

# Package ‘emmeans’

November 28, 2019

**Type** Package

**Title** Estimated Marginal Means, aka Least-Squares Means

**Version** 1.4.3.01

**Date** 2019-11-27

**Depends** R (>= 3.2)

**Imports** estimability (>= 1.3), graphics, methods, numDeriv, stats,  
utils, plyr, mvtnorm, xtable (>= 1.8-2)

**Suggests** bayesplot, biglm, brms, car, coda (>= 0.17), ggplot2,  
lattice, mediation, mgcv, multcomp, multcompView, nlme, ordinal  
(>= 2014.11-12), pbkrtest (>= 0.4-1), lme4, lmerTest (>=  
2.0.32), MASS, MuMIn, rsm, knitr, rmarkdown, scales, testthat

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

**URL** <https://github.com/rvlenth/emmeans>

**BugReports** <https://github.com/rvlenth/emmeans/issues>

**LazyData** yes

**ByteCompile** yes

**Description** Obtain estimated marginal means (EMMs) for many linear, generalized linear, and mixed models. Compute contrasts or linear functions of EMMs, trends, and comparisons of slopes. Plots and 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** 6.1.1

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Russell Lenth [aut, cre, cph],  
 Henrik Singmann [ctb],  
 Jonathon Love [ctb],  
 Paul Buerkner [ctb],  
 Maxime Herve [ctb]

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

**Repository** CRAN

**Date/Publication** 2019-11-28 11:50:05 UTC

## R topics documented:

emmeans-package . . . . .	3
add_grouping . . . . .	4
as.emmGrid . . . . .	5
as.mcmc.emmGrid . . . . .	6
auto.noise . . . . .	8
CLD . . . . .	9
contrast . . . . .	11
contrast-methods . . . . .	14
eff_size . . . . .	17
emm . . . . .	19
emmeans . . . . .	20
emmGrid-class . . . . .	23
emmip . . . . .	25
emmobj . . . . .	27
emm_list . . . . .	28
emm_options . . . . .	29
emtrends . . . . .	31
extending-emmeans . . . . .	33
feedlot . . . . .	36
fiber . . . . .	37
hpd.summary . . . . .	38
joint_tests . . . . .	39
lsmeans . . . . .	41
make.tran . . . . .	42
MOats . . . . .	44
models . . . . .	45
neuralgia . . . . .	46
nutrition . . . . .	47
oranges . . . . .	48
pigs . . . . .	49
plot.emmGrid . . . . .	49
pwpp . . . . .	51
qdrq . . . . .	53
rbind.emmGrid . . . . .	55
ref_grid . . . . .	56
regrid . . . . .	61

str.emmGrid . . . . .	63
summary.emmGrid . . . . .	64
update.emmGrid . . . . .	70
xtable.emmGrid . . . . .	72

<b>Index</b>	<b>74</b>
--------------	-----------

---

emmeans-package	<i>Estimated marginal means (aka Least-squares means)</i>
-----------------	---

---

## 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.

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

---

add\_grouping

*Add a grouping factor*

---

## Description

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

## Usage

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

## Arguments

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

**Value**

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

**Note**

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

**Examples**

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

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

emmmeans(gfrg, "machine")

emmmeans(gfrg, "brand")
```

---

as.emmGrid

---

*Convert to and from emmGrid objects*


---

**Description**

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

**Usage**

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

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

**Arguments**

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

**Details**

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

**Value**

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

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

**See Also**

[emmobj](#)

**Examples**

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

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

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

## End(Not run)
```

**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 'emmGrid'
as.mcmc.list(x, names = TRUE, ...)
```

**Arguments**

x	An object of class <code>emmGrid</code>
names	Logical scalar or vector specifying whether variable names are appended to levels in the column labels for the <code>as.mcmc</code> or <code>as.mcmc.list</code> result – e.g., column names of <code>treat A</code> and <code>treat B</code> versus just <code>A</code> and <code>B</code> . When there is more than one variable involved, the elements of <code>names</code> are used cyclically.
sep.chains	Logical value. If <code>TRUE</code> , and there is more than one MCMC chain available, an <code>mcmc.list</code> object is returned by <code>as.mcmc</code> , 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 <code>TRUE</code> , non-estimable columns are kept but returned as columns of NA values (this may create errors or warnings in subsequent analyses using, say, <code>coda</code> ). If <code>FALSE</code> , non-estimable columns are dropped, and a warning is issued. (If all are non-estimable, an error is thrown.)
...	arguments passed to other methods

**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,  $\theta$  is the parameter estimated in object, and  $\phi$  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  $\theta$  values (each corresponding to one row

of `object@grid`). Any  $\phi$  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  $\theta$  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  $\theta$  and shape parameter specified by `shape`

## Examples

```
require("coda")

### A saved reference grid for a mixed logistic model (see lme4::cbpp)
cbpp.rg <- do.call(emmobj,
  readRDS(system.file("extdata", "cbpplist", package = "emmeans")))
# Predictive distribution for herds of size 20
# (perhaps a bias adjustment should be applied; see "sophisticated" vignette)
pred.incidence <- as.mcmc(regrid(cbpp.rg), likelihood = "binomial", trials = 20)
```

---

auto.noise

*Auto Pollution Filter Noise*

---

## Description

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

## Usage

```
auto.noise
```

## Format

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

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

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

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

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

**Details**

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

**Source**

The dataset was obtained from the Data and Story Library (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**

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

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

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

---

CLD	<i>Extract and display information on all pairwise comparisons of estimated marginal means.</i>
-----	---

---

**Description**

Extract and display information on all pairwise comparisons of estimated marginal means.

**Usage**

```
CLD(object, ...)

