Package 'emmeans'

July 11, 2025

Type Package

Title Estimated Marginal Means, aka Least-Squares Means

Version 1.11.2

Date 2025-07-11

Depends R (>= 4.1.0)

Imports estimability (>= 1.4.1), graphics, methods, mvtnorm, numDeriv, stats, utils

Suggests bayesplot, bayestestR, biglm, brms, car, coda (>= 0.17), compositions, ggplot2, knitr, lattice, lme4, lmerTest (>= 2.0.32), logspline, MASS, mediation, mgcv, multcomp, multcompView, MuMIn, nlme, ordinal (>= 2014.11-12), pbkrtest (>= 0.4-1), rmarkdown, rsm, sandwich, scales, splines, testthat, tibble, xtable (>= 1.8-2)

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

URL https://rvlenth.github.io/emmeans/,https://rvlenth.github.io/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 other 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

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Russell V. Lenth [aut, cre, cph],
Balazs Banfai [ctb],
Ben Bolker [ctb],
Paul Buerkner [ctb],
Iago Giné-Vázquez [ctb],
Maxime Herve [ctb],
Maarten Jung [ctb],
Jonathon Love [ctb],
Fernando Miguez [ctb],
Julia Piaskowski [ctb],
Hannes Riebl [ctb],
Henrik Singmann [ctb]

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

Repository CRAN

Date/Publication 2025-07-11 15:20:06 UTC

Contents

`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.

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. For models not explicitly supported, it may still be possible to do basic post hoc analyses of them via the `qdrg` function.

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.

Graphs The `plot.emmGrid` method will display side-by-side confidence intervals for the estimates, and/or "comparison arrows" whereby the *P* values of pairwise differences can be observed by how much the arrows 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`.

`as.list.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

```
## S3 method for class 'emmGrid'
as.list(x, model.info.slot = FALSE, ...)

as.emm_list(object, ...)

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

Arguments

x	An emmGrid object
model.info.slot	Logical value: Include the model.info slot? Set this to TRUE if you want to preserve the original call and information needed by the submodel option. If FALSE, only the nesting information (if any) is saved
...	In as.emmGrid, additional arguments passed to update.emmGrid before returning the object. This argument is ignored in as.list.emmGrid
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.

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.list.emmGrid returns an object of class list.

as.emm_list returns an object of class emm_list.

as.emmGrid returns an object of class emmGrid. However, in fact, both as.emmGrid and as.emm_list check for an attribute in object to decide whether to return an emmGrid or emm_list) object.

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")
```

`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, likelihood,
  NE.include = FALSE, ...)

## S3 method for class 'emm_list'
as.mcmc(x, which = 1, ...)

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

## S3 method for class 'emm_list'
as.mcmc.list(x, which = 1, ...)
```

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.
`likelihood`	Character value or function. If given, simulations are made from the corresponding posterior predictive distribution. If not given, we obtain the posterior distribution of the parameters in `object`. See Prediction section below.

NE.include	Logical value. If TRUE, non-estimable columns are kept but returned as columns of NA values (this may create errors or warnings in subsequent analyses using, say, **coda**). If FALSE, non-estimable columns are dropped, and a warning is issued. (If all are non-estimable, an error is thrown.)
...	arguments passed to other methods
which	item in the emm_list to use

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.

Prediction

When likelihood is specified, it is used to simulate values from the posterior predictive distribution corresponding to the given likelihood and the posterior distribution of parameter values. Denote the likelihood function as $f(y|\theta, \phi)$, where y is a response, θ is the parameter estimated in object, and ϕ comprises zero or more additional parameters to be specified. If likelihood is a function, that function should take as its first argument a vector of θ values (each corresponding to one row of object@grid). Any ϕ values should be specified as additional named function arguments, and passed to likelihood via .... This function should simulate values of y.

A few standard likelihoods are available by specifying likelihood as a character value. They are:

"normal" The normal distribution with mean θ and standard deviation specified by additional argument sigma

"binomial" The binomial distribution with success probability $theta$, and number of trials specified by trials

"poisson" The Poisson distribution with mean $theta$ (no additional parameters)

"gamma" The gamma distribution with scale parameter θ and shape parameter specified by shape

Examples

```
if(requireNamespace("coda"))
    emm_example("as.mcmc-coda")
    # Use emm_example("as.mcmc-coda", list = TRUE) # to see just the code
```

`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 (but see note) - 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.

Note

While the data source claims that `noise` is measured in decibels, the values are implausible. I believe that these measurements are actually in tenths of dB (centibels?). Looking at the values in the dataset, note that every measurement ends in 0 or 5, and it is reasonable to believe that measurements are accurate to the nearest half of a decibel.

Source

The dataset was obtained from the Data and Story Library (DASL) at Carnegie-Mellon University. Apparently it has since been removed. The original dataset was altered by assigning meaningful names to the factors and sorting the observations in random order as if this were the run order of the experiment.

Examples

```
# (Based on belief that noise/10 is in decibel units)
noise.lm <- lm(noise/10 ~ 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` *Compact letter displays*

Description

A method for `multcomp::cld()` is provided for users desiring to produce compact-letter displays (CLDs). This method uses the Piepho (2004) algorithm (as implemented in the **multcompView** package) to generate a compact letter display of all pairwise comparisons of estimated marginal 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.

Usage

```
## S3 method for class 'emmGrid'
cld(object, details = FALSE, sort = TRUE, by,
  alpha = 0.05, Letters = c("1234567890", LETTERS, letters),
  reversed = decreasing, decreasing = FALSE, signif.sets = FALSE,
  delta = 0, ...)

## S3 method for class 'emm_list'
cld(object, ..., which = 1)
```

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`, `decreasing` Logical value (passed to `multcompView::multcompLetters`.) If `TRUE`, the order of use of the letters is reversed. Either `reversed` or `decreasing` may be specified, thus providing compatibility with both `multcompView::multcompLetters(..., reversed, ...)` and `multcomp::cld(..., decreasing, ...)`. In addition, if both `sort` and `reversed` are TRUE, the sort order of results is reversed.

`signif.sets` Logical value. If `FALSE` (and `delta = 0`), a 'traditional' compact-letter display is constructed with groupings representing sets of estimates that are not statistically different. If `TRUE`, the criteria are reversed so that two estimates sharing the same symbol test as significantly different. See also `delta`.

`delta` Numeric value passed to `test.emmGrid`. If this is positive, it is used as an equivalence threshold in the TOST procedure for two-sided equivalence testing. In the resulting compact letter display, two estimates share the same grouping letter only if they are found to be statistically equivalent – that is, groupings reflect actual *findings* of equivalence rather than failure to find a significant difference. When `delta` is nonzero, `signif.sets` is ignored.

`...` Arguments passed to `contrast` (for example, an `adjust` method)

`which` Which element of the `emm_list` object to process (If length exceeds one, only the first one is used)

Value

A `summary_emm` object showing the estimated marginal means plus an additional column labeled `.group` (when `signif.sets = FALSE`), `.signif.set` (when `signif.sets = TRUE`), or `.equiv.set` (when `delta > 0`).

Note

We warn that the default display encourages a poor practice in interpreting significance tests. Such CLDs are misleading because they visually group means with comparisons $P >$ `alpha` as though they are equal, when in fact we have only failed to prove that they differ. A better alternative if one wants to show groupings is to specify an equivalence threshold `delta`; then groupings will be based on actual findings of equivalence. Another way to display actual findings is to set `signif.sets = TRUE`, so that estimates in the same group are those found to be statistically *different*. Obviously, these different options require different interpretations of the results; the annotations and the label given the final column help guide how to assess the results.

As further alternatives, consider `pwpp` (graphical display of P values) or `pwpm` (matrix display).

References

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

Examples

```
if(requireNamespace("multcomp"))
    emm_example("cld-multcomp")
    # Use emm_example("cld-multcomp", list = TRUE) # to just list the code
```

---

comb_facs *Manipulate factors in a reference grid*

Description

These functions manipulate the levels of factors comprising a reference grid by combining factor levels, splitting a factor's levels into combinations of newly-defined factors, creating a grouping factor in which factor(s) levels are nested, or permuting the order of levels of a factor

Usage

```
comb_facs(object, facs, newname = paste(facs, collapse = "."),
  drop = FALSE, ...)

split_fac(object, fac, newfacs, ...)

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

add_submodels(object, ..., newname = "model")

permute_levels(object, fac, pos)
```

Arguments

object	An object of class emmGrid
facs	Character vector. The names of the factors to combine
newname	Character name of grouping factor to add (different from any existing factor in the grid)
drop	Logical value. If TRUE, any levels of the new factor that are dropped if all occurrences in the newly reconstructed object have weight zero. If FALSE, all levels are retained. (This argument is ignored if there is no .wgt. column in object@grid.)
...	arguments passed to other methods
fac	The name of a factor that is part of the grid in object
newfacs	A named list with the names of new factors and their levels. The names must not already exist in the object, and the product of the lengths of the levels must equal the number of levels of fac.
refname	Character name(s) of the reference factor(s)

`newlevs` Character vector or factor of the same length as that of the (combined) levels for `refname`. The grouping factor `newname` will have the unique values of `newlevs` as its levels. The order of levels in `newlevs` is the same as the order of the level combinations produced by `expand.grid` applied to the levels of `refname` – that is, the first factor's levels change the fastest and the last one's vary the slowest.

`pos` Integer vector consisting of some permutation of the sequence `1:k`, where `k` is the number of levels of `fac`. This determines which position each level of `fac` will occupy after the levels are permuted; thus, if the levels of `fac` are `A,B,C,D`, and `pos = c(3,1,2,4)`, then the permuted levels will be `B,C,A,D`.

Value

A modified object of class `emmGrid`

The `comb_facs` function

`comb_facs` combines the levels of factors into a single factor in the reference grid (similar to `interaction`). This new factor replaces the factors that comprise it.

Additional note: The choice of whether to drop levels or not can make a profound difference. If the goal is to combine factors for use in `joint_tests`, we advise *against* `drop = TRUE` because that might change the weights used in deriving marginal means. If combining factors in a nested structure, dropping unused cases can considerably reduce the storage required.

The `split_fac` function

The levels in `newfacs` are expanded via `expand.grid` into combinations of levels, and the factor `fac` is replaced by those factor combinations. Unlike `add_grouping`, this creates a crossed, rather than a nested structure. Note that the order of factor combinations is systematic with the levels of first factor in `newfacs` varying the fastest; and those factor combinations are assigned respectively to the levels of `fac` as displayed in `str(object)`.

The `add_grouping` function

This function adds a grouping factor to an existing reference grid or other `emmGrid` object, such that the levels of one or more existing factors (call them the reference factors) are mapped to a smaller number of levels of the new grouping factor. The reference factors are then nested in a new grouping factor named `newname`, and a new nesting structure `refname %in% newname`. This facilitates obtaining marginal means of the grouping factor, and contrasts thereof.

Additional notes: 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 desired order.

When `refname` specifies more than one factor, this can fundamentally (and permanently) change what is meant by the levels of those individual factors. For instance, in the `gwrg` example below, there are two levels of `wool` nested in each `prod`; and that implies that we now regard these as four different kinds of wool. Similarly, there are five different tensions (L, M, H in prod 1, and L, M in prod 2).

The `add_submodels` function

This function updates `object` with a named list of submodels specified in `...`. These are `rbind`ed together and the corresponding rows for each submodel are assigned a factor named `newname` with levels equal to the names in `...`. This facilitates comparing estimates obtained from different submodels. For this to work, the underlying model object must be of a class supported by the `submodel` argument of `update.emmGrid`.

The `permute_levels` function

This function permutes the levels of `fac`. The returned object has the same factors, same by variables, but with the levels of `fac` permuted. The order of the columns in `object@grid` may be altered.

NOTE: `fac` must not be nested in another factor. `permute_levels` throws an error when `fac` is nested.

NOTE: Permuting the levels of a numeric predictor is tricky. For example, if you want to display the new ordering of levels in `emmip()`, you must add the arguments `style = "factor"` and `nesting.order = TRUE`.

Examples

```
mtcars.lm <- lm(mpg ~ factor(vs)+factor(cyl)*factor(gear), data = mtcars)
(v.c.g <- ref_grid(mtcars.lm))
(v.cg <- comb_facs(v.c.g, c("cyl", "gear")))

# One use is obtaining a single test for the joint contributions of two factors:
joint_tests(v.c.g)

joint_tests(v.cg)

# undo the 'comb_facs' operation:
split_fac(v.cg, "cyl.gear", list(cyl = c(4, 6, 8), gear = 3:5))

IS.glm <- glm(count ~ spray, data = InsectSprays, family = poisson)
IS.emm <- emmeans(IS.glm, "spray")
IS.new <- split_fac(IS.emm, "spray", list(A = 1:2, B = c("low", "med", "hi")))
str(IS.new)

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")

### More than one reference factor
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
```

```
gwrg <- add_grouping(ref_grid(warp.lm),
    "prod",  c("tension", "wool"),  c(2, 1, 1,  1, 2, 1))
        # level combinations:         LA MA HA  LB MB HB

emmeans(gwrg, ~ wool * tension)   # some NAs due to impossible combinations

emmeans(gwrg, "prod")

## Using 'add_submodels' to compare adjusted and unadjusted means
fibint.lm <- lm(strength ~ machine * diameter, data = fiber)
fibsub <- add_submodels(emmeans(fibint.lm, "machine"),
    full = ~ ., additive = ~ . - machine:diameter, unadj = ~ machine)
emmeans(fibsub, pairwise ~ model | machine, adjust = "none")

# Permuting factor levels...
str(v.c.g)
str(permute_levels(v.c.g, "cyl", c(2,3,1)))
```

`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, scale = NULL, name = "contrast",
  options = get_emm_option("contrast"), type, adjust, simple,
  combine = FALSE, ratios = TRUE, parens, enhance.levels = TRUE, wts,
  ...)

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

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

## S3 method for class 'emmGrid'
weights(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 function of the same form, or 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, logical value, or list. 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, scale	Numeric vectors of the same length as each by group. The scale values, if supplied, multiply their respective linear estimates, and any offset values are added. Scalar values are also allowed. (These arguments are 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.
ratios	Logical value determining how log and logit transforms are handled. These transformations are exceptional cases in that there is a valid way to back-transform contrasts: differences of logs are logs of ratios, and differences of logits are odds ratios. If ratios = TRUE and summarized with type = "response", contrast results are back-transformed to ratios whenever we have true contrasts (coefficients sum to zero). For other transformations, there is no natural way to back-transform contrasts, so even when summarized with type = "response", contrasts are computed and displayed on the linear-predictor scale. Similarly, if ratios = FALSE, log and logit transforms are treated in the same way as any other transformation.
parens	character or NULL. If a character value, the labels for levels being contrasted are parenthesized if they match the regular expression in parens[1] (via grep). The default is emm_option("parens"). Optionally, parens may contain second and third elements specifying what to use for left and right parentheses (default "(" and ")"). Specify parens = NULL or parens = "a^" (which won't match anything) to disable all parenthesization.

enhance.levels	character or logical. If character, the levels of the named factors that are contrasted are enhanced by appending the name of the factor; e.g., if a factor named "trt" has levels A and B, a trt comparison is labeled trtA - trtB. If enhance.levels is logical, then if TRUE (the default), only factors with numeric levels are enhanced; and of course if FALSE, no levels are enhanced. The levels of by variables are not enhanced, and any names of factors that don't exist are silently ignored. To enhance the labels beyond what is done here, change them directly via levels<-.
wts	The wts argument for some contrast methods. You should omit this argument unless you want unequal wts. Otherwise we recommend specifying wts = NA which instructs that wts be obtained from object, *separately* for each by group. If numerical wts are specified, they must conform to the number of levels in each by group, and those same weights are used in each group.
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 combine is FALSE.

coef returns a data.frame containing the "parent" object's grid, along with columns named c.1, c.2, ... containing the contrast coefficients used to produce the linear functions embodied in the object. coef() only returns coefficients if object is the result of a call to contrast(), and the parent object is the object that was handed to contrast. This is most useful for understanding interaction contrasts.

weights returns the weights stored for each row of object, or a vector of 1s if no weights are saved.

