Package 'effects'

September 15, 2017

Version 4.0-0

Date 2017-09-14

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

Depends R (>= 3.2.0), carData

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

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

LazyLoad yes

LazyData yes

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, http://socserv.socsci.mcmaster.ca/jfox/

VignetteBuilder knitr

NeedsCompilation no

Author John Fox [aut, cre],
Sanford Weisberg [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 2017-09-15 20:40:30 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:	4.0-0
Date:	2017-09-14
Depends:	R (>= 3.2.0), carData
Suggests:	pbkrtest (>= 0.4-4), nlme, MASS, poLCA, heplots, splines, ordinal, car
Imports:	lme4, nnet, lattice, grid, colorspace, graphics, grDevices, stats, utils
LazyLoad:	yes
LazyData:	yes
License:	GPL (>= 2)
URL:	https://www.r-project.org, http://socserv.socsci.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 <jfox@mcmaster.ca>, Sanford Weisberg, Michael Friendly, and Jangman Hong. We are grateful to Robert Andersen and David Firth for various suggestions.

Maintainer: John Fox <jfox@mcmaster.ca>

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, <http://www.jstatsoft.org/v08/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, <http://www.jstatsoft.org/v32/i01/>.

Fox, J. and S. Weisberg (forthcoming). Visualizing Fit and Lack of Fit in Comples Regression Models: Effect Plots with Partial Residuals. *Journal of Statistical Software*.

---

effect	*Functions For Constructing Effect Displays*

Description

`Effect` and `effect` construct an `"eff"` object for a term (usually a high-order term) in a linear model (fit by `lm` or `gls`) or generalized linear model (fit by `glm`), or an `"effpoly"` object for a term in a multinomial or proportional-odds logit model (fit respectively by `multinom` or `polr`), absorbing the lower-order terms marginal to the term in question, and averaging over other terms in the model. For multivariate linear models (of class `"mlm"`, fit by `lm`), the function constructs a list of `"eff"` objects separately for the various response variables.

`effect` builds the required object by specifying explicity a focal term like `"a:b"` for an a by b interaction. `Effect` specifies the predictors in the 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`.

The `Effect` and `effect` functions can also be used with some mixed-effects models, including linear and generalized linear mixed-effects models fit by `lmer` and `glmer` from the **lme4** package and `lme` from the **nlme** package; ordinal logit mixed effects fit with `clm2` or `clmm` from the **ordinal** package, polytomous latent-class models fit by the `poLCA` function in the **poLCA** package, and generalized linear models fit to data from complex surveys using the `svyglm` function in the **survey** package. The displays for mixed-effects models are for the fixed effects only, not for random effects.

There is a default method for `Effect` that should work with most model objects that have a linear predictor and that respond to the `coef`, `model.frame`, `formula`, and `vcov` functions.

`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 type `"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, ...)

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

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

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

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

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

## S3 method for class 'clm2'
Effect(focal.predictors, mod, ...)

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

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

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

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

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

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

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

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

## Default S3 method:
Effect(focal.predictors, mod, xlevels = list(),
    fixed.predictors, vcov. = vcov, confint=TRUE,
    transformation = list(link = I, inverse = I), ...,
    #legacy arguments:
    se, confidence.level, given.values, typical, offset)

allEffects(mod, ...)


## S3 method for class 'eff'
as.data.frame(x, row.names=NULL, optional=TRUE,
    transform=x$transformation$inverse, ...)

## 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

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.

focal.predictors

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

mod   an object of class ″lm″, ″gls″, ″glm″, ″multinom″, ″polr″, ″mer″ (or ″merMod″), ″lme″, or ″poLCA″.

xlevels   this argument is used to set the number of levels for any focal predictor that is not a factor. If xlevels=NULL, then each numeric predictor is represented by five values equally spaced over its range 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, 7),   x2=5) would use the values 2, 4 and 7 for the levels of x1, use 5 equally spaced levels for the levels of 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 a numeric vector of named elements, setting particular columns of the model matrix to specific values for predictors that are *not* focal predictors; if specified, these takes precedence over the application of the function given in the typical list element (below). Care must be taken in specifying these values — e.g., for a factor, the values of all contrasts should be given and these should be consistent with each other.

        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.   A function or the name of a function that will be used to get the estimated variance-covariance matrix of the estimated coefficients. This will ordinarily be the default, [vcov](#), which will result in the function call vcov(mod) to get the variance-covariance matrix. You can use the name of any function that takes

	the model object as its first argument and returns an estimated sample covariance matrix, such as the hccm function in the **car** package, which returns a heteroscedasticity corrected estimate for a linear model.
confint	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.
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 in the **lme4** package. The default is FALSE because the computation can be very slow.
transformation	a two-element list with elements link and inverse. For a generalized linear model, these are by default the link function and inverse-link (mean) function. For a linear model, these default to NULL. If NULL, the identity (or inhibit) function, [I](I), is used; this effect can also be achieved by setting the argument to NULL. The inverse-link may be used to transform effects when they are printed or plotted; the link may be used in positioning axis labels (see below). If the link is not given, an attempt will be made to approximate it from the inverse-link.
partial.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. This argument may also be used for mixed-effects models.
quantiles	quantiles at which to evaluate numeric focal predictors *not* on the horizontal axis, used only when partial residuals are displayed; superceded if the xlevels argument gives specific values for a predictor.
x.var	the 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 partial.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".
transform	a transformation to be applied to the effects and confidence limits, by default taken from the inverse link function saved in the "eff" object.
row.names, optional	not used.
response	for an "mlm" object, a vector containing the name(s) or indices of one or more response variable(s). The default is to use all responses in the model.

