Package 'marginaleffects'

March 10, 2023

Title Predictions, Comparisons, Slopes, Marginal Means, and Hypothesis Tests

Version 0.11.0

Description Compute and plot predictions, slopes, marginal means, and comparisons (contrasts, risk ratios, odds, etc.) for over 75 classes of statistical models in R. Conduct linear and non-linear hypothesis tests, or equivalence tests. Calculate uncertainty estimates using the delta method, bootstrapping, or simulation-based inference.

License GPL (>= 3)

Copyright inst/COPYRIGHTS

Encoding UTF-8

URL https://vincentarelbundock.github.io/marginaleffects/

BugReports https://github.com/vincentarelbundock/marginaleffects/issues

RoxygenNote 7.2.3

Depends R (>= 3.6.0)

Imports checkmate, data.table, generics, insight (>= 0.19.0), methods, Rcpp (>= 1.0.10)

LinkingTo Rcpp, RcppEigen

Suggests AER, Amelia, afex, aod, bench, betareg, BH, bife, biglm, blme, boot, brglm2, brms, brmsmargins, broom, collapse, conflicted, covr, crch, distributional, dplyr, emmeans, equivalence, estimatr, fixest, future, fwb, gam, gamlss, gamlss.dist, geepack, ggdag, ggdist, ggokabeito, ggplot2, ggrepel, glmmTMB, glmx, haven, here, itsadug, ivreg, kableExtra, knitr, lme4, lmerTest, magrittr, margins, MatchIt, MASS, mclogit, MCMCglmm, missRanger, mgcv, mhurdle, mice, miceadds, mlogit, modelbased, modelsummary, nlme, nnet, numDeriv, optmatch, ordinal, parameters, patchwork, pkgdown, plm, polspline, poorman, posterior, prediction, pscl, purrr, quantreg, Rchoice, remotes, rmarkdown, rms, robust, robustbase, robustlmm, rsample, rstanarm, rstantools, rsvg, sampleSelection, sandwich, scam, speedglm, spelling, survey, survival, svglite, systemfonts, tibble, tidymodels, tidyr, tidyverse, tinytest, titanic, truncreg, tsModel, withr

Collate 'RcppExports.R' 'backtransform.R' 'bootstrap_boot.R' 'bootstrap_fwb.R' 'bootstrap_rsample.R' 'broom.R' 'by.R' 'ci.R' 'comparisons.R' 'complete_levels.R' 'datagrid.R' 'equivalence.R' 'get_averages.R' 'get_coef.R' 'get_contrast_data.R' 'get_contrast_data_character.R' 'get_contrast_data_factor.R' 'get_contrast_data_logical.R' 'get_contrast_data_numeric.R' 'get_contrasts.R' 'get_group_names.R' 'get_hypothesis.R' 'get_jacobian.R' 'get_model_matrix.R' 'get_model_matrix_attribute.R' 'get_modeldata.R' 'get_predict.R' 'get_se_delta.R' 'get_vcov.R' 'github_issue.R' 'hush.R' 'hypotheses.R' 'imputation.R' 'inferences.R' 'marginal_means.R' 'mean_or_mode.R' 'set_coef.R' 'methods_MASS.R' 'methods_MCMCglmm.R' 'methods_Rchoice.R' 'methods_afex.R' 'methods_aod.R' 'methods_betareg.R' 'methods_bife.R' 'methods_biglm.R' 'methods_nnet.R' 'methods_brglm2.R' 'sanity_model.R' 'methods_brms.R' 'methods_crch.R' 'methods_fixest.R' 'methods_gamlss.R' 'methods_glmmTMB.R' 'methods_glmx.R' 'methods_inferences_simulation.R' 'methods_lme4.R' 'methods_mclogit.R' 'methods_mgcv.R' 'methods_mhurdle.R' 'methods_mlm.R' 'methods_mlogit.R' 'methods_ordinal.R' 'methods_plm.R' 'methods_pscl.R' 'methods_quantreg.R' 'methods_rms.R' 'methods_robustlmm.R' 'methods_rstanarm.R' 'methods_sampleSelection.R' 'methods_scam.R' 'methods_stats.R' 'methods_survival.R' 'methods_tobit1.R' 'modelarchive.R' 'myTryCatch.R' 'package.R' 'plot.R' 'plot_build.R' 'plot_comparisons.R' 'plot_predictions.R' 'plot_slopes.R' 'posterior_draws.R' 'predictions.R' 'print.R' 'recall.R' 'sanitize_comparison.R' 'sanitize_condition.R' 'sanitize_conf_level.R' 'sanitize_hypothesis.R' 'sanitize_interaction.R' 'sanitize_newdata.R' 'sanitize_type.R' 'sanitize_variables.R' 'sanitize_vcov.R' 'sanity.R' 'sanity_by.R' 'sanity_dots.R' 'settings.R' 'slopes.R' 'summary.R' 'tinytest.R' 'type_dictionary.R' 'unpack_matrix_cols.R' 'utils.R'

Language en-US

NeedsCompilation yes

Author Vincent Arel-Bundock [aut, cre, cph] (<https://orcid.org/0000-0003-2042-7063>), Marcio Augusto Diniz [ctb] (<https://orcid.org/0000-0002-2427-7843>), Noah Greifer [ctb] (<https://orcid.org/0000-0003-3067-7154>), Etienne Bacher [ctb] (<https://orcid.org/0000-0002-9271-5075>)

Maintainer Vincent Arel-Bundock <vincent.arel-bundock@umontreal.ca>

Repository CRAN

Date/Publication 2023-03-10 10:10:02 UTC

R topics documented:

`comparisons` *Comparisons Between Predictions Made With Different Regressor Values*

Description

Predict the outcome variable at different regressor values (e.g., college graduates vs. others), and compare those predictions by computing a difference, ratio, or some other function. `comparisons()` can return many quantities of interest, such as contrasts, differences, risk ratios, changes in log odds, slopes, elasticities, etc.

- `comparisons()`: unit-level (conditional) estimates.
- `avg_comparisons()`: average (marginal) estimates.

`variables` identifies the focal regressors whose "effect" we are interested in. `comparison` determines how predictions with different regressor values are compared (difference, ratio, odds, etc.). The `newdata` argument and the `datagrid()` function control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the comparisons vignette and package website for worked examples and case studies:

- https://vincentarelbundock.github.io/marginaleffects/articles/comparisons.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
comparisons(
  model,
  newdata = NULL,
  variables = NULL,
  comparison = "difference",
  type = NULL,
```

```
  vcov = TRUE,
  by = FALSE,
  conf_level = 0.95,
  transform = NULL,
  cross = FALSE,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  eps = NULL,
  ...
)