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. Specifically 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. Any named elements of interaction are assigned to contrast methods; others are assigned in order of appearance in object@levels. The contrast factors in the resulting emmGrid object are ordered the same as in interaction.

interaction may be a character vector or list of valid contrast methods (as documented for the method argument). If the vector or list is shorter than the number needed, it is recycled. Alternatively, if the user specifies contrast = TRUE, the contrast specified in method is used for all factors involved.

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

### Custom contrast coefs (we already have wool as 'by' thus 3 means to contrast)
contrast(warp.emm, list(mid.vs.ends = c(-1,2,-1)/2, lo.vs.hi = c(1,0,-1)))

pairs(warp.emm)

# Effects (dev from mean) of the 6 factor combs, with enhanced levels:
contrast(warp.emm, "eff", by = NULL,
    enhance.levels = c("wool", "tension"))

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(tension = "poly", wool = "consec"),
                   by = NULL)
tw.emm          # see the estimates
coef(tw.emm)    # see the contrast coefficients

# Use of scale and offset
#   an unusual use of the famous stack-loss data...
mod <- lm(Water.Temp ~ poly(stack.loss, degree = 2), data = stackloss)
(emm <- emmeans(mod, "stack.loss", at = list(stack.loss = 10 * (1:4))))
# Convert results from Celsius to Fahrenheit:
confint(contrast(emm, "identity", scale = 9/5, offset = 32))
```

`contrast-methods` *Contrast families*

Description

Functions with an extension of `.emmc` provide for named contrast families. One of the standard ones documented here may be used, or the user may write such a function.

Usage

```
pairwise.emmc(levs, exclude = integer(0), include, ...)

revpairwise.emmc(levs, exclude = integer(0), include, ...)

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

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

opoly.emmc(levs, max.degree = min(6, k - 1), scores, exclude = integer(0),
  include, ...)

trt.vs.ctrl.emmc(levs, ref = 1, reverse = FALSE, exclude = integer(0),
  include, ...)

trt.vs.ctrl1.emmc(levs, ref = 1, ...)

trt.vs.ctrlk.emmc(levs, ref = length(levs), ...)

dunnett.emmc(levs, ref = 1, ...)

eff.emmc(levs, exclude = integer(0), include, wts = rep(1, length(levs)),
  ...)

del.eff.emmc(levs, exclude = integer(0), include, wts = rep(1,
```

```
  length(levs)), ...)

consec.emmc(levs, reverse = FALSE, exclude = integer(0), include, ...)

mean_chg.emmc(levs, reverse = FALSE, exclude = integer(0), include, ...)

helmert.emmc(levs, exclude = integer(0), include, ...)

nrmlz.emmc(levs, family, ...)

wtcon.emmc(levs, wts = rep(1, length(levs)), cmtype = "GrandMean", ...)

identity.emmc(levs, exclude = integer(0), include, ...)
```

Arguments

`levs`	Vector of factor levels
`exclude`	integer vector of indices, or character vector of levels to exclude from consideration. These levels will receive weight 0 in all contrasts. Character levels must exactly match elements of `levs`.
`include`	integer or character vector of levels to include (the complement of `exclude`). An error will result if the user specifies both `exclude` and `include`.
`...`	Additional arguments, passed to related methods as appropriate
`reverse`	Logical value to determine the direction of comparisons
`max.degree`	Integer specifying the maximum degree of polynomial contrasts
`scores`	Set of values of length `length(levs)` over which orthogonal polynomials are computed. The default scores are the consecutive integers `seq_along(levs)`. (If `exclude` or `include` are used, the default scores are subsetted accordingly; and if `scores` is specified, its length must be the same as that of the subsetted `levs`).
`ref`	Integer(s) or character(s) specifying which level(s) to use as the reference. Character values must exactly match elements of `levs` (including any enhancements – see examples)
`wts`	Optional weights to use with `eff.emmc` and `del.eff.emmc` contrasts. These default to equal weights. If `exclude` or `include` are specified, `wts` may be either the same length as `levs` or the length of the included levels. In the former case, weights for any excluded levels are set to zero. `wts` has no impact on the results unless there are at least three levels included in the contrast.
`family`	name of contrast family to use
`cmtype`	the `type` argument passed to `contrMat`

Details

Each standard 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 estimated marginal 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 and opoly.emmc generate orthogonal polynomial contrasts. poly.emmc uses equally-spaced factor levels; coefficients 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. On the other hand, opoly.emmc's coefficients are always normalized (sum of squares equals 1), but allows the user to choose alternate reference points in scores, as in the contr.poly function. In both cases, 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. *Note* in all of these functions, it is illegal to have any overlap between the ref levels and the exclude levels. If any is found, an error is thrown.

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.

nrmlz.emmc is a wrapper that can be used with any other .emmc function that will normalize the contrast coefficients so that the sum of its squares equals 1. Just provide the root name of the function in family, along with any other arguments to pass to it.

wtcon.emmc generates weighted contrasts based on the function contrMat function in the **multcomp** package, using the provided type as documented there. If the user provides wts, they have to conform to the length of levs; however, if wts is not specified, contrast will fill-in what is required, and usually this is safer (especially when by != NULL which usually means that the weights are different in each by group).

identity.emmc simply returns the identity matrix (as a data frame), minus any columns specified in exclude. It is potentially useful in cases where a contrast function must be specified, but none is desired.

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.

Note

Caution is needed in cases where the user alters the ordering of results (e.g., using the the "[...]" operator), because the contrasts generated depend on the order of the levels provided. For example, suppose `trt.vs.ctrl1` contrasts are applied to two by groups with levels ordered (Ctrl, T1, T2) and (T1, T2, Ctrl) respectively, then the contrasts generated will be for (T1 - Ctrl, T2 - Ctrl) in the first group and (T2 - T1, Ctrl - T1) in the second group, because the first level in each group is used as the reference level.

Examples

```
warp.lm <- lm(breaks ~ wool*tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
contrast(warp.emm, "poly")
contrast(warp.emm, "trt.vs.ctrl", ref = "M")
## Not run:
## Same when enhanced labeling is used:
contrast(warp.emm, "trt.vs.ctrl",
         enhance.levels = "tension", ref = "tensionM")
## End(Not run)

# Comparisons with grand mean
contrast(warp.emm, "eff")
# Comparisons with a weighted grand mean
contrast(warp.emm, "eff", wts = c(2, 5, 3))

# Compare only low and high tensions
# Note pairs(emm, ...) calls contrast(emm, "pairwise", ...)
pairs(warp.emm, exclude = 2)
# (same results using exclude = "M" or include = c("L","H") or include = c(1,3))

# Same contrasts as above but with normalized contrast coefficients
contrast(warp.emm, "nrmlz", family = "pairwise", include = c(1, 3))

### Setting up a custom contrast function
revhelmert.emmc <- function(levs, ...) {
    M <- as.data.frame(contr.helmert(levs)[rev(seq_along(levs)), ])
    names(M) <- paste(rev(levs)[-1],"vs later")
    attr(M, "desc") <- "reverse Helmert contrasts"
    M
}
contrast(warp.emm, "revhelmert")

## Not run:
# See what is used for polynomial contrasts with 6 levels
emmeans:::poly.emmc(1:6)

## End(Not run)
```

`eff_size` *Calculate Cohen effect sizes and confidence bounds thereof*

Description

Standardized effect sizes are typically calculated using pairwise differences of estimates, divided by the SD of the population providing the context for those effects. This function calculates effect sizes from an `emmGrid` object, and confidence intervals for them, accounting for uncertainty in both the estimated effects and the population SD.

Usage

```
eff_size(object, sigma, edf, method = "pairwise", ...)
```

Arguments

`object`	an `emmGrid` object, typically one defining the EMMs to be contrasted. If instead, `class(object) == "emm_list"`, such as is produced by `emmeans(model, pairwise ~ treatment)`, a message is displayed; the contrasts already therein are used; and `method` is replaced by `"identity"`.
`sigma`	numeric scalar, value of the population SD.
`edf`	numeric scalar that specifies the equivalent degrees of freedom for the `sigma`. This is a way of specifying the uncertainty in `sigma`, in that we regard our estimate of `sigma^2` as being proportional to a chi-square random variable with `edf` degrees of freedom. (`edf` should not be confused with the `df` argument that may be passed via `...` to specify the degrees of freedom to use in t statistics and confidence intervals.)
`method`	the contrast method to use to define the effects. This is passed to `contrast` after the elements of `object` are scaled.
`...`	Additional arguments passed to `contrast`

Details

Any by variables specified in `object` will remain in force in the returned effects, unless overridden in the optional arguments.

For models having a single random effect, such as those fitted using `lm`; in that case, the `stats::sigma` and `stats::df.residual` functions may be useful for specifying `sigma` and `edf`. For models with more than one random effect, `sigma` may be based on some combination of the random-effect variances.

Specifying `edf` can be rather unintuitive but is also relatively uncritical; but the smaller the value, the wider the confidence intervals for effect size. The value of `sqrt(2/edf)` can be interpreted as the relative accuracy of `sigma`; for example, with `edf = 50`, $\sqrt{(2/50)} = 0.2$, meaning that `sigma` is accurate to plus or minus 20 percent. Note in an example below, we tried two different `edf` values as kind of a bracketing/sensitivity-analysis strategy. A value of `Inf` is allowable, in which case you are assuming that `sigma` is known exactly. Obviously, this narrows the confidence intervals for the effect sizes – unrealistically if in fact `sigma` is unknown.

Value

an emmGrid object containing the effect sizes

Computation

This function uses calls to regrid to put the estimated marginal means (EMMs) on the log scale. Then an extra element is added to this grid for the log of sigma and its standard error (where we assume that sigma is uncorrelated with the log EMMs). Then a call to contrast subtracts log{sigma} from each of the log EMMs, yielding values of log(EMM/sigma). Finally, the results are re-gridded back to the original scale and the desired contrasts are computed using method. In the log-scaling part, we actually rescale the absolute values and keep track of the signs.

Note

The effects are always computed on the scale of the *linear-predictor*; any response transformation or link function is completely ignored. If you wish to base the effect sizes on the response scale, it is *not* enough to replace object with regrid(object), because this back-transformation changes the SD required to compute effect sizes.

Paired data: Be careful with paired-data situations, where Cohen's d is typically referenced to the SD of the *paired differences* rather than the *residual* SD. You may need to enlarge sigma by a factor of sqrt(2) to obtain comparable results with other software.

Disclaimer: There is substantial disagreement among practitioners on what is the appropriate sigma to use in computing effect sizes; or, indeed, whether *any* effect-size measure is appropriate for some situations. The user is completely responsible for specifying appropriate parameters (or for failing to do so).

Cohen effect sizes do not even exist for generalized linear models or other models lacking an additive residual error term.

The examples here illustrate a sobering message that effect sizes are often not nearly as accurate as you may think.

Examples

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

emm <- emmeans(fiber.lm, "machine")
eff_size(emm, sigma = sigma(fiber.lm), edf = df.residual(fiber.lm))

# or equivalently:
eff_size(pairs(emm), sigma(fiber.lm), df.residual(fiber.lm), method = "identity")


### Mixed model example:
if (require(nlme)) withAutoprint({
  Oats.lme <- lme(yield ~ Variety + factor(nitro),
                  random = ~ 1 | Block / Variety,
                  data = Oats)
  # Combine variance estimates
  VarCorr(Oats.lme)
  (totSD <- sqrt(214.4724 + 109.6931 + 162.5590))
```

```
  # I figure edf is somewhere between 5 (Blocks df) and 51 (Resid df)
  emmV <- emmeans(Oats.lme, ~ Variety)
  eff_size(emmV, sigma = totSD, edf = 5)
  eff_size(emmV, sigma = totSD, edf = 51)
}, spaced = TRUE)
```

---

emm *Support for* multcomp::glht

Description

These functions and methods provide an interface between **emmeans** and the multcomp::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 are passed to other methods. You may also include a which argument; see Details.
object	An object of class emmGrid or emm_list

Value

emm returns an object of an intermediate class for which there is a multcomp::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 for emm

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

If the specifications in ... would result in a list (i.e., an emm_list object), then by default, only the last element of that list is passed to glht. However, if ... contains a which argument consisting of integer values, the list elements with those indexes are selected and combined and passed on to glht. No checking is done on whether the indexes are valid, and the keyword which must be spelled-out.

Details for as.glht

When no by variable is in force, we obtain a glht object; otherwise it is a glht_list. The latter is defined in **emmeans**, not **multcomp**, and is simply a list of glht objects. 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

```
if(require(multcomp, quietly = TRUE))
    emm_example("glht-multcomp")
    # Use emm_example("glht-multcomp", list = TRUE) # to see just the code
```

---

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, offset, ...,
  tran)
```

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 Overview section below. Specifying . as the only factor name creates a list of specifications for all model terms. **Note:** We recommend *against* using two-sided formulas; see the note below for contr.
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. **Note 2:**: We recommend *against* using this argument; obtaining means and obtaining contrasts are two different things, and it is best to do them in separate steps, using the `contrast` function for the contrasts.

`options` If non-`NULL`, a named `list` of arguments to pass to `update.emmGrid`, just after the object is constructed. (Options may also be included in `...`; see the 'options' section below.)

`weights` Character value, numeric vector, or numeric matrix specifying weights to use in averaging predictions. See "Weights" section below. Also, if `object` is not already a reference grid, `weights` (if it is character) is passed to `ref_grid` as `wt.nuis` in case nuisance factors are specified. We can override this by specifying `wt.nuis` explicitly. This more-or-less makes the weighting of nuisance factors consistent with that of primary factors.

`offset` Numeric vector or scalar. If specified, this adds an offset to the predictions, or overrides any offset in the model or its reference grid. If a vector of length differing from the number of rows in the result, it is subsetted or cyclically recycled.

`...` When `object` is not already a `"emmGrid"` object, these arguments are passed to `ref_grid`. Common examples are `at`, `cov.reduce`, `data`, `type`, `regrid`, `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. These arguments may also be used in lieu of `options`. See the 'Options' section below.

`tran` Placeholder to prevent it from being included in `....` If non-missing, it is added to 'options'. See the 'Options' section.

Details

Users should also consult the documentation for `ref_grid`, because many important options for EMMs are implemented there, via the `...` argument.

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`, and `pairs.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.

Overview

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 (and we don't recommend it), 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.

We now also allow using `.` in `specs`. If this is done, we run `joint_tests` on the side to determine all relevant model terms, then replace `specs` with a corresponding list of specifications. This is a convenience, but it can create a sizeable `emm_list` object and it is coded rather inefficiently. While it is permissible to include contrasts via a `contr` argument or formula left-hand-side, we recommend instead doing this in a follow-up test with `contrast.emm_list`. *Caution*: In models with nested fixed effects, using `.` creates results where nested factors interact with nesting factors; in those cases, any contrasts you specify will go across nests, which is likely not what is desired.

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.

Counterfactuals

Counterfactual reference grids (see the documentation for ref_grid) contain pairs of imputed and actual factor levels, and are handled in a special way. For starters, the weights argument is ignored and we always use "cells" weights. Our understanding is that if factors A, B are specified as counterfactuals, the marginal means for A should still be the same as if A were the only counterfactual. Accordingly, in computing these marginal means, we exclude all cases where B != actual_B, because if A were the only counterfactual, B will stay at its actual level. We also take special pains to "remember" information about actual and imputed levels of counterfactuals so that appropriate results are obtained when emmeans is applied to a previous emmeans result.

Offsets

Unlike in ref_grid, an offset need not be scalar. If not enough values are supplied, they are cyclically recycled. For a vector of offsets, it is important to understand that the ordering of results goes with the first name in specs varying fastest. If there are any by factors, those vary slower than all the primary ones, but the first by variable varies the fastest within that hierarchy. See the examples.

Options and ...

Arguments that could go in options may instead be included in ..., typically, arguments such as type, infer, etc. that in essence are passed to summary.emmGrid. Arguments in both places are overridden by the ones in ....

There is a danger that ... arguments could partially match those used by both ref_grid and update.emmGrid, creating a conflict. If these occur, usually they can be resolved by providing complete (or at least longer) argument names; or by isolating non-ref_grid arguments in options; or by calling ref_grid separately and passing the result as object. See a not-run example below.

Also, when specs is a two-sided formula, or contr is specified, there is potential confusion concerning which ... arguments apply to the means, and which to the contrasts. When such confusion is possible, we suggest doing things separately (a call to emmeans with no contrasts, followed by a call to contrast). We treat adjust as a special case: it is applied to the emmeans results *only* if there are no contrasts specified, otherwise it is passed only to contrast.

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")
```

```
# 'adjust' argument ignored in emmeans, passed to contrast part...
emmeans (warp.lm, poly ~ tension | wool, adjust = "sidak")

# 'adjust' argument NOT ignored ...
emmeans (warp.lm, ~ tension | wool, adjust = "sidak")

# Get all sets of EMMs for this model
( allsets <- emmeans(warp.lm, ".") )
contrast(allsets, "eff")    # all effects


## Not run:
  ### Offsets: Consider a silly example:
  emmeans(warp.lm, ~ tension | wool, offset = c(17, 23, 47)) @ grid
  # note that offsets are recycled so that each level of tension receives
  # the same offset for each wool.
  # But using the same offsets with ~ wool | tension will probably not
  # be what you want because the ordering of combinations is different.

## End(Not run)
```

