Package 'effects'

October 13, 2022

Version 4.2-2

Date 2022-07-12

Title Effect Displays for Linear, Generalized Linear, and Other Models

Depends R (>= 3.5.0), carData

Suggests pbkrtest (>= 0.4-4), nlme, MASS, poLCA, heplots, splines,
ordinal, car, knitr, betareg, alr4, robustlmm

Imports lme4, nnet, lattice, grid, colorspace, graphics, grDevices,
stats, survey, utils, estimability, insight

Description Graphical and tabular effect displays, e.g., of interactions, for
various statistical models with linear predictors.

License GPL (>= 2)

URL https://www.r-project.org,

https://socialsciences.mcmaster.ca/jfox/

VignetteBuilder knitr

NeedsCompilation no

Author John Fox [aut, cre],
Sanford Weisberg [aut],
Brad Price [aut],
Michael Friendly [aut],
Jangman Hong [aut],
Robert Andersen [ctb],
David Firth [ctb],
Steve Taylor [ctb],
R Core Team [ctb]

Maintainer John Fox <jfox@mcmaster.ca>

Repository CRAN

Date/Publication 2022-07-13 15:00:03 UTC

R **topics documented:**

---

effects-package   *Effect Displays for Linear, Generalized Linear, and Other Models*

Description

Graphical and tabular effect displays, e.g., of interactions, for various statistical models with linear predictors.

Details

Package:	effects
Version:	4.2-2
Date:	2022-02-16
Depends:	R (>= 3.5.0), carData
Suggests:	pbkrtest (>= 0.4-4), nlme, MASS, poLCA, heplots, splines, ordinal, car, knitr, betareg, alr4, robustlmm
Imports:	lme4, nnet, lattice, grid, colorspace, graphics, grDevices, stats, survey, utils, estimability, insight
LazyLoad:	yes
License:	GPL (>= 2)
URL:	https://www.r-project.org, https://socialsciences.mcmaster.ca/jfox/

This package creates effect displays for various kinds of models, as partly explained in the references. Typical usage is plot(allEffects(model)) or plot(predictorEffects(model)), where model is an appropriate fitted-model object. Additional arguments to allEffects, predictorEffects and plot can be used to customize the resulting displays. The function effect can be employed to produce an effect display for a particular term in the model, or to which terms in the model are marginal. The function predictorEffect can be used to construct an effect display for a particularly predictor. The function Effect may similarly be used to produce an effect display for any combination of predictors. In any of the cases, use plot to graph the resulting effect object. For linear and generalized linear models it is also possible to plot partial residuals to obtain (multidimensional) component+residual plots. See ?effect, ?Effect, ?predictorEffect, and ?plot.eff for details.

Author(s)

John Fox, Sanford Weisberg, Brad Price, Michael Friendly, Jangman Hong, Robert Anderson, David Firth, Steve Taylor, and the R Core Team.

Maintainer: John Fox <jfox@mcmaster.ca>

References

Fox, J. and S. Weisberg (2019) *An R Companion to Applied Regression, Third Edition* Sage Publications.

Fox, J. (1987) Effect displays for generalized linear models. *Sociological Methodology* **17**, 347–361.

Fox, J. (2003) Effect displays in R for generalised linear models. *Journal of Statistical Software* **8:15**, 1–27, doi:10.18637/jss.v008.i15.

Fox, J. and R. Andersen (2006) Effect displays for multinomial and proportional-odds logit models. *Sociological Methodology* **36**, 225–255.

Fox, J. and J. Hong (2009). Effect displays in R for multinomial and proportional-odds logit models: Extensions to the effects package. *Journal of Statistical Software* **32:1**, 1–24, doi:10.18637/jss.v032.i01.

Fox, J. and S. Weisberg (2018). Visualizing Fit and Lack of Fit in Complex Regression Models: Effect Plots with Partial Residuals. *Journal of Statistical Software* **87:9**, 1–27, doi:10.18637/jss.v087.i09.

---

effCoef	*Function to get coefficient estimates from regression models for use in the effects package.*

Description

This function uses the get_parameters function in the insight package to get a vector of regression coefficients for use in the effects package. It converts the two-column data.frame returned by get_parameters to a vector of named elements.

Usage

```
effCoef(mod, ...)

## Default S3 method:
effCoef(mod, ...)
```

Arguments

mod	A model object with a linear predictor representing fixed effects.
...	Additional parameter passed to get_parameters.

Details

The `get_parameters` function can be used to retrieve the coefficient estimates corresponding to a linear predictor for many regression models, and return them as a two column `data.frame`, with regressor names in the first column and estimates in the second column. This function converts this output to a named vector as is expected by the `effects` package.

Value

A vector of coefficient estimates

Author(s)

Sanford Weisberg <sandy@umn.edu>

See Also

`get_parameters`, and vignette Regression Models Supported by the effects Package

Examples

```
m1 <- lm(prestige ~ type + income + education, Duncan)
effCoef(m1)
```

---

effect	*Functions For Constructing Effect Displays*

Description

`Effect` and `effect` construct an `"eff"` object for a term (usually a high-order term) in a regression that models a response as a linear function of main effects and interactions of factors and covariates. These models include, among others, linear models (fit by `lm` and `gls`), and generalized linear models (fit by `glm`), for which an `"eff"` object is created, and multinomial and proportional-odds logit models (fit respectively by `multinom` and `polr`), for which an `"effpoly"` object is created. The computed effect absorbs the lower-order terms marginal to the term in question, and averages over other terms in the model. For multivariate linear models (of class `"mlm"`, fit by `lm`), the functions construct a list of `"eff"` objects, separately for the various response variables in the model.

`effect` builds the required object by specifying explicitly a focal term like `"a:b"` for an a by b interaction. `Effect` in contrast specifies the predictors in a term, for example `c("a", "b")`, rather than the term itself. `Effect` is consequently more flexible and robust than `effect`, and will succeed with some models for which `effect` fails. The `effect` function works by constructing a call to `Effect` and continues to be included in **effects** so older code that uses it will not break.

The `Effect` and `effect` functions can also be used with many other models; see `Effect.default` and the Regression Models Supported by the effects Package vignette.

`allEffects` identifies all of the high-order terms in a model and returns a list of `"eff"` or `"effpoly"` objects (i.e., an object of class `"efflist"`).

For information on computing and displaying *predictor effects*, see `predictorEffect` and `plot.predictoreff`.

For further information about plotting effects, see `plot.eff`.

Usage

```
effect(term, mod, vcov.=vcov, ...)

## Default S3 method:
effect(term, mod, vcov.=vcov, ...)

Effect(focal.predictors, mod, ...)

## S3 method for class 'lm'
Effect(focal.predictors, mod, xlevels=list(),
    fixed.predictors, vcov. = vcov, se=TRUE,
    residuals=FALSE, quantiles=seq(0.2, 0.8, by=0.2),
    x.var=NULL,  ...,
    #legacy arguments:
    given.values, typical, offset, confint, confidence.level,
    partial.residuals, transformation)

## S3 method for class 'multinom'
Effect(focal.predictors, mod,
    xlevels=list(), fixed.predictors,
    vcov. = vcov, se=TRUE, ...,
    #legacy arguments:
    confint, confidence.level, given.values, typical)

## S3 method for class 'polr'
Effect(focal.predictors, mod,
    xlevels=list(), fixed.predictors,
    vcov.=vcov, se=TRUE, latent=FALSE, ...,
    #legacy arguments:
    confint, confidence.level, given.values, typical)

## S3 method for class 'svyglm'
Effect(focal.predictors, mod, fixed.predictors, ...)

## S3 method for class 'merMod'
Effect(focal.predictors, mod, ..., KR=FALSE)

## S3 method for class 'poLCA'
Effect(focal.predictors, mod, ...)

## S3 method for class 'mlm'
Effect(focal.predictors, mod, response, ...)

allEffects(mod, ...)

## Default S3 method:
allEffects(mod, ...)
```

Arguments

term
the quoted name of a term, usually, but not necessarily, a high-order term in the model. The term must be given exactly as it appears in the printed model, although either colons (:) or asterisks (*) may be used for interactions. If term is NULL, the function returns the formula for the linear predictor.

focal.predictors
a character vector of one or more predictors in the model in any order.

mod
a regression model object. If no specific method exists for the class of mod, Effect.default will be called.

xlevels
this argument is used to set the number of levels for any focal numeric predictor (that is predictors that are not factors, character variables, or logical variables, all of which are treated as factors). If xlevels=NULL, then each numeric predictor is represented by five values over its range, equally spaced and then rounded to 'nice' numbers. If xlevels=n is an integer, then each numeric predictor is represented by n equally spaced values rounded to 'nice' numbers.

