Package 'modelbased'

June 6, 2021

Type Package

Title Estimation of Model-Based Predictions, Contrasts and Means

Version 0.7.0

Maintainer Dominique Makowski <dom.makowski@gmail.com>

Description Implements a general interface for model-based estimations for a wide variety of models (see support list of insight; Lüdecke, Waggoner & Makowski (2019) <doi:10.21105/joss.01412>), used in the computation of marginal means, contrast analysis and predictions.

License GPL-3

URL https://easystats.github.io/modelbased/

BugReports https://github.com/easystats/modelbased/issues

Imports graphics, stats, utils, bayestestR (>= 0.10.0), effectsize (>= 0.4.5), insight (>= 0.14.1), parameters (>= 0.14.0), performance

Suggests brms, coda, correlation, dplyr, emmeans (>= 1.5.3), gamm4, gganimate, ggplot2, glmmTMB, knitr, lme4, logspline, MASS, Matrix, merTools, mgcv, rmarkdown, rstanarm, rtdists, see, testthat, spelling

Encoding UTF-8

RoxygenNote 7.1.1.9001

Config/testthat/edition 3

Config/testthat/parallel true

Language en-US

NeedsCompilation no

Author Dominique Makowski [aut, cre] (<https://orcid.org/0000-0001-5375-9967>, @Dom_Makowski),
Daniel Lüdecke [aut] (<https://orcid.org/0000-0002-8895-3206>, @strengejacke),
Mattan S. Ben-Shachar [aut] (<https://orcid.org/0000-0002-4287-4801>,

@mattansb),
Indrajeet Patil [aut] (<https://orcid.org/0000-0003-1995-6531>,
@patilindrajeets)

Repository CRAN

Date/Publication 2021-06-06 05:40:02 UTC

R topics documented:

describe_nonlinear 2
estimate_contrasts 3
estimate_expectation 5
estimate_grouplevel 7
estimate_means 8
estimate_slopes 10
find_inversions 11
model_emmeans 12
smoothing 13
visualisation_matrix 14
visualisation_recipe 15
zero_crossings 19

Index **21**

`describe_nonlinear` *Describe the smooth term (for GAMs) or non-linear predictors*

Description

This function summarises the smooth term trend in terms of linear segments. Using the approximative derivative, it separates a non-linear vector into quasi-linear segments (in which the trend is either positive or negative). Each of this segment its characterized by its beginning, end, size (in proportion, relative to the total size) trend (the linear regression coefficient) and linearity (the R2 of the linear regression).

Usage

```
describe_nonlinear(data, ...)

## S3 method for class 'data.frame'
describe_nonlinear(data, x = NULL, y = NULL, ...)

estimate_smooth(data, ...)
```

Arguments

`data`	The data containing the link, as for instance obtained by `estimate_relation`.
`...`	Other arguments to be passed to or from.
`x, y`	The name of the responses variable (`y`) predicting variable (`x`).

Value

A dataframe of linear description of non-linear terms.

Examples

```
library(modelbased)

# Create data
data <- data.frame(x = rnorm(200))
data$y <- data$x^2 + rnorm(200, 0, 0.5)

model <- lm(y ~ poly(x, 2), data = data)
link_data <- estimate_relation(model, length = 100)

describe_nonlinear(link_data, x = "x")
```

---

estimate_contrasts *Estimate Marginal Contrasts*

Description

Run a contrast analysis by estimating the differences between each level of a factor. See also other related functions such as `estimate_means` and `estimate_slopes`.

Usage

```
estimate_contrasts(
  model,
  levels = NULL,
  fixed = NULL,
  modulate = NULL,
  transform = "none",
  ci = 0.95,
  adjust = "holm",
  ...
)
```

Arguments

`model`	A statistical model.
`levels`	A character vector or formula specifying the names of the predicting factors over which to estimate means or contrasts.
`fixed`	A character vector indicating the names of the predictors to be "fixed" (i.e., maintained), so that the estimation is made at these values.
`modulate`	A character vector indicating the names of a numeric variable along which the means or the contrasts will be estimated. Other arguments from `visualisation_matrix`, such as `length` to adjust the number of data points.

transform	Is passed to the type argument in emmeans::emmeans(). See this vignette. Can be "none" (default for contrasts), "response" (default for means), "mu", "unlink", "log". "none" will leave the values on scale of the linear predictors. "response" will transform them on scale of the response variable. Thus for a logistic model, "none" will give estimations expressed in log-odds (probabilities on logit scale) and "response" in terms of probabilities.
ci	Uncertainty Interval (CI) level. Default to 95% (0.95).
adjust	The p-values adjustment method for frequentist multiple comparisons. Can be one of "holm" (default), "tukey", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr" or "none". See the p-value adjustment section in the emmeans::test documentation.
...	Other arguments passed for instance to visualisation_matrix.

