Defining Effect Methods for Other Models

John Fox and Sanford Weisberg

Last revision: 2020-06-13

The **effects** package for R draws graphs that visualize the fitted response surface of a regression model with a linear predictor. Many modeling paradigms implemented by functions in tthe standard R distribution and in contributed CRAN packages fit into this framework, including functions 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., 2018); 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 `clm()` and `clm2()` from the **ordinal** package (Christensen, 2015); linear and generalized linear mixed models using the `lme()` function in the **nlme** package (Pinheiro et al., 2018) and the `lmer()` and `glmer()` functions in the **lme4** package (Bates et al., 2015); linear and generalized linear mixed models fit by penalized quasilikelihood (PQL) using the `glmmPQL()` function in the **MASS** package (Venables and Ripley, 2002); and latent class models fit by `poLCA()` in the **poLCA** package (Linzer and Lewis, 2011). This is hardly an exhaustive list of regression functions in R that are based on a linear predictor, and we have been asked from time to time to write functions to use **effects** with additional such functions. The mechanism for accommodating new classes of models with linear predictors is fairly simple. This vignette describes that mechanism, assuming familiarity with R's S3 object-oriented programming system.

The default method for the central `Effect()` generic function in the **effects** package, `Effect.default()`, works properly without modification for objects produced by *some* modeling functions—for example, objects of class `"merMod"`, which we describe below in Section 2—but, as illustrated in this vignette, specific adaptations will often be required.

The **effects** package has five generic functions that create the information needed for drawing effects plots: `Effect()`, `allEffects()`, `effect()`, `predictorEffect()`, and `predictorEffects()`. The other generic functions generate calls to `Effect()`, and so to support a new modeling function, only a new `Effect()` method is required.

This revision of the vignette describes Version 4.2-0 of **effects** package (and later versions), which makes use of the **insight** package (see `https://easystats.github.io/insight/`). The **insight** package can be used to simplify writing `Effect()` methods for classes of models not included directly in the **effects** package, as we illustrate in the examples below.

1 Using the effects Package with Other Modeling Methods: `gls()()` in the nlme Package as an Example

Applying the functions in the **effects** package to classes of regression models beyond those generated by `lm()` and `glm()` may require writing a method for the `Effect()` generic function for the corresponding class of model objects. For example, the `gls()` function in the **nlme** package (Pinheiro et al., 2018) fits linear models by generalized least squares, creating an object of class `"gls"`. The `Effect.gls()` method for this class proceeds by assembling the information required to compute an effect and then calls the default method with the necessary arguments to perform the computation:

```
Effect.gls <- function(focal.predictors, mod, ...){
     cl <- mod$call
     cl$weights <- NULL
```

```
  args <- list(
    type = "glm",
    call = cl,
    formula = insight::find_formula(mod),
    family = NULL,
    coefficients = coef(mod),
    vcov = insight::get_varcov(mod),
    method=NULL)
  Effect.default(focal.predictors, mod, ..., sources=args)
}
```

The `Effect.gls()` function has three required arguments: `focal.predictors` and `mod`, which match the named arguments of the `Effect()` generatic function and the first two named arguments of `Effect.default()`; and `...`, which matches any other arguments to be passed to `Effect.default()` (see `help("Effect")` for a list of these arguments).

The body of the function simply harvests the required information from the `mod` object passed to the function, and stores it in a list of named elements called `args`. The `args` list is then passed to the `sources` argument of the default `Effect()` method.

More generally, the named elements in the `sources` argument include:

`type` The **effects** package supports three basic types of modeling functions:

- `type = "glm"`, the default, is used for functions with a univariate response, a linear predictor, and possibly a link function. This model type includes linear models, generalized linear models, robust regression models, linear models fit by generalized least squares, linear and generalized linear mixed effects models, and many others.
- `type = "polr"` is used for ordinal regression models, exemplified by the `polr()` function in the **MASS** package, and similar functions described below in Section 7.
- `type = "multinom"` is used for multinomial response models, exemplified by the `multinom()` function in the **nnet** package, and for polytomous latent class models created by the `poLCA()` function in the **poLCA** package.

The default is `type = "glm"`.

`call` The `Effect.default()` method may use the call to set additional arguments. For `type="glm"`, for example, 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. In the `Effect.gls()` method shown above, the call is therefore modified by setting `weights=NULL`.

