LinRegInteractive: An R Package for the Interactive Interpretation of Linear Regression Models

Martin Meermeyer*
Schumpeter School of Business and Economics
Bergische Universität Wuppertal

27. August 2014

Abstract

The interpretation of the results for linear regression models with non-constant marginal effects can be tedious. For the multiple linear regression model this is the case when the covariates are incorporated in a nonparametric way. For binary response models this is true even for very simple linear link functions. Things becomes more complicated when categorical covariates are employed, especially when interaction effects between categorical covariates or between metric and categorical covariates are present. To facilitate the interpretation of such models the package provides two major functions, `glm.interactive()` and `lm.interactive()`, which allow the user to observe the effects of variations of the metric covariates in an interactive manner by means of termplots. Due to this interactive approach the functions provide an intuition about the mechanics of a particular model and are therefore useful particularly for educational purposes. Technically the functions are based on the package **rpanel** and the only mandatory argument is an appropriate fitted-model object. Given this, the linear predictors, the marginal effects and, for models with binary outcomes, the probabilities are calculated automatically. For the marginal effects a numerical approach is employed to handle nonlinear effects automatically. For printing purposes the user can customize the appearance of the termplots to a large extent. At the same time some tables capturing the model output can be printed to the R Console, optionally as copy-and-paste-ready LaTeX-code.

1 Introduction

The function `glm.interactive()` was developed to put suggestions for the interpretation of logit and probit models made by Hoetker (2007) into practice. The `lm.interactive()` is a byproduct of this effort. The implementation of the interactive GUI is based on the package **rpanel** (Bowman et al., 2007). Printing LaTeX-output to the console was inspired by the package **xtable** (Dahl, 2012), but an own implementation based on the `format()`-function is used. The noninteractive visualization of the results for various types of regression models can be done with the package **effects** (Fox, 2003).

*Bergische Universität Wuppertal, Schumpeter School of Business and Economics, Gaussstr. 20, 42119 Wuppertal, e-mail: meermeyer@statistik.uni-wuppertal.de

2 Basic idea

Even in the case of a simple linear predictor the interpretation of binary response models is nontrivial due to the nonlinearity of the response functions and the resulting nonconstant marginal effects. As pointed out by Hoetker (2007) the interpretation becomes more complex, if

- more than one categorical covariate is contained in the model,
- metric covariates are included nonparametrically,
- interaction effects between metric and categorical covariates are included.

For the three mentioned cases the interpretation of multiple regression models can be complex, too. Graphical displays are most often the best way to present the results of such models, especially if non-constant marginal effects are present.

The functions `glm.interactive()` and `lm.interactive()` facilitate the interpretation of these models by providing an interactive graphical representation of the results. For each metric covariate the linear predictors, the marginal effects and, for models with binary responses, the probabilities can be displayed by means of termplots. The values of the other metric covariates can be adjusted interactively and for the specified covariate constellation tables of the effects can be printed to the console. Especially in case of the three circumstances mentioned above the functions are helpful:

- If more than one categorical variable is contained in the model, the effects in the termplots are calculated for every combination of factor levels. Each combination is referred to as *group* and can be selected to be displayed or not.
- For metric covariates which are incorporated nonparametrically the direct (analytical) calculation of the marginal effects is tedious. Therefore, these are calculated numerically with `splinefun()`.
- The mechanics of interaction effects between metric and categorical covariates becomes obvious due to the separate treatment of the groups and the interactive nature of the functions.

3 Quick start

In terms of mandatory arguments the interfaces of the functions are kept as simple as possible. For the function `glm.interactive()` only a fitted-model object of class `glm` and for `lm.interactive()` a fitted-model object of class `lm` must be passed. For the fitted-model objects the following prerequisites must be met:

- The model must contain at least one metric covariate.
- The model must be specified with the formula interface and the data frame containing the variables must be passed with the `data` argument.
- The categorical variables must be `factors` (ordered or unordered).
- For `glm.interactive()`: the `family` of the fitted-model object must be `binomial`.

The functions may work as well for other classes of fitted-model objects. The requirements for this along with two examples are given in section 8. Some formulas in the call of `glm()` or `lm()` may cause a problem with the internal raw data extraction. A possible approach to solve this issue is also described in section 8.

For a suitable fitted-model object the basic usage is simple. This is demonstrated by means of a probit model for defaults of consumer credits. The dataset `creditdata` used for this is contained in the package and was originally obtained from the Data Archive of the Department of Statistics, University of Munich and of the SFB 386 (2014a). A probit model with 3 metric covariates and 2 factors with 2 and 3 levels respectively is used as example:

```
data("creditdata")
model.2.fac <- glm(credit ~ amount + I(amount^2)  + age + duration*teleph + housing,
    family = binomial(link="probit"), data = creditdata)
```

The variables are described in the documentation of the dataset `creditdata`. Given the fitted-model object the start is simple:

```
glm.interactive(model.2.fac)
```

Users of the IDE **RStudio** may need to change the graphic device with `options(device = "x11")` before calling the function because in current versions of **RStudio** multiple graphic devices occasionally do not work properly.[1] After calling the function the basic handling is as follows:

1. Select metric covariate to be displayed as termplot using the dialog (see figure 1, left). Within the IDE **RStudio** the selection dialog appears as text list in the integrated console (see figure 1, right).

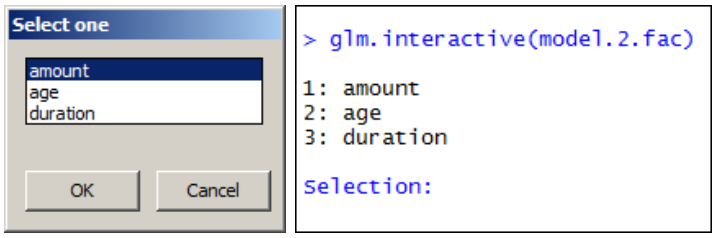

Figure 1: Dialog to select metric covariate

2. Within the GUI-panel (see figure 2, left):
   - Choose values of metric covariates by sliders.
   - Select type of termplot to be displayed in radiobox *Type*.
   - Select the level-combinations of the factors to be displayed in checkbox *Groups*.
   - Print tables to the R Console and optionally save the actual plot by pushing the button *Snapshot*.