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

cld.emmGrid(object, ...)
```

**Arguments**

object	An object of class <code>emmGrid</code>
...	Arguments passed to <code>contrast</code> (for example, an <code>adjust</code> method)
	This function uses the Piepho (2004) algorithm (as implemented in the <b>multcompView</b> 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 <code>contrast</code> function with <code>method = "pairwise"</code> . When a P value exceeds <code>alpha</code> , then the two means have at least one letter in common.
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 <code>by.vars</code> setting, if any, is used.
alpha	Numeric value giving the significance level for the comparisons
Letters	Character vector of letters to use in the display. Any strings of length greater than 1 are expanded into individual characters
reversed	Logical value (passed to <code>multcompView::multcompLetters</code> .) If TRUE, the order of use of the letters is reversed. In addition, if both <code>sort</code> and <code>reversed</code> are TRUE, the sort order of results is reversed.

**Value**

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

**Deprecated**

The CLD function and methods are deprecated. Compact-letter displays (CLDs) encourage a misleading interpretation of significance testing by visually grouping means whose comparisons have  $P > \alpha$  as though they are equal. However, failing to prove two means are different does not prove that they are the same. In addition, CLDs make a hard distinction between  $P$  values nearly equal to  $\alpha$  but on opposite sides.

Some users may find `pwpp` to be a useful alternative. It produces a plot showing all P values for all pairwise comparisons (or other set of comparisons), and can also show one-sided P values and tests of equivalence or noninferiority. Also, if you insist, `cld` is still available in the **multcomp** package, and `multcomp::cld(emm)` still works.

**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.

**See Also**

`cld` in the **multcomp** package, for which a `courtesy` method is provided for `emmGrid` objects.

---

 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, ...)

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

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

**Arguments**

<code>object</code>	An object of class <code>emmGrid</code>
<code>...</code>	Additional arguments passed to other methods
<code>method</code>	Character value giving the root name of a contrast method (e.g. "pairwise" – see <a href="#">emmc-functions</a> ). 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.
<code>interaction</code>	Character vector, logical value, or list. If this is specified, <code>method</code> is ignored. See the "Interaction contrasts" section below for details.
<code>by</code>	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 <code>object</code> was created with by groups, those are used unless overridden. Use <code>= NULL</code> 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 <code>update.emmGrid</code> , 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) <i>not</i> 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.
x	An emmGrid object
reverse	Logical value - determines whether to use "pairwise" (if TRUE) or "revpairwise" (if FALSE).

### Value

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

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

### Pairs method

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

### Interaction contrasts

When interaction is specified, interaction contrasts are computed. 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
pairs(warp.emm)           # ditto
contrast(warp.emm, "eff", by = NULL) # contrasts of the 6 factor combs
pairs(warp.emm, simple = "wool") # same as pairs(warp.emm, by = "tension")

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

## Not run:

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

```
## End(Not run)

# An interaction contrast for tension:wool
tw.emm <- contrast(warp.emm, interaction = c(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), ...)
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, ...)
del.eff.emmc(levs, exclude = integer(0), include, ...)
consec.emmc(levs, reverse = FALSE, exclude = integer(0), include, ...)
```

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

```
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
ref	Integer(s) or character(s) specifying which level(s) to use as the reference. Character values must exactly match elements of levs.

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

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

`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 levels. 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.ctrl1", ref = "M")

# 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))

### Setting up a custom contrast function
helmert.emmc <- function(levs, ...) {
  M <- as.data.frame(contr.helmert(levs))
  names(M) <- paste(levs[-1], "vs earlier")
  attr(M, "desc") <- "Helmert contrasts"
  M
}
contrast(warp.emm, "helmert")
## Not run:
# See what is used for polynomial contrasts with 6 levels
emmeans:::poly.emmc(1:6)
```

```
## End(Not run)
```

---

eff_size	<i>Calculate effect sizes and confidence bounds thereof</i>
----------	---

---

## 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

<code>object</code>	an <code>emmGrid</code> object, typically one defining the EMMs to be contrasted.
<code>sigma</code>	numeric scalar, value of the population SD.
<code>edf</code>	numeric scalar that specifies the equivalent degrees of freedom for the <code>sigma</code> . This is a way of specifying the uncertainty in <code>sigma</code> , in that we regard our estimate of <code>sigma^2</code> as being proportional to a chi-square random variable with <code>edf</code> degrees of freedom. ( <code>edf</code> should not be confused with the <code>df</code> argument that may be passed via <code>...</code> to specify the degrees of freedom to use in <i>t</i> statistics and confidence intervals.)
<code>method</code>	the contrast method to use to define the effects. This is passed to <code>contrast</code> after the elements of <code>object</code> are scaled.
<code>...</code>	Additional arguments passed to <code>contrast</code>

## 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  $\sqrt{2/\text{edf}}$  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  $\log(\text{EMM}/\sigma)$ . 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.

**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).

**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)) {

  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)
  print(eff_size(emmV, sigma = totSD, edf = 5))
  print(eff_size(emmV, sigma = totSD, edf = 51))
}

```

```
# Multivariate model for the same data:
MOats.lm <- lm(yield ~ Variety, data = MOats)
eff_size(emmeans(MOats.lm, "Variety"),
         sigma = sqrt(mean(sigma(MOats.lm)^2)), # RMS of sigma()
         edf = df.residual(MOats.lm))

# These results illustrate a sobering message that effect sizes are often
# not nearly as accurate as you may think.
```

---

 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 that are passed to other methods.

`object` An object of class `emmGrid` or `emm_list`

## Details

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

## Value

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

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

## Note

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

## Examples

```
if(require(multcomp)) { # --- multcomp must be installed

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

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

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

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

} # --- was tested only if multcomp is installed
```