`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` (see **coda**), `cld.emmGrid` (see **multcomp**), `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, PIs = FALSE,
```

```
  style, engine = get_emm_option("graphics.engine"), plotit = TRUE,
  nesting.order = FALSE, ...)

emmip_ggplot(emms, style = "factor", dodge = 0.1, xlab = labs$xlab,
  ylab = labs$ylab, tlab = labs$tlab, facetlab = "label_context", scale,
  dotarg = list(shape = "circle"), linearg = list(linetype = "solid"),
  CIarg = list(lwd = 2, alpha = 0.5), PIarg = list(lwd = 1.25, alpha =
  0.33), col, ...)

emmip_lattice(emms, style = "factor", xlab = labs$xlab, ylab = labs$ylab,
  tlab = labs$tlab, pch = c(1, 2, 6, 7, 9, 10, 15:20), lty = 1,
  col = NULL, ...)
```

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), `predict.emmGrid`, `emmip_ggplot`, or `emmip_lattice`.
`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`. In addition, the user may specify `type = "scale"` to create a transformed scale for the vertical axis based on `object`'s response transformation or link function.
`CIs`	Logical value. If `TRUE`, confidence intervals (or HPD intervals for Bayesian models) are added to the plot (works only with `engine = "ggplot"`).
`PIs`	Logical value. If `TRUE`, prediction intervals are added to the plot (works only with `engine = "ggplot"`). This is allowed only if the underlying model family is `"gaussian"`. If both `CIs` and `PIs` are `TRUE`, the prediction intervals will be somewhat longer, lighter, and thinner than the confidence intervals. Additional parameters to `predict.emmGrid` (e.g., `sigma`) may be passed via `....` For Bayesian models, PIs require `frequentist = TRUE` and a value for `sigma`.
`style`	Optional character value. This has an effect only when the horizontal variable is a single numeric variable. If `style` is unspecified or `"numeric"`, the horizontal scale will be numeric and curves are plotted using lines (and no symbols). With `style = "factor"`, the horizontal variable is treated as the levels of a factor (equally spaced along the horizontal scale), and curves are plotted using lines and symbols. When the horizontal variable is character or factor, or a combination of more than one predictor, `"factor"` style is always used.
`engine`	Character value matching `"ggplot"` (default), `"lattice"`, or `"none"`. The graphics engine to be used to produce the plot. These require, respectively,

	the **ggplot2** or **lattice** package to be installed. Specifying "none" is equivalent to setting plotit = FALSE.
plotit	Logical value. If TRUE, a graphical object is returned; if FALSE, a data.frame is returned containing all the values used to construct the plot.
nesting.order	Logical value. If TRUE, factors that are nested are presented in order according to their nesting factors, even if those nesting factors are not present in formula. If FALSE, only the variables in formula are used to order the variables.
emms	A data.frame created by calling emmip with plotit = FALSE. Certain variables and attributes are expected to exist in this data frame; see the section detailing the rendering functions.
dodge	Numerical amount passed to ggplot2::position_dodge by which points and intervals are offset so they do not collide.
xlab, ylab, tlab	Character labels for the horizontal axis, vertical axis, and traces (the different curves), respectively. The emmip function generates these automatically and provides therm via the labs attribute, but the user may override these if desired.
facetlab	Labeller for facets (when by variables are in play). Use "label_value" to show just the factor levels, or "label_both" to show both the factor names and factor levels. The default of "label_context" decides which based on how many by factors there are. See the documentation for ggplot2::label_context.
scale	If not missing, an object of class scales::trans specifying a (usually) nonlinear scaling for the vertical axis. For example, scales = scales::log_trans() specifies a logarithmic scale. For fine-tuning purposes, additional arguments to ggplot2::scale_y_continuous may be included in ....
dotarg	list of arguments passed to geom_point to customize appearance of points
linearg	list of arguments passed to geom_line to customize appearance of lines
CIarg, PIarg	lists of arguments passed to geom_linerange to customize appearance of intervals. (Note: the linetype aesthetic defaults to "solid" under the hood)
col	With emmip_ggplot, this adds color = col (not colour) to all of the *arg lists. This is intended for setting a common color for everything, such as a black-and-white plot. With emmip_lattice, col specifies the colors to use for each group, recycled as needed. If not specified, the default trellis colors are used.
pch	(Lattice only) The plotting characters to use for each group (i.e., levels of trace.factors). They are recycled as needed.
lty	(Lattice only) The line types to use for each group. Recycled as needed.

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`.

Rendering functions

The functions `emmip_ggplot` and `emmip_lattice` are called when `plotit == TRUE` to render the plots; but they may also be called later on an object saved via `plotit = FALSE` (or `engine = "none"`). The functions require that `emms` contains variables `xvar`, `yvar`, and `tvar`, and attributes `"labs"` and `"vars"`. Confidence intervals are plotted if variables `LCL` and `UCL` exist; and prediction intervals are plotted if `LPL` and `UPL` exist. Finally, it must contain the variables named in `attr(emms, "vars")`.

In `emmip_ggplot`, colors, linetypes, and shapes are all assigned to groups (according to `tvar`) unless overridden. So, for example, one may have different symbols for each group by simply specifying `dotarg = list()`.

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
# ... with added confidence intervals and some formatting changes
emmip(noise.lm, type ~ side * size, CIs = TRUE,
    CIarg = list(lwd = 1, alpha = 1, color = "cyan"),
    dotarg = list(color = "black"))

# Create a black-and-white version of above with different linetypes
# (Let the linetypes and symbols default to the palette)
emmip(noise.lm, type ~ side * size, CIs = TRUE, col = "black",
      linearg = list(), dotarg = list(size = 2), CIarg = list(alpha = 1)) +
    ggplot2::theme_bw()

# 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)

# Example for the 'style' argument
fib.lm = lm(strength ~ machine * sqrt(diameter), data = fiber)
fib.rg = ref_grid(fib.lm, at = list(diameter = c(3.5, 4, 4.5, 5, 5.5, 6)^2))
emmip(fib.rg, machine ~ diameter)   # curves (because diameter is numeric)
emmip(fib.rg, machine ~ diameter, style = "factor")  # points and lines

# For an example using extra ggplot2 code, see 'vignette("messy-data")',
# in the section on nested models.

### Options with transformations or link functions
neuralgia.glm <- glm(Pain ~ Treatment * Sex + Age, family = binomial(),
                     data = neuralgia)

# On link scale:
emmip(neuralgia.glm, Treatment ~ Sex)

# On response scale:
emmip(neuralgia.glm, Treatment ~ Sex, type = "response")

# With transformed axis scale and custom scale divisions
emmip(neuralgia.glm, Treatment ~ Sex, type = "scale",
    breaks = seq(0.10, 0.90, by = 0.10))
```

`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 = diag(length(bhat)), df = NA, dffun,
  dfargs = list(), post.beta = matrix(NA), nesting = NULL, se.bhat,
  se.diff, ...)
```

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.
nesting	Nesting specification as in ref_grid. This is ignored if model.info is supplied.
se.bhat, se.diff	Alternative way of specifying V. If se.bhat is specified, V is constructed using se.bhat, the standard errors of bhat, and se.diffs, the standard errors of its pairwise differences. se.diff should be a vector of length k*(k-1)/2 where k is the length of se.bhat, and its elements should be in the order 12,13,...,1k,23,...2k,.... If se.diff is missing, V is computed as if the bhat are independent.
...	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.

The functions qdrg and emmobj are close cousins, in that they both produce emmGrid objects. When starting with summary statistics for an existing grid, emmobj is more useful, while qdrg is more useful when starting from an unsupported fitted model.

Value

An emmGrid object

See Also

qdrg, an alternative that is useful when starting with a fitted model not supported in **emmeans**.

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.emm <- 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")

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

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

### Create an emmobj from means and SEs
### (This illustration reproduces the MOats example for Variety = "Victory")
means = c(71.50000,  89.66667, 110.83333, 118.50000)
semeans = c(5.540591, 6.602048, 8.695358, 7.303221)
sediffs = c(7.310571,  9.894724,  7.463615, 10.248306,  4.935698,  8.694507)
foo = emmobj(bhat = means, se.bhat = semeans, se.diff = sediffs,
             levels = list(nitro = seq(0, .6, by = .2)), df = 10)
plot(foo, comparisons = TRUE)
```

`emm_example` *Run or list additional examples*

Description

This function exists so as to provide cleaner-looking examples in help files when it must be run conditionally on another package. Typically we want to run the code (`run = TRUE` is the default), or otherwise just list it on the console (`list = TRUE`).

Usage

```
emm_example(name, run = !list, list = FALSE, ...)
```

Arguments

`name`	Character name of file to run. We look for a file with this name (with `".R"` appended) in the system files provided with **emmeans**.
`run`	Logical choosing whether or not to run the example code
`list`	Logical choosing whether or not to list the example code
`...`	Used only by the developer

Examples

```
# List an example
emm_example("qdrg-biglm", list = TRUE)

# Run an example
```

```
if (require(biglm))
    emm_example("qdrg-biglm")
```

---

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. Several methods for this class are provided, as detailed below. Typically, these methods just quietly do the same thing as their emmGrid methods, using the first element of the list. You can specify which to select a different element, or just run the corresponding emmGrid method on object[[k]].

Usage

```
## S3 method for class 'emm_list'
contrast(object, ..., which = NULL)

## S3 method for class 'emm_list'
pairs(x, ..., which = NULL)

## S3 method for class 'emm_list'
test(object, ..., which = seq_along(object))

## S3 method for class 'emm_list'
confint(object, ..., which = seq_along(object))

## S3 method for class 'emm_list'
plot(x, ..., which = 1)

## S3 method for class 'emm_list'
coef(object, ..., which = NULL)

## S3 method for class 'emm_list'
linfct(object, ..., which = seq_along(object))

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

## S3 method for class 'emm_list'
summary(object, ..., which = seq_along(object))

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

```
## S3 method for class 'emm_list'
as.data.frame(x, ...)

## S3 method for class 'summary_eml'
as.data.frame(x, row.names = NULL, optional = FALSE,
  which, ...)
```

Arguments

object, x	an object of class emm_list
...	additional arguments passed to corresponding emmGrid method. In addition, the user may include a logical argument drop that is akin to drop and the argument of the same name in subscripting matrices and data frames. When drop is TRUE (the default), then when the result is a list of length 1, the list structure is removed.
which	integer vector specifying which elements to select; if NULL, we try to guess which elements make sense. Usually, this is all elements having names that start with 'em' or 'ls', or the first element if no matches are found. However, in coef.emm_list, these are the ones we *exclude*.
row.names, optional	Required arguments of as.data.frame, ignored

Value

a list of objects returned by the corresponding emmGrid method (thus, often, another emm_list object). However, if which has length 1, the one result is not wrapped in a list.

summary.emm_list returns an object of class summary_eml, which is a list of summary_emm objects.

The as.data.frame methods return a single data frame via as.data.frame(rbind(x)). See also rbind.emm_list and as.data.frame.emmGrid

Note

The plot method uses only the first element of which; the others are ignored.

No export option is provided for printing an emm_list (see print.emmGrid). If you wish to export these objects, you must do so separately for each element in the list.

Examples

```
mod <- lm(conc ~ source, data = pigs)
obj <- emmeans(mod, pairwise ~ source)

linfct(obj)

coef(obj)     # done only for the contrasts

contrast(obj, "consec")  # done only for the means

contrast(obj, "eff", drop = FALSE)   # kept as a list
```

emm_options *Set or change emmeans options*

Description

Use emm_options to set or change various options that are used in the **emmeans** package. These options are set separately for different contexts in which emmGrid objects are created, in a named list of option lists.

Usage

```
emm_options(..., disable)

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

with_emm_options(..., expr)

emm_defaults
```

Arguments

...	Option names and values (see Details)
disable	If non-missing, this will reset all options to their defaults if disable tests TRUE (but first save them for possible later restoration). Otherwise, all previously saved options are restored. This is important for bug reporting; please see the section below on reproducible bugs. When disable is specified, the other arguments are ignored.
x	Character value - the name of an option to be queried
default	Value to return if x is not found
expr	Expression to evaluate. If missing, the last element of ... is used.

Format

An object of class list of length 22.

Details

emmeans's options are stored as a list in the system option "emmeans". Thus, emm_options(foo = bar) is the same as options(emmeans = list(..., foo = bar)) where ... represents any previously existing options. The list emm_defaults contains the default values in case the corresponding element of system option emmeans is NULL.

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"`.

`allow.na.levs` A logical value that if `TRUE` (the default), allows `NA` to be among the levels of a factor. Older versions of **emmeans** did not allow this. So if problems come up (say in a messy dataset that includes incomplete cases), try setting this to `FALSE`.

`sep` A character value to use as a separator in labeling factor combinations. Such labels are potentially used in several places such as `contrast` and `plot.emmGrid` when combinations of factors are compared or plotted. The default is `" "`.

`parens` Character vector that determines which labels are parenthesized when they are contrasted. The first element is a regular expression, and the second and third elements are used as left and right parentheses. See details for the `parens` argument in `contrast`. The default will parenthesize labels containing the four arithmetic operators, using round parentheses.

`cov.keep` The default value of `cov.keep` in `ref_grid`. Defaults to `"2"`, i.e., two-level covariates are treated like factors.

`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`.

`rg.limit` An integer value setting a limit on the number of rows in a newly constructed reference grid. This is checked based on the number of levels of the factors involved; but it excludes the levels of any multivariate responses because those are not yet known. The reference grid consists of all possible combinations of the predictors, and this can become huge if there are several factors. An error is thrown if this limit is exceeded. One can use the `nuisance` argument of `ref_grid` to collapse on nuisance factors, thus making the grid smaller. Defaults to 10,000.

`simplify.names` A logical value controlling whether to simplify (when possible) names in the model formula that refer to datasets – for example, should we simplify a predictor name like "`data$trt`" to just "`trt`"? Defaults to `TRUE`.

`opt.digits` A logical value controlling the precision with which summaries are printed. If `TRUE` (default), the number of digits displayed is just enough to reasonably distinguish estimates from the ends of their confidence intervals; but always at least 3 digits. If `FALSE`, the system value `getOption("digits")` is used.

`back.bias.adj` A logical value controlling whether we try to adjust bias when back-transforming. If `FALSE`, we use naive back transformation. If `TRUE` *and* `sigma` *is available and valid*, a second-order adjustment is applied to estimate the mean on the response scale. A warning is issued if no valid `sigma` is available

`enable.submodel` A logical value. If `TRUE`, enables support for selected model classes to implement the `submodel` option. If `FALSE`, this support is disabled. Setting this option to `FALSE` could save excess memory consumption.

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.

`save.ref_grid` Logical value of `TRUE` if you wish the latest reference grid created to be saved in `.Last.ref_grid`. The default is `FALSE`.

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.

Value

`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.

`with_emm_options()` temporarily sets the options in `...`, then evaluates `try(expr)` and returns the result.

Reproducible bugs

Most options set display attributes and such that are not likely to be associated with bugs in the code. However, some other options (e.g., `cov.keep`) are essentially configuration settings that may affect how/whether the code runs, and the settings for these options may cause subtle effects that may be hard to reproduce. Therefore, when sending a bug report, please create a reproducible example and make sure the bug occurs with all options set at their defaults. This is done by preceding it with `emm_options(disable = TRUE)`.

By the way, `disable` works like a stack (LIFO buffer), in that `disable = TRUE` is equivalent to `emm_options(saved.opts = emm_options())` and `emm_options(disable = FALSE)` is equivalent to `options(emmeans = get_emm_option("saved.opts"))`. To completely erase all options, use `options(emmeans = NULL)`

See Also

`update.emmGrid`

Examples

```
## 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")

## Not run:
# Set all options to their defaults
emm_options(disable = TRUE)
# ... and perhaps follow with code for a minimal reproducible bug,
#     which may include emm_options() clls if they are pertinent ...

