'plgraphics': A user-oriented collection of graphical R-functions based on the 'pl' concept

Werner A. Stahel, ETH Zurich

May 17, 2026

Abstract

The package, `plgraphics`, collects enhanced versions of basic plotting functions. It is based on a paradigm between the basic R graphics elements and the more computer science oriented ggplot concepts. The intention is to furnish user-oriented functions that allow efficient production of useful graphics.

Unique features include:

- a simple consistent way to specify plotting symbols, colors, sizes, and many more elements to tailor plots, including properties of variables such as labels, axis ticks, line types and more, and to store all these elements for later use.
- a way to show data including outliers in an informative type of scatterplot;
- a novel, more informative alternative to box plots;
- enhanced graphical regression diagnostics for all kinds of models;
- a versatile function to label time axes nicely that automatically adjusts to all ranges of time, from seconds to years;
- flexible versions of plot matrices (" `pairs` ") and conditional plot matrices (" `coplot` ").

Contents

1 Introduction

The plotting functionality is the historical origin of the R package. It has been introduced half a century ago and has grown for a while. For the sake of upward compatibility, it has been essentially unchanged for several decades.

New graphical concepts, adjusted to the development of graphical devices and computer science ideas have been implemented in new packages, notably `ggplot` ...

The intention of the package `plgraphics` is to implement some functions that provide for efficient production of simple to quite sophisticated plots, but are still based on the core R functionality. They have been developed over a long time with a focus on allowing for readily interpretable graphical diagnostics for regression model development.

The general idea is that it should be easy to produce standard plots by a simple call like `plot(x,y)` or `plot(y~x, data=dd)` , as well as enhancing the plot by adding an argument like `smooth=TRUE` to ask for a smoother or specifying a column in the dataset that drives the color or yields labels to mark the points to be shown. Furthermore, scatterplots should remain useful if there are outliers or one of the two variables is a grouping factor instead of a quantitative variable. This basic function for plotting `y` on `x` is called `plyx` . It includes plots of several `y` -variables—as does the standard function `matplot` —and produces separate plots for multiple `x` -variables (Section **??**).

When studying several variables, a popular standard function is `pairs` . This package features `plmatrix` , extends the functionality of `pairs` . Similarly, `plcond` corresponds to the standard function `coplot` producing an array of scatterplots of `y` on `x` for the combination of different ranges of two conditioning variables.

Graphical methods are essential as diagnostics for failures of assumptions in regression models. Plots that have a high potential for showing such flaws are furnished by `plregr` which makes heavy use of the basic function `plyx` .

Asking that a basic function provides many variations under the control of the user means that a large list of arguments must be available. Some of these variations depend on the taste of the user. They can be specified in a kind of "style list," analogous to `options` and `par` , which is called `ploptions` . Variables may call for specific styles, and points in scatterplots may be made distinguishable by specific identifying symbols throughout a study. [Plgraphics] stores meta-data along with data.frames that avoids the need to repeatedly specify such graphical elements when producing multiple displays of the same data. When working with time series, suitable specifications of axis labelling often need quite some user input defining tick positions and labels. The package provide a sophisticated function that usually provides a well adapted solution to this problem.

The package also includes enhanced low level graphical functions like `plpoints` , which extends the functionality of `points` . This leads to a short basic scatterplot function `plyx` that can easily be

modified by the user.

This document presents the main features of the package `plgraphics` and explains the concepts behind them.

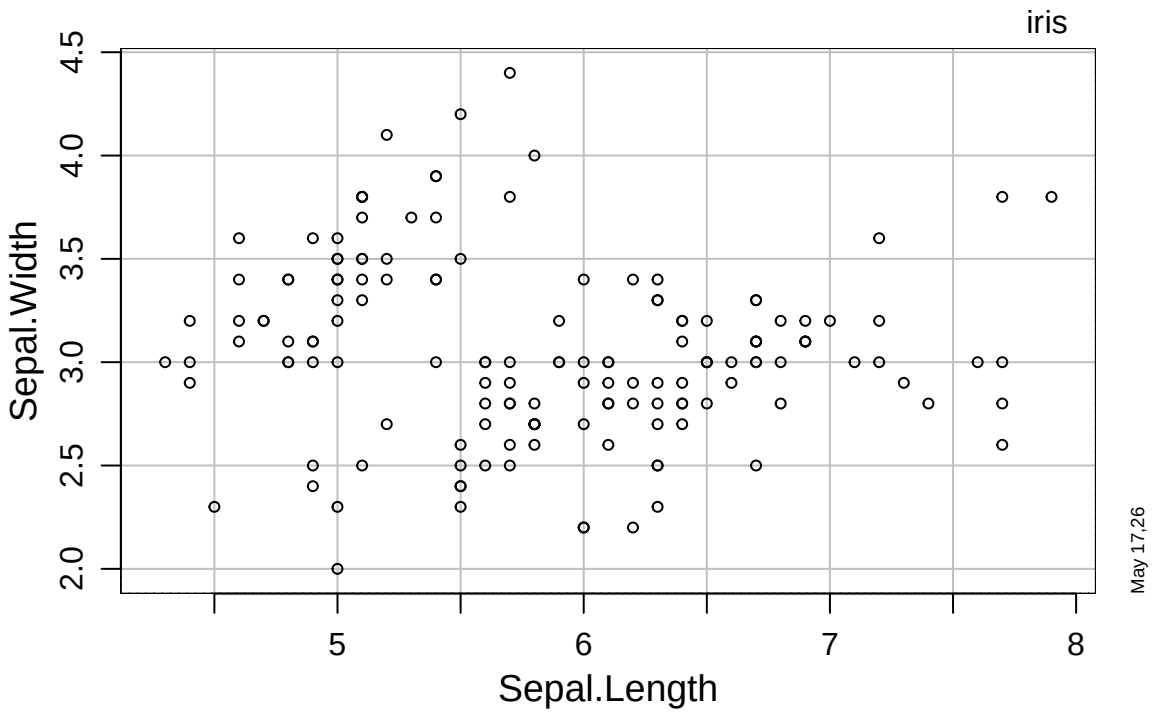
The package is available from `R-forge`, e.g. by calling
`install.packages("plgraphics", repos="http://r-forge.r-project.org")`.

2 Scatterplots

2.1 The basic scatterplot

A basic scatterplot is generated by calling `plyx` (plot y on x),

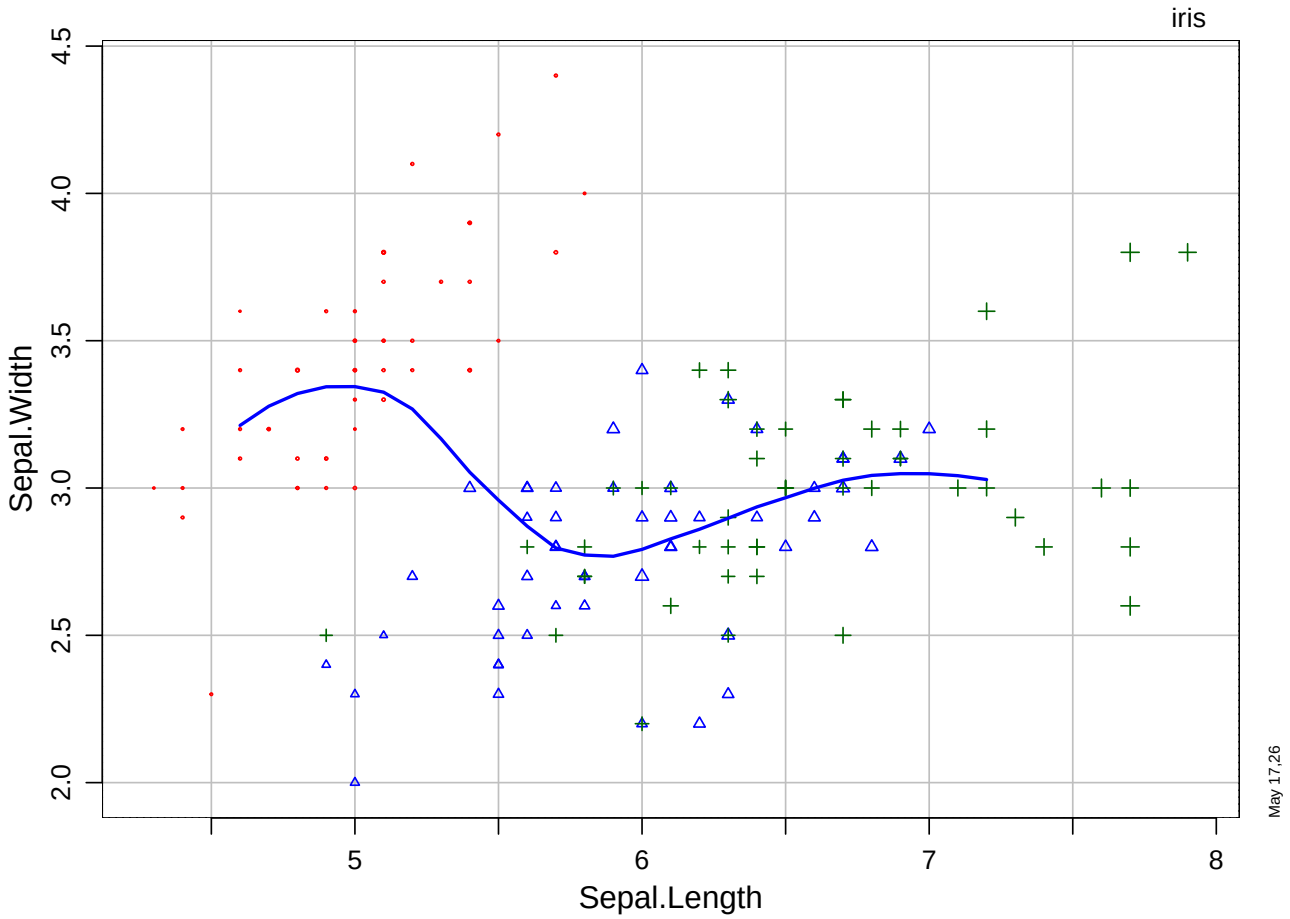
```
plyx(Sepal.Width~Sepal.Length, data=iris, smooth=FALSE)
```



Clearly, this strongly resembles the result of simply calling `plot`, except for the thin gridlines and some documentation added by default: The name of the dataset is shown as a (sub-) title, and there is a small text in the lower right corner that shows the date. Without `smooth=FALSE`, a smooth line is added, see below.