Value

A data frame of estimated contrasts.

Examples

```
library(modelbased)

# Basic usage
model <- lm(Sepal.Width ~ Species, data = iris)
estimate_contrasts(model)

# Dealing with interactions
model <- lm(Sepal.Width ~ Species * Petal.Width, data = iris)
estimate_contrasts(model)
estimate_contrasts(model, fixed = "Petal.Width")
estimate_contrasts(model, modulate = "Petal.Width", length = 4)
estimate_contrasts(model, levels = "Petal.Width", length = 4)

# Standardized differences
estimated <- estimate_contrasts(lm(Sepal.Width ~ Species, data = iris))
effectsize::standardize(estimated)

# Other models (mixed, Bayesian, ...)
if (require("lme4")) {
  data <- iris
  data$Petal.Length_factor <- ifelse(data$Petal.Length < 4.2, "A", "B")

  model <- lmer(Sepal.Width ~ Species + (1 | Petal.Length_factor), data = data)
  estimate_contrasts(model)
}

data <- mtcars
data$cyl <- as.factor(data$cyl)
data$am <- as.factor(data$am)
## Not run:
if (require("rstanarm")) {
  model <- stan_glm(mpg ~ cyl * am, data = data, refresh = 0)
```

```
  estimate_contrasts(model)
  estimate_contrasts(model, fixed = "am")

  model <- stan_glm(mpg ~ cyl * wt, data = data, refresh = 0)
  estimate_contrasts(model)
  estimate_contrasts(model, fixed = "wt")
  estimate_contrasts(model, modulate = "wt", length = 4)
  estimate_contrasts(model, levels = "wt", length = 4)

 model <- stan_glm(Sepal.Width ~ Species + Petal.Width + Petal.Length, data = iris, refresh = 0)
 estimate_contrasts(model, fixed = "Petal.Width", modulate = "Petal.Length", test = "bf")
}

if (require("brms")) {
  model <- brm(mpg ~ cyl * am, data = data, refresh = 0)
  estimate_contrasts(model)
}

## End(Not run)
```

`estimate_expectation` *Generates predictions from models*

Description

`estimate_link` is a shortcut to `estimate_response` with `data = "grid"`. `estimate_response` would be used in the context of generating actual predictions for the existing or new data, whereas `estimate_link` is more relevant in the context of visualisation and plotting. There are many control parameters that are not listed here but can be used, such as the arguments from `visualisation_matrix` (used when `data = "grid"`) and from `insight::get_predicted()` (the function to compute predictions used internally). For plotting, check the examples in `visualisation_recipe`.

Usage

```
estimate_expectation(
  model,
  data = "grid",
  ci = 0.95,
  keep_iterations = FALSE,
  ...
)

estimate_relation(
  model,
  data = "grid",
  ci = 0.95,
  keep_iterations = FALSE,
  ...
```

```
)

estimate_link(model, data = "grid", ci = 0.95, keep_iterations = FALSE, ...)

estimate_prediction(
  model,
  data = NULL,
  ci = 0.95,
  keep_iterations = FALSE,
  ...
)

estimate_response(model, data = NULL, ci = 0.95, keep_iterations = FALSE, ...)
```

Arguments

model	A statistical model.
data	A data frame with model's predictors to estimate the response. If NULL, the model's data is used. If "grid", the model matrix is obtained (through visualisation_matrix).
ci	The interval level (default 0.95, i.e., 95% CI).
keep_iterations	Only relevant for Bayesian models or simulated models. If TRUE, will keep all prediction iterations (draws). You can reshape them by running bayestestR::reshape_iterations().
...	You can add all the additional control arguments from visualisation_matrix (used when data = "grid") and insight::get_predicted().

Value

A dataframe of predicted values.

