides an `emm` function that works as an alternative to `mcp` in a call to `glht`.

`add_grouping` *Add a grouping factor*

Description

This function adds a grouping factor to an existing reference grid or other `emmGrid` object, such that the levels of an existing factor (call it the reference factor) are mapped to a smaller number of levels of the new grouping factor. The reference factor is then nested in the grouping factor. This facilitates obtaining marginal means of the grouping factor, and contrasts thereof.

Usage

```
add_grouping(object, newname, refname, newlevs)
```

Arguments

`object`	An `emmGrid` object
`newname`	Character name of grouping factor to add (different from any existing factor in the grid)
`refname`	Character name of the reference factor
`newlevs`	Character vector or factor of the same length as that of the levels for `refname`. The grouping factor `newname` will have the unique values of `newlevs` as its levels.

Value

A revised emmGrid object having an additional factor named newname, and a new nesting structure refname %in% newname

Note

By default, the levels of newname will be ordered alphabetically. To dictate a different ordering of levels, supply newlevs as a factor having its levels in the required order.

Examples

```
fiber.lm <- lm(strength ~ diameter + machine, data = fiber)
( frg <- ref_grid(fiber.lm) )

# Suppose the machines are two different brands
brands <- factor(c("FiberPro", "FiberPro", "Acme"), levels = c("FiberPro", "Acme"))
( gfrg <- add_grouping(frg, "brand", "machine", brands) )

emmeans(gfrg, "machine")

emmeans(gfrg, "brand")
```

---

as.emmGrid *Convert to and from* emmGrid *objects*

Description

These are useful utility functions for creating a compact version of an emmGrid object that may be saved and later reconstructed, or for converting old ref.grid or lsmobj objects into emmGrid objects.

Usage

```
as.emmGrid(object, ...)

## S3 method for class 'emmGrid'
as.list(x, ...)
```

Arguments

object	Object to be converted to class emmGrid. It may be a list returned by as.list.emmGrid, or a ref.grid or lsmobj object created by **emmeans**'s predecessor, the **lsmeans** package. An error is thrown if object cannot be converted.
...	In as.emmGrid, additional arguments passed to update.emmGrid before returning the object. This argument is ignored in as.list.emmGrid
x	An emmGrid object

Details

An emmGrid object is an S4 object, and as such cannot be saved in a text format or saved without a lot of overhead. By using as.list, the essential parts of the object are converted to a list format that can be easily and compactly saved for use, say, in another session or by another user. Providing this list as the arguments for emmobj allows the user to restore a working emmGrid object.

Value

as.emmGrid returns an object of class emmGrid.

as.list.emmGrid returns an object of class list.

See Also

emmobj

Examples

```
pigs.lm <- lm(log(conc) ~ source + factor(percent), data = pigs)
pigs.sav <- as.list(ref_grid(pigs.lm))

pigs.anew <- as.emmGrid(pigs.sav)
emmeans(pigs.anew, "source")

## Not run:
## Convert an entire workspace saved from an old **lsmeans** session
a.problem <- lsmeans::lsmeans(pigs.lm, "source")
#- Now global env contains at least two ref.grid and lsmobj objects,
#- and the "lsmeans" namespace is loaded
emmeans:::convert_workspace()
class(a.problem)
"lsmeans" %in% loadedNamespaces()
#- It's all better now

## End(Not run)
```

---

as.mcmc.emmGrid *Support for MCMC-based estimation*

Description

When a model is fitted using Markov chain Monte Carlo (MCMC) methods, its reference grid contains a post.beta slot. These functions transform those posterior samples to posterior samples of EMMs or related contrasts. They can then be summarized or plotted using, e.g., functions in the **coda** package.

Usage

```
## S3 method for class 'emmGrid'
as.mcmc(x, names = TRUE, sep.chains = TRUE, ...)

## S3 method for class 'emmGrid'
as.mcmc.list(x, names = TRUE, ...)
```

Arguments

x	An object of class emmGrid
names	Logical scalar or vector specifying whether variable names are appended to levels in the column labels for the as.mcmc or as.mcmc.list result – e.g., column names of treat A and treat B versus just A and B. When there is more than one variable involved, the elements of names are used cyclically.
sep.chains	Logical value. If TRUE, and there is more than one MCMC chain available, an mcmc.list object is returned by as.mcmc, with separate EMMs posteriors in each chain.
...	(Required but ignored)

Value

An object of class mcmc or mcmc.list.

Details

When the object's post.beta slot is non-trivial, as.mcmc will return an mcmc or mcmc.list object that can be summarized or plotted using methods in the **coda** package. In these functions, post.beta is transformed by post-multiplying it by t(linfct), creating a sample from the posterior distribution of LS means. In as.mcmc, if sep.chains is TRUE and there is in fact more than one chain, an mcmc.list is returned with each chain's results. The as.mcmc.list method is guaranteed to return an mcmc.list, even if it comprises just one chain.

---

auto.noise *Auto Pollution Filter Noise*

Description

Three-factor experiment comparing pollution-filter noise for two filters, three sizes of cars, and two sides of the car.

Usage

```
auto.noise
```

Format

A data frame with 36 observations on the following 4 variables.

`noise` Noise level in decibels - a numeric vector.

`size` The size of the vehicle - an ordered factor with levels `S`, `M`, `L`.

`type` Type of anti-pollution filter - a factor with levels `Std` and `Octel`

`side` The side of the car where measurement was taken – a factor with levels `L` and `R`.

Details

The data are from a statement by Texaco, Inc., to the Air and Water Pollution Subcommittee of the Senate Public Works Committee on June 26, 1973. Mr. John McKinley, President of Texaco, cited an automobile filter developed by Associated Octel Company as effective in reducing pollution. However, questions had been raised about the effects of filters on vehicle performance, fuel consumption, exhaust gas back pressure, and silencing. On the last question, he referred to the data included here as evidence that the silencing properties of the Octel filter were at least equal to those of standard silencers.

Source

The dataset was obtained from the Data and Story Library, http://lib.stat.cmu.edu/DASL/Datafiles/airpullutionfiltersdat.html (sic). However, the factor levels were assigned meaningful names, and the observations were sorted in random order as if this were the run order of the experiment.

Examples

```
noise.lm <- lm(noise ~ size * type * side, data = auto.noise)

# Interaction plot of predictions
emmip(noise.lm, type ~ size | side)

# Confidence intervals
plot(emmeans(noise.lm, ~ size | side*type))
```

`cld.emmGrid` *Extract and display information on all pairwise comparisons of least-squares means.*

Description

Extract and display information on all pairwise comparisons of least-squares means.

Usage

```
## S3 method for class 'emmGrid'
cld(object, details = FALSE, sort = TRUE, by,
  alpha = 0.05, Letters = c("1234567890", LETTERS, letters),
  reversed = FALSE, ...)
```

Arguments

object	An object of class emmGrid
details	Logical value determining whether detailed information on tests of pairwise comparisons is displayed
sort	Logical value determining whether the EMMs are sorted before the comparisons are produced. When TRUE, the results are displayed according to reversed.
by	Character value giving the name or names of variables by which separate families of comparisons are tested. If NULL, all means are compared. If missing, the object's by.vars setting, if any, is used.
alpha	Numeric value giving the significance level for the comparisons
Letters	Character vector of letters to use in the display. Any strings of length greater than 1 are expanded into individual characters
reversed	Logical value (passed to multcompView::multcompLetters.) If TRUE, the order of use of the letters is reversed. In addition, if both sort and reversed are TRUE, the sort order of results is reversed.
...	Arguments passed to contrast (for example, an adjust method)

This function uses the Piepho (2004) algorithm (as implemented in the **multcompView** package) to generate a compact letter display of all pairwise comparisons of least-squares means. The function obtains (possibly adjusted) P values for all pairwise comparisons of means, using the contrast function with method = "pairwise". When a P value exceeds alpha, then the two means have at least one letter in common.

Value

When details == FALSE, an object of class summary.ref_grid (which inherits from data.frame) showing the summary of EMMs with an added column named .groups containing the cld information. When details == TRUE, a list with the object just described, as well as the summary of the contrast results showing each comparison, its estimate, standard error, t ratio, and adjusted P value.

References

Hans-Peter Piepho (2004) An algorithm for a letter-based representation of all pairwise comparisons, Journal of Computational and Graphical Statistics, 13(2), 456-466.

See Also

cld in the **multcomp** package

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
cld(warp.emm)                  # implicitly uses by = "wool"
cld(warp.emm, by = "tension")  # overrides implicit 'by'

# Mimic grouping bars and compare all 6 means
cld(warp.emm, by = NULL, Letters = "||||||||", alpha = .01)
```

---

contrast *Contrasts and linear functions of EMMs*

Description

These methods provide for follow-up analyses of `emmGrid` objects: Contrasts, pairwise comparisons, tests, and confidence intervals. They may also be used to compute arbitrary linear functions of predictions or EMMs.

Usage

```
contrast(object, ...)

## S3 method for class 'emmGrid'
contrast(object, method = "eff", interaction = FALSE, by,
  offset = NULL, name = "contrast", options = get_emm_option("contrast"),
  type, adjust, simple, combine = FALSE, ...)

## S3 method for class 'emmGrid'
pairs(x, reverse = FALSE, ...)

## S3 method for class 'emmGrid'
coef(object, ...)
```

Arguments

`object`	An object of class `emmGrid`
`...`	Additional arguments passed to other methods
`method`	Character value giving the root name of a contrast method (e.g. `"pairwise"` – see emmc-functions). Alternatively, a named `list` of coefficients (for a contrast or linear function) that must each conform to the number of results in each `by` group. In a multi-factor situation, the factor levels are combined and treated like a single factor.
`interaction`	Character vector or logical value. If this is specified, `method` is ignored. See the "Interaction contrasts" section below for details.

by	Character names of variable(s) to be used for "by" groups. The contrasts or joint tests will be evaluated separately for each combination of these variables. If object was created with by groups, those are used unless overridden. Use by = NULL to use no by groups at all.
offset	Numeric vector of the same length as each by group. These values are added to their respective linear estimates. (It is ignored when interaction is specified.)
name	Character name to use to override the default label for contrasts used in table headings or subsequent contrasts of the returned object.
options	If non-NULL, a named list of arguments to pass to update.emmGrid, just after the object is constructed.
type	Character: prediction type (e.g., "response") – added to options
adjust	Character: adjustment method (e.g., "bonferroni") – added to options
simple	Character vector or list: Specify the factor(s) *not* in by, or a list thereof. See the section below on simple contrasts.
combine	Logical value that determines what is returned when simple is a list. See the section on simple contrasts.
x	An emmGrid object
reverse	Logical value - determines whether to use "pairwise" (if TRUE) or "revpairwise" (if FALSE).

Value

contrast and pairs return an object of class emmGrid. Its grid will correspond to the levels of the contrasts and any by variables. The exception is that an emm_list object is returned if simple is a list and complete is FALSE.

coef returns a data.frame containing the object's grid, along with columns named c.1, c.2, ... containing the contrast coefficients. If

Pairs method

The call pairs(object) is equivalent to contrast(object, method = "pairwise"); and pairs(object, reverse = TRUE) is the same as contrast(object, method = "revpairwise").

Interaction contrasts

When interaction is specified, interaction contrasts are computed: Contrasts are generated for each factor separately, one at a time; and these contrasts are applied to the object (the first time around) or to the previous result (subsequently). (Any factors specified in by are skipped.) The final result comprises contrasts of contrasts, or, equivalently, products of contrasts for the factors involved. Processing is done in the order of appearance in object@levels. With interaction = TRUE, method (if specified as character) is used for each contrast. If interaction is a character vector, the elements specify the respective contrast method(s); they are recycled as needed.

Simple contrasts

`simple` is essentially the complement of `by`: When `simple` is a character vector, `by` is set to all the factors in the grid *except* those in `simple`. If `simple` is a list, each element is used in turn as `simple`, and assembled in an `"emm_list"`. To generate *all* simple main effects, use `simple = "each"` (this works unless there actually is a factor named `"each"`). Note that a non-missing `simple` will cause `by` to be ignored.

Ordinarily, when `simple` is a list or `"each"`, the return value is an `emm_list` object with each entry in correspondence with the entries of `simple`. However, with `combine = TRUE`, the elements are all combined into one family of contrasts in a single `emmGrid` object using `rbind.emmGrid`.. In that case, the `adjust` argument sets the adjustment method for the combined set of contrasts.

Note

When `object` has a nesting structure (this can be seen via `str(object)`), then any grouping factors involved are forced into service as `by` variables, and the contrasts are thus computed separately in each nest. This in turn may lead to an irregular grid in the returned `emmGrid` object, which may not be valid for subsequent `emmeans` calls.

Examples

```
warp.lm <- lm(breaks ~ wool*tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
contrast(warp.emm, "poly")    # inherits 'by = "wool"' from warp.emm
pairs(warp.emm)               # ditto
contrast(warp.emm, "eff", by = NULL)  # contrasts of the 6 factor combs
pairs(warp.emm, simple = "wool") # same as pairs(warp.emm, by = "tension")

# Do all "simple" comparisons, combined into one family
pairs(warp.emm, simple = "each", combine = TRUE)

## Not run:

## Note that the following are NOT the same:
contrast(warp.emm, simple = c("wool", "tension"))
contrast(warp.emm, simple = list("wool", "tension"))
## The first generates contrasts for combinations of wool and tension
##   (same as by = NULL)
## The second generates contrasts for wool by tension, and for
##   tension by wool, respectively.

