Package 'margins'

May 8, 2026

Type Package

Title Marginal Effects for Model Objects

Description An R port of the margins command from 'Stata', which can be used to calculate marginal (or partial) effects from model objects.

License MIT + file LICENSE

Version 0.3.28

URL https://github.com/bbolker/margins

BugReports https://github.com/bbolker/margins/issues

Imports utils, stats, prediction (>= 0.3.6), data.table, graphics, grDevices, MASS

Suggests methods, knitr, rmarkdown, testthat, ggplot2, gapminder, sandwich, stargazer, lme4

Enhances AER, betareg, nnet, ordinal, survey

ByteCompile true

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Thomas J. Leeper [aut] (ORCID: <https://orcid.org/0000-0003-4097-6326>), Jeffrey Arnold [ctb], Vincent Arel-Bundock [ctb], Jacob A. Long [ctb] (ORCID: <https://orcid.org/0000-0002-1582-6214>), Ben Bolker [ctb, cre] (ORCID: <https://orcid.org/0000-0002-2127-0443>)

Maintainer Ben Bolker <bolker@mcmaster.ca>

Repository CRAN

Date/Publication 2024-07-31 20:20:02 UTC

Contents

`cplot` *Conditional predicted value and average marginal effect plots for models*

Description

Draw one or more conditional effects plots reflecting predictions or marginal effects from a model, conditional on a covariate. Currently methods exist for "lm", "glm", "loess" class models.

Usage

```
cplot(object, ...)

## Default S3 method:
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = prediction::seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "prediction") paste0("Predicted value") else
    paste0("Marginal effect of ", dx),
  xlim = NULL,
  ylim = NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  se.type = c("shade", "lines", "none"),
  se.col = "black",
```

```
  se.fill = grDevices::gray(0.5, 0.5),
  se.lwd = lwd,
  se.lty = if (match.arg(se.type) == "lines") 1L else 0L,
  factor.lty = 0L,
  factor.pch = 19L,
  factor.col = se.col,
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = se.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)

## S3 method for class 'clm'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "classprediction", "stackedprediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "effect") paste0("Marginal effect of ", dx) else
    paste0("Predicted value"),
  xlim = NULL,
  ylim = if (match.arg(what) %in% c("prediction", "stackedprediction")) c(0, 1.04) else
    NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  factor.lty = 1L,
  factor.pch = 19L,
  factor.col = col,
```

```
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = factor.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)

## S3 method for class 'glm'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = prediction::seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "prediction") paste0("Predicted value") else
    paste0("Marginal effect of ", dx),
  xlim = NULL,
  ylim = NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  se.type = c("shade", "lines", "none"),
  se.col = "black",
  se.fill = grDevices::gray(0.5, 0.5),
  se.lwd = lwd,
  se.lty = if (match.arg(se.type) == "lines") 1L else 0L,
  factor.lty = 0L,
  factor.pch = 19L,
  factor.col = se.col,
  factor.fill = factor.col,
  factor.cex = 1L,
```

```
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = se.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)

## S3 method for class 'lm'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = prediction::seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "prediction") paste0("Predicted value") else
    paste0("Marginal effect of ", dx),
  xlim = NULL,
  ylim = NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  se.type = c("shade", "lines", "none"),
  se.col = "black",
  se.fill = grDevices::gray(0.5, 0.5),
  se.lwd = lwd,
  se.lty = if (match.arg(se.type) == "lines") 1L else 0L,
  factor.lty = 0L,
  factor.pch = 19L,
  factor.col = se.col,
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
```

```
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = se.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)

## S3 method for class 'loess'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = prediction::seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "prediction") paste0("Predicted value") else
    paste0("Marginal effect of ", dx),
  xlim = NULL,
  ylim = NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  se.type = c("shade", "lines", "none"),
  se.col = "black",
  se.fill = grDevices::gray(0.5, 0.5),
  se.lwd = lwd,
  se.lty = if (match.arg(se.type) == "lines") 1L else 0L,
  factor.lty = 0L,
  factor.pch = 19L,
  factor.col = se.col,
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
```

```
  scatter.pch = 19L,
  scatter.col = se.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)

## S3 method for class 'polr'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "classprediction", "stackedprediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "effect") paste0("Marginal effect of ", dx) else
    paste0("Predicted value"),
  xlim = NULL,
  ylim = if (match.arg(what) %in% c("prediction", "stackedprediction")) c(0, 1.04) else
    NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  factor.lty = 1L,
  factor.pch = 19L,
  factor.col = col,
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = factor.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
```

```
  rug.size = -0.02,
  ...
)

## S3 method for class 'multinom'
cplot(
  object,
  x = attributes(terms(object))[["term.labels"]][1L],
  dx = x,
  what = c("prediction", "classprediction", "stackedprediction", "effect"),
  data = prediction::find_data(object),
  type = c("response", "link"),
  vcov = stats::vcov(object),
  at,
  n = 25L,
  xvals = seq_range(data[[x]], n = n),
  level = 0.95,
  draw = TRUE,
  xlab = x,
  ylab = if (match.arg(what) == "effect") paste0("Marginal effect of ", dx) else
    paste0("Predicted value"),
  xlim = NULL,
  ylim = if (match.arg(what) %in% c("prediction", "stackedprediction")) c(0, 1.04) else
    NULL,
  lwd = 1L,
  col = "black",
  lty = 1L,
  factor.lty = 1L,
  factor.pch = 19L,
  factor.col = col,
  factor.fill = factor.col,
  factor.cex = 1L,
  xaxs = "i",
  yaxs = xaxs,
  las = 1L,
  scatter = FALSE,
  scatter.pch = 19L,
  scatter.col = factor.col,
  scatter.bg = scatter.col,
  scatter.cex = 0.5,
  rug = TRUE,
  rug.col = col,
  rug.size = -0.02,
  ...
)
```

Arguments

object	A model object.