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

`formula` This element should return the formula for the fixed-effects part of the fitted model. For many models, the correct formula is returned by the `find_formula()` function in the **insight** package as `insight::find_formula(mod)$conditional`, and this is the default for the argument. If the default does not work for a particular class of models, the user must set `formula` to the fixed-effects formula. In particular, should the default not work, try using `formula(mod)`, and if this is successful, simply set `formula = formula(mod)`. Direct use of `formula()` will likely not work if your model has both fixed and random effects, or another complex type of linear predictor. In that case, you will have to write your own `find_formula()` method. For example, for the imaginary regression function `mymod()`, we might define

```
find_formula.mymod <- function(mod, ...){
  formula <- code to extract the formula
  list(conditional=formula)
}
```

This function will then be used by `Effect()` as needed. In this case, you need not set the `formula` element of the `sources` argument.

`family` This element is for GLM-like models that include a `family` that specifies both an error distribution and a link function, and is required only if `family=family(mod)` is not appropriate. See the `betareg()` example in Section 6 below for an example that includes a user-selected link function along with a fixed error distribution.

`coefficients` For linear and generalized linear models (including linear models fit by `gls()`), the fixed-effect coefficient estimates are returned by `coef(mod)`, but this is often not the case for more complicated modeling functions. The default for this element if it is not explicitly supplied is `effCoef(mod)`, where `effCoef()` is the following generic function:

```
effCoef <- function(mod, ...){
     UseMethod("effCoef", mod)
   }
```

with default method:

```
effCoef.default <- function(mod, ...){
     est1 <- insight::get_parameters(mod, ...)
     # est1 is a data frame, with labels in col 1
     #     and values in col 2
     # convert to a vector with named elements:
     est <- est1[, 2]
     names(est) <- est1[, 1]
     est
   }
```

This function returns fixed-effects coefficients for a variety of models, including models with random effects. You should consult `help("get_parameters")` to see if `effCoef.default()` will work for a particular modeling class, and if it does not work correctly, you can write your own `effCoef()` method.

`zeta` Ordinal regression models return both a set of regression coefficients and also a set of *thresholds*. The `polr()` function stores the regression coefficients and the thresholds in separate vectors, but other ordinal regression functions, such as `clm()` in the **ordinal** package, store them as a single vector. See Section 7 for an example of the use of this element for specifying the values of the thresholds.

`vcov` The function call `insight::get_varcov(mod)` is the default for this argument, and it often returns the estimated covariance matrix of the fixed effects, but in some cases users may have to write their own function for this purpose; see `help("get_varcov")` for more information.

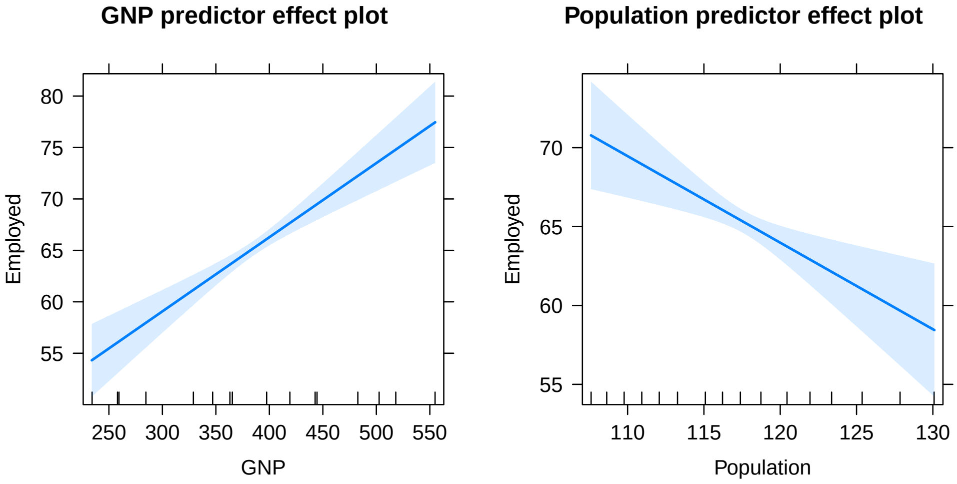
method This element is only for methods that produce effects similar to those for the `polr()` function, where the `method` argument is the name of a link function; see `help("polr")` for a list of the accepted links and Section 7.1 below for an example.

