Using R to Interpret Interaction Effects in Statistical Models

Linear and nonlinear statistical models frequently involve interaction terms. These terms are often notoriously difficult to interpret when looking only at the summary of a model object. Here, I provide an overview of R packages and tools helpful to "tease apart" the effects in statistical models. The scope ranges from traditional linear models to generalized linear mixed and additive models.

What you will learn…

- R packages to interpret interaction terms: car, effects and Hmisc.
- Dealing with contrast matrices in R.
- Graphical tools for complex models: effects and xyplot.

What you should know…

- How to import data into R.
- Basic R syntax.

Parametric statistical models usually consist of a vector or a matrix of response variables Y and a set of explanatory variables. Ideally, the explanatory variables are controlled experimentally so that they are essentially measured without an error. Almost any model you fit will contain interaction terms, at least initially. For example, a simple model could be fit using the lm() command, and interactions would then be specified as y~a*b*c, where a, b and c are either numeric or categorical explanatory variables, and the star operator indicates interactions.

While it is easy to incorporate interactions in your models, it is much more difficult to interpret what they mean. For example, in the famous "airquality" dataset that ships with R´s standard distribution, the variable Ozone (a concentration expressed in ppb) was measured in New York between May and September 1973. The explanatory variables were solar radiation (Solar.R, numeric, in Langley), wind speed in mph (Wind, numeric), temperature in degrees F (Temp, numeric), the month (Month, numeric) and day of the month (Day, 1-31). Now, which factors would we expect to determine the Ozone concentration in New York City? Clearly, a high wind speed will likely decrease Ozone concentration, while temperature and radiation may be expected to interactively determine Ozone concentration. We could test this using a linear, nonlinear or even generalized additive model, including interaction terms in the explanatory variables.

Some preparations before we start

The “airquality” dataset contains information on the sampling dates at which the air quality measurements were taken. These need to be converted to a “Date” object in R. In addition, the dataset contains some missing values that we need to drop before proceeding with further analyses (as we are not interested in imputing the missing values at this time).

```
# remove missing values for variable “Ozone”
airquality=subset(airquality,complete.cases(Ozone))


# create a DateTime object
date.p=with(airquality,paste("1972",Month,Day,sep="-"))
date.d=as.Date(date.p,"%Y-%m-%d")
airquality$Date=date.d
```

Graphical inspection of the dataset (Listing 1)

We use the xyplot() function in the “lattice” library to inspect the data graphically. This should be the first step in any analysis – even before we run our first statistical models. The dataset contains only numeric variables. We therefore need to “cut” them into smaller pieces to be able to create graphs in two-dimensional space. These are called “shingles” and are available using the equal.count() algorithm in lattice:

```
# create a shingles for numeric covariates
require(lattice)

Solar.R.s=equal.count(airquality$Solar.R,2,overlap=0)
Wind.s=equal.count(airquality$Wind,2,overlap=0)
Temp.s=equal.count(airquality$Temp,2,overlap=0)
#Date.s=equal.count(airquality$Solar.R,2,overlap=0)
```

Now, we are ready to inspect the interactive effects of several potential explanatory variables on our response variable, “Ozone”. We use the xyplot() function with arguments type=c(“p”,”smooth”) to create points and local regression smoothers in each panel:

```
# graphically inspect single variables (check for
nonlinearity)

xyplot(Ozone~Date,airquality,type=c("p","smooth")) # quadratic
xyplot(Ozone~Wind,airquality,type=c("p","smooth")) # linear
xyplot(Ozone~Solar.R,airquality,type=c("p","smooth")) # linear
xyplot(Ozone~Temp,airquality,type=c("p","smooth")) #nonlinear
```

To systematically inspect up to 2-way interactions in the dataset, it is useful to use the combn() function. This function gives all pairwise combinations of a given set of entities:

```
variables=names(airquality[c(2,3,4,7)])
combn(variables,2)
```

```
     [,1]      [,2]      [,3]      [,4]   [,5]   [,6]
[1,] "Solar.R" "Solar.R" "Solar.R" "Wind" "Wind" "Temp"
[2,] "Wind"    "Temp"    "Date"    "Temp" "Date" "Date"
```

We now set up the interaction graphs in xyplot() using the "|" operator to condition each panel on the values of Wind, Temp or Solar. R created using the equal.count algorithm. For example, Wind.s now contains two "pseudo-levels" of low and high wind speed.

```
xyplot(Ozone~Solar.R|Wind.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Solar.R|Temp.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Wind|Temp.s,airquality,type=c("p","smooth"))

xyplot(Ozone~Date|Solar.R.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Date|Wind.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Date|Temp.s,airquality,type=c("p","smooth"))
```

We see that all variables have approximately linear effects on Ozone concentration, except for Date. Slight nonlinearities for solar radiation or wind speed will be ignored here to keep the analyses simpler.