## End(Not run)

# An interaction contrast for tension:wool
tw.emm <- contrast(warp.emm, interaction = c("poly", "consec"), by = NULL)
tw.emm          # see the estimates
coef(tw.emm)    # see the contrast coefficients
```

contrast-methods *Contrast families*

Description

Contrast families

Usage

```
pairwise.emmc(levs, ...)

revpairwise.emmc(levs, ...)

tukey.emmc(levs, reverse = FALSE)

poly.emmc(levs, max.degree = min(6, k - 1))

trt.vs.ctrl.emmc(levs, ref = 1)

trt.vs.ctrl1.emmc(levs, ...)

trt.vs.ctrlk.emmc(levs, ...)

dunnett.emmc(levs, ref = 1)

eff.emmc(levs, ...)

del.eff.emmc(levs, ...)

consec.emmc(levs, reverse = FALSE, ...)

mean_chg.emmc(levs, reverse = FALSE, ...)
```

Arguments

levs	Vector of factor levels
...	Additional arguments (these are ignored, but needed to make these functions interchangeable) Each contrast family has a default multiple-testing adjustment as noted below. These adjustments are often only approximate; for a more exacting adjustment, use the interfaces provided to glht in the **multcomp** package. pairwise.emmc, revpairwise.emmc, and tukey.emmc generate contrasts for all pairwise comparisons among least-squares means at the levels in levs. The distinction is in which direction they are subtracted. For factor levels A, B, C, D, pairwise.emmc generates the comparisons A-B, A-C, A-D, B-C, B-D, and C-D, whereas revpairwise.emmc generates B-A, C-A, C-B, D-A, D-B, and D-C. tukey.emmc invokes pairwise.emmc or revpairwise.emmc depending on

reverse. The default multiplicity adjustment method is "tukey", which is only approximate when the standard errors differ.

poly.emmc generates orthogonal polynomial contrasts, assuming equally-spaced factor levels. These are derived from the poly function, but an *ad hoc* algorithm is used to scale them to integer coefficients that are (usually) the same as in published tables of orthogonal polynomial contrasts. The default multiplicity adjustment method is "none".

trt.vs.ctrl.emmc and its relatives generate contrasts for comparing one level (or the average over specified levels) with each of the other levels. The argument ref should be the index(es) (not the labels) of the reference level(s). trt.vs.ctrl1.emmc is the same as trt.vs.ctrl.emmc with a reference value of 1, and trt.vs.ctrlk.emmc is the same as trt.vs.ctrl with a reference value of length(levs). dunnett.emmc is the same as trt.vs.ctrl. The default multiplicity adjustment method is "dunnettx", a close approximation to the Dunnett adjustment.

consec.emmc and mean_chg.emmc are useful for contrasting treatments that occur in sequence. For a factor with levels A, B, C, D, E, consec.emmc generates the comparisons B-A, C-B, and D-C, while mean_chg.emmc generates the contrasts (B+C+D)/3 - A, (C+D)/2 - (A+B)/2, and D - (A+B+C)/3. With reverse = TRUE, these differences go in the opposite direction.

eff.emmc and del.eff.emmc generate contrasts that compare each level with the average over all levels (in eff.emmc) or over all other levels (in del.eff.emmc). These differ only in how they are scaled. For a set of k EMMs, del.eff.emmc gives weight 1 to one EMM and weight -1/(k-1) to the others, while eff.emmc gives weights (k-1)/k and -1/k respectively, as in subtracting the overall EMM from each EMM. The default multiplicity adjustment method is "fdr". This is a Bonferroni-based method and is slightly conservative; see p.adjust.

reverse	Logical value to determine the direction of comparisons
max.degree	Integer specifying the maximum degree of polynomial contrasts
ref	Integer(s) specifying which level(s) to use as the reference

Value

A data.frame, each column containing contrast coefficients for levs. The "desc" attribute is used to label the results in emmeans, and the "adjust" attribute gives the default adjustment method for multiplicity.

Examples

```
warp.lm <- lm(breaks ~ wool*tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
contrast(warp.emm, "poly")

### Setting up a custom contrast function
helmert.emmc <- function(levs, ...) {
    M <- as.data.frame(contr.helmert(levs))
    names(M) <- paste(levs[-1],"vs earlier")
    attr(M, "desc") <- "Helmert contrasts"
```

```
    M
}
contrast(warp.emm, "helmert")
## Not run:
# See what is used for polynomial contrasts with 6 levels
emmeans:::poly.emmc(1:6)

## End(Not run)
```

---

emm *Support for* multcomp::glht

Description

These functions and methods provide an interface between **emmeans** and the glht function for simultaneous inference provided by the **multcomp** package.

Usage

```
emm(...)

as.glht(object, ...)

## S3 method for class 'emmGrid'
as.glht(object, ...)
```

Arguments

...	In emm, the specs, by, and contr arguments you would normally supply to emmeans. Only specs is required. Otherwise, arguments that are passed to other methods.
object	An object of class emmGrid or emm_list

Details

emm is meant to be called only *from* "glht" as its second (linfct) argument. It works similarly to mcp, except with specs (and optionally by and contr arguments) provided as in a call to emmeans.

Value

emm returns an object of an intermediate class for which there is a glht method.

as.glht returns an object of class glht or glht_list according to whether object is of class emmGrid or emm_list. See Details below for more on glht_lists.

Details

A glht_list object is simply a list of glht objects. It is created as needed – for example, when there is a by variable. Appropriate convenience methods coef, confint, plot, summary, and vcov are provided, which simply apply the corresponding glht methods to each member.

Note

The multivariate-t routines used by `glht` require that all estimates in the family have the same integer degrees of freedom. In cases where that is not true, a message is displayed that shows what df is used. The user may override this via the `df` argument.

Examples

```
require(multcomp)

warp.lm <- lm(breaks ~ wool*tension, data = warpbreaks)

# Using 'emm'
summary(glht(warp.lm, emm(pairwise ~ tension | wool)))

# Same, but using an existing 'emmeans' result
warp.emm <- emmeans(warp.lm, ~ tension | wool)
summary(as.glht(pairs(warp.emm)))

# Same contrasts, but treat as one family
summary(as.glht(pairs(warp.emm), by = NULL))
```

`emmeans` *Estimated marginal means (Least-squares means)*

Description

Compute estimated marginal means (EMMs) for specified factors or factor combinations in a linear model; and optionally, comparisons or contrasts among them. EMMs are also known as least-squares means.

Usage

```
emmeans(object, specs, by = NULL, fac.reduce = function(coefs) apply(coefs,
  2, mean), contr, options = get_emm_option("emmeans"), weights, trend, ...)
```

Arguments

`object`	An object of class `emmGrid`; or a fitted model object that is supported, such as the result of a call to `lm` or `lmer`. Many fitted-model objects are supported; see `vignette("models", "emmeans")` for details.
`specs`	A `character` vector specifying the names of the predictors over which EMMs are desired. `specs` may also be a `formula` or a `list` (optionally named) of valid `specs`. Use of formulas is described in the Details section below.
`by`	A character vector specifying the names of predictors to condition on.

fac.reduce	A function that combines the rows of a matrix into a single vector. This implements the "marginal averaging" aspect of EMMs. The default is the mean of the rows. Typically if it is overridden, it would be some kind of weighted mean of the rows. If fac.reduce is nonlinear, bizarre results are likely, and EMMs will not be interpretable. NOTE: If the weights argument is non-missing, fac.reduce is ignored.
contr	A character value or list specifying contrasts to be added. See contrast. NOTE: contr is ignored when specs is a formula.
options	If non-NULL, a named list of arguments to pass to update.emmGrid, just after the object is constructed.
weights	Character value, numeric vector, or numeric matrix specifying weights to use in averaging predictions. See "Weights" section below.
trend	This is now deprecated. Use emtrends instead.
...	This is used only when object is not already a "ess" object, these arguments are passed to ref_grid. Common examples are at, cov.reduce, data, cdetype, transform, df, nesting, and vcov.. Model-type-specific options (see vignette("models", "emmeans")), commonly mode, may be used here as well. In addition, if the model formula contains references to variables that are not predictors, you must provide a params argument with a list of their names.

Value

When specs is a character vector or one-sided formula, an object of class "emmGrid". A number of methods are provided for further analysis, including summary.emmGrid, confint.emmGrid, test.emmGrid, contrast.emmGrid, pairs.emmGrid, and cld.emmGrid. When specs is a list or a formula having a left-hand side, the return value is an emm_list object, which is simply a list of emmGrid objects.

Details

Estimated marginal means or EMMs (sometimes called least-squares means) are predictions from a linear model over a *reference grid*; or marginal averages thereof. The ref_grid function identifies/creates the reference grid upon which emmeans is based.

For those who prefer the terms "least-squares means" or "predicted marginal means", functions lsmeans and pmmeans are provided as wrappers. See wrappers.

If specs is a formula, it should be of the form ~ specs, ~ specs | by, contr ~ specs, or contr ~ specs | by. The formula is parsed and the variables therein are used as the arguments specs, by, and contr as indicated. The left-hand side is optional, but if specified it should be the name of a contrast family (e.g., pairwise). Operators like * or : are needed in the formula to delineate names, but otherwise are ignored.

In the special case where the mean (or weighted mean) of all the predictions is desired, specify specs as ~ 1 or "1".

A number of standard contrast families are provided. They can be identified as functions having names ending in .emmc – see the documentation for emmc-functions for details – including how to write your own .emmc function for custom contrasts.

Weights

If `weights` is a vector, its length must equal the number of predictions to be averaged to obtain each EMM. If a matrix, each row of the matrix is used in turn, wrapping back to the first row as needed. When in doubt about what is being averaged (or how many), first call `emmeans` with `weights = "show.levels"`.

If `weights` is a string, it should partially match one of the following:

`"equal"` Use an equally weighted average.

`"proportional"` Weight in proportion to the frequencies (in the original data) of the factor combinations that are averaged over.

`"outer"` Weight in proportion to each individual factor's marginal frequencies. Thus, the weights for a combination of factors are the outer product of the one-factor margins

`"cells"` Weight according to the frequencies of the cells being averaged.

`"flat"` Give equal weight to all cells with data, and ignore empty cells.

`"show.levels"` This is a convenience feature for understanding what is being averaged over. Instead of a table of EMMs, this causes the function to return a table showing the levels that are averaged over, in the order that they appear.

Outer weights are like the 'expected' counts in a chi-square test of independence, and will yield the same results as those obtained by proportional averaging with one factor at a time. All except `"cells"` uses the same set of weights for each mean. In a model where the predicted values are the cell means, cell weights will yield the raw averages of the data for the factors involved. Using `"flat"` is similar to `"cells"`, except nonempty cells are weighted equally and empty cells are ignored.

See Also

`ref_grid`, `contrast`, vignette("models", "emmeans")

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
emmeans (warp.lm,  ~ wool | tension)
# or equivalently emmeans(warp.lm, "wool", by = "tension")

emmeans (warp.lm, poly ~ tension | wool)
```

---

emmGrid-class *The* emmGrid *class*

Description

The `emmGrid` class encapsulates linear functions of regression parameters, defined over a grid of predictors. This includes reference grids and grids of marginal means thereof (aka estimated marginal means). Objects of class 'emmGrid' may be used independently of the underlying model object. Instances are created primarily by `ref_grid` and `emmeans`, and several related functions.