The only non-default element in `sources` for `Effect.gls()` is the modification of the `call` to remove weights from the call to `gls()`. Had this adaptation not been necessary, we would not have needed an `Effect.gls()` method, as the default method would have worked. The `Effect.gls()` method with all the default elements in `sources` omitted is therefore much more compact than the version given above:

```
Effect.gls <- function(focal.predictors, mod, ...){
     cl <- mod$call
     cl$weights <- NULL
     args <- list(call=cl)
     Effect.default(focal.predictors, mod, ..., sources=args)
   }
```

An example:

```
 library("effects")

Loading required package: carData

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

 library("nlme")
 g1 <- gls(Employed ~ GNP + Population,
             correlation=corAR1(form= ~ Year), data=longley)
 plot(predictorEffects(g1))
```



2 Linear Mixed Effects Models with `lme()` in the nlme package

With the use of functions in the **insight** package (as described above), linear mixed models created by the `lme()` function in the **nlme** package (Pinheiro et al., 2018) can be accommodated by the default `Effect()` method because the functions in the **insight** package extract the correct quantities:

```
 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 `lmer()` and `glmer()` in the lme4 package

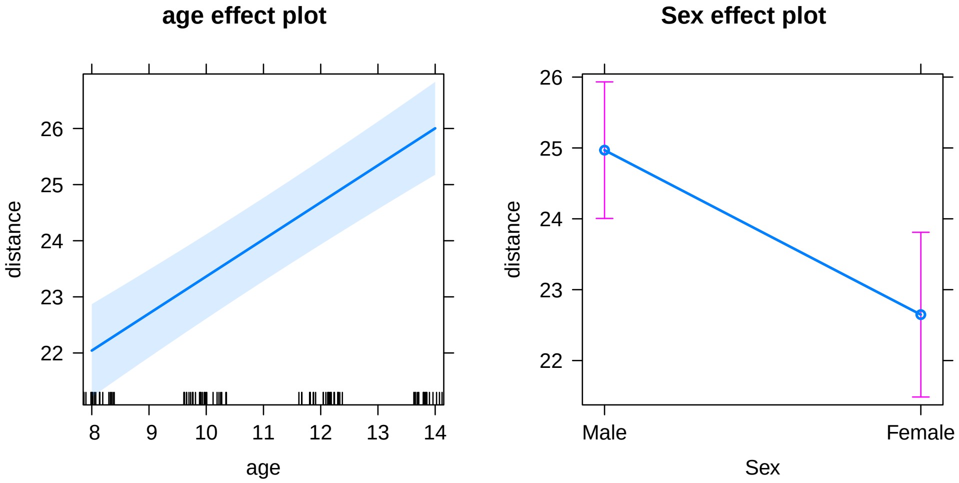
The **lme4** package (Bates et al., 2015) fits linear and generalized linear mixed effects models with the `lmer()` and `glmer()` functions, respectively. The same `Effect()` method can be used for `lmer()` and `glmer()`, both of which produce `"merMod"` objects.

The `Effect()` method for `lmer()` models allows two choices for computing the estimated coefficient covariance matrix, using the Kenward-Roger estimate if the argument `KR = TRUE`, and the usual asymptotic estimate from the information matrix when `KR = FALSE`. The default is `FALSE` because the KR computation can be very slow. If `KR = TRUE`, then the function also checks if the **pbkrtest** package is present. The `family` component of `sources` needs to be explicitly set, but the defaults for all other components are appropriate for `"merMod"` models.

```
 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(
    family=fam,
    vcov = if (fam$family == "gaussian" && fam$link == "identity" && KR)
      as.matrix(pbkrtest::vcovAdj(mod)) else insight::get_varcov(mod))
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x000000001e91dba0>
<environment: namespace:effects>
```

Usage examples:

```
 fm2 <- lme4::lmer(distance ~ age + Sex + (1 |Subject), data
                    = Orthodont)
 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 0.13569523 0.2798569
2      2 0.08391784 0.02363110 0.04775454 0.1433443
3      3 0.07401714 0.02241761 0.04040242 0.1317591
4      4 0.04842565 0.01959184 0.02163870 0.1048199
```

4 Linear and Generalized Linear Mixed Models Fit With glmm-PQL() in the MASS package

Venables and Ripley (2002) provide a penalized quasi-likelihood function for fitting linear and generalized linear mixed models in the **MASS** package. The `Effect.glmmPQL()` method only has to explicitly provide the `family` component of `sources`:

```
 print(Effect.glmmPQL)

function(focal.predictors, mod, ...){
  args <- list(
    family = mod$family)
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x0000000022382c60>
<environment: namespace:effects>
```

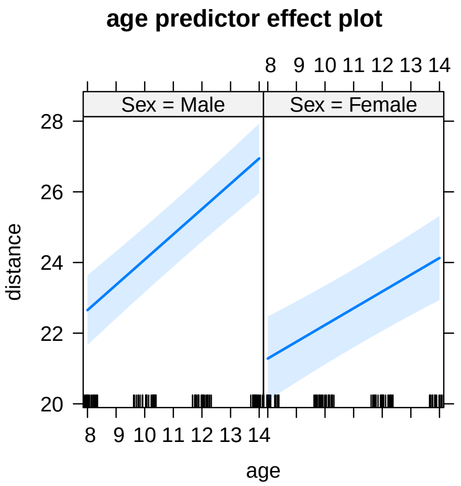
5 Robust Linear Mixed Models with rlmer() in the robustlmm Package)

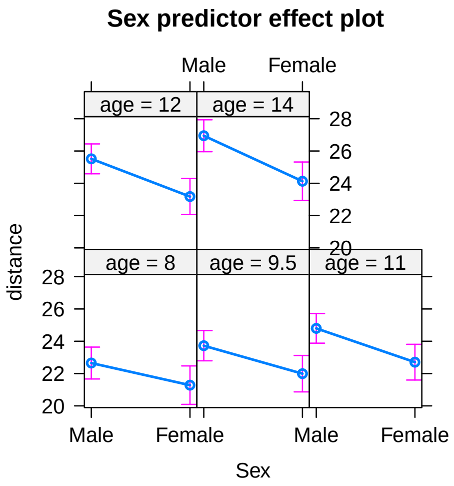
The `rlmer()` function in the **robustlmm** package (Koller, 2016) fits linear mixed models robustly. As `rlmer()` closely parallels the `lmer()` function, objects created by `rlmer()` are easily used with functions in the **effects** package:

```
 print(Effect.rlmerMod)

function(focal.predictors, mod, ...){
  args <- list(
    family=family(mod))
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x00000000224c5330>
<environment: namespace:effects>
```

For example:

```
 library("lme4")

Loading required package: Matrix


Attaching package: 'lme4'

The following object is masked from 'package:nlme':

    lmList

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





6 Beta Regression with `betareg()` in the betareg Package

The `betareg()` function in the **betareg** package (Grün et al., 2012) fits regressions with Beta distributed errors. Beta regression has a response $y \in [0, 1]$, with the connection between the mean μ of the Beta distribution and a set of regressors $\mathbf{x}$ given by a link function $\mathbf{x}'\boldsymbol{\beta} = g(\mu)$. The variance function for the Beta distribution is $\mathrm{var}(y) = \mu(1 - \mu)/(1 + \phi)$, for the precision parameter ϕ estimated by `betareg()`.

The `Effect.betareg()` method is more complicated than the methods previously described:

```
 print(Effect.betareg)

function(focal.predictors, mod, ...){
  coef <- mod$coefficients$mean
  vco <- vcov(mod)[1:length(coef), 1:length(coef)]
# betareg uses beta errors with mean link given in mod$link$mean.
# Construct a family based on the binomial() family
  fam <- binomial(link=mod$link$mean)
# adjust the varince function to account for beta variance
  fam$variance <- function(mu){
    f0 <- function(mu, eta) (1-mu)*mu/(1+eta)
    do.call("f0", list(mu, mod$coefficient$precision))}
# adjust initialize
  fam$initialize <- expression({mustart <- y})
# collect arguments
  args <- list(
    call = mod$call,
    formula = formula(mod),
    family=fam,
    coefficients = coef,
    vcov = vco)
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x0000000024a47718>
<environment: namespace:effects>
```

coefficients The default, `insight::find_parameters(mod)`, returns the coefficients for the linear predictor and for the precision parameter, as shown on the help page for `insight::find_parameters`. The correct call is `insight::find_parameters(mod, component="conditional")`.

vcov Similarly, the estimated covariance matrix of the coefficients in the linear predictor is `insight::get_varcov(mod, component="conditional")`.