Fitting an appropriate model (Listing 2)

Before we can fit an appropriate statistical model (to inspect it for significant interactions), we need to make sure that the errors (residuals, ε_i) will show constant variance, and (less importantly) that these are approximately normally distributed. But, we can inspect these errors only after we have fitted a particular model. A better solution is to look at the distribution of the individual data points: From the simple equalities (i) $var(y_i)=var(\varepsilon_i)=\sigma^2$ and (ii) $Cov(y_i)=Cov(\varepsilon_i)= \sigma^2 I$, where I is an identity matrix, we can see that looking at the distribution of y_i , it is sufficient to check if errors are independent of the mean and our response variable is normally distributed.

This can be done conveniently using the "fitdistrplus" package in R. Using the fitdist() function, we generate several objects and extract their AIC (Akaike´s information criterion). These objects form the so-called set of candidate models. From these candidate models, we select the one with the lowest AIC value; note that the absolute value of AIC is irrelevant. We find that, for these particular data, a Gamma

distribution for the response variable (Ozone) can be assumed. This distribution is frequently used to describe positive, real-valued data with high variance.

```
# load package to inspect distributional assumptions
require(fitdistrplus)

# plot empirical density of response variable
plot(density(na.exclude(airquality$Ozone)))

# indicates that data are right-skewed

# remove missing values and convert to numeric
Ozone=as.numeric(na.exclude(airquality$Ozone))

fit1 <- fitdist(Ozone,"norm")
fit2 <- fitdist(Ozone,"lnorm")
fit3 <- fitdist(Ozone,"pois")
fit4 <- fitdist(Ozone,"nbinom")
fit5 <- fitdist(Ozone,"gamma")

# combine fits into a list object
fit.list=list(fit1,fit2,fit3,fit4,fit5)

# extact AIC values
fit.aic=lapply(fit.list,function(x)summary(x)$aic)

# which distribution fits best?
which(fit.aic==min(as.numeric(fit.aic)))
```

Fitting an appropriate model (Listing 3)

We now know that a traditional linear model (lm) is not appropriate to model Ozone as a function of wind speed, temperature and solar radiation. The most appropriate model will be a generalized linear model with Gamma errors. The details are not important for now, but a slight complication arises as the coefficients of this model will be reciprocals (1/x). Every time we want to interpret any of the coefficients, we need to take its reciprocal (1/x) to understand which value it corresponds to on its original scale. Before we start the analysis, we need to load the “splines” package (that ships with R´s standard distribution). We use the bs() function to account for nonlinearities in Ozone concentration as a function of date. Furthermore, we exclude missing values from the “airquality” dataset and call our novel dataset “air2”:

```
require(splines)
air2=na.exclude(airquality)
```

We are ready to set up our generalized linear model with up to two-way interactions (the size of the dataset prevents us from incorporating higher-order interactions). This is done using the brackets (), followed by the “^2” operator that tells us “fit all main

effects and up to two-way interactions among these terms". We specify a "Gamma" family for the GLM:

```
model1=glm(Ozone~(bs(Date,3)+Solar.R+Wind+Temp)^2,air2,family=
"Gamma")
```

To automatically simplify our model based on Akaike´s information criterion (AIC), we use a self-written variant of the stepAIC function from R´s MASS library (see Listing 4). We then inspect the model using the plot() function (to look at residuals), the summary() function (to extract parameter estimates) and the Anova() function in the "car" library. The advantage of the Anova() function is that, the order of terms in the model is not important and we can assess the significance of all terms without order dependence:

```
model2=stepAICc(model1)
plot(model2)
Anova(model2)

Analysis of Deviance Table (Type II tests)

Response: Ozone
             LR Chisq Df Pr(>Chisq)
Solar.R        14.162  1  0.0001678 ***
Wind           20.855  1  4.955e-06 ***
Temp           45.187  1  1.791e-11 ***
Solar.R:Temp    2.355  1  0.1248504
Wind:Temp      20.708  1  5.348e-06 ***

summary(model2)

Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.738e-02  4.620e-02   0.593   0.5547
Solar.R      -2.937e-04  1.557e-04  -1.886   0.0621 .
Wind          1.631e-02  3.452e-03   4.723 7.22e-06 ***
Temp         -1.474e-04  5.455e-04  -0.270   0.7875
Solar.R:Temp  2.894e-06  1.904e-06   1.520   0.1315
Wind:Temp    -1.714e-04  3.997e-05  -4.289 4.00e-05 ***
```