Examples

```
library(modelbased)

# Linear Models
model <- lm(mpg ~ wt, data = mtcars)

# Get predicted and prediction interval (see insight::get_predicted)
estimate_response(model)

# Get expected values with confidence interval
pred <- estimate_relation(model)
pred

# Visualisation (see visualisation_recipe())
plot(pred)

# Standardize predictions
pred <- estimate_relation(lm(mpg ~ wt + am, data = mtcars))
z <- effectsize::standardize(pred, include_response = FALSE)
```

```
z
effectsize::unstandardize(z, include_response = FALSE)

# Logistic Models
model <- glm(vs ~ wt, data = mtcars, family = "binomial")
estimate_response(model)
estimate_relation(model)

# Mixed models
if (require("lme4")) {
  model <- lmer(mpg ~ wt + (1 | gear), data = mtcars)
  estimate_response(model)
  estimate_relation(model)
}

# Bayesian models

if (require("rstanarm")) {
  model <- rstanarm::stan_glm(mpg ~ wt, data = mtcars, refresh = 0, iter = 200)
  estimate_response(model)
  estimate_relation(model)
}
```

`estimate_grouplevel` *Group-specific parameters of mixed models random effects*

Description

Extract random parameters of each individual group in the context of mixed models. Can be reshaped to be of the same dimensions as the original data, which can be useful to add the random effects to the original data.

Usage

```
estimate_grouplevel(model, type = "random", ...)

reshape_grouplevel(x, indices = "all", ...)
```

Arguments

`model`	A mixed model with random effects.
`type`	If `"random"` (default), the coefficients are the ones estimated natively by the model (as they are returned by, for instance, `lme4::ranef()`). They correspond to the deviation of each individual group from their fixed effect. As such, a coefficient close to 0 means that the participants' effect is the same as the population-level effect (in other words, it is "in the norm"). If "total", it will return the sum of the random effect and its corresponding fixed effects. These are known as

	BLUPs (Best Linear Unbiased Predictions). This argument can be used to reproduce the results given by `lme4::ranef()` and `coef()` (see `?coef.merMod`). Note that BLUPs currently don't have uncertainty indices (such as SE and CI), as these are not computable.
`...`	Other arguments passed to or from other methods.
`x`	The output of `estimate_grouplevel()`.
`indices`	A list containing the indices to extract.

Examples

```
if (require("lme4") && require("see")) {

  # Random intercept ------------
  model <- lmer(mpg ~ hp + (1 | carb), data = mtcars)
  random <- estimate_grouplevel(model)
  random

  # Visualize random effects
  plot(random)

  # Show group-specific effects
  estimate_grouplevel(model, deviation = FALSE)

  # Reshape to wide data so that it matches the original dataframe...
  reshaped <- reshape_grouplevel(random, indices = c("Coefficient", "SE"))

  # ... and can be easily combined
  alldata <- cbind(mtcars, reshaped)

  # Use summary() to remove duplicated rows
  summary(reshaped)
}
```

`estimate_means` *Estimate Marginal Means (Model-based average at each factor level)*

Description

Estimate average value of response variable at each factor levels. For plotting, check the examples in `visualisation_recipe`. See also other related functions such as `estimate_contrasts` and `estimate_slopes`.

Usage

```
estimate_means(
  model,
  levels = NULL,
  fixed = NULL,
```

```
  modulate = NULL,
  transform = "response",
  ci = 0.95,
  ...
)
```

Arguments

model	A statistical model.
levels	A character vector or formula specifying the names of the predicting factors over which to estimate means or contrasts.
fixed	A character vector indicating the names of the predictors to be "fixed" (i.e., maintained), so that the estimation is made at these values.
modulate	A character vector indicating the names of a numeric variable along which the means or the contrasts will be estimated. Other arguments from visualisation_matrix, such as length to adjust the number of data points.
transform	Is passed to the type argument in emmeans::emmeans(). See this vignette. Can be "none" (default for contrasts), "response" (default for means), "mu", "unlink", "log". "none" will leave the values on scale of the linear predictors. "response" will transform them on scale of the response variable. Thus for a logistic model, "none" will give estimations expressed in log-odds (probabilities on logit scale) and "response" in terms of probabilities.
ci	Uncertainty Interval (CI) level. Default to 95% (0.95).
...	Other arguments passed for instance to visualisation_matrix.

Value

A dataframe of estimated marginal means.

