

Defining Effect Methods for Other Models

John Fox and Sanford Weisberg

November 30, 2018

The `effects` package in R is designed primarily to draw graphs that visualize a fitted response surface of a fitted model in problems with a linear predictor. Many modeling paradigms that can be fit with base R or contributed packages fit into this framework, including methods for linear, multivariate linear, and generalized linear models fit by the standard `lm` and `glm` functions and by the `svyglm` function in the `survey` package (Lumley, 2004); linear models fit by generalized least squares using the `gls` function in the `nlme` package (Pinheiro et al., 2016); multinomial regression models fit by `multinom` in the `nnet` package (Venables and Ripley, 2002); ordinal regression models using `polr` from the `MASS` package (Venables and Ripley, 2002) and `c1m` and `c1m2` from the `ordinal` package (Christensen, 2015); linear and generalized linear mixed models using the `lme` function in the `nlme` package (Pinheiro et al., 2016) and the `lmer` and `glmer` functions in the `lme4` package (Bates et al., 2015); and latent class models fit by `poLCA` in the `poLCA` package (Linzer and Lewis, 2011). This is hardly an exhaustive list of fitting methods that are based on a linear predictor, and we have been asked from time to time to write functions to use `effects` with this other fitting methods. The mechanism for this is fairly simple. This vignette assumes you are familiar with R's S3 methods.

The default `Effect.default` may work with some modeling functions, as would objects of the class `gls` that we describe below in Section 1, but as illustrated in later sections you may need to modify some of the arguments that are sent to `Effect.default`.

The `effect` package has five functions that create the information needed for drawing effects plots, `Effect`, `allEffects`, `effect` and `predictorEffect` and `predictorEffects`. To add new modeling to the package only a new `Effect` needs to be written; the package will take care of all the other functions.

1 Using `effects` with Other Modeling Methods, with Generalized Least Squares in the `nlme` package as an Example

Applying `effects` to other than `lm` and `glm` objects may require writing a method for the `Effect` generic function for that type of model object. For

example, the `gls` function in the `nlme` package (Pinheiro et al., 2018) fits linear models via generalized least squares. A call to `gls` creates an object of class `gls`. The following method of `Effect` for `gls` objects finds the information needed to draw effects plots from `gls` objects:

```
Effect.gls <- function(focal.predictors, mod, ...){
  cl <- mod$call
  cl$weights <- NULL
  args <- list(
    type = "glm",
    call = cl,
    formula = formula(mod),
    family = NULL,
    coefficients = coef(mod),
    vcov = as.matrix(vcov(mod)),
    method=NULL)
  Effect.default(focal.predictors, mod, ..., sources=args)
}
```

This function simply harvests the needed information into a variable `sources`, and then passes to the result to the default `Effect` method. The three required arguments to the method are: `focal.predictors` and `mod` that match the first two arguments of `Effect.default`, and `...` that matches all other arguments. The list `sources` has up to 6 named elements and is created in the function:

type The `effects` package has three basic modeling functions: `type = "glm"`, the default, is used for functions with a univariate response and a linear predictor and possibly a link function. This class includes linear models, generalized linear models, robust regression, generalized least squares fitting, linear and generalized linear mixed effects models, and many others. The `type = "polr"` is used for ordinal regression models, as in the `polr` function in the `MASS` package, and similar methods described below in Section 6. The `type = "multinom"` for multinomial log-linear models as fit by the `multinom` function in `nnet`, and to polytomous latent class models created with the `poLCA` function in the `poLCA` package.

call The `Effect.default` method uses the `call` to harvest additional arguments that it needs. For `type="glm"`, these arguments are `formula`, `data`, `contrasts`, `subset`, `family`, `weights`, and `offset`, although only the `formula` argument is required. The `gls` function includes an optional `weights` argument that is used differently from the `weights` argument for a generalized linear model and is not needed for computing effects or predictor effects plots. In the function shown above the `call` is modified by setting `weights=NULL`.