...	Additional arguments passed to plot.
x	A character string specifying the name of variable to use as the x-axis dimension in the plot.
dx	If what = "effect", the variable whose conditional marginal effect should be displayed. By default it is x (so the plot displays the marginal effect of x across values of x); ignored otherwise. If dx is a factor with more than 2 levels, an error will be issued.
what	A character string specifying whether to draw a "prediction" (fitted values from the model, calculated using predict) or an "effect" (average marginal effect of dx conditional on x, using margins). Methods for classes other than "lm" or "glm" may provided additional options (e.g., cplot.polr() provides "stacked-prediction" and "class" alternatives).
data	A data frame to override the default value offered in object[["model"]].
type	A character string specifying whether to calculate predictions on the response scale (default) or link (only relevant for non-linear models).
vcov	A matrix containing the variance-covariance matrix for estimated model coefficients, or a function to perform the estimation with model as its only argument.
at	Currently ignored.
n	An integer specifying the number of points across x at which to calculate the predicted value or marginal effect, when x is numeric. Ignored otherwise.
xvals	A numeric vector of values at which to calculate predictions or marginal effects, if x is numeric. By default, it is calculated from the data using seq_range. If x is a factor, this is ignored, as is n.
level	The confidence level required (used to draw uncertainty bounds).
draw	A logical (default TRUE), specifying whether to draw the plot. If FALSE, the data used in drawing are returned as a list of data.frames. This might be useful if you want to plot using an alternative plotting package (e.g., ggplot2). Also, if set to value "add", then the resulting data is added to the existing plot.
xlab	A character string specifying the value of xlab in plot.
ylab	A character string specifying the value of ylab in plot.
xlim	A two-element numeric vector specifying the x-axis limits. Set automatically if missing.
ylim	A two-element numeric vector specifying the y-axis limits. Set automatically if missing.
lwd	An integer specifying the width of the prediction or marginal effect line. See lines. If x is a factor variable in the model, this is used to set the line width of the error bars.
col	A character string specifying the color of the prediction or marginal effect line. If x is a factor variable in the model, this is used to set the color of the error bars.
lty	An integer specifying the "line type" of the prediction or marginal effect line. See par. If x is a factor variable in the model, this is used to set the line type of the error bars.

se.type	A character string specifying whether to draw the confidence interval as "lines" (the default, using lines) or a "shade" (using polygon).
se.col	If se.type = "lines", a character string specifying the color of the confidence interval lines. If se.type = "shade", the color of the shaded region border.
se.fill	If se.type = "shade", the color of the shaded region. Ignored otherwise.
se.lwd	If se.type = "lines", the width of the confidence interval lines. See lines.
se.lty	If se.type = "lines", an integer specifying the "line type" of the confidence interval lines; if se.type = "shade", the line type of the shaded polygon border. See par.
factor.lty	If x is a factor variable in the model, this is used to set the line type of an optional line connecting predictions across factor levels. If factor.lty = 0L (the default), no line is drawn.. See par.
factor.pch	If x is a factor variable in the model, the shape to use when drawing points. See points.
factor.col	If x is a factor variable in the model, the color to use for the border of the points. See points.
factor.fill	If x is a factor variable in the model, the color to use for the fill of the points. See points.
factor.cex	If x is a factor variable in the model, the "expansion factor" to use for the point size. See points.
xaxs	A character string specifying xaxs. See par.
yaxs	A character string specifying xaxs. See par.
las	An integer string specifying las. See par.
scatter	A logical indicating whether to plot the observed data in data as a scatterplot.
scatter.pch	If scatter = TRUE, an integer specifying a shape to use for plotting the data. See points.
scatter.col	If scatter = TRUE, a character string specifying a color to use for plotting the data. See points.
scatter.bg	If scatter = TRUE, a character string specifying a color to use for plotting the data. See points.
scatter.cex	If scatter = TRUE, an integer specifying the size of the points. See points.
rug	A logical specifying whether to include an x-axis "rug" (see rug).
rug.col	A character string specifying col to rug.
rug.size	A numeric value specifying ticksize to rug.

Details

Note that when what = "prediction", the plots show predictions holding values of the data at their mean or mode, whereas when what = "effect" average marginal effects (i.e., at observed values) are shown.

When examining generalized linear models (e.g., logistic regression models), confidence intervals for predictions can fall outside of the response scale (again, for logistic regression this means confidence intervals can exceed the (0,1) bounds). This is consistent with the behavior of predict but

may not be desired. The examples (below) show ways of constraining confidence intervals to these bounds.

The overall aesthetic is somewhat similar to to the output produced by the `marginalModelPlot()` function in the **car** package.

Value

A tidy data frame containing the data used to draw the plot. Use `draw = FALSE` to simply generate the data structure for use elsewhere.