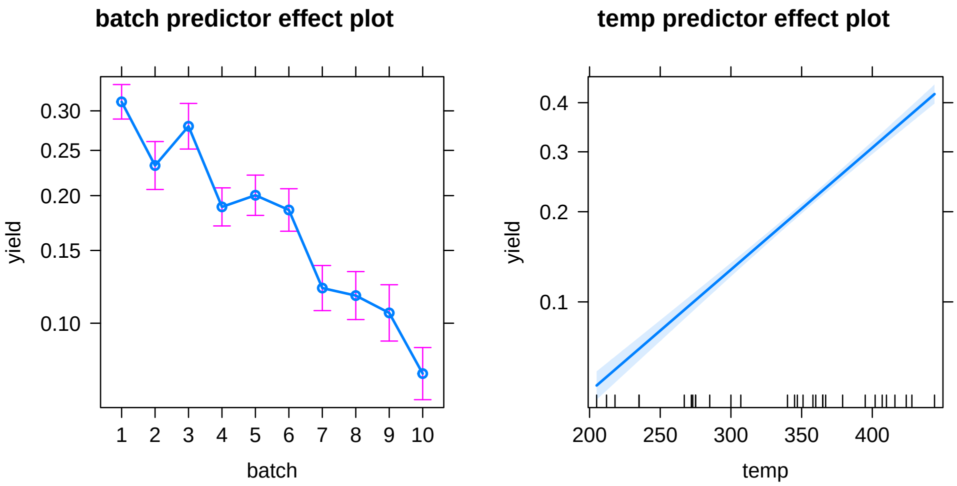
family `betareg()` does not have a family argument, although it does store a link in `mod$link$mean`. The `Effect.betareg()` method creates a family object suitable for use with `Effect.default()` from the `binomial()` family generator function. It then adjusts the family object by changing the binomial variance to the variance for the Beta distribution. Because 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. We also must replace the `initialize` method in the family to one appropriate for $y \in [0, 1]$.

For example:

```
library("betareg")
library("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))
```



7 Ordinal Regression Models Via the (ordinal Package)

Proportional odds logit and probit regression models fit with the `polr()` function in the **MASS** package (Venables and Ripley, 2002) are supported by the **effects** package. The **ordinal** package, (Christensen, 2015) contains three functions that are very similar to `polr()`: The `clm()` and `clm2()` functions provide more link functions and a number of other generalizations. The `clmm()` function fits mixed models that include random as well as fixed effects.

`Effect()` methods for ordinal models are considerably more complex than the preceding examples. In particular, the function `clm()` is not supported by the **insight** package, although `clm2()` and `clmm()` are supported, and thus `Effect.clm()` has a relatively complicated definition:

7.1 clm()