The default for `call` is `mod$call` for S3 objects and `mod@call` for S4 objects.

formula In most cases the formula for the linear predictor is returned by `formula(mod)`, the default, but if this is not the case the value of this argument should be the value of the formula for fixed effects.

family The default is `family=NULL`. This argument is required for GLM-like models that include a `family` that specifies both an error distribution and a link function only if `family=family(mod)` is not appropriate. See the `betareg` example in Section 5 below for an example that includes a user-selected link function, but a fixed error distribution.

coefficients In many cases the (fixed-effect) coefficient estimates are returned by `coef(mod)`, the default, but if this is not the case then the value of this argument should be the estimates of the coefficients in the linear predictor. The functions in the `effects` package do not use estimates of random effects.

vcov In many cases the estimated covariance matrix of the (fixed-effect) coefficient estimates is returned by `vcov(mod)`, the default, but if this is not the case then the value of this argument should be the estimated covariance matrix of the (fixed-effect) coefficient estimates in the linear predictor.

method This argument is used only for methods that use effects graphics based on the `polr` function, where the argument `method` is the name of a link function; see `help(polr)` for a list of the accepted links, and see Section 6.1 below for an example.

The only non-default argument in `sources` is the modification of the `call` to omit weights in the call to `gls`. Without this change, there is no need to have written the `Effect.gls` method, as the default method would have worked.

```
library(effects)
```

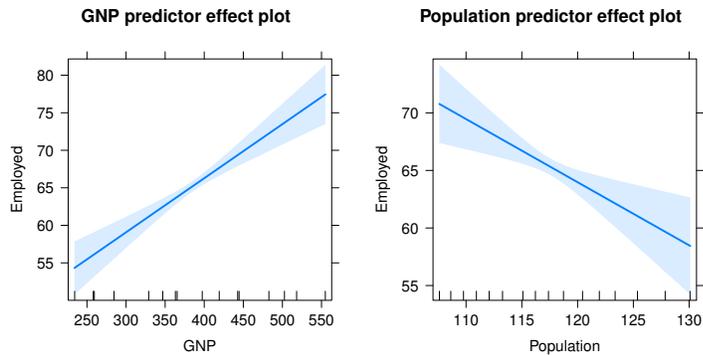
Loading required package: carData

```
lattice theme set by effectsTheme()  
See ?effectsTheme for details.
```

```
require(nlme)
```

Loading required package: nlme

```
g <- gls(Employed ~ GNP + Population,  
          correlation=corAR1(form= ~ Year), data=longley)  
plot(predictorEffects(g))
```



2 Mixed Effects with lme (nlme package)

The `lme` function in the `nlme` package (Pinheiro et al., 2018) fits linear mixed models. The required function for fitted objects from this function is included in the `effects` package. It is given by

```
print(Effect.lme)

function (focal.predictors, mod, ...)
{
  args <- list(call = mod$call, formula = mod$call$fixed, coefficients = mod$coefficients,
              vcov = mod$varFixed)
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f985cd6e5f0>
<environment: namespace:effects>
```

The `formula`, `coefficients` and `vcov` arguments are set to non-default values. The other arguments are automatically set to default values.

```
data(Orthodont, package="nlme")
m1 <- nlme::lme(distance ~ age + Sex, data=Orthodont,
               random= ~ 1 | Subject)
as.data.frame(Effect("age", m1))
```

	age	fit	se	lower	upper
1	8.0	22.04259	0.4172841	21.21520	22.86999
2	9.5	23.03287	0.3853671	22.26876	23.79698
3	11.0	24.02315	0.3741236	23.28133	24.76497
4	12.0	24.68333	0.3791619	23.93153	25.43514
5	14.0	26.00370	0.4172841	25.17631	26.83110

3 Mixed Effects with the lmer (lme4 package)

The `lme4` package (Bates et al., 2015) fits linear and generalized linear mixed effects models with the `lmer` and `glmer` functions, respectively. The same `Effect` function can be used for `lmer` and `glmer` models.

The following method is a little more complicated because it contains an additional argument `KR` to determine if the Kenward-Roger coefficient covariance matrix is to be used to compute effect standard errors. The default is `FALSE` because the computation is very slow. If `KR = TRUE`, the function also checks if the `pbkrtest` package is present.