---

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, trend, ...)
```

**Arguments**

object	An object of class <code>emmGrid</code> ; or a fitted model object that is supported, such as the result of a call to <code>lm</code> or <code>lmer</code> . Many fitted-model objects are supported; see <code>vignette("models", "emmeans")</code> for details.
specs	A character vector specifying the names of the predictors over which EMMs are desired. <code>specs</code> may also be a formula or a list (optionally named) of valid specs. Use of formulas is described in the Overview section below.
by	A character vector specifying the names of predictors to condition on.
fac.reduce	A function that combines the rows of a matrix into a single vector. This implements the “marginal averaging” aspect of EMMs. The default is the mean of the rows. Typically if it is overridden, it would be some kind of weighted mean of the rows. If <code>fac.reduce</code> is nonlinear, bizarre results are likely, and EMMs will not be interpretable. NOTE: If the <code>weights</code> argument is non-missing, <code>fac.reduce</code> is ignored.
contr	A character value or list specifying contrasts to be added. See <code>contrast</code> . NOTE: <code>contr</code> is ignored when <code>specs</code> is a formula.
options	If non-NULL, a named list of arguments to pass to <code>update.emmGrid</code> , just after the object is constructed.
weights	Character value, numeric vector, or numeric matrix specifying weights to use in averaging predictions. See “Weights” section below.
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.
trend	This is now deprecated. Use <code>emttrends</code> instead.
...	This is used only when <code>object</code> is not already a “ <code>emmGrid</code> ” object, these arguments are passed to <code>ref_grid</code> . Common examples are <code>at</code> , <code>cov.reduce</code> , <code>data</code> , <code>codetype</code> , <code>transform</code> , <code>df</code> , <code>nesting</code> , and <code>vcov..</code> Model-type-specific options (see <code>vignette("models", "emmeans")</code> ), commonly <code>mode</code> , may be used here as well. In addition, if the model formula contains references to variables that are not predictors, you must provide a <code>params</code> argument with a list of their names.

**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`, `pairs.emmGrid`, and `CLD.emmGrid`. When `specs` is a list or a formula having a left-hand side, the return value is an `emm_list` object, which is simply a list of `emmGrid` objects.

## 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, but if specified it should be the name of a contrast family (e.g., `pairwise`). Operators like `*` or `:` are needed in the formula to delineate names, but otherwise are ignored.

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

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

## Weights

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

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

"equal" Use an equally weighted average.

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

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

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

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

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

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

**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.

**See Also**

[ref\\_grid](#), [contrast](#), [vignette\("models", "emmeans"\)](#)

**Examples**

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

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

## 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 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 `codata`), `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, engine = get_emm_option("graphics.engine"), pch = c(1,
       2, 6, 7, 9, 10, 15:20), lty = 1, col = NULL, plotit = TRUE, ...)
```

**Arguments**

object	An object of class <code>emmGrid</code> , or a fitted model of a class supported by the <b>emmeans</b> package
formula	Formula of the form <code>trace.factors ~ x.factors   by.factors</code> . The EMMs are plotted against <code>x.factor</code> for each level of <code>trace.factors</code> . <code>by.factors</code> 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 <code>*</code> operator.
...	Additional arguments passed to <code>emmeans</code> (when <code>object</code> is not already an <code>emmGrid</code> object), <code>predict.emmGrid</code> , <code>ggplot</code> , or <code>xyplot</code> .
type	As in <code>predict.emmGrid</code> , this determines whether we want to inverse-transform the predictions ( <code>type = "response"</code> ) or not (any other choice). The default is <code>"link"</code> , unless the <code>"predict.type"</code> option is in force; see <code>emm_options</code> .
CIs	Logical value. If <code>TRUE</code> , confidence intervals (or HPD intervals for Bayesian models) are added to the plot (works only with <code>engine = "ggplot"</code> ).
PIs	Logical value. If <code>TRUE</code> , prediction intervals are added to the plot (works only with <code>engine = "ggplot"</code> ). If both <code>CIs</code> and <code>PIs</code> are <code>TRUE</code> , the prediction intervals will be somewhat longer, lighter, and thinner than the confidence intervals. Additional parameters to <code>predict.emmGrid</code> (e.g., <code>sigma</code> ) may be passed via <code>...</code> . For Bayesian models, <code>PIs</code> require <code>frequentist = TRUE</code> and a value for <code>sigma</code> .
engine	Character value matching <code>"ggplot"</code> (default) or <code>"lattice"</code> . The graphics engine to be used to produce the plot. These require, respectively, the <b>ggplot2</b> or <b>lattice</b> package to be installed.
pch	The plotting characters to use for each group (i.e., levels of <code>trace.factors</code> ). They are recycled as needed.
lty	The line types to use for each group. Recycled as needed.

col	The colors to use for each group, recycled as needed. If not specified, the default trellis colors are used.
plotit	Logical value. If TRUE, the plot is displayed. Otherwise, one may use the "lattice" attribute of the returned object and print it, perhaps after additional manipulation.

### Value

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

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

### Details

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

### Note

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

### See Also

`emmeans`, `interaction.plot`

### Examples

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

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

# One interaction plot, using combinations of size and side as the x factor
# ... with added confidence intervals
emmip(noise.lm, type ~ side * size, CIs = TRUE)

# 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)
```



**See Also**

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

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

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

---

 emm\_list

*The emm\_list class*


---

**Description**

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

**Details**

Methods for `emm_list` objects include `summary`, `CLD`, `coef`, `confint`, `contrast`, `pairs`, `plot`, `print`, and `test`. These are all the same as those methods for `emmGrid` objects, with an additional `which` argument (integer) to specify which members of the list to use. The default is `which = seq_along(object)`; i.e., the method is applied to every member of the `emm_list` object. The exception is `plot`, where only the `which[1]`th element is plotted.

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

---

emm_options	<i>Set or change emmeans options</i>
-------------	--------------------------------------

---

### 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(...)  
  
get_emm_option(x, default = emm_defaults[[x]])  
  
emm_defaults
```

### Arguments

<code>...</code>	Option names and values (see Details)
<code>x</code>	Character value - the name of an option to be queried
<code>default</code>	Value to return if <code>x</code> is not found

### Format

An object of class `list` of length 17.

### Details

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".

`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.

`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*, a second-order adjustment is applied to estimate the mean on the response scale.

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`

**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.

## 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")
```

emrends

*Estimated marginal means of linear trends***Description**

The `emrends` 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**

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

**Arguments**

<code>object</code>	A supported model object ( <i>not</i> a reference grid)
<code>specs</code>	Specifications for what marginal trends are desired – as in <a href="#">emmeans</a> . If <code>specs</code> is missing or <code>NULL</code> , <code>emmeans</code> is not run and the reference grid for specified trends is returned.
<code>var</code>	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, <code>var</code> should be a numeric predictor that interacts with at least one factor in <code>specs</code> . Then instead of computing EMMs, we compute and compare the slopes of the <code>var</code> trend over levels of the specified other predictor(s). As in EMMs, marginal averages are computed for the predictors in <code>specs</code> and <code>by</code> . See also the “Generalizations” section below.
<code>delta.var</code>	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 <code>var</code> over the dataset.
<code>max.degree</code>	Integer value. The maximum degree of trends to compute (this is capped at 5). If greater than 1, an additional factor <code>degree</code> is added to the grid, with corresponding numerical derivatives of orders 1, 2, . . . , <code>max.degree</code> 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 `emttrends`, 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 `emttrends` 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

`link{emmeans}`, `ref_grid`

## Examples

```

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

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

# Obtaining a reference grid
mtcars.lm <- lm(mpg ~ poly(dis, 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 the methods that `ref_grid` calls. A user or package developer may add **emmeans** support for a model class by writing `recover_data` and `emm_basis` methods for that class.