avg_comparisons(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  vcov = TRUE,
  by = TRUE,
  conf_level = 0.95,
  comparison = "difference",
  transform = NULL,
  cross = FALSE,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  eps = NULL,
  ...
)
```

Arguments

`model` Model object

`newdata` Grid of predictor values at which we evaluate the comparisons.

- `NULL` (default): Unit-level contrasts for each observed value in the original dataset (empirical distribution).
- data frame: Unit-level contrasts for each row of the `newdata` data frame.
- string:
  - "mean": Contrasts at the Mean. Contrasts when each predictor is held at its mean or mode.
  - "median": Contrasts at the Median. Contrasts when each predictor is held at its median or mode.
  - "marginalmeans": Contrasts at Marginal Means.

  - "tukey": Contrasts at Tukey's 5 numbers.
  - "grid": Contrasts on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).
- `datagrid()` call to specify a custom grid of regressors. For example:
  - `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes.
  - `newdata = datagrid(mpg = fivenum)`: `mpg` variable held at Tukey's five numbers (using the `fivenum` function), and other regressors fixed at their means or modes.
  - See the Examples section and the datagrid documentation.

`variables` Focal variables

- `NULL`: compute comparisons for all the variables in the model object (can be slow).
- Character vector: subset of variables (usually faster).
- Named list: names identify the subset of variables of interest, and values define the type of contrast to compute. Acceptable values depend on the variable type:
  - Factor or character variables:
    - "reference": Each factor level is compared to the factor reference (base) level
    - "all": All combinations of observed levels
    - "sequential": Each factor level is compared to the previous factor level
    - "pairwise": Each factor level is compared to all other levels
    - "minmax": The highest and lowest levels of a factor.
    - Vector of length 2 with the two values to compare.
  - Logical variables:
    - NULL: contrast between TRUE and FALSE
  - Numeric variables:
    - Numeric of length 1: Contrast for a gap of `x`, computed at the observed value plus and minus `x / 2`. For example, estimating a `+1` contrast compares adjusted predictions when the regressor is equal to its observed value minus 0.5 and its observed value plus 0.5.
    - Numeric vector of length 2: Contrast between the 2nd element and the 1st element of the `x` vector.
    - Function which accepts a numeric vector and returns a data frame with two columns of "low" and "high" values to compare. See examples below.
    - "iqr": Contrast across the interquartile range of the regressor.
    - "sd": Contrast across one standard deviation around the regressor mean.
    - "2sd": Contrast across two standard deviations around the regressor mean.
    - "minmax": Contrast between the maximum and the minimum values of the regressor.

- Examples:
  - `variables = list(gear = "pairwise", hp = 10)`
  - `variables = list(gear = "sequential", hp = c(100, 120))`
  - See the Examples section below for more.

`comparison` How should pairs of predictions be compared? Difference, ratio, odds ratio, or user-defined functions.

- string: shortcuts to common contrast functions.
  - Supported shortcuts strings: difference, differenceavg, differenceavgwts, dydx, eyex, eydx, dyex, dydxavg, eyexavg, eydxavg, dyexavg, dydxavgwts, eyexavgwts, eydxavgwts, dyexavgwts, ratio, ratioavg, ratioavgwts, lnratio, lnratioavg, lnratioavgwts, lnor, lnoravg, lnoravgwts, expdydx, expdydxavg, expdydxavgwts
  - See the Comparisons section below for definitions of each transformation.
- function: accept two equal-length numeric vectors of adjusted predictions (`hi` and `lo`) and returns a vector of contrasts of the same length, or a unique numeric value.
  - See the Transformations section below for examples of valid functions.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.

- TRUE: aggregate estimates for each term.
- Character vector of column names in newdata or in the data frame produced by calling the function without the by argument.
- Data frame with a by column of group labels, and merging columns shared by newdata or the data frame produced by calling the same function without the by argument.
- See examples below.

conf_level
: numeric value between 0 and 1. Confidence level to use to build a confidence interval.

transform
: string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

cross
: - FALSE: Contrasts represent the change in adjusted predictions when one predictor changes and all other variables are held constant.
  - TRUE: Contrasts represent the changes in adjusted predictions when all the predictors specified in the variables argument are manipulated simultaneously (a "cross-contrast").

wts
: string or numeric: weights to use when computing average contrasts or slopes. These weights only affect the averaging in avg_*() or with the by argument, and not the unit-level estimates themselves.
  - string: column name of the weights variable in newdata. When supplying a column name to wts, it is recommended to supply the original data (including the weights variable) explicitly to newdata.
  - numeric: vector of length equal to the number of rows in the original data or in newdata (if supplied).

hypothesis
: specify a hypothesis test or custom contrast using a numeric value, vector, or matrix, a string, or a string formula.
  - Numeric:
    - Single value: the null hypothesis used in the computation of Z and p (before applying transform).
    - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the hypothesis argument.
    - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
  - String formula to specify linear or non-linear hypothesis tests. If the term column uniquely identifies rows, terms can be used in the formula. Otherwise, use b1, b2, etc. to identify the position of each parameter. Examples:
    - hp = drat
    - hp + drat = 12
    - b1 + b2 + b3 = 0
  - String:
    - "pairwise": pairwise differences between estimates in each row.

- "reference": differences between the estimates in each row and the estimate in the first row.
- "sequential": difference between an estimate and the estimate in the next row.
- "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.

- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffec

equivalence
: Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. See Details section below.

p_adjust
: Adjust p-values for multiple comparisons: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", or "fdr". See stats::p.adjust

df
: Degrees of freedom used to compute p values and confidence intervals. A single numeric value between 1 and `Inf`. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used. See insight::get_df for a convenient function to extract degrees of freedom. Ex: `slopes(model, df = insight::get_df(model))`

eps
: NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

...
: Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `data.frame` with one row per observation (per term/group) and several columns:

- `rowid`: row number of the `newdata` data frame
- `type`: prediction type, as defined by the `type` argument
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the variable whose marginal effect is computed
- `dydx`: slope of the outcome with respect to the term, for a given combination of predictor values
- `std.error`: standard errors computed by via the delta method.

See `?print.marginaleffects` for printing options.

Functions

- `avg_comparisons()`: Average comparisons

Standard errors using the delta method

Standard errors for all quantities estimated by `marginaleffects` can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to `1e-8`, or to `1e-4` times the smallest absolute model coefficient, whichever is largest.

`marginaleffects` can delegate numeric differentiation to the `numDeriv` package, which allows more flexibility. To do this, users can pass arguments to the `numDeriv::jacobian` function through a global option. For example:

- `options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))`
- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Standard Errors and Confidence Intervals" vignette on the `marginaleffects` website for more details on the computation of standard errors:

https://vincentarelbundock.github.io/marginaleffects/articles/uncertainty.html

Note that the `inferences()` function can be used to compute uncertainty estimates using a bootstrap or simulation-based inference. See the vignette:

https://vincentarelbundock.github.io/marginaleffects/articles/bootstrap.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

https://github.com/vincentarelbundock/marginaleffects/issues

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	brms::posterior_predict
`lme4`	`merMod`	`re.form`	lme4::predict.merMod
		`allow.new.levels`	lme4::predict.merMod
`glmmTMB`	`glmmTMB`	`re.form`	glmmTMB::predict.glmmTMB
		`allow.new.levels`	glmmTMB::predict.glmmTMB
		`zitype`	glmmTMB::predict.glmmTMB
`mgcv`	`bam`	`exclude`	mgcv::predict.bam
`robustlmm`	`rlmerMod`	`re.form`	robustlmm::predict.rlmerMod
		`allow.new.levels`	robustlmm::predict.rlmerMod
`MCMCglmm`	`MCMCglmm`	`ndraws`	

comparison argument functions

The following transformations can be applied by supplying one of the shortcut strings to the `comparison` argument. `hi` is a vector of adjusted predictions for the "high" side of the contrast. `lo` is a vector of adjusted predictions for the "low" side of the contrast. `y` is a vector of adjusted predictions for the original data. `x` is the predictor in the original data. `eps` is the step size to use to compute derivatives and elasticities.

Shortcut	Function
difference	\(hi, lo) hi - lo
differenceavg	\(hi, lo) mean(hi) - mean(lo)
differenceavgwts	\(hi, lo, w) wmean(hi, w) - wmean(lo, w)
dydx	\(hi, lo, eps) (hi - lo)/eps
eyex	\(hi, lo, eps, y, x) (hi - lo)/eps * (x/y)
eydx	\(hi, lo, eps, y, x) ((hi - lo)/eps)/y
dyex	\(hi, lo, eps, x) ((hi - lo)/eps) * x
dydxavg	\(hi, lo, eps) mean((hi - lo)/eps)
eyexavg	\(hi, lo, eps, y, x) mean((hi - lo)/eps * (x/y))
eydxavg	\(hi, lo, eps, y, x) mean(((hi - lo)/eps)/y)
dyexavg	\(hi, lo, eps, x) mean(((hi - lo)/eps) * x)
dydxavgwts	\(hi, lo, eps, w) wmean((hi - lo)/eps, w)
eyexavgwts	\(hi, lo, eps, y, x, w) wmean((hi - lo)/eps * (x/y), w)
eydxavgwts	\(hi, lo, eps, y, x, w) wmean(((hi - lo)/eps)/y, w)
dyexavgwts	\(hi, lo, eps, x, w) wmean(((hi - lo)/eps) * x, w)
ratio	\(hi, lo) hi/lo
ratioavg	\(hi, lo) mean(hi)/mean(lo)
ratioavgwts	\(hi, lo) wmean(hi)/wmean(lo)
lnratio	\(hi, lo) log(hi/lo)
lnratioavg	\(hi, lo) log(mean(hi)/mean(lo))
lnratioavgwts	\(hi, lo) log(wmean(hi)/wmean(lo))
lnor	\(hi, lo) log((hi/(1 - hi))/(lo/(1 - lo)))
lnoravg	\(hi, lo) log((mean(hi)/(1 - mean(hi)))/(mean(lo)/(1 - mean(lo))))
lnoravgwts	\(hi, lo, w) log((wmean(hi, w)/(1 - wmean(hi, w)))/(wmean(lo, w)/(1 - wmean(lo, w))))
expdydx	\(hi, lo, eps) ((exp(hi) - exp(lo))/exp(eps))/eps
expdydxavg	\(hi, lo, eps) mean(((exp(hi) - exp(lo))/exp(eps))/eps)
expdydxavgwts	\(hi, lo, eps, w) wmean(((exp(hi) - exp(lo))/exp(eps))/eps, w)

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

`options("marginaleffects_posterior_interval" = "eti")`

`options("marginaleffects_posterior_interval" = "hdi")`

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

`options("marginaleffects_posterior_center" = "mean")`

`options("marginaleffects_posterior_center" = "median")`

When estimates are averaged using the `by` argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Examples

```
library(marginaleffects)

# Linear model
tmp <- mtcars
tmp$am <- as.logical(tmp$am)
mod <- lm(mpg ~ am + factor(cyl), tmp)
avg_comparisons(mod, variables = list(cyl = "reference"))
avg_comparisons(mod, variables = list(cyl = "sequential"))
avg_comparisons(mod, variables = list(cyl = "pairwise"))