More generally, xlevels can be a named list of values at which to set each numeric predictor. For example, xlevels=list(x1=c(2, 4.5, 7), x2=4) would use the values 2, 4.5, and 7 for x1, use 4 equally spaced values for x2, and use the default for any other numeric predictors.

If partial residuals are computed, then the focal predictor that is to appear on the horizontal axis of an effect plot is evaluated at 100 equally spaced values along its full range, and, by default, other numeric predictors are evaluated at the quantiles specified in the quantiles argument, unless their values are given explicitly in xlevels.

fixed.predictors
an optional list of specifications affecting the values at which fixed predictors for an effect are set, potentially including:

given.values given.values="default" (which is, naturally, the default) specifies averaging over levels of a non-focal factor, weighting levels of the factor in proportion to sample size.
given.values="equal" computes unweighted averages over the levels of non-focal factors.
For finer control, the user can also provide a named numeric vector of weights for particular columns of the model matrix that correspond to the regressors for the factor.
Character and logical predictors are treated as factors.
For example, for a factor X with three levels a, b and c, the regressors generated using the default contr.treatment parameterization for a factor will be named Xb and Xc, as the regressor for level a is excluded as the baseline level. The specification given.values=c(Xb=1/2, Xc=1/4) would average over the levels of X with weight 1/2 for level b, 1/4 for c, and weight 1 = 1/2 - 1/4 = 1/4 for the baseline level a. Setting given.values=c(Xb=1) would fix X at level b.

typical a function to be applied to the columns of the model matrix over which the effect is "averaged"; with the exception of the "svyglm" method, the default is mean. For "svyglm" objects, the default is to use the survey-design weighted mean.

apply.typical.to.factors It generally doesn't make sense to apply typical values that aren't means (e.g., medians) to the columns of the model-matrix representing contrasts for factors. This value generally defaults to FALSE except for "svyglm" objects, for which the default is TRUE, using the the survey-design weighted mean.

offset a function to be applied to the offset values (if there is an offset) in a linear or generalized linear model, or a mixed-effects model fit by lmer or glmer; or a numeric value, to which the offset will be set. The default is the mean function, and thus the offset will be set to its mean; in the case of "svyglm" objects, the default is to use the survey-design weighted mean. *Note:* Only offsets defined by the offset argument to lm, glm, svyglm, lmer, or glmer will be handled correctly; use of the offset function in the model formula is not supported.

vcov. Effect methods generally use the matrix returned by vcov(mod) to compute standard errors and confidence bounds. Alternatively, the user may specify the name of a function that returns a matrix of the same dimension and structure as the matrix returned by vcov(mod). For example, vcov. = hccm uses the hccm function from the **car** package to use a heteroscedasticity corrected covariance matrix for a linear model in place of the standard covariance estimate. This argument can be set to equal matrix of the same size and structure as the matrix returned by vcov(mod). For example, using vcov. = vcov(Boot(mod)) uses Boot from the **car** package to get a bootstrap estimate of the covariance matrix for linear, generalized linear, and possibly other modeling frameworks.

se TRUE (the default), FALSE, or a list with any or all of the following elements, controlling whether and how standard errors and confidence limits are computed for the effects:

compute (default TRUE) whether or not to compute standard errors and confidence limits.

level (default 0.95) confidence level for confidence limits.

type one of "pointwise" (the default), "Scheffe", or "scheffe", whether to compute confidence limits with specified coverage at each point for an effect or to compute limits for a Scheffe-type confidence envelope. For mer, merMod, and lme objects, the normal distribution is used to get confidence limits.

residuals if TRUE, residuals for a linear or generalized linear model will be computed and saved; if FALSE (the default), residuals are suppressed. If residuals are saved, partial residuals are computed when the effect is plotted: see plot.eff and the vignette Effect Displays with Partial Residuals. This argument may also be used for mixed-effects and some other models.

quantiles quantiles at which to evaluate numeric focal predictors *not* on the horizontal axis, used only when partial residuals are displayed; superseded if the xlevels argument gives specific values for a predictor.

x.var the (quoted) name or index of the numeric predictor to define the horizontal axis of an effect plot for a linear or generalized linear model; the default is NULL, in which case the first numeric predictor in the effect will be used *if* partial residuals are to be computed. This argument is intended to be used when residuals

	is TRUE; otherwise, the variable on the horizontal axis can be chosen when the effect object is plotted: see `plot.eff`.
latent	if TRUE, effects in a proportional-odds logit model are computed on the scale of the latent response; if FALSE (the default) effects are computed as individual-level probabilities and logits.
x	an object of class `"eff"`, `"effpoly"`, or `"efflatent"`.
KR	if TRUE and the **pbkrtest** package is installed, use the Kenward-Roger coefficient covariance matrix to compute effect standard errors for linear mixed models fit with `lmer`; the default is FALSE because the computation can be time-consuming.
response	for an `"mlm"` object, a vector containing the (quoted) name(s) or indices of one or more response variable(s). The default is to use all responses in the model.
...	arguments to be passed down.
confint, confidence.level, given.values, typical, offset, partial.residuals, transformation	legacy arguments retained for backwards compatibility; if present, these arguments take precedence over the `level` element of the `confint` list argument and the `given.values`, `typical`, and `offset` elements of the `fixed.predictors` list argument; `confint` may be used in place of the `se` argument; `partial.residuals` may be used in place of the `residuals` argument. See `LegacyArguments` for details.

Details

Normally, the functions to be used directly are `allEffects`, to return a list of high-order effects, and the generic `plot` function to plot the effects (see `plot.efflist`, `plot.eff`, and `plot.effpoly`). Alternatively, `Effect` can be used to vary a subset of predictors over their ranges, while other predictors are held to typical values.

Plotting methods for effect objects call the `xyplot` (or in some cases, the `densityplot`) function in the **lattice** package. Effects may also be printed (implicitly or explicitly via `print`) or summarized (using `summary`) (see `print.efflist`, `summary.efflist`, `print.eff`, `summary.eff`, `print.effpoly`, and `summary.effpoly`).

If asked, the `effect` function will compute effects for terms that have higher-order relatives in the model, averaging over those terms (which rarely makes sense), or for terms that do not appear in the model but are higher-order relatives of terms that do. For example, for the model Y ~ A*B + A*C + B*C, one could compute the effect corresponding to the absent term A:B:C, which absorbs the constant, the A, B, and C main effects, and the three two-way interactions. In either of these cases, a warning is printed.

See `predictorEffects` for an alternative paradigm for defining effects.

Value

For `"lm"`, `"glm"`, `"svyglm"`, `"lmerMod"`, `"glmerMod"`, and `"lme"`, model objects, `effect` and `Effect` return an `"eff"` object, and for `"multinom"`, `"polr"`, `"clm"`, `"clmm"`, and `"clm2"` models, an `"effpoly"` object, with the components listed below. For an `"mlm"` object with one response specified, an `"eff"` object is returned, otherwise an `"efflist"` object is returned, containing one `"eff"` object for each `response`.

term	the term to which the effect pertains.

formula	the complete model formula.
response	a character string giving the name of the response variable.
y.levels	(for "effpoly" objects) levels of the polytomous response variable.
variables	a list with information about each predictor, including its name, whether it is a factor, and its levels or values.
fit	(for "eff" objects) a one-column matrix of fitted values, representing the effect on the scale of the linear predictor; this is a raveled table, representing all combinations of predictor values.
prob	(for "effpoly" objects) a matrix giving fitted probabilities for the effect for the various levels of the the response (columns) and combinations of the focal predictors (rows).
logit	(for "effpoly" objects) a matrix giving fitted logits for the effect for the various levels of the the response (columns) and combinations of the focal predictors (rows).
x	a data frame, the columns of which are the predictors in the effect, and the rows of which give all combinations of values of these predictors.
model.matrix	the model matrix from which the effect was calculated.
data	a data frame with the data on which the fitted model was based.
discrepancy	the percentage discrepancy for the 'safe' predictions of the original fit; should be very close to 0. Note: except for gls models, this is now necessarily 0.
offset	value to which the offset is fixed; 0 if there is no offset.
model	(for "effpoly" objects) "multinom" or "polr", as appropriate.
vcov	(for "eff" objects) a covariance matrix for the effect, on the scale of the linear predictor.
se	(for "eff" objects) a vector of standard errors for the effect, on the scale of the linear predictor.
se.prob, se.logit	
	(for "effpoly" objects) matrices of standard errors for the effect, on the probability and logit scales.
lower, upper	(for "eff" objects) one-column matrices of confidence limits, on the scale of the linear predictor.
lower.prob, upper.prob, lower.logit, upper.logit	
	(for "effpoly" objects) matrices of confidence limits for the fitted logits and probabilities; the latter are computed by transforming the former.
confidence.level	
	for the confidence limits.
transformation	(for "eff" objects) a two-element list, with element link giving the link function, and element inverse giving the inverse-link (mean) function.
residuals	(working) residuals for linear or generalized linear models (and some similar models), to be used by plot.eff to compute and plot partial residuals.
x.var	the name of the predictor to appear on the horizontal axis of an effect plot made from the returned object; will usually be NULL if partial residuals aren't computed.

family   for a "glm" model, the name of the distributional family of the model; for an
                   "lm" model, this is "gaussian"; otherwise NULL. The family controls how
                   partial residuals are smoothed in plots.

link   the value returned by family(mod). Down-stream methods may need the link,
                   inverse link and derivative functions.

allEffects returns an "efflist" object, a list of "eff" or "effpoly" objects corresponding to
the high-order terms of the model.