Examples

```
library(modelbased)

# Frequentist models
# -------------------
model <- lm(Petal.Length ~ Sepal.Width * Species, data = iris)

estimate_means(model)
estimate_means(model, fixed = "Sepal.Width")
estimate_means(model, levels = c("Species", "Sepal.Width"), length = 2)
estimate_means(model, levels = "Species=c('versicolor', 'setosa')")
estimate_means(model, levels = "Sepal.Width=c(2, 4)")
estimate_means(model, levels = c("Species", "Sepal.Width=0"))
estimate_means(model, modulate = "Sepal.Width", length = 5)
estimate_means(model, modulate = "Sepal.Width=c(2, 4)")

# Methods that can be applied to it:
means <- estimate_means(model, fixed = "Sepal.Width")
plot(means) # which runs visualisation_recipe()
```

```
effectsize::standardize(means)

if (require("lme4")) {
  data <- iris
  data$Petal.Length_factor <- ifelse(data$Petal.Length < 4.2, "A", "B")

 model <- lmer(Petal.Length ~ Sepal.Width + Species + (1 | Petal.Length_factor), data = data)
  estimate_means(model)
  estimate_means(model, modulate = "Sepal.Width", length = 3)
}

# Bayesian models
# -------------------
data <- mtcars
data$cyl <- as.factor(data$cyl)
data$am <- as.factor(data$am)

if (require("rstanarm")) {
  model <- stan_glm(mpg ~ cyl * am, data = data, refresh = 0)
  estimate_means(model)

  model <- stan_glm(mpg ~ cyl * wt, data = data, refresh = 0)
  estimate_means(model)
  estimate_means(model, modulate = "wt")
  estimate_means(model, fixed = "wt")
}


## Not run:
if (require("brms")) {
  model <- brm(mpg ~ cyl * am, data = data, refresh = 0)
  estimate_means(model)
}

## End(Not run)
```

`estimate_slopes` *Estimate Marginal Effects*

Description

Estimate the slopes (i.e., the coefficient) of a predictor over different factor levels. See also other related functions such as `estimate_contrasts` and `estimate_means`.

Usage

```
estimate_slopes(model, trend = NULL, levels = NULL, ci = 0.95, ...)
```

Arguments

model	A statistical model.
trend	A character vector indicating the name of the numeric variable for which to compute the slopes.
levels	A character vector indicating the variables over which the slope will be computed. If NULL (default), it will select all the remaining predictors.
ci	Uncertainty Interval (CI) level. Default to 95% (0.95).
...	Other arguments passed for instance to visualisation_matrix.

Value

A data.frame.

Examples

```
model <- lm(Sepal.Width ~ Species * Petal.Length, data = iris)
slopes <- estimate_slopes(model, trend = "Petal.Length")
slopes
effectsize::standardize(slopes)
## Not run:
if (require("rstanarm")) {
  model <- stan_glm(Sepal.Width ~ Species * Petal.Length, data = iris, refresh = 0)
  estimate_slopes(model)
}

## End(Not run)
```

---

find_inversions *Find points of inversion*

Description

Find points of inversion of a curve.

Usage

```
find_inversions(x)
```

Arguments

x	A numeric vector.

Value

Vector of inversion points.

Examples

```
x <- sin(seq(0, 4 * pi, length.out = 100))
plot(x, type = "b")
find_inversions(x)
```

`model_emmeans` *Easy Creation of 'emmeans' Marginal Means Objects*

Description

The `model_emmeans` function is a wrapper to facilitate the usage of `emmeans::emmeans()`, providing a somewhat simpler and smart API to find the variables of interest.

Usage

```
model_emmeans(
  model,
  levels = NULL,
  fixed = NULL,
  modulate = NULL,
  transform = "response",
  ...
)
```

Arguments

`model`	A statistical model.
`levels`	A character vector or formula specifying the names of the predicting factors over which to estimate means or contrasts.
`fixed`	A character vector indicating the names of the predictors to be "fixed" (i.e., maintained), so that the estimation is made at these values.
`modulate`	A character vector indicating the names of a numeric variable along which the means or the contrasts will be estimated. Other arguments from `visualisation_matrix`, such as `length` to adjust the number of data points.
`transform`	Is passed to the `type` argument in `emmeans::emmeans()`. See this vignette. Can be `"none"` (default for contrasts), `"response"` (default for means), `"mu"`, `"unlink"`, `"log"`. `"none"` will leave the values on scale of the linear predictors. `"response"` will transform them on scale of the response variable. Thus for a logistic model, `"none"` will give estimations expressed in log-odds (probabilities on logit scale) and `"response"` in terms of probabilities.
`...`	Other arguments passed for instance to `visualisation_matrix`.