object	an object of class `"eff"` for which the covariance matrix of the effects is desired.
...	arguments to be passed down.
se, confidence.level, given.values, typical, offset	
	legacy arguments retained for backwards compatability; if present, these arguments take precedence of the `compute` and `level` elements over the `confint` list argument and the `given.values`, `typical`, and `offset` elements of the `fixed.predictors` list 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. Plots are drawn using 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.

The `as.data.frame` methods convert effect objects to data frames to facilitate the construction of custom displays. In the case of `"eff"` objects, the `se` element in the data frame is always on the scale of the linear predictor, and the transformation used for the fit and confidence limits is saved in a `"transformation"` attribute.

See predictorEffects for an alternative paradigm for getting effects.

Value

For `lm`, `glm`,`svyglm`, `mer` and `lme`, `effect` and `Effect` return an `"eff"` object, and for `multinom`, `polr`, `clm`, `clmm` and `clm2`, 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 ravelled 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, to be used by plot.eff to 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.

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.

Factors cannot 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").

The functions in the **effects** package work properly with predictors that are numeric or factors; consequently, e.g., convert logical predictors to factors, and dates to numeric.

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 apparently 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. The effect methods can also be used with objects created using clm or clm2 fitting ordinal response logit models with no random effects, with results similar to those from polr in the **MASS** package.

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 for 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, <http://www.jstatsoft.org/v08/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, <http://www.jstatsoft.org/v32/i01/>.

Fox, J. and S. Weisberg (forthcoming). Visualizing Fit and Lack of Fit in Complex Regression Models: Effect Plots with Partial Residuals. *Journal of Statistical Software*.

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.

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,
            confint=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,
         confint=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, partial.residuals=TRUE)) # standard C+R plots
plot(allEffects(mod.prestige.1, partial.residuals=TRUE,
    confint=list(type="scheffe"))) # with Scheffe-type confidence bands

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

mod.prestige.3 <- lm(prestige ~ type + income*education, data=Prestige)
plot(Effect(c("income", "education"), mod.prestige.3, partial.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, partial.residuals=TRUE)) # correct model
plot(Effect(c("x1", "x2"), mod.2, partial.residuals=TRUE)) # wrong model
plot(Effect(c("x1", "x2"), mod.3, partial.residuals=TRUE)) # wrong model
```

| 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))))
```

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.

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 plots methods:

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

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 tran

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")

se=TRUE/FALSE for Effect, confint=list(compute=TRUE/FALSE)

confidence.level={number} for Effect, confint=list(level={number})
```

```
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:

```
se=TRUE/FALSE confint=list(compute=TRUE/FALSE)

confidence.level={number} confint=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})
```

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

Effect, plot.eff, plot.effpoly

---

plot.predictoreff　　　　　*Draw Predictor Effect Plots*

Description

These functions call `plot.eff` and `plot.efflist` to draw predictor effect plots.

Usage

```
## 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, ...)
```

Arguments

x	An object of class `predictoreff` or `predictorefflist`.
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.
main	the title for the plot, printed at the top; the default title is constructed from the name of the effect.
...	arguments to be passed to `plot.eff` or `plot.efflist`.
selection	the optional index (number) or quoted name of the effect in an effect list 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.
lattice	argument passed to `plot.efflist`.

Details

The `plot.predictoreff` calls the method `plot.eff` and `plot.predictorefflist` calls `plot.efflist`. Both of these functions are documented at `plot.eff`. Warning: By default, the functions documented here use the argument `lines=list(multiline=TRUE)` while direct calls to the underlying functions use `lines=list(multiline = FALSE)` if standard errors were computed by the call to create the object `x`.

Value

See the documentation for `plot.eff`.

Author(s)

S. Weisberg, <sandy@umn.edu>

See Also

`predictorEffect`, `plot.eff`.

Examples

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

---

predictorEffects   *Functions For Computing Predictor Effects*

Description

Alternatives to the `Effect` and `allEffects` functions that use a different paradigm for conditioning in an effects display. The user specifies one predictor, either continuous or a factor, for the horizontal axis of a plot, and the function determines the appropriate plot to display (which is drawn by `plot`).