If mod is of class "poLCA" (from the **poLCA** package), representing a polytomous latent class
model, effects are computed for the predictors given the estimated latent classes. The result is of
class "eff" if the latent class model has 2 categories and of class "effpoly" with more than 2
categories.

Warnings and Limitations

The Effect function handles factors and covariates differently, and is likely to be confused if one is
changed to the other in a model formula. Consequently, formulas that include calls to as.factor,
factor, or numeric (as, e.g., in y ~ as.factor(income)) will cause errors. Instead, create the
modified variables outside of the model formula (e.g., fincome <- as.factor(income)) and use
these in the model formula.

The effect function doesn't work with factors that have colons in level names (e.g., "level:A");
the effect function will confuse the colons with interactions; rename levels to remove or replace
the colons (e.g., "level.A"). Level names with colons are perfectly fine for use with Effect.

The functions in the **effects** package work properly with predictors that are numeric variables, fac-
tors, character variables, or logical variables; consequently, e.g., convert dates to numeric. Character
predictors and logical predictors are treated as factors, the latter with "levels" "FALSE" and "TRUE".

Empty cells in crossed-factors are now permitted for "lm", "glm", and "multinom" models. For
"multinom" models with two or more crossed factors with an empty cell, stacked area plots appar-
ently do not work because of a bug in the barchart function in the **lattice** package. However, the
default line plots do work.

Offsets in linear and generalized linear models are supported, as are offsets in mixed models fit
by lmer or glmer, but must be supplied through the offset argument to lm, glm, lmer or glmer;
offsets supplied via calls to the offset function on the right-hand side of the model formula are not
supported.

Fitting ordinal mixed models using clmm or clmm2 permits many options, including a variety of link
functions, scale functions, nominal regressors, and various methods for setting thresholds. Effects
are currently generated only for the default values of the arguments scale, nominal, link, and
threshold, which is equivalent to fitting an ordinal-response mixed-effects model with a logit link.
Effect can also be used with objects created by clm or clm2, fitting ordinal response models with
the same links permitted by polr in the **MASS** package, with no random effects, and with results
similar to those from polr.

Calling any of these functions from within a user-written function may result in errors due to R's
scoping rules. See the vignette embedding.pdf in the **car** package for a solution to this problem.

Author(s)

John Fox <jfox@mcmaster.ca>, Sanford Weisberg <sandy@umn.edu> and Jangman Hong.

References

Fox, J. (1987). Effect displays for generalized linear models. *Sociological Methodology* **17**, 347–361.

Fox, J. (2003) Effect displays in R for generalised linear models. *Journal of Statistical Software* **8:15**, 1–27, doi:10.18637/jss.v008.i15.

Fox, J. and R. Andersen (2006). Effect displays for multinomial and proportional-odds logit models. *Sociological Methodology* **36**, 225–255.

Fox, J. and J. Hong (2009). Effect displays in R for multinomial and proportional-odds logit models:? Extensions to the effects package. *Journal of Statistical Software* **32:1**, 1–24, doi:10.18637/jss.v032.i01.

Fox, J. and S. Weisberg (2019). *An R Companion to Applied Regression, third edition*, Thousand Oaks: Sage.

Fox, J. and S. Weisberg (2018). Visualizing Fit and Lack of Fit in Complex Regression Models with Predictor Effect Plots with Partial Residuals. *Journal of Statistical Software* **87:9**, 1–27, doi:10.18637/jss.v087.i09.

Hastie, T. J. (1992). Generalized additive models. In Chambers, J. M., and Hastie, T. J. (eds.) *Statistical Models in S*, Wadsworth.

Weisberg, S. (2014). *Applied Linear Regression*, 4th edition, Wiley, http://z.umn.edu/alr4ed.

See Also

LegacyArguments. For information on printing, summarizing, and plotting effects: print.eff, summary.eff, plot.eff, print.summary.eff, print.effpoly, summary.effpoly, plot.effpoly, print.efflist, summary.efflist, plot.efflist, xyplot, densityplot, and the Effect Displays with Partial Residuals and Regression Models Supported by the effects Package vignettes.