## Usage

```
recover_data(object, ...)
```

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

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

.recover_data(object, ...)

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

.emm_register(classes, pkgname)
```

### 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 <a href="#">terms</a> component of object (typically with the response deleted, e.g. via <a href="#">delete.response</a> )
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 <i>not</i> 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.
xlev	Named list of factor levels ( <i>excluding</i> ones coerced to factors in the model formula)
grid	A <code>data.frame</code> (provided by <code>ref_grid</code> ) 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 <code>recover_data.foo</code> and <code>emm_basis.foo</code> for each class <code>foo</code> listed in <code>classes</code> .
pkgname	Character name of package providing the methods (usually should be the second argument of <code>.onLoad</code> )

### 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  $\text{sum}(k * \text{bhat})$ .

**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.

## 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 be the same as `'object@linfct'`. The `vcovHook` function should have arguments `'(object, tol, ...)'` as described. It should return the covariance matrix for the estimates. Finally, `postGridHook`, if present, is called at the very end of `ref_grid`; it takes one argument, the constructed object, and should return a suitably modified `emmGrid` object.

### 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.

### 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
emmmeans(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
emmmeans(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,
  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 <a href="#">emm_options</a> )
by	factors to use as by variables
type	prediction type as in <a href="#">summary.emmGrid</a>
point.est	function to use to compute the point estimates from the posterior sample for each grid point
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 an error is thrown if it is not found. <i>Note:</i> 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`

**See Also**

[summary.emmGrid](#)

**Examples**

```
if(require("coda")) {
  # Create an emmGrid object from a system file
  cbpp.rg <- do.call(emmobj,
    readRDS(system.file("extdata", "cbplist", package = "emmeans")))
  hpd.summary(emmeans(cbpp.rg, "period"))
}
```

---

joint\_tests

*Compute joint tests of the terms in a model*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**See Also**

[test](#)

**Examples**

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

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

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

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

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

```

# (These two models have identical fitted values and residuals)

joint_tests(toy.fac)

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

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

```

---

lsmeans

*Wrappers for alternative naming of EMMs*


---

## Description

These are wrappers for [emmeans](#) and related functions to provide backward compatibility, or for users who may prefer to use other terminology than “estimated marginal means” – namely “least-squares means” or “predicted marginal means”.

## Usage

```
lsmeans(...)
```

```
pmmeans(...)
```

```
lstrends(...)
```

```
pmtrends(...)
```

```
lsmip(...)
```

```
pmcip(...)
```

```
lsm(...)
```

```
pmm(...)
```

```
lsmobj(...)
```

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

### Arguments

...	Arguments passed to the corresponding <code>emxxx</code> function
x	Character name of desired option
default	default value to return if x not found

### Details

For each function with `lsxxx` or `pmxxx` in its name, the same function named `emxxx` is called. Any estimator names or list items beginning with “em” are replaced with “ls” or “pm” before the results are returned

### Value

The result of the call to `emxxx`, suitably modified.

`get.lsm.option` and `lsm.options` remap options from and to corresponding options in the **lsmeans** 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",
  "bcnPower"), param = 1)
```

## Arguments

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

## Details

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

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

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

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

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

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

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

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

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

"bcnPower" Box-Cox with negatives allowed, as described for the `bcnPower` function in the `car` package. It is defined as the Box-Cox transformation  $(z^p - 1)/p$  of the variable  $z = y + (y^2 + g^2)^{1/2}$ . This requires `param` to have two elements: the power  $p$  and the offset  $g > 0$ .

The user may include a second element in `param` to specify an alternative origin (other than zero) for the "power", "boxcox", or "sympower" transformations. For example, `'type = "power", param = c(1.5, 4)'` specifies the transformation  $(y - 4)^{1.5}$ . In the "genpower" transformation, a second

param element may be used to specify a base other than the default natural logarithm. For example, 'type = "genlog", param = c(.5, 10)' specifies the  $\log_{10}(y + .5)$  transformation. In the "bcnPower" transformation, the second element is required and must be positive.

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

### Value

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

### Note

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")

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

## End(Not run)
```

---

MOats

*Oats data in multivariate form*


---

### Description

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

### Usage

MOats

**Format**

A data frame with 18 observations and 3 variables

Variety a factor with levels Golden Rain, Marvellous, Victory

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

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

**Details**

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

**Source**

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

**References**

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

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

**Examples**

```
M0ats.lm <- lm (yield ~ Block + Variety, data = M0ats)
M0ats.rg <- ref_grid (M0ats.lm, mult.name = "nitro")
emmeans(M0ats.rg, ~ nitro | Variety)
```

---

models

*Models supported in emmeans*

---

**Description**

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

---

neuralgia

*Neuralgia data*

---

### Description

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

### Usage

neuralgia

### Format

A data frame with 60 observations and 5 variables:

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

Sex Factor with two levels F and M

Age Numeric covariate – patient’s age in years

Duration Numeric covariate – duration of the condition before beginning treatment

Pain Binary response factor with levels No and Yes

### Source

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

### Examples

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

---

nutrition

*Nutrition data*

---

### Description

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

### Usage