# restore options that had existed previously
emm_options(disable = FALSE)

## End(Not run)


# Illustration of how 'opt.digits' affects the results of print()
# Note that the returned value is printed with the default setting (opt.digits = TRUE)
pigs.lm <- lm(inverse(conc) ~ source, data = pigs)
with_emm_options(opt.digits = FALSE, print(emmeans(pigs.lm, "source")))
```

`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(object, specs, var, delta.var = 0.001 * rng, max.degree = 1, ...)
```

Arguments

`object`	A supported model object (*not* a reference grid)

specs	Specifications for what marginal trends are desired – as in emmeans. If specs is missing or NULL, emmeans is not run and the reference grid for specified trends is returned.
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 1/1000 of the range of var over the dataset.
max.degree	Integer value. The maximum degree of trends to compute (this is capped at 5). If greater than 1, an additional factor degree is added to the grid, with corresponding numerical derivatives of orders 1, 2, ..., max.degree as the estimates.
...	Additional arguments passed to ref_grid or emmeans as appropriate. See Details.

Details

The function works by constructing reference grids for object with various values of var, and then calculating difference quotients of predictions from those reference grids. Finally, emmeans is called with the given specs, thus computing marginal averages as needed of the difference quotients. Any ... arguments are passed to the ref_grid and emmeans; examples of such optional arguments include optional arguments (often mode) that apply to specific models; ref_grid options such as data, at, cov.reduce, mult.names, nesting, or transform; and emmeans options such as weights (but please avoid trend or offset.

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 object 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.

Note

In earlier versions of emtrends, the first argument was named model rather than object. (The name was changed because of potential mis-matching with a mode argument, which is an option for

several types of models.) For backward compatibility, `model` still works *provided all arguments are named*.

It is important to understand that trends computed by `emtrends` are *not* equivalent to polynomial contrasts in a parallel model where `var` is regarded as a factor. That is because the model `object` here is assumed to fit a smooth function of `var`, and the estimated trends reflect *local* behavior at particular value(s) of `var`; whereas when `var` is modeled as a factor and polynomial contrasts are computed, those contrasts represent the *global* pattern of changes over *all* levels of `var`.

See the `pigs.poly` and `pigs.fact` examples below for an illustration. The linear and quadratic trends depend on the value of `percent`, but the cubic trend is constant (because that is true of a cubic polynomial, which is the underlying model). The cubic contrast in the factorial model has the same P value as for the cubic trend, again because the cubic trend is the same everywhere.

See Also

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)))

# Obtaining a reference grid
mtcars.lm <- lm(mpg ~ poly(disp, degree = 2) * (factor(cyl) + factor(am)), data = mtcars)

# Center trends at mean disp for each no. of cylinders
mtcTrends.rg <- emtrends(mtcars.lm, var = "disp",
                         cov.reduce = disp ~ factor(cyl))
summary(mtcTrends.rg)  # estimated trends at grid nodes
emmeans(mtcTrends.rg, "am", weights = "prop")


### Higher-degree trends ...

pigs.poly <- lm(conc ~ poly(percent, degree = 3), data = pigs)
emt <- emtrends(pigs.poly, ~ degree | percent, "percent", max.degree = 3,
                at = list(percent = c(9, 13.5, 18)))
       # note: 'degree' is an extra factor created by 'emtrends'

summary(emt, infer = c(TRUE, TRUE))

# Compare above results with poly contrasts when 'percent' is modeled as a factor ...
pigs.fact <- lm(conc ~ factor(percent), data = pigs)
emm <- emmeans(pigs.fact, "percent")
```

```
contrast(emm, "poly")
# Some P values are comparable, some aren't! See Note in documentation
```

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

Description

This documents some functions and methods that may be useful to package developers wishing to add support for **emmeans** for their model objects.A user or package developer may add **emmeans** support for a model class by writing `recover_data` and `emm_basis` methods for that class. (Users in need for a quick way to obtain results for a model that is not supported may be better served by the `qdrg` function.) There are several other exported functions that may be useful. See the "xtending" vignette for more details.

Usage

```
recover_data(object, ...)

## S3 method for class 'call'
recover_data(object, trms, na.action, data = NULL,
  params = "pi", frame, pwts, addl.vars, ...)

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

.recover_data(object, ...)

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

.emm_register(classes, pkgname)

.std.link.labels(fam, misc)

.combine.terms(...)

.aovlist.dffun(k, dfargs)

.cmpMM(X, weights = rep(1, nrow(X)), assign = attr(X$qr, "assign"))

.get.excl(levs, exc, inc)

.get.offset(terms, grid)

.my.vcov(object, vcov. = .statsvcov, ...)

.all.vars(expr, retain = c("\\$", "\\[\\[", "\\]\\]", "'", "\""),
  ...)
```

```
.diag(x, nrow, ncol)

.num.key(levs, key)

.emm_vignette(css = system.file("css", "clean-simple.css", package =
  "emmeans"), highlight = NULL, ...)

.hurdle.support(cmu, cshape, cp0, cmean, zmu, zshape, zp0)

.zi.support(zmu, zshape, zp0)
```

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. For example, a spline model may involve a local variable `knots` that is not a predictor, but its value is needed to fit the model. Names of parameters not actually used are harmless, and the default value `"pi"` (the only numeric constant in base R) is provided in case the model involves it. An example involving splines may be found at https://github.com/rvlenth/emmeans/issues/180.
`frame`	Optional `data.frame`. Many model objects contain the model frame used when fitting the model. In cases where there are no predictor transformations, this model frame has all the original predictor values and so is usable for recovering the data. Thus, if `frame` is non-missing and `data` is `NULL`, a check is made on `trms` and if there are no function calls, we use `data = frame`. This can be helpful because it provides a modicum of security against the possibility that the original data used when fitting the model has been altered or removed.
`pwts`	Optional vector of prior weights. Typically, this may be obtained from the fitted `model` via `weights(model)`. If this is provided, it is used to set weights as long as it is non-`NULL` and the same length as the number of rows of the data.
`addl.vars`	Character value or vector specifying additional predictors to include in the reference grid. These must be names of variables that exist, or you will get an error. This may be useful if you need to do additional computations later on that depend on these variables; e.g., bias adjustments for random slopes of variables not among the fixed predictors.
`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

`classes` Character names of one or more classes to be registered. The package must contain the functions `recover_data.foo` and `emm_basis.foo` for each class `foo` listed in `classes`.

`pkgname` Character name of package providing the methods (usually should be the second argument of `.onLoad`)

`fam` Result of call to `family(object)`

`misc` A `list` intended for the `@misc` slot of an `emmGrid` object

`k, dfargs` Arguments to `.aovlist.dffun`, which is made available as a convenience to developers providing support similar to that provided for `aovlist` objects

`X, weights, assign`
Arguments for `.cmpMM`, which compacts a model matrix `X` into a much smaller matrix that has the same row space. Specifically, it returns the R portion of its QR decomposition. If `X` is already of class `qr`, it is used directly. `weights` should be the weights used in the model fit, and `assign` is used for unravelling any pivoting done by `qr`.

`levs, key` The `.num.key` function returns the numeric indices of the levels in `levs` to the set of all levels in `key`

`exc, inc` Arguments for `.get.excl` which is useful in writing `.emmc` functions for generating contrast coefficients, and supports arguments `exclude` or `include` for excluding or specifying which levels to use.

`terms` A `terms` component

`vcov.` Function or matrix that returns a suitable covariance matrix. The default is `.statsvcov` which is `stats::vcov`. The `.my.vcov` function should be called in place of `vcov`, and it supports the user being able to specify a different matrix or function via the optional `vcov.` argument.

`expr, retain` Arguments for `.all.vars`, which is an alternative to `all.vars` that has special provisions for retaining the special characters in `retain`, thus allowing model specifications like `y ~ data$trt * df[["dose"]]`

`x, nrow, ncol` Arguments for `.diag`, which is an alternative to `diag` that lacks its idiosyncrasy of returning an identity matrix when `x` is of length 1.

`css, package, highlight`
Arguments for `.emm_vignette`, which is a clean and simple alternative to such as `html_document` for use as the output style of a Markdown file. All the vignettes in the **emmeans** package use this output style.

`cmu, zmu` In `.hurdle.support` and `.zi.support`, these specify a vector of back-transformed estimates for the count and zero model, respectively

`cshape, zshape` Shape parameter for the count and zero model, respectively

`cp0, zp0` Function of `(mu, shape)` for computing Prob(Y = 0) for the count and zero model, respectively

`cmean` Function of `(mu, shape)` for computing the mean of the count model. Typically, this just returns `mu`

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 NAs 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.

.recover_data and .emm_basis are hidden exported versions of recover_data and emm_basis, respectively. They run in **emmeans**'s namespace, thus providing access to all existing methods.

.std.link.llabels returns a modified version of misc with the appropriate information included corresponding to the information in fam

combine.terms returns a terms object resulting from combining all the terms or formulas in ....

.get.offset returns the values, based on grid, of any offset component in terms

.hurdle.support returns a matrix with 3 rows containing the estimated mean responses and the differentials wrt cmu and zmu, resp.

.zi.support returns a matrix with 2 rows containing the estimated probabilities of 0 and the differentials wrt mu. See the section on hurdle and zero-inflated models.

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("xtending", "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.

Communication between methods

If the `recover_data` method generates information needed by `emm_basis`, that information may be incorporated by creating a `"misc"` attribute in the returned recovered data. That information is then passed as the `misc` argument when `ref_grid` calls `emm_basis`.

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 equal '`nrow(linfct(object)`'. 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.

Registering S3 methods for a model class

The `.emm_register` function is provided as a convenience to conditionally register your S3 methods for a model class, `recover_data.foo` and `emm_basis.foo`, where `foo` is the class name. Your package should implement an `.onLoad` function and call `.emm_register` if **emmeans** is installed. See the example.

Support for Hurdle and Zero-inflated models

The functions `.hurdle.support` and `.zi.support` help facilitate calculations needed to estimate the mean response (count model and zero model combined) of these models. `.hurdle.support` returns a matrix of three rows. The first is the estimated mean for a hurdle model, and the 2nd and 3rd rows are differentials for the count and zero models, which needed for delta-method calculations. To use these, regard the `@linfct` slot as comprising two sets of columns, for the count and zero models respectively. To do the delta method calculations, multiply the rows of the count part by its differentials times `link$mu.eta` evcaluated at that part of the linear predictor. Do the same for the zero part, using its differentials and `mu.eta`. If the resulting matrix is **A**, then the covariance of the mean response is **AVA'** where **V**is the `@V` slot of the object.

The function `zi.support` works the same way, only it is much simpler, and is used to estimate the probability of 0 and its differential for either part of a zero-inflated model or hurdle model.

See the code for `emm_basis.zeroinfl` and `emm_basis.hurdle` for how these are used with models fitted by the **pscl** package.

Note

Without an explicit `data` argument, `recover_data` returns the *current version* of the dataset. If the dataset has changed since the model was fitted, then this will not be the data used to fit the model. It is especially important to know this in simulation studies where the data are randomly generated or permuted, and in cases where several datasets are processed in one step (e.g., using `dplyr`). In those cases, users should be careful to provide the actual data used to fit the model in the `data` argument.

See Also

Vignette on extending emmeans

Examples

```
## Not run:
#--- If your package provides recover_data and emm_grid methods for class 'mymod',
#--- put something like this in your package code -- say in zzz.R:
  .onLoad <- function(libname, pkgname) {
    if (requireNamespace("emmeans", quietly = TRUE))
      emmeans::.emm_register("mymod", pkgname)
  }

## End(Not run)
```

`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)
```

---

hpd.summary *Summarize an emmGrid from a Bayesian model*

Description

This function computes point estimates and HPD intervals for each factor combination in `object@emmGrid`. While this function may be called independently, it is called automatically by the S3 method `summary.emmGrid` when the object is based on a Bayesian model. (Note: the `level` argument, or its default, is passed as `prob`).

Usage

```
hpd.summary(object, prob, by, type, point.est = median, delta,
  bias.adjust = get_emm_option("back.bias.adj"), sigma, ...)
```

Arguments

`object`	an `emmGrid` object having a non-missing `post.beta` slot
`prob`	numeric probability content for HPD intervals (note: when not specified, the current `level` option is used; see `emm_options`)
`by`	factors to use as by variables
`type`	prediction type as in `summary.emmGrid`
`point.est`	function to use to compute the point estimates from the posterior sample for each grid point
`delta`	Numeric equivalence threshold (on the linear predictor scale regardless of `type`). See the section below on equivalence testing.
`bias.adjust`	Logical value for whether to adjust for bias in back-transforming (`type = "response"`). This requires a value of `sigma` to exist in the object or be specified.
`sigma`	Error SD assumed for bias correction (when `type = "response"`. If not specified, `object@misc$sigma` is used, and a warning if it is not found or invalid. *Note:* `sigma` may be a vector, as long as it conforms to the number of observations in the posterior sample.
`...`	required but not used

Value

an object of class `summary_emm`

Equivalence testing note

If `delta` is positive, two columns labeled `p.equiv` and `odds.eq` are appended to the summary. `p.equiv` is the fraction of posterior estimates having absolute values less than `delta`. The `odds.eq` column is just `p.equiv` converted to an odds ratio; so it is the posterior odds of equivalence.

A high value of `p.equiv` is evidence in favor of equivalence. It can be used to obtain something equivalent (in spirit) to the frequentist Schuirmann (TOST) procedure, whereby we would conclude equivalence at significance level α if the $(1 - 2\alpha)$ confidence interval falls entirely in the interval $[-\delta, \delta]$. Similarly in the Bayesian context, an equally strong argument for equivalence is obtained if `p.equiv` exceeds $1 - 2\alpha$.

A closely related quantity is the ROPE (region of practical equivalence), obtainable via `bayestestR::rope(object, range = c(-delta, delta))`. Its value is approximately `100 * p.equiv / 0.95` if the default `ci = 0.95` is used. See also **bayestestR**'s issue #567.

Finally, a Bayes factor for equivalence is obtainable by dividing `odds.eq` by the prior odds of equivalence, assessed or elicited separately.