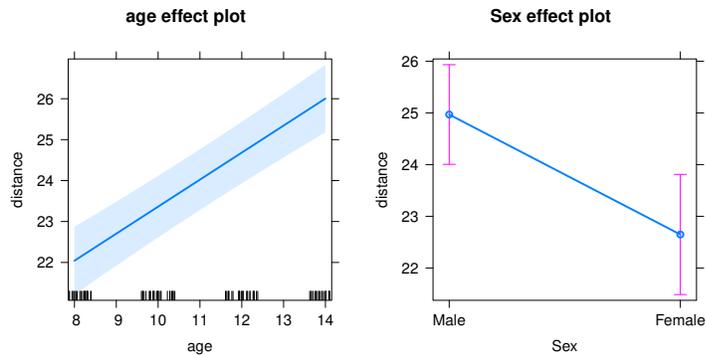
```
print(Effect.merMod)

function (focal.predictors, mod, ..., KR = FALSE)
{
  if (KR && !requireNamespace("pbkrtest", quietly = TRUE)) {
    KR <- FALSE
    warning("pbkrtest is not available, KR set to FALSE")
  }
  fam <- family(mod)
  args <- list(call = mod@call, coefficients = lme4::fixef(mod),
             family = fam, vcov = if (fam$family == "gaussian" &&
                                     fam$link == "identity" && KR) as.matrix(pbkrtest::vcovAdj(mod)) else as.matrix(vcov(mod)),
             Effect.default(focal.predictors, mod, ..., sources = args))
}
<bytecode: 0x7f985a63d2c0>
<environment: namespace:effects>
```

Because `lmer` is an S4 object, the default for `call` is `mod@call`, and this argument would have been set automatically had we not included it in the above method. The fixed-effect estimates for an object created by a call to `lmer` or `glmer` are not returned by `coef(mod)`, so the value of `coefficients` is the value returned by `lme4::fixef(mod)`. The `vcov` estimate contains its estimated variance covariance matrix of the fixed effects. The Kenward-Roger method is used to estimate the covariance matrix for linear models if the additional argument `KR=TRUE`. The default is `KR=FALSE` because The Kenward-Roger estimate requires a long computation; see `help(Effect)`.

The `formula` for a mixed-effects model in the `lme4` package specifies linear predictors for both the mean function and the variance functions, specified by, for example `(1 + age | Subject)`. The `effects` code will automatically remove any terms like these in any formula, as the effects package only displays the mean function.

```
fm2 <- lme4::lmer(distance ~ age + Sex + (1 |Subject), data
                = Orthodox)
plot(allEffects(fm2))
```



```
data(cbpp, package="lme4")
gm1 <- lme4::glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
  data = cbpp, family = binomial)
as.data.frame(predictorEffect("period", gm1))
```

	period	fit	se	lower	upper
1	1	0.19807921	0.03672693	0.13569522	0.2798569
2	2	0.08391784	0.02363110	0.04775453	0.1433443
3	3	0.07401714	0.02241762	0.04040242	0.1317591
4	4	0.04842565	0.01959184	0.02163871	0.1048199

4 Robust Linear Mixed Models (robustlmm package)

The `rlmer` function in the `robustlmm` package (Koller, 2016) fits linear mixed models with a robust estimation method. As `rlmer` closely parallels the `lmer` function, an object created by `rlmer` is easily used with `effects`:

```
print(Effect.rlmerMod)

function (focal.predictors, mod, ...)
{
  args <- list(coefficients = lme4::fixef(mod), family = family(mod))
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f985af1c820>
<environment: namespace:effects>

require(lme4)
fm3 <- robustlmm::rlmer(distance ~ age * Sex + (1 |Subject),
  data = Orthodont)
plot(predictorEffects(fm3))
```

5 Beta Regression

The `betareg` function in the `betareg` package (Grün et al., 2012) fits regressions with a link function but with Beta distributed errors.

```
print(Effect.betareg)

function (focal.predictors, mod, ...)
{
  coef <- mod$coefficients$mean
  vco <- vcov(mod)[1:length(coef), 1:length(coef)]
  fam <- binomial(link = mod$link$mean)
  fam$variance <- function(mu) {
    f0 <- function(mu, eta) (1 - mu) * mu/(1 + eta)
    do.call("f0", list(mu, mod$coefficient$precision))
  }
  fam$initialize <- expression({
    mustart <- y
  })
  args <- list(call = mod$call, formula = formula(mod), family = fam,
    coefficients = coef, vcov = vco)
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f985b2c5bc8>
<environment: namespace:effects>
```

Beta regression has a response $y \in [0, 1]$, with the connection between the mean μ of the Beta and a set for predictors \mathbf{x} through a link function $\mathbf{x}'\boldsymbol{\beta} = g(\mu)$. The variance function for the beta is $\text{var}(y) = \mu(1 - \mu)/(1 + \phi)$, for a precision parameter ϕ estimated by `betareg`.

