Using car Functions in Other Functions

John Fox*& Sanford Weisberg†

March 22, 2016

Abstract

The car package (Fox and Weisberg, 2011) provides many functions that are applied to a fitted regression model, perform additional calculations on the model or possibly compute a different model, and then return values and graphs. In some cases, users may wish to write functions that call functions in car for a particular purpose. Because of the scoping rules used in R, several functions in car that work when called from the command prompt may fail when called inside another function. We discuss how users can modify their programs to avoid this problem.

1 deltaMethod

The car package includes many functions that require an object created by a modeling function like lm, glm or nls as input. For a simple example, the function deltaMethod uses the delta method (Fox and Weisberg, 2011, Sec. 4.4.6) to estimate the value and standard error of a nonlinear combination of parameter estimates. For example

```
library(car)
m1 <- lm(time ~ t1 + t2, Transact)
deltaMethod(m1, "t1/(t2 + 2)")
```

```
             Estimate     SE
t1/(t2 + 2)     1.354 0.1333
```

Here deltaMethod returns the standard error of the estimate of $\beta_1/(\beta_2+2)$, where β_j is the parameter corresponding to the regressor $\texttt{t}_j$. The code

```
ans <- NULL
for (z in 1:4) {
 ans <- rbind(ans, deltaMethod(m1, "t1/(t2 + z)",
     func = gsub("z", z, "t1/(t1+z)"))) }
ans
```

*Department of Sociology, McMaster University
†School of Statistics, University of Minnesota

```
          Estimate     SE
t1/(t1+1)   1.8000 0.1893
t1/(t1+2)   1.3538 0.1333
t1/(t1+3)   1.0849 0.1026
t1/(t1+4)   0.9051 0.0832
```

also works as expected. The `func` argument uses `gsub` to get the right row labels.

Consider the function:

```
f1 <- function(mod) {
 ans <- NULL
 for (x in 1:4) {
    ans <- rbind(ans, deltaMethod(mod, "t1/(t2 + x)",
        func = gsub("x", x, "t1/(t1+x)")) )}
 ans
 }
```

which simply puts the code used above into a function. Executing this function fails:

```
f1(m1)
```

```
Error in eval(expr, envir, enclos) : object 'x' not found
```

Worse yet, if `x` is defined in the same environment as `m1`, this function gives the wrong answer:

```
x <- 10
f1(m1)
```

```
          Estimate      SE
t1/(t1+1)   0.4539 0.03881
t1/(t1+2)   0.4539 0.03881
t1/(t1+3)   0.4539 0.03881
t1/(t1+4)   0.4539 0.03881
```

The core of the problem is the way that R does scoping. The regression object `m1` was created in the global environment, whereas the argument `z` in the `f1` function is created in the local environment of the function. The call to `deltaMethod` is evaluated in the global environment where `m1` is defined, leading to the error message if `z` does not exist in the global environment, and to wrong answers if it does exist.

For `deltaMethod`, there is an additional argument `constants` that can be used to fix the problem:

```
f2 <- function(mod) {
 ans <- NULL
 for (x in 1:4) {
```

```
    ans <- rbind(ans, deltaMethod(mod, "t1/(t2 + x)",
        func = gsub("x", x, "t1/(t1+x)"), constants=list(x=x)) )}
 ans
 }
f2(m1)
```

```
          Estimate     SE
t1/(t1+1)   1.8000 0.1893
t1/(t1+2)   1.3538 0.1333
t1/(t1+3)   1.0849 0.1026
t1/(t1+4)   0.9051 0.0832
```

The `constants` argument is a named list of quantities defined in the local function that are needed in the evaluation of `deltaMethod`.

2 ncvTest

The function `ncvTest` (Fox and Weisberg, 2011, Sec. 6.5.2) computes tests for non-constant variance in linear models as a function of the mean, the default, or any other linear function of regressors, even for regressors not part of the mean function. For example,

```
m2 <- lm(prestige ~ education, Prestige)
ncvTest(m2, ~ income)
```

```
Non-constant Variance Score Test
Variance formula: ~ income
Chisquare = 1.521    Df = 1     p = 0.2175
```

fits `prestige` as a linear function of `education`, and tests for nonconstant variance as a function of `income`, another regressor in the data set `Prestige`. Embedding this in a function fails:

```
f3 <- function(meanmod, dta, varmod) {
  m3 <- lm(meanmod, dta)
  ncvTest(m3, varmod)
  }
f3(prestige ~ education, Prestige, ~ income)
```

```
Error in is.data.frame(data) : object 'dta' not found
```

In this case the model `m3` is defined in the environment of the function, and the argument `dta` is defined in the global environment, and is therefore invisible when `ncvTest` is called. A solution is to copy `dta` to the global environment.

```
f4 <- function(meanmod, dta, varmod) {
   assign(".dta", dta, envir=.GlobalEnv)
   assign(".meanmod", meanmod, envir=.GlobalEnv)
```

```
    m1 <- lm(.meanmod, .dta)
    ans <- ncvTest(m1, varmod)
    remove(".dta", envir=.GlobalEnv)
    remove(".meanmod", envir=.GlobalEnv)
    ans
    }
 f4(prestige ~ education, Prestige, ~income)
```

```
Non-constant Variance Score Test
Variance formula: ~ income
Chisquare = 1.521    Df = 1     p = 0.2175
```

```
 f4(prestige ~ education, Prestige, ~income)
```


```
Non-constant Variance Score Test
Variance formula: ~ income
Chisquare = 1.521    Df = 1     p = 0.2175
```


The `assign` function copies the `dta` and `meanmod` arguments to the global environment where `ncvTest` will be evaluated, and the `remove` function removes them before exiting the function. This is an inherently problematic strategy, because an object assigned in the global environment will replace an existing object of the same name. Consequently we renamed the `dta` argument `.dta`, with an initial period, but this is not a *guarantee* that there was no preexisting object with this name.