See Also

`plot.margins`, `persp.lm`

Examples

```
## Not run:
require('datasets')
# prediction from several angles
m <- lm(Sepal.Length ~ Sepal.Width, data = iris)
cplot(m)

# more complex model
m <- lm(Sepal.Length ~ Sepal.Width * Petal.Width * I(Petal.Width ^ 2),
        data = head(iris, 50))
## marginal effect of 'Petal.Width' across 'Petal.Width'
cplot(m, x = "Petal.Width", what = "effect", n = 10)

# factor independent variables
mtcars[["am"]] <- factor(mtcars[["am"]])
m <- lm(mpg ~ am * wt, data = mtcars)
## predicted values for each factor level
cplot(m, x = "am")
## marginal effect of each factor level across numeric variable
cplot(m, x = "wt", dx = "am", what = "effect")

# marginal effect of 'Petal.Width' across 'Sepal.Width'
## without drawing the plot
## this might be useful for using, e.g., ggplot2 for plotting
tmp <- cplot(m, x = "Sepal.Width", dx = "Petal.Width",
             what = "effect", n = 10, draw = FALSE)
if (require("ggplot2")) {
  # use ggplot2 instead of base graphics
  ggplot(tmp, aes(x = Petal.Width, y = "effect")) +
         geom_line(lwd = 2) +
         geom_line(aes(y = effect + 1.96*se.effect)) +
         geom_line(aes(y = effect - 1.96*se.effect))
}

# a non-linear model
m <- glm(am ~ wt*drat, data = mtcars, family = binomial)
cplot(m, x = "wt") # prediction (response scale)
```

```
cplot(m, x = "wt") # prediction (link scale)
if (require("ggplot2")) {
  # prediction (response scale, constrained to [0,1])
  cplotdat <- cplot(m, x = "wt", type = "link", draw = FALSE)
  ggplot(cplotdat, aes(x = xvals, y = plogis(yvals))) +
         geom_line(lwd = 1.5) +
         geom_line(aes(y = plogis(upper))) +
         geom_line(aes(y = plotis(lower)))
}

# effects on linear predictor and outcome
cplot(m, x = "drat", dx = "wt", what = "effect", type = "link")
cplot(m, x = "drat", dx = "wt", what = "effect", type = "response")

# plot conditional predictions across a third factor
local({
  iris$long <- rbinom(nrow(iris), 1, 0.6)
  x <- glm(long ~ Sepal.Width*Species, data = iris)
  cplot(x, x = "Sepal.Width", data = iris[iris$Species == "setosa", ],
        ylim = c(0,1), col = "red", se.fill = rgb(1,0,0,.5), xlim = c(2,4.5))
  cplot(x, x = "Sepal.Width", data = iris[iris$Species == "versicolor", ],
        draw = "add", col = "blue", se.fill = rgb(0,1,0,.5))
  cplot(x, x = "Sepal.Width", data = iris[iris$Species == "virginica", ],
        draw = "add", col = "green", se.fill = rgb(0,0,1,.5))
})

# ordinal outcome
if (require("MASS")) {
  # x is a factor variable
  house.plr <- polr(Sat ~ Infl + Type + Cont, weights = Freq,
                    data = housing)
  ## predicted probabilities
  cplot(house.plr)
  ## cumulative predicted probabilities
  cplot(house.plr, what = "stacked")
  ## ggplot2 example
  if (require("ggplot2")) {
    ggplot(cplot(house.plr), aes(x = xvals, y = yvals, group = level)) +
      geom_line(aes(color = level))
  }

  # x is continuous
  cyl.plr <- polr(factor(cyl) ~ wt, data = mtcars)
  cplot(cyl.plr, col = c("red", "purple", "blue"), what = "stacked")
  cplot(cyl.plr, what = "class")
}


## End(Not run)
```

---

dydx *Marginal Effect of a Given Variable*

Description

Differentiate an Estimated Model Function with Respect to One Variable, or calculate a discrete difference ("first difference") as appropriate.

Usage

```
dydx(data, model, variable, ...)

## Default S3 method:
dydx(
  data,
  model,
  variable,
  type = c("response", "link"),
  change = c("dydx", "minmax", "iqr", "sd"),
  eps = 1e-07,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'factor'
dydx(
  data,
  model,
  variable,
  type = c("response", "link"),
  fwrap = FALSE,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'ordered'
dydx(
  data,
  model,
  variable,
  type = c("response", "link"),
  fwrap = FALSE,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'logical'
dydx(
  data,
  model,
  variable,
  type = c("response", "link"),
```

```
  as.data.frame = TRUE,
  ...
)
```

Arguments

`data`	The dataset on which to to calculate $\hat{y}$.
`model`	The model object to pass to `prediction`.
`variable`	A character string specifying the variable to calculate the derivative or discrete change for.
`...`	Ignored.
`type`	The type of prediction. Default is "response".
`change`	For numeric variables, a character string specifying the type of change to express. The default is the numerical approximation of the derivative. Alternative values are occasionally desired quantities: "minmax" (the discrete change moving from `min(x)` to `max(x)`), "iqr" (the move from the 1st quartile to 3rd quartile of `x`), or "sd" (the change from `mean(x) - sd(x)` to `mean(x) + sd(x)`), or a two-element numeric vector expressing values of the variable to calculate the prediction for (and difference the associated predictions).
`eps`	If `change == "dydx"` (the default), the value of the step ϵ to use in calculation of the numerical derivative for numeric variables.
`as.data.frame`	A logical indicating whether to return a data frame (the default) or a matrix.
`fwrap`	A logical specifying how to name factor columns in the response.