More arguments allow to specify many aspects of the plot.

```
plyx(Sepal.Width~Sepal.Length, data=iris,
     psize=Petal.Length^2, pcol=Species, pch=Species)
```



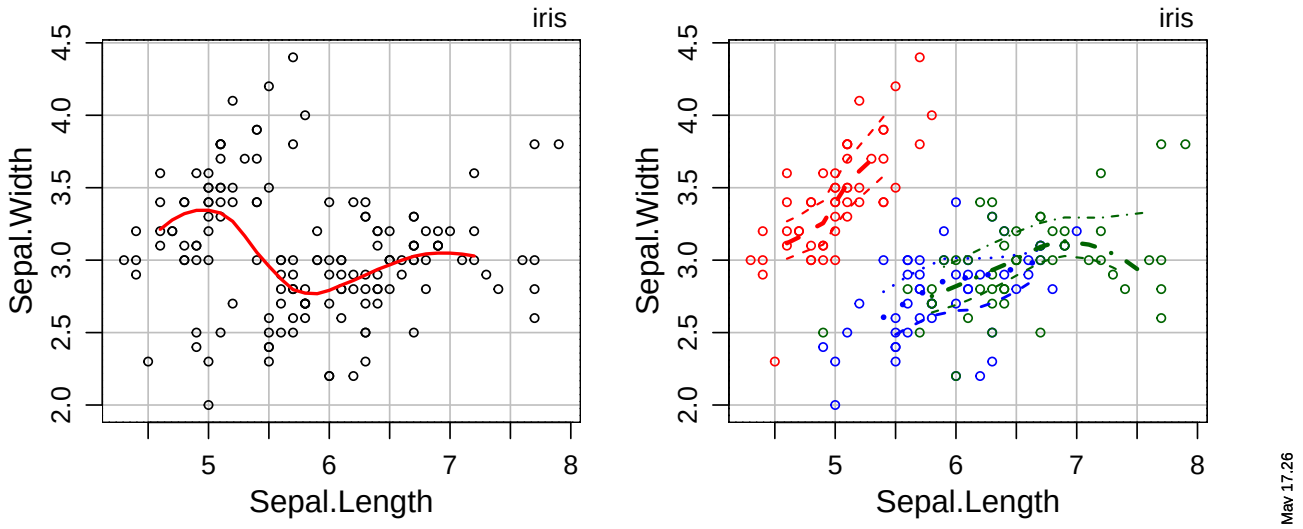
The argument `psize` sets the size of the plotting symbols such that their area is proportional to its value. The median size adjusts to the number of observations. `pcol` specifies the colors of the symbols.

See 5.2 for details.

Smooth. A smooth line fitting the data in the plot is produced if `smooth=TRUE`, which is the default value. The line type, width and color are modified by respective arguments.

Smooths can also be fitted groupwise by specifying `smooth.group`.

```
plmframes(1,2)
plyx(Sepal.Width~Sepal.Length, data=iris, smooth.col="red")

plyx(Sepal.Width~Sepal.Length, data=iris, smooth.group=Species,
     pcol=Species)
```

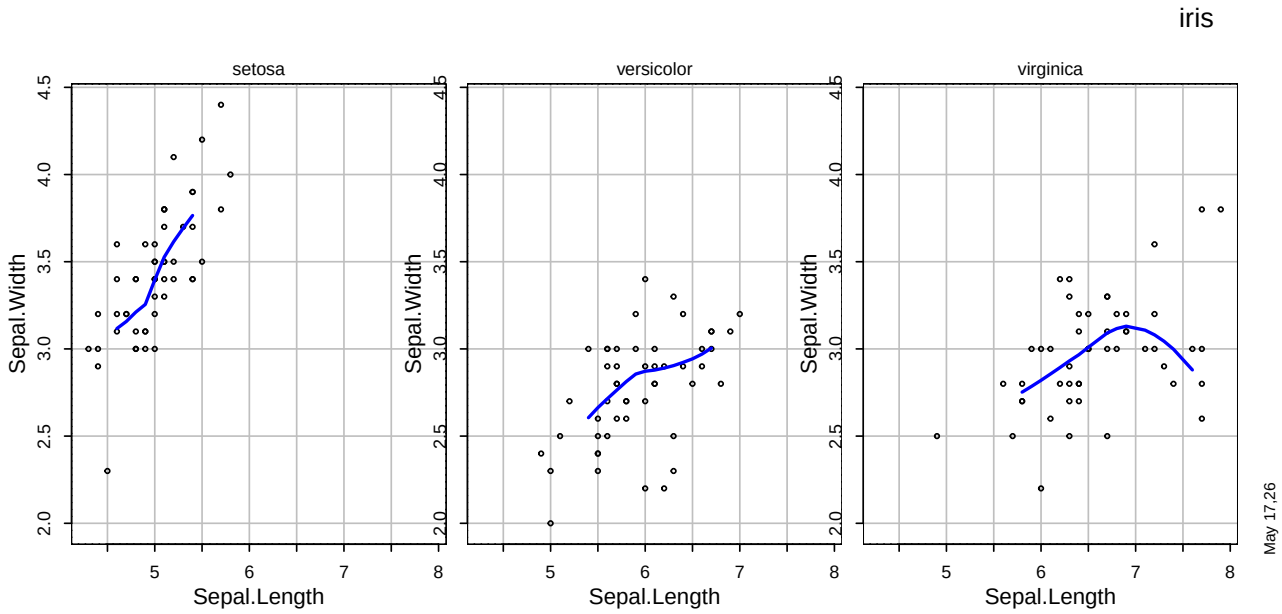


Setting `pcol=Species` allowed the color of plotting symbols and smooth lines to be the same. The call to `plmframes` splits the screen essentially like `par(mfrow=c(1,2))` .

Groups. If the argument `by` is specified, separate plots will be generated for the different groups specified by the levels of `by` variable, thereby maintaining the plot ranges. (An alternative way of specifying the 'by' group is a conditional formula, containing a `|` symbol.)

* The argument `mf` asks for multiple figures, saving space between panels. If left unspecified, `plyx` chooses a matrix of panels. Setting `mf=FALSE` avoids this.

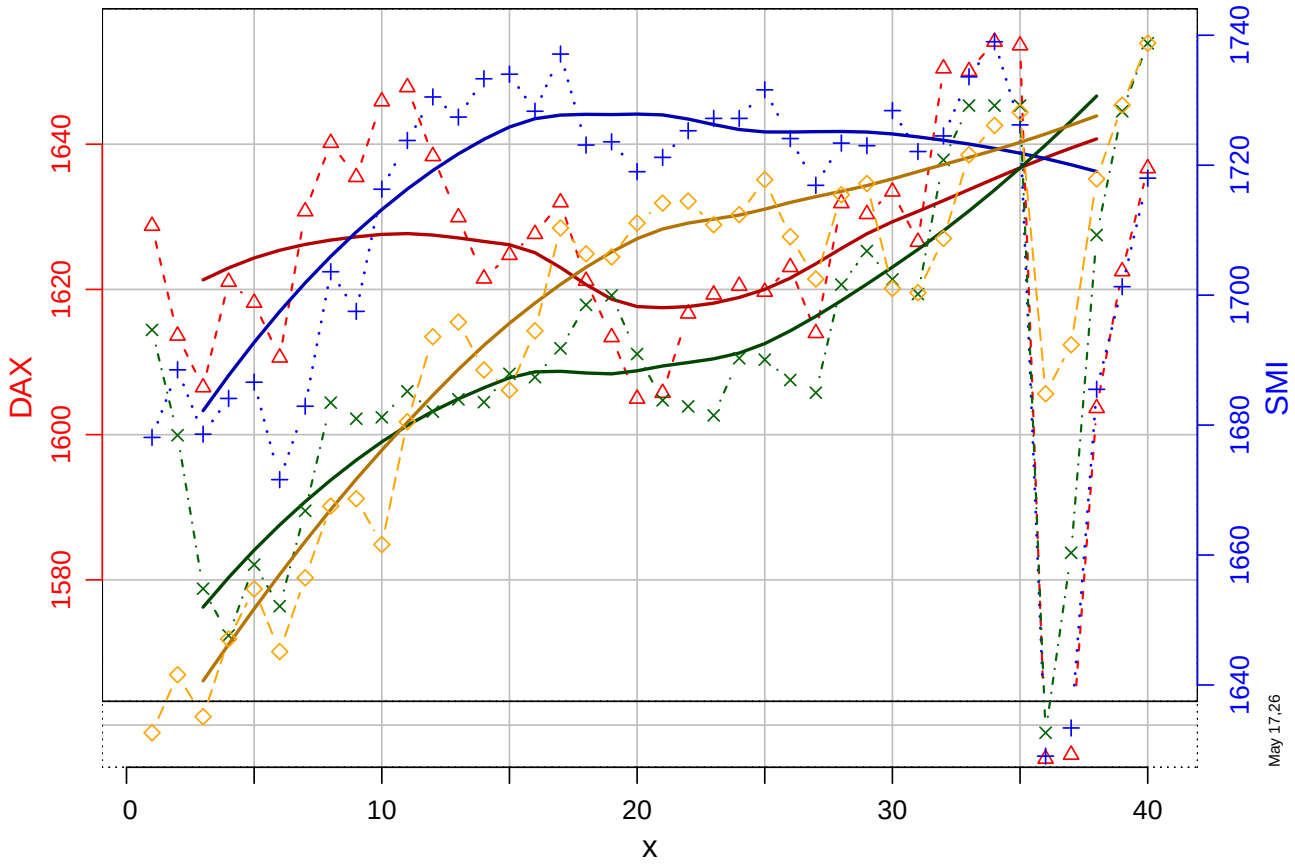
```
plyx(Sepal.Width~Sepal.Length, data=iris, by=Species, mf=c(1,3))
```




```
## or plyx(Sepal.Width~Sepal.Length | Species, data=iris, mf=c(1,3))
```

Inner range of plots. When there are outliers in the data, plots are dominated by them through their effect on the plotting range. This means that the user who would like to see more detail about the "regular" observations needs to generate a new plot, specifying the limits of the plotting range by `xlim` and `ylim`, lines connecting the points, either `type="l"` or `type="b"`. `plyx` will choose different scales for the different variables unless `rescale=FALSE`.

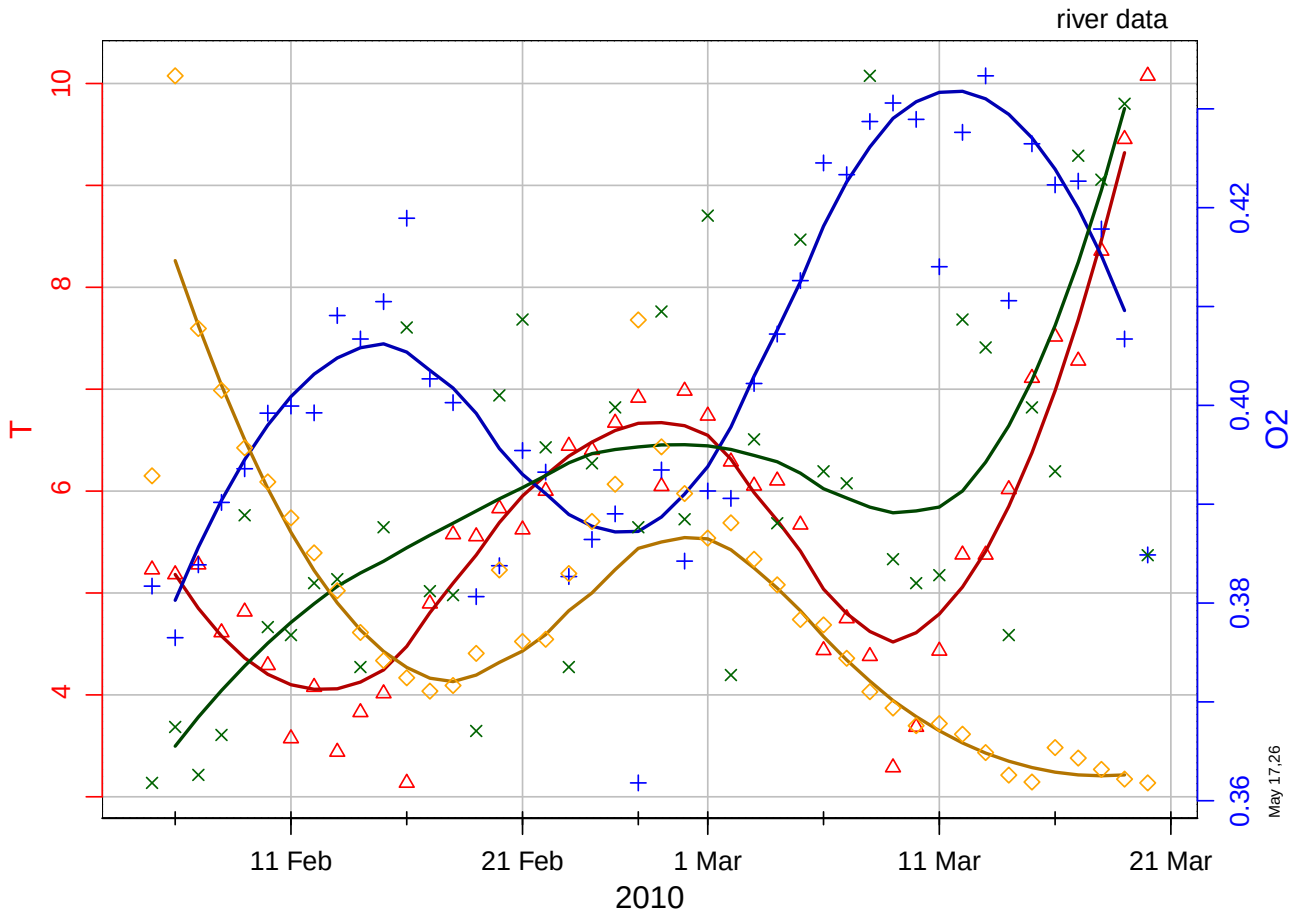
```
plyx(1:40, EuStockMarkets[1:40,], type="b", smooth.xtrim=0.05)
```