Slots

`model.info` list. Contains the elements `call` (the call that produced the model), `terms` (its `terms` object), and `xlev` (factor-level information)

`roles` list. Contains at least the elements `predictors`, `responses`, and `multresp`. Each is a character vector of names of these variables.

`grid` data.frame. Contains the combinations of the variables that define the reference grid. In addition, there is an auxiliary column named `".wgt."` holding the observed frequencies or weights for each factor combination (excluding covariates). If the model has one or more `offset()` calls, there is an another auxiliary column named `".offset."`. Auxiliary columns are not considered part of the reference grid. (However, any variables included in `offset` calls *are* in the reference grid.)

`levels` list. Each entry is a character vector with the distinct levels of each variable in the reference grid. Note that `grid` is obtained by applying the function `expand.grid` to this list

`matlevs` list. Like `levels` but has the levels of any matrices in the original dataset. Matrix columns are always concatenated and treated as a single variable for purposes of the reference grid

`linfct` matrix. Each row consists of the linear function of the regression coefficients for predicting its corresponding element of the reference grid. The rows of this matrix go in one-to-one correspondence with the rows of `grid`, and the columns with elements of `bhat`.

`bhat` numeric. The regression coefficients. If there is a multivariate response, the matrix of coefficients is flattened to a single vector, and `linfct` and `V` redefined appropriately. Important: `bhat` must *include* any `NA` values produced as a result of collinearity in the predictors. These are taken care of later in the estimability check.

`nbasis` matrix. The basis for the non-estimable functions of the regression coefficients. Every EMM will correspond to a linear combination of rows of `linfct`, and that result must be orthogonal to all the columns of `nbasis` in order to be estimable. If everything is estimable, `nbasis` should be a 1 x 1 matrix of `NA`.

`V` matrix. The symmetric variance-covariance matrix of `bhat`

`dffun` function having two arguments. `dffun(k, dfargs)` should return the degrees of freedom for the linear function `sum(k*bhat)`, or `NA` if unavailable

`dfargs` list. Used to hold any additional information needed by `dffun`.

`misc` list. Additional information used by methods. These include at least the following: `estName` (the label for the estimates of linear functions), and the default values of `infer`, `level`, and `adjust` to be used in the `summary.emmGrid` method. Elements in this slot may be modified if desired using the `update.emmGrid` method.

`post.beta` matrix. A sample from the posterior distribution of the regression coefficients, if MCMC methods were used; or a 1 x 1 matrix of `NA` otherwise. When it is non-trivial, the `as.mcmc.emmGrid` method returns `post.beta %*% t(linfct)`, which is a sample from the posterior distribution of the EMMs.

Methods

All methods for these objects are S3 methods except for `show`. They include `[.emmGrid`, `as.glht.emmGrid`, `as.mcmc.emmGrid`, `as.mcmc.list.emmGrid`, `cld.emmGrid`, `coef.emmGrid`, `confint.emmGrid`, `contrast.emmGrid`, `pairs.emmGrid`, `plot.emmGrid`, `predict.emmGrid`, `print.emmGrid`, `rbind.emmGrid`, `show.emmGrid`, `str.emmGrid`, `summary.emmGrid`, `test.emmGrid`, `update.emmGrid`, `vcov.emmGrid`, and `xtable.emmGrid`

emmip *Interaction-style plots for estimated marginal means*

Description

Creates an interaction plot of EMMs based on a fitted model and a simple formula specification.

Usage

```
emmip(object, formula, ...)

## Default S3 method:
emmip(object, formula, type, CIs = FALSE,
  engine = get_emm_option("graphics.engine"), pch = c(1, 2, 6, 7, 9, 10,
  15:20), lty = 1, col = NULL, plotit = TRUE, ...)
```

Arguments

`object`	An object of class `emmGrid`, or a fitted model of a class supported by the **emmeans** package
`formula`	Formula of the form `trace.factors ~ x.factors | by.factors`. The EMMs are plotted against `x.factor` for each level of `trace.factors`. `by.factors` is optional, but if present, it determines separate panels. Each element of this formula may be a single factor in the model, or a combination of factors using the `*` operator.
`...`	Additional arguments passed to `emmeans` (when `object` is not already an `emmGrid` object), `ggplot`, or `xyplot`.
`type`	As in `predict.emmGrid`, this determines whether we want to inverse-transform the predictions (`type = "response"`) or not (any other choice). The default is `"link"`, unless the `"predict.type"` option is in force; see `emm_options`.
`CIs`	Logical value. If `TRUE`, confidence intervals are added to the plot (works only with `engine = "ggplot"`)
`engine`	Character value matching `"ggplot"` (default) or `"lattice"`. The graphics engine to be used to produce the plot. These require, respectively, the **ggplot2** or **lattice** package to be installed.
`pch`	The plotting characters to use for each group (i.e., levels of `trace.factors`). They are recycled as needed.
`lty`	The line types to use for each group. Recycled as needed.
`col`	The colors to use for each group, recycled as needed. If not specified, the default trellis colors are used.
`plotit`	Logical value. If `TRUE`, the plot is displayed. Otherwise, one may use the `"lattice"` attribute of the returned object and print it, perhaps after additional manipulation.

Value

If `plotit = FALSE`, a `data.frame` (actually, a `summary_emm` object) with the table of EMMs that would be plotted. The variables plotted are named `xvar` and `yvar`, and the trace factor is named `tvar`. This data frame has an added `"labs"` attribute containing the labels `xlab`, `ylab`, and `tlab` for these respective variables. The confidence limits are also included, renamed `LCL` and `UCL`.

If `plotit = TRUE`, the function returns an object of class `"ggplot"` or a `"trellis"`, depending on `engine`.

Details

If `object` is a fitted model, `emmeans` is called with an appropriate specification to obtain estimated marginal means for each combination of the factors present in `formula` (in addition, any arguments in `...` that match `at`, `trend`, `cov.reduce`, or `fac.reduce` are passed to `emmeans`). Otherwise, if `object` is an `emmGrid` object, its first element is used, and it must contain one estimate for each combination of the factors present in `formula`.

Note

Conceptually, this function is equivalent to `interaction.plot` where the summarization function is thought to return the EMMs.

See Also

`emmeans`, `interaction.plot`

Examples

```
#--- Three-factor example
noise.lm = lm(noise ~ size * type * side, data = auto.noise)

# Separate interaction plots of size by type, for each side
emmip(noise.lm, type ~ size | side)

# One interaction plot, using combinations of size and side as the x factor
emmip(noise.lm, type ~ side * size)

# One interaction plot using combinations of type and side as the trace factor
emmip(noise.lm, type * side ~ size)

# Individual traces in panels
emmip(noise.lm, ~ size | type * side)
```

emmobj *Construct an* emmGrid *object from scratch*

Description

This allows the user to incorporate results obtained by some analysis into an emmGrid object, enabling the use of emmGrid methods to perform related follow-up analyses.

Usage

```
emmobj(bhat, V, levels, linfct, df = NA, dffun, dfargs = list(),
  post.beta = matrix(NA), ...)
```

Arguments

bhat	Numeric. Vector of regression coefficients
V	Square matrix. Covariance matrix of bhat
levels	Named list or vector. Levels of factor(s) that define the estimates defined by linfct. If not a list, we assume one factor named "level"
linfct	Matrix. Linear functions of bhat for each combination of levels.
df	Numeric value or function with arguments (x, dfargs). If a number, that is used for the degrees of freedom. If a function, it should return the degrees of freedom for sum(x*bhat), with any additional parameters in dfargs.
dffun	Overrides df if specified. This is a convenience to match the slot names of the returned object.
dfargs	List containing arguments for df. This is ignored if df is numeric.
post.beta	Matrix whose columns comprise a sample from the posterior distribution of the regression coefficients (so that typically, the column averages will be bhat). A 1 x 1 matrix of NA indicates that such a sample is unavailable.
...	Arguments passed to update.emmGrid

Details

The arguments must be conformable. This includes that the length of bhat, the number of columns of linfct, and the number of columns of post.beta must all be equal. And that the product of lengths in levels must be equal to the number of rows of linfct. The grid slot of the returned object is generated by expand.grid using levels as its arguments. So the rows of linfct should be in corresponding order.

Value

An emmGrid object

Examples

```
# Given summary statistics for 4 cells in a 2 x 2 layout, obtain
# marginal means and comparisons thereof. Assume heteroscedasticity
# and use the Satterthwaite method
levels <- list(trt = c("A", "B"), dose = c("high", "low"))
ybar <- c(57.6, 43.2, 88.9, 69.8)
s <-    c(12.1, 19.5, 22.8, 43.2)
n <-    c(44,   11,   37,   24)
se2 = s^2 / n
Satt.df <- function(x, dfargs)
    sum(x * dfargs$v)^2 / sum((x * dfargs$v)^2 / (dfargs$n - 1))

expt.rg <- emmobj(bhat = ybar, V = diag(se2),
    levels = levels, linfct = diag(c(1, 1, 1, 1)),
    df = Satt.df, dfargs = list(v = se2, n = n), estName = "mean")
plot(expt.rg)

( trt.emm <- emmeans(expt.rg, "trt") )
( dose.emm <- emmeans(expt.rg, "dose") )

rbind(pairs(trt.emm), pairs(dose.emm), adjust = "mvt")
```

`emm_list` *The* `emm_list` *class*

Description

An `emm_list` object is simply a list of `emmGrid` objects. Such a list is returned, for example, by `emmeans` with a two-sided formula or a list as its `specs` argument.

Details

Methods for `emm_list` objects include `summary`, `cld`, `coef`, `confint`, `contrast`, `pairs`, `print`, and `test`. These are all the same as those methods for `emmGrid` objects, with an additional `which` argument (integer) to specify which members of the list to use. The default is `which = seq_along(object)`; i.e., the method is applied to every member of the `emm_list` object.

As an example, to summarize a single member – say the second one – of an `emm_list`, one may use `summary(object, which = 2)`, but it is probably preferable to directly summarize it using `summary(object[[2]])`.

`emtrends` *Estimated marginal means of linear trends*

Description

The `emtrends` function is useful when a fitted model involves a numerical predictor x interacting with another predictor `a` (typically a factor). Such models specify that x has a different trend depending on a; thus, it may be of interest to estimate and compare those trends. Analogous to the `emmeans` setting, we construct a reference grid of these predicted trends, and then possibly average them over some of the predictors in the grid.

Usage

```
emtrends(model, specs, var, delta.var = 0.01 * rng, data,
  transform = c("none", "response"), ...)
```

Arguments

`model`	A supported model object (*not* a reference grid)
`specs`	Specifications for what marginal trends are desired – as in `emmeans`
`var`	Character value giving the name of a variable with respect to which a difference quotient of the linear predictors is computed. In order for this to be useful, `var` should be a numeric predictor that interacts with at least one factor in `specs`. Then instead of computing EMMs, we compute and compare the slopes of the `var` trend over levels of the specified other predictor(s). As in EMMs, marginal averages are computed for the predictors in `specs` and `by`. See also the "Generalizations" section below.
`delta.var`	The value of h to use in forming the difference quotient $(f(x+h) - f(x))/h$. Changing it (especially changing its sign) may be necessary to avoid numerical problems such as logs of negative numbers. The default value is 1/100 of the range of `var` over the dataset.
`data`	As in `ref_grid`, you may use this argument to supply the dataset used in fitting the model, for situations where it is not possible to reconstruct the data. Otherwise, leave it missing.
`transform`	If `object` has a response transformation or link function, then specifying `transform = "response"` will cause `emtrends` to calculate the trends after back-transforming to the response scale. This is done using the chain rule, and standard errors are estimated via the delta method. With `transform = "none"` (the default), the trends are calculated on the scale of the linear predictor, without back-transforming it. This argument works similarly to the `transform` argument in `ref_grid`, in that the returned object is re-gridded to the new scale (see also `regrid`).
`...`	Additional arguments passed to other methods or to `ref_grid`

Value

An `emmGrid` or `emm_list` object, according to `specs`. See `emmeans` for more details on when a list is returned.