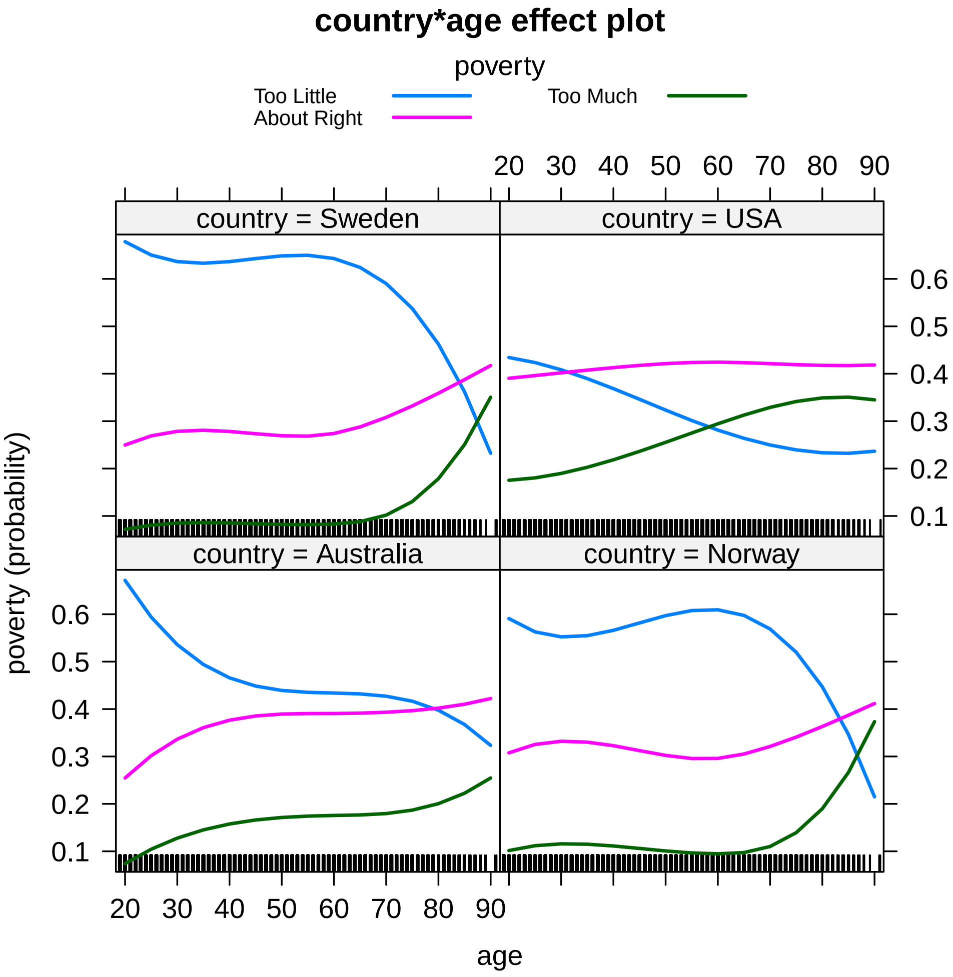
```
 print(Effect.clm)

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")
  numTheta <- length(mod$Theta)
  numBeta <- length(mod$beta)
  or <- c( (numTheta+1):(numTheta + numBeta), 1:(numTheta))
  args <- list(
    type = "polr",
    coefficients = mod$beta,
    zeta = mod$alpha,
    method=method,
    vcov = as.matrix(vcov(mod)[or, or]))
  Effect.default(focal.predictors, mod, ..., sources=args)
}
```

```
<bytecode: 0x000000001fc4d240>
<environment: namespace:effects>
```

This method first checks that the **MASS** package is available. The `clm()` function returns parameters in the order (threshold parameters, linear-predictor parameters), so the next few lines of code identify the elements of`vcov` that are needed by `Effects()`. Because the `polr()` function does not support thresholds other than `flexible`, we don't support them either. The `zeta` element of `source` supplies the estimated thresholds, which are called `zeta` in `polr()`, and `Alpha` in `clm()`. The `polr() method` argument is equivalent to the `clm() link` argument, with the `clm() "logit"` link equivalent to the `polr() "logistic"` method.

An example:

```
library("ordinal")
library("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))
```



7.2 `clm2()`

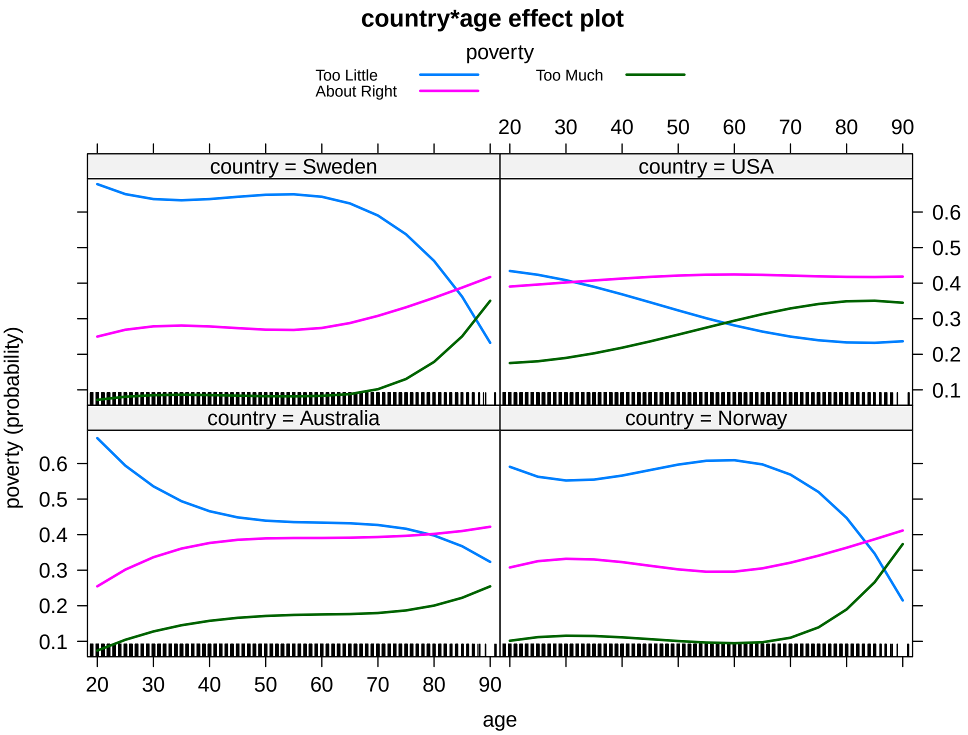
Although the fitted models are similar, `"clm2"` objects are not the same as `"clm"` objects, so a separate `Effect.clm2()` method is required:

```
 print(Effect.clm2)
```

```
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,
    zeta = mod$Theta,
    method=method,
    vcov = as.matrix(vcov(mod)[or, or]))
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x0000000027775308>
<environment: namespace:effects>
```