The plotting scales on the left and right side apply to the first two of the four variables displayed.

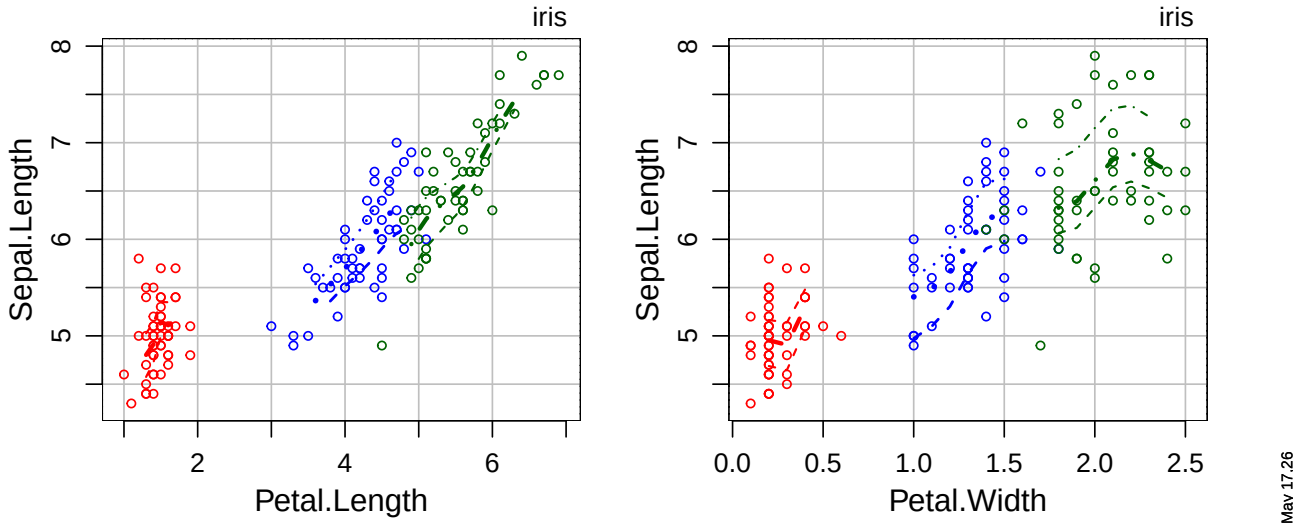
The following plot shows a more elaborate example of a time series plot, see 5.2 and 5.1 for details about the generation of a time axis and the additional arguments of `plyx`, respectively.

```
data(d.river)
plyx(T+O2+ra+Q~date, data=d.river,
     subset=date<as.Date("2010-03-31")&hour==14,
     smooth.par=0.5, smooth.xtrim=0.03, smooth.lty=1, sub="river data")
```



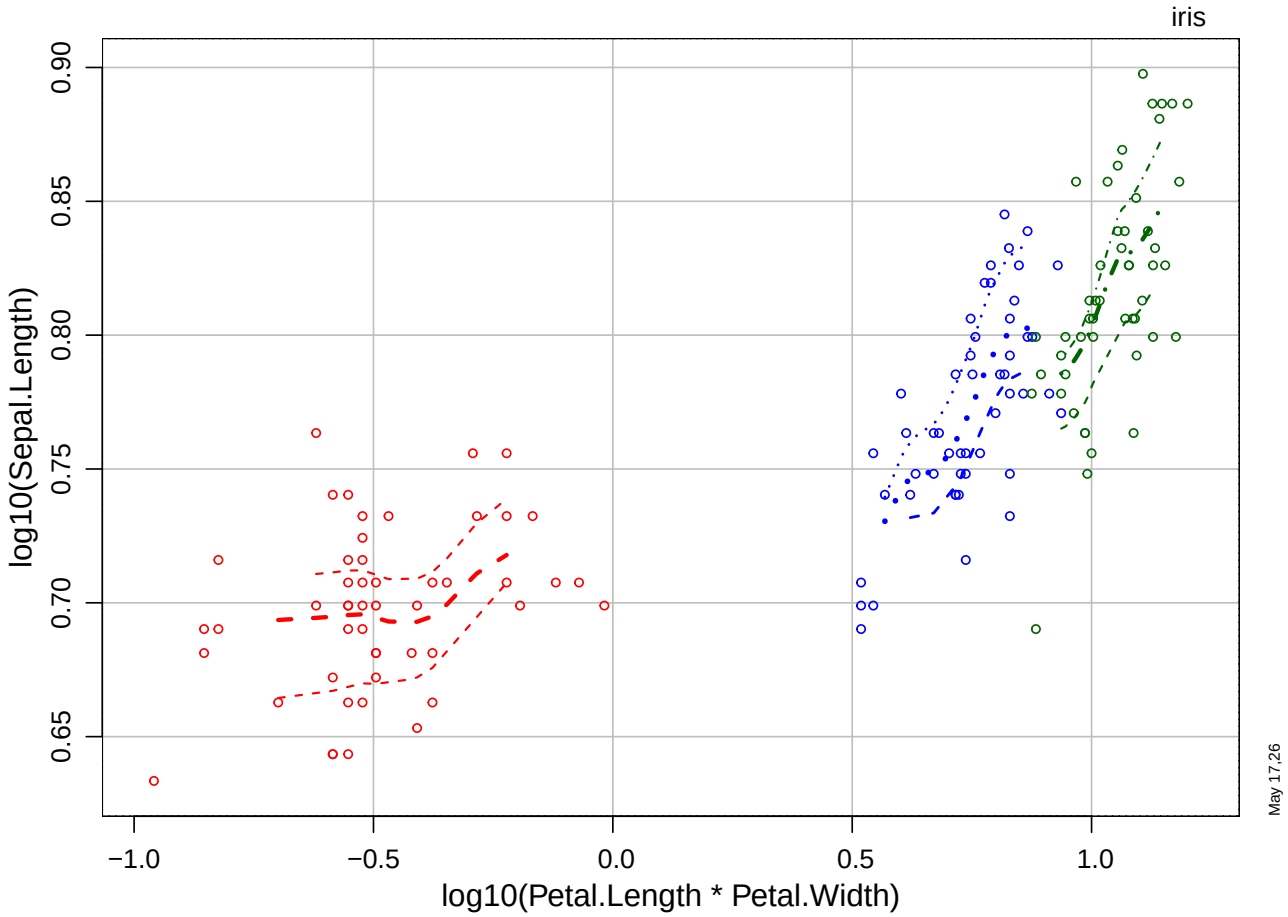
If multiple x variables are given, a separate plot is drawn for each of them.

```
plmframes(1,2, mar=c(3,3,3,1))
plyx(Sepal.Length~Petal.Length+Petal.Width, data=iris,
     smooth.group=Species, pcol=Species)
```



Raw or transformed variables? Simple formulas just include names of variables on both sides of the ∼ symbol, separated by "+" if there are more than one. More advanced formulas consist of terms. (Interaction terms act as two separate terms here.) The user can choose between plotting the terms or the variables that are involved. The most common terms beyond raw variables are transformed variables. If the argument `transformed` is `TRUE`, the terms will be used as plotting variables (horizontal or vertical). Otherwise, the plotting variables are obtained by applying `all.vars` to both sides of the formula.

```
plyx(log10(Sepal.Length) ~ log10(Petal.Length*Petal.Width),
     data=iris, smooth.group=Species, pcol=Species, transformed=TRUE)
```



Setting `transformed = FALSE` produces the same figure as 2.1

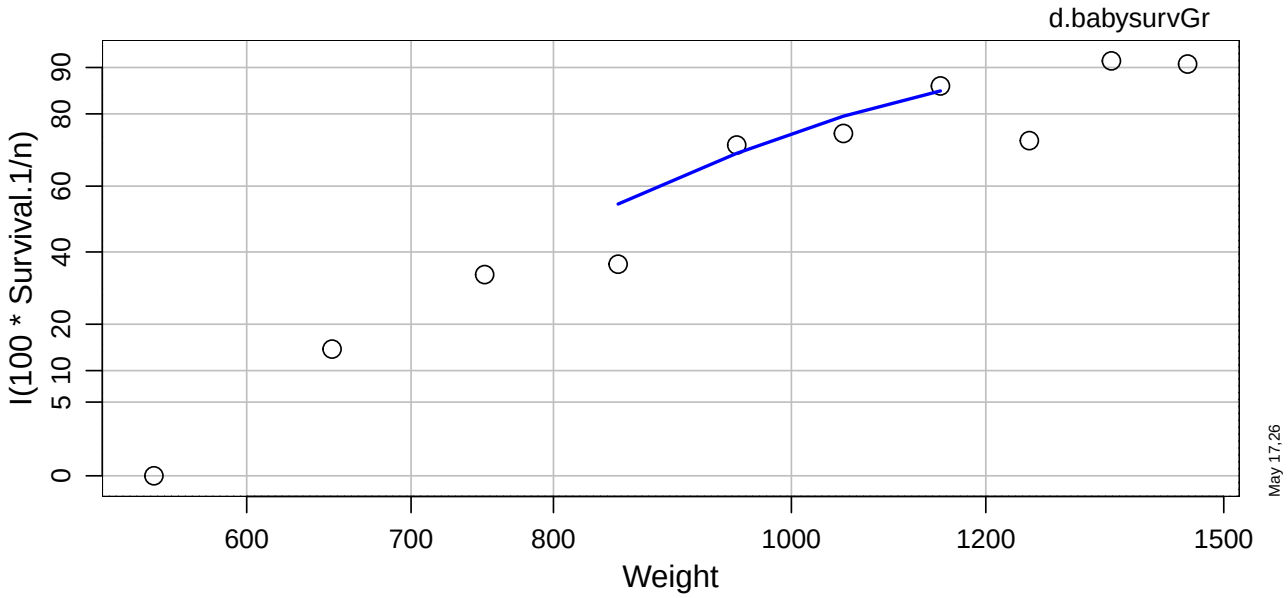
2.2 Nonlinear plotting scales

In basic R graphics, "log scale" may be selected by specifying `log="y"` (or `="x"` or `="xy"`). `plgraphics` allows for any monotone transformation. For example, the arc sine transformation $\text{asinp}(x) = \arcsin\left(\sqrt{x/100}\right)/\arcsin(1)$ is recommended as a "first aid transformation" for percentages. Rather than plotting the transformed variable, the `plgraphics` functions offer the argument `plscale`, which leads to using the function `plscale` and shows the transformed data with tick marks reflecting the original scale (as `log="y"` would do it for basic R).

```
  data(d.babysurvGr)
  showd(d.babysurvGr)


## dim:  10 4
```

```
##       n Survival.0 Survival.1 Weight
## 1    10         10          0    550
## 2    14         12          2    650
## 3    27         18          9    750
## ...
## 5    32          9         23    950
## 6    28          7         21   1050
## 8    26          7         19   1250
## 10   32          3         29   1450
```

```
plyx(I(100*Survival.1/n) ~ Weight, data=d.babysurvGr, plscale=c("log","asinp"))
```