Generalizations

Instead of a single predictor, the user may specify some monotone function of one variable, e.g., `var = "log(dose)"`. If so, the chain rule is applied. Note that, in this example, if `model` contains `log(dose)` as a predictor, we will be comparing the slopes estimated by that model, whereas specifying `var = "dose"` would perform a transformation of those slopes, making the predicted trends vary depending on `dose`.

See Also

`link{emmeans}`, `ref_grid`

Examples

```
fiber.lm <- lm(strength ~ diameter*machine, data=fiber)
# Obtain slopes for each machine ...
( fiber.emt <- emtrends(fiber.lm, "machine", var = "diameter") )
# ... and pairwise comparisons thereof
pairs(fiber.emt)

# Suppose we want trends relative to sqrt(diameter)...
emtrends(fiber.lm, ~ machine | diameter, var = "sqrt(diameter)",
         at = list(diameter = c(20, 30)))
```

`extending-emmeans` *Support functions for model extensions*

Description

This documents the methods that `ref_grid` calls. A user or package developer may add **emmeans** support for a model class by writing `recover_data` and `emm_basis` methods for that class.

Usage

```
recover_data(object, ...)

## S3 method for class 'call'
recover_data(object, trms, na.action, data = NULL,
  params = NULL, ...)

emm_basis(object, trms, xlev, grid, ...)
```

Arguments

`object`	An object of the same class as is supported by a new method.
`...`	Additional parameters that may be supported by the method.
`trms`	The `terms` component of `object` (typically with the response deleted, e.g. via `delete.response`)

`na.action`	Integer vector of indices of observations to ignore; or `NULL` if none
`data`	Data frame. Usually, this is `NULL`. However, if non-null, this is used in place of the reconstructed dataset. It must have all of the predictors used in the model, and any factor levels must match those used in fitting the model.
`params`	Character vector giving the names of any variables in the model formula that are *not* predictors. An example would be a variable `knots` specifying the knots to use in a spline model.
`xlev`	Named list of factor levels (*excluding* ones coerced to factors in the model formula)
`grid`	A `data.frame` (provided by `ref_grid`) containing the predictor settings needed in the reference grid

Value

The `recover_data` method must return a `data.frame` containing all the variables that appear as predictors in the model, and attributes `"call"`, `"terms"`, `"predictors"`, and `"responses"`. (`recover_data.call` will provide these attributes.)

The `emm_basis` method should return a `list` with the following elements:

X The matrix of linear functions over `grid`, having the same number of rows as `grid` and the number of columns equal to the length of `bhat`.

bhat The vector of regression coefficients for fixed effects. This should *include* any `NA`s that result from rank deficiencies.

nbasis A matrix whose columns form a basis for non-estimable functions of beta, or a 1x1 matrix of `NA` if there is no rank deficiency.

V The estimated covariance matrix of `bhat`.

dffun A function of `(k, dfargs)` that returns the degrees of freedom associated with `sum(k * bhat)`.

dfargs A `list` containing additional arguments needed for `dffun`.

Details

To create a reference grid, the `ref_grid` function needs to reconstruct the data used in fitting the model, and then obtain a matrix of linear functions of the regression coefficients for a given grid of predictor values. These tasks are performed by calls to `recover_data` and `emm_basis` respectively. A vignette giving details and examples is available via vignette("extending", "emmeans")

To extend **emmeans**'s support to additional model types, one need only write S3 methods for these two functions. The existing methods serve as helpful guidance for writing new ones. Most of the work for `recover.data` can be done by its method for class `"call"`, providing the `terms` component and `na.action` data as additional arguments. Writing an `emm_basis` method is more involved, but the existing methods (e.g., `emmeans:::emm_basis.lm`) can serve as models. Certain `recover_data` and `emm_basis` methods are exported from **emmeans**. (To find out, do `methods("recover_data")`.) If your object is based on another model-fitting object, it may be that all that is needed is to call one of these exported methods and perhaps make modifications to the results. Contact the developer if you need others of these exported.

If the model has a multivariate response, `bhat` needs to be "flattened" into a single vector, and `X` and `V` must be constructed consistently.

In models where a non-full-rank result is possible (often you can tell by seeing if there is a `singular.ok` argument in the model-fitting function), `summary.emmGrid` and its relatives check the estimability of each prediction, using the `nonest.basis` function in the **estimability** package.

The models already supported are detailed in the "models" vignette. Some packages may provide additional **emmeans** support for its object classes.

Optional hooks

Some models may need something other than standard linear estimates and standard errors. If so, custom functions may be pointed to via the items `misc$estHook`, `misc$vcovHook` and `misc$postGridHook`. If just the name of the hook function is provided as a character string, then it is retrieved using `get`.

The `estHook` function should have arguments '`(object, do.se, tol,...)`' where `object` is the `emmGrid` object, `do.se` is a logical flag for whether to return the standard error, and `tol` is the tolerance for assessing estimability. It should return a matrix with 3 columns: the estimates, standard errors (`NA` when `do.se==FALSE`), and degrees of freedom (`NA` for asymptotic). The number of rows should be the same as '`object@linfct`'. The `vcovHook` function should have arguments '`(object, tol, ...)`' as described. It should return the covariance matrix for the estimates. Finally, `postGridHook`, if present, is called at the very end of `ref_grid`; it takes one argument, the constructed `object`, and should return a suitably modified `emmGrid` object.

See Also

Vignette on extending emmeans

`feedlot` *Feedlot data*

Description

This is an unbalanced analysis-of-covariance example, where one covariate is affected by a factor. Feeder calves from various herds enter a feedlot, where they are fed one of three diets. The weight of the animal at entry is the covariate, and the weight at slaughter is the response.

Usage

```
feedlot
```

Format

A data frame with 67 observations and 4 variables:

`herd` a factor with levels `9 16 3 32 24 31 19 36 34 35 33`, designating the herd that a feeder calf came from.

`diet` a factor with levels `Low Medium High`: the energy level of the diet given the animal.

`swt` a numeric vector: the weight of the animal at slaughter.

`ewt` a numeric vector: the weight of the animal at entry to the feedlot.

Details

The data arise from a Western Regional Research Project conducted at New Mexico State University. Calves born in 1975 in commercial herds entered a feedlot as yearlings. Both diets and herds are of interest as factors. The covariate, `ewt`, is thought to be dependent on `herd` due to different genetic backgrounds, breeding history, etc. The levels of `herd` ordered to similarity of genetic background.

Note: There are some empty cells in the cross-classification of `herd` and `diet`.

Source

Urquhart NS (1982) Adjustment in covariates when one factor affects the covariate. *Biometrics* 38, 651-660.

Examples

```
feedlot.lm <- lm(swt ~ ewt + herd*diet, data = feedlot)

# Obtain EMMs with a separate reference value of ewt for each
# herd. This reproduces the last part of Table 2 in the reference
emmeans(feedlot.lm,  ~ diet | herd,  cov.reduce = ewt ~ herd)
```

---

fiber *Fiber data*

Description

Fiber data from Montgomery Design (8th ed.), p.656 (Table 15.10). Useful as a simple analysis-of-covariance example.

Usage

```
fiber
```

Format

A data frame with 15 observations and 3 variables:

`machine` a factor with levels `A` `B` `C`. This is the primary factor of interest.

`strength` a numeric vector. The response variable.

`diameter` a numeric vector. A covariate.

Details

The goal of the experiment is to compare the mean breaking strength of fibers produced by the three machines. When testing this, the technician also measured the diameter of each fiber, and this measurement may be used as a concomitant variable to improve precision of the estimates.

Source

Montgomery, D. C. (2013) *Design and Analysis of Experiments* (8th ed.). John Wiley and Sons, ISBN 978-1-118-14692-7.

Examples

```
fiber.lm <- lm(strength ~ diameter + machine, data=fiber)
ref_grid(fiber.lm)

# Covariate-adjusted means and comparisons
emmeans(fiber.lm, pairwise ~ machine)
```

---

joint_tests *Compute joint tests of the terms in a model*

Description

This function produces an analysis-of-variance-like table based on linear functions of predictors in a model or `emmGrid` object. Specifically, the function constructs, for each combination of factors (or covariates reduced to two or more levels), a set of (interaction) contrasts via `contrast`, and then tests them using `test` with `joint = TRUE`. Optionally, one or more of the predictors may be used as by variable(s), so that separate tables of tests are produced for each combination of them.

Usage

```
joint_tests(object, by = NULL, show0df = FALSE, ...)
```

Arguments

`object`	a fitted model or an `emmGrid`. If a fitted model, it is replaced by `ref_grid(object, cov.reduce = range`
`by`	character names of `by` variables. Separate sets of tests are run for each combination of these.
`show0df`	logical value; if `TRUE`, results with zero numerator degrees of freedom are displayed, if `FALSE` they are skipped
`...`	additional arguments passed to `ref_grid` and `emmeans`

Details

In models with only factors, no covariates, we believe these tests correspond to "type III" tests a la **SAS**, as long as equal-weighted averaging is used and there are no estimability issues. When covariates are present and interact with factors, the results depend on how the covariate is handled in constructing the reference grid. See the example at the end of this documentation. The point that one must always remember is that `joint_tests` always tests contrasts among EMMs, in the context of the reference grid, whereas type III tests are tests of model coefficients – which may or may not have anything to do with EMMs or contrasts.

Value

a summary_emm object (same as is produced by summary.emmGrid). All effects for which there are no estimable contrasts are omitted from the results.

See Also

test

Examples

```
pigs.lm <- lm(log(conc) ~ source * factor(percent), data = pigs)

joint_tests(pigs.lm)                    ## will be same as type III ANOVA

joint_tests(pigs.lm, weights = "outer")  ## differently weighted

joint_tests(pigs.lm, by = "source")      ## separate joint tests of 'percent'

### Comparisons with type III tests
toy = data.frame(
    treat = rep(c("A", "B"), c(4, 6)),
    female = c(1, 0, 0, 1,   0, 0, 0, 1, 1, 0 ),
    resp = c(17, 12, 14, 19, 28, 26, 26, 34, 33, 27))
toy.fac = lm(resp ~ treat * factor(female), data = toy)
toy.cov = lm(resp ~ treat * female, data = toy)
# (These two models have identical fitted values and residuals)

joint_tests(toy.fac)

joint_tests(toy.cov)                     # ref grid uses mean(female) = 0.4
joint_tests(toy.cov, cov.reduce = FALSE)  # ref grid uses female = c(0, 1)
joint_tests(toy.cov, at = list(female = c(-1, 1)))  # center on intercept

# -- Compare with SAS output -- female as factor --
## Source          DF    Type III SS    Mean Square   F Value   Pr > F
## treat            1    488.8928571    488.8928571    404.60   <.0001
## female           1     78.8928571     78.8928571     65.29   0.0002
## treat*female     1      1.7500000      1.7500000      1.45   0.2741
#
# -- Compare with SAS output -- female as covariate --
## Source          DF    Type III SS    Mean Square   F Value   Pr > F
## treat            1    252.0833333    252.0833333    208.62   <.0001
## female           1     78.8928571     78.8928571     65.29   0.0002
## female*treat     1      1.7500000      1.7500000      1.45   0.2741
```

---

lsmeans *Wrappers for alternative naming of EMMs*

Description

These are wrappers for emmeans and realated functions to provide backward compatibility, or for users who may prefer to use other terminology than "estimated marginal means" – namely "least-squares means" or "predicted marginal means".

Usage

```
lsmeans(...)

pmmeans(...)

lstrends(...)

pmtrends(...)

lsmip(...)

pmmip(...)

lsm(...)

pmm(...)

lsmobj(...)

pmmobj(...)

lsm.options(...)

get.lsm.option(x, default = emm_defaults[[x]])
```

Arguments

...	Arguments passed to the corresponding em*xxxx* function
x	Character name of desired option
default	default value to return if x not found

Details

For each function with ls*xxxx* or pm*xxxx* in its name, the same function named em*xxxx* is called. Any estimator names or list items beginning with "em" are replaced with "ls" or "pm" before the results are returned

Value

The result of the call to em*xxxx*, suitably modified.

get.lsm.option and lsm.options remap options from and to corresponding options in the **lsmeans** options system.