Value

An `emmeans` object.

Examples

```
library(modelbased)

model <- lm(Sepal.Length ~ Species + Petal.Width, data = iris)

# By default, 'levels' is set to "Species"
model_emmeans(model)

# One can estimate marginal means at several values of a 'modulate' variable
model_emmeans(model, modulate = "Petal.Width", length = 3)
```

`smoothing` *Smoothing a vector or a time series*

Description

Smoothing a vector or a time series. For data.frames, the function will smooth all numeric variables stratified by factor levels (i.e., will smooth within each factor level combination).

Usage

```
smoothing(x, method = "loess", strength = 0.25, ...)
```

Arguments

`x`	A numeric vector.
`method`	Can be "loess" (default) or "smooth". A loess smoothing can be slow.
`strength`	This argument only applies when `method = "loess"`. Degree of smoothing passed to `span` (see `loess`).
`...`	Arguments passed to or from other methods.

Value

A smoothed vector or data frame.

Examples

```
x <- sin(seq(0, 4 * pi, length.out = 100)) + rnorm(100, 0, 0.2)
plot(x, type = "l")
lines(smoothing(x, method = "smooth"), type = "l", col = "blue")
lines(smoothing(x, method = "loess"), type = "l", col = "red")

x <- sin(seq(0, 4 * pi, length.out = 10000)) + rnorm(10000, 0, 0.2)
plot(x, type = "l")
lines(smoothing(x, method = "smooth"), type = "l", col = "blue")
lines(smoothing(x, method = "loess"), type = "l", col = "red")
```

`visualisation_matrix` *Create a reference grid*

Description

Create a reference matrix, useful for visualisation, with evenly spread and combined values.

Usage

```
visualisation_matrix(x, ...)

## S3 method for class 'data.frame'
visualisation_matrix(
  x,
  target = "all",
  factors = "reference",
  numerics = "mean",
  preserve_range = FALSE,
  reference = x,
  ...
)

## S3 method for class 'numeric'
visualisation_matrix(x, length = 10, range = "range", ...)

## S3 method for class 'factor'
visualisation_matrix(x, ...)
```

Arguments

`x`	An object from which to construct the reference grid.
`...`	Arguments passed to or from other methods (for instance, `length` or `range` to control the spread of numeric variables.).
`target`	Can be "all" or list of characters indicating columns of interest. Can also contain assignments (e.g., `target = "Sepal.Length = 2"` or `target = c("Sepal.Length = 2","Species = 'setosa'")` - note the usage of single and double quotes to assign strings within strings). The remaining variables will be fixed.
`factors`	Type of summary for factors. Can be "reference" (set at the reference level), "mode" (set at the most common level) or "all" to keep all levels.
`numerics`	Type of summary for numeric values. Can be "all" (will duplicate the grid for all unique values), any function ("mean", "median", ...) or a value (e.g., `numerics = 0`).
`preserve_range`	In the case of combinations between numeric variables and factors, setting `preserve_range = TRUE` will drop the observations where the value of the numeric variable is

originally not present in the range of its factor level. This leads to an unbalanced grid. Also, if you want the minimum and the maximum to closely match the actual ranges, you should increase the `length` argument.

`reference` The reference vector from which to compute the mean and SD.

`length` Length of numeric target variables.

`range` Can be one of `c("range","iqr","ci","hdi","eti")`. If `"range"` (default), will use the min and max of the original vector as end-points. If any other interval, will spread within the range (the default CI width is 95% but this can be changed by setting something else, e.g., `ci = 0.90`). See `IQR` and `ci`.

Value

Reference grid data frame.