# GLM with different scale types
mod <- glm(am ~ factor(gear), data = mtcars)
avg_comparisons(mod, type = "response")
avg_comparisons(mod, type = "link")

# Contrasts at the mean
```

```
comparisons(mod, newdata = "mean")

# Contrasts between marginal means
comparisons(mod, newdata = "marginalmeans")

# Contrasts at user-specified values
comparisons(mod, newdata = datagrid(am = 0, gear = tmp$gear))
comparisons(mod, newdata = datagrid(am = unique, gear = max))

m <- lm(mpg ~ hp + drat + factor(cyl) + factor(am), data = mtcars)
comparisons(m, variables = "hp", newdata = datagrid(FUN_factor = unique, FUN_numeric = median))

# Numeric contrasts
mod <- lm(mpg ~ hp, data = mtcars)
avg_comparisons(mod, variables = list(hp = 1))
avg_comparisons(mod, variables = list(hp = 5))
avg_comparisons(mod, variables = list(hp = c(90, 100)))
avg_comparisons(mod, variables = list(hp = "iqr"))
avg_comparisons(mod, variables = list(hp = "sd"))
avg_comparisons(mod, variables = list(hp = "minmax"))

# using a function to specify a custom difference in one regressor
dat <- mtcars
dat$new_hp <- 49 * (dat$hp - min(dat$hp)) / (max(dat$hp) - min(dat$hp)) + 1
modlog <- lm(mpg ~ log(new_hp) + factor(cyl), data = dat)
fdiff <- \(x) data.frame(x, x + 10)
avg_comparisons(modlog, variables = list(new_hp = fdiff))

# Adjusted Risk Ratio: see the contrasts vignette
mod <- glm(vs ~ mpg, data = mtcars, family = binomial)
avg_comparisons(mod, comparison = "lnratioavg", transform = exp)

# Adjusted Risk Ratio: Manual specification of the `comparison`
avg_comparisons(
     mod,
     comparison = function(hi, lo) log(mean(hi) / mean(lo)),
     transform = exp)
# cross contrasts
mod <- lm(mpg ~ factor(cyl) * factor(gear) + hp, data = mtcars)
avg_comparisons(mod, variables = c("cyl", "gear"), cross = TRUE)

# variable-specific contrasts
avg_comparisons(mod, variables = list(gear = "sequential", hp = 10))

# hypothesis test: is the `hp` marginal effect at the mean equal to the `drat` marginal effect
mod <- lm(mpg ~ wt + drat, data = mtcars)

comparisons(
    mod,
    newdata = "mean",
    hypothesis = "wt = drat")

# same hypothesis test using row indices
```

```
comparisons(
    mod,
    newdata = "mean",
    hypothesis = "b1 - b2 = 0")

# same hypothesis test using numeric vector of weights
comparisons(
    mod,
    newdata = "mean",
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
comparisons(
    mod,
    newdata = "mean",
    hypothesis = lc)


# `by` argument
mod <- lm(mpg ~ hp * am * vs, data = mtcars)
comparisons(mod, by = TRUE)

mod <- lm(mpg ~ hp * am * vs, data = mtcars)
avg_comparisons(mod, variables = "hp", by = c("vs", "am"))

library(nnet)
mod <- multinom(factor(gear) ~ mpg + am * vs, data = mtcars, trace = FALSE)
by <- data.frame(
    group = c("3", "4", "5"),
    by = c("3,4", "3,4", "5"))
comparisons(mod, type = "probs", by = by)
```

`datagrid` *Data grids*

Description

Generate a data grid of user-specified values for use in the `newdata` argument of the `predictions()`, `comparisons()`, and `slopes()` functions. This is useful to define where in the predictor space we want to evaluate the quantities of interest. Ex: the predicted outcome or slope for a 37 year old college graduate.

- `datagrid()` generates data frames with combinations of "typical" or user-supplied predictor values.
- `datagridcf()` generates "counter-factual" data frames, by replicating the entire dataset once for every combination of predictor values supplied by the user.

Usage

```
datagrid(
  ...,
  model = NULL,
  newdata = NULL,
  grid_type = "typical",
  FUN_character = get_mode,
  FUN_factor = get_mode,
  FUN_logical = get_mode,
  FUN_numeric = function(x) mean(x, na.rm = TRUE),
  FUN_integer = function(x) round(mean(x, na.rm = TRUE)),
  FUN_other = function(x) mean(x, na.rm = TRUE)
)

datagridcf(..., model = NULL, newdata = NULL)
```

Arguments

`...` named arguments with vectors of values or functions for user-specified variables.

- Functions are applied to the variable in the `model` dataset or `newdata`, and must return a vector of the appropriate type.
- Character vectors are automatically transformed to factors if necessary. +The output will include all combinations of these variables (see Examples below.)

`model` Model object

`newdata` data.frame (one and only one of the `model` and `newdata` arguments

`grid_type` character

- "typical": variables whose values are not explicitly specified by the user in `...` are set to their mean or mode, or to the output of the functions supplied to `FUN_type` arguments.
- "counterfactual": the entire dataset is duplicated for each combination of the variable values specified in `...`. Variables not explicitly supplied to `datagrid()` are set to their observed values in the original dataset.

`FUN_character` the function to be applied to character variables.

`FUN_factor` the function to be applied to factor variables.

`FUN_logical` the function to be applied to factor variables.

`FUN_numeric` the function to be applied to numeric variables.

`FUN_integer` the function to be applied to integer variables.

`FUN_other` the function to be applied to other variable types.

Details

If `datagrid` is used in a `predictions()`, `comparisons()`, or `slopes()` call as the `newdata` argument, the model is automatically inserted in the `model` argument of `datagrid()` call, and users do not need to specify either the `model` or `newdata` arguments.

If users supply a model, the data used to fit that model is retrieved using the `insight::get_data` function.

Value

A `data.frame` in which each row corresponds to one combination of the named predictors supplied by the user via the `...` dots. Variables which are not explicitly defined are held at their mean or mode.

Functions

- `datagridcf()`: Counterfactual data grid

Examples

```
# The output only has 2 rows, and all the variables except `hp` are at their
# mean or mode.
datagrid(newdata = mtcars, hp = c(100, 110))

# We get the same result by feeding a model instead of a data.frame
mod <- lm(mpg ~ hp, mtcars)
datagrid(model = mod, hp = c(100, 110))

# Use in `marginaleffects` to compute "Typical Marginal Effects". When used
# in `slopes()` or `predictions()` we do not need to specify the
#`model` or `newdata` arguments.
slopes(mod, newdata = datagrid(hp = c(100, 110)))

# datagrid accepts functions
datagrid(hp = range, cyl = unique, newdata = mtcars)
comparisons(mod, newdata = datagrid(hp = fivenum))

# The full dataset is duplicated with each observation given counterfactual
# values of 100 and 110 for the `hp` variable. The original `mtcars` includes
# 32 rows, so the resulting dataset includes 64 rows.
dg <- datagrid(newdata = mtcars, hp = c(100, 110), grid_type = "counterfactual")
nrow(dg)

# We get the same result by feeding a model instead of a data.frame
mod <- lm(mpg ~ hp, mtcars)
dg <- datagrid(model = mod, hp = c(100, 110), grid_type = "counterfactual")
nrow(dg)
```

`hypotheses` *(Non-)Linear Tests for Null Hypotheses, Equivalence, Non Superiority, and Non Inferiority*

Description

Uncertainty estimates are calculated as first-order approximate standard errors for linear or non-linear functions of a vector of random variables with known or estimated covariance matrix. In that sense, `hypotheses` emulates the behavior of the excellent and well-established car::deltaMethod and car::linearHypothesis functions, but it supports more models; requires fewer dependencies; expands the range of tests to equivalence and superiority/inferiority; and offers convenience features like robust standard errors.

To learn more, read the hypothesis tests vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/hypothesis.html
- https://vincentarelbundock.github.io/marginaleffects/

Warning #1: Tests are conducted directly on the scale defined by the `type` argument. For some models, it can make sense to conduct hypothesis or equivalence tests on the `"link"` scale instead of the `"response"` scale which is often the default.

Warning #2: For hypothesis tests on objects produced by the `marginaleffects` package, it is safer to use the `hypothesis` argument of the original function. Using `hypotheses()` may not work in certain environments, in lists, or when working programmatically with *apply style functions.

Usage