```
nutrition
```

### Format

A data frame with 107 observations and 4 variables:

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

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

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

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

### Details

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

### Source

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

### Examples

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

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

emmip(nutr.emm, race ~ group)

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

---

oranges

*Sales of oranges*

---

### Description

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

### Usage

oranges

### Format

A data frame with 36 observations and 6 variables:

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

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

price1 a numeric vector. Price of variety 1.

price2 a numeric vector. Price of variety 2.

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

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

### Source

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

### References

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

### Examples

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

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

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

---

pigs

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

---

### Description

A two-factor experiment with some observations lost

### Usage

pigs

### Format

A data frame with 29 observations and 3 variables:

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

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

**conc** Concentration of free plasma leucine, in mcg/ml

### Source

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

### Examples

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

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

**Arguments**

x	Object of class <code>emmGrid</code> or <code>summary_emm</code>
y	(Required but ignored)
type	Character value specifying the type of prediction desired (matching <code>"linear.predictor"</code> , <code>"link"</code> , or <code>"response"</code> ). See details under <a href="#">summary.emmGrid</a> .
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 <code>frequentist = TRUE</code> and <code>sigma = (some value)</code> . Note that the PIs option is <i>not</i> available with <code>summary_emm</code> objects – only for <code>emmGrid</code> objects. Also, prediction intervals are not available with <code>engine = "lattice"</code> .
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.)
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 <i>only</i> .
int.adjust	Character value: Multiplicity adjustment method for the plotted confidence intervals <i>only</i> .
intervals	If specified, it is used to set CIs. This is the previous name of CIs and is provided for backward compatibility.
frequentist	Logical value. If there is a posterior MCMC sample and <code>frequentist</code> is non-missing and TRUE, a frequentist summary is used for obtaining the plot data, rather than the posterior point estimate and HPD intervals. This argument is ignored when it is not a Bayesian model.
...	Additional arguments passed to <a href="#">update.emmGrid</a> , <a href="#">predict.emmGrid</a> , or <a href="#">dotplot</a>
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 <code>dotplot</code>

### Details

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

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

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

### Examples

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

---

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 = "", add.space = 0, ...)
```

### Arguments

emm	An emmGrid object
method	Character or list. Passed to <code>contrast</code> , 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 <code>contrast(object, method)</code> 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 <i>primary</i> 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 <i>not</i> 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.
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).
...	Additional arguments passed to contrast and <a href="#">summary.emmGrid</a>

### 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

The **ggplot2** package must be installed in order for pwpp to work.

**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 = ">")
```

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, mcmc, vcov, object, df, subset, weights,
      contrasts, link, qr, ...)
```

**Arguments**

formula	Formula for the fixed effects
data	Dataset containing the variables in the model
coef	Fixed-effect regression coefficients (must conform to formula)
mcmc	Posterior sample of fixed-effect coefficients
vcov	Variance-covariance matrix of the fixed effects
object	Optional model object. If provided, it is used to set certain other arguments, if not specified. See Details.
df	Error degrees of freedom
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; needed only if there are NAs in coef.
...	Optional arguments passed to <a href="#">ref_grid</a>

## Details

If `object` is specified, it is used to try to obtain certain other arguments, as detailed below. The user should ensure that these defaults will work. The default values for the arguments are as follows:

- `formula`: Required unless obtainable via `formula(object)`
- `data`: Required if variables are not in `parent.frame()` or obtainable via `object$data`
- `coef`: `coef(object)`
- `mcmc`: `object$sample`
- `vcov`: `vcov(object)`
- `df`: Set to `Inf` if not available in `object$df.residual`
- `subset`: `NULL` (so that all observations in `data` are used)
- `contrasts`: `NULL` (so that `getOption("contrasts")` is used)

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

## See Also

`emmobj` for an alternative way to construct an `emmGrid`.

## Examples

```
if(require(coda) && require(lme4)) {
  # Use a stored example having a posterior sample
  # Model is based on the data in lme4::cbpp

  post <- readRDS(system.file("extdata", "cbpplist", package = "emmeans"))$post.beta
  rg1 <- qdrg(~ size + period, data = lme4::cbpp, mcmc = post, link = "logit")
  summary(rg1, type = "response")
}
if (require(biglm)) {
  bigmod <- biglm(log(conc) ~ source + factor(percent), data = pigs)

  rg2 <- qdrg(object = bigmod, data = pigs)
  summary(emmeans(rg2, "source"), type = "response")
}
```

---

rbind.emmGrid

*Combine or subset emmGrid objects*


---

### Description

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

### Usage

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

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

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

### Arguments

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

### Value

A revised object of class `emmGrid`  
The result of `e1 + e2` is the same as `rbind(e1, e2)`

### Note

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

## Examples

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

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

# Do all pairwise comparisons within rows or within columns,
# all considered as one family 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
```

---

ref_grid	<i>Create a reference grid from a fitted model</i>
----------	--

---

## 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,
  transform = c("none", "response", "mu", "unlink", "log"), nesting,
  offset, sigma, ...)
```

## Arguments

object	An object produced by a supported model-fitting function, such as <code>lm</code> . Many models are supported. See <a href="#">vignette("models", "emmeans")</a> .
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 <i>not</i> specified in <code>cov.keep</code> or <code>at</code> is reduced according to these specifications. See the section below on “Using <code>cov.reduce</code> and <code>cov.keep</code> ”.
cov.keep	Character vector: names of covariates that are <i>not</i> to be reduced; these are treated as factors and used in weighting calculations. <code>cov.keep</code> 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 <code>cov.keep</code> .

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 <code>class(object)</code> 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 <code>expand.grid</code> order. The (total) number of levels must match the number of dimensions. If <code>mult.name</code> is specified, this argument is ignored.
options	If non-NULL, a named list of arguments to pass to <code>update.emmGrid</code> , just after the object is constructed.
data	A <code>data.frame</code> to use to obtain information about the predictors (e.g. factor levels). If missing, then <code>recover_data</code> is used to attempt to reconstruct the data. See the note with <code>recover_data</code> for an important precaution.
df	Numeric value. This is equivalent to specifying <code>options(df = df)</code> . See <code>update.emmGrid</code> .
type	Character value. If provided, this is saved as the "predict.type" setting. See <code>update.emmGrid</code> and the section below on prediction types and transformations.
transform	Character value. If other than "none", the reference grid is reconstructed via <code>regrid</code> with the given transform argument. See the section below on prediction types and transformations.
nesting	If the model has nested fixed effects, this may be specified here via a character vector or named list specifying the nesting structure. Specifying <code>nesting</code> 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 <code>offset = 0</code> 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 <code>sigma(object)</code> , if available, and NULL otherwise.
...	Optional arguments passed to <code>emm_basis</code> , and <code>recover_data</code> , such as <code>params</code> , <code>vcov</code> . (see <b>Covariance matrix</b> below), or options such as <code>mode</code> for specific model types (see <code>vignette("models", "emmeans")</code> ).