These coefficients are interpreted as follows: The intercept is 9.97 ppm Ozone at a solar radiation of 0 Langley, a wind speed of 0 mph and a temperature of 0 °F. Note that this scenario is unrealistic (we will solve it later on):

```
1/coef(model1)[1]
```

Solar.R is the effect of Solar.R on Ozone for all other variables being 0. Wind is the effect of Solar.R on Ozone for all other variables being 0. Temp is the effect of Solar.R on Ozone for all other variables being 0.

The interactions are interpreted as follows:

Solar.R:Temp: Temp is the moderator variable (it changes the slope of Ozone vs. Solar.R). For every 1-unit change in Temp, the slope of Ozone vs. Solar.R is predicted to change by 1/Solar.R: Temp units

```
1/coef(model2)[5]
```

This gives an unrealistically high coefficient of 345597.3 ppb per Langley.

```
#[1 lan = 11.62 Wh per m²]
```

This means, the slope of Ozone vs. Solar.R will increase if Temp increases.

Wind:Temp: Temp is the moderator variable, it changes the slope of Ozone vs. Wind: For every 1-unit change in Temp, the slope of Ozone vs. Wind is predicted to change by 1/Wind:Temp units:

```
1/coef(model2)[6]
```

The estimate is -5833.009 ppb per mph; that is, the slope will become more negative if Temp increases.

Interpreting the interactions

Even in models that are only “moderately” complex, such as the one we use here, it soon becomes clear that interpreting the interactions can become a complex business. In general, there are two ways that always work if we want to interpret interaction terms: We either generate predictions at particular values of our explanatory variables or we graphically inspect them. For both of these exercises, the “effects” package is of tremendous help.

To look at the predicted Ozone concentrations at particular values of our explanatory variables, we need to vary one of the explanatory variables in turn, and set all others to particular values (often the mean value). We first need to generate these values. In our case, this turns out to be the mean solar radiation and the mean wind speed:

```
msr=mean(air2$Solar.R)
```

```
mw=mean(air2$Wind)
```

We are now ready to inspect the Solar.R:Temp interaction:

```
effect("Solar.R:Temp",model2,type="response",default.levels=2)
```

```
predict(model2,data.frame(Wind=mw,Solar.R=7,Temp=57),type="res
ponse")
```

```
predict(model2,data.frame(Wind=mw,Solar.R=7,Temp=97),type="res
ponse")

predict(model2,data.frame(Wind=mw,Solar.R=334,Temp=57),type="r
esponse")

predict(model2,data.frame(Wind=mw,Solar.R=334,Temp=97),type="r
esponse")
```

Now for the Wind:Temp interaction:

```
effect("Wind:Temp",model2,type="response",default.levels=2)

predict(model2,data.frame(Solar.R=msr,Wind=2.3,Temp=57),type="
response")

predict(model2,data.frame(Solar.R=msr,Wind=2.3,Temp=97),type="
response")

# plot these effects graphically:

require(car)
require(effects)

plot(allEffects(model2,default.levels=2),type="response",multi
line=T)
```

Note that these parameter estimates get unstable as we approach zero wind speed. In general, we can say that predictions for zero values of our explanatory variables (no wind, zero degree Fahrenheit\ and no sunlight) are unrealistic. It is therefore advisable to mean-center the explanatory variables, and re-run the model and all other commands using these new mean-centered variables. Intuitively, a value of zero would mean: an Ozone concentration at average wind speed, average temperature and average solar radiation. This makes sense in many situations:

now improve interpretability of these coefficients by mean-centering the explanatory variables:

```
scale(air2$Solar.R,scale=F)

air2$Solar.R.c=scale(air2$Solar.R,scale=F)
air2$Wind.c=scale(air2$Wind,scale=F)
air2$Temp.c=scale(air2$Temp,scale=F)

model3=glm(Ozone~(bs(Date,3)+Solar.R.c+Wind.c+Temp.c)^2,air2,f
amily="Gamma")
```

```
model4=stepAICc(model3)

plot(allEffects(model4,default.levels=2),type="response",multi
line=T)
```

Summary on analysis strategies in presence of interactions

We have seen that even a seemingly “simple” dataset such as the “airquality” dataset, can show complex properties such as nonlinearity in one or more of the explanatory variables or non-constant variance (as seen from the Gamma distribution in our case here). When faced with interaction terms, many researchers try to interpret the main effects only. However, this strategy is prone to errors. It is advisable to carefully interpret the summary() table and use extensively the effects() package. This allows a quick and graphical interpretation of interaction terms. Effects() objects allow further properties to be extracted such as the individual interaction means and their standard errors. Moreover, values on a transformed scale (such as the reciprocal here) can easily be converted to the original scale.