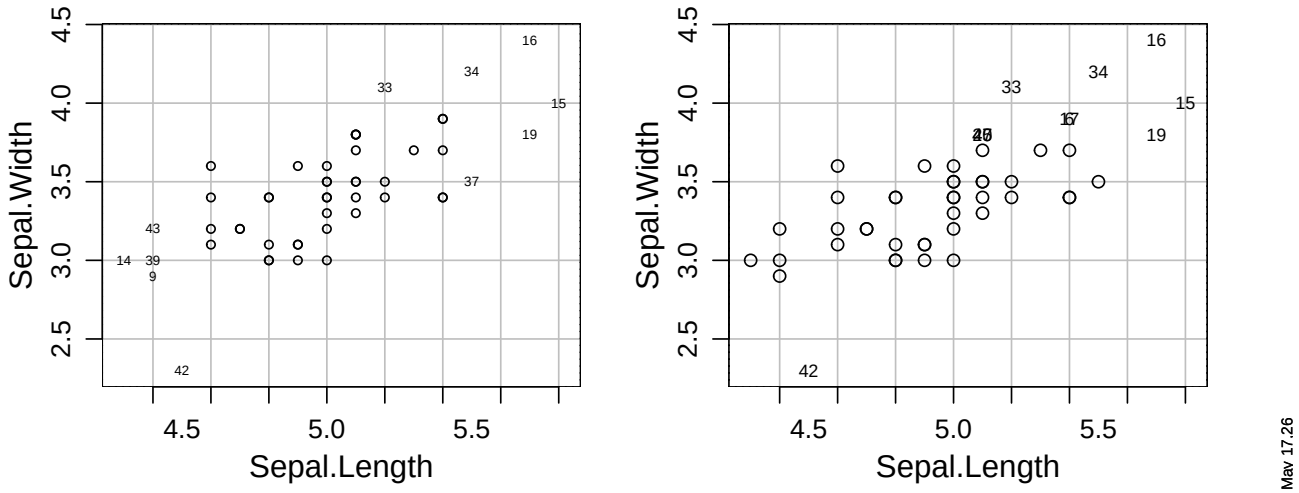


2.3 Marking extreme points

Extreme points are often of interest. They can easily be identified if they are labelled. This is achieved by setting the argument `markextremes`.

```
plmframes(1,2, mar=c(3,3,3,1))

plyx(Sepal.Width~Sepal.Length, data=iris[1:50,], smooth=F,
     markextremes=0.1, csize.pch=0.7)
## different proportions marked in different margins:
```

```
plyx(Sepal.Width~Sepal.Length, data=iris[1:50,], smooth=F,
     markextremes=list(0,c(0.02,0.2)), csize.pch=1)


## Warning in check.ploption(list = largs):  :check.ploption:  argument 'markextremes'
not suitable.  It should
##     be a function or the name of an existing function.  -- or
##   be numeric
##   instead of (str())

## List of 2
##  $ : num 0
##  $ : num [1:2] 0.02 0.2

## Warning in check.ploption(lnam, lopt):  :check.ploption:  argument 'markextremes' not
suitable.  It should
##     be a function or the name of an existing function.  -- or
##   be numeric
##   instead of (str())

## List of 2
##  $ : num 0
##  $ : num [1:2] 0.02 0.2
```



The default value of `markextremes` is 0 for `plyx`. If the argument is `NA`, it depends on the number

of observations: It is $1/(2\sqrt{n})$.

2.4 The Multibox Plot for factors

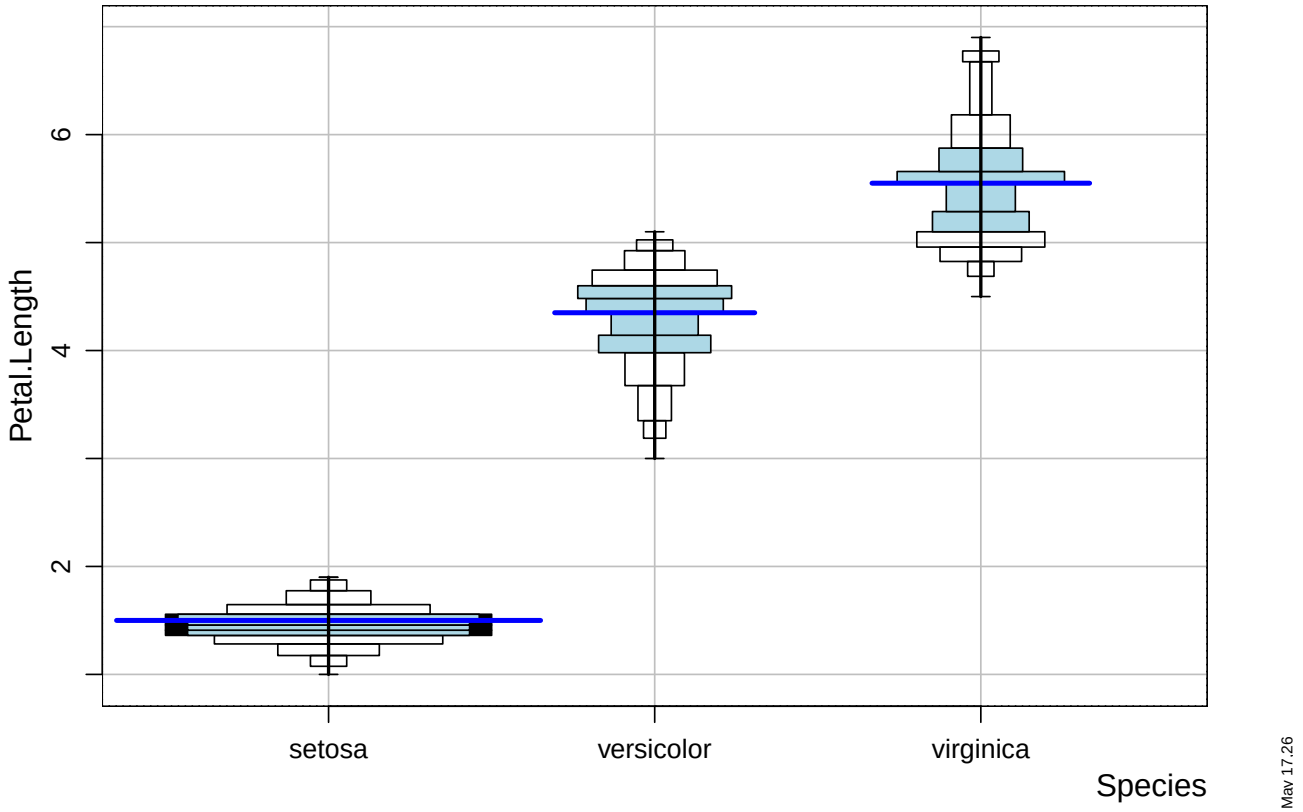
If the x variable is a factor, R's generic plot function for formula objects draws box plots. When there are many observations per group, a box plot constitutes a very rough image of the distribution. The "multibox plot" is a refinement that shows the distribution in more detail.

The idea is to combine the virtues of the box plot, which is based on the quartiles, with those of a histogram that depicts the density of the distribution. Essentially, a histogram is drawn using quantiles as class limits. (Since raw quantiles may be poor characterizations for discrete data, "interpolated quantiles" are used as class limits.)

The number of quantiles – or the number of classes – used should be adapted to the number of observations to be represented. For moderate sample sizes, "octiles" appear suitable.

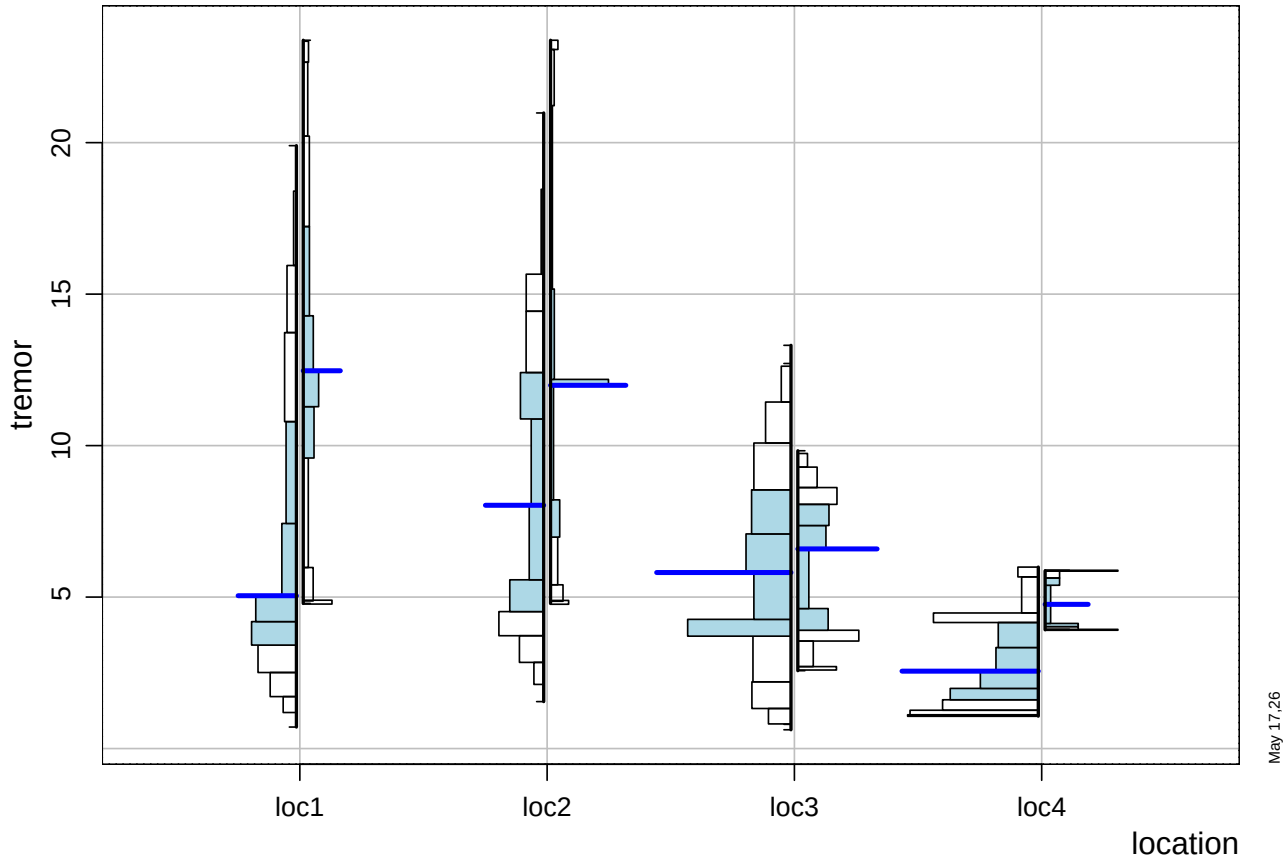
The multibox plot can be called directly.

```
plmframes()  ## reset to just 1 figure per plot
## plyx(Sepal.Width~Species, data=iris)  ## -- or --
plmboxes(Petal.Length~Species, data=iris)
```