## 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`. 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.

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` or `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 NAs that caused one or more factor levels to be excluded when fitting the model, then `data` should also exclude those levels.)

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

**Nested factors.** Having a nesting structure affects marginal averaging in `emmeans` in that it is done separately for each level (or combination thereof) of the grouping factors. `ref_grid` tries to discern which factors are nested in other factors, but it is not always obvious, and if it misses some, the user must specify this structure via `nesting`; or later using `update.emmGrid`. The `nesting` argument may be a character vector, 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  $\text{sqrt}(y) \sim \dots$ . Even transformations containing multiplicative or additive constants, such as  $2*\text{sqrt}(y + \text{pi}) \sim \dots$ , are auto-detected. A response transformation of  $y + 1 \sim \dots$  is *not* auto-detected, but  $I(y + 1) \sim \dots$  is interpreted as  $\text{identity}(y + 1) \sim \dots$ . 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 ‘transform = "response"’. While the summary statistics for the grid itself are the same, subsequent use in [emmeans](#) will yield different results if there is a response transformation or link function. With ‘type = "response"’, EMMs are computed by averaging together predictions on the *linear-predictor* scale and then back-transforming to the response scale; while with ‘transform = "response"’, the predictions are already on the response scale so that the EMMs will be the arithmetic means of those response-scale predictions. To add further to the possibilities, *geometric* means of the response-scale predictions are obtainable via ‘transform = "log", type = "response"’. See also the help page for [regrid](#).

### Side effect

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

### 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\("models", "emmeans"\)](#).

### Examples

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

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

```
## Not run:
```

```

# We could substitute the sandwich estimator vcovHAC(fiber.lm)
# as follows:
summary(ref_grid(fiber.lm, vcov. = sandwich::vcovHAC))

## End(Not run)

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

### 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

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

```

---

regrid	<i>Reconstruct a reference grid with a new transformation or posterior sample</i>
--------	---

---

## 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 <code>emmGrid</code>
--------	---

<code>transform</code>	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 <code>bhat</code> slot contains <code>predict(object)</code> and its <code>linfct</code> slot is the identity. Any internal transformation information is preserved. If <code>transform = "pass"</code> , the object is not re-gridded in any way (this may be useful in conjunction with <code>N.sim</code> ).  If <code>transform</code> is a character value in <code>links</code> (which is the set of valid arguments for the <code>make.link</code> function, excepting "identity"), or if <code>transform</code> is a list of the same form as returned by <code>make.links</code> or <code>make.tran</code> , the results are formulated as if the response had been transformed with that link function.
<code>inv.link.lbl</code>	Character value. This applies only when <code>transform</code> is in <code>links</code> , and is used to label the predictions if subsequently summarized with <code>type = "response"</code> .
<code>predict.type</code>	Character value. If provided, the returned object is updated with the given type to use by default by <code>summary.emmGrid</code> (see <code>update.emmGrid</code> ). This may be useful if, for example, when one specifies <code>transform = "log"</code> but desires summaries to be produced by default on the response scale.
<code>bias.adjust</code>	Logical value for whether to adjust for bias in back-transforming ( <code>transform = "response"</code> ). This requires a value of <code>sigma</code> to exist in the object or be specified.
<code>sigma</code>	Error SD assumed for bias correction (when <code>transform = "response"</code> and a transformation is in effect). If not specified, <code>object@misc\$sigma</code> is used, and an error is thrown if it is not found.
<code>N.sim</code>	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 <code>sim</code> .
<code>sim</code>	A function of three arguments (no names are assumed). If <code>N.sim</code> is supplied with a frequentist model, this function is called with respective arguments <code>N.sim</code> , <code>object@bhat</code> , and <code>object@V</code> . The default is the multivariate normal distribution.
<code>...</code>	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 convert a reference grid for a frequentist model to one for a Bayesian model. To do so, specify a value for `N.sim` and a posterior 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 posterior 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 `transform` argument to `ref_grid` (either directly or indirectly via `emmeans`). This is often a simpler approach if the reference grid has not already been constructed.

**Examples**

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

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

**Description**

Miscellaneous methods for emmGrid objects

**Usage**

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

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

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

**Arguments**

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

**Value**

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

---

summary.emmGrid	<i>Summaries, predictions, intervals, and tests for emmGrid objects</i>
-----------------	---

---

**Description**

These are the primary methods for obtaining numerical or tabular results from an `emmGrid` object. 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, type, df, null,
  delta, side, frequentist,
  bias.adjust = get_emm_option("back.bias.adj"), sigma, ...)

## 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 = FALSE,
  ...)

## S3 method for class 'summary_emm'
```

```
x[... , as.df = TRUE]

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

test(object, null, ...)

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

### Arguments

object	An object of class "emmGrid" (see <a href="#">emmGrid-class</a> )
infer	A vector of one or two logical values. The first determines whether confidence intervals are displayed, and the second determines whether <i>t</i> tests and <i>P</i> 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 <code>infer[1]</code> is TRUE.
adjust	Character value naming the method used to adjust <i>p</i> 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 <i>P</i> values.
type	Character: type of prediction desired. This only has an effect if there is a known transformation or link function. "response" specifies that the inverse transformation be applied. "mu" (or equivalently, "unlink") is usually the same as "response", but in the case where the model has both a link function and a response transformation, only the link part is back-transformed. Other valid values are "link", "lp", and "linear.predictor"; these are equivalent, and request that results be shown for the linear predictor, with no back-transformation. The default is "link", unless the "predict.type" option is in force; see <a href="#">emm_options</a> , 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 <i>P</i> values (NA specifies asymptotic results).
null	Numeric. Null hypothesis value(s), on the linear-predictor scale, against which estimates are tested. May be a single value used for all, or a numeric vector of length equal to the number of tests in each family (i.e., by group in the displayed table).
delta	Numeric value (on the linear-predictor scale). If zero, ordinary tests of significance are performed. If positive, this specifies a threshold for testing equivalence (using the TOST or two-one-sided-test method), non-inferiority, or non-superiority, depending on side. See Details for how the test statistics are defined.