The call to `betareg` does not have a family argument, although it does have a link stored in `mod$link$mean`. For use with `Effect.default`, the method above creates a family from the binomial family generator. It then adjusts this family by changing from binomial variance to the variance for the beta distribution. Since the `glm` function expects a variance that is a function of only one parameter, we fix the value of the precision ϕ at its estimator from the `betareg` fit, as shown in the method. We need to replace the `initialize` method in the family to one appropriate for $y \in [0, 1]$.

```
require(betareg)
```

Loading required package: betareg

```
require(lme4)
data("GasolineYield", package = "betareg")
gy_logit <- betareg(yield ~ batch + temp, data = GasolineYield)
summary(gy_logit)
```

Call:
betareg(formula = yield ~ batch + temp, data = GasolineYield)

Standardized weighted residuals 2:
Min 1Q Median 3Q Max
-2.8750 -0.8149 0.1601 0.8384 2.0483

Coefficients (mean model with logit link):

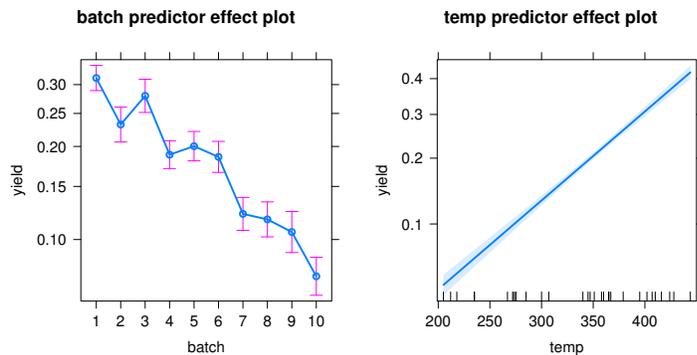
	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-6.1595710	0.1823247	-33.784	< 2e-16
batch1	1.7277289	0.1012294	17.067	< 2e-16
batch2	1.3225969	0.1179020	11.218	< 2e-16
batch3	1.5723099	0.1161045	13.542	< 2e-16
batch4	1.0597141	0.1023598	10.353	< 2e-16
batch5	1.1337518	0.1035232	10.952	< 2e-16
batch6	1.0401618	0.1060365	9.809	< 2e-16
batch7	0.5436922	0.1091275	4.982	0.000000629
batch8	0.4959007	0.1089257	4.553	0.000005297
batch9	0.3857930	0.1185933	3.253	0.00114
temp	0.0109669	0.0004126	26.577	< 2e-16

Phi coefficients (precision model with identity link):

	Estimate	Std. Error	z value	Pr(> z)
(phi)	440.3	110.0	4.002	0.0000629

Type of estimator: ML (maximum likelihood)
Log-likelihood: 84.8 on 12 Df
Pseudo R-squared: 0.9617
Number of iterations: 51 (BFGS) + 3 (Fisher scoring)

`plot(predictorEffects(gy_logit))`



6 Ordinal Models (ordinal package)

Proportional odds logit and probit regression models fit with the `polr` function in the `MASS` package (Venables and Ripley, 2002) are supported in the `effects` package. The `ordinal` package, (Christensen, 2015) contains three functions that are very similar to `polr`. The `clm` and `clm2` functions allow more link functions and a number of other generalizations. The `clmm` function allows including random effects.