Examples

```
mod.cowles <- glm(volunteer ~ sex + neuroticism*extraversion,
                  data=Cowles, family=binomial)
eff.cowles <- allEffects(mod.cowles, xlevels=list(extraversion=seq(0, 24, 6)),
                    fixed.predictors=list(given.values=c(sexmale=0.5)))
eff.cowles
as.data.frame(eff.cowles[[2]])


# the following are equivalent:
eff.ne <- effect("neuroticism*extraversion", mod.cowles)
Eff.ne <- Effect(c("neuroticism", "extraversion"), mod.cowles)
all.equal(eff.ne$fit, Eff.ne$fit)

plot(eff.cowles, 'sex', axes=list(y=list(lab="Prob(Volunteer)")))

plot(eff.cowles, 'neuroticism:extraversion',
     axes=list(y=list(lab="Prob(Volunteer)",
        ticks=list(at=c(.1,.25,.5,.75,.9)))))

plot(Effect(c("neuroticism", "extraversion"), mod.cowles,
```

```
            se=list(type="Scheffe"),
            xlevels=list(extraversion=seq(0, 24, 6)),
            fixed.predictors=list(given.values=c(sexmale=0.5))),
      axes=list(y=list(lab="Prob(Volunteer)",
        ticks=list(at=c(.1,.25,.5,.75,.9)))))

plot(eff.cowles, 'neuroticism:extraversion', lines=list(multiline=TRUE),
      axes=list(y=list(lab="Prob(Volunteer)")))

plot(effect('sex:neuroticism:extraversion', mod.cowles,
            xlevels=list(extraversion=seq(0, 24, 6))),
      lines=list(multiline=TRUE))


# a nested model:

mod <- lm(log(prestige) ~ income:type + education, data=Prestige)

plot(Effect(c("income", "type"), mod, transformation=list(link=log, inverse=exp)),
      axes=list(y=list(lab="prestige")))


if (require(nnet)){
    mod.beps <- multinom(vote ~ age + gender + economic.cond.national +
                             economic.cond.household + Blair + Hague + Kennedy +
                             Europe*political.knowledge, data=BEPS)

    plot(effect("Europe*political.knowledge", mod.beps,
                xlevels=list(political.knowledge=0:3)))


    plot(Effect(c("Europe", "political.knowledge"), mod.beps,
                xlevels=list(Europe=1:11, political.knowledge=0:3),
                fixed.predictors=list(given.values=c(gendermale=0.5))),
        lines=list(col=c("blue", "red", "orange")),
        axes=list(x=list(rug=FALSE), y=list(style="stacked")))


    plot(effect("Europe*political.knowledge", mod.beps, # equivalent
                xlevels=list(Europe=1:11, political.knowledge=0:3),
                fixed.predictors=list(given.values=c(gendermale=0.5))),
        lines=list(col=c("blue", "red", "orange")),
        axes=list(x=list(rug=FALSE), y=list(style="stacked")))

}

if (require(MASS)){
    mod.wvs <- polr(poverty ~ gender + religion + degree + country*poly(age,3),
                    data=WVS)

    plot(effect("country*poly(age, 3)", mod.wvs))
```

```
    plot(Effect(c("country", "age"), mod.wvs),
         axes=list(y=list(style="stacked")))


    plot(effect("country*poly(age, 3)", mod.wvs),
         axes=list(y=list(style="stacked"))) # equivalent

    plot(effect("country*poly(age, 3)", latent=TRUE, mod.wvs))
    plot(effect("country*poly(age, 3)", latent=TRUE, mod.wvs,
         se=list(type="scheffe"))) # Scheffe-type confidence envelopes

}


mod.pres <- lm(prestige ~ log(income, 10) + poly(education, 3) + poly(women, 2),
               data=Prestige)
eff.pres <- allEffects(mod.pres, xlevels=50)
plot(eff.pres)
plot(eff.pres[1],
     axes=list(x=list(income=list(
            transform=list(trans=log10, inverse=function(x) 10^x),
            ticks=list(at=c(1000, 2000, 5000, 10000, 20000))
    )))))

# linear model with log-response and log-predictor
# to illustrate transforming axes and setting tick labels
mod.pres1 <- lm(log(prestige) ~ log(income) + poly(education, 3) + poly(women, 2),
               data=Prestige)
# effect of the log-predictor
eff.log <- Effect("income", mod.pres1)
# effect of the log-predictor transformed to the arithmetic scale
eff.trans <- Effect("income", mod.pres1, transformation=list(link=log, inverse=exp))
#variations:
# y-axis:  scale is log, tick labels are log
# x-axis:  scale is arithmetic, tick labels are arithmetic
plot(eff.log)

# y-axis:  scale is log, tick labels are log
# x-axis:  scale is log, tick labels are arithmetic
plot(eff.log, axes=list(x=list(income=list(
    transform=list(trans=log, inverse=exp),
    ticks=list(at=c(5000, 10000, 20000)),
    lab="income, log-scale"))))

# y-axis:  scale is log, tick labels are arithmetic
# x-axis:  scale is arithmetic, tick labels are arithmetic
plot(eff.trans, axes=list(y=list(lab="prestige")))

# y-axis:  scale is arithmetic, tick labels are arithmetic
# x-axis:  scale is arithmetic, tick labels are arithmetic
plot(eff.trans, axes=list(y=list(type="response", lab="prestige")))

# y-axis:  scale is log, tick labels are arithmetic
```

```
# x-axis:  scale is log, tick labels are arithmetic
plot(eff.trans, axes=list(
        x=list(income=list(
              transform=list(trans=log, inverse=exp),
              ticks=list(at=c(1000, 2000, 5000, 10000, 20000)),
              lab="income, log-scale")),
          y=list(lab="prestige, log-scale")),
      main="Both response and X in log-scale")

# y-axis:  scale is arithmetic, tick labels are arithmetic
# x-axis:  scale is log, tick labels are arithmetic
plot(eff.trans, axes=list(
        x=list(
            income=list(transform=list(trans=log, inverse=exp),
                        ticks=list(at=c(1000, 2000, 5000, 10000, 20000)),
                        lab="income, log-scale")),
        y=list(type="response", lab="prestige")))


if (require(nlme)){ # for gls()
    mod.hart <- gls(fconvict ~ mconvict + tfr + partic + degrees, data=Hartnagel,
                    correlation=corARMA(p=2, q=0), method="ML")
    plot(allEffects(mod.hart))
    detach(package:nlme)
}

if (require(lme4)){
    data(cake, package="lme4")
    fm1 <- lmer(angle ~ recipe * temperature + (1|recipe:replicate), cake,
                REML = FALSE)
    plot(Effect(c("recipe", "temperature"), fm1))

    plot(effect("recipe:temperature", fm1),
         axes=list(grid=TRUE)) # equivalent (plus grid)

    if (any(grepl("pbkrtest", search()))) detach(package:pbkrtest)
    detach(package:lme4)
}


if (require(nlme) && length(find.package("lme4", quiet=TRUE)) > 0){
    data(cake, package="lme4")
    cake$rep <- with(cake, paste( as.character(recipe), as.character(replicate), sep=""))
    fm2 <- lme(angle ~ recipe * temperature, data=cake,
                random = ~ 1 | rep, method="ML")
    plot(Effect(c("recipe", "temperature"), fm2))
    plot(effect("recipe:temperature", fm2),
         axes=list(grid=TRUE))  # equivalent (plus grid)
    }
    detach(package:nlme)
```

```
if (require(poLCA)){
    data(election)
    f2a <- cbind(MORALG,CARESG,KNOWG,LEADG,DISHONG,INTELG,
                 MORALB,CARESB,KNOWB,LEADB,DISHONB,INTELB)~PARTY*AGE
    nes2a <- poLCA(f2a,election,nclass=3,nrep=5)
    plot(Effect(c("PARTY", "AGE"), nes2a),
         axes=list(y=list(style="stacked")))
}


# mlm example
if (require(heplots)) {
    data(NLSY, package="heplots")
    mod <- lm(cbind(read,math) ~ income+educ, data=NLSY)
    eff.inc <- Effect("income", mod)
    plot(eff.inc)
    eff.edu <- Effect("educ", mod)
    plot(eff.edu, axes=list(x=list(rug=FALSE), grid=TRUE))

    plot(Effect("educ", mod, response="read"))

    detach(package:heplots)
}

# svyglm() example (adapting an example from the survey package)


if (require(survey)){
  data("api")
  dstrat<-svydesign(id=~1, strata=~stype, weights=~pw,
    data=apistrat, fpc=~fpc)
  mod <- svyglm(sch.wide ~ ell + meals + mobility, design=dstrat,
    family=quasibinomial())
  plot(allEffects(mod),
    axes=list(y=list(lim=log(c(0.4, 0.99)/c(0.6, 0.01)),
      ticks=list(at=c(0.4, 0.75, 0.9, 0.95, 0.99)))))
}


# component + residual plot examples


Prestige$type <- factor(Prestige$type, levels=c("bc", "wc", "prof"))

mod.prestige.1 <- lm(prestige ~ income + education, data=Prestige)
plot(allEffects(mod.prestige.1, residuals=TRUE)) # standard C+R plots
plot(allEffects(mod.prestige.1, residuals=TRUE,
    se=list(type="scheffe"))) # with Scheffe-type confidence bands

mod.prestige.2 <- lm(prestige ~ type*(income + education), data=Prestige)
plot(allEffects(mod.prestige.2, residuals=TRUE))

mod.prestige.3 <- lm(prestige ~ type + income*education, data=Prestige)
```

```
plot(Effect(c("income", "education"), mod.prestige.3, residuals=TRUE),
     partial.residuals=list(span=1))


#  artificial data

set.seed(12345)
x1 <- runif(500, -75, 100)
x2 <- runif(500, -75, 100)
y <- 10 + 5*x1 + 5*x2 + x1^2 + x2^2 + x1*x2 + rnorm(500, 0, 1e3)
Data <- data.frame(y, x1, x2)
mod.1 <- lm(y ~ poly(x1, x2, degree=2, raw=TRUE), data=Data)
# raw=TRUE necessary for safe prediction
mod.2 <- lm(y ~ x1*x2, data=Data)
mod.3 <- lm(y ~ x1 + x2, data=Data)

plot(Effect(c("x1", "x2"), mod.1, residuals=TRUE)) # correct model
plot(Effect(c("x1", "x2"), mod.2, residuals=TRUE)) # wrong model
plot(Effect(c("x1", "x2"), mod.3, residuals=TRUE)) # wrong model
```

---

effectsHexsticker	*View the Official Hex Sticker for the effects Package*

Description

Open the official hex sticker for the effects package in your browser

Usage

```
effectsHexsticker()
```

Value

Used for its side effect of openning the hex sticker for the effects package in your browser.

Author(s)

John Fox <jfox@mcmaster.ca>

Examples

```
## Not run:
effectsHexsticker()

## End(Not run)
```

---

effectsTheme   *Set the lattice Theme for Effect Plots*

Description

Set the **lattice** theme (see `trellis.device`) appropriately for effect plots. This function is invoked automatically when the **effects** package is loaded *if* the **lattice** package hasn't previously been loaded. A typical call is `lattice::trellis.par.set(effectsTheme())`.

Usage

```
effectsTheme(strip.background = list(col = gray(seq(0.95, 0.5, length = 3))),
          strip.shingle = list(col = "black"), clip = list(strip = "off"),
          superpose.line = list(lwd = c(2, rep(1, 6))), col)
```

Arguments

strip.background
:   colors for the background of conditioning strips at the top of each panel; the default uses shades of gray and makes allowance for up to three conditioning variables.

strip.shingle
:   when lines rather than numeric values are used to indicate the values of conditioning variables, the default sets the color of the lines to black.

clip
:   the default allows lines showing values of conditioning variables to extend slightly beyond the boundaries of the strips—making the lines more visible at the extremes.

superpose.line
:   the default sets the line width of the first (of seven) lines to 2.

col
:   an optional argument specifying the colors to use for lines and symbolst: if col = "car", then the color palette for the **car** package is used (see `carPalette`); col = "R", then the current R palette (ignoring the first entry which is "black" in the standard R palette) is used (see `palette`); if col = "colorblind", then a colorblind-friendly palette (from https://jfly.uni-koeln.de/color/ but ignoring black) is used; if a vector of color specifications, then these are used. If col isn't specified then the current **lattice** colors are used.