Details

These functions provide a simple interface to the calculation of marginal effects for specific variables used in a model, and are the workhorse functions called internally by `marginal_effects`.

`dydx` is an S3 generic with classes implemented for specific variable types. S3 method dispatch, somewhat atypically, is based upon the class of `data[[variable]]`.

For numeric (and integer) variables, the method calculates an instantaneous marginal effect using a simple "central difference" numerical differentiation:

$$\frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{dh}$$

, where ($h = \max(|x|, 1)\sqrt{\epsilon}$ and the value of ϵ is given by argument `eps`. This procedure is subject to change in the future.

For factor variables (or character variables, which are implicitly coerced to factors by modelling functions), discrete first-differences in predicted outcomes are reported instead (i.e., change in predicted outcome when factor is set to a given level minus the predicted outcome when the factor is set to its baseline level). These are sometimes called "partial effects". If you want to use numerical differentiation for factor variables (which you probably do not want to do), enter them into the original modelling function as numeric values rather than factors.

For ordered factor variables, the same approach as factors is used. This may contradict the output of modelling function summaries, which rely on `options("contrasts")` to determine the contrasts

to use (the default being contr.poly rather than contr.treatment, the latter being used normally for unordered factors).

For logical variables, the same approach as factors is used, but always moving from FALSE to TRUE.

Value

A data frame, typically with one column unless the variable is a factor with more than two levels. The names of the marginal effect columns begin with "dydx_" to distinguish them from the substantive variables of the same names.

References

Miranda, Mario J. and Paul L. Fackler. 2002. *Applied Computational Economics and Finance*. p. 103.

Greene, William H. 2012. *Econometric Analysis*. 7th edition. pp. 733–741.

Cameron, A. Colin and Pravin K. Trivedi. 2010. *Microeconometric Using Stata*. Revised edition. pp. 106–108, 343–356, 476–478.

See Also

marginal_effects, margins

Examples

```
require("datasets")
x <- lm(mpg ~ cyl * hp + wt, data = head(mtcars))
# marginal effect (numerical derivative)
dydx(head(mtcars), x, "hp")

# other discrete differences
## change from min(mtcars$hp) to max(mtcars$hp)
dydx(head(mtcars), x, "hp", change = "minmax")
## change from 1st quartile to 3rd quartile
dydx(head(mtcars), x, "hp", change = "iqr")
## change from mean(mtcars$hp) +/- sd(mtcars$hp)
dydx(head(mtcars), x, "hp", change = "sd")
## change between arbitrary values of mtcars$hp
dydx(head(mtcars), x, "hp", change = c(75,150))

# factor variables
mtcars[["cyl"]] <- factor(mtcars$cyl)
x <- lm(mpg ~ cyl, data = head(mtcars))
dydx(head(mtcars), x, "cyl")
```

image.lm *Perspective and heatmap/contour plots for models*

Description

Draw one or more perspectives plots reflecting predictions or marginal effects from a model, or the same using a flat heatmap or "filled contour" (image) representation. Currently methods exist for "lm", "glm", and "loess" models.

Usage

```
## S3 method for class 'lm'
image(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  nz = 20,
  xlab = xvar,
  ylab = yvar,
  xaxs = "i",
  yaxs = xaxs,
  bty = "o",
  col = gray(seq(0.05, 0.95, length.out = nz), alpha = 0.75),
  contour = TRUE,
  contour.labels = NULL,
  contour.drawlabels = TRUE,
  contour.cex = 0.6,
  contour.col = "black",
  contour.lty = 1,
  contour.lwd = 1,
  ...
)

## S3 method for class 'glm'
image(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
```

```
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  nz = 20,
  xlab = xvar,
  ylab = yvar,
  xaxs = "i",
  yaxs = xaxs,
  bty = "o",
  col = gray(seq(0.05, 0.95, length.out = nz), alpha = 0.75),
  contour = TRUE,
  contour.labels = NULL,
  contour.drawlabels = TRUE,
  contour.cex = 0.6,
  contour.col = "black",
  contour.lty = 1,
  contour.lwd = 1,
  ...
)

## S3 method for class 'loess'
image(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  nz = 20,
  xlab = xvar,
  ylab = yvar,
  xaxs = "i",
  yaxs = xaxs,
  bty = "o",
  col = gray(seq(0.05, 0.95, length.out = nz), alpha = 0.75),
  contour = TRUE,
  contour.labels = NULL,
  contour.drawlabels = TRUE,
  contour.cex = 0.6,
  contour.col = "black",
  contour.lty = 1,
  contour.lwd = 1,
  ...
)
```

```
## S3 method for class 'lm'
persp(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  theta = 45,
  phi = 10,
  shade = 0.75,
  xlab = xvar,
  ylab = yvar,
  zlab = if (match.arg(what) == "prediction") "Predicted value" else
    paste0("Marginal effect of ", dx),
  ticktype = c("detailed", "simple"),
  ...
)

## S3 method for class 'glm'
persp(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  theta = 45,
  phi = 10,
  shade = 0.75,
  xlab = xvar,
  ylab = yvar,
  zlab = if (match.arg(what) == "prediction") "Predicted value" else
    paste0("Marginal effect of ", dx),
  ticktype = c("detailed", "simple"),
  ...
)

## S3 method for class 'loess'
persp(
  x,
  xvar = attributes(terms(x))[["term.labels"]][1],
```

```
  yvar = attributes(terms(x))[["term.labels"]][2],
  dx = xvar,
  what = c("prediction", "effect"),
  type = c("response", "link"),
  vcov = stats::vcov(x),
  nx = 25L,
  ny = nx,
  theta = 45,
  phi = 10,
  shade = 0.75,
  xlab = xvar,
  ylab = yvar,
  zlab = if (match.arg(what) == "prediction") "Predicted value" else
    paste0("Marginal effect of ", dx),
  ticktype = c("detailed", "simple"),
  ...
)
```