```
hypotheses(
  model,
  hypothesis = NULL,
  vcov = NULL,
  conf_level = 0.95,
  df = Inf,
  equivalence = NULL,
  FUN = NULL,
  ...
)
```

Arguments

`model` Model object or object generated by the `comparisons()`, `slopes()`, `predictions()`, or `marginal_means()` functions.

`hypothesis` specify a hypothesis test or custom contrast using a numeric value, vector, or matrix, a string, or a string formula.

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.

- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`df` Degrees of freedom used to compute p values and confidence intervals. A single numeric value between 1 and `Inf`. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used. See insight::get_df for a convenient function to extract degrees of freedom. Ex: `slopes(model, df = insight::get_df(model))`

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. See Details section below.

`FUN` `NULL` or function.

- `NULL` (default): hypothesis test on a model's coefficients, or on the quantities estimated by one of the `marginaleffects` package functions.
- Function which accepts a model object and returns a numeric vector or a data.frame with two columns called `term` and `estimate`. This argument can be useful when users want to conduct a hypothesis test on an arbitrary function of quantities held in a model object.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp + wt + factor(cyl), data = mtcars)

# When `FUN` and `hypotheses` are `NULL`, `hypotheses()` returns a data.frame of parameters
hypotheses(mod)

# Test of equality between coefficients
hypotheses(mod, hypothesis = "hp = wt")
```

```
# Non-linear function
hypotheses(mod, hypothesis = "exp(hp + wt) = 0.1")

# Robust standard errors
hypotheses(mod, hypothesis = "hp = wt", vcov = "HC3")

# b1, b2, ... shortcuts can be used to identify the position of the
# parameters of interest in the output of FUN
hypotheses(mod, hypothesis = "b2 = b3")

# term names with special characters have to be enclosed in backticks
hypotheses(mod, hypothesis = "`factor(cyl)6` = `factor(cyl)8`")

mod2 <- lm(mpg ~ hp * drat, data = mtcars)
hypotheses(mod2, hypothesis = "`hp:drat` = drat")

# predictions(), comparisons(), and slopes()
mod <- glm(am ~ hp + mpg, data = mtcars, family = binomial)
cmp <- comparisons(mod, newdata = "mean")
hypotheses(cmp, hypothesis = "b1 = b2")

mfx <- slopes(mod, newdata = "mean")
hypotheses(cmp, hypothesis = "b2 = 0.2")

pre <- predictions(mod, newdata = datagrid(hp = 110, mpg = c(30, 35)))
hypotheses(pre, hypothesis = "b1 = b2")

# The `FUN` argument can be used to compute standard errors for fitted values
mod <- glm(am ~ hp + mpg, data = mtcars, family = binomial)

f <- function(x) predict(x, type = "link", newdata = mtcars)
p <- hypotheses(mod, FUN = f)
head(p)

f <- function(x) predict(x, type = "response", newdata = mtcars)
p <- hypotheses(mod, FUN = f)
head(p)

# Equivalence, non-inferiority, and non-superiority tests
mod <- lm(mpg ~ hp + factor(gear), data = mtcars)
p <- predictions(mod, newdata = "median")
hypotheses(p, equivalence = c(17, 18))

mfx <- avg_slopes(mod, variables = "hp")
hypotheses(mfx, equivalence = c(-.1, .1))

cmp <- avg_comparisons(mod, variables = "gear", hypothesis = "pairwise")
hypotheses(cmp, equivalence = c(0, 10))
```

`inferences` *(EXPERIMENTAL) Bootstrap and Simulation-Based Inference*

Description

Warning: This function is experimental. It may be renamed, the user interface may change, or the functionality may migrate to arguments in other `marginaleffects` functions.

Apply this function to a `marginaleffects` object to change the inferential method used to compute uncertainty estimates.

Usage

```
inferences(x, method, R = 1000, conf_type = "perc", ...)
```

Arguments

`x` Object produced by one of the core `marginaleffects` functions.

`method` String

- "delta": delta method standard errors
- "boot" package
- "fwb": fractional weighted bootstrap
- "rsample" package
- "simulation" from a multivariate normal distribution (Krinsky & Robb, 1986)
- "mi" multiple imputation for missing data

`R` Number of resamples or simulations.

`conf_type` String: type of bootstrap interval to construct.

- `boot`: "perc", "norm", "basic", or "bca"
- `fwb`: "perc", "norm", "basic", "bc", or "bca"
- `rsample`: "perc" or "bca"
- `simulation`: argument ignored.

`...`

- If `method="boot"`, additional arguments are passed to `boot::boot()`.
- If `method="fwb"`, additional arguments are passed to `fwb::fwb()`.
- If `method="rsample"`, additional arguments are passed to `rsample::bootstraps()`.
- If `method="simulation"`, additional arguments are ignored.

Details

When `method="simulation"`, we conduct simulation-based inference following the method discussed in Krinsky & Robb (1986):

1. Draw `R` sets of simulated coefficients from a multivariate normal distribution with mean equal to the original model's estimated coefficients and variance equal to the model's variance-covariance matrix (classical, "HC3", or other).

2. Use the R sets of coefficients to compute R sets of estimands: predictions, comparisons, or slopes.
3. Take quantiles of the resulting distribution of estimands to obtain a confidence interval and the standard deviation of simulated estimates to estimate the standard error.

When method="fwb", drawn weights are supplied to the model fitting function's weights argument; if the model doesn't accept non-integer weights, this method should not be used. If weights were included in the original model fit, they are extracted by weights() and multiplied by the drawn weights. These weights are supplied to the wts argument of the estimation function (e.g., comparisons()).

Value

A marginaleffects object with simulation or bootstrap resamples and objects attached.

References

Krinsky, I., and A. L. Robb. 1986. "On Approximating the Statistical Properties of Elasticities." Review of Economics and Statistics 68 (4): 715–9.

King, Gary, Michael Tomz, and Jason Wittenberg. "Making the most of statistical analyses: Improving interpretation and presentation." American journal of political science (2000): 347-361

Dowd, Bryan E., William H. Greene, and Edward C. Norton. "Computation of standard errors." Health services research 49.2 (2014): 731-750.

Examples

```
## Not run:
library(marginaleffects)
library(magrittr)
set.seed(1024)
mod <- lm(Sepal.Length ~ Sepal.Width * Species, data = iris)

# bootstrap
avg_predictions(mod, by = "Species") %>%
  inferences(method = "boot")

avg_predictions(mod, by = "Species") %>%
  inferences(method = "rsample")

# Fractional (bayesian) bootstrap
avg_slopes(mod, by = "Species") %>%
  inferences(method = "fwb") %>%
  posterior_draws("rvar") %>%
  data.frame()

# Simulation-based inference
slopes(mod) %>%
  inferences(method = "simulation") %>%
  head()

## End(Not run)
```

`marginal_means` *Marginal Means*

Description

Marginal means are adjusted predictions, averaged across a grid of categorical predictors, holding other numeric predictors at their means. To learn more, read the marginal means vignette, visit the package website, or scroll down this page for a full list of vignettes:

- https://vincentarelbundock.github.io/marginaleffects/articles/marginalmeans.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
marginal_means(
  model,
  variables = NULL,
  newdata = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  transform = NULL,
  cross = FALSE,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  wts = "equal",
  by = NULL,
  ...
)
```

Arguments

`model` Model object

`variables` Focal variables

- Character vector of variable names: compute marginal means for each category of the listed variables.
- `NULL`: calculate marginal means for all logical, character, or factor variables in the dataset used to fit `model`. Hint: Set `cross=TRUE` to compute marginal means for combinations of focal variables.

`newdata` Grid of predictor values over which we marginalize.

- `NULL` create a grid with all combinations of all categorical predictors in the model. Warning: can be expensive.

- Character vector: subset of categorical variables to use when building the balanced grid of predictors. Other variables are held to their mean or mode.
- Data frame: A data frame which includes all the predictors in the original model. The full dataset is replicated once for every combination of the focal variables in the `variables` argument, using the `datagridcf()` function.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`type` string indicates the type (scale) of the predictions used to compute marginal effects or contrasts. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe. If `type` is `NULL` and the default value is "response", the function tries to compute marginal means on the link scale before backtransforming them using the inverse link function.

`transform` A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

`cross` TRUE or FALSE

- `FALSE` (default): Marginal means are computed for each predictor individually.
- `TRUE`: Marginal means are computed for each combination of predictors specified in the `variables` argument.

`hypothesis` specify a hypothesis test or custom contrast using a numeric value, vector, or matrix, a string, or a string formula.

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffect

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. See Details section below.

`p_adjust` Adjust p-values for multiple comparisons: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", or "fdr". See stats::p.adjust

`df` Degrees of freedom used to compute p values and confidence intervals. A single numeric value between 1 and `Inf`. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used. See insight::get_df for a convenient function to extract degrees of freedom. Ex: `slopes(model, df = insight::get_df(model))`

`wts` character value. Weights to use in the averaging.

- "equal": each combination of variables in `newdata` gets equal weight.
- "cells": each combination of values for the variables in the `newdata` gets a weight proportional to its frequency in the original data.
- "proportional": each combination of values for the variables in `newdata` – except for those in the `variables` argument – gets a weight proportional to its frequency in the original data.

`by` Collapse marginal means into categories. Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.

... Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

This function begins by calling the `predictions` function to obtain a grid of predictors, and adjusted predictions for each cell. The grid includes all combinations of the categorical variables listed in the `variables` and `newdata` arguments, or all combinations of the categorical variables used to fit the model if `newdata` is `NULL`. In the prediction grid, numeric variables are held at their means.

After constructing the grid and filling the grid with adjusted predictions, `marginal_means` computes marginal means for the variables listed in the `variables` argument, by average across all categories in the grid.

`marginal_means` can only compute standard errors for linear models, or for predictions on the link scale, that is, with the `type` argument set to "link".

The `marginaleffects` website compares the output of this function to the popular `emmeans` package, which provides similar but more advanced functionality: https://vincentarelbundock.github.io/marginaleffects/

Value

Data frame of marginal means with one row per variable-value combination.

Standard errors using the delta method

Standard errors for all quantities estimated by `marginaleffects` can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to `1e-8`, or to `1e-4` times the smallest absolute model coefficient, whichever is largest.

`marginaleffects` can delegate numeric differentiation to the `numDeriv` package, which allows more flexibility. To do this, users can pass arguments to the `numDeriv::jacobian` function through a global option. For example:

- `options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))`
- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Standard Errors and Confidence Intervals" vignette on the `marginaleffects` website for more details on the computation of standard errors:

https://vincentarelbundock.github.io/marginaleffects/articles/uncertainty.html

Note that the `inferences()` function can be used to compute uncertainty estimates using a bootstrap or simulation-based inference. See the vignette:

https://vincentarelbundock.github.io/marginaleffects/articles/bootstrap.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

https://github.com/vincentarelbundock/marginaleffects/issues

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	brms::posterior_predict
`lme4`	`merMod`	`re.form`	lme4::predict.merMod
		`allow.new.levels`	lme4::predict.merMod
`glmmTMB`	`glmmTMB`	`re.form`	glmmTMB::predict.glmmTMB
		`allow.new.levels`	glmmTMB::predict.glmmTMB
		`zitype`	glmmTMB::predict.glmmTMB
`mgcv`	`bam`	`exclude`	mgcv::predict.bam
`robustlmm`	`rlmerMod`	`re.form`	robustlmm::predict.rlmerMod
		`allow.new.levels`	robustlmm::predict.rlmerMod
`MCMCglmm`	`MCMCglmm`	`ndraws`	

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

`options("marginaleffects_posterior_interval" = "eti")`

`options("marginaleffects_posterior_interval" = "hdi")`

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

`options("marginaleffects_posterior_center" = "mean")`

`options("marginaleffects_posterior_center" = "median")`

When estimates are averaged using the by argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent emmeans package and documentation which inspired this feature.

Examples

```
library(marginaleffects)

# simple marginal means for each level of `cyl`
dat <- mtcars
dat$carb <- factor(dat$carb)
dat$cyl <- factor(dat$cyl)
dat$am <- as.logical(dat$am)
mod <- lm(mpg ~ carb + cyl + am, dat)

marginal_means(
  mod,
  variables = "cyl")

# collapse levels of cyl by averaging
by <- data.frame(
  cyl = c(4, 6, 8),
  by = c("4 & 6", "4 & 6", "8"))
marginal_means(mod,
  variables = "cyl",
  by = by)

# pairwise differences between collapsed levels
marginal_means(mod,
  variables = "cyl",
  by = by,
  hypothesis = "pairwise")

# cross
marginal_means(mod,
  variables = c("cyl", "carb"),
```

```
  cross = TRUE)

# collapsed cross
by <- expand.grid(
  cyl = unique(mtcars$cyl),
  carb = unique(mtcars$carb))
by$by <- ifelse(
  by$cyl == 4,
  paste("Control:", by$carb),
  paste("Treatment:", by$carb))


# Convert numeric variables to categorical before fitting the model
dat <- mtcars
dat$am <- as.logical(dat$am)
dat$carb <- as.factor(dat$carb)
mod <- lm(mpg ~ hp + am + carb, data = dat)

# Compute and summarize marginal means
marginal_means(mod)

# Contrast between marginal means (carb2 - carb1), or "is the 1st marginal means equal to the 2nd?"
# see the vignette on "Hypothesis Tests and Custom Contrasts" on the `marginaleffects` website.
lc <- c(-1, 1, 0, 0, 0, 0)
marginal_means(mod, variables = "carb", hypothesis = "b2 = b1")

marginal_means(mod, variables = "carb", hypothesis = lc)

# Multiple custom contrasts
lc <- matrix(c(
    -2, 1, 1, 0, -1, 1,
    -1, 1, 0, 0, 0, 0
    ),
  ncol = 2,
  dimnames = list(NULL, c("A", "B")))
marginal_means(mod, variables = "carb", hypothesis = lc)
```

`plot_comparisons` *Plot Conditional or Marginal Comparisons*

Description

Plot comparisons on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The by argument is used to plot marginal comparisons, that is, comparisons made on the original data, but averaged by subgroups. This is analogous to using the by argument in the `comparisons()` function.

The `condition` argument is used to plot conditional comparisons, that is, comparisons made on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `comparisons()` call. Unspecified variables are held at their mean or mode.

See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://vincentarelbundock.github.io/marginaleffects/articles/plot.html
- https://vincentarelbundock.github.io/marginaleffects

Usage

```
plot_comparisons(
  model,
  variables = NULL,
  condition = NULL,
  by = NULL,
  type = "response",
  vcov = NULL,
  conf_level = 0.95,
  comparison = "difference",
  transform = NULL,
  rug = FALSE,
  gray = FALSE,
  draw = TRUE,
  ...
)
```

Arguments

`model` Model object

`variables` Name of the variable whose contrast we want to plot on the y-axis.

`condition` Conditional slopes

- Character vector (max length 3): Names of the predictors to display.
- Named list (max length 3): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"
- 1: x-axis. 2: color/shape. 3: facets.
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.

- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`comparison` How should pairs of predictions be compared? Difference, ratio, odds ratio, or user-defined functions.

- string: shortcuts to common contrast functions.
  - Supported shortcuts strings: difference, differenceavg, differenceavgwts, dydx, eyex, eydx, dyex, dydxavg, eyexavg, eydxavg, dyexavg, dydxavgwts, eyexavgwts, eydxavgwts, dyexavgwts, ratio, ratioavg, ratioavgwts, lnratio, lnratioavg, lnratioavgwts, lnor, lnoravg, lnoravgwts, expdydx, expdydxavg, expdydxavgwts
  - See the Comparisons section below for definitions of each transformation.
- function: accept two equal-length numeric vectors of adjusted predictions (`hi` and `lo`) and returns a vector of contrasts of the same length, or a unique numeric value.

- See the Transformations section below for examples of valid functions.

transform	string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"
rug	TRUE displays tick marks on the axes to mark the distribution of raw data.
gray	FALSE grayscale or color plot
draw	`TRUE` returns a `ggplot2` plot. `FALSE` returns a `data.frame` of the underlying data.
...	Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object

Examples

```
mod <- lm(mpg ~ hp * drat * factor(am), data = mtcars)

plot_comparisons(mod, variables = "hp", condition = "drat")

plot_comparisons(mod, variables = "hp", condition = c("drat", "am"))

plot_comparisons(mod, variables = "hp", condition = list("am", "drat" = 3:5))

plot_comparisons(mod, variables = "am", condition = list("hp", "drat" = range))

plot_comparisons(mod, variables = "am", condition = list("hp", "drat" = "threenum"))
```

`plot_predictions` *Plot Conditional or Marginal Predictions*

Description

Plot predictions on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The by argument is used to plot marginal predictions, that is, predictions made on the original data, but averaged by subgroups. This is analogous to using the by argument in the `predictions()` function.

The `condition` argument is used to plot conditional predictions, that is, predictions made on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `predictions()` call. Unspecified variables are held at their mean or mode.

See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://vincentarelbundock.github.io/marginaleffects/articles/plot.html
- https://vincentarelbundock.github.io/marginaleffects

Usage

```
plot_predictions(
  model,
  condition = NULL,
  by = NULL,
  type = NULL,
  vcov = NULL,
  conf_level = 0.95,
  transform = NULL,
  points = 0,
  rug = FALSE,
  gray = FALSE,
  draw = TRUE,
  ...
)
```

Arguments

`model` Model object

`condition` Conditional predictions

- Character vector (max length 3): Names of the predictors to display.
- Named list (max length 3): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"
- 1: x-axis. 2: color/shape. 3: facets.
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`

`by` Marginal predictions

- Character vector (max length 3): Names of the categorical predictors to marginalize across.
- 1: x-axis. 2: color. 3: facets.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

vcov
: Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

conf_level
: numeric value between 0 and 1. Confidence level to use to build a confidence interval.

transform
: A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

points
: Number between 0 and 1 which controls the transparency of raw data points. 0 (default) does not display any points.

rug
: TRUE displays tick marks on the axes to mark the distribution of raw data.

gray
: FALSE grayscale or color plot

draw
: `TRUE` returns a `ggplot2` plot. `FALSE` returns a `data.frame` of the underlying data.

...
: Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Value

A `ggplot2` object or data frame (if `draw=FALSE`)

Examples