The multiboxes resemble a well-known kind of presentation of age distributions. Often, these displays are asymmetric, showing the age distribution of the two genders on the two sides. In much the same way, asymmetric multibox plots can be drawn to visualize the relationship of the continuous variable with any binary variable.

```
data(d.blast)
dd <- plsubset(d.blast, as.numeric(location)<=4)
plmboxes(tremor~location+I(charge>5), data=dd)
```

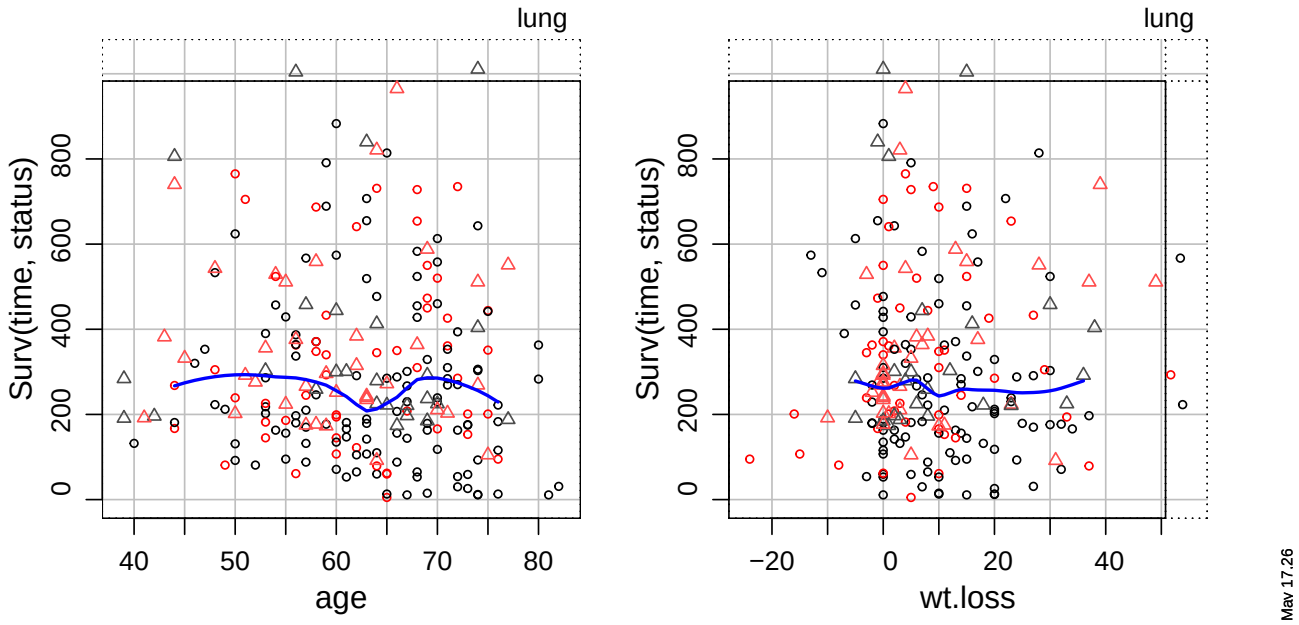


2.5 Censored data

Censored data is shown with special symbols, determined by the ploption `censored.pch`. The default for this option contains 8 elements corresponding to the combined cases of no, right and left censoring of the x and y variables. For the most common case when the y variable is right censored, as is the case in the following example, the censored observations are shown by (upward) triangles.

```
require("survival")


## Loading required package:  survival


## data(lung, package="survival")  ## not needed temporarily, produces erroneous warning


plmframes(1,2,mar=c(3,3,1,1))
```

```
plyx(Surv(time,status) ~ age+wt.loss, data=lung, pcol=sex)
```

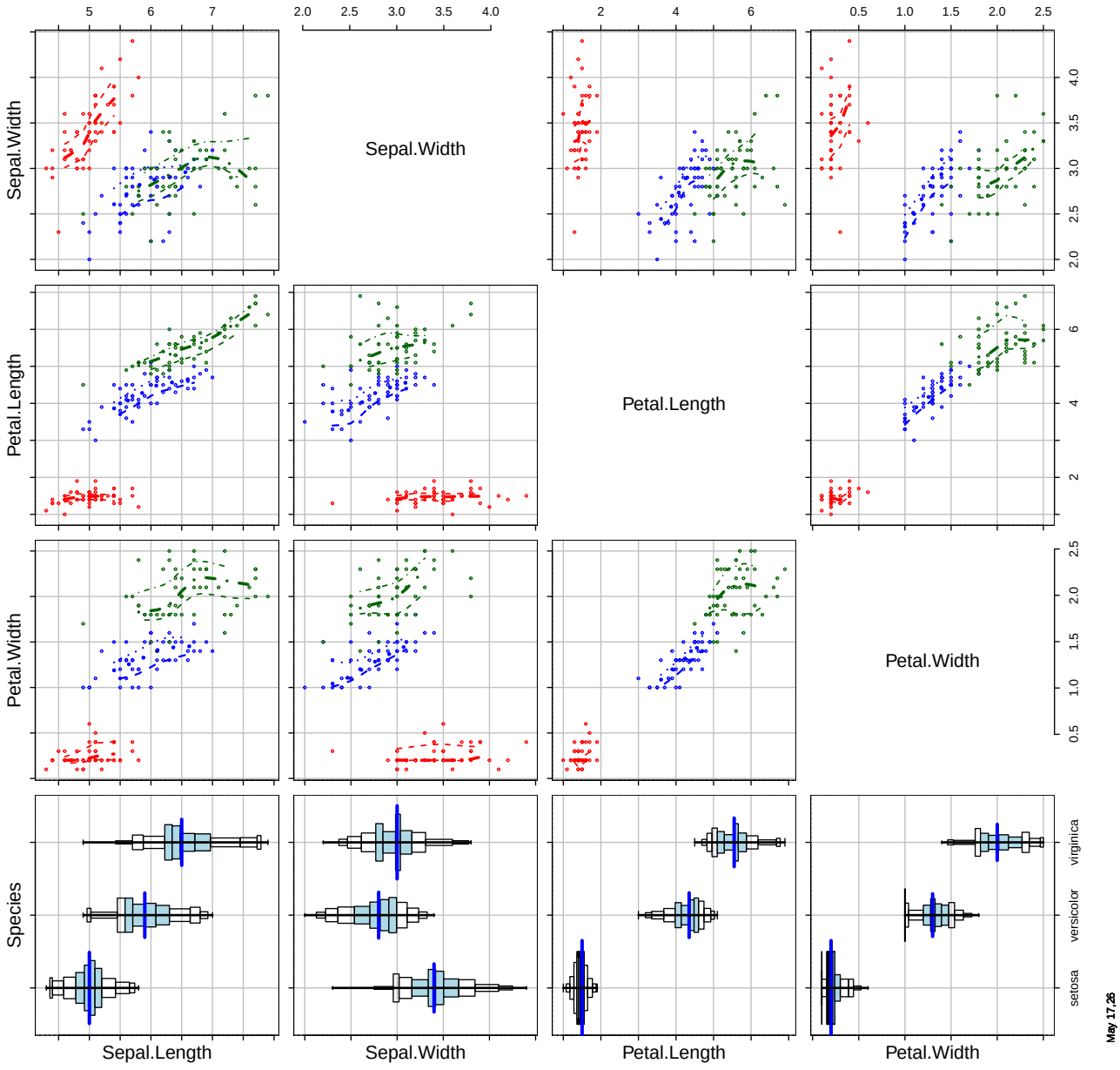


3 Scatterplot matrices

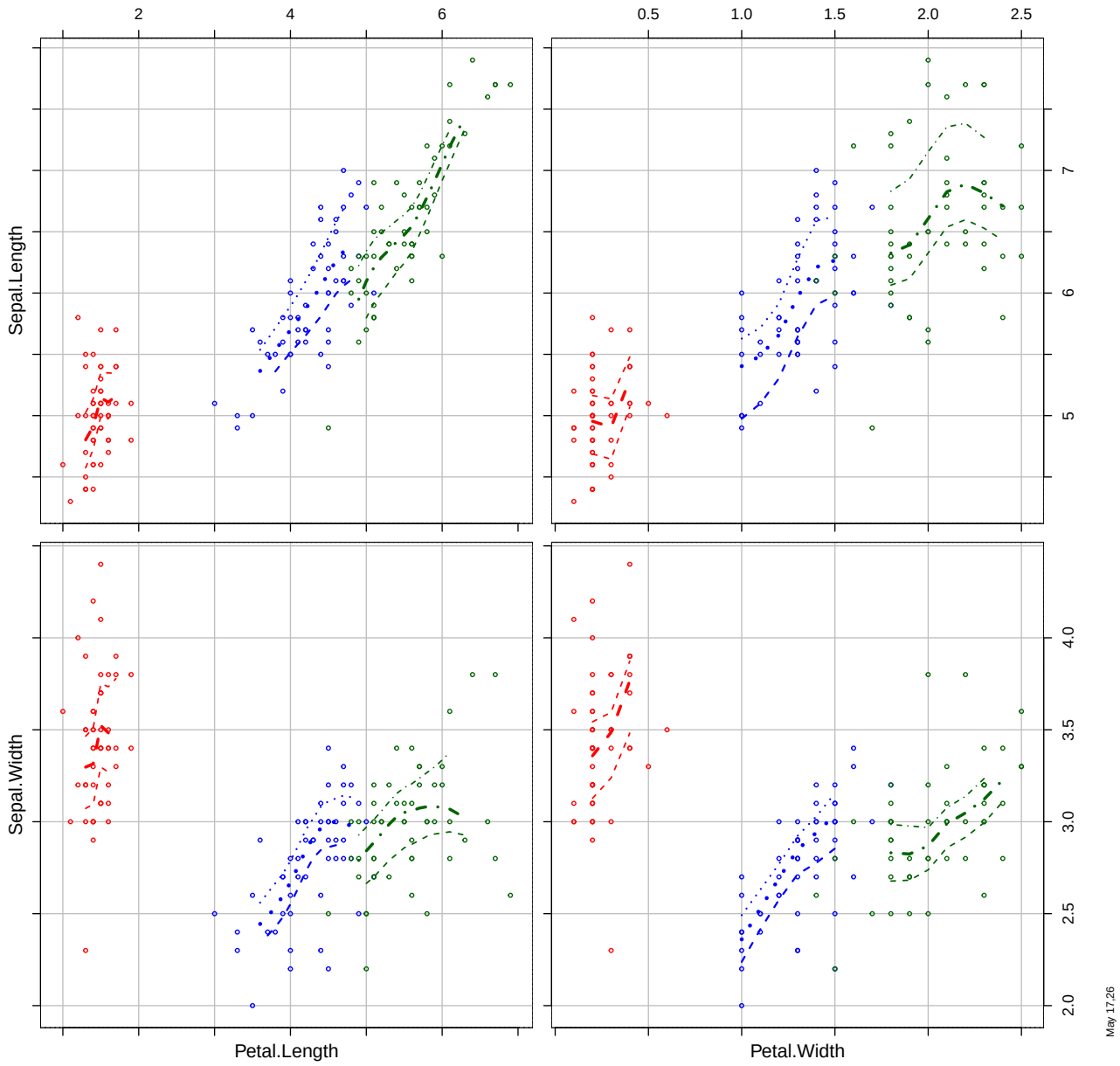
3.1 plmatrix

The common type of scatterplot matrices are produced by the `pairs` function in basic R. The function `plmatrix` does a similar job.

```
plmatrix(iris, smooth.group=Species, pcol=Species)
```




```
plmatrix(~Petal.Length+Petal.Width, ~Sepal.Length+Sepal.Width, data=iris,
         smooth.group=Species, pcol=Species)
```