Arguments

x	A model object.
xvar	A character string specifying the name of variable to use as the 'x' dimension in the plot. See persp for details.
yvar	A character string specifying the name of variable to use as the 'y' dimension in the plot. See persp for details.
dx	A character string specifying the name of the variable for which the conditional average marginal effect is desired when what = "effect". By default this is xvar.
what	A character string specifying whether to draw "prediction" (fitted values from the model, calculated using predict) or "effect" (marginal effect of dx, using margins).
type	A character string specifying whether to calculate predictions on the response scale (default) or link (only relevant for non-linear models).
vcov	A matrix containing the variance-covariance matrix for estimated model coefficients, or a function to perform the estimation with model as its only argument.
nx	An integer specifying the number of points across x at which to calculate the predicted value or marginal effect.
ny	An integer specifying the number of points across y at which to calculate the predicted value or marginal effect.
nz	An integer specifying, for image, the number of breakpoints to use when coloring the plot.
xlab	A character string specifying the value of xlab in persp or image.
ylab	A character string specifying the value of ylab in persp or image.
xaxs	A character string specifying the x-axis type (see par).
yaxs	A character string specifying the y-axis type (see par).

bty	A character string specifying the box type (see par).
col	A character vector specifying colors to use when coloring the contour plot.
contour	For image, a logical specifying whether to overlay contour lines onto the heatmap using contour.
contour.labels	For image, if contour = TRUE a logical specifying whether to overlay contour lines onto the heatmap.
contour.drawlabels	For image, if contour = TRUE a logical specifying whether to overlay contour lines onto the heatmap.
contour.cex	For image, if contour = TRUE and contour.drawlabels = TRUE a numeric specifying the label size for contour line labels (see par).
contour.col	For image, if contour = TRUE a character string specifying a color for contour lines.
contour.lty	For image, if contour = TRUE an integer specifying a line type for contour lines (see par).
contour.lwd	For image, if contour = TRUE an integer specifying a line width for contour lines (see par).
...	Additional arguments passed to persp or image.
theta	For persp, an integer vector specifying the value of theta in persp. If length greater than 1, multiple subplots are drawn with different rotations.
phi	For persp, an integer vector specifying the value of phi in persp. If length greater than 1, multiple subplots are drawn with different rotations.
shade	For persp, an integer vector specifying the value of shade in persp.
zlab	A character string specifying the value of zlab (vertical axis label) in persp.
ticktype	A character string specifying one of: "detailed" (the default) or "simple". See persp.

See Also

plot.margins, cplot

Examples

```
## Not run:
  require('datasets')
  # prediction from several angles
  m <- lm(mpg ~ wt*drat, data = mtcars)
  persp(m, theta = c(45, 135, 225, 315))

  # flat/heatmap representation
  image(m)

  # marginal effect of 'drat' across drat and wt
  m <- lm(mpg ~ wt*drat*I(drat^2), data = mtcars)
  persp(m, xvar = "drat", yvar = "wt", what = "effect",
        nx = 10, ny = 10, ticktype = "detailed")
```

```
  # a non-linear model
  m <- glm(am ~ wt*drat, data = mtcars, family = binomial)
  persp(m, theta = c(30, 60)) # prediction
  # flat/heatmap representation
  image(m)

  # effects on linear predictor and outcome
  persp(m, xvar = "drat", yvar = "wt", what = "effect", type = "link")
  persp(m, xvar = "drat", yvar = "wt", what = "effect", type = "response")

## End(Not run)
```

`marginal_effects` *Differentiate a Model Object with Respect to All (or Specified) Variables*

Description

Extract marginal effects from a model object, conditional on data, using dydx.

Usage

```
marginal_effects(model, data, variables = NULL, ...)

## S3 method for class 'margins'
marginal_effects(model, data, variables = NULL, ...)

## S3 method for class 'clm'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = NULL,
  eps = 1e-07,
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)

## Default S3 method:
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
```

```
  as.data.frame = TRUE,
  varslist = NULL,
  ...
)

## S3 method for class 'glm'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'lm'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'loess'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
  as.data.frame = TRUE,
  varslist = NULL,
  ...
)

## S3 method for class 'merMod'
marginal_effects(
  model,
  data = find_data(model),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
```

```
  as.data.frame = TRUE,
  varslist = NULL,
  ...
)

## S3 method for class 'lmerMod'
marginal_effects(
  model,
  data = find_data(model),
  variables = NULL,
  type = c("response", "link"),
  eps = 1e-07,
  as.data.frame = TRUE,
  varslist = NULL,
  ...
)

## S3 method for class 'multinom'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = NULL,
  eps = 1e-07,
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'nnet'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = NULL,
  eps = 1e-07,
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)

## S3 method for class 'polr'
marginal_effects(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  type = NULL,
  eps = 1e-07,
```

```
  varslist = NULL,
  as.data.frame = TRUE,
  ...
)
```