Value

a list suitable as an argument for `trellis.par.set`; current values of modified parameters are supplied as an attribute.

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

`trellis.device`, `trellis.par.set`

Examples

```
## Not run:
lattice::trellis.par.set(effectsTheme())

## End(Not run)
```

LegacyArguments   *Legacy Arguments for* plot *and* Effect *Methods*

Description

Prior to verson 4.0-0 of the **effects** package, there were many (literally dozens) of arguments to the plot methods for "eff" and "effpoly" objects.

In version 4.0-0 of the package, we have consolidated these arguments into a much smaller number of arguments (e.g., lines, points, axes) that take lists of specifications. We have similarly consolidated some of the arguments to Effect methods into the confint and fixed.predictors arguments.

For backwards compatibility, we have to the extent possible retained the older arguments. If specified, these legacy arguments take precedence over the newer list-style arguments

Details

Here is the correspondence between the old and new arguments.

For plot methods:

multiline=TRUE/FALSE lines=list(multiline=TRUE/FALSE)

type=c("rescale", "link", "response") For models with a link function, "link" plots in linear predictor scale, "response" plots in the response scale, and the default "rescale" plots in linear predictor scale but labels tick-marks in response scale.

z.var=which.min(levels) lines=list(z.var=which.min(levels)) relevant only when lines=list(multiline=TRUE

colors={vector of colors} lines=list(col={vector of colors})

lty={vector of line types} lines=list(lty={vector of line types})

lwd={vector of line widths} lines=list(lwd={vector of line widths})

use.splines=TRUE/FALSE lines=list(splines=TRUE/FALSE)

cex={number} points=list(cex={number})

rug=TRUE/FALSE axes=list(x=list(rug=TRUE/FALSE)

xlab={"axis title"} axes=list(x=list(lab={"axis title"}))

xlim={c(min, max)} axes=list(x=list(lim={c(min, max)}))

rotx={degrees} axes=list(x=list(rot={degrees}))

ticks.x=list({tick specifications}) axes=list(x=list(ticks=list({tick specifications})))

transform.x=list(link={function}, inverse={function}) axes=list(x=list(transform=list({lists of transformations by predictors})))

```
ylab={"axis title"} axes=list(y=list(lab={"axis title"}))
ylim={c(min, max)} axes=list(y=list(lim={c(min, max)}))
roty={degrees} axes=list(y=list(rot={degrees}))
ticks=list({tick specifications}) axes=list(y=list(ticks=list({tick specifications})))
alternating=TRUE/FALSE axes=list(alternating=TRUE/FALSE)
grid=TRUE/FALSE axes=list(grid=TRUE/FALSE)
ci.style="bands"/"lines"/"bars"/"none" confint=list(style="bands"/"lines"/"bars"/"none")
band.transparency={number} confint=list(alpha={number})
band.colors={vector of colors} confint=list(col={vector of colors})
residuals.color={color} partial.residuals=list(col={color})
residuals.pch={plotting character} partial.residuals=list(pch={plotting character})
residuals.cex={number} partial.residuals=list(cex={number})
smooth.residuals=TRUE/FALSE partial.residuals=list(smooth=TRUE/FALSE)
residuals.smooth.color={color} partial.residuals=list(smooth.col={color})
span={number} partial.residuals=list(span={number})
show.fitted=TRUE/FALSE partial.residuals=list(fitted=TRUE/FALSE)
factor.names=TRUE/FALSE lattice=list(strip=list(factor.names=TRUE/FALSE))
show.strip.values=TRUE/FALSE lattice=list(strip=list(values=TRUE/FALSE))
layout={lattice layout} lattice=list(layout={lattice layout})
key.args={lattice key args} lattice=list(key.args={lattice key args})
style="lines"/"stacked" for plot.effpoly, axes=list(y=list(style="lines"/"stacked"))
rescale.axis=TRUE/FALSE type="rescale"/"response"/"link"
```

For `Effect` methods:

`confint=TRUE/FALSE` **or a list** may be substituted for the se argument.

`confidence.level={number} se=list(level={number})`

`given.values={named vector} fixed.predictors=list(given.values={named vector})`

`typical={function} fixed.predictors=list(typical={function})`

`offset={function} fixed.predictors=list(offset={function})`

`partial.residuals=TRUE/FALSE residuals=TRUE/FALSE`

`transformation` This argument to `Effect` is not needed to compute effects. It can now be set directly with the plot method with the argument `axes = list(y = list(transformation=specification))`.

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

[Effect](), [plot.eff](), [plot.effpoly]()

---

plot.effects   *Plots of Effects and Predictor Effects*

Description

plot methods for `predictoreff`, `predictorefflist`, `eff`, `efflist` and `effpoly` objects created
by calls other methods in the `effects` package. The plot arguments were substantially changed
in mid-2017. For more details and many examples, see the Predictor Effects Graphics Gallery
vignette.

Usage

```
## S3 method for class 'eff'
plot(x, x.var,
    main=paste(effect, "effect plot"),
    symbols=TRUE, lines=TRUE, axes, confint,
    partial.residuals, id, lattice, ...,
    # legacy arguments:
    multiline, z.var, rug, xlab, ylab, colors, cex, lty, lwd,
    ylim, xlim, factor.names, ci.style,
    band.transparency, band.colors, type, ticks,
    alternating, rotx, roty, grid, layout,
    rescale.axis, transform.x, ticks.x, show.strip.values,
    key.args, use.splines,
    residuals.color, residuals.pch, residuals.cex, smooth.residuals,
    residuals.smooth.color, show.fitted, span)

## S3 method for class 'efflist'
plot(x, selection, rows, cols, ask=FALSE, graphics=TRUE, lattice, ...)

## S3 method for class 'predictoreff'
plot(x, x.var,
    main = paste(names(x$variables)[1], "predictor effect plot"), ...)

## S3 method for class 'predictorefflist'
plot(x, selection, rows, cols, ask = FALSE,
    graphics = TRUE, lattice, ...)

## S3 method for class 'effpoly'
plot(x, x.var=which.max(levels),
    main=paste(effect, "effect plot"),
    symbols=TRUE, lines=TRUE, axes, confint, lattice, ...,
    # legacy arguments:
    type, multiline, rug, xlab, ylab, colors, cex, lty, lwd,
    factor.names, show.strip.values,
    ci.style, band.colors, band.transparency, style,
```

```
    transform.x, ticks.x, xlim,
    ticks, ylim, rotx, roty, alternating, grid,
    layout, key.args, use.splines)

## S3 method for class 'mlm.efflist'
plot(x, ...)
```

Arguments

x	an object of class "predictoreff", "predictorefflist", "eff", "effpoly", "efflist", "mlm.efflist", or "summary.eff", as appropriate.
x.var	the index (number) or quoted name of the covariate or factor to place on the horizontal axis of each panel of the effect plot. The default is the predictor with the largest number of levels or values. This argument is ignored with predictoreff objects.
main	the title for the plot, printed at the top; the default title is constructed from the name of the effect.
symbols	TRUE, FALSE, or an optional list of specifications for plotting symbols; if not given, symbol properties are taken from superpose.symbol in the lattice theme. See Detailed Argument Descriptions under Details for more information.
lines	TRUE, FALSE, or an optional list of specifications for plotting lines (and possibly areas); if not given, line properties are taken from superpose.line in the lattice theme. See Detailed Argument Descriptions under Details for more information.
axes	an optional list of specifications for the x and y axes; if not given, axis properties take generally reasonable default values. See Details for more information.
confint	an optional list of specifications for plotting confidence regions and intervals; if not given, generally reasonable default values are used. See Detailed Argument Descriptions under Details for more information.
partial.residuals	an optional list of specifications for plotting partial residuals for linear and generalized linear models; if not given, generally reasonable default values are used. See Detailed Argument Descriptions under Details for more information, along with the Effect Displays with Partial Residuals vignette.
id	an optional list of specifications for identifying points when partial residuals are plotted; if not specified, no points are labelled. See Detailed Argument Descriptions under Details for more information.
lattice	an optional list of specifications for various lattice properties, such as legend placement; if not given, generally reasonable default values are used. See Detailed Argument Descriptions under Details for more information.
selection	the optional index (number) or quoted name of the effect in an efflist object to be plotted; if not supplied, a menu of high-order terms is presented or all effects are plotted.
rows, cols	Number of rows and columns in the "meta-array" of plots produced for an efflist object; if either argument is missing, then the meta-layout will be computed by the plot method.

ask	if `selection` is not supplied and `ask` is TRUE, a menu of high-order terms is presented; if `ask` is FALSE (the default), effects for all high-order terms are plotted in an array.
graphics	if TRUE (the default), then the menu of terms to plot is presented in a dialog box rather than as a text menu.
...	arguments to be passed down. For `"predictoreff"` or `"predictorefflist"` objects, the arguments passed down can include all the arguments for `"eff"`.
multiline, z.var, rug, xlab, ylab, colors, cex, lty, lwd, ylim, xlim, factor.names, ci.style, band.transpa	legacy arguments retained for backwards compatibility; if specified, these will take precedence over the newer list-style arguments described above. See `LegacyArguments` for details.

