Using \texttt{car} Functions in Other Functions

John Fox∗& Sanford Weisberg†

February 20, 2015

Abstract

The \texttt{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 \texttt{car} for a particular purpose. Because of the scoping rules used in \texttt{R}, several functions in \texttt{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 \texttt{deltaMethod}

The \texttt{car} package includes many functions that require an object created by a modeling function like \texttt{lm}, \texttt{glm} or \texttt{nls} as input. For a simple example, the function \texttt{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

\begin{verbatim}
library(car)
m1 <- lm(time ~ t1 + t2, Transact)
deltaMethod(m1, "t1/(t2 + 2)")
\end{verbatim}

\begin{tabular}{lc}
\hline
Estimate & SE \\
\hline
t1/(t2 + 2) & 1.354 0.1333 \\
\hline
\end{tabular}

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

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

∗Department of Sociology, McMaster University
†School of Statistics, University of Minnesota
also works as expected. The \texttt{func} argument uses \texttt{gsub} to get the right row labels.

Consider the function:

\begin{verbatim}
  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
  }
\end{verbatim}

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

\begin{verbatim}
f1(m1)
\end{verbatim}

\texttt{Error in eval(expr, envir, enclos) : object \textquoteleft x\textquoteright not found}

Worse yet, if \texttt{x} is defined in the same environment as \texttt{m1}, this function gives the wrong answer:

\begin{verbatim}
x <- 10
f1(m1)
\end{verbatim}

\texttt{Estimate \quad SE}
t1/(t1+1) \quad 0.4539 \quad 0.03881
t1/(t1+2) \quad 0.4539 \quad 0.03881
t1/(t1+3) \quad 0.4539 \quad 0.03881
t1/(t1+4) \quad 0.4539 \quad 0.03881

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

For \texttt{deltaMethod}, there is an additional argument \texttt{constants} that can be used to fix the problem:

\begin{verbatim}
f2 <- function(mod) {
  ans <- NULL
  for (x in 1:4) {
    \texttt{\texttt{Estimate \quad SE}}
    t1/(t1+1) \quad 0.4539 \quad 0.03881
t1/(t1+2) \quad 0.4539 \quad 0.03881
t1/(t1+3) \quad 0.4539 \quad 0.03881
t1/(t1+4) \quad 0.4539 \quad 0.03881
  }
\end{verbatim}
\begin{verbatim}
ans <- rbind(ans, deltaMethod(mod, "t1/(t2 + x)",
func = gsub("x", x, "t1/(t1+x)"), constants=list(x=x)) )
ans
}
f2(m1)

\begin{tabular}{ll}
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 \\
\end{tabular}

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,

\begin{verbatim}
m2 <- lm(prestige ~ education, Prestige)
nvTest(m2, ~ income)
\end{verbatim}

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:

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

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.

\begin{verbatim}
f4 <- function(meanmod, dta, varmod) {
  assign(".dta", dta, envir=.GlobalEnv)
  assign(".meanmod", meanmod, envir=.GlobalEnv)
\end{verbatim}

3
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)
f4 <- function(dta, formula, terms) {
  print(m1 <- lm(formula, .dta))
  Effect(terms, m1)
}
form <- prestige ~ income*type + education
terms <- c("income", "type")
f4(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)
f4.working <- function(dta, formula, terms) {
  assign(".dta", dta, env=.GlobalEnv)
  print(m1 <- lm(formula, .dta))
  Effect(terms, m1)
  remove(".dta", envir=.GlobalEnv)
}
f4.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:

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

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>original</th>
<th>bootBias</th>
<th>bootSE</th>
<th>bootMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>999</td>
<td>144.37</td>
<td>17.2724</td>
<td>199.196</td>
<td>162.68</td>
</tr>
<tr>
<td>t1</td>
<td>999</td>
<td>5.46</td>
<td>0.0432</td>
<td>0.677</td>
<td>5.50</td>
</tr>
<tr>
<td>t2</td>
<td>999</td>
<td>2.03</td>
<td>-0.0127</td>
<td>0.155</td>
<td>2.02</td>
</tr>
</tbody>
</table>

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.,

```r
confint(b1)
```

Bootstrap quantiles, type = bca

2.5 %  97.5 %
(Intercept) -289.297  510.706
 t1       3.750   6.589
 t2       1.775   2.412

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

```r
.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

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

<environment: 0x5e704c0>

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$b$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.