Details on the tables produced by the *Snapshot*-button are explained in section 5. On the right hand side of figure 2 the initial termplot for the example model is shown. The plot is optimized for screens and needs to be shrunken here to fit on the page, therefore the annotations may be hard to read. The termplots are calculated over the span of the selected metric covariate. The first slider controls the value of the selected metric covariate which is used to calculate the output triggered by the *Snapshot*-button. The initial values of the sliders are by default the means of the metric covariates. When the model contains only one metric covariate no selection dialog shows up and no legend is added to the plot. Since the number of resulting groups can become quite large no confidence intervals are implemented yet to keep the plots clear.

[1] The graphic device is called with `dev.new(..., noRStudioGD = TRUE)`. During the tests of the package it turned out that this option is ignored on some machines for unknown reasons.

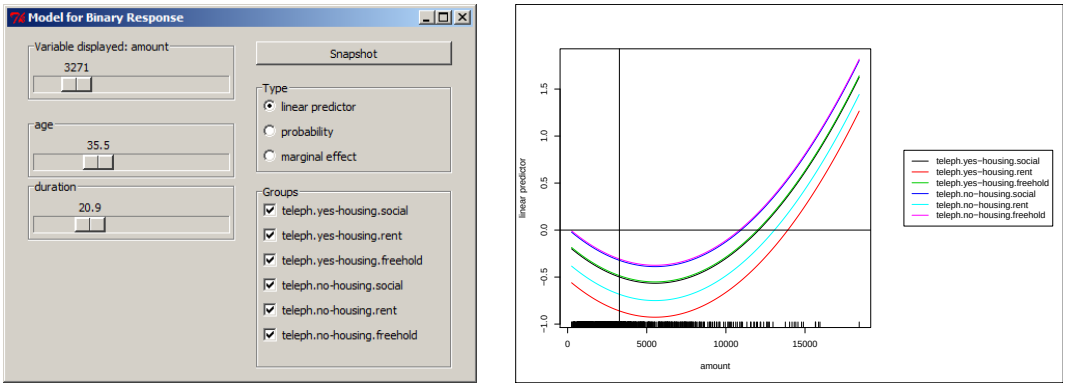

Figure 2: GUI-panel and initial termplot

The remainder of the vignette is structured as follows: In section 4 the benefits of the implemented visualization approach are demonstrated by means of examples. The details of the text output produced by the *Snapshot*-button are addressed in section 5. While the main focus of the functions relies on the interactive usage it is nevertheless easy to reproduce the results obtained by interaction. This is explained in section 6 and especially useful for publication purposes. For the same reason the format of the text output and the layout of the plots can be controlled to a large extent by a number of non-mandatory arguments which is described in-depth in section 7.1 and 7.2. To achieve even more flexibility the fundamental layout of the plots can be specified in advance which is shown in section 7.3. Additional arguments control the appearance of the GUI-panel which is addressed in section 7.4. In section 8 the requirements for other fitted-model objects to work with the functions are explained and a workaround if there occurs a problem with the raw data extraction.

4 Visualization of statistical concepts

In the light of the problems pointed out in section 2 the usefulness of the functions is demonstrated by means of examples in this section. Furthermore the problem of quasi-complete separation is addressed as last example. The plots in this section are customized for printing and the code to reproduce the figures is used as demo:

```
demo(vigngraphs, package = "LinRegInteractive", ask = FALSE)
```

Note that 16 PDF-files are stored in the actual working directory by calling the demo.

4.1 Nonlinear and nonparametric effects

In the example model of section 3 the covariate “amount” is included quadratically. Selecting this covariate to be displayed the strong nonlinear effect can be observed in the linear predictors, the responses and the marginal effects, these plots are shown in figure 3. For each of the 6 groups an individual line is plotted, the levels of the factor “teleph” are represented by shades of red and blue respectively and the levels of the factor “housing” by a decreasing hue of these colors. Note that for this particular model specification the levels “freehold” and “social” of the factor “housing” could be unified. Since model specification is not an issue here this will not be discussed further.