Details

Effects plots and predictor effects plots are produced by these methods. The plots are highly customizable using the optional arguments described here. For example, effects in a GLM are plotted on the scale of the linear predictor, but the vertical axis is labelled on the response scale. This preserves the linear structure of the model while permitting interpretation on what is usually a more familiar scale. This approach may also be used with linear models, for example to display effects on the scale of the response even if the data are analyzed on a transformed scale, such as log or square-root. See the `axes` argument details below to change the scale to response scale, or to linear predictor scale with tick marks labeled in response scale.

When a factor is on the x-axis, the `plot` method for `eff` objects connects the points representing the effect by line segments, creating a response "profile." If you wish to suppress these lines, add `lty=0` to the `lines` argument to the call to `plot` (see below and the examples).

In a polytomous multinomial or proportional-odds logit model, by default effects are plotted on the probability scale; they may alternatively be plotted on the scale of the individual-level logits.

All of the arguments to plot objects created by `Effect` or `allEffects` can also be used with objects created by `predictorEffect` or `predictorEffects`.

Detailed Argument Descriptions

For more information about these arguments and many examples, see the Predictor Effects Graphics Gallery vignette.

Maximizing the flexibility of these plot commands requires inclusion of a myriad of options. In an attempt to simplify the use of these options, they have been organized into just a few arguments that each accept a list of specifications as an argument. In a few cases the named entries in the list are themselves lists.

Each of the following arguments takes an optional list of specifications; any specification absent from the list assumes its default value. Some of the list elements are themselves lists, so in complex cases, the argument can take the form of nested lists. All of these arguments can also be used on objects created with `predictorEffects`.

symbols TRUE, FALSE, or a list of options that controls the plotting symbols and their sizes for use with factors; if FALSE symbols are suppressed; if TRUE default values are used:

   pch ploting symbols, a vector of plotting characters, with the default taken from `trellis.par.get("superpose.symb` typically a vector of 1s (circles).

cex plotting character sizes, a vector of values, with the default taken from `trellis.par.get("superpose.symbol")$`
typically a vector of 0.8s.

lines TRUE, FALSE, or a list that controls the characteristics of lines drawn on a plot, and also
whether or not multiple lines should be drawn in the same panel in the plot; if FALSE lines are
suppressed; if TRUE default values are used:

multiline display a multiline plot in each panel; the default is TRUE if there are no standard
errors in the "eff" object, FALSE otherwise. For an "effpoly" object multline=TRUE
causes all of the response levels to be shown in the same panel rather than in separate
panels.

z.var for linear, generalized linear or mixed models, the index (number) or quoted name of
the covariate or factor for which individual lines are to be drawn in each panel of the
effect plot. The default is the predictor with the smallest number of levels or values. This
argument is only used for multipline plots.

lty vector of line types, with the default taken from `trellis.par.get("superpose.line")$lty`,
typically a vector of 1s (solid lines).

lwd vector of line widths, with the default taken from `trellis.par.get("superpose.line")$lwd`,
typically a vector with 2 in the first position followed by 1s.

col a vector of line colors, with the default taken from from `trellis.par.get("superpose.line")$col`,
used both for lines and for areas in stacked area plots for "effpoly" objects; in the latter
case, the default colors for an ordered response are instead generated by sequential_hcl
in the **colorspace** package.

splines use splines to smooth plotted effect lines; the default is TRUE.

axes a list with elements x, y, alternating, and grid that control axis limits, ticks, and labels.
The x and y elements may themselves be lists.

The x entry is a list with elements named for predictors, with each predictor element itself a
list with the following elements:

lab axis label, defaults to the name of the predictor; may either be a text string or a list with
the text label (optionally named label) as its first element and the named element cex as
its second element.

lim a two-element vector giving the axis limits, with the default determined from the data.

ticks a list with either element at, a vector specifying locations for the ticks marks, or n, the
number of tick marks.

transform transformations to be applied to the horizontal axis of a numeric predictor, in the
form of a list of two functions, with element names trans and inverse. The trans
function is applied to the values of the predictor, and inverse is used for computing
proper axis tick labels. The default is not to transform the predictor axis.

Two additional elements may appear in the x list, and apply to all predictors:

rotate angle in degrees to rotate tick labels; the default is 0.

rug display a rug plot showing the marginal distribution of a numeric predictor; the default is
TRUE.

The y list contains lab, lim, ticks, and rotate elements (similar to those specified for in-
dividual predictors in the x list), along with the additional type, transform, and style ele-
ments:

type for plotting linear or generalized linear models, "rescale" (the default) plots the verti-
cal axis on the link scale (e.g., the logit scale for a logit model) but labels the axis on the

response scale (e.g., the probability scale for a logit model); "response" plots and labels the vertical axis on the scale of the response (e.g., the probability scale for a logit model); and "link" plots and labels the vertical axis on the scale of the link (e.g., the logit scale for a logit model). For polytomous logit models, this element is either "probability" or "logit", with the former as the default.

transform primarily for linear or linear mixed models, this argument is used to apply an arbitrary transformation to the vertical axis. For example, if fitting a linear model with response log(y), then setting transform=exp would plot exp(log(y)) = y on the vertical axis. If the response were 1/y, then use transform=function(yt) 1/yt, since the reciprocal is its own inverse. The transform argument can also be a list of two functions. For example with a response log(y), the specification transform=list(trans=log, inverse=log), type="rescale" will plot in log-scale, but will label tick marks in arithmetic scale; see the example below. The specification transform=list(trans=log, inverse=exp), type="response" is equivalent to transform=exp. When type="response" the lab argument will geneally be used to get a label for the axis that matches the untransformed response. If this argument is used with a generalized linear model or another model with a non-identity link function, the function is applied to the linear predictor, and will probably not be of interest.

style for polytomous logit models, this element can take on the value "lines" (the default) or "stacked" for line plots or stacked-area plots, respectively.

Other elements:

alternating if TRUE (the default), the tick labels alternate by panels in multi-panel displays from left to right and top to bottom; if FALSE, tick labels appear at the bottom and on the left.

grid if TRUE (the default is FALSE), add grid lines to the plot.

confint specifications to add/remove confidence intervals or regions from a plot, and to set the nominal confidence level.

style one of "auto", "bars", "lines", "bands", and "none"; the default is "bars" for factors, "bands" for numeric predictors, and "none" for multiline plots; "auto" also produces "bars" for factors and "bands" for numeric predictors, even in multiline plots.

alpha transparency of confidence bands; the default is 0.15.

col colors; the default is taken from the line colors.

partial.residuals specifications concerning the addition of partial residuals to the plot.

plot display the partial residuals; the default is TRUE if residuals are present in the "eff" object, FALSE otherwise.

fitted show fitted values as well as residuals; the default is FALSE.

col color for partial residuals; the default is the second line color.

pch plotting symbols for partial residuals; the default is 1, a circle.

cex size of symbols for partial residuals; the default is 1.

smooth draw a loess smooth of the partial residuals; the default is TRUE.

span span for the loess smooth; the default is 2/3.

smooth.col color for the loess smooth; the default is the second line color.

lty line type for the loess smooth; the default is the first line type, normally 1 (a solid line).

lwd line width for the loess smooth; the default is the first line width, normally 2.

id specifications for optional point identification when partial residuals are plotted.

> n number of points to identify; default is 2 if id=TRUE and 0 if id=FALSE. Points are selected based on the Mahalanobis distances of the pairs of x-values and partial residuals from their centroid.
>
> col color for the point labels; default is the same as the color of the partial residuals.
>
> cex relative size of text for point labels; default is 0.75.
>
> labels vector of point labels; the default is the names of the residual vector, which is typically the row names of the data frame to which the model is fit.

lattice the plots are drawn with the **[lattice](#)** package, generally by the [xyplot](#) function. These specifications are passed as arguments to the functions that actually draw the plots.