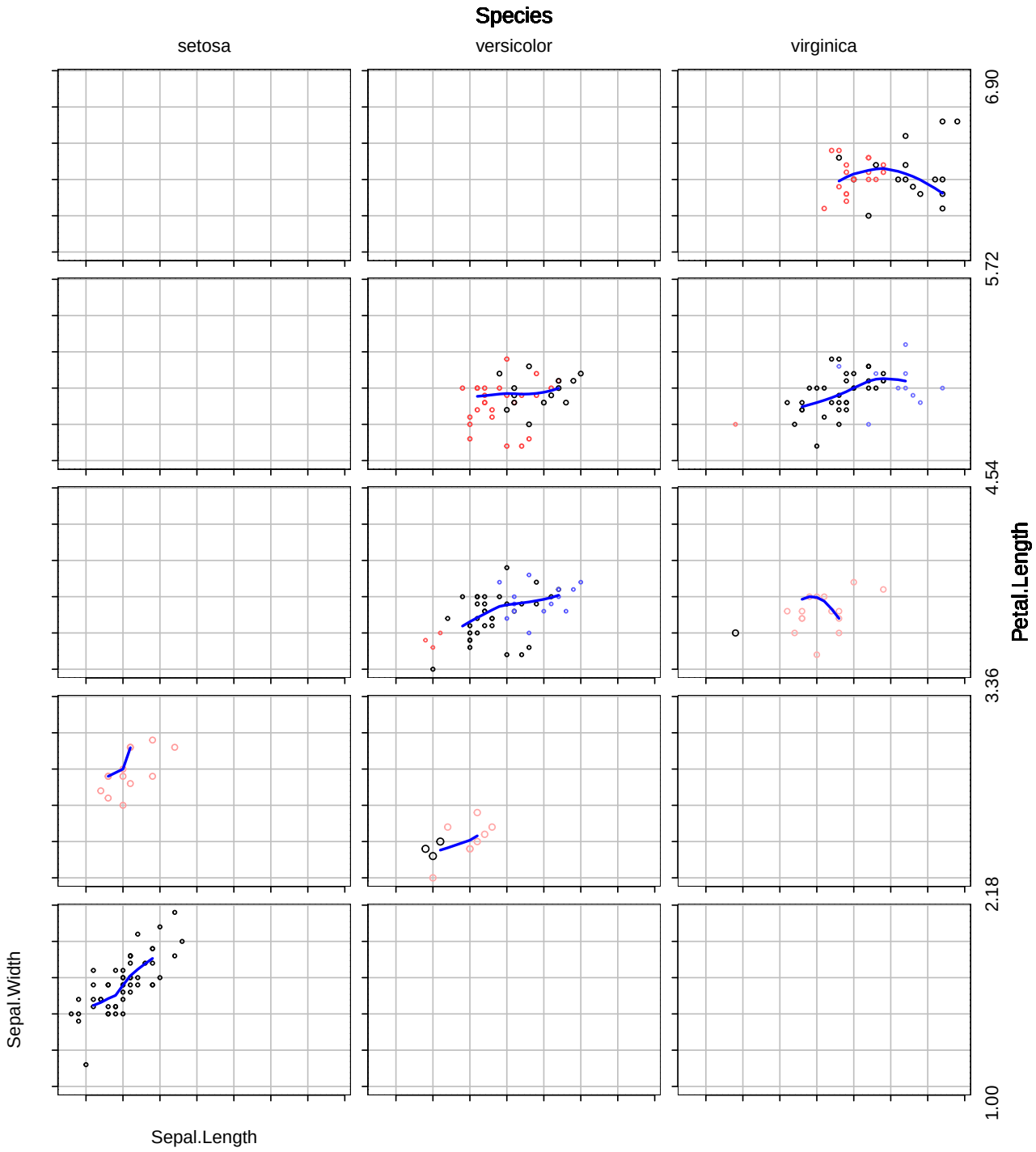
The second call to `plmatrix` shows that different variables can be selected for the horizontal and vertical axes.

If `plmatrix` is called for a large dataset, with more than 8 variables, say, then it splits the scatterplot matrix into parts that appear on separate pages in order to avoid that the panels become too small. The number of panels to be shown in either direction can be set by arguments `nrow` and `ncol`. Otherwise, the function will determine suitable numbers if the total number of panels exceeds the threshold set in `ploptions("mfgtotal")`. The default is 30.

3.2 Conditional plots

A second handsome function in basic graphics that produces a "matrix" of scatterplots is the `coplot`. Each panel is the scatterplot of a pair `x` and `y` of variables, where only a subset of points is shown, and the subset is defined by the intersection of subsets of two variables, the "conditioning" variables. The panels are arranged in the natural way to form a matrix of scatterplots. The plgraphics version of `coplot` is called `plcond`.

```
plcond(Sepal.Width~Sepal.Length, data=iris, condvar=~Species+Petal.Length)
```




```
## or  plcond(Sepal.Width~Sepal.Length | Species+Petal.Length, data=iris)
```

The conditioning variables may be factors or numerical variables. In the latter case, the (robust) range of the variable is partitioned in a number of equally long intervals. Each panel then shows the points

in the interval in their ordinary appearance, and, in addition, some points of the neighboring intervals, with reduced size and paled color. The color is changed to predefined colors indicating whether the point's conditioning variable(s) is (are) below or above the interval. All these features are controlled by pl options.

In the example, the panel for species setosa and `Petal.Length` between `2.18` and `3.36` only contains points from the neighboring interval below, and they are all colored in varying shades of blue, the paling depending on their distance from the lower limit `2.18`. Similarly, for species virginica, the middle interval contains only one point, and the panel shows several points that properly belong to the interval above, in shades of purple.

If only one variable is used for conditioning, the function can also be used. It is then an alternative to using `plyx` with a `by` grouping variable.

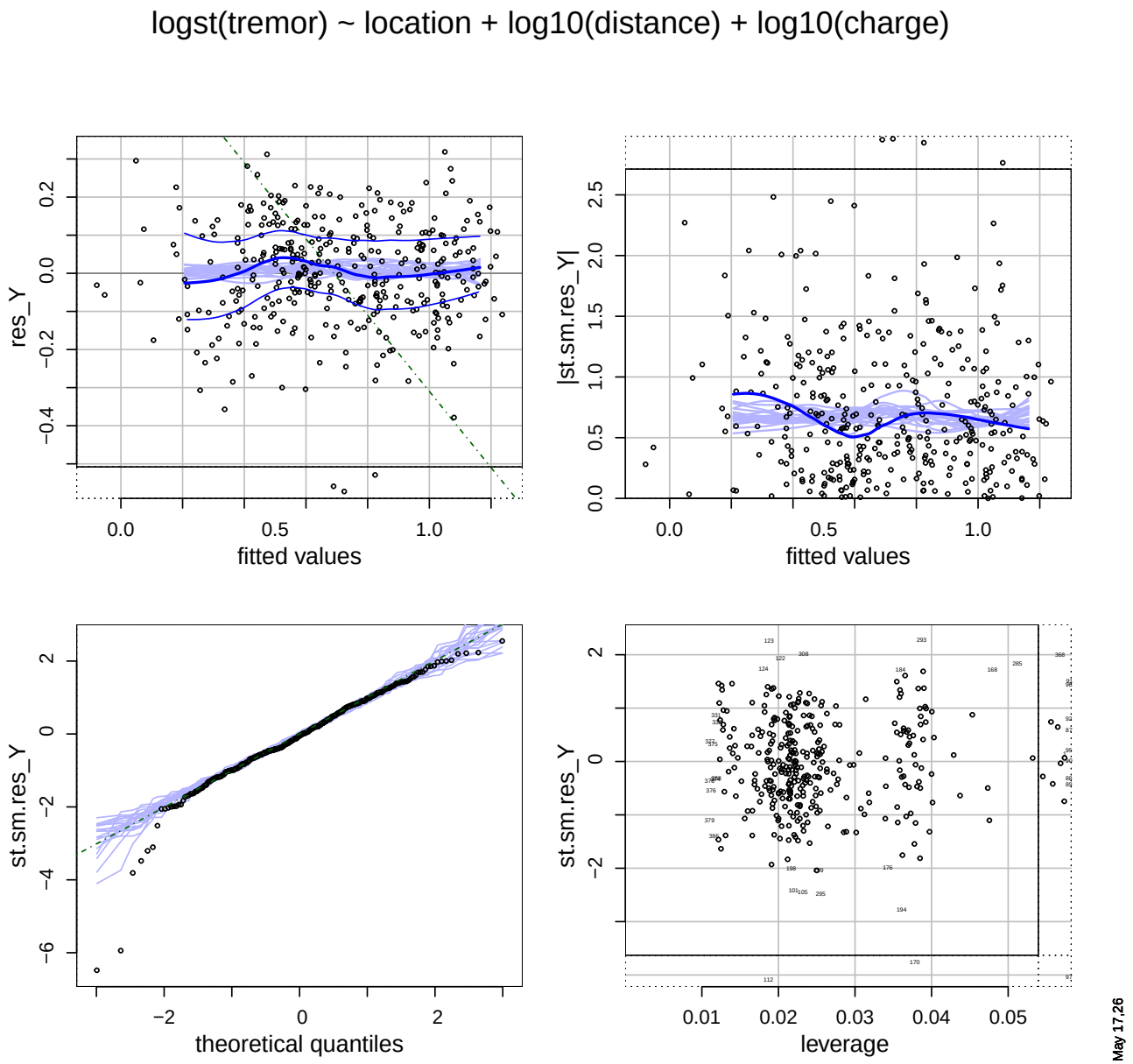
4 Regression diagnostic plots

Graphical regression diagnostics are the essential tools for developing adequate models in many statistical problems. The primary purpose of developing `plgraphics` has been to improve regression diagnostic plots. The features are obtained by using `plregr`.

4.1 The basic diagnostic plots

When R objects obtained from fitting a model are fed into R's `plot` function, some fundamental diagnostic plots appear. The figure shows the versions of these displays obtained by `plregr` in the case of an ordinary regression.

```
data(d.blast)
r.blast <-
  lm(logst(tremor)~location+log10(distance)+log10(charge), data=d.blast)
plregr(r.blast, xvar=FALSE)
```



Before we describe the plots in some detail, let us first explain principles guiding the design of diagnostics.