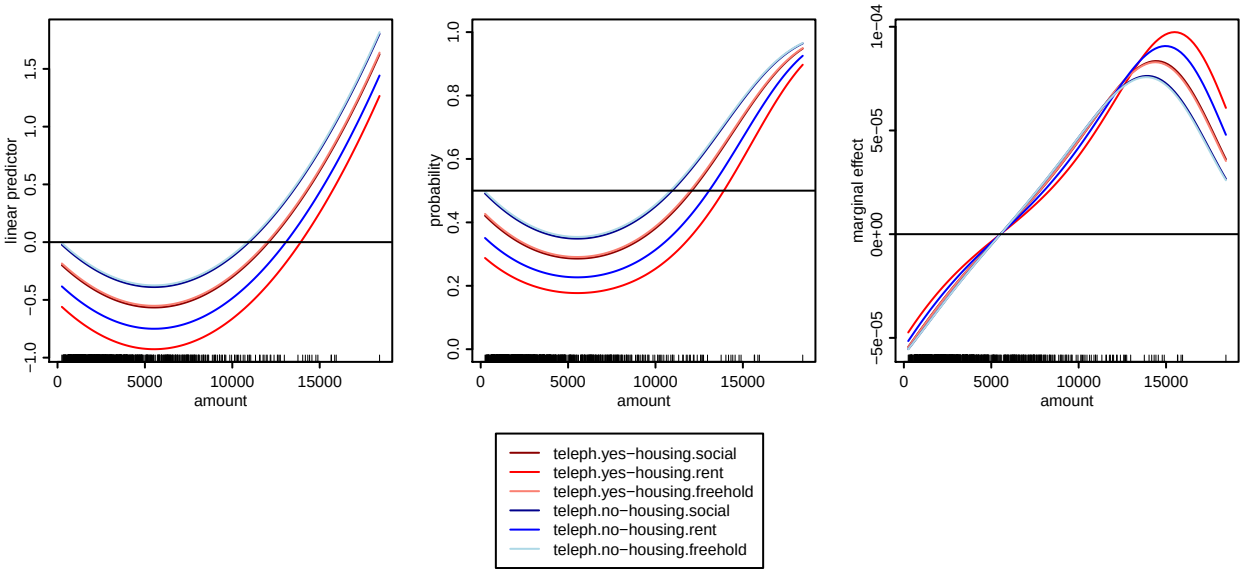

Figure 3: Quadratic effect of a metric covariate

To allow more flexibility in the fit the amount can be included nonparametrically by means of a spline function. The modified function call is

```
require("splines")
model.2.fac.npamount <- glm(credit ~ bs(amount) + age + duration*teleph + housing,
   family = binomial(link="probit"), data = creditdata)
glm.interactive(model.2.fac.npamount)
```

The resulting plots are arranged in figure 4. Since the legend is identical to figure 3 it is omitted here.

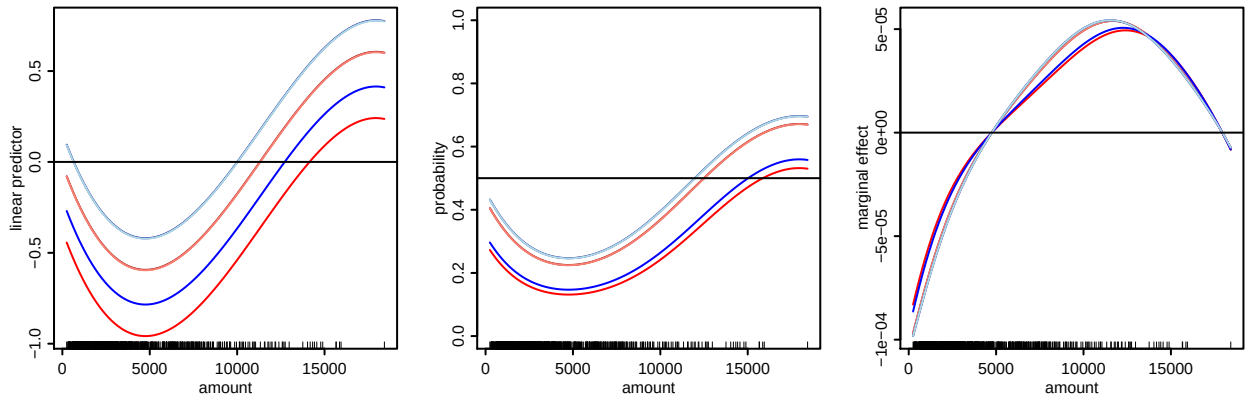

Figure 4: Nonparametric effect of a metric covariate

4.2 Interaction effects

The mechanics of interaction effects between metric and categorical covariates can easily be observed because in the termplots every group is represented by an individual line. In the example model from section 3 there is an interaction between the metric covariate “duration” and the factor “teleph”. By choosing “duration” to be displayed this interaction effect can be directly observed in the linear predictors, the responses and the marginal effects. The corresponding plots are shown in figure 5, for

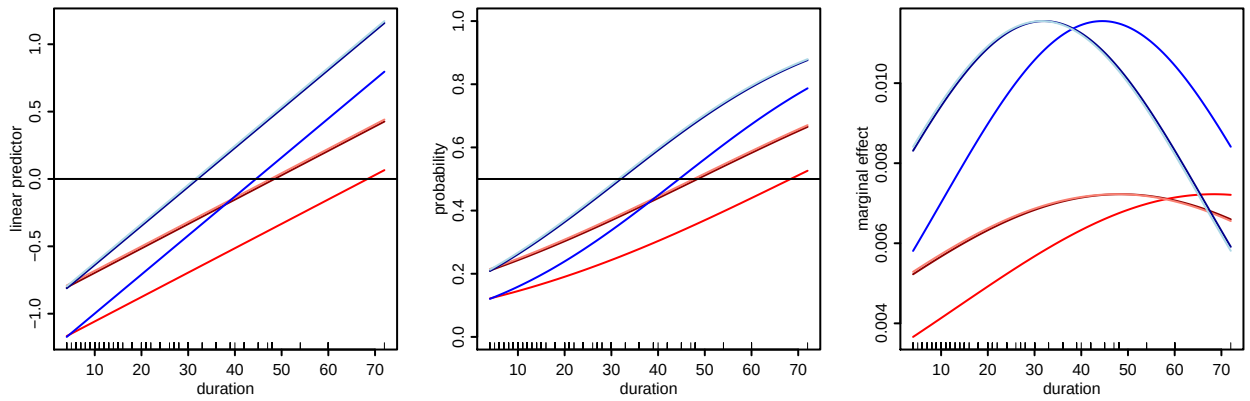

Figure 5: Interaction between a metric covariate and a factor - direct visualization

the plots the other metric covariates are left at their means. The legend is identical to figure 3 and therefore omitted.

Due to the interactive nature of the functions the interaction effect becomes also visible when other covariates are selected to be displayed. The top row of figure 6 shows the linear predictors, the responses and the marginal effects for the covariate "age", the covariate "amount" is left at the mean and the covariate "duration" is set to 12 months. The legend is identical to figure 3 and therefore omitted again. For the plots in the second row the covariate "duration" is set to 36 months. Due to the interaction effect the red lines literally move away from the blue lines compared to the plots in the first row and by this the interaction effect becomes visible indirectly.