```
mod <- lm(mpg ~ hp + wt, data = mtcars)
plot_predictions(mod, condition = "wt")

mod <- lm(mpg ~ hp * wt * am, data = mtcars)
plot_predictions(mod, condition = c("hp", "wt"))

plot_predictions(mod, condition = list("hp", wt = "threenum"))

plot_predictions(mod, condition = list("hp", wt = range))
```

---

plot_slopes *Plot Conditional or Marginal Slopes*

Description

Plot slopes on the y-axis against values of one or more predictors (x-axis, colors/shapes, and facets).

The `by` argument is used to plot marginal slopes, that is, slopes made on the original data, but averaged by subgroups. This is analogous to using the `by` argument in the `slopes()` function.

The `condition` argument is used to plot conditional slopes, that is, slopes made on a user-specified grid. This is analogous to using the `newdata` argument and `datagrid()` function in a `slopes()` call. Unspecified variables are held at their mean or mode.

See the "Plots" vignette and website for tutorials and information on how to customize plots:

- https://vincentarelbundock.github.io/marginaleffects/articles/plot.html
- https://vincentarelbundock.github.io/marginaleffects

Usage

```
plot_slopes(
  model,
  variables = NULL,
  condition = NULL,
  by = NULL,
  type = "response",
  vcov = NULL,
  conf_level = 0.95,
  slope = "dydx",
  rug = FALSE,
  gray = FALSE,
  draw = TRUE,
  ...
)
```

Arguments

`model` Model object

`variables` Name of the variable whose marginal effect (slope) we want to plot on the y-axis.

`condition` Conditional slopes

- Character vector (max length 3): Names of the predictors to display.
- Named list (max length 3): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"
- 1: x-axis. 2: color/shape. 3: facets.
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.

- One-sided formula which indicates the name of cluster variables (e.g., ~unit_id). This formula is passed to the cluster argument of the sandwich::vcovCL function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., stats::vcov(model))

conf_level numeric value between 0 and 1. Confidence level to use to build a confidence interval.

slope string indicates the type of slope or (semi-)elasticity to compute:
- "dydx": dY/dX
- "eyex": dY/dX * Y / X
- "eydx": dY/dX * Y
- "dyex": dY/dX / X

rug TRUE displays tick marks on the axes to mark the distribution of raw data.

gray FALSE grayscale or color plot

draw TRUE returns a ggplot2 plot. FALSE returns a data.frame of the underlying data.

... Additional arguments are passed to the predict() method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the marginaleffects website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Value

A ggplot2 object

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp * drat * factor(am), data = mtcars)

plot_slopes(mod, variables = "hp", condition = "drat")

plot_slopes(mod, variables = "hp", condition = c("drat", "am"))

plot_slopes(mod, variables = "hp", condition = list("am", "drat" = 3:5))

plot_slopes(mod, variables = "am", condition = list("hp", "drat" = range))

plot_slopes(mod, variables = "am", condition = list("hp", "drat" = "threenum"))
```

posterior_draws *Extract Posterior Draws or Bootstrap Resamples from* marginaleffects *Objects*

Description

Extract Posterior Draws or Bootstrap Resamples from marginaleffects Objects

Usage

```
posterior_draws(x, shape = "long")
```

Arguments

x	An object produced by a marginaleffects package function, such as predictions(), avg_slopes(), hypotheses(), etc.
shape	string indicating the shape of the output format: • "long": long format data frame • "DxP": Matrix with draws as rows and parameters as columns • "PxD": Matrix with draws as rows and parameters as columns • "rvar": Random variable datatype (see posterior package documentation).

Value

A data.frame with drawid and draw columns.

---

predictions *Predictions*

Description

Outcome predicted by a fitted model on a specified scale for a given combination of values of the predictor variables, such as their observed values, their means, or factor levels (a.k.a. "reference grid").

- predictions(): unit-level (conditional) estimates.
- avg_predictions(): average (marginal) estimates.

The newdata argument and the datagrid() function can be used to control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the predictions vignette and package website for worked examples and case studies:

- https://vincentarelbundock.github.io/marginaleffects/articles/predictions.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
predictions(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  by = FALSE,
  byfun = NULL,
  wts = NULL,
  transform = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  ...
)

avg_predictions(
  model,
  newdata = NULL,
  variables = NULL,
  vcov = TRUE,
  conf_level = 0.95,
  type = NULL,
  by = TRUE,
  byfun = NULL,
  wts = NULL,
  transform = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  ...
)
```

Arguments

`model` Model object

`newdata` Grid of predictor values at which we evaluate predictions.

- `NULL` (default): Predictions for each observed value in the original dataset.
- data frame: Predictions for each row of the `newdata` data frame.
- string:
  - "mean": Predictions at the Mean. Predictions when each predictor is held at its mean or mode.

  - "median": Predictions at the Median. Predictions when each predictor is held at its median or mode.
  - "marginalmeans": Predictions at Marginal Means. See Details section below.
  - "tukey": Predictions at Tukey's 5 numbers.
  - "grid": Predictions on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).
- `datagrid()` call to specify a custom grid of regressors. For example:
  - `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes.
  - See the Examples section and the `datagrid()` documentation.

`variables` Counterfactual variables.

- Output:
  - `predictions()`: The entire dataset is replicated once for each unique combination of `variables`, and predictions are made.
  - `avg_predictions()`: The entire dataset is replicated, predictions are made, and they are marginalized by `variables` categories.
  - Warning: This can be expensive in large datasets.
  - Warning: Users who need "conditional" predictions should use the `newdata` argument instead of `variables`.
- Input:
  - `NULL`: computes one prediction per row of `newdata`
  - Character vector: the dataset is replicated once of every combination of unique values of the variables identified in `variables`.
  - Named list: names identify the subset of variables of interest and their values. For numeric variables, the `variables` argument supports functions and string shortcuts:
    * A function which returns a numeric value
    * Numeric vector: Contrast between the 2nd element and the 1st element of the `x` vector.
    * "iqr": Contrast across the interquartile range of the regressor.
    * "sd": Contrast across one standard deviation around the regressor mean.
    * "2sd": Contrast across two standard deviations around the regressor mean.
    * "minmax": Contrast between the maximum and the minimum values of the regressor.
    * "threenum": mean and 1 standard deviation on both sides
    * "fivenum": Tukey's five numbers

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.

- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.
- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe. See the details section for a note on backtransformation.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.

`byfun` A function such as `mean()` or `sum()` used to aggregate estimates within the subgroups defined by the `by` argument. `NULL` uses the `mean()` function. Must accept a numeric vector and return a single numeric value. This is sometimes used to take the sum or mean of predicted probabilities across outcome or predictor levels. See examples section.

`wts` string or numeric: weights to use when computing average contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not the unit-level estimates themselves.

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).

`transform` A function applied to unit-level adjusted predictions and confidence intervals just before the function returns results. For bayesian models, this function is applied to individual draws from the posterior distribution, before computing summaries.

`hypothesis` specify a hypothesis test or custom contrast using a numeric value, vector, or matrix, a string, or a string formula.

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffec

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. See Details section below.

`p_adjust` Adjust p-values for multiple comparisons: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", or "fdr". See stats::p.adjust

`df` Degrees of freedom used to compute p values and confidence intervals. A single numeric value between 1 and `Inf`. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used. See insight::get_df for a convenient function to extract degrees of freedom. Ex: `slopes(model, df = insight::get_df(model))`

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the ?marginaleffects documentation for a non-exhaustive list of available arguments.

Details

For glm() or gam::gam() models with type=NULL (the default), predictions() first predicts on the link scale, and then backtransforms the estimates and confidence intervals. This implies that the estimate produced by avg_predictions() will not be exactly equal to the average of the estimate column produced by predictions(). Users can circumvent this behavior and average predictions directly on the response scale by setting type="response" explicitly. With type="response", the intervals are symmetric and may have undesirable properties (e.g., stretching beyond the [0,1] bounds for a binary outcome regression).

Value

A data.frame with one row per observation and several columns:

- rowid: row number of the newdata data frame
- type: prediction type, as defined by the type argument
- group: (optional) value of the grouped outcome (e.g., categorical outcome models)
- estimate: predicted outcome
- std.error: standard errors computed using the delta method.
- conf.low: lower bound of the confidence interval (or equal-tailed interval for bayesian models)
- conf.high: upper bound of the confidence interval (or equal-tailed interval for bayesian models)

See ?print.marginaleffects for printing options.

Functions

- avg_predictions(): Average predictions

Standard errors using the delta method

Standard errors for all quantities estimated by marginaleffects can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to 1e-8, or to 1e-4 times the smallest absolute model coefficient, whichever is largest.

marginaleffects can delegate numeric differentiation to the numDeriv package, which allows more flexibility. To do this, users can pass arguments to the numDeriv::jacobian function through a global option. For example:

- options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))

- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Standard Errors and Confidence Intervals" vignette on the `marginaleffects` website for more details on the computation of standard errors:

https://vincentarelbundock.github.io/marginaleffects/articles/uncertainty.html

Note that the `inferences()` function can be used to compute uncertainty estimates using a bootstrap or simulation-based inference. See the vignette:

https://vincentarelbundock.github.io/marginaleffects/articles/bootstrap.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

https://github.com/vincentarelbundock/marginaleffects/issues

Package	Class	Argument	Documentation
brms	brmsfit	ndraws	brms::posterior_predict
		re_formula	brms::posterior_predict
lme4	merMod	re.form	lme4::predict.merMod
		allow.new.levels	lme4::predict.merMod
glmmTMB	glmmTMB	re.form	glmmTMB::predict.glmmTMB
		allow.new.levels	glmmTMB::predict.glmmTMB
		zitype	glmmTMB::predict.glmmTMB
mgcv	bam	exclude	mgcv::predict.bam
robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod
		allow.new.levels	robustlmm::predict.rlmerMod
MCMCglmm	MCMCglmm	ndraws	

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

`options("marginaleffects_posterior_interval" = "eti")`

`options("marginaleffects_posterior_interval" = "hdi")`

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

`options("marginaleffects_posterior_center" = "mean")`

`options("marginaleffects_posterior_center" = "median")`

When estimates are averaged using the by argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in by argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Examples

