HAL
archives-ouvertes.fr

Extended Model Formulas in R: Multiple Parts and Multiple Responses

Achim Zeileis, Yves Croissant

▶ To cite this version:

Achim Zeileis, Yves Croissant. Extended Model Formulas in R: Multiple Parts and Multiple Responses. Journal of Statistical Software, University of California, Los Angeles, 2010, 34 (1), <10.18637/jss.v034.i01>. <hal-01245303>

HAL Id: hal-01245303
http://hal.univ-reunion.fr/hal-01245303

Submitted on 30 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Statistical Software

April 2010, Volume 34, Issue 1. http://www.jstatsoft.org/

Extended Model Formulas in R: Multiple Parts and Multiple Responses

Achim Zeileis
Universität Innsbruck

Yves Croissant
Université de la Réunion

Abstract

Model formulas are the standard approach for specifying the variables in statistical models in the S language. Although being eminently useful in an extremely wide class of applications, they have certain limitations including being confined to single responses and not providing convenient support for processing formulas with multiple parts. The latter is relevant for models with two or more sets of variables, e.g., different equations for different model parameters (such as mean and dispersion), regressors and instruments in instrumental variable regressions, two-part models such as hurdle models, or alternative-specific and individual-specific variables in choice models among many others. The R package **Formula** addresses these two problems by providing a new class "`Formula`" (inheriting from "`formula`") that accepts an additional formula operator `|` separating multiple parts and by allowing all formula operators (including the new `|`) on the left-hand side to support multiple responses.

Keywords: formula processing, model frame, model matrix, R.

1. Introduction

Since publication of the seminal "white book" (Chambers and Hastie 1992) the standard approach for fitting statistical models in the S language is to apply some model-fitting function (such as `lm()` or `glm()`) to a "`formula`" description of the variables involved in the model and typically stored in a "`data.frame`". The semantics of formula processing are based on the ideas of the Wilkinson and Rogers (1973) notation which in turn was targeted at specification of analysis of variance models. Despite this emphasis on specification of terms in models with linear predictors, formula notation has always been used much more generally in S, e.g., for specifying variables in classification and regression trees, margins in contingency tables, or variables in graphical displays. In such applications, the precise meaning of a particular formula depends on the function that processes it. Typically, the standard formula processing

approach would encompass extraction of the specified terms using `terms()`, preparation of a preprocessed data frame using `model.frame()`, and computation of a "design" or "regressor" matrix using `model.matrix()`.

However, there are certain limitations to these standard formula processing tools in S that can be rather inconvenient in certain applications:

1. The formula notation can just be used on the right-hand side (RHS) of a formula (to the right of the `~`) while it has its original arithmetic meaning on the left-hand side (LHS). This makes it difficult to specify multiple responses, especially if these are not numeric (e.g., factors). This feature would be useful for specifying multivariate outcomes of mixed types in independence tests (e.g., in the **coin** package, Hothorn, Hornik, van de Wiel, and Zeileis 2006, 2008) or in models with multivariate responses (e.g., supported in the **party** package Zeileis, Hothorn, and Hornik 2008a).

2. There is no simple construct in standard formula notation that allows one to separate several groups of variables from which separate model matrices can be derived. This task occurs in many types of models, e.g., when processing separate sets of variables for mean and dispersion (e.g., in the **betareg** package, Cribari-Neto and Zeileis 2010), separate equations for location, scatter, and shape (e.g., in the **gamlss** package, Stasinopoulos and Rigby 2007), regressors and instruments in instrumental variable regressions (e.g., in the **plm** package, Croissant and Millo 2008, or the **AER** package, Kleiber and Zeileis 2008), variables in two-part models such as hurdle models or zero-inflated regressions (e.g., in the **pscl** package, Zeileis, Kleiber, and Jackman 2008b), alternative-specific and individual-specific variables in choice models (e.g., in the **mlogit** package, Croissant 2010), efficiency level variables in stochastic frontier analysis (e.g., in the **frontier** package, Coelli and Henningsen 2010), or modeling variables and partitioning variables in model-based recursive partitioning techniques (e.g., in the **party** package, Zeileis *et al.* 2008a).