Some general recommendations

Whenever you are interested in interactions among explanatory variables, you should follow a simple four-step protocol described below:

(1) Graphically inspect your data using xyplot() and (if necessary) the equal.count() algorithm.
(2) Assess the distribution of your response variable using the fitdistrplus() package.
(3) Run models appropriate to your experimental design and to your response variable(s) .
(4) Assess interaction effects using the effects() package.

Note that multiple comparisons among treatment means are not possible in presence of interactions. The effects() package, and/or the calculation of predicted values by hand, combined with a careful interpretation of the summary table, are the only tools we have. Yet, these are powerful tools which are highly recommended.

<<LISTING 1>>

Listing 1. R code to inspect distributional assumptions in a dataset

```
# import dataset

data(airquality)
names(airquality)

# information on dataset can be retrieved by typing
?airquality

# load package to inspect distributional assumptions
require(fitdistrplus)

# plot empirical density of response variable
plot(density(na.exclude(airquality$Ozone)))

# indicates that data are right-skewed

# remove missing values for variable “Ozone”
airquality=subset(airquality,complete.cases(Ozone))

fit1 <- fitdist(Ozone,"norm")
fit2 <- fitdist(Ozone,"lnorm")
fit3 <- fitdist(Ozone,"pois")
fit4 <- fitdist(Ozone,"nbinom")
fit5 <- fitdist(Ozone,"gamma")

# combine fits into a list object
fit.list=list(fit1,fit2,fit3,fit4,fit5)

# extact AIC values
fit.aic=lapply(fit.list,function(x)summary(x)$aic)

# which distribution fits best?
which(fit.aic==min(as.numeric(fit.aic)))
```

<<LISTING 2>>

Listing 2. R code showing how to graphically inspect interactions in a complex dataset

```
# create a DateTime object

date.p=with(airquality,paste("1972",Month,Day,sep="-"))
date.d=as.Date(date.p,"%Y-%m-%d")
airquality$Date=date.d


# create a shingles for numeric covariates
require(lattice)

Solar.R.s=equal.count(airquality$Solar.R,2,overlap=0)
Wind.s=equal.count(airquality$Wind,2,overlap=0)
Temp.s=equal.count(airquality$Temp,2,overlap=0)
#Date.s=equal.count(airquality$Solar.R,2,overlap=0)

# graphically inspect single variables (check for
nonlinearity)

xyplot(Ozone~Date,airquality,type=c("p","smooth")) # quadratic
xyplot(Ozone~Wind,airquality,type=c("p","smooth")) # linear
xyplot(Ozone~Solar.R,airquality,type=c("p","smooth")) # linear
xyplot(Ozone~Temp,airquality,type=c("p","smooth")) #nonlinear

# graphically inspect interactions (max. 2-way)

variables=names(airquality[c(2,3,4,7)])
combn(variables,2)

     [,1]      [,2]      [,3]      [,4]   [,5]   [,6]
[1,] "Solar.R" "Solar.R" "Solar.R" "Wind" "Wind" "Temp"
[2,] "Wind"    "Temp"    "Date"    "Temp" "Date" "Date"


xyplot(Ozone~Solar.R|Wind.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Solar.R|Temp.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Wind|Temp.s,airquality,type=c("p","smooth"))

xyplot(Ozone~Date|Solar.R.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Date|Wind.s,airquality,type=c("p","smooth"))
xyplot(Ozone~Date|Temp.s,airquality,type=c("p","smooth"))
```