Examples

```
pigs.lm <- lm(log(conc) ~ source + factor(percent), data = pigs)
lsmeans(pigs.lm, "source")
```

`make.tran` *Response-transformation extensions*

Description

The `make.tran` function creates the needed information to perform transformations of the response variable, including inverting the transformation and estimating variances of back-transformed predictions via the delta method. `make.tran` is similar to `make.link`, but it covers additional transformations. The result can be used as an environment in which the model is fitted, or as the `tran` argument in `update.emmGrid` (when the given transformation was already applied in an existing model).

Usage

```
make.tran(type = c("genlog", "power", "boxcox", "sympower", "asin.sqrt"),
  param = 1)
```

Arguments

`type`	The name of the transformation. See Details.
`param`	Numeric parameter needed for the transformation. Optionally, it may be a vector of two numeric values; the second element specifies an alternative base or origin for certain transformations. See Details.

Details

The functions `emmeans`, `ref_grid`, and related ones automatically detect response transformations that are recognized by examining the model formula. These are `log`, `log2`, `log10`, `sqrt`, `logit`, `probit`, `cauchit`, `cloglog`; as well as (for a response variable `y`) `asin(sqrt(y))`, `asinh(sqrt(y))`, and `sqrt(y) + sqrt(y+1)`. In addition, any constant multiple of these (e.g., `2*sqrt(y)`) is auto-detected and appropriately scaled (see also the `tran.mult` argument in `update.emmGrid`).

A few additional character strings may be supplied as the `tran` argument in `update.emmGrid`: `"identity"`, `"1/mu^2"`, `"inverse"`, `"reciprocal"`, `"asin.sqrt"`, and `"asinh.sqrt"`.

More general transformations may be provided as a list of functions and supplied as the `tran` argument as documented in `update.emmGrid`. The `make.tran` function returns a suitable list of functions for several popular transformations. Besides being usable with `update`, the user may use this list as an enclosing environment in fitting the model itself, in which case the transformation is auto-detected when the special name `linkfun` (the transformation itself) is used as the response transformation in the call. See the examples below.

Most of the transformations available in "make.tran" require a parameter, specified in `param`; in the following discussion, we use p to denote this parameter, and y to denote the response variable. The `type` argument specifies the following transformations:

`"genlog"` Generalized logarithmic transformation: $log(y + p)$, where $y > -p$

`"power"` Power transformation: y^p, where $y > 0$. When $p = 0$, `"log"` is used instead

`"boxcox"` The Box-Cox transformation (unscaled by the geometric mean): $(y^p - 1)/p$, where $y > 0$. When $p = 0$, $log(y)$ is used.

`"sympower"` A symmetrized power transformation on the whole real line: $abs(y)^p * sign(y)$. There are no restrictions on y, but we require $p > 0$ in order for the transformation to be monotone and continuous.

`"asin.sqrt"` Arcsin-square-root transformation: $sin^(-1)(y/p)^{1/2)}$. Typically, the parameter p is equal to 1 for a fraction, or 100 for a percentage.

The user may include a second element in `param` to specify an alternative origin (other than zero) for the `"power"`, `"boxcox"`, or `"sympower"` transformations. For example, '`type = "power",param = c(1.5, 4)`' specifies the transformation $(y - 4)^1.5$. In the `"genpower"` transformation, a second `param` element may be used to specify a base other than the default natural logarithm. For example, '`type = "genlog", param = c(.5, 10)`' specifies the $log10(y + .5)$ transformation.

For purposes of back-transformation, the '`sqrt(y) + sqrt(y+1)`' transformation is treated exactly the same way as '`2*sqrt(y)`', because both are regarded as estimates of $2\sqrt{\mu}$.

Value

A `list` having at least the same elements as those returned by `make.link`. The `linkfun` component is the transformation itself.

Note

We modify certain `make.link` results in transformations where there is a restriction on valid prediction values, so that reasonable inverse predictions are obtained, no matter what. For example, if a `sqrt` transformation was used but a predicted value is negative, the inverse transformation is zero rather than the square of the prediction. A side effect of this is that it is possible for one or both confidence limits, or even a standard error, to be zero.

Examples

```
# Fit a model using an oddball transformation:
bctran <- make.tran("boxcox", 0.368)
warp.bc <- with(bctran,
    lm(linkfun(breaks) ~ wool * tension, data = warpbreaks))
# Obtain back-transformed LS means:
emmeans(warp.bc, ~ tension | wool, type = "response")

## Not run:
# An existing model 'mod' was fitted with a log(y + 1) transformation...
mod.rg <- update(ref_grid(mod), tran = make.tran("genlog", 1))
emmeans(mod.rg, "treatment")

## End(Not run)
```

`MOats` *Oats data in multivariate form*

Description

This is the `Oats` dataset provided in the **nlme** package, but it is rearranged as one multivariate observation per plot.

Usage

```
MOats
```

Format

A data frame with 18 observations and 3 variables

`Variety` a factor with levels `Golden Rain`, `Marvellous`, `Victory`

`Block` an ordered factor with levels `VI` < `V` < `III` < `IV` < `II` < `I`

`yield` a matrix with 4 columns, giving the yields with nitrogen concentrations of 0, .2, .4, and .6.

Details

These data arise from a split-plot experiment reported by Yates (1935) and used as an example in Pinheiro and Bates (2000) and other texts. Six blocks were divided into three whole plots, randomly assigned to the three varieties of oats. The whole plots were each divided into 4 split plots and randomized to the four concentrations of nitrogen.

Source

The dataset `Oats` in the **nlme** package.

References

Pinheiro, J. C. and Bates D. M. (2000) *Mixed-Effects Models in S and S-PLUS*, Springer, New York. (Appendix A.15)

Yates, F. (1935) Complex experiments, *Journal of the Royal Statistical Society* Suppl. 2, 181-247

Examples

```
MOats.lm <- lm (yield ~ Block + Variety, data = MOats)
MOats.rg <- ref_grid (MOats.lm, mult.name = "nitro")
emmeans(MOats.rg, ~ nitro | Variety)
```

models *Models supported in* **emmeans**

Description

Documentation for models has been moved to a vignette. To access it, use `vignette("models", "emmeans")`.

neuralgia *Neuralgia data*

Description

These data arise from a study of analgesic effects of treatments of elderly patients who have neuralgia. Two treatments and a placebo are compared. The response variable is whether the patient reported pain or not. Researchers recorded the age and gender of 60 patients along with the duration of complaint before the treatment began.

Usage

```
neuralgia
```

Format

A data frame with 60 observations and 5 variables:

`Treatment` Factor with 3 levels `A`, `B`, and `P`. The latter is placebo

`Sex` Factor with two levels `F` and `M`

`Age` Numeric covariate – patient's age in years

`Duration` Numeric covariate – duration of the condition before beginning treatment

`Pain` Binary response factor with levels `No` and `Yes`

Source

Cai, Weijie (2014) *Making Comparisons Fair: How LS-Means Unify the Analysis of Linear Models*, SAS Institute, Inc. Technical paper 142-2014, page 12, http://support.sas.com/resources/papers/proceedings14/SAS060-2014.pdf

Examples

```
# Model and analysis shown in the SAS report:
neuralgia.glm <- glm(Pain ~ Treatment * Sex + Age, family = binomial(),
   data = neuralgia)
pairs(emmeans(neuralgia.glm, ~ Treatment, at = list(Sex = "F")),
    reverse = TRUE, type = "response", adjust = "bonferroni")
```

`nutrition` *Nutrition data*

Description

This observational dataset involves three factors, but where several factor combinations are missing. It is used as a case study in Milliken and Johnson, Chapter 17, p.202. (You may also find it in the second edition, p.278.)

Usage

```
nutrition
```

Format

A data frame with 107 observations and 4 variables:

`age` a factor with levels `1`, `2`, `3`, `4`. Mother's age group.

`group` a factor with levels `FoodStamps`, `NoAid`. Whether or not the family receives food stamp assistance.

`race` a factor with levels `Black`, `Hispanic`, `White`. Mother's race.

`gain` a numeric vector (the response variable). Gain score (posttest minus pretest) on knowledge of nutrition.

Details

A survey was conducted by home economists "to study how much lower-socioeconomic-level mothers knew about nutrition and to judge the effect of a training program designed to increase their knowledge of nutrition." This is a messy dataset with several empty cells.

Source

Milliken, G. A. and Johnson, D. E. (1984) *Analysis of Messy Data – Volume I: Designed Experiments*. Van Nostrand, ISBN 0-534-02713-7.

Examples

```
nutr.aov <- aov(gain ~ (group + age + race)^2, data = nutrition)

# Summarize predictions for age group 3
nutr.emm <- emmeans(nutr.aov, ~ race * group, at = list(age="3"))

emmip(nutr.emm, race ~ group)

# Hispanics seem exceptional; but this doesn't test out due to very sparse data
cld(nutr.emm, by = "group")
cld(nutr.emm, by = "race")
```

`oranges` *Sales of oranges*

Description

This example dataset on sales of oranges has two factors, two covariates, and two responses. There is one observation per factor combination.

Usage

```
oranges
```

Format

A data frame with 36 observations and 6 variables:

`store` a factor with levels `1 2 3 4 5 6`. The store that was observed.

`day` a factor with levels `1 2 3 4 5 6`. The day the observation was taken (same for each store).

`price1` a numeric vector. Price of variety 1.

`price2` a numeric vector. Price of variety 2.

`sales1` a numeric vector. Sales (per customer) of variety 1.

`sales2` a numeric vector. Sales (per customer) of variety 2.

Source

SAS sample dataset. Download from http://ftp.sas.com/samples/A56655.

References

Littell, R., Stroup W., Freund, R. (2002) *SAS For Linear Models* (4th edition). SAS Institute. ISBN 1-59047-023-0.

Examples

```
# Example on p.244 of Littell et al.
oranges.lm <- lm(sales1 ~ price1*day, data = oranges)
emmeans(oranges.lm, "day")

# Example on p.246 of Littell et al.
emmeans(oranges.lm, "day", at = list(price1 = 0))

# A more sensible model to consider, IMHO (see vignette("interactions"))
org.mlm <- lm(cbind(sales1, sales2) ~ price1 * price2 + day + store,
              data = oranges)
```

pigs *Effects of dietary protein on free plasma leucine concentration in pigs*

Description

A two-factor experiment with some observations lost

Usage

```
pigs
```

Format

A data frame with 29 observations and 3 variables:

source Source of protein in the diet (factor with 3 levels: fish meal, soybean meal, dried skim milk)

percent Protein percentage in the diet (numeric with 4 values: 9, 12, 15, and 18)

conc Concentration of free plasma leucine, in mcg/ml

Source

Windels HF (1964) PhD thesis, Univ. of Minnesota. (Reported as Problem 10.8 in Oehlert G (2000) *A First Course in Design and Analysis of Experiments*, licensed under Creative Commons, http://users.stat.umn.edu/~gary/Book.html.) Observations 7, 22, 23, 31, 33, and 35 have been omitted, creating a more notable imbalance.

Examples

```
pigs.lm <- lm(log(conc) ~ source + factor(percent), data = pigs)
emmeans(pigs.lm, "source")
```

plot.emmGrid *Plot an* emmGrid *or* summary_emm *object*

Description

Methods are provided to plot EMMs as side-by-side intervals, and optionally to display "comparison arrows" for displaying pairwise comparisons.

Usage

```
## S3 method for class 'emmGrid'
plot(x, y, type, intervals = TRUE, comparisons = FALSE,
  alpha = 0.05, adjust = "tukey", int.adjust = "none", ...)

## S3 method for class 'summary_emm'
plot(x, y, horizontal = TRUE, xlab, ylab, layout, ...)
```

Arguments

x	Object of class emmGrid or summary_emm
y	(Required but ignored)
type	Character value specifying the type of prediction desired (matching "linear.predictor", "link", or "response"). See details under summary.emmGrid.
intervals	Logical value. If TRUE, confidence intervals are plotted for each estimate.
comparisons	Logical value. If TRUE, "comparison arrows" are added to the plot, in such a way that the degree to which arrows overlap reflects as much as possible the significance of the comparison of the two estimates. (A warning is issued if this can't be done.)
alpha	The significance level to use in constructing comparison arrows
adjust	Character value: Multiplicity adjustment method for comparison arrows *only*.
int.adjust	Character value: Multiplicity adjustment method for the plotted confidence intervals *only*.
...	Additional arguments passed to update.emmGrid or dotplot
horizontal	Logical value specifying whether the intervals should be plotted horizontally or vertically
xlab	Character label for horizontal axis
ylab	Character label for vertical axis
layout	Numeric value passed to dotplot

Details

If any by variables are in force, the plot is divided into separate panels. These functions use the dotplot function, and thus require that the **lattice** package be installed. For "summary_emm" objects, the ... arguments in plot are passed *only* to dotplot, whereas for "emmGrid" objects, the object is updated using ... before summarizing and plotting.