side	Numeric or character value specifying whether the test is left-tailed (-1, "-", code"<", "left", or "nonsuperiority"); right-tailed (1, "+", ">", "right", or "noninferiority"); or two-sided (0, 2, "!=", "two-sided", "both", "equivalence", or "="). See the special section below for more details.
frequentist	Ignored except if a Bayesian model was fitted. If missing or FALSE, the object is passed to <a href="#">hpd.summary</a> . 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 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 an error is thrown if it is not found. <i>Note:</i> sigma may be a vector, as long as it conforms to the number of rows of the reference grid.
...	(Not used by summary.emmGrid.) In as.data.frame.emmGrid, confint.emmGrid, predict.emmGrid, and test.emmGrid, these arguments are passed to summary.emmGrid.
interval	Type of interval desired (partial matching is allowed): "none" for no intervals, otherwise confidence or prediction intervals with given arguments, via <a href="#">confint.emmGrid</a> .
x	object of the given class
row.names	passed to <a href="#">as.data.frame</a>
optional	passed to <a href="#">as.data.frame</a>
as.df	Logical value. With x[... , as.df = TRUE], the result is object is coerced to an ordinary <a href="#">data.frame</a> ; otherwise, it is left as a summary_emm object.
parm	(Required argument for confint methods, but not used)
joint	Logical value. If FALSE, the arguments are passed to <a href="#">summary.emmGrid</a> with infer=c(FALSE, TRUE). If joint = TRUE, a joint test of the hypothesis $L\beta = \text{null}$ is performed, where L is object@linfct and beta is the vector of fixed effects estimated by object@betahat. This will be either an F test or a chi-square (Wald) test depending on whether degrees of freedom are available. See also <a href="#">joint_tests</a> .
verbose	Logical value. If TRUE and joint = TRUE, a table of the effects being tested is printed.
rows	Integer values. The rows of L to be tested in the joint test. If missing, all rows of L are used. If not missing, by variables are ignored.
status	logical. If TRUE, a note column showing status flags (for rank deficiencies and estimability issues) is displayed even when empty. If FALSE, the column is included only if there are such issues.

## Details

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

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

## Value

`summary.emmGrid`, `confint.emmGrid`, and `test.emmGrid` return an object of class `"summary_emm"`, which is an extension of `data.frame` but with a special `print` method that displays it with custom formatting. For models fitted using MCMC methods, the 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 a plain data frame, equivalent to `as.data.frame(summary(.))`.

## Defaults

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

## Transformations and links

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

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.

## P-value adjustments

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

The valid values of `adjust` are as follows:

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

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

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

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

"dunnetttx" 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", "dunnetttx", or others more attractive.

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

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

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

### Testing nonsuperiority, noninferiority, or equivalence

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

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

**Left-sided (nonsuperiority)**  $H_0 : \theta \geq \theta_0 + \delta$  versus  $H_1 : \theta < \theta_0 + \delta$

$$t = (Q - \theta_0 - \delta)/SE$$

The  $p$  value is the lower-tail probability.

**Right-sided (noninferiority)**  $H_0 : \theta \leq \theta_0 - \delta$  versus  $H_1 : \theta > \theta_0 - \delta$

$$t = (Q - \theta_0 + \delta)/SE$$

The  $p$  value is the upper-tail probability.

**Two-sided (equivalence)**  $H_0 : |\theta - \theta_0| \geq \delta$  versus  $H_1 : |\theta - \theta_0| < \delta$

$$t = (|Q - \theta_0| - \delta)/SE$$

The  $p$  value is the lower-tail probability.

### Non-estimable cases

When the model is rank-deficient, each row  $x$  of object's `linfct` slot is checked for estimability. If `sum(x*bhat)` is found to be non-estimable, then the string `NonEst` is displayed for the estimate,

and associated statistics are set to NA. The estimability check is performed using the orthonormal basis  $N$  in the `nbasis` slot for the null space of the rows of the model matrix. Estimability fails when  $\|Nx\|^2/\|x\|^2$  exceeds `tol`, which by default is  $1e-8$ . You may change it via `emm_options` by setting `estble.tol` to the desired value.

### Warning about potential misuse of P values

A growing consensus in the statistical and scientific community is 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 are just too misleading and 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 (usually) is not a natural way to back-transform it.

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

### See Also

`link{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")

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

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

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

---

update.emmGrid	<i>Update an emmGrid object</i>
----------------	---------------------------------

---

### 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)
```

### Arguments

<code>object</code>	An <code>emmGrid</code> object
<code>...</code>	Options to be set. These must match a list of known options (see Details)
<code>silent</code>	Logical value. If <code>FALSE</code> (the default), a message is displayed if any options are not matched. If <code>TRUE</code> , no messages are shown.

### Value

an updated `emmGrid` object.

### 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`.
- `estType` (character) is the type of the estimate. It should match one of `'c("prediction", "contrast", "pairs")'`. This is used along with `"adjust"` to determine appropriate adjustments to P values and confidence intervals.
- `famSize` (integer) is the 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.
- `null` (numeric) null hypothesis for `summary` or `test` (taken to be zero if missing).
- `side` (numeric or character) side specification 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"`.
- `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`, `avgd.over`, and `nesting`. *Note:* All six letters of `levels` is needed in order to distinguish it from `level`.

**(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.

## See Also

[emm\\_options](#)

## Examples

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

# Reference grid that knows about the transformation:
mypigs.rg <- update(ref_grid(mypigs.lm), tran = "log",
                   predict.type = "response")
emmmeans(mypigs.rg, "source")
```

---

xtable.emmGrid

*Using xtable for EMMs*

---

## Description

These methods provide support for the **xtable** package, enabling polished presentations of tabular output from [emmmeans](#) 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 <code>xtableList</code>
label	Passed to <code>xtableList</code>
align	Passed to <code>xtableList</code>
digits	Passed to <code>xtableList</code>
display	Passed to <code>xtableList</code>
auto	Passed to <code>xtableList</code>
...	Arguments passed to <code>summary.emmGrid</code>
type	Passed to <code>print.xtable</code>
include.rownames	Passed to <code>print.xtable</code>
sanitize.message.function	Passed to <code>print.xtable</code>

**Details**

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

**Value**

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

**Examples**

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

# Index

## \*Topic **datasets**

- auto.noise, 8
- emm\_options, 29
- feedlot, 36
- fiber, 37
- MOats, 44
- neuralgia, 46
- nutrition, 47
- oranges, 48
- pigs, 49
- + .emmGrid (rbind.emmGrid), 55
- .emm\_basis (extending-emmeans), 33
- .emm\_register (extending-emmeans), 33
- .recover\_data (extending-emmeans), 33
- [, 55
- [.emmGrid, 24
- [.emmGrid (rbind.emmGrid), 55
- [.summary\_emm (summary.emmGrid), 64
  