```
# Adjusted Prediction for every row of the original dataset
mod <- lm(mpg ~ hp + factor(cyl), data = mtcars)
pred <- predictions(mod)
head(pred)

# Adjusted Predictions at User-Specified Values of the Regressors
predictions(mod, newdata = datagrid(hp = c(100, 120), cyl = 4))

m <- lm(mpg ~ hp + drat + factor(cyl) + factor(am), data = mtcars)
predictions(m, newdata = datagrid(FUN_factor = unique, FUN_numeric = median))

# Average Adjusted Predictions (AAP)
library(dplyr)
mod <- lm(mpg ~ hp * am * vs, mtcars)

avg_predictions(mod)

predictions(mod, by = "am")

# Conditional Adjusted Predictions
plot_predictions(mod, condition = "hp")
```

```
# Counterfactual predictions with the `variables` argument
# the `mtcars` dataset has 32 rows

mod <- lm(mpg ~ hp + am, data = mtcars)
p <- predictions(mod)
head(p)
nrow(p)

# average counterfactual predictions
avg_predictions(mod, variables = "am")

# counterfactual predictions obtained by replicating the entire for different
# values of the predictors
p <- predictions(mod, variables = list(hp = c(90, 110)))
nrow(p)


# hypothesis test: is the prediction in the 1st row equal to the prediction in the 2nd row
mod <- lm(mpg ~ wt + drat, data = mtcars)

predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = "b1 = b2")

# same hypothesis test using row indices
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = "b1 - b2 = 0")

# same hypothesis test using numeric vector of weights
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
predictions(
    mod,
    newdata = datagrid(wt = 2:3),
    hypothesis = lc)


# `by` argument
mod <- lm(mpg ~ hp * am * vs, data = mtcars)
predictions(mod, by = c("am", "vs"))
```

```
library(nnet)
nom <- multinom(factor(gear) ~ mpg + am * vs, data = mtcars, trace = FALSE)

# first 5 raw predictions
predictions(nom, type = "probs") |> head()

# average predictions
avg_predictions(nom, type = "probs", by = "group")

by <- data.frame(
    group = c("3", "4", "5"),
    by = c("3,4", "3,4", "5"))

predictions(nom, type = "probs", by = by)

# sum of predicted probabilities for combined response levels
mod <- multinom(factor(cyl) ~ mpg + am, data = mtcars, trace = FALSE)
by <- data.frame(
    by = c("4,6", "4,6", "8"),
    group = as.character(c(4, 6, 8)))
predictions(mod, newdata = "mean", byfun = sum, by = by)
```

`print.marginaleffects` *Print* `marginaleffects` *objects*

Description

This function controls the text which is printed to the console when one of the core `marginalefffects` functions is called and the object is returned: `predictions()`, `comparisons()`, `slopes()`, `marginal_means()`, `hypotheses()`, `avg_predictions()`, `avg_comparisons()`, `avg_slopes()`.

All of those functions return standard data frames. Columns can be extracted by name, `predictions(model)$estimate`, and all the usual data manipulation functions work out-of-the-box: `colnames()`, `head()`, `subset()`, `dplyr::filter()`, `dplyr::arrange()`, etc.

Some of the data columns are not printed by default. You can disable pretty printing and print the full results as a standard data frame using the `style` argument or by applying `as.data.frame()` on the object. See examples below.

Usage

```
## S3 method for class 'marginaleffects'
print(
  x,
  digits = getOption("marginaleffects_print_digits", default = 3),
  p_eps = getOption("marginaleffects_print_p_eps", default = 0.001),
  topn = getOption("marginaleffects_print_topn", default = 5),
  nrows = getOption("marginaleffects_print_nrows", default = 30),
  ncols = getOption("marginaleffects_print_ncols", default = 30),
```

```
  style = getOption("marginaleffects_print_style", default = "summary"),
  ...
)
```

Arguments

x	An object produced by one of the marginaleffects package functions.
digits	The number of digits to display.
p_eps	p values smaller than this number are printed in "<0.001" style.
topn	The number of rows to be printed from the beginning and end of tables with more than nrows rows.
nrows	The number of rows which will be printed before truncation.
ncols	The maximum number of column names to display at the bottom of the printed output.
style	"summary" or "data.frame"
...	Other arguments are currently ignored.

Examples

```
library(marginaleffects)
mod <- lm(mpg ~ hp + am + factor(gear), data = mtcars)
p <- predictions(mod, by = c("am", "gear"))
p

subset(p, am == 1)

print(p, style = "data.frame")

data.frame(p)
```

---

slopes *Slopes (aka Partial derivatives, Marginal Effects, or Trends)*

Description

Partial derivative of the regression equation with respect to a regressor of interest.

- slopes(): unit-level (conditional) estimates.
- avg_slopes(): average (marginal) estimates.

The newdata argument and the datagrid() function can be used to control where statistics are evaluated in the predictor space: "at observed values", "at the mean", "at representative values", etc.

See the slopes vignette and package website for worked examples and case studies:

- https://vincentarelbundock.github.io/marginaleffects/articles/slopes.html
- https://vincentarelbundock.github.io/marginaleffects/

Usage

```
slopes(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  by = FALSE,
  vcov = TRUE,
  conf_level = 0.95,
  slope = "dydx",
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  eps = NULL,
  ...
)