In many of the aformentioned packages, standard "`formula`" objects are employed but their processing is generalized, e.g., by using multiple formulas, multiple terms, by adding new formula operators or even more elaborate solutions. However, in many situations it is not easy to reuse these generalizations outside the package/function they were designed for. Therefore, as we repeatedly needed such generalizations in our own packages and addressed this in the past by various different solutions, it seemed natural to provide a more general unified approach by means of our new **Formula** package. This has already been reused in some of our own packages (including **AER**, **betareg**, **mlogit**, and **plm**) but can also be easily employed by other package developers (e.g., as in the **frontier** package). More applications in our own and other packages will hopefully follow.

In the remainder of this paper we discuss how multiple responses and multiple parts (both on the LHS and RHS) are enabled in the **Formula** package written in the R system for statistical computing (R Development Core Team 2009) and available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=Formula`. Built on top of basic "`formula`" objects, the "`Formula`" class just adds a thin additional layer which is based on a single additional operator, namely `|`, that can be used to separate different parts (or groups) of variables. **Formula** essentially just handles the different formula parts and leverages the

existing methods for "formula" objects for all remaining operations.[1] In Section 2 we show two small motivating examples that convey the main ideas implemented in the package and how easily they can be employed. The details of the "Formula" class and its associated methods are discussed in Section 3 and Section 4 illustrates the usage of **Formula** in developing new model-fitting functions. A short summary in Section 5 concludes the paper.

2. Motivating examples

To illustrate the basic ideas of the **Formula** package, we first generate a small artificial data set (with both numeric and categorical variables) and subsequently illustrate its usage with a multi-part and a multi-response "Formula", respectively.

```
R> set.seed(1090)
R> dat <- as.data.frame(matrix(round(runif(21), digits = 2), ncol = 7))
R> colnames(dat) <- c("y1", "y2", "y3", "x1", "x2", "x3", "x4")
R> for(i in c(2, 6:7)) dat[[i]] <- factor(dat[[i]] < 0.5,
+    labels = c("a", "b"))
R> dat$y2[1] <- NA
R> dat

    y1   y2   y3   x1   x2 x3 x4
1 0.82 <NA> 0.27 0.09 0.22  a  b
2 0.70    a 0.17 0.26 0.46  b  b
3 0.65    b 0.28 0.03 0.37  a  a
```

2.1. Multiple parts

We start out with a simple formula `log(y1) ~ x1 + x2 | I(x1^2)` which has a single response `log(y1)` on the LHS and two parts on the RHS, separated by `|`. The first part contains `x1` and `x2`, the second contains `I(x1^2)`, i.e., the squared values of `x1`. The initial "formula" can be transformed to a "Formula" using the constructor function `Formula()`:

```
R> F1 <- Formula(log(y1) ~ x1 + x2 | I(x1^2))
R> length(F1)

[1] 1 2
```

The `length()` method indicates that there is one part on the LHS and two parts on the RHS. The first step of processing data using a formula is typically the construction of a so-called model frame containing only the variables required by the formula. As usual, this can be obtained with the `model.frame()` method.

[1] Throughout the paper, no terminological distinction is made between classic "formula" objects as such and the way they are interpreted by standard processing tools (e.g., `terms()`, `model.frame()`, `model.matrix()`). The reasons for this are twofold: First, it reflects their use as described in Chambers and Hastie (1992) and, second, generalizations generally require extra effort from the user.

```
R> mf1 <- model.frame(F1, data = dat)
R> mf1

     log(y1)   x1   x2 I(x1^2)
1 -0.1984509 0.09 0.22  0.0081
2 -0.3566749 0.26 0.46  0.0676
3 -0.4307829 0.03 0.37   9e-04
```

As this model just has a single response (as in base "`formula`" objects), the extractor function `model.response()` can be employed:

```
R> model.response(mf1)

         1          2          3
-0.1984509 -0.3566749 -0.4307829
```

For constructing separate model matrices for the two parts on the RHS, the `model.matrix()` can be employed and additionally specifying the argument `rhs`:

```
R> model.matrix(F1, data = mf1, rhs = 1)

  (Intercept)   x1   x2
1           1 0.09 0.22
2           1 0.26 0.46
3           1 0.03 0.37
attr(,"assign")
[1] 0 1 2

R> model.matrix(F1, data = mf1, rhs = 2)

  (Intercept) I(x1^2)
1           1  0.0081
2           1  0.0676
3           1  0.0009
attr(,"assign")
[1] 0 1
```