Arguments

model	A model object, perhaps returned by lm or glm
data	A data.frame over which to calculate marginal effects. This is optional, but may be required when the underlying modelling function sets model = FALSE.
variables	A character vector with the names of variables for which to compute the marginal effects. The default (NULL) returns marginal effects for all variables.
...	Arguments passed to methods, and onward to dydx methods and possibly further to prediction methods. This can be useful, for example, for setting type (predicted value type), eps (precision), or category (category for multi-category outcome models), etc.
type	A character string indicating the type of marginal effects to estimate. Mostly relevant for non-linear models, where the reasonable options are "response" (the default) or "link" (i.e., on the scale of the linear predictor in a GLM).
eps	A numeric value specifying the "step" to use when calculating numerical derivatives. By default this is the smallest floating point value that can be represented on the present architecture.
varslist	A list structure used internally by margins. Users should not set this.
as.data.frame	A logical indicating whether to return a data frame (the default) or a matrix.

Details

Users likely want to use the fully featured margins function rather than marginal_effects, which merely performs estimation of the marginal effects but simply returns a data frame. margins, by contrast, does some convenient packaging around these results and supports additional functionality, like variance estimation and counterfactual estimation procedures. The methods for this function provide lower-level functionality that extracts unit-specific marginal effects from an estimated model with respect to *all* variables specified in data (or the subset specified in variables) and returns a data frame. See dydx for computational details. Note that for factor and logical class variables, discrete changes in the outcome are reported rather than instantaneous marginal effects.

Methods are currently implemented for the following object classes:

- "betareg", see betareg
- "glm", see glm, glm.nb
- "ivreg", see ivreg
- "lm", see lm
- "loess", see loess
- "merMod", see lmer, glmer
- "multinom", see multinom
- "nnet", see nnet

- "polr", see polr
- "svyglm", see svyglm

A method is also provided for the object classes "margins" to return a simplified data frame from complete "margins" objects.

Value

An data frame with number of rows equal to nrow(data), where each row is an observation and each column is the marginal effect of a variable used in the model formula.

See Also

dydx, margins

Examples

```
require("datasets")
x <- lm(mpg ~ cyl * hp + wt, data = mtcars)
marginal_effects(x)

# factor variables report discrete differences
x <- lm(mpg ~ factor(cyl) * factor(am), data = mtcars)
marginal_effects(x)

# get just marginal effects from "margins" object
require('datasets')
m <- margins(lm(mpg ~ hp, data = mtcars[1:10,]))
marginal_effects(m)
marginal_effects(m)

# multi-category outcome
if (requireNamespace("nnet")) {
  data("iris3", package = "datasets")
  ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                    species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))
  m <- nnet::nnet(species ~ ., data = ird, size = 2, rang = 0.1,
                  decay = 5e-4, maxit = 200, trace = FALSE)
  marginal_effects(m) # default
  marginal_effects(m, category = "v") # explicit category
}
```

---

margins *Marginal Effects Estimation*

Description

This package is an R port of Stata's 'margins' command, implemented as an S3 generic margins() for model objects, like those of class "lm" and "glm". margins() is an S3 generic function for building a "margins" object from a model object. Methods are currently implemented for several model classes (see Details, below).

margins provides "marginal effects" summaries of models. Marginal effects are partial derivatives of the regression equation with respect to each variable in the model for each unit in the data; average marginal effects are simply the mean of these unit-specific partial derivatives over some sample. In ordinary least squares regression with no interactions or higher-order term, the estimated slope coefficients are marginal effects. In other cases and for generalized linear models, the coefficients are not marginal effects at least not on the scale of the response variable. margins therefore provides ways of calculating the marginal effects of variables to make these models more interpretable.

The package also provides a low-level function, marginal_effects, to estimate those quantities and return a data frame of unit-specific effects and another even lower-level function, dydx, to provide variable-specific derivatives from models. Some of the underlying architecture for the package is provided by the low-level function prediction, which provides a consistent data frame interface to predict for a large number of model types. If a prediction method exists for a model class, margin should work for the model class but only those classes listed here have been tested and specifically supported.

Usage

```
margins(model, ...)

## S3 method for class 'betareg'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model, phi = FALSE),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'clm'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vce = "none",
  eps = 1e-07,
```

```
  ...
)

## Default S3 method:
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'glm'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'lm'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)
```

```
## S3 method for class 'loess'
margins(
  model,
  data,
  variables = NULL,
  at = NULL,
  vce = "none",
  eps = 1e-07,
  ...
)

## S3 method for class 'merMod'
margins(
  model,
  data = find_data(model),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'lmerMod'
margins(
  model,
  data = find_data(model),
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'multinom'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = NULL,
```

```
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

## S3 method for class 'nnet'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  vce = "none",
  eps = 1e-07,
  ...
)

## S3 method for class 'polr'
margins(
  model,
  data = find_data(model, parent.frame()),
  variables = NULL,
  at = NULL,
  type = NULL,
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
  eps = 1e-07,
  ...
)

margins_summary(model, ..., level = 0.95, by_factor = TRUE)

## S3 method for class 'svyglm'
margins(
  model,
  data = find_data(model, parent.frame()),
  design,
  variables = NULL,
  at = NULL,
  type = c("response", "link"),
  vcov = stats::vcov(model),
  vce = c("delta", "simulation", "bootstrap", "none"),
  iterations = 50L,
  unit_ses = FALSE,
```

```
  eps = 1e-07,
  ...
)
```