Examples

```
library(modelbased)

# Add one row to change the "mode" of Species
data <- rbind(iris, iris[149, ], make.row.names = FALSE)

# Single variable is of interest; all others are "fixed"
visualisation_matrix(data, target = "Sepal.Length")
visualisation_matrix(data, target = "Sepal.Length", length = 3)
visualisation_matrix(data, target = "Sepal.Length", range = "ci", ci = 0.90)
visualisation_matrix(data, target = "Sepal.Length", factors = "mode")

# Multiple variables are of interest, creating a combination
visualisation_matrix(data, target = c("Sepal.Length", "Species"), length = 3)
visualisation_matrix(data, target = c(1, 3), length = 3)
visualisation_matrix(data, target = c("Sepal.Length", "Species"), preserve_range = TRUE)
visualisation_matrix(data, target = c("Sepal.Length", "Species"), numerics = 0)
visualisation_matrix(data, target = c("Sepal.Length = 3", "Species"))
visualisation_matrix(data, target = c("Sepal.Length = c(3, 1)", "Species = 'setosa'"))

# Standardize
vizdata <- visualisation_matrix(data, target = "Sepal.Length")
effectsize::standardize(vizdata)
```

---

visualisation_recipe *Prepare objects for visualisation*

Description

This function prepares some objects for visualisation by returning a list of layers with data and geoms that can be easily plotted using `ggplot2`.

Usage

```
visualisation_recipe(x, ...)

## S3 method for class 'estimate_grouplevel'
visualisation_recipe(
  x,
  hline = NULL,
  pointrange = NULL,
  facet_wrap = NULL,
  labs = NULL,
  ...
)

## S3 method for class 'estimate_means'
visualisation_recipe(
  x,
  show_data = "jitter",
  point = NULL,
  jitter = point,
  boxplot = NULL,
  violin = NULL,
  line = NULL,
  pointrange = NULL,
  labs = NULL,
  ...
)

## S3 method for class 'estimate_predicted'
visualisation_recipe(
  x,
  show_data = "points",
  point = NULL,
  density_2d = NULL,
  line = NULL,
  ribbon = NULL,
  labs = NULL,
  ...
)
```

Arguments

x	An easystats object.
...	Other arguments passed to other functions.
show_data	Display the "raw" data as a background to the model-based estimation. Can be set to "none" to remove it. When input is the result of estimate_means, show_data can be "points" (the jittered observation points), "boxplot", "violin" a combination of them (see examples). When input is the result of estimate_response

or `estimate_expectation`, `show_data` can be "points" (the points of the original data corresponding to the x and y axes), "density_2d", "density_2d_filled", "density_2d_polygon" or "density_2d_raster".

`point`, `jitter`, `boxplot`, `violin`, `pointrange`, `density_2d`, `line`, `hline`, `ribbon`, `labs`, `facet_wrap` Additional aesthetics and parameters for the geoms (see customization example).