<<LISTING 3>> Statistical models in presence of interactions

```
require(splines)
air2=na.exclude(airquality)

model1=glm(Ozone~(bs(Date,3)+Solar.R+Wind+Temp)^2,air2,family=
"Gamma")
model2=stepAICc(model1)

Anova(model2)
summary(model2)

               Estimate Std. Error t value Pr(>|t|)
Intercept)    2.738e-02  4.620e-02   0.593   0.5547
Solar.R      -2.937e-04  1.557e-04  -1.886   0.0621 .
Wind          1.631e-02  3.452e-03   4.723 7.22e-06 ***
Temp         -1.474e-04  5.455e-04  -0.270   0.7875
Solar.R:Temp  2.894e-06  1.904e-06   1.520   0.1315
Wind:Temp    -1.714e-04  3.997e-05  -4.289 4.00e-05 ***

# the intercept is
1/coef(model1)[1] # 9.97 ppm Ozone at solar=0,Wind=0,Temp=0 [&
Date not part of the model]

# now interpret the parameters:
# Solar.R is the effect of Solar.R on Ozone for all other
variables being 0
# Wind is the effect of Solar.R on Ozone for all other
variables being 0
# Temp is the effect of Solar.R on Ozone for all other
variables being 0
# Solar.R:Temp: Temp is the moderator variable (it changes the
slope of Ozone vs. Solar.R)
# For every 1-unit change in Temp, the slope of Ozone vs.
Solar.R is predicted to change by
# 1/ Solar.R:Temp units:
1/coef(model2)[5]

# This gives an unrealistically high coefficient of 345597.3
ppb per Langley
#[1 lan = 11.62 Wh per m²]
# this means, the slope of Ozone vs. Solar.R will increase if
temp increases.

# Wind:Temp: Temp is the moderator variable, it changes the
slope pf Ozone vs. Wind:
# For every 1-unit change in Temp, the slope of Ozone vs. Wind
is predicted to change by
# 1/Wind:Temp units:
1/coef(model2)[6]
```

```
# the estimate is -5833.009 ppb per mph; that is, the slope
will become more negative if Temp increases

# now look at individual estimates (setting some parameters to
their mean
msr=mean(air2$Solar.R)
mw=mean(air2$Wind)

# Solar.R:Temp:
effect("Solar.R:Temp",model2,type="response",default.levels=2)
predict(model2,data.frame(Wind=mw,Solar.R=7,Temp=57),type="res
ponse")
predict(model2,data.frame(Wind=mw,Solar.R=7,Temp=97),type="res
ponse")
predict(model2,data.frame(Wind=mw,Solar.R=334,Temp=57),type="r
esponse")
predict(model2,data.frame(Wind=mw,Solar.R=334,Temp=97),type="r
esponse")

# Wind:Temp:
effect("Wind:Temp",model2,type="response",default.levels=2)
predict(model2,data.frame(Solar.R=msr,Wind=2.3,Temp=57),type="
response")
predict(model2,data.frame(Solar.R=msr,Wind=2.3,Temp=97),type="
response")

# parameter estimates get unstable as we approach 0 wind
speed.

# plot these effects graphically:

require(car)
require(effects)

plot(allEffects(model2,default.levels=2),type="response",multi
line=T)


# now improve interpretability of these coefficients by
mean-centering the explanatory variables:

air2=na.exclude(airquality)
head(air2)

scale(air2$Solar.R,scale=F)

air2$Solar.R.c=scale(air2$Solar.R,scale=F)
air2$Wind.c=scale(air2$Wind,scale=F)
air2$Temp.c=scale(air2$Temp,scale=F)
```

```
model3=glm(Ozone~(bs(Date,3)+Solar.R.c+Wind.c+Temp.c)^2,air2,f
amily="Gamma")
model4=stepAICc(model3)

summary(model4)
                     Estimate Std. Error t value Pr(>|t|)
Intercept)       3.274e-02  1.605e-03  20.394  < 2e-16 ***
Solar.R.c        -6.863e-05  1.742e-05  -3.939 0.000147 ***
Wind.c            2.969e-03  4.576e-04   6.488 2.93e-09 ***
Temp.c           -1.317e-03  1.503e-04  -8.760 3.66e-14 ***
Solar.R.c:Temp.c  2.894e-06  1.904e-06   1.520 0.131546
Wind.c:Temp.c    -1.714e-04  3.997e-05  -4.289 4.00e-05 ***

plot(allEffects(model4,default.levels=2),type="response",multi
line=T)
```