For example:

```
 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))
```



7.3 clmm()

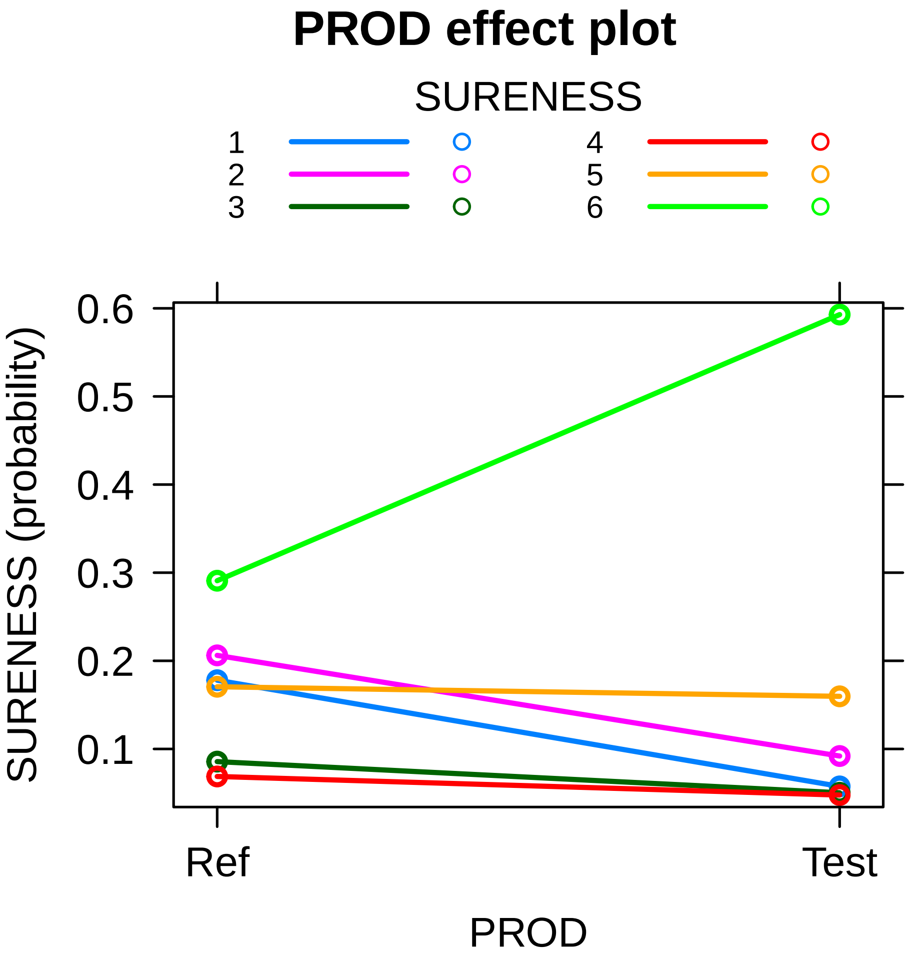
The clmm() function fits ordinal mixed effects models, and the Effect.clmm() method is defined as follows:

```
 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$beta)
  or <- c( (numTheta+1):(numTheta + numBeta), 1:(numTheta))
  Vcov <- as.matrix(vcov(mod)[or, or])
  args <- list(
    type = "polr",
    formula = insight::find_formula(mod)$conditional,
    coefficients = mod$beta,
    zeta=mod$alpha,
    method=method,
    vcov = as.matrix(Vcov))
  Effect.default(focal.predictors, mod, ..., sources=args)
}
<bytecode: 0x00000000223e41c8>
<environment: namespace:effects>
```