See Also

summary.emmGrid

Examples

```
if(require("coda"))
    emm_example("hpd.summary-coda")
    # Use emm_example("hpd.summary-coda", list = TRUE) # to see just the code
```

---

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, showconf = TRUE,
  cov.reduce = make.meanint(1), ...)

make.meanint(delta = 1, npts = 2)

meanint(x)

make.symmint(ctr, delta = 1, npts = 2)

symmint(ctr)
```

Arguments

object	a fitted model, emmGrid, or emm_list. If the latter, its first element is used.
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
showconf	logical value. When we have models with estimability issues (e.g., missing cells), then with showconf = TRUE, we test any remaining effects that are not purely due to contrasts of a single term. If found, they are labeled (confounded). See vignette("xplanations") for more information.
cov.reduce	a function. If object is a fitted model, it is replaced by ref_grid(object, cov.reduce = cov.reduce, ...). For this purpose, the functions meanint and symmint are available for returning an interval around the mean or around zero, respectively. Se the section below on covariates.
...	additional arguments passed to ref_grid and emmeans
delta, ctr	arguments for make.meanint and make.symmint. delta sets the distance each side of the center, so that the width of the interval is 2*delta.
npts	number of points to include in the interval
x	argument for meanint and symmint

Details

In models with only factors, no covariates, 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 they interact with factors, the results depend on how the covariate is handled in constructing the reference grid. See the section on covariates below. The point that one must always remember is that joint_tests always tests contrasts among EMMs, in the context of the reference grid, whereas SAS's 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. There may be an additional row named

(confounded) which accounts for additional degrees of freedom for effects not accounted for in the preceding rows.

The returned object also includes an "est.fcns" attribute, which is a named list containing the linear functions associated with each joint test. Each row of these is standardized to have length 1. No estimable functions are included for confounded effects.

make.meanint returns the function function(x) mean(x) + delta * c(-1, 1), and make.symmint(ctr, delta) returns the function function(x) ctr + delta * c(-1, 1) (which does not depend on x). The cases with delta = 1, meanint = make.meanint(1) and symmint(ctr) = make.symmint(ctr, 1) are retained for back-compatibility reasons. These functions are available primarily for use with cov.reduce.

Dealing with covariates

A covariate (or any other predictor) must have *more than one value in the reference grid* in order to test its effect and be included in the results. Therefore, when object is a model, we default to cov.reduce = meanint which sets each covariate at a symmetric interval about its mean. But when object is an existing reference grid, it often has only one value for covariates, in which case they are excluded from the joint tests.

While having two points is sufficient when the covariate term has a linear trend, you need more than two when some kind of curved trend (polynomial, spline, etc.) is present – else joint_tests() will not show enough degrees of freedom for terms involving the covariate. You may specify these points manually using at, or by including an npts argument in cov.reduce, via make.meanint or make.symmint(). With some kinds of curved trends, the joint tests of covariate terms may become somewhat meaningless.

Covariates present further complications in that their values in the reference grid can affect the joint tests of *other* effects. When covariates are centered around their means (the default), then the tests we obtain can be described as joint tests of covariate-adjusted means; and that is our intended use here. However, some software such as **SAS** and car::Anova adopt the convention of centering covariates around zero; and for that purpose, one can use cov.reduce = symmint(0) when calling with a model object (or in constructing a reference grid). However, adjusted means with covariates set at or around zero do not make much sense in the context of interpreting estimated marginal means, unless the covariate means really are zero.

See the examples below with the toy dataset.

Note

joint_tests is flaky with models having nested fixed effects. In some cases, terms that could be relevant are not identified, or confounded with unidentifiable terms.

See Also

test

Examples

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

(jt <- joint_tests(pigs.lm))            ## will be same as type III ANOVA
```

```
### Estimable functions associated with "percent"
attr(jt, "est.fcns") $ "percent"

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

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

### Comparisons with type III tests in SAS
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)

# -- SAS output we'd get with toy.fac --
## 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
#
# -- SAS output we'd get with toy.cov --
## 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

joint_tests(toy.fac)
joint_tests(toy.cov)   # female is regarded as a 2-level factor by default

## Treat 'female' as a numeric covariate (via cov.keep = 0)
## ... then tests depend on where we center things

# Center around the mean
joint_tests(toy.cov, cov.keep = 0, cov.reduce = make.meanint(delta = 1))
# Center around zero (like SAS's results for toy.cov)
joint_tests(toy.cov, cov.keep = 0, cov.reduce = make.symmint(ctr = 0, delta = 1))
# Center around 0.5 (like SAS's results for toy.fac)
joint_tests(toy.cov, cov.keep = 0, cov.reduce = range)

### Example with empty cells and confounded effects
low3 <- unlist(attr(ubds, "cells")[1:3])
ubds.lm <- lm(y ~ A*B*C, data = ubds, subset = -low3)

# Show overall joint tests by C:
ref_grid(ubds.lm, by = "C") |> contrast("consec") |> test(joint = TRUE)

# Break each of the above into smaller components:
joint_tests(ubds.lm, by = "C")
```

`lsmeans` *Wrappers for alternative naming of EMMs*

Description

These are wrappers for `emmeans` and related functions to provide backward compatibility, or for users who may prefer to use other terminology than "estimated marginal means" – namely "least-squares means". These functions also provide the functionality formerly provided by the **lsmeans** package, which is now just a front-end for **emmeans**.

Usage

```
lsmeans(...)

lstrends(...)

lsmip(...)

lsm(...)

lsmobj(...)

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* in its name, the same function named em*xxxx* is called. Any estimator names or list items beginning with "em" are replaced with "ls" 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 **emmeans** options system.

See Also

`emmeans`, `emtrends`, `emmip`, `emm`, `emmobj`, `emm_options`, `get_emm_option`

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",
  "atanh", "bcnPower", "scale"), alpha = 1, beta = 0, param, y, inner, ...)

inverse(y)
```

Arguments

`type`	The name of a standard transformation supported by `stat::make.link`, or of a special transformation described under Details.
`alpha, beta`	Numeric parameters needed for special transformations.
`param`	If non-missing, this specifies either `alpha` or `c(alpha, beta)` (provided for backward compatibility). Also, for the same reason, if `alpha` is of length more than 1, it is taken as `param`.
`y`	A numeric response variable used (*and required*) with `type = "scale"`, where `scale(y)` determines `alpha` and `beta`.
`inner`	another transformation. See the section on compound transformations
`...`	Additional arguments passed to other functions/methods

Value

A `list` having at least the same elements as those returned by `make.link`. The `linkfun` component is the transformation itself. Each of the functions is associated with an environment where any parameter values are defined.

`inverse` returns the reciprocal of its argument. It allows the `"inverse"` link to be auto-detected as a response transformation.

Details

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.

The primary purpose of `make.tran` is to support transformations that require additional parameters, specified as `alpha` and `beta`; these are the onse shown in the argument-matching list. However, standard transformations supported by `stats::make.link` are also supported. In the following discussion of ones requiring parameters, we use α and β to denote `alpha` and `beta`, and y to denote the response variable. The `type` argument specifies the following transformations:

`"genlog"` Generalized logarithmic transformation: $\log_\beta(y+\alpha)$, where $y > -\alpha$. When $\beta = 0$ (the default), we use $\log_e(y+\alpha)$

`"power"` Power transformation: $(y-\beta)^\alpha$, where $y > \beta$. When $\alpha = 0$, $\log(y-\beta)$ is used instead.

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

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

`"asin.sqrt"` Arcsin-square-root transformation: $\sin^{-1}(y/\alpha)^{1/2}$. Typically, `alpha` will be either 1 (default) or 100.

`"atanh"` Arctanh transformation: $\tanh^{-1}(y/\alpha)$. Typically, `alpha` will be either 1 (default) or 100.

`"bcnPower"` Box-Cox with negatives allowed, as described for the `bcnPower` function in the **car** package. It is defined as the Box-Cox transformation $(z^\alpha - 1)/\alpha$ of the variable $z = y + (y^2 + \beta^2)^{1/2}$. Note that this requires both parameters and that `beta > 0`.

`"scale"` This one is a little different than the others, in that `alpha` and `beta` are ignored; instead, they are determined by calling `scale(y, ...)`. The user should give as `y` the response variable in the model to be fitted to its scaled version.

Note that with the `"power"`, `"boxcox"`, or `"sympower"` transformations, the argument `beta` specifies a location shift. In the `"genpower"` transformation, `beta` specifies the base of the logarithm – however, quirkily, the default of `beta = 0` is taken to be the natural logarithm. For example, `make.tran(0.5, 10)` sets up the $\log_{10}(y + \frac{1}{2})$ transformation. In the `"bcnPower"` transformation, `beta` must be specified as a positive value.

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 $2\sqrt{\mu}$.

Cases where `make.tran` may not be needed

For standard transformations with no parameters, we usually don't need to use `make.tran`; just the name of the transformation is all that is needed. 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`, `log1p`, `sqrt`, `logit`, `probit`, `cauchit`, `cloglog`; as well as (for a response variable `y`) `asin(sqrt(y))`, `asinh(sqrt(y))`, `atanh(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 transformations may be specified as character strings and are auto-detected: `"identity"`, `"1/mu^2"`, `"inverse"`, `"reciprocal"`, `"log10"`, `"log2"`, `"asin.sqrt"`, `"asinh.sqrt"`, and `"atanh"`.

Compound transformations

A transformation that is a function of another function can be created by specifying `inner` for the other function. For example, the transformation $1/\sqrt{y}$ can be created either by `make.tran("inverse", inner = "sqrt")` or by `make.tran("power",-0.5)`. In principle, transformations can be compounded to any depth. Also, if `type` is `"scale"`, `y` is replaced by `inner$linkfun(y)`, because that will be the variable that is scaled.

Note

The `genlog` transformation is technically unneeded, because a response transformation of the form `log(y + c)` is now auto-detected by `ref_grid`.

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")

### Using a scaled response...
# Case where it is auto-detected:
mod <- lm(scale(yield[, 1]) ~ Variety, data = MOats)
emmeans(mod, "Variety", type = "response")

# Case where scaling is not auto-detected -- and what to do about it:
copt <- options(contrasts = c("contr.sum", "contr.poly"))
mod.aov <- aov(scale(yield[, 1]) ~ Variety + Error(Block), data = MOats)
emm.aov <- suppressWarnings(emmeans(mod.aov, "Variety", type = "response"))

# Scaling was not retrieved, but we can do:
emm.aov <- update(emm.aov, tran = make.tran("scale", y = MOats$yield[, 1]))
emmeans(emm.aov, "Variety", type = "response")

### Compound transformations
# The following amount to the same thing:
t1 <- make.tran("inverse", inner = "sqrt")
t2 <- make.tran("power", -0.5)

options(copt)
```

```
## Not run:
### An existing model 'mod' was fitted with a y^(2/3) transformation...
  ptran = make.tran("power", 2/3)
  emmeans(mod, "treatment", tran = ptran)

## End(Not run)

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

`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")`.

`mvcontrast` *Multivariate contrasts*

Description

This function displays tests of multivariate comparisons or contrasts. The contrasts are constructed at each level of the variable in `mult.name`, and then we do a multivariate test that the vector of estimates is equal to `null` (zero by default). The F statistic and degrees of freedom are determined via the Hotelling distribution. that is, if there are m error degrees of freedom and multivariate dimensionality d, then the resulting F statistic has degrees of freedom $(d, m - d + 1)$ as shown in Hotelling (1931).

Usage

```
mvcontrast(object, method = "eff", mult.name = object@roles$multresp,
  null = 0, by = object@misc$by.vars, adjust = c("sidak",
  p.adjust.methods), show.ests = FALSE, ...)
```

Arguments

`object`	An object of class `emmGrid`
`method`	A contrast method, per `contrast.emmGrid`
`mult.name`	Character vector of names of the factors whose levels define the multivariate means to contrast. If the model itself has a multivariate response, that is what is used. Otherwise, `mult.name` *must* be specified.
`null`	Scalar or conformable vector of null-hypothesis values to test against
`by`	Any by variable(s). These should not include the primary variables to be contrasted. For convenience, the by variable is nulled-out if it would result in no primary factors being contrasted.
`adjust`	Character value of a multiplicity adjustment method (`"none"` for no adjustment). The available adjustment methods are more limited that in `contrast`, and any default adjustment returned via `method` is ignored.
`show.ests`	Logical flag determining whether the multivariate means are displayed
`...`	Additional arguments passed to `contrast`

Value

An object of class `summary_emm` containing the multivariate test results; or a list of the estimates and the tests if `show.ests` is `TRUE`. The test results include the Hotelling T^2 statistic, F ratios, degrees of freedom, and P values.

Note

If some interactions among the primary and `mult.name` factors are absent, the covariance of the multivariate means is singular; this situation is accommodated, but the result has reduced degrees of freedom and a message is displayed. If there are other abnormal conditions such as non-estimable results, estimates are shown as `NA`.

While designed primarily for testing contrasts, multivariate tests of the mean vector itself can be implemented via `method = "identity"`) (see the examples).

References

Hotelling, Harold (1931) "The generalization of Student's ratio", *Annals of Mathematical Statistics* 2(3), 360–378. doi:10.1214/aoms/1177732979

Examples

```
MOats.lm <- lm(yield ~ Variety + Block, data = MOats)
MOats.emm <- emmeans(MOats.lm, ~ Variety | rep.meas)
mvcontrast(MOats.emm, "consec", show.ests = TRUE)  # mult.name defaults to rep.meas

# Test each mean against a specified null vector
mvcontrast(MOats.emm, "identity", name = "Variety",
           null = c(80, 100, 120, 140), adjust = "none")
# (Note 'name' is passed to contrast() and overrides default name "contrast")

# 'mult.name' need not refer to a multivariate response
mvcontrast(MOats.emm, "trt.vs.ctrl1", mult.name = "Variety")
```

---

mvregrid *Multivariate regridding*

Description

This function is similar to `regrid` except it performs a multivariate transformation. This is useful, for instance, in multivariate models that have a compositional response.

Usage

```
mvregrid(object, newname = "component", newlevs = seq_len(ncol(newy)),
  mult.name = names(levels)[length(levels)], fcn = paste0(tran, "Inv"),
  ...)
```

Arguments

object	An emmGrid object
newname	The name to give to the newly created multivariate factor
newlevs	Character levels of the newly created factor (must conform to the number of columns created by fcn)
mult.name	The name of the multivariate factor to be transformed. By default, we use the last factor
fcn	The multivariate function to apply. If character, we look for it in the namespace of the **compositions** package.
...	Additional arguments passed to fcn

Details

If a multivariate response transformation was used in fitting the model, its name is auto-detected, and in that case we need not specify fcn as long as its inverse can be found in the namespace of the **compositions** package. (That package need not be installed unless fcn is a character value.) For some such models, auto-detection process throws a warning message, especially if cbind is also present in the model formula.

Currently, no bias-adjustment option is available.

Value

A new emmGrid object with the newly created factor as its last factor

Examples

```
if(requireNamespace("compositions"))
    emm_example("mvregrid")
    # Use emm_example("mvregrid", list = TRUE) # to see just the code
```

---

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
pairs(nutr.emm, by = "group")
pairs(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

This is (or once was) available as a SAS sample dataset.

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(inverse(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 CIs, and optionally to display "comparison arrows" for displaying pairwise comparisons.