> layout the layout argument to the **lattice** function [xyplot](#) (or, in some cases [densityplot](#)), which is used to draw the effect display; if not specified, the plot will be formatted so that it appears on a single page.
>
> key.args a key, or legend, is added to the plot if multiline=TRUE. This argument is a list with components that determine the the placement and other characteristics of the key. The default if not set by the user is key.args = list(space="top", columns=2, border=FALSE, fontfamily="serif", cex.title=.80, cex=0.75). If there are more than 6 groups in the plot, columns is set to 3. For stacked-area plots, the default is a one-column key. In addition to the arguments shown explicitly below, any of the arguments listed in the [xyplot](#) documentation in the key section can be used.
>
>> space determines the placement of the key outside the plotting area, with default space="above" for above the plot and below its title. Setting space="right" uses space to the right of the plot for the key.
>>
>> x, y, corner used to put the key on the graph itself. For example, x=.05, y=.95, corner=c(0,1) will locate the upper-left corner of the key at (.05, .95), thinking of the graph as a unit square.
>>
>> columns number of columns in the key. If space="top", columns should be 2, 3 or 4; if space="right", set columns=1.
>>
>> border if TRUE draw a border around the key; omit the border if FALSE.
>>
>> fontfamily the default is "sans" for the sans-serif font used in the rest of the plot; the alternative is "serif" for a serif font.
>>
>> cex, cex.title the default relative size of the font for labels and the title, respectively. To save space set these to be smaller than 1.
>
> strip a list with three elements: factor.names, which if TRUE, the default, shows conditioning variable names in the panel headers; values, which if TRUE, the default unless partial residuals are plotted, displays conditioning variable values in the panel headers, and cex, the relative size of the text displayed in the strip.
>
> array a list with elements row, col, nrow, ncol, and more, used to graph an effect as part of an array of plots; row, col, nrow, and ncol are used to compose the split argument and more the more argument to [print.trellis](#). The array argument is automatically set by plot.efflist and will be ignored if used with that function.

Value

The summary method for "eff" objects returns a "summary.eff" object with the following components (those pertaining to confidence limits need not be present):

header	a character string to label the effect.
effect	an array containing the estimated effect.
lower.header	a character string to label the lower confidence limits.
lower	an array containing the lower confidence limits.
upper.header	a character string to label the upper confidence limits.
upper	an array containing the upper confidence limits.

The plot method for "eff" objects returns a "plot.eff" object (an enhanced "trellis" object);
the provided print method plots the object.

The [method for "efflist" objects is used to subset an "efflist" object and returns an object
of the same class.

Author(s)

John Fox <jfox@mcmaster.ca> and Jangman Hong.

See Also

LegacyArguments, effect, allEffects, effectsTheme, xyplot, densityplot, print.trellis,
loess, sequential_hcl, and the Predictor Effects Graphics Gallery and Effect Displays with Par-
tial Residuals vignettes.

Examples

```
# also see examples in ?effect

# plot predictorEffects
mod <- lm(prestige ~ education + log(income)*type + women, Prestige)
plot(predictorEffects(mod, ~ income), axes=list(grid=TRUE))
plot(predictorEffects(mod, ~ income), lines=list(multiline=TRUE),
                                              axes=list(grid=TRUE))
plot(predictorEffects(mod, ~ type), lines=list(multiline=TRUE),
                                              axes=list(grid=TRUE),
                                              confint=list(style="bars"))

mod.cowles <- glm(volunteer ~ sex + neuroticism*extraversion,
                  data=Cowles, family=binomial)
eff.cowles <- allEffects(mod.cowles, xlevels=list(extraversion=seq(0, 24, 6)))
eff.cowles
as.data.frame(eff.cowles[[2]]) # neuroticism*extraversion interaction

plot(eff.cowles, 'sex', axes=list(grid=TRUE,
                                  y=list(lab="Prob(Volunteer)"),
                                  x=list(rotate=90)),
                        lines=list(lty=0))

plot(eff.cowles, 'neuroticism:extraversion',
     axes=list(y=list(lab="Prob(Volunteer)",
        ticks=list(at=c(.1,.25,.5,.75,.9)))))
```

```
plot(Effect(c("neuroticism", "extraversion"), mod.cowles,
            se=list(type="Scheffe"),
            xlevels=list(extraversion=seq(0, 24, 6))),
     axes=list(y=list(lab="Prob(Volunteer)",
         ticks=list(at=c(.1,.25,.5,.75,.9)))))


    # change color of the confidence bands to 'black' with .15 transparency
plot(eff.cowles, 'neuroticism:extraversion',
     axes=list(y=list(lab="Prob(Volunteer)",
                      ticks=list(at=c(.1,.25,.5,.75,.9)))),
     confint=list(col="red", alpha=.3))

plot(eff.cowles, 'neuroticism:extraversion',
     lines=list(multiline=TRUE),
     axes=list(y=list(lab="Prob(Volunteer)")),
     lattice=list(key.args = list(x = 0.65, y = 0.99, corner = c(0, 1))))

# use probability scale in place of logit scale, all lines are black.
plot(eff.cowles, 'neuroticism:extraversion',
     lines=list(multiline=TRUE, lty=1:8, col="black"),
     axes=list(y=list(type="response", lab="Prob(Volunteer)")),
     lattice=list(key.args = list(x = 0.65, y = 0.99, corner = c(0, 1))),
     confint=list(style="bands"))

plot(effect('sex:neuroticism:extraversion', mod.cowles,
            xlevels=list(extraversion=seq(0, 24, 6))),
     lines=list(multiline=TRUE))

plot(effect('sex:neuroticism:extraversion', mod.cowles,
            xlevels=list(extraversion=seq(0, 24, 6))),
     lines=list(multiline=TRUE),
     axes=list(y=list(type="response")),
     confint=list(style="bands"),
     lattice=list(key.args = list(x=0.75, y=0.75, corner=c(0, 0))))


if (require(nnet)){
    mod.beps <- multinom(vote ~ age + gender + economic.cond.national +
                             economic.cond.household + Blair + Hague + Kennedy +
                             Europe*political.knowledge, data=BEPS)

    plot(effect("Europe*political.knowledge", mod.beps,
                xlevels=list(political.knowledge=0:3)))


    plot(effect("Europe*political.knowledge", mod.beps,
                xlevels=list(political.knowledge=0:3),
                fixed.predictors=list(given.values=c(gendermale=0.5))),
         axes=list(y=list(style="stacked"), x=list(rug=FALSE), grid=TRUE),
         lines=list(col=c("blue", "red", "orange")))
}
```

```
if (require(MASS)){
    mod.wvs <- polr(poverty ~ gender + religion + degree + country*poly(age,3),
                    data=WVS)
    plot(effect("country*poly(age, 3)", mod.wvs))


    plot(effect("country*poly(age, 3)", mod.wvs), lines=list(multiline=TRUE))
    plot(effect("country*poly(age, 3)", mod.wvs),
        axes=list(y=list(style="stacked")),
        lines=list(col=c("gray75", "gray50", "gray25")))

    plot(effect("country*poly(age, 3)", latent=TRUE, mod.wvs))

}

mod.pres <- lm(prestige ~ log(income, 10) + poly(education, 3) + poly(women, 2),
               data=Prestige)
eff.pres <- allEffects(mod.pres)

plot(eff.pres)
plot(eff.pres[1:2])

plot(eff.pres[1],
     axes=list(x=list(income=list(transform=list(
         trans=log10, inverse=function(x) 10^x),
         ticks=list(at=c(1000, 2000, 5000, 10000, 20000))))))

mod <- lm(log(prestige) ~ income:type + education, data=Prestige)
p1 <- predictorEffects(mod, ~ income)
# log-scale for response
plot(p1, lines=list(multiline=TRUE))
# log-scale, with arithmetic tick marks
plot(p1, lines=list(multiline=TRUE),
     axes=list(y=list(transform=list(trans=log, inverse = exp),
                      lab="prestige", type="rescale")))
# arithmetic scale and tick marks, with other arguments
plot(p1, lines=list(multiline=TRUE), grid=TRUE,
     lattice=list(key.args=list(space="right", border=TRUE)),
     axes=list(y=list(transform=exp, lab="prestige")))
```

---

predictorEffects   *Functions For Computing Predictor Effects*

Description

Alternatives to the Effect and allEffects functions that use a different paradigm for conditioning in an effect display. The user specifies one predictor, either numeric or a factor (where character and logical variables are treated as factors), for the horizontal axis of a plot, and the function determines the appropriate plot to display (which is drawn by plot). See the vignette Predictor Effects Graphics Gallery for details and examples.