6.1 `clm`

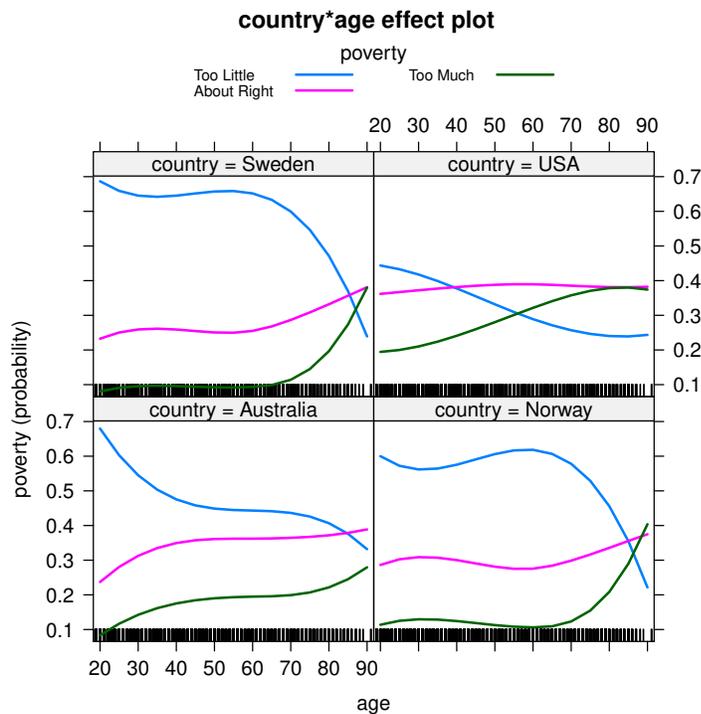
```
print(Effect.clm)

function (focal.predictors, mod, ...)
{
  if (requireNamespace("MASS", quietly = TRUE)) {
    polr <- MASS::polr
  }
  else stop("MASS package is required")
  polr.methods <- c("logistic", "probit", "loglog", "cloglog",
    "cauchit")
  method <- mod$link
  if (method == "logit")
    method <- "logistic"
  if (!(method %in% polr.methods))
    stop("'link' must be a 'method' supported by polr; see help(polr)")
  if (mod$threshold != "flexible")
    stop("Effects only supports the 'flexible' threshold")
  if (is.null(mod$Hessian)) {
    message("\nRe-fitting to get Hessian\n")
    mod <- update(mod, Hess = TRUE)
  }
  numTheta <- length(mod$Theta)
  numBeta <- length(mod$beta)
  or <- c((numTheta + 1):(numTheta + numBeta), 1:(numTheta))
  args <- list(type = "polr", coefficients = mod$beta, method = method,
    vcov = as.matrix(vcov(mod)[or, or]))
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f98426ad550>
<environment: namespace:effects>
```

This method first checks that the `MASS` package is available. Since the `clm` function allows suppressing the computation of the Hessian, the function checks and computes it if needed to get the estimated covariance matrix. The `clm` function orders the parameters in the order (threshold parameters, linear predictor

parameters), so the next few lines identify the elements of `vcov` that are needed by `Effects`. Since the `polr` function does not allow thresholds other than `flexible`, we don't allow them either. The `polr` argument `method` is equivalent to the `clm` argument `link`, except that the `clm` link "logit" is equivalent to the `polr` method "logit" "logistic".

```
require(ordinal)
require(MASS)
mod.wvs1 <- clm(poverty ~ gender + religion + degree + country*poly(age,3),
  data=WVS)
plot(Effect(c("country", "age"), mod.wvs1),
  lines=list(multiline=TRUE), layout=c(2, 2))
```



6.2 c1m2

Although the fitted models are similar, syntax for `c1m2` is not the same as `clm`, so a separate method is required.

```
print(Effect.c1m2)

function (focal.predictors, mod, ...)
{
  if (requireNamespace("MASS", quietly = TRUE)) {
```

```

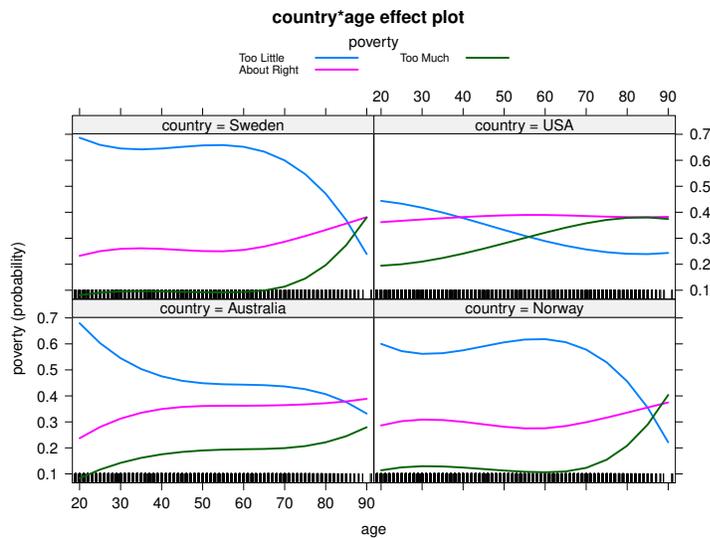
    polr <- MASS::polr
  }
  polr.methods <- c("logistic", "probit", "loglog", "cloglog",
    "cauchit")
  method <- mod$link
  if (!(method %in% polr.methods))
    stop("'link' must be a 'method' supported by polr; see help(polr)")
  if (is.null(mod$Hessian)) {
    message("\nRe-fitting to get Hessian\n")
    mod <- update(mod, Hess = TRUE)
  }
  if (mod$threshold != "flexible")
    stop("Effects only supports the flexible threshold")
  numTheta <- length(mod$Theta)
  numBeta <- length(mod$beta)
  or <- c((numTheta + 1):(numTheta + numBeta), 1:(numTheta))
  args <- list(type = "polr", formula = mod$call$location,
    coefficients = mod$beta, method = method, vcov = as.matrix(vcov(mod)[or,
    or]))
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f9852ebbf70>
<environment: namespace:effects>