2.2. Multiple responses

To accommodate multiple responses, all formula operators can be employed on the LHS in "`Formula`" objects (whereas they would have their original arithmetic meaning in "`formula`" objects). This also includes the new `|` operator for separating different parts. Thus, one could specify a two-part response via `y1 | y2 ~ x3` or a single part with two variables via `y1 + y2 ~ x3`. We do the latter in the following illustration.

```
R> F2 <- Formula(y1 + y2 ~ x3)
R> length(F2)
```

```
[1] 1 1
```

As usual, the model frame can be derived by

```
R> mf2 <- model.frame(F2, data = dat)
R> mf2
```

```
    y1 y2 x3
2 0.70  a  b
3 0.65  b  a
```

However, there is an important difference to the model frame `mf1` derived in the previous example. As the (non-generic) `model.response()` function would only be able to extract a single response column from a model frame, multi-response model frames in **Formula** are implemented to have no response at all in the `model.response()` sense:

```
R> model.response(mf2)
```

```
NULL
```

As `model.response()` cannot be extended (without overloading the base R function), **Formula** provides a new generic function `model.part()` which can be used to extract all variables from a model frame pertaining to specific parts. This can also be used to extract multiple responses. Its syntax is modeled after `model.matrix()` taking a "`Formula`" as the first argument. For further details see Section 3. Its application is straightforward and all LHS variables (in the first and only part of the LHS) can be extracted via

```
R> model.part(F2, data = mf2, lhs = 1)
```

```
    y1 y2
2 0.70  a
3 0.65  b
```

The same method also works for single response models as in the previous example:

```
R> model.part(F1, data = mf1, lhs = 1, drop = TRUE)
```

```
         1          2          3
-0.1984509 -0.3566749 -0.4307829
```

3. Implementation

Below we discuss the ideas for the design of the "`Formula`" class and methods. As all tools are built on top of the "`formula`" class and its associated methods whose most important feature are briefly outlined as well.

3.1. Working with classic "formula" objects

Classic "formula" objects (Chambers and Hastie 1992) are constructed by using ~ to separate LHS and RHS, typically (but not necessarily) interpreted as "dependent" and "explanatory" variables. For describing relationships between variables on the RHS, several operators can be used: +, -, *, /, :, %in%, ^. Thus, these do not have their original meaning in the top-level RHS while they keep their original arithmetic meaning on higher levels of the RHS (thus, within function calls such as I(x1^2)) and on the LHS in general. A first step in using "formula" objects is often to compute the associated "terms" using the function terms(). Based on the formula or the associated terms and a suitable set of variables (typically either in a data frame or in the global environment) model.frame() can build a so-called model frame that contains all variables in the formula/terms. This might include processing of missing values (NAs), carrying out variable transformations (such as logs, squares, or other functions of one or more variables) and providing further variables like weights, offset, etc. A model frame is simply a "data.frame" with additional attributes (including "terms") but without a specific class. From this preprocessed model frame several components can be extracted using model.extract() or model.response(), model.weights(), and model.offset(), all of which are non-generic. Last not least, model.matrix() (which is generic) can compute "design" or "regressor" matrices based on the formula/terms and the associated model frame.

3.2. Constructing "Formula" objects

To accomplish the main objectives of the "Formula" class, the following design principles have been used: reuse of "formula" objects, introduction of a single new operator |, and support of all formula operators on the LHS. Thus, | loses its original meaning (logical "or") on the first level of formulas but can still be used with its original meaning on higher levels, e.g., factor(x1 > 0.5 | x3 == "a") still works as before. For assigning a new class to formulas containing |, the constructor function Formula() is used:

```
R> F3 <- Formula(y1 + y2 | log(y3) ~ x1 + I(x2^2) | 0 + log(x1) | x3 / x4)
R> F3

y1 + y2 | log(y3) ~ x1 + I(x2^2) | 0 + log(x1) | x3/x4

R> length(F3)

[1] 2 3
```

In this example, F3 is an artificially complex formula with two parts on the LHS and three parts on the RHS, both containing multiple terms, transformations or other formula operators. Apart from assigning the new class "Formula" (in addition to the old "formula" class), Formula() also splits up the formula into LHS and RHS parts which are stored as list attributes "lhs" and "rhs", respectively, e.g.,

```
R> attr(F3, "lhs")

[[1]]
y1 + y2
```

```
[[2]]
log(y3)
```

and analogously `attr(F3, "rhs")`. The user never has to compute on these attributes directly, but many methods for "`Formula`" objects take `lhs` and/or `rhs` arguments. These always refer to index vectors for the two respective lists.