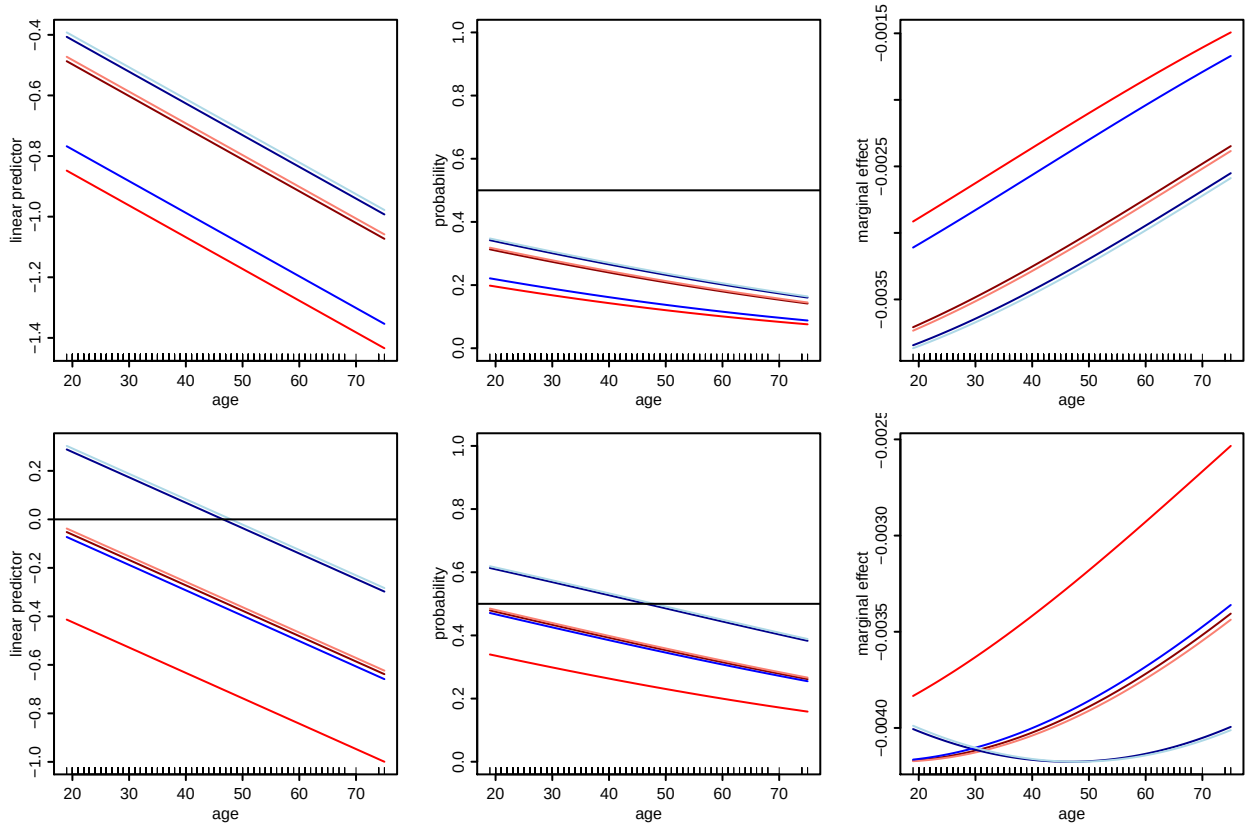

Figure 6: Interaction between a metric covariate and a factor - indirect visualization

4.3 Uncover quasi-complete separation

This issue is discussed in Kleiber, Zeileis (2008), p. 130ff by means of the `MurderRates`-data which originate from a study on the deterrent effect of capital punishment in the USA in 1950. The authors uncover the problem of quasi-complete separation by tracing a suspiciously large standard deviation of the coefficient estimate for the dummy variable representing the level "yes" of the factor "southern". It is pointed out that this issue is not uncommon with small data sets but is rarely discussed in textbooks and therefore can easily be missed. Using `glm.interactive()`, choosing for instance the covariate "income" and selecting the probability termplot (see figure 7) directly reveals the source of the problem even for inexperienced users:

```
require("AER")
data("MurderRates")
model <- glm(I(executions > 0) ~ time + income + noncauc + lfp + southern,
    data = MurderRates, family = binomial)
glm.interactive(model)
```

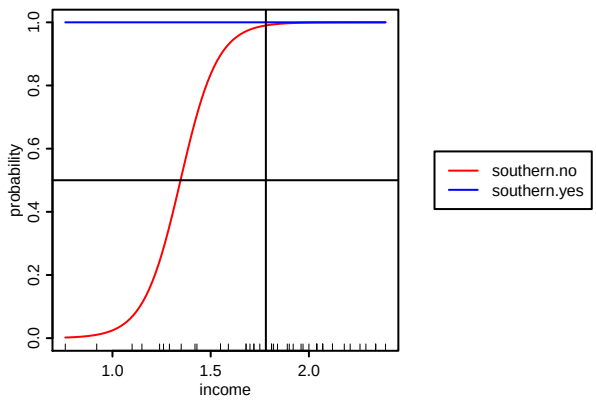

Figure 7: Quasi-complete separation in the *MurderRates*-data

5 Details on the text output

The following tables are calculated and printed to the console by clicking the *Snapshot*-button:

- Summary of the model obtained by `summary()` (not shown).
- Tables of the chosen values of the metric covariates and their ECDF-values (see table 1).

```
Actual values of metric covariates
  amount      age duration
3271.248   35.542   20.903

ECDF of actual values of metric covariates
  amount      age duration
   0.658    0.587    0.554
```

Table 1: Tables of the chosen values of the metric covariates and their ECDF-values

- For `glm.interactive()` the table of the link and response function at the chosen values of the metric covariates for each group (see table 2). For `lm.interactive()` the table of the effects at the chosen values of the metric covariates for each group.

	link	response
teleph.yes-housing.social	-0.4988935	0.3089272
teleph.yes-housing.rent	-0.8599626	0.1949048
teleph.yes-housing.freehold	-0.4843101	0.3140829
teleph.no-housing.social	-0.3219410	0.3737487
teleph.no-housing.rent	-0.6830100	0.2473003
teleph.no-housing.freehold	-0.3073576	0.3792856

Table 2: Table of the link and response function at the chosen values of the metric covariates for each group