<<LISTING 4>> stepAICc function for model simplification based on Akaike´s information criterion, corrected for small sample sizes (AICc).

```
# StepAICc for lme models
#
# (1) extractAICc
# (2) dropterm.AICc
# (3) addterm.AICc
# (4) stepAICc
#
# Code originally written by B.D. Ripley and W.N. Venables
# with changes in the extractAICc function and others written
# by C. Scherber, 4th/5th May 2009. Adjusted to work for
# lm, glm and lme models on 8th October 2011.
#
# The function stepAICc may be used to perform AICc-based
# model selection
# if sample sizes are too small to use AIC.
#
# AICc is a version of AIC corrected for small sample sizes
# as defined in Burnham & Anderson, 2002.

extractAICc=function (fit, scale, k = 2, ...)
{

  if ( any(class(fit)=="lme")==T)
    if (fit$method != "ML")
      stop("AIC undefined for REML fit")

  res <- logLik(fit)
  edf <- attr(res, "df")
  n=length(residuals(fit))
  c(edf,-2 * res + k * edf+2*edf*(edf+1)/(n-edf-1))
}

dropterm.AICc=
  function (object, scope, scale = 0, test = c("none",
"Chisq"),
            k = 2, sorted = FALSE, trace = FALSE, ...)
  {
    tl <- attr(terms(object), "term.labels")
    if (missing(scope))
      scope <- drop.scope(object)
    else {
      if (!is.character(scope))
        scope <- attr(terms(update.formula(object, scope)),
                      "term.labels")
      if (!all(match(scope, tl, 0L)))
        stop("scope is not a subset of term labels")
```

```
    }
    ns <- length(scope)
    ans <- matrix(nrow = ns + 1L, ncol = 2L, dimnames =
list(c("<none>",

scope), c("df", "AIC")))
    ans[1, ] <- extractAICc(object, scale, k = k, ...)
    env <- environment(formula(object))
    n0 <- length(object$residuals)
    for (i in seq(ns)) {
      tt <- scope[i]
      if (trace) {
        message("trying -", tt)
        utils::flush.console()
      }
      nfit <- update(object, as.formula(paste("~ . -", tt)),
                     evaluate = FALSE)
      nfit <- eval(nfit, envir = env)
      ans[i + 1, ] <- extractAICc(nfit, scale, k = k, ...)
      if (length(nfit$residuals) != n0)
        stop("number of rows in use has changed: remove
missing values?")
    }
    dfs <- ans[1L, 1L] - ans[, 1L]
    dfs[1L] <- NA
    aod <- data.frame(Df = dfs, AIC = ans[, 2])
    o <- if (sorted)
      order(aod$AIC)
    else seq_along(aod$AIC)
    test <- match.arg(test)
    if (test == "Chisq") {
      dev <- ans[, 2L] - k * ans[, 1L]
      dev <- dev - dev[1L]
      dev[1L] <- NA
      nas <- !is.na(dev)
      P <- dev
      P[nas] <- safe_pchisq(dev[nas], dfs[nas], lower.tail =
FALSE)
      aod[, c("LRT", "Pr(Chi)")] <- list(dev, P)
    }
    aod <- aod[o, ]
    head <- c("Single term deletions", "\nModel:",
              deparse(as.vector(formula(object))))
    if (scale > 0)
      head <- c(head, paste("\nscale: ", format(scale), "\n"))
    class(aod) <- c("anova", "data.frame")
    attr(aod, "heading") <- head
    aod
  }
```

```
##
addterm.AICc=function (object, scope, scale = 0, test =
c("none", "Chisq"),
                       k = 2, sorted = FALSE, trace = FALSE,
...)
{
  if (missing(scope) || is.null(scope))
    stop("no terms in scope")
  if (!is.character(scope))
    scope <- add.scope(object, update.formula(object, scope))
  if (!length(scope))
    stop("no terms in scope for adding to object")
  ns <- length(scope)
  ans <- matrix(nrow = ns + 1L, ncol = 2L, dimnames =
list(c("<none>",

scope), c("df", "AIC")))
  ans[1L, ] <- extractAICc(object, scale, k = k, ...)
  n0 <- length(object$residuals)
  env <- environment(formula(object))
  for (i in seq(ns)) {
    tt <- scope[i]
    if (trace) {
      message("trying +", tt)
      utils::flush.console()
    }
    nfit <- update(object, as.formula(paste("~ . +", tt)),
                   evaluate = FALSE)
    nfit <- eval(nfit, envir = env)
    ans[i + 1L, ] <- extractAICc(nfit, scale, k = k, ...)