Usage

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

## S3 method for class 'summary_emm'
plot(x, y, horizontal = TRUE, CIs = TRUE, xlab, ylab,
  layout, scale = NULL, colors = c("black", "blue", "blue", "red"),
  intervals, plotit = TRUE, ...)
```

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`. In addition, the user may specify `type = "scale"`, in which case a transformed scale (e.g., a log scale) is displayed based on the transformation or link function used. Additional customization of this scale is available through including arguments to `ggplot2::scale_x_continuous` in `...`.
`CIs`	Logical value. If `TRUE`, confidence intervals are plotted for each estimate.
`PIs`	Logical value. If `TRUE`, prediction intervals are plotted for each estimate. If `object` is a Bayesian model, this requires the `...` arguments to include `frequentist = TRUE` and `sigma` = (some value). Note that the `PIs` option is *not* available with `summary_emm` objects – only for `emmGrid` objects. Also, prediction intervals are not available with `engine = "lattice"`.
`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.) Note that comparison arrows are not available with 'summary_emm' objects.

colors	Character vector of color names to use for estimates, CIs, PIs, and comparison arrows, respectively. CIs and PIs are rendered with some transparency, and colors are recycled if the length is less than four; so all plot elements are visible even if a single color is specified.
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*.
intervals	If specified, it is used to set CIs. This is the previous argument name for CIs and is provided for backward compatibility.
...	Additional arguments passed to update.emmGrid, summary.emmGrid, predict.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 when engine == "lattice".
scale	Object of class trans (in the **scales** package) to specify a nonlinear scale. This is used in lieu of type = "scale" when plotting a summary_emm object created with type = "response". This is ignored with other types of summaries.
plotit	Logical value. If TRUE, a graphical object is returned; if FALSE, a data.frame is returned containing all the values used to construct the plot.

Value

If plotit = TRUE, a graphical object is returned.

If plotit = FALSE, a data.frame with the table of EMMs that would be plotted. In the latter case, the estimate being plotted is named the.emmean, and any factors involved have the same names as in the object. Confidence limits are named lower.CL and upper.CL, prediction limits are named lpl and upl, and comparison-arrow limits are named lcmpl and ucmpl. There is also a variable named pri.fac which contains the factor combinations that are *not* among the by variables.

Details

If any by variables are in force, the plot is divided into separate panels. 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. See the vignette("xplanations", "emmeans") for details on how these are derived.

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).

Note

In order to play nice with the plotting functions, any variable names that are not syntactically correct (e.g., contain spaces) are altered using `make.names`.

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, colors = "darkgreen")

### Using a transformed scale
pigs.lm <- lm(log(conc + 2) ~ source * factor(percent), data = pigs)
pigs.emm <- emmeans(pigs.lm, ~ percent | source)
plot(pigs.emm, type = "scale", breaks = seq(20, 100, by = 10))

# Based on a summary.
# To get a transformed axis, must specify 'scale'; but it does not necessarily
# have to be the same as the actual response transformation
pigs.ci <- confint(pigs.emm, type = "response")
plot(pigs.ci, scale = scales::log10_trans())
```

---

pwpm *Pairwise P-value matrix (plus other statistics)*

Description

This function presents results from `emmeans` and pairwise comparisons thereof in a compact way. It displays a matrix (or matrices) of estimates, pairwise differences, and P values. The user may opt to exclude any of these via arguments `means`, `diffs`, and `pvals`, respectively. To control the direction of the pairwise differences, use `reverse`; and to control what appears in the upper and lower triangle(s), use `flip`. Optional arguments are passed to `contrast.emmGrid` and/or `summary.emmGrid`, making it possible to control what estimates and tests are displayed.

Usage

```
pwpm(emm, by, reverse = FALSE, pvals = TRUE, means = TRUE,
  diffs = TRUE, flip = FALSE, digits, ...)
```

Arguments

`emm`	An `emmGrid` object
`by`	Character vector of variable(s) in the grid to condition on. These will create different matrices, one for each level or level-combination. If missing, `by` is set to `emm@misc$by.vars`. Grid factors not in `by` are the *primary* factors: whose levels or level combinations are compared pairwise.

reverse	Logical value passed to pairs.emmGrid. Thus, FALSE specifies "pairwise" comparisons (earlier vs. later), and TRUE specifies "revpairwise" comparisons (later vs. earlier).
pvals	Logical value. If TRUE, the pairwise differences of the EMMs are included in each matrix according to flip.
means	Logical value. If TRUE, the estimated marginal means (EMMs) from emm are included in the matrix diagonal(s).
diffs	Logical value. If TRUE, the pairwise differences of the EMMs are included in each matrix according to flip.
flip	Logical value that determines where P values and differences are placed. FALSE places the P values in the upper triangle and differences in the lower, and TRUE does just the opposite.
digits	Integer. Number of digits to display. If missing, an optimal number of digits is determined.
...	Additional arguments passed to contrast.emmGrid and summary.emmGrid. You should *not* include method here, because pairwise comparisons are always used.

Value

A matrix or 'list' of matrices, one for each 'by' level.

Note

If emm is the result of a Bayesian analysis, pwpm is based on a frequentist analysis

See Also

A graphical display of essentially the same results is available from pwpp

Examples

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

pwpm(warp.emm)

# use dot options to specify noninferiority tests
pwpm(warp.emm, by = NULL, side = ">", delta = 5, adjust = "none")
```

---

pwpp *Pairwise P-value plot*

Description

Constructs a plot of P values associated with pairwise comparisons of estimated marginal means.

Usage

```
pwpp(emm, method = "pairwise", by, sort = TRUE, values = TRUE,
  rows = ".", xlab, ylab, xsub = "", plim = numeric(0), add.space = 0,
  aes, ...)
```

Arguments

emm	An emmGrid object
method	Character or list. Passed to contrast, and defines the contrasts to be displayed. Any contrast method may be used, provided that each contrast includes one coefficient of 1, one coefficient of -1, and the rest 0. That is, calling contrast(object, method) produces a set of comparisons, each with one estimate minus another estimate.
by	Character vector of variable(s) in the grid to condition on. These will create different panels, one for each level or level-combination. Grid factors not in by are the *primary* factors: whose levels or level combinations are compared pairwise.
sort	Logical value. If TRUE, levels of the factor combinations are ordered by their marginal means. If FALSE, they appear in order based on the existing ordering of the factor levels involved. Note that the levels are ordered the same way in all panels, and in many cases this implies that the means in any particular panel will *not* be ordered even when sort = TRUE.
values	Logical value. If TRUE, the values of the EMMs are included in the plot. When there are several side-by-side panels due to by variable(s), the labels showing values start stealing a lot of space from the plotting area; in those cases, it may be desirable to specify FALSE or use rows so that some panels are vertically stacked.
rows	Character vector of which by variable(s) are used to define rows of the panel layout. Those variables in by not included in rows define columns in the array of panels. A "." indicates that only one row is used, so all panels are stacked side-by-side.
xlab	Character label to use in place of the default for the P-value axis.
ylab	Character label to use in place of the default for the primary-factor axis.
xsub	Character label used as caption at the lower right of the plot.
plim	numeric vector of value(s) between 0 and 1. These are included among the observed p values so that the range of tick marks includes at least the range of plim. Choosing plim = c(0,1) will ensure the widest possible range.
add.space	Numeric value to adjust amount of space used for value labels. Positioning of value labels is tricky, and depends on how many panels and the physical size of the plotting region. This parameter allows the user to adjust the position. Changing it by one unit should shift the position by about one character width (right if positive, left if negative). Note that this interacts with aes$label below.
aes	optional named list of lists. Entries considered are point, segment, and label, and contents are passed to the respective ggplot2::geom_xxx() functions. These affect rendering of points, line segments joining them, and value labels. Defaults are point = list(size = 2), segment = list(), and label = list(size = 2.5).

... Additional arguments passed to `contrast` and `summary.emmGrid`, as well as to `geom_segment` and `geom_label`

Details

Factor levels (or combinations thereof) are plotted on the vertical scale, and P values are plotted on the horizontal scale. Each P value is plotted twice – at vertical positions corresponding to the levels being compared – and connected by a line segment. Thus, it is easy to visualize which P values are small and large, and which levels are compared. In addition, factor levels are color-coded, and the points and half-line segments appear in the color of the other level. The P-value scale is nonlinear, so as to stretch-out smaller P values and compress larger ones. P values smaller than 0.0004 are altered and plotted in a way that makes them more distinguishable from one another.

If `xlab`, `ylab`, and `xsub` are not provided, reasonable labels are created. `xsub` is used to note special features; e.g., equivalence thresholds or one-sided tests.

Note

If `emm` is the result of a Bayesian analysis, the plot is based on summaries with `frequentist = TRUE`.

The **ggplot2** and **scales** packages must be installed in order for `pwpp` to work.

Additional plot aesthetics are available by adding them to the returned object; see the examples

See Also

A numerical display of essentially the same results is available from `pwpm`

Examples

```
pigs.lm <- lm(log(conc) ~ source * factor(percent), data = pigs)
emm = emmeans(pigs.lm, ~ percent | source)
pwpp(emm)
pwpp(emm, method = "trt.vs.ctrl1", type = "response", side = ">")

# custom aesthetics:
my.aes <- list(point = list(shape = "square"),
               segment = list(linetype = "dashed", color = "red"),
               label = list(family = "serif", fontface = "italic"))
my.pal <- c("darkgreen", "blue", "magenta", "orange")
pwpp(emm, aes = my.aes) + ggplot2::scale_color_manual(values = my.pal)
```

---

qdrg *Quick and dirty reference grid*

Description

This function may make it possible to compute a reference grid for a model object that is otherwise not supported.

Usage

```
qdrg(formula, data, coef, vcov, df, mcmc, object, subset, weights, contrasts,
  link, qr, ordinal, ...)
```

Arguments

formula	Formula for the fixed effects
data	Dataset containing the variables in the model
coef	Fixed-effect regression coefficients (must conform to formula)
vcov	Variance-covariance matrix of the fixed effects
df	Error degrees of freedom
mcmc	Posterior sample of fixed-effect coefficients
object	Optional model object. *This rarely works!*; but if provided, we try to set other arguments based on an expectation that 'object' has a similar structure to 'lm' objects. See Details.
subset	Subset of data used in fitting the model
weights	Weights used in fitting the model
contrasts	List of contrasts specified in fitting the model
link	Link function (character or list) used, if a generalized linear model. (Note: response transformations are auto-detected from formula)
qr	QR decomposition of the model matrix; used only if there are NAs in coef.
ordinal	list with elements dim and mode. ordinal$dim (integer) is the number of levels in an ordinal response. If ordinal is provided, the intercept terms are modified appropriate to predicting an ordinal response, as described in vignette("models"), Section O, using ordinal$mode as the mode argument (if not provided, "latent" is assumed). (All modes are supported except 'scale') For this to work, we expect the first ordinal$dim - 1 elements of coef to be the estimated threshold parameters, followed by the coefficients for the linear predictor.
...	Optional arguments passed to ref_grid

Details

Usually, you need to provide either object; or formula, coef, vcov, data, and perhaps other parameters. It is usually fairly straightforward to figure out how to get these from the model object; see the documentation for the model class that was fitted. Sometimes one or more of these quantities contains extra parameters, and if so, you may need to subset them to make everything conformable. For a given formula and data, you can find out what is needed via colnames(model.matrix(formula, data)). (However, for an ordinal model, we expect the first ordinal.dim - 1 coefficients to replace (Intercept). And for a multivariate model, we expect coef to be a matrix with these row names, and vcov to have as many rows and columns as the total number of elements of coef.)

If your model object follows fairly closely the conventions of an lm or glmobject, you may be able to get by providing the model as object, and perhaps some other parameters to override the defaults. When object is specified, it is used as detailed below to try to obtain the other arguments. The user should ensure that the defaults shown below do indeed work. The default values for the arguments are as follows:

- formula: formula(object)
- data: recover_data.lm(object) is tried, and if an error is thrown, we also check object$data.
- coef: coef(object)
- vcov: vcov(object)
- df: Set to Inf if not available in df.residual(object)
- mcmc: object$sample
- subset: NULL (so that all observations in data are used)
- contrasts: object$contrasts

The functions qdrg and emmobj are close cousins, in that they both produce emmGrid objects. When starting with summary statistics for an existing grid, emmobj is more useful, while qdrg is more useful when starting from a fitted model.

Value

An emmGrid object constructed from the arguments

Rank deficiencies

Different model-fitting packages take different approaches when the model matrix is singular, but qdrg tries to reconcile them by comparing the linear functions created by formula to coefs and vcov. We may then use the **estimability** package to determine what quantities are estimable. For reconciling to work properly, coef should be named and vcov should have dimnames. To disable this name-matching action, remove the names from coef, e.g., by calling unname(). No reconciliation is attempted in multivariate-response cases. For more details on estimability, see the documentation in the **estimability** package.

Note

For backwards compatibility, an argument ordinal.dim is invisibly supported as part of ..., and if present, sets ordinal = list(dim = ordinal.dim, mode = "latent")

See Also

emmobj for an alternative way to construct an emmGrid.

Examples

```
# In these examples, use emm_example(..., list = TRUE) # to see just the code

if (require(biglm, quietly = TRUE))
    emm_example("qdrg-biglm")

if(require(coda, quietly = TRUE) && require(lme4, quietly = TRUE))
    emm_example("qdrg-coda")

if(require(ordinal, quietly = TRUE))
    emm_example("qdrg-ordinal")
```

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, ...]

## S3 method for class 'emmGrid'
head(x, n = 6, ...)

## S3 method for class 'emmGrid'
tail(x, n = 6, ...)

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

## S3 method for class 'emm_list'
rbind(..., which, adjust = "bonferroni")

## S3 method for class 'summary_emm'
rbind(..., which)

force_regular(object)
```

Arguments

...	In rbind, object(s) of class emmGrid or summary_emm. In others, additional arguments passed to other methods
deparse.level	(required but not used)
adjust	Character value passed to update.emmGrid
e1, e2, x, object	Objects of class emmGrid
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
n	integer number of entries to include (or exclude if negative)
subset	logical expression indicating which rows of the grid to keep
which	Integer vector of subset of elements to use; if missing, we use all elements.

Value

A revised object of class emmGrid

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

The rbind method for emm_list objects simply combines the emmGrid objects comprising the first element of .... Note that the returned object is not yet summarized, so any adjust parameters apply to the combined emmGrid.

The rbind method for summary_emm objects (or a list thereof) returns a single summary_emm object. This combined object *preserves* any adjusted P values or confidence limits in the original summaries, since those quantities have already been computed.

force_regular adds extra (invisible) rows to an emmGrid object to make it a regular grid (all combinations of factors). This regular structure is needed by emmeans. An object can become irregular by, for example, subsetting rows, or by obtaining contrasts of a nested structure.

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.

These functions generally reset by.vars to NULL; so if you want to keep any "by" variables, you should follow-up with update.emmGrid.

Examples

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

# 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 above except for adjustment

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

# After-the-fact 'at' specification
subset(warp.rg, wool == "A")  ## or warp.rg |> subset(wool == "A")


### Working with 'emm_list' objects
```

```
mod <- lm(conc ~ source + factor(percent), data = pigs)
all <- emmeans(mod, list(src = pairwise ~ source, pct = consec ~ percent))
rbind(all, which = c(2, 4), adjust = "mvt")

### Irregular object
tmp <- warp.rg[-1]
## emmeans(tmp, "tension")   # will fail because tmp is irregular
emmeans(force_regular(tmp), "tension")   # will show some results
```

`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,
  cov.keep = get_emm_option("cov.keep"), mult.names, mult.levs,
  options = get_emm_option("ref_grid"), data, df, type, regrid, nesting,
  offset, sigma, counterfactuals, nuisance = character(0), non.nuisance,
  wt.nuis = "equal", rg.limit = get_emm_option("rg.limit"), ...)
```

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 `cov.keep` or `at` is reduced according to these specifications. See the section below on "Using `cov.reduce` and `cov.keep`".
`cov.keep`	Character vector: names of covariates that are *not* to be reduced; these are treated as factors and used in weighting calculations. `cov.keep` may also include integer value(s), and if so, the maximum of these is used to set a threshold such that any covariate having no more than that many unique values is automatically included in `cov.keep`.
`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. See the note with `recover_data` for an important precaution.

`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.

`regrid` Character, logical, or list. If non-missing, the reference grid is reconstructed via `regrid` with the argument `transform = regrid`. See the section below on prediction types and transformations. *Note:* This argument was named `transform` in version 1.7.2 and earlier. For compatibility with old code, `transform` is still accepted if found among `...`, as long as it doesn't match `tran`.

`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 the section below on Recovering or Overriding Model Information.

`offset` Numeric scalar value (if a vector, only the first element is used). This may be used to add an offset, or override offsets based on the model. A common usage would be to specify `offset = 0` for a Poisson regression model, so that predictions from the reference grid become rates relative to the offset that had been specified in the model.

`sigma` Numeric value to use for subsequent predictions or back-transformation bias adjustments. If not specified, we use `sigma(object)`, if available, and `NULL` otherwise. Note: This applies only when the family is `"gaussian"`; for other families, `sigma` is set to `NA` and cannot be overridden.

`counterfactuals`

`counterfactuals` specifies character names of counterfactual factors. If this is non-missing, a reference grid is created consisting of combinations of counterfactual levels and the actual levels of those same factors. This grid is always converted to the response transformation scale and averaged over the actual factor levels. See the section below on counterfactuals.