- Each diagnostic (plot) should be specific for a well-identified potential deficiency of the model.
- Plots are selectively shown in accordance with the type of model that has been fitted. For example, a normal plot is only shown if the model assumes a normal distribution of the random deviations.
- Most importantly, residuals are plotted against explanatory variables. This can also be achieved in plain R by calling `termplot` for other fit objects, but experience shows that this is often neglected by the users.

- Residuals can be plotted easily against the index (sequence number) of the observation (by setting the argument `plregr(..., sequence=TRUE)` , since this may show trends or correlations of random deviations in time – if the sequence reflects time.
- The plots are augmented by a smooth line fitted to them – which is also available in the classical R functions – and two additional smooth lines. These allow to judge the scale dependence on the variable shown: If their distance varies, so does the scale of the residuals. In addition, if their distance to the central smooth differs, this points to a skewed distribution of the random deviations—that can also be seen in the qq plot. See the detailed description of the "Smooth band" below. In addition, simulated smooths are added in order to assess an informal "significance" of curvature in the smooth of the data. Furthermore, a reference line is given, which helps finding an appropriate modification of the model if significant curvature is found.
- Finally, plotting methods are defined for models for which no useful methods are available in the basic R packages, notably ordered response regression and censored responses.

* The plotselect argument is used to modify the default selection and to indicate if a smooth should be shown. For example, `plregr(rr, plotselect=list(yfit=2, qq=0))` . The plots of residuals against input variables can be suppressed by setting the argument `xvar=FALSE` .

* `plregr` saves space around the panels: it reduces `ploptions("mar")[3]` by 1.5 (but not lower than 0.5). Use `mar=c(...)` with a correspondingly larger `mar[3]` element if desired.

Let us now describe the panels shown above in some detail.

Residuals against fit, Tukey-Anscombe plot. The plot of residuals against fitted values is the most important diagnostic tool to assess the validity of model assumptions. It allows for detecting deviations from linearity, homoscedasticity, symmetry and normality of the random deviations' distribution.

By default, the scatterplot of residuals against fitted values shows the points with the feature of outlier margins and marking of extremes in the residual direction. It adds a smooth line to show deviations from the linearity assumption. Another 19 smooth lines are shown to mimic the variability of this smooth line under the hypothesis that the model is correct.

If there is curvature, one remedy may be to transform the response variable. This can help only if the relation between fitted and response values is monotone – otherwise, quadratic terms are the most straightforward modification of the model. Monotonicity is easy to see if the response is plotted on the vertical axis instead of the residuals. This display can also be requested from `plregr` (by setting `plotselect=c(yfit=3)`).

In order to avoid the necessity to either call `plregr` again or to ask for the plot of response against fitted values routinely, a **reference line** (green dot-dashed line in the Figure) is shown in the Tukey-Anscombe plot. It connects points with equal response values. Since the response can be determined

by adding the two coordinates of the plot – fitted plus residual – lines of equal response values have slope -1. The reference line shown is such a line, the one going through the center of the points in the plot. If the smooth never gets steeper than the reference line, then the suggested relation between fitted and response values is monotone, and a transformation of the response may remove the non-linearity.

One might argue that a plot of the response on the fit is more straightforwardly interpreted than the Tukey-Anscombe plot with the reference line. We think that the Tukey-Anscombe plot can show the deviations from model assumptions more clearly and should therefore be preferred, and that the reference line, after a short learning period, will be easy enough to work with and helps avoiding an additional display. Of course, some users may disagree.

* The smoother used by default to generate the smooth lines in the plot is `loess(..., span=smooth.par, iter=smooth.iter)`, where `smooth.par` and `smooth.iter` are given in `ploptions` and the default `smooth.par` adapts to the number of observations by calling the function `smoothpar`. If the response of the model is a count (binary-binomial, Poisson with a low maximal count, or of class `polr`), the non-robust version is called by setting `iter` to 0 and the `family` argument to `gaussian`. Otherwise, `loess` produces a robust smoother.

* **Smooth band.** As mentioned above, in addition to a smooth line fitted to the points, a "low" and a "high" smooth line are shown (unless suppressed). They are obtained as follows: Residuals are calculated as the difference between the observations and the respective smoothed values $\widehat{s}_i$. Then a smooth is calculated for the square roots of the positive residuals, and the squared fitted values are added to the $\widehat{s}_i$. (The transformation by square roots makes the distribution of the residuals more symmetric.) This defines the "high" smooth line. The construction of the "low" one is analogous. The two lines characterize how something like the low and high quartiles depend on the horizontal axis, and thus allow to examine the scale and skewness of the distribution of the random deviations.

Absolute residuals against fit. As a second diagram (often below the first one), the plot of absolute residuals against fitted values is shown. It is intended to detect heteroscedasticity, more precisely, variances of the random deviations that depend on the expected value of the response.

Note that the "absolute residuals" shown in this plot are not the absolute values of the residuals used in the first plot. They differ in two ways:

- They are standardized to have the same variances. If there are weights of observations (argument `weights` in the call to the fitting function), these are taken as inverse relative variances of the random deviations.
- By default, they are modified in the following way. Note first that the plot should show any dependence of the scale of the random deviations on the model value. If the plot of residuals against fit shows a clear curvature, the residuals do not show only the random deviations but also the bias of the regression function, which should be best approximated by the smooth line in that first plot. Therefore, the residuals from the smooth line are used in the plot of absolute residuals against fit. Additionally, they are standardized using the same factor that is commonly

used for standardizing the ordinary residuals.

* Basic R's plot method for `lm` models (and related ones) shows the square root of the absolute residuals. They are more symmetrically distributed and therefore more suitable for calculating a smooth. `plregr` also uses these transformed values for generating the smooth, but still shows the absolute residuals and the smoothing line in the original scale, since this version is easier to understand.

QQ-plot. The next plot is the normal quantile-quantile plot. It is only shown if the residuals are expected to follow a normal distribution. (QQ-plots for other distributions are lacking.) Note that the residuals used here are NOT the raw or standardized residuals, but instead, they are generated as follows: The "residuals from smooth" as explained for the last plot are used to estimate a "smoothed scale" by obtaining a smooth fit to the square roots of their absolute values against the model fit and squaring the resulting smoothed values. They are then divided by these smooth-scale values in order to make them homoscedastic even in the presence of a scale that depends on the model fit.

Residuals against leverage. The influence of individual observations on the results of fitting the model is measured by the quantities produced by the function `influence`. The most important measures are functions of the residuals and the leverage values, often denoted as h_i, which are proportional to Mahalanobis distances from the center of the design based on the (formal) covariance matrix of the design. Therefore, a plot of residuals against leverages should reveal the overly influential observations.

The leverage plot of `plregr` uses *standardized* residuals in contrast to R's standard leverage plot shown by `plot`. In the case of weighted observations, "de-weighted" leverages,

$$h_i^{(dw)} = h_i/w_i = \underline{x}_i^T(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\underline{x}_i$$

are used, but weights are shown by the symbols' sizes. This version maintains the idea that leverages should be proportional to Mahalanobis distances.

* The plot also shows contour lines of constant Cook's distance, defined as

$$D_i^{(C)} = \frac{r_i^2}{p\widehat{\sigma}^2} \cdot \frac{h_i}{(1-h_i)^2} = \tfrac{1}{p}\, r_i^{*2} \frac{h_i}{1-h_i} \;,$$

where r_i^* denotes the standardized residual. Since the mean of h_i is $1/p$, an observation with this leverage and a standardized residual of 1 has $D^{(C)} = 1/(p-1)$. Contour lines are drawn for $D_i^{(C)} = \pm c^2/(p-1)$, where c is given by `ploptions("cookdistlines")`. Note that this is different from standard R.

In several non-Gaussian models, the estimator can be regarded as a weighted Least Squares estimator with suitable weights. Therefore, the weighted version of the leverage plot is produced for such models.

Weights. If weights are given by the user, they should usually reflect different variances of the random deviations E_i, and the residuals, multiplied by the square root of the weights, should have constant variance. The standardized absolute residuals are therefore plotted against the weights. If the size of these residuals is not constant but depends on the weight, the rule or weighting function should be revised.

If weights are given, they are also used as the sizes of the plotting symbols (circles) in other plots.

The argument `xvar=FALSE` in the statement generating the last plot indicates that by default, `plregr` shows more diagrams: the plots of residuals against input variables.

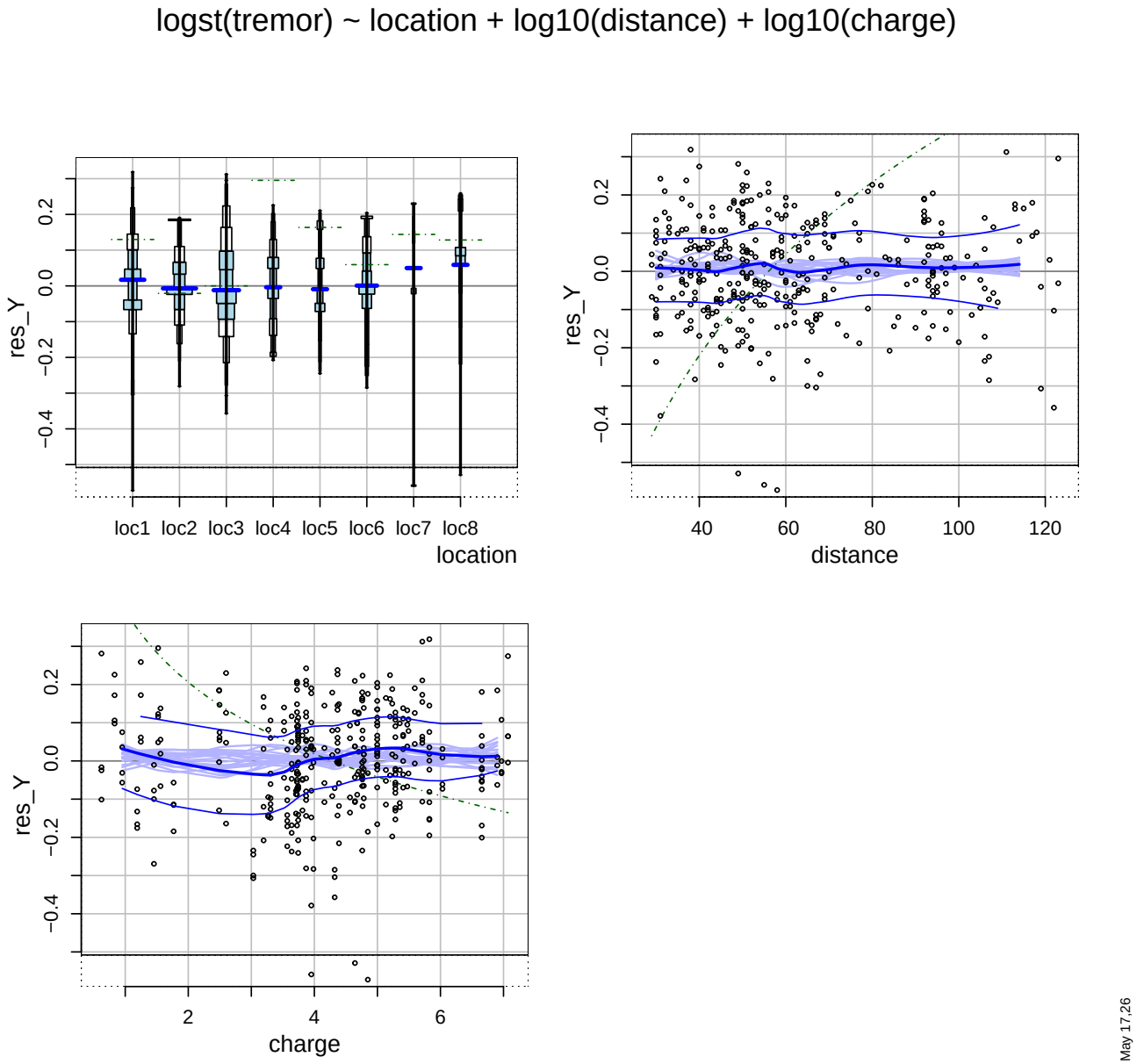
The diagrams shown by `plregr` and their sequence are determined by the argument `plotselect`, see `?plregr` for details.