avg_slopes(
  model,
  newdata = NULL,
  variables = NULL,
  type = NULL,
  by = TRUE,
  vcov = TRUE,
  conf_level = 0.95,
  slope = "dydx",
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  p_adjust = NULL,
  df = Inf,
  eps = NULL,
  ...
)
```

Arguments

`model` Model object

`newdata` Grid of predictor values at which we evaluate the slopes.

- `NULL` (default): Unit-level slopes for each observed value in the original dataset.
- data frame: Unit-level slopes for each row of the `newdata` data frame.
- `datagrid()` call to specify a custom grid of regressors. For example:
  - `newdata = datagrid(cyl = c(4, 6))`: `cyl` variable equal to 4 and 6 and other regressors fixed at their means or modes.

- See the Examples section and the `datagrid()` documentation.

- string:
  - "mean": Marginal Effects at the Mean. Slopes when each predictor is held at its mean or mode.
  - "median": Marginal Effects at the Median. Slopes when each predictor is held at its median or mode.
  - "marginalmeans": Marginal Effects at Marginal Means. See Details section below.
  - "tukey": Marginal Effects at Tukey's 5 numbers.
  - "grid": Marginal Effects on a grid of representative numbers (Tukey's 5 numbers and unique values of categorical predictors).

`variables` Focal variables

- `NULL`: compute slopes or comparisons for all the variables in the model object (can be slow).
- Character vector: subset of variables (usually faster).

`type` string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the default value is used. This default is the first model-related row in the `marginaleffects:::type_dictionary` dataframe.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.

`vcov` Type of uncertainty estimates to report (e.g., for robust standard errors). Acceptable values:

- FALSE: Do not compute standard errors. This can speed up computation considerably.
- TRUE: Unit-level standard errors using the default `vcov(model)` variance-covariance matrix.
- String which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`
  - Heteroskedasticity and autocorrelation consistent: `"HAC"`
  - Mixed-Models degrees of freedom: "satterthwaite", "kenward-roger"
  - Other: `"NeweyWest"`, `"KernHAC"`, `"OPG"`. See the `sandwich` package documentation.

- One-sided formula which indicates the name of cluster variables (e.g., `~unit_id`). This formula is passed to the `cluster` argument of the `sandwich::vcovCL` function.
- Square covariance matrix
- Function which returns a covariance matrix (e.g., `stats::vcov(model)`)

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`slope` string indicates the type of slope or (semi-)elasticity to compute:

- "dydx": dY/dX
- "eyex": dY/dX * Y / X
- "eydx": dY/dX * Y
- "dyex": dY/dX / X

`wts` string or numeric: weights to use when computing average contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not the unit-level estimates themselves.

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).

`hypothesis` specify a hypothesis test or custom contrast using a numeric value, vector, or matrix, a string, or a string formula.

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String formula to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
- String:
  - "pairwise": pairwise differences between estimates in each row.
  - "reference": differences between the estimates in each row and the estimate in the first row.
  - "sequential": difference between an estimate and the estimate in the next row.

- "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- See the Examples section below and the vignette: https://vincentarelbundock.github.io/marginaleffec

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. See Details section below.

`p_adjust` Adjust p-values for multiple comparisons: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", or "fdr". See stats::p.adjust

`df` Degrees of freedom used to compute p values and confidence intervals. A single numeric value between 1 and `Inf`. When `df` is `Inf`, the normal distribution is used. When `df` is finite, the `t` distribution is used. See insight::get_df for a convenient function to extract degrees of freedom. Ex: `slopes(model, df = insight::get_df(model))`

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`...` Additional arguments are passed to the `predict()` method supplied by the modeling package.These arguments are particularly useful for mixed-effects or bayesian models (see the online vignettes on the `marginaleffects` website). Available arguments can vary from model to model, depending on the range of supported arguments by each modeling package. See the "Model-Specific Arguments" section of the `?marginaleffects` documentation for a non-exhaustive list of available arguments.

Details

A "slope" or "marginal effect" is the partial derivative of the regression equation with respect to a variable in the model. This function uses automatic differentiation to compute slopes for a vast array of models, including non-linear models with transformations (e.g., polynomials). Uncertainty estimates are computed using the delta method.

Numerical derivatives for the `slopes` function are calculated using a simple epsilon difference approach: $\partial Y/\partial X = (f(X + \varepsilon/2) - f(X + \varepsilon/2))/\varepsilon$, where f is the `predict()` method associated with the model class, and ε is determined by the `eps` argument.

Value

A `data.frame` with one row per observation (per term/group) and several columns:

- `rowid`: row number of the `newdata` data frame
- `type`: prediction type, as defined by the `type` argument
- `group`: (optional) value of the grouped outcome (e.g., categorical outcome models)
- `term`: the variable whose marginal effect is computed
- `dydx`: slope of the outcome with respect to the term, for a given combination of predictor values

- `std.error`: standard errors computed by via the delta method.

See `?print.marginaleffects` for printing options.

Functions

- `avg_slopes()`: Average slopes

Standard errors using the delta method

Standard errors for all quantities estimated by `marginaleffects` can be obtained via the delta method. This requires differentiating a function with respect to the coefficients in the model using a finite difference approach. In some models, the delta method standard errors can be sensitive to various aspects of the numeric differentiation strategy, including the step size. By default, the step size is set to `1e-8`, or to `1e-4` times the smallest absolute model coefficient, whichever is largest.

`marginaleffects` can delegate numeric differentiation to the `numDeriv` package, which allows more flexibility. To do this, users can pass arguments to the `numDeriv::jacobian` function through a global option. For example:

- `options(marginaleffects_numDeriv = list(method = "simple", method.args = list(eps = 1e-6)))`
- `options(marginaleffects_numDeriv = list(method = "Richardson", method.args = list(eps = 1e-5)))`
- `options(marginaleffects_numDeriv = NULL)`

See the "Standard Errors and Confidence Intervals" vignette on the `marginaleffects` website for more details on the computation of standard errors:

https://vincentarelbundock.github.io/marginaleffects/articles/uncertainty.html

Note that the `inferences()` function can be used to compute uncertainty estimates using a bootstrap or simulation-based inference. See the vignette:

https://vincentarelbundock.github.io/marginaleffects/articles/bootstrap.html

Model-Specific Arguments

Some model types allow model-specific arguments to modify the nature of marginal effects, predictions, marginal means, and contrasts. Please report other package-specific `predict()` arguments on Github so we can add them to the table below.

https://github.com/vincentarelbundock/marginaleffects/issues

Package	Class	Argument	Documentation
`brms`	`brmsfit`	`ndraws`	brms::posterior_predict
		`re_formula`	brms::posterior_predict
`lme4`	`merMod`	`re.form`	lme4::predict.merMod
		`allow.new.levels`	lme4::predict.merMod
`glmmTMB`	`glmmTMB`	`re.form`	glmmTMB::predict.glmmTMB
		`allow.new.levels`	glmmTMB::predict.glmmTMB
		`zitype`	glmmTMB::predict.glmmTMB
`mgcv`	`bam`	`exclude`	mgcv::predict.bam

robustlmm	rlmerMod	re.form	robustlmm::predict.rlmerMod
		allow.new.levels	robustlmm::predict.rlmerMod
MCMCglmm	MCMCglmm	ndraws	

Bayesian posterior summaries

By default, credible intervals in bayesian models are built as equal-tailed intervals. This can be changed to a highest density interval by setting a global option:

`options("marginaleffects_posterior_interval" = "eti")`

`options("marginaleffects_posterior_interval" = "hdi")`

By default, the center of the posterior distribution in bayesian models is identified by the median. Users can use a different summary function by setting a global option:

`options("marginaleffects_posterior_center" = "mean")`

`options("marginaleffects_posterior_center" = "median")`

When estimates are averaged using the `by` argument, the `tidy()` function, or the `summary()` function, the posterior distribution is marginalized twice over. First, we take the average *across* units but *within* each iteration of the MCMC chain, according to what the user requested in `by` argument or `tidy()/summary()` functions. Then, we identify the center of the resulting posterior using the function supplied to the `"marginaleffects_posterior_center"` option (the median by default).

Equivalence, Inferiority, Superiority

θ is an estimate, σ_θ its estimated standard error, and $[a, b]$ are the bounds of the interval supplied to the `equivalence` argument.

Non-inferiority:

- H_0: $\theta \leq a$
- H_1: $\theta > a$
- $t = (\theta - a)/\sigma_\theta$
- p: Upper-tail probability

Non-superiority:

- H_0: $\theta \geq b$
- H_1: $\theta < b$
- $t = (\theta - b)/\sigma_\theta$
- p: Lower-tail probability

Equivalence: Two One-Sided Tests (TOST)

- p: Maximum of the non-inferiority and non-superiority p values.

Thanks to Russell V. Lenth for the excellent `emmeans` package and documentation which inspired this feature.

Examples

```
# Unit-level (conditional) Marginal Effects
mod <- glm(am ~ hp * wt, data = mtcars, family = binomial)
mfx <- slopes(mod)
head(mfx)

# Average Marginal Effect (AME)
avg_slopes(mod, by = TRUE)


# Marginal Effect at the Mean (MEM)
slopes(mod, newdata = datagrid())

# Marginal Effect at User-Specified Values
# Variables not explicitly included in `datagrid()` are held at their means
slopes(mod, newdata = datagrid(hp = c(100, 110)))

# Group-Average Marginal Effects (G-AME)
# Calculate marginal effects for each observation, and then take the average
# marginal effect within each subset of observations with different observed
# values for the `cyl` variable:
mod2 <- lm(mpg ~ hp * cyl, data = mtcars)
avg_slopes(mod2, variables = "hp", by = "cyl")

# Marginal Effects at User-Specified Values (counterfactual)
# Variables not explicitly included in `datagrid()` are held at their
# original values, and the whole dataset is duplicated once for each
# combination of the values in `datagrid()`
mfx <- slopes(mod,
              newdata = datagrid(hp = c(100, 110),
              grid_type = "counterfactual"))
head(mfx)

# Heteroskedasticity robust standard errors
mfx <- slopes(mod, vcov = sandwich::vcovHC(mod))
head(mfx)

# hypothesis test: is the `hp` marginal effect at the mean equal to the `drat` marginal effect
mod <- lm(mpg ~ wt + drat, data = mtcars)

slopes(
    mod,
    newdata = "mean",
    hypothesis = "wt = drat")

# same hypothesis test using row indices
slopes(
    mod,
    newdata = "mean",
```

```
    hypothesis = "b1 - b2 = 0")

# same hypothesis test using numeric vector of weights
slopes(
    mod,
    newdata = "mean",
    hypothesis = c(1, -1))

# two custom contrasts using a matrix of weights
lc <- matrix(c(
    1, -1,
    2, 3),
    ncol = 2)
colnames(lc) <- c("Contrast A", "Contrast B")
slopes(
    mod,
    newdata = "mean",
    hypothesis = lc)
```

Index