`nuisance`, `non.nuisance`, `wt.nuis`

If `nuisance` is a vector of predictor names, those predictors are omitted from the reference grid. Instead, the result will be as if we had averaged over the levels of those factors, with either equal or proportional weights as specified in `wt.nuis` (see the `weights` argument in `emmeans`). The factors in `nuisance` must not interact with other factors, not even other nuisance factors. Specifying nuisance factors can save considerable storage and computation time, and help avoid exceeding the maximum reference-grid size (`get_emm_option("rg.limit")`). (*Note:* For certain models where the `emm_basis` method returns a re-gridded parameterization, nuisance factors cannot be used, and an error is thrown.)

`rg.limit` Integer limit on the number of reference-grid rows to allow (checked before any multivariate responses are included).

`...` Optional arguments passed to `summary.emmGrid`, `emm_basis`, and `recover_data`, such as `params`, `vcov.` (see **Covariance matrix** below), or options such as `mode` for specific model types (see vignette("models", "emmeans")).

Details

To users, the ref_grid function itself is important because most of its arguments are in effect arguments of emmeans and related functions, in that those functions pass their ... arguments to ref_grid.

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.

By "independent variables," we mean (in most cases) the results of all.vars() applied to the fixed-effects part of the right-hand side of the model formula. Any random effects are excluded. Thus, if the model formula in an lme4::lmer call is yield ~ fert + seed*density + log(rain) + (1|block/plot), the independent variables are fert, seed, density, and rain (not log(rain)). In multivariate models, the dimension of the multivariate response is also considered an independent variable.

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 and cov.keep

The cov.keep argument was not available in **emmeans** versions 1.4.1 and earlier. Any covariates named in this list are treated as if they are factors: all the unique levels are kept in the reference grid. The user may also specify an integer value, in which case any covariate having no more than that number of unique values is implicitly included in cov.keep. The default for cov.keep is set and retrieved via the emm_options framework, and the system default is "2", meaning that covariates having only two unique values are automatically treated as two-level factors. See also the Note below on backward compatibility.

There is a subtle distinction between including a covariate in cov.keep and specifying its values manually in at: Covariates included in cov.keep are treated as factors for purposes of weighting, while specifying levels in at will not include the covariate in weighting. See the mtcars.lm example below for an illustration.

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 including all covariates in cov.keep. Use of 'cov.reduce = FALSE' is inadvisable because it can result in a huge reference grid; it is far better to use cov.keep.

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.

To allow for situations where a simple `lm()` call as described above won't be adequate, a formula of the form `ext ~ fcnname` is also supported, where the left-hand side may be `ext`, `extern`, or `external` (and must *not* be a predictor name) and the right-hand side is the name of an existing function. The function is called with one argument, a data frame with columns for each variable in the reference grid. The function is expected to use that frame as new data to be used to obtain predictions for one or more models; and it should return a named list or data frame with replacement values for one or more of the covariates.

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` of `cov.keep` specification for a covariate also named in `at` is ignored.

Interdependent covariates

Care must be taken when covariate values depend on one another. For example, when a polynomial model was fitted using predictors `x`, `x2` (equal to `x^2`), and `x3` (equal to `x^3`), the reference grid will by default set `x2` and `x3` to their means, which is inconsistent. The user should instead use the `at` argument to set these to the square and cube of `mean(x)`. Better yet, fit the model using a formula involving `poly(x, 3)` or `I(x^2)` and `I(x^3)`; then there is only `x` appearing as a covariate; it will be set to its mean, and the model matrix will have the correct corresponding quadratic and cubic terms.

Matrix covariates

Support for covariates that appear in the dataset as matrices is very limited. If the matrix has but one column, it is treated like an ordinary covariate. Otherwise, with more than one column, each column is reduced to a single reference value – the result of applying `cov.reduce` to each column (averaged together if that produces more than one value); you may not specify values in `at`; and they are not treated as variables in the reference grid, except for purposes of obtaining predictions.

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 arguments `(object, ...)`. Be careful with possible unintended conflicts with arguments in

`...`; for example, `sandwich::vcovHAC()` has optional arguments `adjust` and `weights` that may be intended for `emmeans()` but will also be passed to `vcov.()`.

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, a named `list`, or `NULL`. 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)"` (the parentheses are optional). 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*country)"`, and `nesting = c("state %in% country", "city %in% state*country")`.

Predictors with subscripts and data-set references

When the fitted model contains subscripts or explicit references to data sets, the reference grid may optionally be post-processed to simplify the variable names, depending on the `simplify.names` option (see `emm_options`), which by default is `TRUE`. For example, if the model formula is `data1$resp ~ data1$trt + data2[[3]] + data2[["cov"]]`, the simplified predictor names (for use, e.g., in the `specs` for `emmeans`) will be `trt`, `data2[[3]]`, and `cov`. Numerical subscripts are not simplified; nor are variables having simplified names that coincide, such as if `data2$trt` were also in the model.

Please note that this simplification is performed *after* the reference grid is constructed. Thus, non-simplified names must be used in the `at` argument (e.g., `at = list(`data2["cov"]` = 2:4)`.

If you don't want names simplified, use `emm_options(simplify.names = FALSE)`.

Prediction types and transformations

Transformations can exist because of a link function in a generalized linear model, or as a response transformation, or even both. In many cases, they are auto-detected, for example a model formula of the form `sqrt(y) ~ ...`. Even transformations containing multiplicative or additive constants, such as `2*sqrt(y + pi) ~ ...`, are auto-detected. A response transformation of `y + 1 ~ ...` is *not* auto-detected, but `I(y + 1) ~ ...` is interpreted as `identity(y + 1) ~ ...`. A warning is issued if it gets too complicated. Complex transformations like the Box-Cox transformation are not auto-detected; but see the help page for `make.tran` for information on some advanced methods.

There is a subtle difference between specifying '`type = "response"`' and '`regrid = "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 '`regrid = "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 '`regrid = "log", type = "response"`'. See also the help page for `regrid`.

Order-of-processing issues: The `regrid` argument, if present, is acted on immediately after the reference grid is constructed, while some of the `...` arguments may be used to update the object at the very end. Thus, code like `ref_grid(mod, tran = "sqrt", regrid = "response")` will not

work correctly if the intention was to specify the response transformation, because the re-grid is done *before* it processes `tran = "sqrt"`. To get the intended result, do `regrid(ref_grid(mod, tran = "sqrt"), transform = "response")`.

Counterfactuals

If `counterfactuals` is specified, the rows of the entire dataset become part of the reference grid, and the other reference levels are confined to those named in `counterfactuals`. In this type of analysis (called G-computation), we substitute (or impute) each combination of counterfactual levels into the entire dataset. Thus, predictions from this grid are those of each observation under each of the counterfactual levels. For this to make sense, we require an assumption of exchangeability of these levels.

This grid is always converted to the response scale, as G-computation on the linear-predictor scale produces the same results as ordinary weighted EMMs. If we have counterfactual factors `A, B`, the reference grid also includes factors `actual_A, actual_B` which are used to track which observations originally had the `A, B` levels before they were changed by the counterfactuals code. We average the response-scale predictions for each combination of actual levels and imputed levels (and multivariate levels, if any). See additional discussion of how `emmeans` handles counterfactuals under that documentation.

Currently, counterfactuals are not supported when the reference grid requires post-processing (e.g., ordinal models with `mode = "prob"`). Cases where we have nested factor levels can be complicated if mixed-in with counterfactuals, and we make no guarantees. Note that past implementations included arguments `wt.counter` and `avg.counter`, which are now deprecated and are just ignored if specified.

Optional side effect

If the `save.ref_grid` option is set to `TRUE` (see `emm_options`), 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 disabled by default, but may be enabled via '`emm_options(save.ref_grid = TRUE)`'.

Note

The system default for `cov.keep` causes models containing indicator variables to be handled differently than in **emmeans** version 1.4.1 or earlier. To replicate older analyses, change the default via '`emm_options(cov.keep = character(0))`'.

Some earlier versions of **emmeans** offer a `covnest` argument. This is now obsolete; if `covnest` is specified, it is harmlessly ignored. Cases where it was needed are now handled appropriately via the code associated with `cov.keep`.

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")`. See `update.emmGrid` for details of arguments that can be in `options` (or in `...`).

Examples

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

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)

### Model with indicator variables as predictors:
mtcars.lm <- lm(mpg ~ disp + wt + vs * am, data = mtcars)
(rg.default <- ref_grid(mtcars.lm))
(rg.nokeep <- ref_grid(mtcars.lm, cov.keep = character(0)))
(rg.at <- ref_grid(mtcars.lm, at = list(vs = 0:1, am = 0:1)))

# Two of these have the same grid but different weights:
rg.default@grid
rg.at@grid

### Using cov.reduce formulas...
# Above suggests we can vary disp indep. of other factors - unrealistic
rg.alt <- ref_grid(mtcars.lm, at = list(wt = c(2.5, 3, 3.5)),
    cov.reduce = disp ~ vs * wt)
rg.alt@grid

# Alternative to above where we model sqrt(disp)
disp.mod <- lm(sqrt(disp) ~ vs * wt, data = mtcars)
disp.fun <- function(dat)
    list(disp = predict(disp.mod, newdata = dat)^2)
rg.alt2 <- ref_grid(mtcars.lm, at = list(wt = c(2.5, 3, 3.5)),
    cov.reduce = external ~ disp.fun)
rg.alt2@grid


# 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]))

# Comparing estimates with and without counterfactuals
neuralgia.glm <- glm(Pain ~ Treatment + Sex + Age + Duration,
                     family = binomial(), data = neuralgia)
emmeans(neuralgia.glm, "Treatment", type = "response")
```

```
emmeans(neuralgia.glm, "Treatment", counterfactuals = "Treatment")


# Using 'params'
require("splines")
my.knots = c(2.5, 3, 3.5)
mod = lm(Sepal.Length ~ Species * ns(Sepal.Width, knots = my.knots), data = iris)
## my.knots is not a predictor, so need to name it in 'params'
ref_grid(mod, params = "my.knots")
```

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

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", "none", "pass",
  links), inv.link.lbl = "response", predict.type,
  bias.adjust = get_emm_option("back.bias.adj"), sigma, N.sim,
  sim = mvtnorm::rmvnorm, ...)
```

Arguments

`object`	An object of class `emmGrid`
`transform`	Character, list, or logical value. If `"response"`, `"mu"`, or `TRUE`, 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 `"none"` or `FALSE`, `object` is re-gridded so that its `bhat` slot contains `predict(object)` and its `linfct` slot is the identity. Any internal transformation information is preserved. If `transform = "pass"`, the object is not re-gridded in any way (this may be useful in conjunction with `N.sim`). If `transform` is a character value in `links` (which is the set of valid arguments for the `make.link` function, excepting `"identity"`), or if `transform` is a list of the same form as returned by `make.links` or `make.tran`, the results are formulated as if the response had been transformed with that link function.
`inv.link.lbl`	Character value. This applies only when `transform` is in `links`, 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 to use by default by summary.emmGrid (see update.emmGrid). This may be useful if, for example, when one specifies transform = "log" but desires summaries to be produced by default on the response scale.
bias.adjust	Logical value for whether to adjust for bias in back-transforming (transform = "response"). This requires a valid value of sigma to exist in the object or be specified.
sigma	Error SD assumed for bias correction (when transform = "response" and a transformation is in effect). If not specified, object@misc$sigma is used, and a warning is issued if it is not found.
N.sim	Integer value. If specified and object is based on a frequentist model (i.e., does not have a posterior sample), then a fake posterior sample is generated using the function sim.
sim	A function of three arguments (no names are assumed). If N.sim is supplied with a frequentist model, this function is called with respective arguments N.sim, object@bhat, and object@V. The default is the multivariate normal distribution.
...	Ignored.

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.

This function may also be used to simulate a sample of regression coefficients for a frequentist model for subsequent use as though it were a Bayesian model. To do so, specify a value for N.sim and a sample is simulated using the function sim. The grid may be further processed in accordance with the other arguments; or if transform = "pass", it is simply returned with the only change being the addition of the simulated sample.

Value

An emmGrid object with the requested changes

Degrees of freedom

In cases where the degrees of freedom depended on the linear function being estimated (e.g., Satterthwaite method), 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. An annotation is displayed below any subsequent summary results stating that the degrees-of-freedom method is inherited from the previous method at the time of re-gridding.

Note

Another way to use `regrid` is to supply a `regrid` argument to `ref_grid` (either directly of indirectly via `emmeans`), in which case its value is passed to `regrid` as `transform`. 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)
rg <- ref_grid(pigs.lm)

# This will yield EMMs as GEOMETRIC means of concentrations:
(emm1 <- emmeans(rg, "source", type = "response"))
pairs(emm1) ## We obtain RATIOS

# This will yield EMMs as ARITHMETIC means of concentrations:
(emm2 <- emmeans(regrid(rg, transform = "response"), "source"))
pairs(emm2)  ## We obtain DIFFERENCES
# Same result, useful if we hadn't already created 'rg'
# emm2 <- emmeans(pigs.lm, "source", regrid = "response")

# Simulate a sample of regression coefficients
set.seed(2.71828)
rgb <- regrid(rg, N.sim = 200, transform = "pass")
emmeans(rgb, "source", type = "response")  ## similar to emm1
```

`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, ..., export = FALSE)

## S3 method for class 'emmGrid'
vcov(object, ..., sep = get_emm_option("sep"))

linfct(object, ...)

## Default S3 method:
linfct(object, ...)
```

Arguments

object	An emmGrid object
...	(required but not used)
x	An emmGrid object
export	Logical value. If FALSE, the object is printed. If TRUE, a list is invisibly returned, which contains character elements named summary and annotations that may be saved or displayed as the user sees fit. summary is a character matrix (or list of such matrices, if a by variable is in effect). annotations is a character vector of the annotations that would have been printed below the summary or summaries.
sep	separator for pasting levels in creating row and column names for vcov() results

Value

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

The linfct function and method returns the linfct slot of object.

Examples

```
warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
warp.emm <- emmeans(warp.lm, ~ tension | wool)
vcov(warp.emm) |> zapsmall()

vcov(pairs(warp.emm), sep = "|") |> zapsmall()
```

---

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. summary.emmGrid is the general function for summarizing emmGrid objects. It also serves as the print method for these objects; so for convenience, summary() arguments may be included in calls to functions such as emmeans and contrast that construct emmGrid objects. Note that by default, summaries for Bayesian models are diverted to hpd.summary.

Usage

```
## S3 method for class 'emmGrid'
summary(object, infer, level, adjust, by,
  cross.adjust = "none", type, df, calc, null, delta, side, frequentist,
  bias.adjust = get_emm_option("back.bias.adj"), sigma, ...)

## 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, ...)

## S3 method for class 'emmGrid'
predict(object, type, interval = c("none", "confidence",
  "prediction"), level = 0.95,
  bias.adjust = get_emm_option("back.bias.adj"), sigma, ...)

## S3 method for class 'emmGrid'
as.data.frame(x, row.names = NULL, optional,
  check.names = TRUE, destroy.annotations = FALSE, ...)

## S3 method for class 'summary_emm'
x[..., as.df = FALSE]
```

Arguments

`object`	An object of class `"emmGrid"` (see emmGrid-class)
`infer`	A vector of one or 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.
`cross.adjust`	Character: p-value adjustment method to additionally apply *across* the by groups. See the section on P-value adjustments for details.
`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).
`calc`	Named list of character value(s) or formula(s). The expressions in `char` are evaluated and appended to the summary, just after the `df` column. The expression

may include any names up through `df` in the summary, any additional names in `object@grid` (such as `.wgt.` or `.offset.`), or any earlier elements of `calc`.

`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`, `"-"`, `"<"`, `"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.

`frequentist` Ignored except if a Bayesian model was fitted. If missing or `FALSE`, the object is passed to `hpd.summary`. Otherwise, a logical value of `TRUE` will have it return a frequentist summary.

`bias.adjust` Logical value for whether to adjust for bias in back-transforming (`type = "response"`). This requires a valid value of `sigma` to exist in the object or be specified.