Usage

```
predictorEffect(predictor, mod, xlevels, ...)

## S3 method for class 'svyglm'
predictorEffect(predictor, mod, xlevels, ...)

## Default S3 method:
predictorEffect(predictor, mod, xlevels=list(), ...)

predictorEffects(mod, predictors = ~., ...)
```

Arguments

mod	A model object. Supported models include all those described on the help page for [Effect].
predictor	quoted name of the focal predictor
xlevels	this argument is used to set the values for any predictor in the effect that is not a factor. For a predictor effect, the default is to use 50 quantiles of the focal predictor on the x-axis between the 0.01 and 0.98 quantiles. See [Effect] for details and for how other defaults are set.
predictors	If the default ~ ., a predictor effecrts plot is drawn for each predictor (not regressor) in a model. Otherwise, this should be a one-sided formula listing the first-order predictors for which predictor effects plots should be drawn.
...	Additional arguments passed to [Effect].

Details

Effects plots view a fitted regression function E(Y|X) in (sequences of) two-dimensional plots using conditioning and slicing. The functions describe here use a different method of determining the conditioning and slicing than Effects uses. The predictor effects a focal predictor say x1 will be the the usual effect for the generalized interaction of x1 with all the other predictors in a model. When a predictor effects 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, then p1 <- predictorEffects(mod, ~ x1) is essentially equilavent to p2 <- Effect("x1", mod). When plotted, these objects may be different 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, preferring continuous predictors over factors.

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

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

Value

predictorEffect returns an object of class c(predictoreff, eff). The components of the object are described under the detalis at [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>

References

See [Effect].

See Also

Effect, plot.predictoreff

Examples

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

# 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, Printing, and Plotting Effects*

Description

summary, print, plot, and [methods for eff, effpoly, efflist, and mlm.efflist objects. The plot arguments were substantially changed in mid-2017.

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'
plot(x, x.var, z.var=which.min(levels),
    main=paste(effect, "effect plot"),
    symbols=TRUE, lines=TRUE, axes, confint,
    partial.residuals, id, lattice, ...,
    # legacy arguments:
    multiline, 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 '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 'efflist'
plot(x, selection, rows, cols, ask=FALSE, graphics=TRUE, lattice, ...)
## S3 method for class 'mlm.efflist'
plot(x, ...)
```

Arguments

x	an object of class "eff", "effpoly", "efflist", "mlm.efflist", or "summary.eff", as appropriate.
object	an object of class "eff", "effpoly", "efflist", or "mlm.efflist", as appropriate.
type	for printing or summarizing linear and generalized linear models, if "response" (the default), effects are printed on the scale of the response variable; if "link", effects are printed on the scale of the linear predictor. For polytomous logit models, this argument takes either "probability" or "logit", with the former as the default. The type argument is also a legacy argument for plot methods.

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.

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 multiline plots — see the
                  lines argument.

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 gen-
                  eralized linear models; if not given, generally reasonable default values are used.
                  See Detailed Argument Descriptions under Details for more information.

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 De-
                  tailed Argument Descriptions under Details for more information.

selection   the optional index (number) or quoted name of the effect in an effect list 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 com-
                  puted by the plot method.

ask   if selection is not supplied and ask is TRUE, a menu of high-order terms is pre-
                  sented; 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.

multiline, 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

In a generalized linear model, by default, the `print` and `summary` methods for `eff` objects print the
computed effects on the scale of the response variable using the inverse of the link function. In a
logit model, for example, this means that the effects are expressed on the probability scale.

By default, 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.

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.

Detailed Argument Descriptions

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.symbo`
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 chacteristics 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 sup-
pressed; 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 multiline=TRUE
causes all of the response levels to be shown in the same panel rather than in separate
panels.

    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.

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 individual predictors in the x list), along with the additional type element:

type for plotting linear or generalized linear models, "rescale" (the default) plots the vertical 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.

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 additional arguments that control the appearance or location of the key, or legend, on the plot. There are many options, and they are documented by searching for "key:" in [xyplot](#). For example, lattice=list(key.args=list(cex= .75, cex.title=.75, between.columns=0)) reduces the size of the labels and title to .75 of nominal, and reduces the extra space between columns to 0, and lattice=list(key.args = list(x = 0.75, y = 0.75, corner = c(0, 0))) moves the key onto the plot, with the origin corner=c(0,0) at the point (.75, .75), thinking of the total plotting area as a unit square. To conserve space, by default we set between.columns=0.

   strip a list with two elements: factor.names, which if TRUE, the default, shows conditioning variable names in the panel headers; and values, which if TRUE, the default unless partial residuals are plotted, displays conditioning variable values in the panel headers.

   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 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

Examples

```
# also see examples in ?effect

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(y=list(lab="Prob(Volunteer)"),
                                  x=list(rotate=90)),
     lines=list(lty=0), grid=TRUE)

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,
            confint=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 = 0.75, y = 0.75, corner = c(0, 0))))

# 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 = 0.75, y = 0.75, corner = c(0, 0))),
     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))))))
```

Index