In plots with comparisons = TRUE, the resulting arrows are only approximate, and in some cases may fail to accurately reflect the pairwise comparisons of the estimates – especially when estimates having large and small standard errors are intermingled in just the wrong way. Note that the maximum and minimum estimates have arrows only in one direction, since there is no need to compare them with anything higher or lower, respectively.

If adjust or int.adjust are not supplied, they default to the internal adjust setting saved in pairs(x) and x respectively (see update.emmGrid).

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
plot(warp.emm)
plot(warp.emm, by = NULL, comparisons = TRUE, adjust = "mvt",
     horizontal = FALSE)
```

`rbind.emmGrid` *Combine or subset* `emmGrid` *objects*

Description

These functions provide methods for `rbind` and `[` that may be used to combine `emmGrid` objects together, or to extract a subset of cases. The primary reason for doing this would be to obtain multiplicity-adjusted results for smaller or larger families of tests or confidence intervals.

Usage

```
## S3 method for class 'emmGrid'
rbind(..., deparse.level = 1, adjust = "bonferroni")

## S3 method for class 'emmGrid'
e1 + e2

## S3 method for class 'emmGrid'
x[i, adjust, drop.levels = TRUE, ...]
```

Arguments

`...`	In `rbind`, object(s) of class `emmGrid`. In `"["`, it is ignored.
`deparse.level`	(required but not used)
`adjust`	Character value passed to `update.emmGrid`
`e1`	An `emmGrid` object
`e2`	Another `emmGrid` object
`x`	An `emmGrid` object to be subsetted
`i`	Integer vector of indexes
`drop.levels`	Logical value. If `TRUE`, the `"levels"` slot in the returned object is updated to hold only the predictor levels that actually occur

Value

A revised object opf class `emmGrid`

The result of `e1 + e2` is the same as `rbind(e1, e2)`

Note

`rbind` throws an error if there are incompatibilities in the objects' coefficients, covariance structures, etc. But they are allowed to have different factors; a missing level `'.'` is added to factors as needed.

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
warp.rg <- ref_grid(warp.lm)

# Show only 3 of the 6 cases
summary(warp.rg[c(2,4,5)])

# Do all pairwise comparisons within rows or within columns,
# all considered as one faily of tests:
w.t <- pairs(emmeans(warp.rg, ~ wool | tension))
t.w <- pairs(emmeans(warp.rg, ~ tension | wool))
rbind(w.t, t.w, adjust = "mvt")
update(w.t + t.w, adjust = "fdr")  ## same as abve except for adjustment
```

`ref.grid` *Deprecated functions and arguments from* **lsmeans**

Description

Over time, some functions and internal structures have been revised or expanded, and as a consequence this necessitates renaming or reconceptualization of the functions it exports. This is a quick reference on how to get what you want if what worked in the past no longer does.

Usage

```
ref.grid(...)

recover.data(object, ...)

lsm.basis(object, ...)
```

Arguments

`...`	Arguments passed to other methods
`object`	An `emmGrid` object

List of deprecated functions and arguments

`as.stanfit()` We suggest using `as.mcmc()` instead, and plotting the results using functions in **bayesplot**. .

"`lsmobj`" class Both this and the "`ref.grid`" class have been replaced by the "`emmGrid`" class.

`ref.grid()` This has been replaced by `ref_grid()`, in hopes of reducing the chance that `ref.grid` will be mistaken as an S3 method for class `grid`.

"`ref.grid`" class Both this and the "`lsmobj`" class have been replaced by the "`emmGrid`" class.

`trend` The `trend` argument in `lsmeans` (now `emmeans`) is now deprecated. Use `emtrends()` instead.

`ref_grid` *Create a reference grid from a fitted model*

Description

Using a fitted model object, determine a reference grid for which estimated marginal means are defined. The resulting `ref_grid` object encapsulates all the information needed to calculate EMMs and make inferences on them.

Usage

```
ref_grid(object, at, cov.reduce = mean, mult.names, mult.levs,
  options = get_emm_option("ref_grid"), data, df, type,
  transform = c("none", "response", "mu", "unlink", "log"), nesting, ...)
```

Arguments

`object`	An object produced by a supported model-fitting function, such as `lm`. Many models are supported. See `vignette("models", "emmeans")`.
`at`	Optional named list of levels for the corresponding variables
`cov.reduce`	A function, logical value, or formula; or a named list of these. Each covariate *not* specified in `at` is reduced according to these specifications. See the section below on "Using `cov.reduce`".
`mult.names`	Character value: the name(s) to give to the pseudo-factor(s) whose levels delineate the elements of a multivariate response. If this is provided, it overrides the default name(s) used for `class(object)` when it has a multivariate response (e.g., the default is `"rep.meas"` for `"mlm"` objects).
`mult.levs`	A named list of levels for the dimensions of a multivariate response. If there is more than one element, the combinations of levels are used, in `expand.grid` order. The (total) number of levels must match the number of dimensions. If `mult.name` is specified, this argument is ignored.
`options`	If non-`NULL`, a named `list` of arguments to pass to `update.emmGrid`, just after the object is constructed.
`data`	A `data.frame` to use to obtain information about the predictors (e.g. factor levels). If missing, then `recover_data` is used to attempt to reconstruct the data.
`df`	Numeric value. This is equivalent to specifying `options(df = df)`. See `update.emmGrid`.
`type`	Character value. If provided, this is saved as the `"predict.type"` setting. See `update.emmGrid` and the section below on prediction types and transformations.
`transform`	Character value. If other than `"none"`, the reference grid is reconstructed via `regrid` with the given `transform` argument. See the section below on prediction types and transformations.

`nesting` If the model has nested fixed effects, this may be specified here via a character vector or named `list` specifying the nesting structure. Specifying `nesting` overrides any nesting structure that is automatically detected. See Details.

`...` Optional arguments passed to `emm_basis`, such as `vcov.` (see Details below) or options for certain models (see vignette("models", "emmeans")).

Details

The reference grid consists of combinations of independent variables over which predictions are made. Estimated marginal means are defined as these predictions, or marginal averages thereof. The grid is determined by first reconstructing the data used in fitting the model (see `recover_data`), or by using the `data.frame` provided in `data`. The default reference grid is determined by the observed levels of any factors, the ordered unique values of character-valued predictors, and the results of `cov.reduce` for numeric predictors. These may be overridden using `at`. See also the section below on recovering/overriding model information.

Value

An object of the S4 class `"emmGrid"` (see `emmGrid-class`). These objects encapsulate everything needed to do calculations and inferences for estimated marginal means, and contain nothing that depends on the model-fitting procedure.

Using `cov.reduce`

`cov.reduce` may be a function, logical value, formula, or a named list of these.

If a single function, it is applied to each covariate.

If logical and `TRUE`, `mean` is used. If logical and `FALSE`, it is equivalent to specifying '`function(x) sort(unique(x))`', and these values are considered part of the reference grid; thus, it is a handy alternative to specifying these same values in `at`.

If a formula (which must be two-sided), then a model is fitted to that formula using `lm`; then in the reference grid, its response variable is set to the results of `predict` for that model, with the reference grid as `newdata`. (This is done *after* the reference grid is determined.) A formula is appropriate here when you think experimental conditions affect the covariate as well as the response.

If `cov.reduce` is a named list, then the above criteria are used to determine what to do with covariates named in the list. (However, formula elements do not need to be named, as those names are determined from the formulas' left-hand sides.) Any unresolved covariates are reduced using `"mean"`.

Any `cov.reduce` specification for a covariate also named in `at` is ignored.

Recovering or overriding model information

Ability to support a particular class of `object` depends on the existence of `recover_data` and `emm_basis` methods – see extending-emmeans for details. The call `methods("recover_data")` will help identify these.

Data. In certain models, (e.g., results of `glmer.nb`), it is not possible to identify the original dataset. In such cases, we can work around this by setting `data` equal to the dataset used in fitting the model, or a suitable subset. Only the complete cases in `data` are used, so it may be necessary to exclude

some unused variables. Using `data` can also help save computing, especially when the dataset is large. In any case, `data` must represent all factor levels used in fitting the model. It *cannot* be used as an alternative to `at`. (Note: If there is a pattern of `NA`s that caused one or more factor levels to be excluded when fitting the model, then `data` should also exclude those levels.)

Covariance matrix. By default, the variance-covariance matrix for the fixed effects is obtained from `object`, usually via its `vcov` method. However, the user may override this via a `vcov.` argument, specifying a matrix or a function. If a matrix, it must be square and of the same dimension and parameter order of the fixed effects. If a function, must return a suitable matrix when it is called with `object` as its only argument.

Nested factors. Having a nesting structure affects marginal averaging in `emmeans` in that it is done separately for each level (or combination thereof) of the grouping factors. `ref_grid` tries to discern which factors are nested in other factors, but it is not always obvious, and if it misses some, the user must specify this structure via `nesting`; or later using `update.emmGrid`. The `nesting` argument may be a character vector or a named `list`. If a `list`, each name should be the name of a single factor in the grid, and its entry a character vector of the name(s) of its grouping factor(s). `nested` may also be a character value of the form `"factor1 %in% (factor2*factor3)"`. If there is more than one such specification, they may be appended separated by commas, or as separate elements of a character vector. For example, these specifications are equivalent: `nesting = list(state = "country", city = c("state", "country"),` `nesting = "state %in% country, city %in% (state*c` and `nesting = c("state %in% country)", "city %in% (state*country)")`.

Prediction types and transformations

There is a subtle difference between specifying '`type = "response"`' and '`transform = "response"`'. While the summary statistics for the grid itself are the same, subsequent use in `emmeans` will yield different results if there is a response transformation or link function. With '`type = "response"`', EMMs are computed by averaging together predictions on the *linear-predictor* scale and then back-transforming to the response scale; while with '`transform = "response"`', the predictions are already on the response scale so that the EMMs will be the arithmetic means of those response-scale predictions. To add further to the possibilities, *geometric* means of the response-scale predictions are obtainable via '`transform = "log", type = "response"`'.

Side effect

The most recent result of `ref_grid`, whether called directly or indirectly via `emmeans`, `emtrends`, or some other function that calls one of these, is saved in the user's environment as `.Last.ref_grid`. This facilitates checking what reference grid was used, or reusing the same reference grid for further calculations. This automatic saving is enabled by default, but may be disabled via '`emm_options(save.ref_grid = FALSE)`' and re-enabled by specifying `TRUE`.

See Also

Reference grids are of class `emmGrid`, and several methods exist for them – for example `summary.emmGrid`. Reference grids are fundamental to `emmeans`. Supported models are detailed in `vignette("models", "emmeans")`.

Examples

```
fiber.lm <- lm(strength ~ machine*diameter, data = fiber)
```

```
ref_grid(fiber.lm)
summary(.Last.ref_grid)

ref_grid(fiber.lm, at = list(diameter = c(15, 25)))

## Not run:
# We could substitute the sandwich estimator vcovHAC(fiber.lm)
# as follows:
summary(ref_grid(fiber.lm, vcov. = sandwich::vcovHAC))

## End(Not run)

# If we thought that the machines affect the diameters
# (admittedly not plausible in this example), then we should use:
ref_grid(fiber.lm, cov.reduce = diameter ~ machine)

# Multivariate example
MOats.lm = lm(yield ~ Block + Variety, data = MOats)
ref_grid(MOats.lm, mult.names = "nitro")
# Silly illustration of how to use 'mult.levs' to make comb's of two factors
ref_grid(MOats.lm, mult.levs = list(T=LETTERS[1:2], U=letters[1:2]))
```

`regrid` *Reconstruct a reference grid with a new transformation*

Description

The typical use of this function is to cause EMMs to be computed on a different scale, e.g., the back-transformed scale rather than the linear-predictor scale. In other words, if you want back-transformed results, do you want to average and then back-transform, or back-transform and then average?