    if (length(nfit$residuals) != n0)
      stop("number of rows in use has changed: remove missing
values?")
  }
  dfs <- ans[, 1L] - ans[1L, 1L]
  dfs[1L] <- NA
  aod <- data.frame(Df = dfs, AIC = ans[, 2L])
  o <- if (sorted)
    order(aod$AIC)
  else seq_along(aod$AIC)
  test <- match.arg(test)
  if (test == "Chisq") {
    dev <- ans[, 2L] - k * ans[, 1L]
    dev <- dev[1L] - dev
    dev[1L] <- NA
    nas <- !is.na(dev)
    P <- dev
    P[nas] <- safe_pchisq(dev[nas], dfs[nas], lower.tail =
FALSE)
    aod[, c("LRT", "Pr(Chi)")] <- list(dev, P)
```

```
  }
  aod <- aod[o, ]
  head <- c("Single term additions", "\nModel:",
            deparse(as.vector(formula(object))))
  if (scale > 0)
    head <- c(head, paste("\nscale: ", format(scale), "\n"))
  class(aod) <- c("anova", "data.frame")
  attr(aod, "heading") <- head
  aod
}

##
stepAICc=function (object, scope, scale = 0, direction =
c("both",

"backward",

"forward"), trace = 1, keep = NULL, steps = 1000, use.start =
FALSE,
                   k = 2, ...)
{
  mydeviance <- function(x, ...) {
    dev <- deviance(x)
    if (!is.null(dev))
      dev
    else extractAICc(x, k = 0)[2L]
  }
  cut.string <- function(string) {
    if (length(string) > 1L)
      string[-1L] <- paste("\n", string[-1L], sep = "")
    string
  }
  re.arrange <- function(keep) {
    namr <- names(k1 <- keep[[1L]])
    namc <- names(keep)
    nc <- length(keep)
    nr <- length(k1)
    array(unlist(keep, recursive = FALSE), c(nr, nc),
list(namr,

namc))
  }
  step.results <- function(models, fit, object, usingCp =
FALSE) {
    change <- sapply(models, "[[", "change")
    rd <- sapply(models, "[[", "deviance")
    dd <- c(NA, abs(diff(rd)))
    rdf <- sapply(models, "[[", "df.resid")
    ddf <- c(NA, abs(diff(rdf)))
    AIC <- sapply(models, "[[", "AIC")
```

```
    heading <- c("Stepwise Model Path \nAnalysis of Deviance
Table",
                 "\nInitial Model:",
deparse(as.vector(formula(object))),
                 "\nFinal Model:",
deparse(as.vector(formula(fit))),
                 "\n")
    aod <- if (usingCp)
      data.frame(Step = change, Df = ddf, Deviance = dd,
                 `Resid. Df` = rdf, `Resid. Dev` = rd, Cp =
AIC,
                 check.names = FALSE)
    else data.frame(Step = change, Df = ddf, Deviance = dd,
                    `Resid. Df` = rdf, `Resid. Dev` = rd, AIC
= AIC,
                    check.names = FALSE)
    attr(aod, "heading") <- heading
    class(aod) <- c("Anova", "data.frame")
    fit$anova <- aod
    fit
  }
  Terms <- terms(object)
  object$formula <- Terms
  if (inherits(object, "lme"))
    object$call$fixed <- Terms
  else if (inherits(object, "gls"))
    object$call$model <- Terms
  else object$call$formula <- Terms
  if (use.start)
    warning("'use.start' cannot be used with R's version of
glm")
  md <- missing(direction)
  direction <- match.arg(direction)
  backward <- direction == "both" | direction == "backward"
  forward <- direction == "both" | direction == "forward"
  if (missing(scope)) {
    fdrop <- numeric(0)
    fadd <- attr(Terms, "factors")
    if (md)
      forward <- FALSE
  }
  else {
    if (is.list(scope)) {
      fdrop <- if (!is.null(fdrop <- scope$lower))
        attr(terms(update.formula(object, fdrop)), "factors")
      else numeric(0)
      fadd <- if (!is.null(fadd <- scope$upper))
        attr(terms(update.formula(object, fadd)), "factors")
    }
    else {
```

```
      fadd <- if (!is.null(fadd <- scope))
        attr(terms(update.formula(object, scope)), "factors")
      fdrop <- numeric(0)
    }
  }
  models <- vector("list", steps)
  if (!is.null(keep))
    keep.list <- vector("list", steps)
  if (is.list(object) && (nmm <- match("nobs", names(object),
                                       0)) > 0)
    n <- object[[nmm]]
  else n <- length(residuals(object))
  fit <- object
  bAIC <- extractAICc(fit, scale, k = k, ...)