`sigma` Error SD assumed for bias correction (when `type = "response"` and a transformation is in effect), or for constructing prediction intervals. If not specified, `object@misc$sigma` is used, and a warning is issued if it is not found or not valid. *Note:* `sigma` may be a vector, but be careful that it correctly corresponds (perhaps after recycling) to the order of the reference grid.

`...` Optional arguments such as `scheffe.rank` (see "P-value adjustments"). In `confint.emmGrid`, `predict.emmGrid`, and `test.emmGrid`, these arguments are passed to `summary.emmGrid`.

`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 `linfct(object)` 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.

`interval` Type of interval desired (partial matching is allowed): `"none"` for no intervals, otherwise confidence or prediction intervals with given arguments, via `confint.emmGrid`. Note: prediction intervals are not available unless the model family is `"gaussian"`.

x	object of the given class
row.names	passed to as.data.frame
optional	required argument, but ignored in as.data.frame.emmGrid
check.names	passed to data.frame
destroy.annotations	Logical value. If FALSE, an object of class summary_emm is returned (which inherits from data.frame), but if displayed, details like confidence levels, P-value adjustments, transformations, etc. are also shown. But unlike the result of summary, the number of digits displayed is obtained from getOption("digits") rather than using the optimal digits algorithm we usually use. Thus, it is formatted more like a regular data frame, but with any annotations and groupings still intact. If TRUE (not recommended), a "plain vanilla" data frame is returned, based on row.names and check.names.
as.df	Logical value. With x[..., as.df = TRUE], the result is object is coerced to a data.frame before the subscripting is applied. With as.df = FALSE, the result is returned as a summary_emm object when possible.

Details

confint.emmGrid is equivalent to summary.emmGrid with infer = c(TRUE, FALSE). The function test.emmGrid, when called with joint = FALSE, 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. The returned object has an additional "est.fcns" attribute, which is a list of the linear functions associated with the joint test.

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 call is diverted to hpd.summary (with prob set to level, if specified); one may alternatively use general MCMC summarization tools with the results of as.mcmc.

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

The as.data.frame method returns an object that inherits from "data.frame".

Defaults

The misc slot in object may contain default values for by, calc, 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.

Be aware that only univariate transformations and links are supported in this way. Some multivariate transformations are supported by `mvregrid`.

Bias adjustment when back-transforming

When `bias.adjust` is `TRUE`, then back-transformed estimates are adjusted by adding $0.5h''(u)\sigma^2$, where h is the inverse transformation and u is the linear predictor. This is based on a second-order Taylor expansion. There are better or exact adjustments for certain specific cases, and these may be incorporated in future updates.

Note: In certain models, e.g., those with non-gaussian families, `sigma` is initialized as `NA`, and so by default, bias adjustment is skipped and a warning is issued. You may override this by specifying a value for `sigma`. However, *with ordinary generalized linear models, bias adjustment is inappropriate* and you should not try to do it. With GEEs and GLMMs, you probably should *not* use `sigma(model)`, and instead you should create an appropriate value using the estimated random effects, e.g., from `VarCorr(model)`. An example is provided in the "transformations" vignette.

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). If there are non-estimable cases in a `by` group, those cases are *excluded* before determining the adjustment; that means there could be different adjustments in different groups.

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 $\sqrt{rF(\alpha; r, d)}$, where d is the error degrees of freedom and r is the rank of the set of linear functions under consideration. By default, the value of `r` is computed from `linfct(object)` for each by group; however, if the user specifies an argument matching `scheffe.rank`, its value will be used instead. Ordinarily, if there are k means involved, then $r = k - 1$ for a full set of contrasts involving all k means, and $r = k$ for the means themselves. (The Scheffe adjustment is 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.

Warning: Non-estimable cases are *included* in the family to which adjustments are applied. You may wish to subset the object using the `[]` operator to work around this problem.

The `cross.adjust` argument is a way of specifying a multiplicity adjustment across the `by` groups (otherwise by default, each group is treated as a separate family in regards to multiplicity adjustments). It applies only to p values. Valid options are one of the `p.adjust.methods` or `"sidak"`. This argument is ignored unless it is other than `"none"`, there is more than one `by` group, and they are all the same size. Under those conditions, we first use `adjust` to determine the within-group adjusted p values. Imagine each group's adjusted p values arranged in side-by-side columns, thus forming a matrix with the number of columns equal to the number of `by` groups. Then we use the `cross.adjust` method to further adjust the adjusted p values in each row of this matrix. Note that an *overall* Bonferroni (or Sidak) adjustment is obtainable by specifying *both* `adjust` and `cross.adjust` as `"bonferroni"` (or `"sidak"`). However, less conservative (but yet conservative) overall adjustments are available when it is possible to use an "exact" within-group method (e.g., `adjust = "tukey"` for pairwise comparisons) and `cross.adjust` as a conservative adjustment. [`cross.adjust` methods other than `"none"`, `"bonferroni"`, or `"sidak"` do not seem advisable, but other `p.adjust` methods are available if you can make sense of them.]

Tests of significance, nonsuperiority, noninferiority, or equivalence

When `delta = 0`, test statistics are the usual tests of significance. They are of the form '`(estimate - null)/SE`'. Notationally:

Significance $H_0 : \theta = \theta_0$ versus
$H_1 : \theta < \theta_0$ (left-sided), or
$H_1 \theta > \theta_0$ (right-sided), or
$H_1 : \theta \neq \theta_0$ (two-sided)
The test statistic is
$t = (Q - \theta_0)/SE$

where Q is our estimate of θ; then left, right, or two-sided p values are produced, depending on `side`.

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.
Note that t is the maximum of t_{nonsup} and $-t_{noninf}$. This is equivalent to choosing the less significant result in the two-one-sided-test (TOST) procedure.

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.

See the warning above that non-estimable cases are still included when determining the family size for *P*-value adjustments.

Warning about potential misuse of P values

Some in the statistical and scientific community argue that the term "statistical significance" should be completely abandoned, and that criteria such as "p < 0.05" never be used to assess the importance of an effect. These practices can be too misleading and are prone to abuse. See the "basics" vignette for more discussion.

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 (often) is not a natural way to back-transform it.

When we have `type = "response"`, and `bias.adj = TRUE`, the `null` value displayed in the output is both back-transformed and bias-adjusted, leading to a rather non-intuitive-looking null value. However, since the tests themselves are performed on the link scale, this is the response value at which a *P* value of 1 would be obtained.

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.

If a data frame is needed, `summary`, `confint`, and `test` serve this need. `as.data.frame` routes to `summary` by default; calling it with `destroy.annotations = TRUE` is not recommended for exactly that reason. If you want to see more digits in the output, use `print(summary(object), digits = ...)`; and if you *always* want to see more digits, use `emm_options(opt.digits = FALSE)`.

See Also

`hpd.summary`

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")
summary(pigs.emm, calc = c(n = ".wgt."))  # Show sample size

# 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 = ">")

con <- contrast(pigs.emm, "consec")
test(con)

test(con, joint = TRUE)

# default Scheffe adjustment - rank = 3
summary(con, infer = c(TRUE, TRUE), adjust = "scheffe")

# Consider as some of many possible contrasts among the six cell means
summary(con, infer = c(TRUE, TRUE), adjust = "scheffe", scheffe.rank = 5)

# Show estimates to more digits
print(test(con), digits = 7)

# ------------------------------------------------------------
# Cross-adjusting P values
prs <- pairs(warp.emm)   # pairwise comparisons of tension, by wool
test(prs, adjust = "tukey", cross.adjust = "bonferroni")

# Same comparisons taken as one big family (more conservative)
test(prs, adjust = "bonferroni", by = NULL)
```

ubds *Unbalanced dataset*

Description

This is a simulated unbalanced dataset with three factors and two numeric variables. There are true relationships among these variables. This dataset can be useful in testing or illustrating messy-data situations. There are no missing data, and there is at least one observation for every factor combination; however, the `"cells"` attribute makes it simple to construct subsets that have empty cells.

Usage

```
ubds
```

Format

A data frame with 100 observations, 5 variables, and a special `"cells"` attribute:

A Factor with levels 1, 2, and 3

B Factor with levels 1, 2, and 3

C Factor with levels 1, 2, and 3

x A numeric variable

y A numeric variable

In addition, `attr(ubds, "cells")` consists of a named list of length 27 with the row numbers for each combination of `A, B, C`. For example, `attr(ubds, "cells")[["213"]]` has the row numbers corresponding to levels `A == 2, B == 1, C == 3`. The entries are ordered by length, so the first entry is the cell with the lowest frequency.

Examples

```
# Omit the three lowest-frequency cells
low3 <- unlist(attr(ubds, "cells")[1:3])
messy.lm <- lm(y ~ (x + A + B + C)^3, data = ubds, subset = -low3)
```

untidy *Dare to be un-"tidy"!*

Description

Users who use **emmeans** functions as part of a pipeline – or post-process those results in some other way – are likely missing some important information.

Details

Your best bet is to display the actual results without any post-processing. That's because `emmeans` and its relatives have their own `summary` and `print` methods that display annotations that may be helpful in explaining what you have. If you just pipe the results into the next step, those annotations are stripped away and you never see them. Statistical analysis is not just a workflow; it is a discipline that involves care in interpreting intermediate results, and thinking before moving on.

Examples

```
neur.glm <- glm(Pain ~ Treatment + Sex + Age, family = binomial(),
            data = neuralgia)

### The actual results with annotations (e.g. ests are on logit scale):
emmeans(neur.glm, "Treatment")

### Post-processed results lose the annotations
if(requireNamespace("tibble")) {
    emmeans(neur.glm, "Treatment") |> tibble::as_tibble()
}
```

update.emmGrid *Update an* `emmGrid` *object*

Description

Objects of class `emmGrid` contain several settings that affect such things as what arguments to pass to `summary.emmGrid`. The `update` method allows safer management of these settings than by direct modification of its slots.

Usage

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

## S3 replacement method for class 'emmGrid'
levels(x) <- value
```

```
## S3 method for class 'summary_emm'
update(object, by.vars, mesg, ...)
```

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	an emmGrid object
value	list or replacement levels. See the documentation for update.emmGrid with the levels argument, as well as the section below on "Replaciong levels"
by.vars, mesg	Attributes that can be altered in update.summary_emm

Value

an updated emmGrid object.

levels<- replaces the levels of the object in-place. See the section on replacing levels for details.

Details

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.

tran.offset Additive constant before a transformation is applied. For example, a response transformation of log(y + pi) has tran.offset = pi. If no value is present, an offset of 0 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 `plot.emmGrid` when '`CIs = TRUE`'. Be cautious that methods that depend on specifying `alpha` are prone to abuse. See the discussion in `vignette("basics", "emmeans")`.

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

`cross.adjust` (`character`)) is the default for the `cross.adjust` argument in `summary.emmGrid` (used for adjusting between groups).

`famSize` (integer) is the number of means involved in a family of inferences; used in Tukey adjustment

`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`. *Note:* You must specify all five letters of 'level' to distinguish it from the slot name 'levels'.

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

`calc` (list) additional calculated columns. See `summary.emmGrid`.

`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).

`sigma` (numeric) Error SD to use in predictions and for bias-adjusted back-transformations

`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"`.

`bias.adjust`**,** `frequentist` (logical) These are used by `summary` if the value of these arguments are not specified.

`estType` (`character`) is used internally to determine what `adjust` methods are appropriate. It should match one of '`c("prediction", "contrast", "pairs")`'. As an example of why this is needed, the Tukey adjustment should only be used for pairwise comparisons (`estType = "pairs"`); if `estType` is some other string, Tukey adjustments are not allowed.

`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`.

`levels` named `list` of new levels for the elements of the current `emmGrid`. The list name(s) are used as new variable names, and if needed, the list is expanded using `expand.grid`. These results replace current variable names and levels. This specification changes the `levels`, `grid`, `roles`, and `misc` slots in the updated `emmGrid`, and resets `pri.vars`, `by.vars`, `adjust`, `famSize`, and `avgd.over`. In addition, if there is nesting of factors, that may be altered; a warning is issued if it involves something other than mere name changes. *Note:* All six letters of `levels` is needed in order to distinguish it from `level`.

`submodel` `formula` or `character` value specifying a submodel (requires this feature being supported by underlying methods for the model class). When specified, the `linfct` slot is replaced by its aliases for the specified sub-model. Any factors in the sub-model that do not appear in the model matrix are ignored, as are any interactions that are not in the main model, and any factors associate with multivariate responses. The estimates displayed are then computed as if the sub-model had been fitted. (However, the standard errors will be based on the error variance(s) of the full model.) *Note:* The formula should refer only to predictor names, *excluding* any function calls (such as `factor` or `poly`) that appear in the original model formula. See the example.

The character values allowed should partially match `"minimal"` or `"type2"`. With `"minimal"`, the sub-model is taken to be the one only involving the surviving factors in `object` (the ones averaged over being omitted). Specifying `"type2"` is the same as `"minimal"` except only the highest-order term in the submodel is retained, and all effects not containing it are orthogonalized-out. Thus, in a purely linear situation such as an `lm` model, the joint test of the modified object is in essence a type-2 test as in `car::Anova`.

Please note that it is possible (or even likely) that there will be disparity between the `grid` and `linfct` slots when a submodel is used. This is because `grid` contains the *claimed* values of the predictors and `linfct` contains *aliases* of them computed from the submodel.

For some objects such as generalized linear models, specifying `submodel` will typically not produce the same estimates or type-2 tests as would be obtained by actually fitting a separate model with those specifications. The reason is that those models are fitted by iterative-reweighting methods, whereas the `submodel` calculations preserve the final weights used in fitting the full model.

(any other slot name) If the name matches an element of `slotNames(object)` other than `levels`, that slot is replaced by the supplied value, if it is of the required class (otherwise an error occurs).

The user must be very careful in replacing slots because they are interrelated; for example, the lengths and dimensions of `grid`, `linfct`, `bhat`, and `V` must conform.

Replacing levels

The `levels<-` method uses `update.emmGrid` to replace the levels of one or more factors. This method allows selectively replacing the levels of just one factor (via subsetting operators), whereas `update(x, levels = list(...))` requires a list of *all* factors and their levels. If any factors are to be renamed, we must replace all levels and include the new names in the replacements. See the examples.

Method for `summary_emm` objects

This method exists so that we can change the way a summary is displayed, by changing the by variables or the annotations.

Note

When it makes sense, an option set by `update` will persist into future results based on that object. But some options are disabled as well. For example, a `calc` option will be nulled-out if `contrast` is called, because it probably will not make sense to do the same calculations on the contrast results, and in fact the variable(s) needed may not even still exist. `factor(percent)`.

See Also

emm_options

Examples

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

# Reference grid that knows about the transformation
# and asks to include the sample size in any summaries:
pigs.rg <- update(ref_grid(pigs.lm), tran = "log",
                  predict.type = "response",
                  calc = c(n = ~.wgt.))
emmeans(pigs.rg, "source")

# Obtain estimates for the additive model
# [Note that the submodel refers to 'percent', not 'factor(percent)']
emmeans(pigs.rg, "source", submodel = ~ source + percent)

# Type II ANOVA
joint_tests(pigs.rg, submodel = "type2")

## Changing levels of one factor
newrg <- pigs.rg
levels(newrg)$source <- 1:3
newrg

## Unraveling a previously standardized covariate
zd = scale(fiber$diameter)
fibz.lm <- lm(strength ~ machine * zd, data = fiber)
(fibz.rg <- ref_grid(fibz.lm, at = list(zd = -2:2)))   ### 2*SD range
lev <- levels(fibz.rg)
levels(fibz.rg) <- list (
    machine = lev$machine,
    diameter = with(attributes(zd),
                    `scaled:center` + `scaled:scale` * lev$zd) )
fibz.rg

### Compactify results with a by variable
update(joint_tests(pigs.rg, by = "source"), by = NULL)
```

---

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

```
if(requireNamespace("xtable"))
    emm_example("xtable")
    # Use emm_example("xtable", list = TRUE) # to just list the code
```

Index