Arguments

model	A model object. See Details for supported model classes.
...	Arguments passed to methods, and onward to dydx methods and possibly further to prediction methods. This can be useful, for example, for setting type (predicted value type), eps (precision), or category (category for multi-category outcome models), etc.
data	A data frame containing the data at which to evaluate the marginal effects, as in predict. This is optional, but may be required when the underlying modelling function sets model = FALSE.
variables	A character vector with the names of variables for which to compute the marginal effects. The default (NULL) returns marginal effects for all variables.
at	A list of one or more named vectors, specifically values at which to calculate the marginal effects. This is an analogue of Stata's , at() option. The specified values are fully combined (i.e., a cartesian product) to find AMEs for all combinations of specified variable values. Rather than a list, this can also be a data frame of combination levels if only a subset of combinations are desired. These are used to modify the value of data when calculating AMEs across specified values (see build_datalist for details on use). Note: This does not calculate AMEs for *subgroups* but rather for counterfactual datasets where all observaations take the specified values; to obtain subgroup effects, subset data directly.
type	A character string indicating the type of marginal effects to estimate. Mostly relevant for non-linear models, where the reasonable options are "response" (the default) or "link" (i.e., on the scale of the linear predictor in a GLM).
vcov	A matrix containing the variance-covariance matrix for estimated model coefficients, or a function to perform the estimation with model as its only argument.
vce	A character string indicating the type of estimation procedure to use for estimating variances. The default ("delta") uses the delta method. Alternatives are "bootstrap", which uses bootstrap estimation, or "simulation", which averages across simulations drawn from the joint sampling distribution of model coefficients. The latter two are extremely time intensive.
iterations	If vce = "bootstrap", the number of bootstrap iterations. If vce = "simulation", the number of simulated effects to draw. Ignored otherwise.
unit_ses	If vce = "delta", a logical specifying whether to calculate and return unit-specific marginal effect variances. This calculation is time consuming and the information is often not needed, so this is set to FALSE by default.
eps	A numeric value specifying the "step" to use when calculating numerical derivatives.
level	A numeric value specifying the confidence level for calculating p-values and confidence intervals.
by_factor	A logical specifying whether to order the output by factor (the default, TRUE).

`design` Only for models estimated using `svyglm`, the "survey.design" object used to estimate the model. This is required.

Details

Methods for this generic return a "margins" object, which is a data frame consisting of the original data, predicted values and standard errors thereof, estimated marginal effects from the model `model` (for all variables used in the model, or the subset specified by `variables`), along with attributes describing various features of the marginal effects estimates.

The default print method is concise; a more useful `summary` method provides additional details.

`margins_summary` is sugar that provides a more convenient way of obtaining the nested call: `summary(margins(...))`.

Methods are currently implemented for the following object classes:

- "betareg", see `betareg`
- "glm", see `glm`, `glm.nb`
- "ivreg", see `ivreg`
- "lm", see `lm`
- "loess", see `loess`
- "merMod", see `lmer`, `glmer`
- "nnet", see `nnet`
- "polr", see `polr`
- "svyglm", see `svyglm`

The `margins` methods simply construct a list of data frames based upon the values of `at` (using `build_datalist`), calculate marginal effects for each data frame (via `marginal_effects` and, in turn, `dydx` and `prediction`), stacks the results together, and provides variance estimates. Alternatively, you can use `marginal_effects` directly to only retrieve a data frame of marginal effects without constructing a "margins" object or variance estimates. That can be efficient for plotting, etc., given the time-consuming nature of variance estimation.

See `dydx` for details on estimation of marginal effects.

The choice of `vce` may be important. The default variance-covariance estimation procedure (`vce = "delta"`) uses the delta method to estimate marginal effect variances. This is the fastest method. When `vce = "simulation"`, coefficient estimates are repeatedly drawn from the asymptotic (multivariate normal) distribution of the model coefficients and each draw is used to estimate marginal effects, with the variance based upon the dispersion of those simulated effects. The number of iterations used is given by `iterations`. For `vce = "bootstrap"`, the bootstrap is used to repeatedly subsample `data` and the variance of marginal effects is estimated from the variance of the bootstrap distribution. This method is markedly slower than the other two procedures. Again, `iterations` regulates the number of bootstrap subsamples to draw. Some model classes (notably "loess") fix `vce ="none"`.

Value

A data frame of class "margins" containing the contents of `data`, predicted values from `model` for `data`, the standard errors of the predictions, and any estimated marginal effects. If `at = NULL` (the default), then the data frame will have a number of rows equal to `nrow(data)`. Otherwise, the number of rows will be a multiple thereof based upon the number of combinations of values specified in `at`. Columns containing marginal effects are distinguished by their name (prefixed by `dydx_`). These columns can be extracted from a "margins" object using, for example, `marginal_effects(margins(model))`. Columns prefixed by `Var_` specify the variances of the *average* marginal effects, whereas (optional) columns prefixed by `SE_` contain observation-specific standard errors. A special column, `_at_number`, specifies which `at` combination a given row corresponds to; the data frame carries an attribute "at" that specifies which combination of values this index represents. The `summary.margins()` method provides for pretty printing of the results, particularly in cases where `at` is specified. A variance-covariance matrix for the average marginal effects is returned as an attribute (though behavior when `at` is non-NULL is unspecified).