- Table of marginal effects for each metric covariate at the chosen values of the metric covariates for each group (see table 3). The marginal effects are calculated numerically with `splinefun()`.

	marg.eff.amount	marg.eff.age	marg.eff.duration
teleph.yes-housing.social	-2.092169e-05	-0.003687023	0.006380069
teleph.yes-housing.rent	-1.637025e-05	-0.002884925	0.004992109
teleph.yes-housing.freehold	-2.107222e-05	-0.003713551	0.006425974
teleph.no-housing.social	-2.249765e-05	-0.003964754	0.010967022
teleph.no-housing.rent	-1.876479e-05	-0.003306914	0.009147348
teleph.no-housing.freehold	-2.260112e-05	-0.003982988	0.011017462

Table 3: Table of marginal effects for each metric covariate at the chosen values of the metric covariates for each group

When the argument `latex2console` is set to `TRUE` in the function call the tables are printed to the console as LaTeX-code. For the given example the tables 4 to 7 in the appendix show the results. Within a .tex-document the LaTeX-code works with the following preamble, note that the LaTeX-package `booktabs` is required:

```
\documentclass{scrartcl}
\usepackage{booktabs}
\begin{document}
% copy and paste from R Console
\end{document}
```

6 Details on different aspects of usage

By the arguments explained in this section different aspects of usage can be controlled. The model from section 3 is used as example and therefore the code chunks incorporate the `glm.interactive()`. For the `lm.interactive()` the arguments are identical.

Exact control over metric covariates With the sliders the values of the metric covariates cannot be selected with arbitrary precision. To allow exact control over the values these can be set as initial values for the sliders by the argument `initial.values` by means of a named list. The names in the list must exactly match the variable names:

```
glm.interactive(model.2.fac, initial.values = list(amount=5000, duration=24, age=30))
```

If a metric covariate is not explicitly listed the corresponding slider is initialized with the mean (the default). When the initial values should be the medians of the metric covariates this can be achieved by:

```
glm.interactive(model.2.fac, initial.values = list(amount=median(creditdata$amount),
    duration=median(creditdata$duration), age=median(creditdata$age)))
```

Note that the ECDF-values in the text output do not exactly match 0.5 for the metric discrete covariates.

Preselect metric covariate, plot type and groups To avoid the selection menu to appear the name of the metric covariate to be displayed can be preselected:

```
glm.interactive(model.2.fac, preselect.var = "duration")
```

If no metric covariate with the name provided exists the selection menu will pop up instead.

The type of plot to be displayed first can also be specified by the argument `preselect.type`. For the function `glm.interactive()` this must be one of the values `"link"` (the default), `"response"` or `"marginal"`:

```
glm.interactive(model.2.fac, preselect.type = "response")
```

For the function `lm.interactive()` the values `"effect"` (the default) and `"marginal"` are valid.

By default all groups are active in the initial plot. With the argument `preselect.groups` the groups which are displayed in the initial plot can be specified by means of a numeric index vector. The first three groups are preselected by

```
glm.interactive(model.2.fac, preselect.groups = c(1:3))
```

Preselecting groups is useful if the model contains many factors. In this case the panel usually grows beyond the screen and some groups are not accessible via the GUI-panel any more. The groups are constructed with the function `factor.combinations()` which can be used to identify groups of interest. This is illustrated in example B1 in appendix B.

The functionality to prespecify the variable, the plot type and certain groups in advance is beneficial when used in conjunction with the argument `initial.values` and the automatic save functionality. Altogether these allow the user to reproduce plots without user interaction, see the example in the last paragraph of this section.

Save plots When the plot is saved as PDF- or EPS-file there is sometimes a problem concerning the width of the legend annotations. Depending on the font family and the fontsize in some cases the legend annotations do not fit into the box around the legend in the graphics file even though these do so in the graphics window. A workaround for this is to increase the width of the legend with the argument `legend.width.factor`, which is `1` by default. If this problem occurs the width of the legend can be for instance increased about 10% by

```
glm.interactive(model.2.fac, legend.width.factor = 1.1)
```

When the argument `snapshot.plot` is set to `TRUE` the current plot is saved to the working directory by `SavePlot()` when the *Snapshot*-button is pressed. Note that in this case the RGui-window becomes the active window. The advantage to save plots this way instead of using the functionality of the RGui is the handling of the file name. The file name can be specified in advance by the argument `graphics.filename` and a hyphen followed by a sequential number with 3 digits is added to avoid that existing plots are overwritten. The path can also be specified within the filename. In addition the file format can be set by the argument `graphics.extension` which is passed to the `type`-argument in `SavePlot()` (default to `"pdf"`). To save plots of the covariate "duration" to the directory D:\Temp by the *Snapshot*-button one may use

```
glm.interactive(model.2.fac,
    preselect.var     = "duration",
    snapshot.plot     = TRUE,
    graphics.filename = "D:/Temp/fig-credprobit-duration")
```

By setting the argument `autosave.plot` to `TRUE` the initial plot is saved and the GUI-control is closed immediately after initialization. In conjunction with the arguments `initial.values`, `preselect.var`, `preselect.type` and `preselect.groups` this can be used to reproduce plots without user interaction. In case of the example model with 2 factors the plot of the marginal effects with a prespecified covariate constellation and a subset of the groups is saved to the actual working directory by

```
glm.interactive(model.2.fac,
    initial.values      = list(amount=5000, duration=24, age=30),
    preselect.var       = "duration",
    preselect.type      = "marginal",
    preselect.groups    = c(2,3,5,6),
    autosave.plot       = TRUE,
    graphics.filename   = "fig-credprobit-duration-marg",
    legend.width.factor = 1.05)
```

Note that the legend width needs to be increased here for the PDF-files to look as expected. The figures of section 4 are created in this reproducible way, the code can be found in the demo of the package.