Usage

```
predictorEffect(predictor, mod, focal.levels=50, xlevels=5, ...)

## S3 method for class 'poLCA'
predictorEffect(predictor, mod, focal.levels=50,
    xlevels=5, ...)

## S3 method for class 'svyglm'
predictorEffect(predictor, mod, focal.levels=50,
    xlevels=5, ...)

## Default S3 method:
predictorEffect(predictor, mod, focal.levels=50,
    xlevels=5, ..., sources)

predictorEffects(mod, predictors, focal.levels=50, xlevels=5, ...)

## S3 method for class 'poLCA'
predictorEffects(mod, predictors = ~ .,
    focal.levels=50, xlevels=5, ...)

## Default S3 method:
predictorEffects(mod, predictors = ~ .,
    focal.levels=50, xlevels=5,  ..., sources)
```

Arguments

mod	A model object. Supported models include all those described on the help page for [Effect].
predictor	quoted name of the focal predictor.
predictors	If the default, ~ ., a predictor effect plot is drawn for each predictor (not regressor) in a model. Otherwise, this is a one-sided formula specifying the first-order predictors for which predictor effect plots are to be drawn.
focal.levels	for predictorEffect, the number of evenly-spaced values (the default is 50) for the numeric focal predictor or a vector of values for the focal predictor.
	For predictorEffects, the number of evenly-spaced values (default 50) to use for each numeric focal predictor in turn, or a named list, similar to xlevels, giving the number of values or the values themselves for each predictor individually, to be used when that predictor is the focal predictor; if a numeric focal predictor doesn't appear in the list, the default of 50 values is used.
xlevels	this argument is used to set the levels of conditioning predictors; it may either be a single number specifying the number of evenly-spaced values (the default is 5) to which each conditioning predictor is to be set, or it may be a list with elements named for the predictors giving the number of values or a vector of values to which each conditioning predictor is to be set, as explained in the help for [Effect].

> If the focal predictor is included in the xlevels list, it is disregarded; if any conditioning predictor is omitted from the list, its number of values is set to 5. The default behavior of xlevels is different when residuals=TRUE; in that case, it behaves as in Effect.lm, and is effectively set by default to the 0.2, 0.4, 0.6, and 0.8 quantiles of conditioning predictors.
>
> The xlevels argument works similarly for predictorEffect and predictorEffects.

...   Additional arguments passed to Effect.

sources   Provides a mechanism for applying predictorEffect methods to a variety of regression models; see the vignette Regression Models Supported by the effects Package for an explanation.

Details

Effect plots view a fitted regression function E(Y|X) in (sequences of) two-dimensional plots using conditioning and slicing. The functions described here use a different method of determining the conditioning and slicing than allEffects uses. The predictor effect of a focal predictor, say x1, is the usual effect for the generalized interaction of x1 with all the other predictors in a model. When a predictor effect object is plotted, the focal predictor is by default plotted on the horizontal axis.

For example, in the model mod with formula y ~ x1 + x2 + x3, the predictor effect p1 <- predictorEffects(mod, ~ x1) is essentially equivalent to p2 <- Effect("x1", mod). When plotted, these objects may produce different graphs because plot(p1) will always put x1 on the horizontal axis, while plot(p2) uses a rule to determine the horizontal axis based on the characteristics of all the predictors, e.g., preferring numeric predictors over factors.

If mod has the formula y ~ x1 + x2 + x3 + x1:x2, then p1 <- predictorEffects(mod, ~ x1) is essentially equivalent to p2 <- Effect(c("x1", "x2"), mod). As in the last example, the plotted versions of these objects may differ because of different rules used to determine the predictor on the horizontal axis.

If mod has the formula y ~ x1 + x2 + x3 + x1:x2 + x1:x3, then p1 <- predictorEffects(mod, ~ x1) is essentially equivalent to p2 <- Effect(c("x1", "x2", "x3"), mod). Again, the plotted versions of these objects may differ because of the rules used to determine the horizontal axis.

Value

predictorEffect returns an object of class c("predictoreff", "eff"). The components of the object are described in the help for Effect; predictorEffects returns an object of class "predictorefflist", which is a list whose elements are of class c("predictoreff", "eff").

Author(s)

S. Weisberg <sandy@umn.edu> and J. Fox

References

See Effect.

See Also

Effect, plot.predictoreff, the Predictor Effects Graphics Gallery vignette, and the Effect Displays with Partial Residuals vignette.

Examples

```
mod <- lm(prestige ~ type*(education + income) + women, Prestige)
plot(predictorEffect("income", mod))
plot(predictorEffects(mod, ~ education + income + women))

mod.cowles <- glm(volunteer ~ sex + neuroticism*extraversion, data=Cowles, family=binomial)
plot(predictorEffects(mod.cowles, xlevels=4))
plot(predictorEffect("neuroticism", mod.cowles, xlevels=list(extraversion=seq(5, 20, by=5))),
     axes=list(grid=TRUE,
               x=list(rug=FALSE),
               y=list(lab="Probability of Vounteering")),
     lines=list(multiline=TRUE),
     type="response")
predictorEffects(mod.cowles, focal.levels=4, xlevels=4)

# svyglm() example (adapting an example from the survey package)

if (require(survey)){
  data(api)
  dstrat<-svydesign(id=~1, strata=~stype, weights=~pw,
    data=apistrat, fpc=~fpc)
  mod <- svyglm(sch.wide ~ ell + meals + mobility, design=dstrat,
    family=quasibinomial())
  plot(predictorEffects(mod),
    axes=list(y=list(lim=log(c(0.4, 0.99)/c(0.6, 0.01)),
      ticks=list(at=c(0.4, 0.75, 0.9, 0.95, 0.99)))))
}
```

---

summary.eff   *Summarizing and Printing Effects*

Description

summary, print, and as.data.frame methods for objects created using the effects package.

Usage

```
## S3 method for class 'eff'
print(x, type=c("response", "link"), ...)
## S3 method for class 'effpoly'
print(x, type=c("probability", "logits"), ...)
## S3 method for class 'efflatent'
print(x, ...)
## S3 method for class 'efflist'
print(x, ...)
## S3 method for class 'mlm.efflist'
print(x, ...)
## S3 method for class 'summary.eff'
```

```
print(x, ...)
## S3 method for class 'eff'
summary(object, type=c("response", "link"), ...)
## S3 method for class 'effpoly'
summary(object, type=c("probability", "logits"), ...)
## S3 method for class 'efflatent'
summary(object, ...)
## S3 method for class 'efflist'
summary(object, ...)
## S3 method for class 'mlm.efflist'
summary(object, ...)
## S3 method for class 'eff'
as.data.frame(x, row.names=NULL, optional=TRUE,
        type=c("response", "link"), ...)
## S3 method for class 'efflist'
as.data.frame(x, row.names=NULL, optional=TRUE, type, ...)
## S3 method for class 'effpoly'
as.data.frame(x, row.names=NULL, optional=TRUE, ...)
## S3 method for class 'efflatent'
as.data.frame(x, row.names=NULL, optional=TRUE, ...)
## S3 method for class 'eff'
vcov(object, ...)
```

Arguments

x, object   an object consisting of fitted values and other information needed to draw effects
                  plots that is produced by functions in the `effects` package.

type   fitted values are by default printed by these functions in the `"response"` scale.
                  For models with a link function like a GLM, fitted values in the linear predictor
                  scale are obtained by setting `type="link"`. For polytomous response models
                  setting `type="logits"` returns fitted values in the logit scale.

row.names, optional
                  arguments to `as.data.frame` not used by these methods.

...   other arguments passed on

Value

The `print` methods return the fitted values in tables. The `summary` methods return the fitted values
and 95 percent confidence intervals, also in tables. The `as.data.frame` method returns fitted val-
ues, standard errors, and 95 percent confidence intervals as a data frame, or as a list of data frames
for the `efflist` method. The `vcov` method returns the covariance matrix of the fitted values.

Author(s)

John Fox <jfox@mcmaster.ca> and Jangman Hong.

Examples

```
mod.cowles <- glm(volunteer ~ sex + neuroticism*extraversion,
                  data=Cowles, family=binomial)
eff.cowles <- predictorEffects(mod.cowles)
print(eff.cowles)
print(eff.cowles[["neuroticism"]], type="link")
summary(eff.cowles[["neuroticism"]], type="link")
as.data.frame(eff.cowles)
# covariance matrix of fitted values in linear predictor scale
vcov(eff.cowles[[1]])
```

Index