```

```

v2 <- clm2(poverty ~ gender + religion + degree + country*poly(age,3),data=WVS)
plot(emod2 <- Effect(c("country", "age"), v2),
  lines=list(multiline=TRUE), layout=c(2,2))

```



6.3 clmm

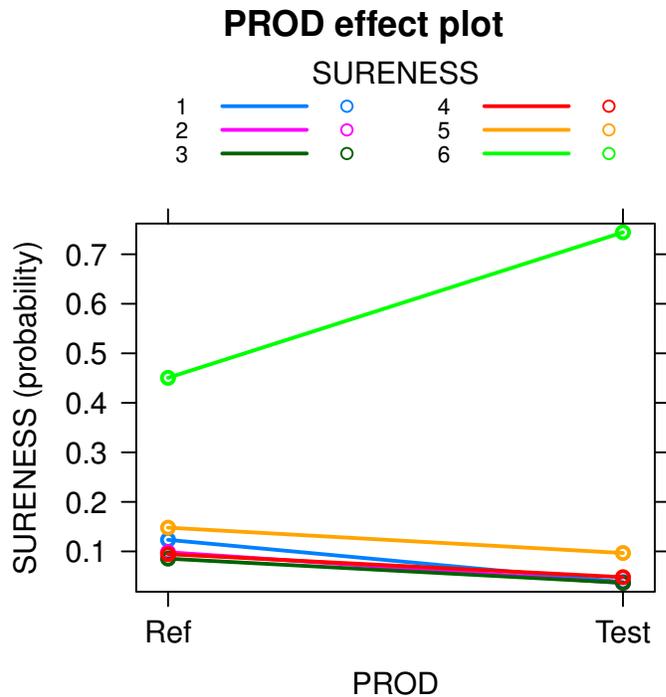
This function allows for random effects in an ordinal model.

```
print(Effect.clmm)

function (focal.predictors, mod, ...)
{
  if (requireNamespace("MASS", quietly = TRUE)) {
    polr <- MASS::polr
  }
  else stop("The MASS package must be installed")
  polr.methods <- c("logistic", "probit", "loglog", "cloglog",
    "cauchit")
  method <- mod$link
  if (method == "logit")
    method <- "logistic"
  if (!(method %in% polr.methods))
    stop("'link' must be a 'method' supported by polr; see help(polr)")
  if (is.null(mod$Hessian)) {
    message("\nRe-fitting to get Hessian\n")
    mod <- update(mod, Hess = TRUE)
  }
  if (mod$threshold != "flexible")
    stop("Only threshold='flexible' supported by Effects")
  numTheta <- length(mod$Theta)
  numBeta <- length(mod$beta)
  or <- c((numTheta + 1):(numTheta + numBeta), 1:(numTheta))
  Vcov <- as.matrix(vcov(mod)[or, or])
  args <- list(type = "polr", formula = formula(mod), coefficients = mod$beta,
    method = method, vcov = as.matrix(Vcov))
  Effect.default(focal.predictors, mod, ..., sources = args)
}
<bytecode: 0x7f985f0c62a8>
<environment: namespace:effects>
```