It would have been conceivable to generalize not only the notion of formulas but also of terms or model frames. However, there are few generics with methods for "`terms`" objects and there is no particular class for model frames at all. Hence, computing with generalized versions of these concepts would have required much more overhead for users of **Formula**. Hence, it was decided not to do so and keep the package interface as simple as possible.

3.3. Extracting "`formula`" and "`terms`" objects

As subsequent computations typically require a "`formula`" or a "`terms`" object, **Formula** provides suitable `formula()` and `terms()` extractors for "`Formula`" objects. For the former, the idea is to be able to switch back and forth between the "`Formula`" and "`formula`" representation, e.g., `formula(Formula(...))` should recover the original input formula. For the latter, the objective is somewhat different: `terms()` should always return a "`terms`" object that can be processed by `model.frame()` and similar functions. Thus, the terms must not contain multiple responses and/or the new `|` operator.

The `formula()` method is straightforward. When no additional arguments are supplied it recovers the original "`formula`". Furthermore, there are two optional additional arguments `lhs` and `rhs`. With these arguments subsets of formulas can be chosen, indexing the LHS and RHS parts. The default value for both is `NULL`, meaning that all parts are employed.

```
R> formula(F3)

y1 + y2 | log(y3) ~ x1 + I(x2^2) | 0 + log(x1) | x3/x4

R> formula(F3, lhs = 2, rhs = -2)

log(y3) ~ x1 + I(x2^2) | x3/x4

R> formula(F3, lhs = c(TRUE, FALSE), rhs = 0)

y1 + y2 ~ 0
```

Similarly, `terms()` computes a "`terms`" object, by default using all parts in the formula, but `lhs` and `rhs` can be used as above. To remove the `|` operator, all parts are collapsed using the `+` operator. Furthermore, the LHS variables can only be kept on the LHS if they contain a single term. Otherwise, to stop subsequent computations from interpreting the formula operators as arithmetic operators, all LHS components are added on the RHS as well. Thus, for `F3` we obtain

```
R> terms(F3)
```

```
~y1 + y2 + log(y3) + (x1 + I(x2^2)) + (0 + log(x1)) + x3/x4
attr(,"variables")
list(y1, y2, log(y3), x1, I(x2^2), log(x1), x3, x4)
attr(,"factors")
        y1 y2 log(y3) x1 I(x2^2) log(x1) x3 x3:x4
y1       1  0       0  0       0       0  0     0
y2       0  1       0  0       0       0  0     0
log(y3)  0  0       1  0       0       0  0     0
x1       0  0       0  1       0       0  0     0
I(x2^2)  0  0       0  0       1       0  0     0
log(x1)  0  0       0  0       0       1  0     0
x3       0  0       0  0       0       0  1     2
x4       0  0       0  0       0       0  0     1
attr(,"term.labels")
[1] "y1"      "y2"      "log(y3)" "x1"      "I(x2^2)" "log(x1)"
[7] "x3"      "x3:x4"
attr(,"order")
[1] 1 1 1 1 1 1 1 2
attr(,"intercept")
[1] 0
attr(,"response")
[1] 0
attr(,".Environment")
<environment: 0x1f5acc0>
```

Instead of using all parts, subsets can again be selected. We illustrate this below but only show the associated "`formula`" to save output space:

```
R> formula(terms(F3))

~y1 + y2 + log(y3) + (x1 + I(x2^2)) + (0 + log(x1)) + x3/x4

R> formula(terms(F3, lhs = 2, rhs = -2))

log(y3) ~ x1 + I(x2^2) + x3/x4

R> formula(terms(F3, lhs = c(TRUE, FALSE), rhs = 0))

~y1 + y2
```