7 Details on customization

The appearance of the GUI-controls, the text output and the plots can be customized in many ways. The most important ones are demonstrated for `glm.interactive()` in this section. For `lm.interactive()` most of the arguments are identical. If there are deviations this is explicitly mentioned. Two probit models are used for demonstration throughout this section. Firstly the model introduced in section 3:

```
data("creditdata")
model.2.fac <- glm(credit ~ amount + I(amount^2)  + age + duration*teleph + housing,
    family = binomial(link="probit"), data = creditdata)
```

Additionally the following probit model with 3 metric covariates and 3 factors with 2, 3 and 4 levels respectively is also used occasionally:

```
data("creditdata")
model.3.fac <- glm(credit ~ amount + I(amount^2)  + age + duration*teleph + housing + job,
    family = binomial(link="probit"), data = creditdata)
```

7.1 Customize text output

Separation characters in group names Within the group names which are used in the legend and the text output the character separating the factor name and the corresponding factor levels can be set with the argument `level.sep` (default to "."). The character separating factor-factor level combinations can be set with the argument `factor.sep` (default to "-"):

```
glm.interactive(model.2.fac,
    factor.sep = "|",
    level.sep  = ">")
```

Decimal mark and big mark in LaTeX-output Changing the LaTeX-output to continental European number formats can be achieved by

```
glm.interactive(model.2.fac,
    latex2console = TRUE,
    decimal.mark  = ",",
    big.mark      = ".")
```

For the LaTeX-code printed to the console the following preamble (here prepared for German language) for a utf8-encoded .tex-file works well:

```
\documentclass[a4paper]{scrartcl}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenx}
\usepackage[ngerman]{babel}
\usepackage[babel,german=quotes]{csquotes}
\usepackage{icomma}
\usepackage{booktabs}
\begin{document}
% copy and paste from R Console
\end{document}
```

Scientific notation and number of digits The number of digits and the behaviour when to switch to scientific notation in the text output (both plain text and LaTeX-output) can be modified by the R options `digits` and `scipen`, see the documentation of `options()` for details. The optimal setting depends on the given model and personal preferences. Changing the options in the R Console directly affects the number formatting and can therefore be modified between two clicks on the *Snapshot*-button.

7.2 Customize graphic device

Many graphical elements can be directly controlled by arguments in the function call. The major appearance of the plots can be controlled by manipulation of `par()`-arguments. If necessary graphical elements can be added to the plots by low-level plotting commands in the R Console. Note that elements added this way are not captured by the autosave functionality described at the end of section 6. The example B2 in appendix B shows how plots can nevertheless be saved without user interaction.

Plot dimension and pointsize The dimension of the graphic device (in centimeters) and the pointsize of plotted text can be set. A more compact plot is obtained by:

```
glm.interactive(model.2.fac,
    dev.height       = 11,
    dev.width        = 11,
    dev.width.legend = 5,
    dev.pointsize    = 8)
```

When a legend is added the overall width of the device is `dev.width` plus `dev.width.legend`. By default a legend is added when at least one factor is used as covariate. For more details on the legend refer to the paragraph “Legend” in this section.

Colors, line types and line widths The colors, line types and line widths for the line representing different groups in the plots and the legend can be set directly. For the model with 2 factors the following scheme makes sense:

```
glm.interactive(model.2.fac,
    lwd = rep(c(1,2),each=3),
    col = c(1,"blue",2))
```

Note that the arguments are recycled if necessary, in this case for the color. The levels of the factor "teleph" are represented by the thickness of lines and the levels of the factor "housing" by the color. For the model with 3 factors one may use

```
glm.interactive(model.3.fac,
    lwd = rep(c(1,2), each=12),
    col = rep(c(1,"blue",2), each=4),
    lty = c(1,2,3,4),
    dev.width.legend = 8)
```

The levels of the additional factor "job" are represented by the linetype here. The visual discrimination of 4 or more factors will be conceptually hard to achieve. In this case it may be advisable to display only a subset of groups by deselecting some of these in the checkbox or by using the argument `preselect.groups`.

Title and axis labels By default no title is added to the plot, the label for the x-axis is the name of the selected covariate and the label for the y-axis is the name of the selected plot type. This can be overridden if necessary:

```
glm.interactive(model.2.fac,
    preselect.var  = "duration",
    preselect.type = "response",
    main           = "Interaction between 'duration' and factor 'teleph'",
    xlab           = "duration (months)",
    ylab           = "probabilities")
```

Legend A legend is added by default when at least one factor is used as covariate. The legend is plotted within an own region with the left and right margin of the legend region set to 0. The legend frequently needs to be modified since the space required by the legend depends on the number of factors used as covariates, the lengths of the group names, the physical screen resolution and the size of the RGui-window (if run in MDI-mode, not relevant for users of **RStudio**). On the one hand the space itself can be modified and on the other hand the magnification of the legend. The first solution for the model with 3 factors is

```
glm.interactive(model.3.fac, dev.width.legend = 8)
```

For the same model reducing the magnification to 70% of the original size also works:

```
glm.interactive(model.3.fac, legend.cex = 0.7)
```

The position of the legend can be modified as well, refer to the documentation of `legend()` for details:

```
glm.interactive(model.2.fac, legend.pos = "top")
```

With the argument `legend.width.factor` the width of the box surrounding the legend can be manipulated, see the explanations in the paragraph "Save plots" in section 6.

When factors are present the legend and the corresponding plot region for it can be completely suppressed with

```
glm.interactive(model.2.fac, legend.add = FALSE)
```

Setting the additional argument `legend.space` to `TRUE` will create the corresponding plot region without the legend:

```
glm.interactive(model.2.fac,
    legend.add   = FALSE,
    legend.space = TRUE)
```

This can be useful if different plots are arranged in a document but for only one of the plots a legend should appear. The empty spaces ensure exact alignments and matching plot dimensions in this case. Figure 3 in section 4 is an example for this, the code to reproduce the figure can be found in the demo. To achieve full flexibility for the arrangement of plots in a document the legend can be plotted alone:

```
glm.interactive(model.2.fac, legend.only = TRUE)
```

Note that in this case the width of the graphic device is solely controlled by `dev.width.legend` and in conjunction with `dev.height` the overall size of the legend can be set precisely.