This same method can be used with functions in the `effects` package. Suppose, for example, you want to write a function that will fit a model, provide printed summaries and also draw a effects plot. The following function will fail:

```
library(effects)
fc <- function(dta, formula, terms) {
 print(m1 <- lm(formula, .dta))
 Effect(terms, m1)
 }
form <- prestige ~ income*type + education
terms <- c("income", "type")
fc(Duncan, form, terms)
```

As with `ncvTest`, `dta` will not be in the correct environment when `Effect` is evaluated. The solution is to copy `dta` to the global environment:

```
library(effects)
fc.working <- function(dta, formula, terms) {
 assign(".dta", dta, env=.GlobalEnv)
 print(m1 <- lm(formula, .dta))
 Effect(terms, m1)
 remove(".dta", envir=.GlobalEnv)
 }
fc.working(Duncan, form, terms)
```

Assigning `formula` to the global environment is not necessary here because it is used by `lm` but not by `Effect`.

3 Boot

The `Boot` function in `car` provides a convenience front-end for the function `boot` in the `boot` package (Canty and Ripley, 2013; Fox and Weisberg, 2012). With no arguments beyond the name of a regression object and the number of replications `R`, `Boot` creates the proper arguments for `boot` for case resampling bootstraps, and returns the coefficient vector for each sample:

```
m1 <- lm(time ~ t1 + t2, Transact)
b1 <- Boot(m1, R=999)
summary(b1)
```

```
              R original bootBias  bootSE bootMed
(Intercept) 999   144.37  7.57729 197.740  153.50
t1          999     5.46  0.02221   0.693    5.49
t2          999     2.03 -0.00487   0.155    2.03
```

The returned object `b1` is of class `"boot"`, as are objects created directly from the `boot` function, so helper functions in the `boot` package and in `car` can be used on these objects, e.g.,

```
confint(b1)
```

```
Bootstrap quantiles, type =  bca

               2.5 %  97.5 %
(Intercept) -254.185 514.631
t1             3.789   6.652
t2             1.774   2.409
```

The `Boot` function would have scoping problems even without the user embedding it in a function because the `boot` function called by `Boot` tries to evaluate the model defined in the global environment in a local environment. In `car` we define an environment

```
.carEnv <- new.env(parent=emptyenv())
```

and then evaluate the model in the environment `.carEnv`. This environment is not exported, so to see that it exists you would need to enter

```
car:::.carEnv
```

```
<environment: 0x741c108>
```

We use this same trick in the `Boot.default` function so that `.carEnv` is globally visible. Here is a copy of `Boot.default` to show how this works.

```
Boot.default <- function(object, f=coef, labels=names(coef(object)),
                     R=999, method=c("case", "residual")) {
  if(!(require(boot))) stop("The 'boot' package is missing")
  f0 <- f(object)
  if(length(labels) != length(f0)) labels <- paste("V", seq(length(f0)), sep="")
  method <- match.arg(method)
  if(method=="case") {
     boot.f <- function(data, indices, .fn) {
      assign(".boot.indices", indices, envir=car:::.carEnv)
      mod <- update(object, subset=get(".boot.indices", envir=car:::.carEnv))
      if(mod$qr$rank != object$qr$rank){
            out <- .fn(object)
            out <- rep(NA, length(out)) } else  {out <- .fn(mod)}
     out
     }
    } else {
    boot.f <- function(data, indices, .fn) {
      first <- all(indices == seq(length(indices)))
      res <- if(first) object$residuals else
                  residuals(object, type="pearson")/sqrt(1 - hatvalues(object))
      res <- if(!first) (res - mean(res)) else res
      val <- fitted(object) + res[indices]
      if (!is.null(object$na.action)){
            pad <- object$na.action
            attr(pad, "class") <- "exclude"
            val <- naresid(pad, val)
            }
      assign(".y.boot", val, envir=car:::.carEnv)
      mod <- update(object, get(".y.boot", envir=car:::.carEnv) ~ .)
      if(mod$qr$rank != object$qr$rank){
            out <- .fn(object)
            out <- rep(NA, length(out)) } else  {out <- .fn(mod)}
      out
      }
  }
  b <- boot(data.frame(update(object, model=TRUE)$model), boot.f, R, .fn=f)
  colnames(b$t) <- labels
  if(exists(".y.boot", envir=car:::.carEnv))
     remove(".y.boot", envir=car:::.carEnv)
  if(exists(".boot.indices", envir=car:::.carEnv))
     remove(".boot.indices", envir=car:::.carEnv)
  b
  }
```

The was also fixed in `bootCase`.

References

Angelo Canty and Brian Ripley. boot: Bootstrap R (S-Plus) functions. R package version 1.3-9, 2013.

J. Fox and S. Weisberg. *An R Companion to Applied Regression.* Sage, Thousand Oaks CA, 2nd edition, 2011. URL http://z.umn.edu/carbook.

J. Fox and S. Weisberg. Bootstrapping regression models in R. Technical report, 2012. URL http://socserv.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Bootstrapping.pdf.