Usage

```
regrid(object, transform = c("response", "mu", "unlink", "log", "none"),
  inv.log.lbl = "response", predict.type)
```

Arguments

`object`	An object of class `emmGrid`
`transform`	Character or logical value. If `"response"` or `"mu"`, the inverse transformation is applied to the estimates in the grid (but if there is both a link function and a response transformation, `"mu"` back-transforms only the link part); if `"log"`, the results are formulated as if the response had been `log`-transformed; if `"none"`, predictions thereof are on the same scale as in `object`, and any internal transformation information is preserved. For compatibility with past versions, `transform` may also be logical; `TRUE` is taken as `"response"`, and `FALSE` as `"none"`.

`inv.log.lbl` Character value. This applies only when `transform = "log"`, and is used to label the predictions if subsequently summarized with `type = "response"`.

`predict.type` Character value. If provided, the returned object is updated with the given type, e.g., `"response"`. See `update.emmGrid`.

Details

The `regrid` function reparameterizes an existing `ref.grid` so that its `linfct` slot is the identity matrix and its `bhat` slot consists of the estimates at the grid points. If `transform` is `TRUE`, the inverse transform is applied to the estimates. Outwardly, when `transform = "response"`, the result of `summary.emmGrid` after applying `regrid` is identical to the summary of the original object using '`type="response"`'. But subsequent EMMs or contrasts will be conducted on the new scale – which is the reason this function exists.

In cases where the degrees of freedom depended on the linear function being estimated, the d.f. from the reference grid are saved, and a kind of "containment" method is substituted in the returned object whereby the calculated d.f. for a new linear function will be the minimum d.f. among those having nonzero coefficients. This is kind of an *ad hoc* method, and it can over-estimate the degrees of freedom in some cases.

Value

An `emmGrid` object with the requested changes

Note

Another way to use `regrid` is to supply a `transform` argument to `ref_grid` (either directly of indirectly via `emmeans`). This is often a simpler approach if the reference grid has not already been constructed.

Examples

```
pigs.lm <- lm(log(conc) ~ source + factor(percent), data = pigs)

# This will yield EMMs as GEOMETRIC means of concentrations:
emmeans(pigs.lm, "source", type = "response")
# NOTE: pairs() of the above will be RATIOS of these results

# This will yield EMMs as ARITHMETIC means of concentrations:
emmeans(regrid(ref_grid(pigs.lm, transform = "response")), "source")
# Same thing, made simpler:
emmeans(pigs.lm, "source", transform = "response")
# NOTE: pairs() of the above will be DIFFERENCES of these results
```

str.emmGrid *Miscellaneous methods for* emmGrid *objects*

Description

Miscellaneous methods for emmGrid objects

Usage

```
## S3 method for class 'emmGrid'
str(object, ...)

## S3 method for class 'emmGrid'
print(x, ...)

## S3 method for class 'emmGrid'
vcov(object, ...)
```

Arguments

object	An emmGrid object
...	(required but not used)
x	An emmGrid object

Value

The vcov method returns a ymmetric matrix of variances and covariances for `predict.emmGrid(object, type = "lp")`

summary.emmGrid *Summaries, predictions, intervals, and tests for* emmGrid *objects*

Description

These are the primary methods for obtaining numerical or tabular results from an emmGrid object.

Usage

```
## S3 method for class 'emmGrid'
summary(object, infer, level, adjust, by, type, df, null,
  delta, side, ...)

## S3 method for class 'emmGrid'
predict(object, type, ...)

## S3 method for class 'emmGrid'
```

```
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'summary_emm'
x[...]

## S3 method for class 'emmGrid'
confint(object, parm, level = 0.95, ...)

test(object, null, ...)

## S3 method for class 'emmGrid'
test(object, null = 0, joint = FALSE, verbose = FALSE,
  rows, by, status = FALSE, ...)
```

Arguments

`object`	An object of class `"emmGrid"` (see emmGrid-class)
`infer`	A vector of one or two two logical values. The first determines whether confidence intervals are displayed, and the second determines whether *t* tests and *P* values are displayed. If only one value is provided, it is used for both.
`level`	Numerical value between 0 and 1. Confidence level for confidence intervals, if `infer[1]` is `TRUE`.
`adjust`	Character value naming the method used to adjust p values or confidence limits; or to adjust comparison arrows in `plot`. See the P-value adjustments section below.
`by`	Character name(s) of variables to use for grouping into separate tables. This affects the family of tests considered in adjusted *P* values.
`type`	Character: type of prediction desired. This only has an effect if there is a known transformation or link function. `"response"` specifies that the inverse transformation be applied. `"mu"` (or equivalently, `"unlink"`) is usually the same as `"response"`, but in the case where the model has both a link function and a response transformation, only the link part is back-transformed. Other valid values are `"link"`, `"lp"`, and `"linear.predictor"`; these are equivalent, and request that results be shown for the linear predictor, with no back-transformation. The default is `"link"`, unless the `"predict.type"` option is in force; see `emm_options`, and also the section below on transformations and links.
`df`	Numeric. If non-missing, a constant number of degrees of freedom to use in constructing confidence intervals and *P* values (`NA` specifies asymptotic results).
`null`	Numeric. Null hypothesis value(s), on the linear-predictor scale, against which estimates are tested. May be a single value used for all, or a numeric vector of length equal to the number of tests in each family (i.e., `by` group in the displayed table).
`delta`	Numeric value (on the linear-predictor scale). If zero, ordinary tests of significance are performed. If positive, this specifies a threshold for testing equivalence (using the TOST or two-one-sided-test method), non-inferiority, or non-

	superiority, depending on side. See Details for how the test statistics are defined.
side	Numeric or character value specifying whether the test is left-tailed (-1, "-", code"<", "left", or "nonsuperiority"); right-tailed (1, "+", ">", "right", or "noninferiority"); or two-sided (0, 2, "!=", "two-sided", "both", "equivalence", or "="). See the special section below for more details.
...	(Not used by summary.emmGrid or predict.emmGrid.) In as.data.frame.emmGrid, confint.emmGrid, and test.emmGrid, these arguments are passed to summary.emmGrid.
x	object of the given class
row.names	passed to as.data.frame
optional	passed to as.data.frame
parm	(Required argument for confint methods, but not used)
joint	Logical value. If FALSE, the arguments are passed to summary.emmGrid with infer=c(FALSE, TRUE). If joint = TRUE, a joint test of the hypothesis L beta = null is performed, where L is object@linfct and beta is the vector of fixed effects estimated by object@betahat. This will be either an F test or a chi-square (Wald) test depending on whether degrees of freedom are available. See also joint_tests.
verbose	Logical value. If TRUE and joint = TRUE, a table of the effects being tested is printed.
rows	Integer values. The rows of L to be tested in the joint test. If missing, all rows of L are used. If not missing, by variables are ignored.
status	logical. If TRUE, a note column showing status flags (for rank deficiencies and estimability issues) is displayed even when empty. If FALSE, the column is included only if there are such issues.

Details

summary.emmGrid is the general function for summarizing emmGrid objects. confint.emmGrid is equivalent to summary.emmGrid with infer = c(TRUE, FALSE). When called with joint = FALSE, test.emmGrid is equivalent to summary.emmGrid with infer = c(FALSE, TRUE).

With joint = TRUE, test.emmGrid calculates the Wald test of the hypothesis linfct %*% bhat = null, where linfct and bhat refer to slots in object (possibly subsetted according to by or rows). An error is thrown if any row of linfct is non-estimable. It is permissible for the rows of linfct to be linearly dependent, as long as null == 0, in which case a reduced set of contrasts is tested. Linear dependence and nonzero null cause an error.

Value

summary.emmGrid, confint.emmGrid, and test.emmGrid return an object of class "summary_emm", which is an extension of data.frame but with a special print method that displays it with custom formatting. For models fitted using MCMC methods, the result is typically a frequentist summary, based on the empirical mean and covariance matrix of the post.beta slot. A Bayesian summary may be obtained using as.mcmc.emmGrid and summarizing that result using tools for Bayesian estimation.

`predict` returns a vector of predictions for each row of `object@grid`.

The `as.data.frame` method returns a plain data frame, equivalent to `as.data.frame(summary(.))`.

Defaults

The `misc` slot in `object` contains default values for `by`, `infer`, `level`, `adjust`, `type`, `null`, `side`, and `delta`. These defaults vary depending on the code that created the object. The `update` method may be used to change these defaults. In addition, any options set using '`emm_options(summary = ...)`' will trump those stored in the object's `misc` slot.

Transformations and links

With `type = "response"`, the transformation assumed can be found in '`object@misc$tran`', and its label, for the summary is in '`object@misc$inv.lbl`'. Any t or z tests are still performed on the scale of the linear predictor, not the inverse-transformed one. Similarly, confidence intervals are computed on the linear-predictor scale, then inverse-transformed.

P-value adjustments

The `adjust` argument specifies a multiplicity adjustment for tests or confidence intervals. This adjustment always is applied *separately* to each table or sub-table that you see in the printed output (see `rbind.emmGrid` for how to combine tables).

The valid values of `adjust` are as follows:

`"tukey"` Uses the Studentized range distribution with the number of means in the family. (Available for two-sided cases only.)

`"scheffe"` Computes p values from the F distribution, according to the Scheffe critical value of $\sqrt{kF(k,d)}$, where d is the error degrees of freedom and k is (family size minus 1) for contrasts, and (number of estimates) otherwise. (Available for two-sided cases only.)

`"sidak"` Makes adjustments as if the estimates were independent (a conservative adjustment in many cases).

`"bonferroni"` Multiplies p values, or divides significance levels by the number of estimates. This is a conservative adjustment.

`"dunnettx"` Uses our own*ad hoc* approximation to the Dunnett distribution for a family of estimates having pairwise correlations of 0.5 (as is true when comparing treatments with a control with equal sample sizes). The accuracy of the approximation improves with the number of simultaneous estimates, and is much faster than `"mvt"`. (Available for two-sided cases only.)

`"mvt"` Uses the multivariate t distribution to assess the probability or critical value for the maximum of k estimates. This method produces the same p values and intervals as the default `summary` or `confint` methods to the results of `as.glht`. In the context of pairwise comparisons or comparisons with a control, this produces "exact" Tukey or Dunnett adjustments, respectively. However, the algorithm (from the **mvtnorm** package) uses a Monte Carlo method, so results are not exactly repeatable unless the same random-number seed is used (see `set.seed`). As the family size increases, the required computation time will become noticeable or even intolerable, making the `"tukey"`, `"dunnettx"`, or others more attractive.

`"none"` Makes no adjustments to the p values.

For tests, not confidence intervals, the Bonferroni-inequality-based adjustment methods in `p.adjust` are also available (currently, these include `"holm"`, `"hochberg"`, `"hommel"`, `"bonferroni"`, `"BH"`, `"BY"`, `"fdr"`, and `"none"`). If a `p.adjust.methods` method other than `"bonferroni"` or `"none"` is specified for confidence limits, the straight Bonferroni adjustment is used instead. Also, if an adjustment method is not appropriate (e.g., using `"tukey"` with one-sided tests, or with results that are not pairwise comparisons), a more appropriate method (usually `"sidak"`) is substituted.

In some cases, confidence and p-value adjustments are only approximate – especially when the degrees of freedom or standard errors vary greatly within the family of tests. The `"mvt"` method is always the correct one-step adjustment, but it can be very slow. One may use `as.glht` with methods in the **multcomp** package to obtain non-conservative multi-step adjustments to tests.

Testing nonsuperiority, noninferiority, or equivalence

When `delta = 0`, test statistics are of the usual form '`(estimate - null)/SE`', or notationally, $t = (Q - \theta_0)/SE$ where Q is our estimate of θ; then left, right, or two-sided p values are produced.

When `delta` is positive, the test statistic depends on `side` as follows.