Rug plot By default a rug representation of the selected metric covariate is added to the southern axis of the plot. The length of the ticks can be controlled with the argument `rug.ticksize` which is `0.02` by default. For many observations the rug considerably slows down the rebuild of the plot. Setting `rug.ticksize` to `0` or `NA` removes the rug representation:

```
glm.interactive(model.2.fac, rug.ticksize = NA)
```

The color of the rug tickmarks can be changed with the argument `rug.col`:

```
glm.interactive(model.2.fac, rug.col = "gray50")
```

If more detailed control over the rug is needed, the rug needs to be suppressed and added with `rug()` from the R Console. The example B2 in section B shows how to add a customized rug plot with transparent colors.

Vertical and horizontal lines To aid visual perception some straight lines are added to the plots by default. A vertical black line shows the actual value of the selected metric covariate. To suppress the vertical line, e. g. for printing the plot, use

```
glm.interactive(model.2.fac, vline.actual = FALSE)
```

For `glm.interactive()` horizontal black lines are drawn in the plot of the linear predictor at height `0`, in the plot of the response at height `0.5` and in the plot of the marginal effect at height `0` by default. These lines can be suppressed by setting the corresponding arguments to `NA`:

```
glm.interactive(model.2.fac,
    pos.hline.link     = NA,
    pos.hline.response = NA,
    pos.hline.marginal = NA)
```

With the same arguments the position of the lines can be changed. For the plot of the response this makes sense if another probability threshold than 0.5, say 0.56, gives a better prediction accuracy:

```
glm.interactive(model.2.fac, pos.hline.response = 0.56)
```

For the `lm.interactive()` the corresponding arguments are `pos.hline.effect` (default to `0`) and `pos.hline.marginal` (default to `0`). The appearance of the lines (solid black lines) cannot be changed by arguments. To modify the lines these must be suppressed and can be added with `abline()`-commands from the R Console.

Number of points used for plotting The effects are drawn for a sequence of equally spaced points over the span of the chosen metric covariate. With the argument `n.effects` the number of points can be controlled. If the lines of the effects are not smooth this value can be increased.

7.3 Predefining the graphic device

To allow for more control over the appearance of the plots the graphic device can be specified in advance. Two plot regions which can be accessed with high-level plotting commands are required. The legend is plotted first and the left and right margin of the legend region are set to 0. In the following example the legend appears on the left side, the colors are specified via `palette()` and the margins as well as the magnification of the text are changed. Note that for pointsize 10 the width of the legend must be increased to achieve that the legend annotations fit into the box of the legend in the PDF-file.

```
windows(10,7, pointsize = 10)
layoutmatrix <- matrix(c(1,2,2), 1, 3)
layout(layoutmatrix)
palette(c("darkred","red","salmon","darkblue","blue","lightblue"))
par(cex = 1, mar = c(5,5,2,2)+0.1)

glm.interactive(model.2.fac,
    preselect.var       = "amount",
    preselect.type      = "response",
    dev.defined         = TRUE,
    legend.width.factor = 1.1,
    snapshot.plot       = TRUE)
```

7.4 Customize GUI-controls

Size of the GUI-controls The size of the entire panel is calculated automatically and primarily depends on the number of covariates and groups. The layout of the panel can be modified by a number of parameters. This can be useful for screens with a low resolution or if the model has a lot of groups. For the latter case the parameters are changed in the following example to save space:

```
glm.interactive(model.3.fac,
    box.type.height           = 90,
    box.group.character.width = 6,
    box.group.line.height     = 25,
    dist.obj.height           = 2)
```

Note that for a large number of groups not every group can be seen in the GUI-panel because the panel grows beyond the screen margin. In former versions of **rpanel** (< 1.1-3) the entries of the checkbox were squeezed together. Currently there is no way to circumvent this problem. The only solution at the moment is to choose the groups in advance using the argument `preselect.groups`, see the paragraph "Preselect metric covariate, plot type and groups" in section 6.

Annotations of the GUI-controls The annotations of the GUI-controls can be changed easily, for instance to German:

```
glm.interactive(model.2.fac,
    panel.title      = "Probit Modell",
    label.button     = "Schnappschuss",
    label.slider.act = "Dargestellte Variable: ",
    label.box.type   = "Typ",
    label.types      = c("Linearer Praediktor", "Wahrscheinlichkeit", "Marginaler Effekt"),
    label.box.groups = "Gruppen")
```

For `lm.interactive()` the argument `label.types` needs to be a character vector of length 2.

8 Other fitted-model objects and problems with raw data extraction

Other classes of fitted-model objects For other classes of fitted-model objects than `glm` for the function `glm.interactive()` and `lm` for the `lm.interactive()` both functions may work as well. Technically there are two requirements for this:

1. From the fitted-model object, say `model`, the raw data must be extractable with `get_all_vars(model$terms, model$data)` or `get_all_vars(model$terms, model$model)`. If there is a problem with the extraction of the raw data the workaround described in the next paragraph may help.
2. Appropriate prediction methods must exist. For `glm.interactive()` these are `predict(model, newdata, type = "link")` and `predict(model, newdata, type = "response")` and for `lm.interactive()` this is `predict(model, newdata)`.

Of course the four prerequisites stated in section 3 must be met, too.

For instance a fitted model object obtained by `gam()` from the package **gam** (Hastie, 2013) is compatible:

```
data("creditdata")
require("gam")
model.gam <- gam(credit ~ s(amount) + lo(age) + duration*teleph + housing,
    family = binomial, data = creditdata)
glm.interactive(model.gam)
```

A fitted-model object of the function `gam()` from the package **mgcv** (Wood, 2014) works as well:

```
data("creditdata")
require("mgcv")
model.mgcv <- gam(credit ~ s(amount) + s(age) + duration*teleph + housing,
family = binomial, data = creditdata)
glm.interactive(model.mgcv)
```

Note that for stability reasons the table of coefficients is usually omitted for other classes of fitted-model objects if the argument `latex2console` is set to `TRUE`.