3.4. Computing model frames, matrices, and responses

Given that suitable "`terms`" can be extracted from "`Formula`" objects, it is straightforward to set up the corresponding model frame. The `model.frame()` method simply first calls the `terms()` method and then applies the default `model.frame()`. Hence, all further arguments are processed as usual, e.g.,

```
R> mf3 <- model.frame(F3, data = dat, subset = y1 < 0.75, weights = x1)
R> mf3

    y1 y2   log(y3)   x1 I(x2^2)   log(x1) x3 x4 (weights)
2 0.70  a -1.771957 0.26  0.2116 -1.347074  b  b      0.26
3 0.65  b -1.272966 0.03  0.1369 -3.506558  a  a      0.03
```

All subsequent computations are then based on this preprocessed model frame (and possibly the original "`Formula`"). Thus, the model matrices for each RHS part can be easily computed, again setting the `rhs` argument:

```
R> model.matrix(F3, data = mf3, rhs = 2)

    log(x1)
2 -1.347074
3 -3.506558
attr(,"assign")
[1] 1
```

Typically, just a single RHS will be selected and hence `rhs = 1` and not `rhs = NULL` is the default in this method. However, multiple RHS parts are also supported. Also, there is a `lhs` argument available (defaulting to `NULL`) which might seem unnecessary at first sight but it is important in case the selected RHS part(s) contain(s) a "`.`" that needs to be resolved (see `?model.matrix.Formula` for an example).

The LHS parts can be extracted using the method for the new `model.part()` generic, employing a syntax similar to `model.matrix()`:

```
R> model.part(F3, data = mf3, lhs = 1)

    y1 y2
2 0.70  a
3 0.65  b

R> model.part(F3, data = mf3, lhs = 2)

    log(y3)
2 -1.771957
3 -1.272966
```

As argued above, introduction of a new generic is necessary for supporting multi-response formulas because `model.response()` is non-generic. For model frames derived from single-response formulas, `model.response()` can be used as usual. The remaining extractors work as usual:

```
R> model.weights(mf3)

[1] 0.26 0.03
```

3.5. Further methods

To conclude the suite of methods available for the new "Formula" class, **Formula** provides an `update()` method and a new `as.Formula()` generic with suitable methods. The former updates the formula part by part, adding new parts if necessary:

```
R> update(F1, . ~ . - x1 | . + x1)

log(y1) ~ x2 | I(x1^2) + x1

R> update(F1, . + y2 | y3 ~ .)

log(y1) + y2 | y3 ~ x1 + x2 | I(x1^2)
```

The `as.Formula()` method coerces to "Formula", possibly also processing multiple arguments:

```
R> as.Formula(y1 ~ x1, y2 ~ x2, ~ x3)

y1 | y2 ~ x1 | x2 | x3
```

4. Usage in model fitting functions

A typical application of **Formula** is to provide the workhorse for formula processing in model-fitting functions that require specification of multiple parts or multiple responses. To provide a very brief and simple example, we show how such a function can be set up. For illustration, we compute the coefficients in an instrumental variables regression using two-stage least squares.

The `ivcoef()` function below takes the usual arguments `formula`, `data`, `subset`, and `na.action` (and further arguments `weights` and `offset` could be included in the same way). The `formula` should be a two-part formula like `y ~ x1 + x2 | z1 + z2 + z3`. There is a single response on the LHS, one RHS part with the regressors and a second RHS part with the instruments. The function `ivcoef()` uses the typical workflow of model-fitting functions and processes its arguments in the following four steps: (1) process the call, (2) set up the model frame (using the "Formula" method), (3) extract response and regressors from the model frame, (4) estimate the model (by calling `lm.fit()` twice to compute the two-stage least squares coefficients).

```
R> ivcoef <- function(formula, data, subset, na.action, ...)
+  {
+    mf <- match.call(expand.dots = FALSE)
+    m <- match(c("formula", "data", "subset", "na.action"), names(mf), 0)
+    mf <- mf[c(1, m)]
+
+    f <- Formula(formula)
+    mf[[1]] <- as.name("model.frame")
+    mf$formula <- f
+    mf <- eval(mf, parent.frame())
```

```
+
+    y <- model.response(mf)
+    x <- model.matrix(f, data = mf, rhs = 1)
+    z <- model.matrix(f, data = mf, rhs = 2)
+
+    xz <- as.matrix(lm.fit(z, x)$fitted.values)
+    lm.fit(xz, y)$coefficients
+  }
```