  edf <- bAIC[1L]
  bAIC <- bAIC[2L]
  if (is.na(bAIC))
    stop("AIC is not defined for this model, so stepAIC cannot
proceed")
  nm <- 1
  Terms <- terms(fit)
  if (trace) {
    cat("Start: AICc=", format(round(bAIC, 2)), "\n",
        cut.string(deparse(as.vector(formula(fit)))),
        "\n\n", sep = "")
    utils::flush.console()
  }
  models[[nm]] <- list(deviance = mydeviance(fit), df.resid =
n -
                         edf, change = "", AIC = bAIC)
  if (!is.null(keep))
    keep.list[[nm]] <- keep(fit, bAIC)
  usingCp <- FALSE
  while (steps > 0) {
    steps <- steps - 1
    AIC <- bAIC
    ffac <- attr(Terms, "factors")
    if (!is.null(sp <- attr(Terms, "specials")) && !is.null(st
<- sp$strata))
      ffac <- ffac[-st, ]
    scope <- factor.scope(ffac, list(add = fadd, drop =
fdrop))
    aod <- NULL
    change <- NULL
    if (backward && length(scope$drop)) {
      aod <- dropterm.AICc(fit, scope$drop, scale = scale,
trace = max(0,

trace - 1), k = k, ...)
      rn <- row.names(aod)
```

```
      row.names(aod) <- c(rn[1L], paste("-", rn[-1L], sep = "
"))
      if (any(aod$Df == 0, na.rm = TRUE)) {
        zdf <- aod$Df == 0 & !is.na(aod$Df)
        nc <- match(c("Cp", "AIC"), names(aod))
        nc <- nc[!is.na(nc)][1L]
        ch <- abs(aod[zdf, nc] - aod[1, nc]) > 0.01
        if (any(ch)) {
          warning("0 df terms are changing AIC")
          zdf <- zdf[!ch]
        }
        if (length(zdf) > 0L)
          change <- rev(rownames(aod)[zdf])[1L]
      }
    }
    if (is.null(change)) {
      if (forward && length(scope$add)) {
        aodf <- addterm.AICc(fit, scope$add, scale = scale,
                             trace = max(0, trace - 1), k = k,
...)
        rn <- row.names(aodf)
        row.names(aodf) <- c(rn[1L], paste("+", rn[-1L],
                                           sep = " "))
        aod <- if (is.null(aod))
          aodf
        else rbind(aod, aodf[-1, , drop = FALSE])
      }
      attr(aod, "heading") <- NULL
      if (is.null(aod) || ncol(aod) == 0)
        break
      nzdf <- if (!is.null(aod$Df))
        aod$Df != 0 | is.na(aod$Df)
      aod <- aod[nzdf, ]
      if (is.null(aod) || ncol(aod) == 0)
        break
      nc <- match(c("Cp", "AIC"), names(aod))
      nc <- nc[!is.na(nc)][1L]
      o <- order(aod[, nc])
      if (trace) {
        print(aod[o, ])
        utils::flush.console()
      }
      if (o[1L] == 1)
        break
      change <- rownames(aod)[o[1L]]
    }
    usingCp <- match("Cp", names(aod), 0) > 0
    fit <- update(fit, paste("~ .", change), evaluate = FALSE)
    fit <- eval.parent(fit)
    if (is.list(fit) && (nmm <- match("nobs", names(fit),
```

```
                                        0)) > 0)
      nnew <- fit[[nmm]]
    else nnew <- length(residuals(fit))
    if (nnew != n)
      stop("number of rows in use has changed: remove missing
values?")
    Terms <- terms(fit)
    bAIC <- extractAICc(fit, scale, k = k, ...)
    edf <- bAIC[1L]
    bAIC <- bAIC[2L]
    if (trace) {
      cat("\nStep: AICc=", format(round(bAIC, 2)), "\n",
          cut.string(deparse(as.vector(formula(fit)))),
          "\n\n", sep = "")
      utils::flush.console()
    }
    if (bAIC >= AIC + 1e-07)
      break
    nm <- nm + 1
    models[[nm]] <- list(deviance = mydeviance(fit), df.resid
= n -
                           edf, change = change, AIC = bAIC)
    if (!is.null(keep))
      keep.list[[nm]] <- keep(fit, bAIC)
  }
  if (!is.null(keep))
    fit$keep <- re.arrange(keep.list[seq(nm)])
  step.results(models = models[seq(nm)], fit, object, usingCp)
}
```

About the author

Christoph Scherber has been using R for statistical computing for 15 years. He gives regular university courses on statistical computing with R and has published a wide range of scientific journals. Additionally, his videos about statistics on YouTube have received more than 300,000 visits.