Solving problems with raw data extraction In fitted-model objects of class `lm` there is no `data` slot. Therefore the raw data must be extractable with `get_all_vars(model$terms, model$model)`. For some formulas this extraction method does not work properly, for instance if a covariate is incorporated as spline function with `bs()`. Adding a `data` slot to the fitted-model object afterwards solves this particular problem. This is demonstrated by means of a linear regression model for the rents of

apartments in Munich, Germany. The dataset `munichrent03` is contained in the package and was originally obtained from the Data Archive of the Department of Statistics, University of Munich and of the SFB 386 (2014b). The variables are described in the documentation of the dataset `munichrent03`:

```
data("munichrent03")
require("splines")
model.rent <- lm(rent ~ bs(yearc) + area*location + upkitchen, data=munichrent03)
model.rent$data <- munichrent03
lm.interactive(model.rent)
```

Adding a `data` slot afterwards may also be helpful if the raw data cannot be extracted for other reasons.

References

Bowman, A., Crawford, E., Alexander, G., and Bowman, R. (2007). rpanel: Simple interactive controls for R functions using the tcltk package. *Journal of Statistical Software*, 17(9):1–18.

Dahl, D. B. (2012). *xtable: Export tables to LaTeX or HTML*. R package version 1.7-0.

Data Archive of the Department of Statistics, University of Munich and of the SFB 386 (2014a). Dataset for credit defaults: http://www.stat.uni-muenchen.de/service/datenarchiv/kredit/kredit.html.

Data Archive of the Department of Statistics, University of Munich and of the SFB 386 (2014b). Dataset for rents of apartments in Munich, Germany: http://www.stat.uni-muenchen.de/service/datenarchiv/miete/miete03.html.

Fox, G. (2003). Effect Displays in R for Generalised Linear Models. *Journal of Statistical Software*, 8(15), 1–27.

Hastie, T. (2013). gam: Generalized Additive Models. R package version 1.09.1. http://CRAN.R-project.org/package=gam.

Hoetker, G. (2007). The use of logit and probit models in strategic management research: Critical issues. *Strategic Management Journal*, 28(4), 331–343.

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

Wood, S. (2014). mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation. R package version 1.7-29. http://CRAN.R-project.org/package=mgcv.

A Example of LaTeX text output

The four tables in this section are the default output for the model given in section 3 with the argument `latex2console` set to `TRUE`.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-2.856711e-01	2.452085e-01	-1.16501299	0.2440137362
amount	-1.456049e-04	5.318493e-05	-2.73771078	0.0061868458
I(amount^2)	1.317726e-08	3.910163e-09	3.37000201	0.0007516764
age	-1.046677e-02	4.196524e-03	-2.49415198	0.0126258471
duration	1.811182e-02	6.508570e-03	2.78276549	0.0053897748
telephno	-4.964901e-02	1.900688e-01	-0.26121597	0.7939259612
housing.L	1.031201e-02	1.220057e-01	0.08452066	0.9326424767
housing.Q	3.007653e-01	7.867191e-02	3.82303248	0.0001318204
duration:telephno	1.084062e-02	7.507286e-03	1.44401356	0.1487351600

Table 4: Model coefficients

variable	amount	age	duration
value	3,271.248	35.542	20.903
ECDF(value)	0.658	0.587	0.554

Table 5: Values of metric covariates where results are calculated

groups	link	response
teleph.yes-housing.social	-0.4988935	0.3089272
teleph.yes-housing.rent	-0.8599626	0.1949048
teleph.yes-housing.freehold	-0.4843101	0.3140829
teleph.no-housing.social	-0.3219410	0.3737487
teleph.no-housing.rent	-0.6830100	0.2473003
teleph.no-housing.freehold	-0.3073576	0.3792856

Table 6: Linear predictor and response in different groups at given values of metric covariates

	amount	age	duration
value	3,271.248	35.542	20.903
ECDF(value)	0.658	0.587	0.554
teleph.yes-housing.social	-2.092170e-05	-3.687023e-03	6.380070e-03
teleph.yes-housing.rent	-1.637027e-05	-2.884925e-03	4.992110e-03
teleph.yes-housing.freehold	-2.107224e-05	-3.713551e-03	6.425975e-03
teleph.no-housing.social	-2.249766e-05	-3.964754e-03	1.096703e-02
teleph.no-housing.rent	-1.876481e-05	-3.306914e-03	9.147355e-03
teleph.no-housing.freehold	-2.260113e-05	-3.982988e-03	1.101747e-02

Table 7: Marginal effects in different groups at given values of metric covariates

B Additional examples

Example B1 For models with many groups the GUI-panel grows beyond the screen. The only way to select or deselect the nonvisible groups is to preselect these. In this example only the groups which occur more than 10 times in the data are choosen to be displayed.

```
model.cd.manygroups <- glm(credit ~ amount + I(amount^2) + age
    + duration*teleph + housing + intuse, family=binomial, data=creditdata)

factor.combs       <- factor.combinations(creditdata[,c("teleph","housing","intuse")])
logic.index.groups <- factor.combs$counts > 10
index.groups       <- seq(along=factor.combs$counts)[logic.index.groups]

glm.interactive(model.cd.manygroups,
    preselect.var    = "amount",
    preselect.groups = index.groups)
```

Example B2 A customized rug plot with transparent color is added by means of the low-level plotting command `segments()` and the result is saved to the current working directory.

```
glm.interactive(model.2.fac,
    preselect.var  = "amount",
    preselect.type = "response",
    rug.ticksize   = 0)
segments(creditdata$amount, par("usr")[3], creditdata$amount,
    par("fig")[3], col = rgb(0,0,0,0.2))
savePlot("credidefault-customrug", "pdf")
```

C Version history

Version	Date published	Changes
0.2-1	26.08.2014	argument `preselect.groups` added argument `select.all.groups.begin removed` problem with flickering plot solved no restrictions for the names of factors/factor levels any more
0.1-3	18.08.2014	first published version