Author(s)

Thomas J. Leeper

References

Greene, W.H. 2012. Econometric Analysis, 7th Ed. Boston: Pearson.

Stata manual: `margins`. Retrieved 2014-12-15 from https://www.stata.com/manuals13/rmargins.pdf.

See Also

`marginal_effects`, `dydx`, `prediction`

Examples

```
# basic example using linear model
require("datasets")
x <- lm(mpg ~ cyl * hp + wt, data = head(mtcars))
margins(x)

# obtain unit-specific standard errors
## Not run:
  margins(x, unit_ses = TRUE)

## End(Not run)

# use of 'variables' argument to estimate only some MEs
summary(margins(x, variables = "hp"))

# use of 'at' argument
## modifying original data values
margins(x, at = list(hp = 150))
## AMEs at various data values
margins(x, at = list(hp = c(95, 150), cyl = c(4,6)))
```

```
# use of 'data' argument to obtain AMEs for a subset of data
margins(x, data = mtcars[mtcars[["cyl"]] == 4,])
margins(x, data = mtcars[mtcars[["cyl"]] == 6,])

# return discrete differences for continuous terms
## passes 'change' through '...' to dydx()
margins(x, change = "sd")

# summary() method
summary(margins(x, at = list(hp = c(95, 150))))
margins_summary(x, at = list(hp = c(95, 150)))
## control row order of summary() output
summary(margins(x, at = list(hp = c(95, 150))), by_factor = FALSE)

# alternative 'vce' estimation
## Not run:
  # bootstrap
  margins(x, vce = "bootstrap", iterations = 100L)
  # simulation (ala Clarify/Zelig)
  margins(x, vce = "simulation", iterations = 100L)

## End(Not run)

# specifying a custom `vcov` argument
if (require("sandwich")) {
  x2 <- lm(Sepal.Length ~ Sepal.Width, data = head(iris))
  summary(margins(x2))
  ## heteroskedasticity-consistent covariance matrix
  summary(margins(x2, vcov = vcovHC(x2)))
}

# generalized linear model
x <- glm(am ~ hp, data = head(mtcars), family = binomial)
margins(x, type = "response")
margins(x, type = "link")

# multi-category outcome
if (requireNamespace("nnet")) {
  data("iris3", package = "datasets")
  ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                    species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))
  m <- nnet::nnet(species ~ ., data = ird, size = 2, rang = 0.1,
                  decay = 5e-4, maxit = 200, trace = FALSE)
  margins(m) # default
  margins(m, category = "v") # explicit category
}

# using margins_summary() for concise grouped operations
list_data <- split(mtcars, mtcars$gear)
list_mod <- lapply(list_data, function(x) lm(mpg ~ cyl + wt, data = x))
mapply(margins_summary, model = list_mod, data = list_data, SIMPLIFY = FALSE)
```

plot.margins *Plot Marginal Effects Estimates*

Description

An implementation of Stata's 'marginsplot' as an S3 generic function

Usage

```
## S3 method for class 'margins'
plot(
  x,
  pos = seq_along(marginal_effects(x, with_at = FALSE)),
  which = colnames(marginal_effects(x, with_at = FALSE)),
  labels = gsub("^dydx_", "", which),
  horizontal = FALSE,
  xlab = "",
  ylab = "Average Marginal Effect",
  level = 0.95,
  pch = 21,
  points.col = "black",
  points.bg = "black",
  las = 1,
  cex = 1,
  lwd = 2,
  zeroline = TRUE,
  zero.col = "gray",
  ...
)
```

Arguments

x	An object of class "margins", as returned by margins.
pos	A numeric vector specifying the x-positions of the estimates (or y-positions, if horizontal = TRUE).
which	A character vector specifying which marginal effect estimate to plot. Default is all.
labels	A character vector specifying the axis labels to use for the marginal effect estimates. Default is the variable names from x.
horizontal	A logical indicating whether to plot the estimates along the x-axis with vertical confidence intervals (the default), or along the y-axis with horizontal confidence intervals.
xlab	A character string specifying the x-axis (or y-axis, if horizontal = TRUE) label.
ylab	A character string specifying the y-axis (or x-axis, if horizontal = TRUE) label.

level	A numeric value between 0 and 1 indicating the confidence level to use when drawing error bars.
pch	The point symbol to use for plotting marginal effect point estimates. See points for details.
points.col	The point color to use for plotting marginal effect point estimates. See points for details.
points.bg	The point color to use for plotting marginal effect point estimates. See points for details.
las	An integer value specifying the orientation of the axis labels. See par for details.
cex	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. See par for details.
lwd	A numerical value giving the width of error bars in points.
zeroline	A logical indicating whether to draw a line indicating zero. Default is TRUE.
zero.col	A character string indicating a color to use for the zero line if zeroline = TRUE.
...	Additional arguments passed to plot.default, such as title, etc.

Details

This function is invoked for its side effect: a basic dot plot with error bars displaying marginal effects as generated by margins, in the style of Stata's 'marginsplot' command.

Value

The original "margins" object x, invisibly.

See Also

margins, persp.lm

Examples

```
## Not run:
  require("datasets")
  x <- lm(mpg ~ cyl * hp + wt, data = mtcars)
  mar <- margins(x)
  plot(mar)

## End(Not run)
```

Index