Left-sided (nonsuperiority) $H_0 : \theta \geq \theta_0 + \delta$ versus $H_1 : \theta < \theta_0 + \delta$
$t = (Q - \theta_0 - \delta)/SE$
The p value is the lower-tail probability.

Right-sided (noninferiority) $H_0 : \theta \leq \theta_0 - \delta$ versus $H_1 : \theta > \theta_0 - \delta$
$t = (Q - \theta_0 + \delta)/SE$
The p value is the upper-tail probability.

Two-sided (equivalence) $H_0 : |\theta - \theta_0| \geq \delta$ versus $H_1 : |\theta - \theta_0| < \delta$
$t = (|Q - \theta_0| - \delta)/SE$
The p value is the *lower*-tail probability.

Non-estimable cases

When the model is rank-deficient, each row `x` of `object`'s `linfct` slot is checked for estimability. If `sum(x*bhat)` is found to be non-estimable, then the string `NonEst` is displayed for the estimate, and associated statistics are set to `NA`. The estimability check is performed using the orthonormal basis `N` in the `nbasis` slot for the null space of the rows of the model matrix. Estimability fails when $||Nx||^2/||x||^2$ exceeds `tol`, which by default is `1e-8`. You may change it via `emm_options` by setting `estble.tol` to the desired value.

Note

In doing testing and a transformation and/or link is in force, any `null` and/or `delta` values specified must always be on the scale of the linear predictor, regardless of the setting for 'type'. If `type = "response"`, the null value displayed in the summary table will be back-transformed from the value supplied by the user. But the displayed `delta` will not be changed, because there (usually) is not a natural way to back-transform it.

The default `show` method for `emmGrid` objects (with the exception of newly created reference grids) is `print(summary())`. Thus, with ordinary usage of `emmeans` and such, it is unnecessary to call `summary` unless there is a need to specify other than its default options.

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
warp.emm    # implicitly runs 'summary'

confint(warp.emm, by = NULL, level = .90)

# ------------------------------------------------------------
pigs.lm <- lm(log(conc) ~ source + factor(percent), data = pigs)
pigs.emm <- emmeans(pigs.lm, "percent", type = "response")
summary(pigs.emm)    # (inherits type = "response")

# For which percents is EMM non-inferior to 35, based on a 10% threshold?
# Note the test is done on the log scale even though we have type = "response"
test(pigs.emm, null = log(35), delta = log(1.10), side = ">")

test(contrast(pigs.emm, "consec"))

test(contrast(pigs.emm, "consec"), joint = TRUE)
```

---

update.emmGrid *Set or retrieve options for objects and summaries*

Description

Objects of class emmGrid contain several settings that affect primarily the defaults used by summary.emmGrid. This update method allows them to be changed more safely than by modifying this slot directly. In addition, the user may set or retrieve defaults for these settings.

Usage

```
## S3 method for class 'emmGrid'
update(object, ..., silent = FALSE)

emm_options(...)

get_emm_option(x, default = emm_defaults[[x]])

emm_defaults
```

Arguments

object	An emmGrid object
...	Options to be set. These must match a list of known options (see Details)
silent	Logical value. If FALSE (the default), a message is displayed if any options are not matched. If TRUE, no messages are shown.
x	Character value - the name of an option to be queried
default	Value to return if x is not found

Format

An object of class `list` of length 13.

Value

`update.emmGrid` returns an updated `emmGrid` object.

`emm_options` returns the current options (same as the result of '`getOption("emmeans")`') – invisibly, unless called with no arguments.

`get_emm_option` returns the currently stored option for `x`, or its default value if not found.

Details for `update.emmGrid`

In `update`, the names in `...` are partially matched against those that are valid, and if a match is found, it adds or replaces the current setting. The valid names are

`tran`, `tran2` (`list` or `character`) specifies the transformation which, when inverted, determines the results displayed by `summary.emmGrid`, `predict.emmGrid`, or `emmip` when `type="response"`. The value may be the name of a standard transformation from `make.link` or additional ones supported by name, such as `"log2"`; or, for a custom transformation, a `list` containing at least the functions `linkinv` (the inverse of the transformation) and `mu.eta` (the derivative thereof). The `make.tran` function returns such lists for a number of popular transformations. See the help page of `make.tran` for details as well as information on the additional named transformations that are supported. `tran2` is just like `tran` except it is a second transformation (i.e., a response transformation in a generalized linear model).

`tran.mult` Multiple for `tran`. For example, for the response transformation '`2*sqrt(y)`' (or '`sqrt(y) + sqrt(y + 1)`', for that matter), we should have `tran = "sqrt"` and `tran.mult = 2`. If absent, a multiple of 1 is assumed.

`estName` (`character`) is the column label used for displaying predictions or EMMs.

`inv.lbl` (`character`)) is the column label to use for predictions or EMMs when `type="response"`.

`by.vars` (`character`) vector or `NULL`) the variables used for grouping in the summary, and also for defining subfamilies in a call to `contrast`.

`pri.vars` (`character` vector) are the names of the grid variables that are not in `by.vars`. Thus, the combinations of their levels are used as columns in each table produced by `summary.emmGrid`.

`alpha` (numeric) is the default significance level for tests, in `summary.emmGrid` as well as `cld.emmGrid` and `plot.emmGrid` when '`intervals = TRUE`'

`adjust` (`character`)) is the default for the `adjust` argument in `summary.emmGrid`.

`estType` (`character`) is the type of the estimate. It should match one of '`c("prediction", "contrast", "pairs")`'. This is used along with `"adjust"` to determine appropriate adjustments to P values and confidence intervals.

`famSize` (integer) is the `nmeans` parameter for `ptukey` when `adjust="tukey"`.

`infer` (`logical` vector of length 2) is the default value of `infer` in `summary.emmGrid`.

`level` (numeric) is the default confidence level, `level`, in `summary.emmGrid`

`df` (numeric) overrides the default degrees of freedom with a specified single value.

`null` (numeric) null hypothesis for `summary` or `test` (taken to be zero if missing).

`side` (numeric or character) `side` specification for for `summary` or `test` (taken to be zero if missing).

`delta` (numeric) `delta` specification for `summary` or `test` (taken to be zero if missing).

`predict.type` **or** `type` (character) sets the default method of displaying predictions in `summary.emmGrid`, `predict.emmGrid`, and `emmip`. Valid values are `"link"` (with synonyms `"lp"` and `"linear"`), or `"response"`.

`avgd.over` (`character`) vector) are the names of the variables whose levels are averaged over in obtaining marginal averages of predictions, i.e., estimated marginal means. Changing this might produce a misleading printout, but setting it to `character(0)` will suppress the "averaged over" message in the summary.

`initMesg` (`character`) is a string that is added to the beginning of any annotations that appear below the `summary.emmGrid` display.

`methDesc` (`character`) is a string that may be used for creating names for a list of `emmGrid` objects.

`nesting` (Character or named `list`) specifies the nesting structure. See "Recovering or overriding model information" in the documentation for `ref_grid`. The current nesting structure is displayed by `str.emmGrid`.

(any slot name) If the name matches an element of `slotNames(object)`, that slot is replaced by the supplied value, if it is of the required class (otherwise an error occurs). Note that other options above are saved in the `misc` slot; hence, you probably don't want to replace that slot. The user must be very careful in replacing slots because they are interrelated; for example, the `levels` and `grid` slots must involve the same variable names, and the lengths and dimensions of `grid`, `linfct`, `bhat`, and `V` must conform.

Using `emm_options`

In `emm_options`, we may set or change the *default* values for the above options. These defaults are set separately for different contexts in which `emmGrid` objects are created, in a named list of option lists. Currently, the following main list entries are supported:

`ref_grid` A named `list` of defaults for objects created by `ref_grid`. This could affect other objects as well. For example, if `emmeans` is called with a fitted model object, it calls `ref_grid` and this option will affect the resulting `emmGrid` object.

`emmeans` A named `list` of defaults for objects created by `emmeans` or `emtrends`.

`contrast` A named `list` of defaults for objects created by `contrast.emmGrid` or `pairs.emmGrid`.

`summary` A named `list` of defaults used by the methods `summary.emmGrid`, `predict.emmGrid`, `test.emmGrid`, `confint.emmGrid`, and `emmip`. The only option that can affect the latter four is `"predict.method"`.

`graphics.engine` A character value matching `c("ggplot", "lattice")`, setting the default engine to use in `emmip` and `plot.emmGrid`. Defaults to `"ggplot"`.

`msg.interaction` A logical value controlling whether or not a message is displayed when `emmeans` averages over a factor involved in an interaction. It is probably not appropriate to do this, unless the interaction is weak. Defaults to `TRUE`.

`msg.nesting` A logical value controlling whether or not to display a message when a nesting structure is auto-detected. The existence of such a structure affects computations of EMMs. Sometimes, a nesting structure is falsely detected – namely when a user has omitted some main

effects but included them in interactions. This does not change the model fit, but it produces a different parameterization that is picked up when the reference grid is constructed. Defaults to TRUE.

Some other options have more specific purposes:

`estble.tol` Tolerance for determining estimability in rank-deficient cases. If absent, the value in `emm_defaults$estble.tol`) is used.

codesave.ref_grid Logical value of TRUE if you wish the latest reference grid created to be saved in `.Last.ref_grid`

Options for `lme4::lmerMod` **models** Options `lmer.df`, `disable.pbkrtest`, `pbkrtest.limit`, `disable.lmerTest`, and `lmerTest.limit` options affect how degrees of freedom are computed for `lmerMod` objects produced by the **lme4** package). See that section of the "models" vignette for details.

Examples

```
# Using an already-transformed response:
mypigs <- transform(pigs, logconc = log(pigs$conc))
mypigs.lm <- lm(logconc ~ source + factor(percent), data = mypigs)

# Reference grid that knows about the transformation:
mypigs.rg <- update(ref_grid(mypigs.lm), tran = "log",
                    predict.type = "response")
emmeans(mypigs.rg, "source")
## Not run:
emm_options(ref_grid = list(level = .90),
            contrast = list(infer = c(TRUE,FALSE)),
            estble.tol = 1e-6)
# Sets default confidence level to .90 for objects created by ref.grid
# AS WELL AS emmeans called with a model object (since it creates a
# reference grid). In addition, when we call 'contrast', 'pairs', etc.,
# confidence intervals rather than tests are displayed by default.

## End(Not run)

## Not run:
emm_options(disable.pbkrtest = TRUE)
# This forces use of asymptotic methods for lmerMod objects.
# Set to FALSE or NULL to re-enable using pbkrtest.

## End(Not run)

# See tolerance being used for determining estimability
get_emm_option("estble.tol")
```

`xtable.emmGrid` *Using* `xtable` *for EMMs*

Description

These methods provide support for the **xtable** package, enabling polished presentations of tabular output from emmeans and other functions.

Usage

```
## S3 method for class 'emmGrid'
xtable(x, caption = NULL, label = NULL, align = NULL,
  digits = 4, display = NULL, auto = FALSE, ...)

## S3 method for class 'summary_emm'
xtable(x, caption = NULL, label = NULL,
  align = NULL, digits = 4, display = NULL, auto = FALSE, ...)

## S3 method for class 'xtable_emm'
print(x, type = getOption("xtable.type", "latex"),
  include.rownames = FALSE, sanitize.message.function = footnotesize, ...)
```

Arguments

x	Object of class emmGrid
caption	Passed to xtableList
label	Passed to xtableList
align	Passed to xtableList
digits	Passed to xtableList
display	Passed to xtableList
auto	Passed to xtableList
...	Arguments passed to summary.emmGrid
type	Passed to print.xtable
include.rownames	Passed to print.xtable
sanitize.message.function	Passed to print.xtable

Details

The methods actually use xtableList, because of its ability to display messages such as those for P-value adjustments. These methods return an object of class "xtable_emm" – an extension of "xtableList". Unlike other xtable methods, the number of digits defaults to 4; and degrees of freedom and t ratios are always formatted independently of digits. The print method uses print.xtableList, and any ... arguments are passed there.

Value

The xtable methods return an xtable_emm object, for which its print method is print.xtable_emm .

Examples

```
pigsint.lm <- lm(log(conc) ~ source * factor(percent), data = pigs)
pigsint.emm <- emmeans(pigsint.lm, ~ percent | source)
xtable::xtable(pigsint.emm, type = "response")
```

Index