The first few lines of the method check for the presence of the MASS package that is needed to use `polr`, makes sure the link used is supported by `polr`, and requires that the argument `threshold` has its default value. The `polr` and `clmm` functions store the fixed effects estimates of regression and threshold coefficients in different orders, so the next few lines rearrange the variance matrix to match the order that `polr` uses.

```
require(ordinal)
require(MASS)
mm1 <- clmm(SURENESS ~ PROD + (1|RESP) + (1|RESP:PROD),
  data = soup, link = "logit", threshold = "flexible")
plot(Effect("PROD", mm1), lines=list(multiline=TRUE))
```



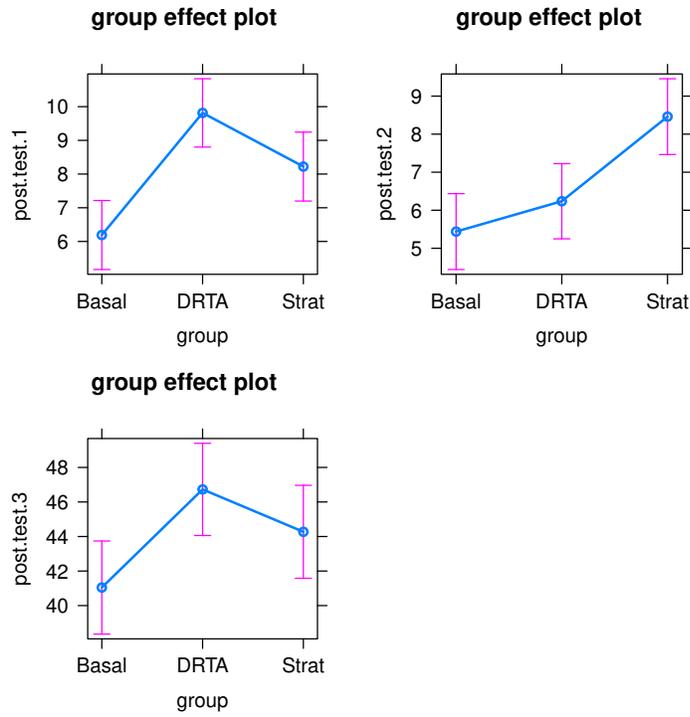
6.4 Others

The `poLCA` function in the `poLCA` package (Linzer and Lewis, 2011) fits polytomous variable latent class models, which uses the multinomial effects plots.

The `svyglm` function in the `survey` package (Lumley, 2004, 2016) fits generalized linear models using survey weights.

The `lm` function can also be used to create a multivariate linear model. The `Effect.mlm` function, with slightly different syntax, will draw effects plots for these models, with separate plots of each response.

```
data(Baumann, package="carData")
b1 <- lm(cbind(post.test.1, post.test.2, post.test.3) ~ group +
         pretest.1 + pretest.2, data = Baumann)
plot(Effect("group", b1))
```



References

- Bates, D., M. Mächler, B. Bolker, and S. Walker (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software* 67(1), 1–48.
- Christensen, R. H. B. (2015). *ordinal*—Regression Models for Ordinal Data. R package version 2015.6-28.
- Grün, B., I. Kosmidis, and A. Zeileis (2012). Extended beta regression in R: Shaken, stirred, mixed, and partitioned. *Journal of Statistical Software* 48(11), 1–25.
- Koller, M. (2016). *robustlmm*: An R package for robust estimation of linear mixed-effects models. *Journal of Statistical Software* 75(6), 1–24.
- Linzer, D. A. and J. B. Lewis (2011). *poLCA*: An R package for polytomous variable latent class analysis. *Journal of Statistical Software* 42(10), 1–29.
- Lumley, T. (2004). Analysis of complex survey samples. *Journal of Statistical Software* 9(1), 1–19. R package version 2.2.
- Lumley, T. (2016). *survey*: analysis of complex survey samples. R package version 3.32.

- Pinheiro, J., D. Bates, S. DebRoy, D. Sarkar, and R Core Team (2016). `nlme`: Linear and Nonlinear Mixed Effects Models. R package version 3.1-127.
- Pinheiro, J., D. Bates, S. DebRoy, D. Sarkar, and R Core Team (2018). `nlme`: *Linear and Nonlinear Mixed Effects Models*. R package version 3.1-137.
- Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S* (4th ed.). New York: Springer-Verlag.