Examples

```
# ================================================
# estimate_grouplevel
# ================================================
if (require("see") && require("lme4")) {
  data <- lme4::sleepstudy
  data <- rbind(data, data)
  data$Newfactor <- rep(c("A", "B", "C", "D"))

  # 1 random intercept
  model <- lmer(Reaction ~ Days + (1 | Subject), data = data)
  x <- estimate_grouplevel(model)
  layers <- visualisation_recipe(x)
  layers
  plot(layers)

  # 2 random intercepts
  model <- lmer(Reaction ~ Days + (1 | Subject) + (1 | Newfactor), data = data)
  x <- estimate_grouplevel(model)
  plot(visualisation_recipe(x))


  model <- lmer(Reaction ~ Days + (1 + Days | Subject) + (1 | Newfactor), data = data)
  x <- estimate_grouplevel(model)
  plot(visualisation_recipe(x))
}
# ================================================
# estimate_means
# ================================================
if (require("ggplot2")) {

  # Simple Model ---------------
  x <- estimate_means(lm(Sepal.Width ~ Species, data = iris))
  layers <- visualisation_recipe(x)
  layers
  plot(layers)

  # Customize aesthetics
  layers <- visualisation_recipe(x,
    jitter = list(width = 0.03, color = "red"),
    line = list(linetype = "dashed")
  )
  plot(layers)
```

```
  # Customize raw data
  plot(visualisation_recipe(x, show_data = c("violin", "boxplot", "jitter")))

  # Two levels ---------------
  data <- mtcars
  data$cyl <- as.factor(data$cyl)
  data$new_factor <- as.factor(rep(c("A", "B"), length.out = nrow(mtcars)))

  model <- lm(mpg ~ new_factor * cyl * wt, data = data)
  x <- estimate_means(model, levels = c("new_factor", "cyl"))
  plot(visualisation_recipe(x))

  # Modulations --------------
  x <- estimate_means(model, levels = c("new_factor"), modulate = "wt")
  plot(visualisation_recipe(x))

  x <- estimate_means(model, levels = c("new_factor", "cyl"), modulate = "wt")
  plot(visualisation_recipe(x))

  #'   # GLMs ---------------------
  data <- data.frame(vs = mtcars$vs, cyl = as.factor(mtcars$cyl))
  x <- estimate_means(glm(vs ~ cyl, data = data, family = "binomial"))
  plot(visualisation_recipe(x))
}
# ============================================
# estimate_expectation, estimate_response, ...
# ============================================
if (require("ggplot2")) {

  # Simple Model ---------------
  x <- estimate_relation(lm(mpg ~ wt, data = mtcars))
  layers <- visualisation_recipe(x)
  layers
  plot(layers)

  # Customize aesthetics
  layers <- visualisation_recipe(x,
    point = list(color = "red", alpha = 0.6, size = 3),
    line = list(color = "blue", size = 4),
    ribbon = list(fill = "green", alpha = 0.7),
    labs = list(subtitle = "Oh yeah!")
  )
  layers
  plot(layers)

  # Customize raw data
  plot(visualisation_recipe(x, show_data = "none"))
  plot(visualisation_recipe(x, show_data = c("density_2d", "points")))
  plot(visualisation_recipe(x, show_data = "density_2d_filled"))
  plot(visualisation_recipe(x, show_data = "density_2d_polygon"))
  plot(visualisation_recipe(x, show_data = "density_2d_raster")) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_y_continuous(expand = c(0, 0))
```

```
  # 2-ways interaction ------------

  # Numeric * numeric
  x <- estimate_relation(lm(mpg ~ wt * qsec, data = mtcars))
  layers <- visualisation_recipe(x)
  plot(layers)

  # Factor * numeric
  x <- estimate_relation(lm(Sepal.Width ~ Species * Sepal.Length, data = iris))
  layers <- visualisation_recipe(x)
  plot(layers)


  # 3-ways interaction ------------
  data <- mtcars
  data$vs <- as.factor(data$vs)
  data$cyl <- as.factor(data$cyl)
  data$new_factor <- as.factor(rep(c("A", "B"), length.out = nrow(mtcars)))

  # Numeric * numeric * numeric
  x <- estimate_relation(lm(mpg ~ wt * qsec * hp, data = data))
  layers <- visualisation_recipe(x)
  plot(layers)

  # Numeric * numeric * factor
  x <- estimate_relation(lm(mpg ~ wt * am * vs, data = data))
  layers <- visualisation_recipe(x)
  plot(layers)

  # Numeric * factor * factor
  x <- estimate_relation(lm(mpg ~ wt * cyl * new_factor, data = data))
  layers <- visualisation_recipe(x)
  plot(layers)

  # GLMs ---------------------
  x <- estimate_relation(glm(vs ~ mpg, data = mtcars, family = "binomial"))
  plot(visualisation_recipe(x))
  plot(visualisation_recipe(x, show_data = "jitter", point = list(height = 0.03)))
}
```

`zero_crossings` *Find zero crossings of a vector*

Description

Find zero crossings of a vector, i.e., indices when the numeric variable crosses 0.

Usage

```
zero_crossings(x)
```

Arguments

x	A numeric vector.

Value

Vector of zero crossings.

See Also

Based on the `uniroot.all` function from the rootSolve package.

Examples

```
x <- sin(seq(0, 4 * pi, length.out = 100))
plot(x)
zero_crossings(x)
```

Index

bayestestR::reshape_iterations(), *6*

ci, *15*

describe_nonlinear, 2

estimate_contrasts, 3, *8*, *10*
estimate_expectation, 5
estimate_grouplevel, 7
estimate_link *(*estimate_expectation*)*, 5
estimate_means, *3*, 8, *10*
estimate_prediction
        *(*estimate_expectation*)*, 5
estimate_relation, *2*
estimate_relation
        *(*estimate_expectation*)*, 5
estimate_response
        *(*estimate_expectation*)*, 5
estimate_slopes, *3*, *8*, 10
estimate_smooth *(*describe_nonlinear*)*, 2

find_inversions, 11

insight::get_predicted(), *5*, *6*
IQR, *15*

loess, *13*

model_emmeans, 12

reshape_grouplevel
        *(*estimate_grouplevel*)*, 7

smoothing, 13

visualisation_matrix, *3–6*, *9*, *11*, *12*, 14
visualisation_recipe, *5*, *8*, 15

zero_crossings, 19