The resulting function can then be applied easily (albeit not very meaningfully) to the `dat` data frame:

```
R> ivcoef(log(y1) ~ x1 | x2, data = dat)

(Intercept)          x1
  -0.169027   -1.260073
```

The same coefficients can be derived along with all the usual inference using the `ivreg()` function from the **AER** package (Kleiber and Zeileis 2008), which also uses the **Formula** tools in its latest release. Apart from providing inference and many other details, `ivreg()` also supports `weights`, `offsets` etc. Finally, for backward compatibility, the function also allows separate formulas for regressors and instruments (i.e., `formula = y ~ x1 + x2` and `instruments = ~ z1 + z2 + z3`) which can be easily incorporated using the **Formula** tools, e.g., replacing `f <- Formula(formula)` by

```
+   f <- if(!is.null(instruments)) as.Formula(formula, instruments)
+     else as.Formula(formula)
+   stopifnot(isTRUE(all.equal(length(f), c(1, 2))))
```

In summary, the usage of **Formula** should reduce the overhead for the developers of model-fitting functions in R with multiple responses and/or multiple parts and make the resulting programs more intelligible. Further R packages employing the "`Formula`" approach can be obtained from CRAN, including **betareg**, **frontier**, **mlogit**, and **plm**.

5. Summary

The **Formula** package provides tools for processing multi-response and multi-part formulas in the R system for statistical computing. The new class "`Formula`" inherits from the existing "`formula`" class, only adds a single new formula operator `|`, and enables the usage of formula operators on the left-hand side of formulas. The methods provided for "`Formula`" objects are as similar as possible to the classic methods, facilitating their usage in model-fitting functions that require support for multiple responses and/or multiple parts of variables.

References

Chambers JM, Hastie TJ (eds.) (1992). *Statistical Models in S*. Chapman & Hall, London.

Coelli T, Henningsen A (2010). ***frontier**: Stochastic Frontier Analysis*. R package version 0.996-6, URL http://CRAN.R-project.org/package=frontier.

Cribari-Neto F, Zeileis A (2010). "Beta Regression in R." *Journal of Statistical Software*, **34**(2), 1–24. URL http://www.jstatsoft.org/v34/i02/.

Croissant Y (2010). ***mlogit**: Multinomial Logit Model*. R package version 0.1-5, URL http://CRAN.R-project.org/package=mlogit.

Croissant Y, Millo G (2008). "Panel Data Econometrics in R: The **plm** Package." *Journal of Statistical Software*, **27**(2), 1–43. URL http://www.jstatsoft.org/v27/i02/.

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2006). "A Lego System for Conditional Inference." *The American Statistician*, **60**(3), 257–263.

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2008). "Implementing a Class of Permutation Tests: The **coin** Package." *Journal of Statistical Software*, **28**(8), 1–23. URL http://www.jstatsoft.org/v28/i08/.

Kleiber C, Zeileis A (2008). *Applied Econometrics with R*. Springer-Verlag, New York.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape (GAMLSS) in R." *Journal of Statistical Software*, **23**(7), 1–46. URL http://www.jstatsoft.org/v23/i07/.

Wilkinson GN, Rogers CE (1973). "Symbolic Description of Factorial Models for Analysis of Variance." *Applied Statistics*, **22**, 392–399.

Zeileis A, Hothorn T, Hornik K (2008a). "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics*, **17**(2), 492–514.

Zeileis A, Kleiber C, Jackman S (2008b). "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8), 1–25. URL http://www.jstatsoft.org/v27/i08/.

Affiliation:

Achim Zeileis
Department of Statistics
Universität Innsbruck
Universitätsstr. 15
6020 Innsbruck, Austria
E-mail: Achim.Zeileis@R-project.org
URL: http://statmath.wu.ac.at/~zeileis/

Yves Croissant
Université de la Réunion
Faculté de Droit et d'Economie
15, avenue René Cassin
BP7151
97715 Saint-Denis Messag Cedex 9, France
E-mail: Yves.Croissant@univ-reunion.fr

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 34, Issue 1 *Submitted:* 2009-09-23
April 2010 *Accepted:* 2010-03-31