The first few lines of the method check for the presence of the **MASS** package, which is needed for the `polr()` function; make sure the link employed is supported by `polr()`; and require that the `threshold` argument was set to its default value. The `polr()` and `clmm()` functions store the fixed effects estimates and threshold coefficients in different orders, and so the next few lines rearrange the coefficient covariance matrix to match the order that `polr()` uses.

An example:

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



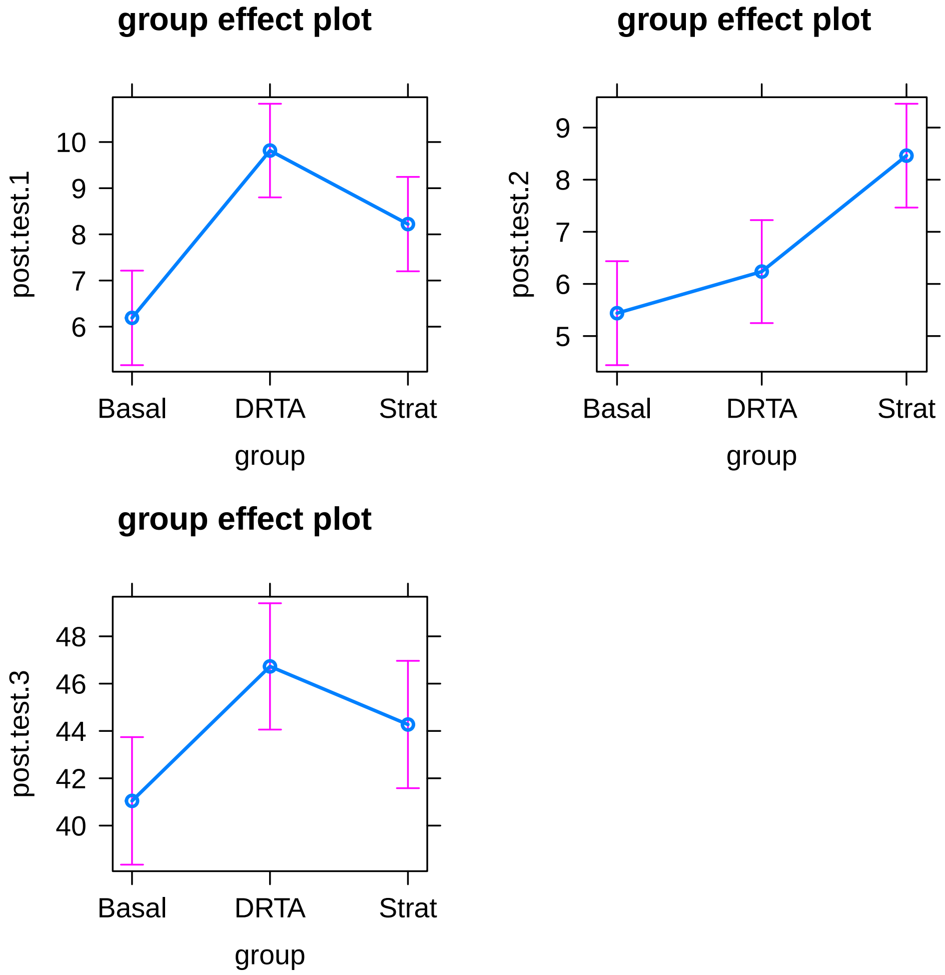
8 Other Regression Functions

The `poLCA()` function in the **poLCA** package (Linzer and Lewis, 2011) fits polytomous latent class models, which produce multinomial effect plots.

The `svyglm()` function in the **survey** package (Lumley, 2004, 2016) fits generalized linear models to data from complex sample surveys, making provision, for example, for strata, clustering, and sampling weights.

The `lm()` function can also be used to fit multivariate linear models. The `Effect.mlm()` method computes effects for these models, producing separate graphs of each response. For example:

```
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 (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.