- add\_grouping, 4
- as.data.frame, 66
- as.data.frame.emmGrid  
(summary.emmGrid), 64
- as.emmGrid, 5
- as.glht, 4, 68
- as.glht (emm), 19
- as.glht.emmGrid, 24
- as.list.emmGrid (as.emmGrid), 5
- as.mcmc.emmGrid, 6, 24
- as.mcmc.list.emmGrid, 24
- as.mcmc.list.emmGrid (as.mcmc.emmGrid),  
6
- auto.noise, 8
  
- CLD, 9
- cld, 10
- CLD.emm\_list (emm\_list), 28
- CLD.emmGrid, 21
- cld.emmGrid, 24
- cld.emmGrid (CLD), 9
  
- coef.emmGrid, 24
- coef.emmGrid (contrast), 11
- confint.emmGrid, 4, 21, 24, 29, 66
- confint.emmGrid (summary.emmGrid), 64
- consec.emmc (contrast-methods), 14
- contrast, 4, 10, 11, 17, 18, 21, 23, 39, 51, 71
- contrast-methods, 14
- contrast.emmGrid, 21, 24, 29
  
- data.frame, 34, 66, 67
- del.eff.emmc (contrast-methods), 14
- delete.response, 34
- dotplot, 50, 51
- dunnett.emmc (contrast-methods), 14
  
- eff.emmc (contrast-methods), 14
- eff\_size, 17
- emm, 4, 19, 42
- emm\_basis, 57
- emm\_basis (extending-emmeans), 33
- emm\_defaults (emm\_options), 29
- emm\_list, 12, 13, 21, 28
- emm\_options, 25, 29, 39, 42, 58, 59, 65, 69, 72
- emmc-functions, 11
- emmc-functions (contrast-methods), 14
- emmeans, 3, 19, 20, 23, 25, 26, 28, 29, 31, 32,  
41–43, 57, 59, 60, 63, 69, 72
- emmeans-package, 3
- emmGrid, 13, 17, 18, 28, 60
- emmGrid-class, 23, 65
- emmip, 4, 25, 29, 42, 70, 71
- emmobj, 6, 27, 27, 42, 54
- emtrends, 3, 21, 29, 31, 42, 60
- expand.grid, 24, 27, 57
- extending-emmeans, 33, 59
  
- feedlot, 36
- fiber, 37
  
- get, 35

- get.lsm.option (lsmeans), 41
- get\_emm\_option, 42
- get\_emm\_option (emm\_options), 29
- ggplot, 25
- glht-support (emm), 19
- glht.emmGrid (emm), 19
- glht.emmlf (emm), 19
- glmer.nb, 59
  
- hpd.summary, 38, 64, 66, 67
  
- identity.emmc (contrast-methods), 14
- interaction.plot, 26
  
- joint\_tests, 39, 66
  
- lm, 17, 58
- lsm (lsmeans), 41
- lsmeans, 41
- lsmip (lsmeans), 41
- lsmobj (lsmeans), 41
- lstrends (lsmeans), 41
  
- make.link, 42, 44, 62, 70
- make.tran, 42, 60, 62, 70
- mcmc, 7
- mcmc-support (as.mcmc.emmGrid), 6
- mcmc.list, 7
- mean\_chg.emmc (contrast-methods), 14
- MOats, 44
- modelparm.emmwrap (emm), 19
- models, 45
  
- neuralgia, 46
- nonest.basis, 35
- nutrition, 47
  
- Oats, 45
- offset, 24
- oranges, 48
  
- p.adjust, 16, 68
- pairs.emmGrid, 21, 24, 29
- pairs.emmGrid (contrast), 11
- pairwise.emmc (contrast-methods), 14
- pigs, 49
- plot.emmGrid, 4, 24, 29, 49, 71
- plot.summary\_emm (plot.emmGrid), 49
- pmm (lsmeans), 41
- pmmeans (lsmeans), 41
  
- pmmip (lsmeans), 41
- pmmobj (lsmeans), 41
- pmtrends (lsmeans), 41
- poly, 15
- poly.emmc (contrast-methods), 14
- predict, 58
- predict.emmGrid, 24, 25, 29, 50, 70, 71
- predict.emmGrid (summary.emmGrid), 64
- print.emmGrid, 24
- print.emmGrid (str.emmGrid), 63
- print.xtable, 73
- print.xtable\_emm (xtable.emmGrid), 72
- print.xtableList, 73
- pwpp, 10, 51
  
- qdr, 28, 53, 54
  
- rbind, 55
- rbind.emmGrid, 13, 24, 55, 67
- recover\_data, 57
- recover\_data (extending-emmeans), 33
- ref\_grid, 3, 21–23, 29, 32, 33, 43, 44, 53, 56, 63, 71
- regrid, 18, 57, 60, 61
- repairwise.emmc (contrast-methods), 14
  
- set.seed, 68
- str.emmGrid, 24, 63, 71
- summary.emmGrid, 4, 21, 24, 29, 35, 38–40, 50, 52, 60, 62, 64, 66, 70, 71, 73
  
- terms, 34
- test, 39, 40
- test (summary.emmGrid), 64
- test.emmGrid, 4, 21, 24, 29
- trt.vs.ctrl.emmc (contrast-methods), 14
- trt.vs.ctrl1.emmc (contrast-methods), 14
- trt.vs.ctrlk.emmc (contrast-methods), 14
- tukey.emmc (contrast-methods), 14
  
- update, 67
- update.emmGrid, 5, 12, 21, 24, 27, 30, 42, 43, 50, 51, 55, 57, 59, 62, 70
  
- vcov, 59
- vcov.emmGrid, 24
- vcov.emmGrid (str.emmGrid), 63
  
- wrappers, 22
- wrappers (lsmeans), 41

`xtable.emmGrid`, [24](#), [72](#)  
`xtable.summary_emm(xtable.emmGrid)`, [72](#)  
`xtableList`, [73](#)  
`xyplot`, [25](#)