4.2 Residuals against input variables

Since the "x" variables in a regression model cannot always be interpreted as *explaining* the variability of the response Y, we call them "input" rather than "explanatory" variables here.

The plots of residuals against these variables are important regression diagnostics since they help to find better models. They are often neglected since R's ordinary plot function for models does not show them. `plregr` does, unless `xvar=FALSE` is used as it was above. It does so by calling `plresx`, which can also be done directly.

```
plresx(r.blast)
```



The diagrams show a smooth and simulated smooth lines as described for the paragraph on "Residuals against fit" above. For the variables appearing in the model, they also show a reference line in analogy to that plot. Its precise interpretation is somewhat more complicated, but the intention is again to show a line of constant response values. Since the other variables also influence the response, this can only be achieved keeping all other variables constant.

* More precisely, the reference line connects points for which the sum of the "component effect" $\widehat{\beta}_j x^{(j)}$ plus residual is equal. The function `fitcomp` calculates the coordinates of the reference line. The "other variables" are set to their median if numeric and to their mode if ordinal.

A band for it can be requested by setting `refline=2`. It reflects a confidence band for the regression function value.

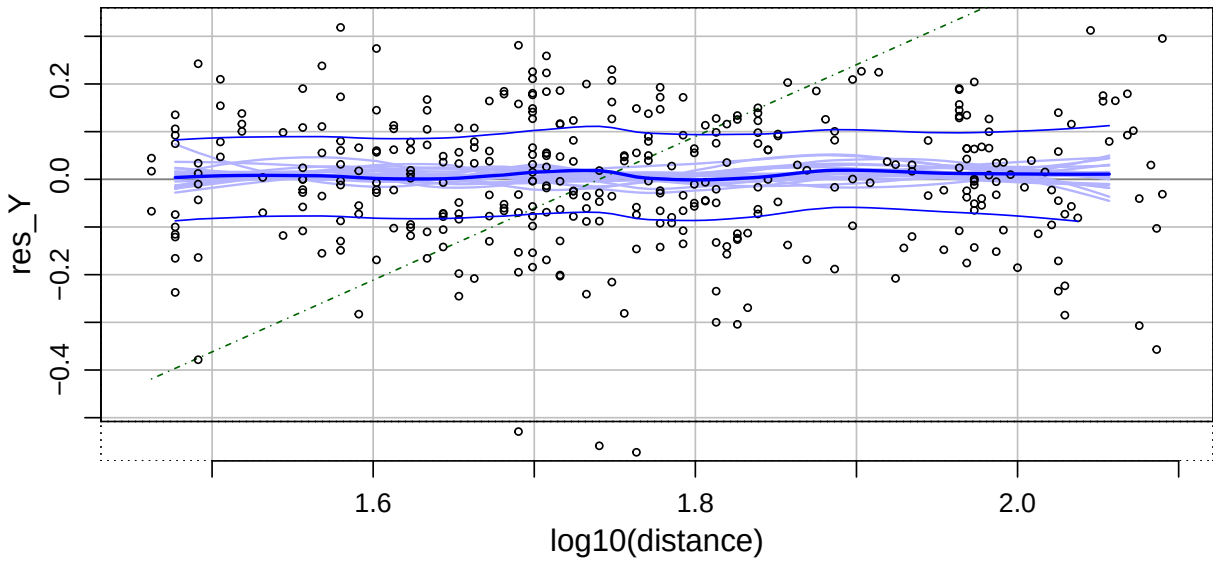
Note that the slope of the reference line is negative if the regressor has a positive effect on the response.

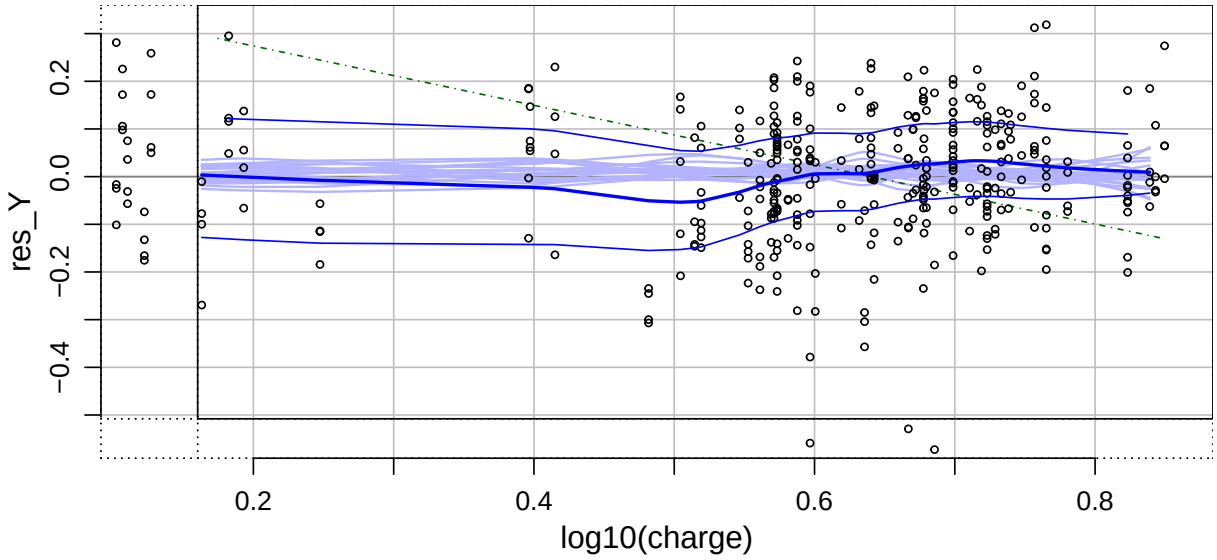
The line is again useful for finding an adequate modification if a curved smooth appears: If the smooth never gets steeper than the reference line, a monotone transformation of the regressor can help. Otherwise, adding its square to the model may help.

Alternatively, the sums "component effect $\widehat{\beta}_j x^{(j)}$ plus residual" may be used as the vertical axis instead of the residuals for plotting. This display is usually called the "partial residual plot" and can be obtained from `plregr`, too. It is related to the plots shown by default in the same way as the response versus fitted plot is to the Tukey-Anscombe plot.

The input variables are often transformed before they are used in the linear predictor, and the main purpose of showing a plot of residuals against them is to possibly find a (more) adequate transformation. For those that have been transformed already, the adequate transformation may be more easily guessed if the untransformed version is used in the plot. The transformed variables can be called for by setting `transformed=TRUE`.

```
plresx(r.blast, transformed=TRUE, refline=2, xvar=~.-location, mf=c(1,2))
```





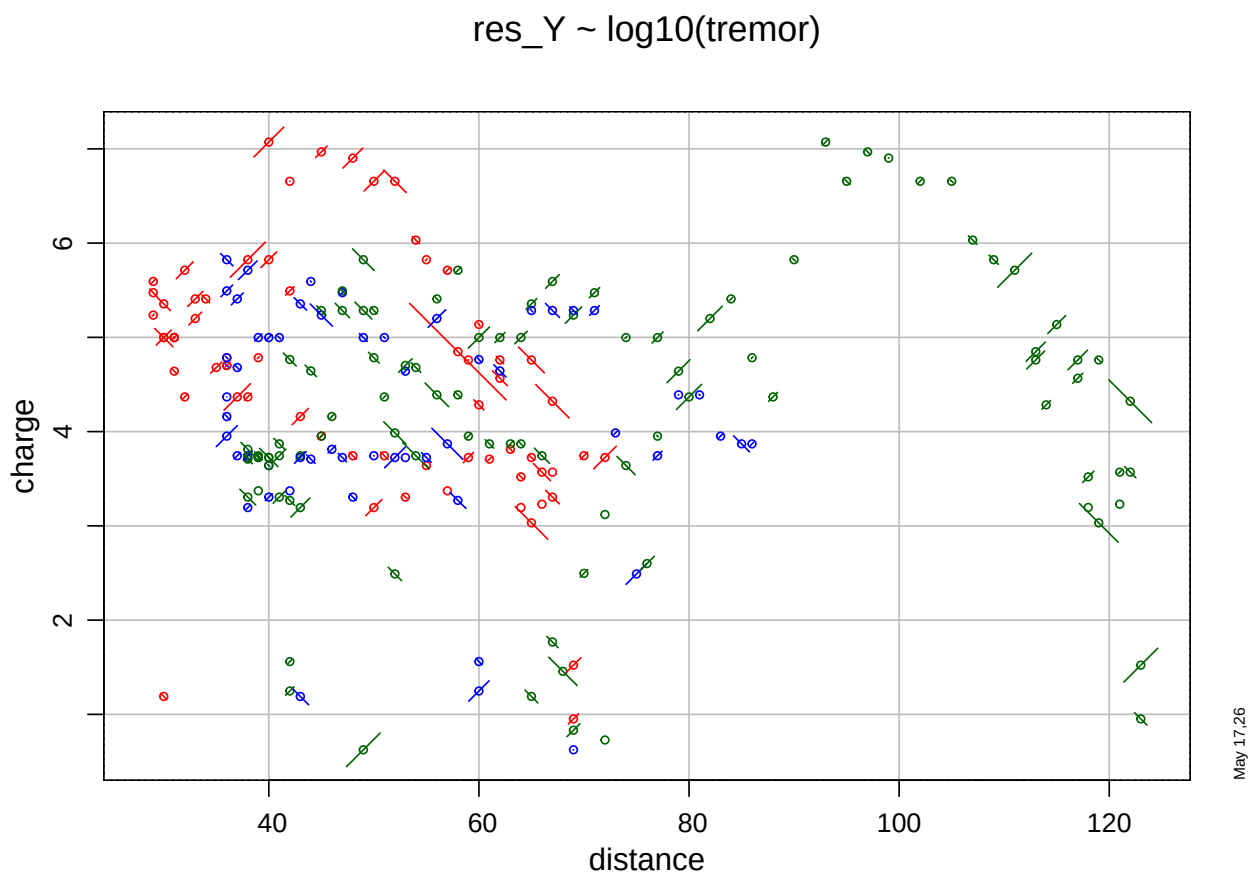
* The raw input variables are those appearing in the formula, as delivered by `all.vars(formula)`. The transformed input variables are those appearing in the terms of the formula, as delivered by `rownames(attr(terms(formula[1:2]), "factors"))`.

* If the fit object contains a variable `weight`, then residuals will be plotted against these weights by default, unless it is the result of `glm`.

The argument `sequence=TRUE` will produce a plot of residuals against the sequence of the observations in the dataset. This diagram may show serial correlations or groupings if the sequence reflects such patterns.

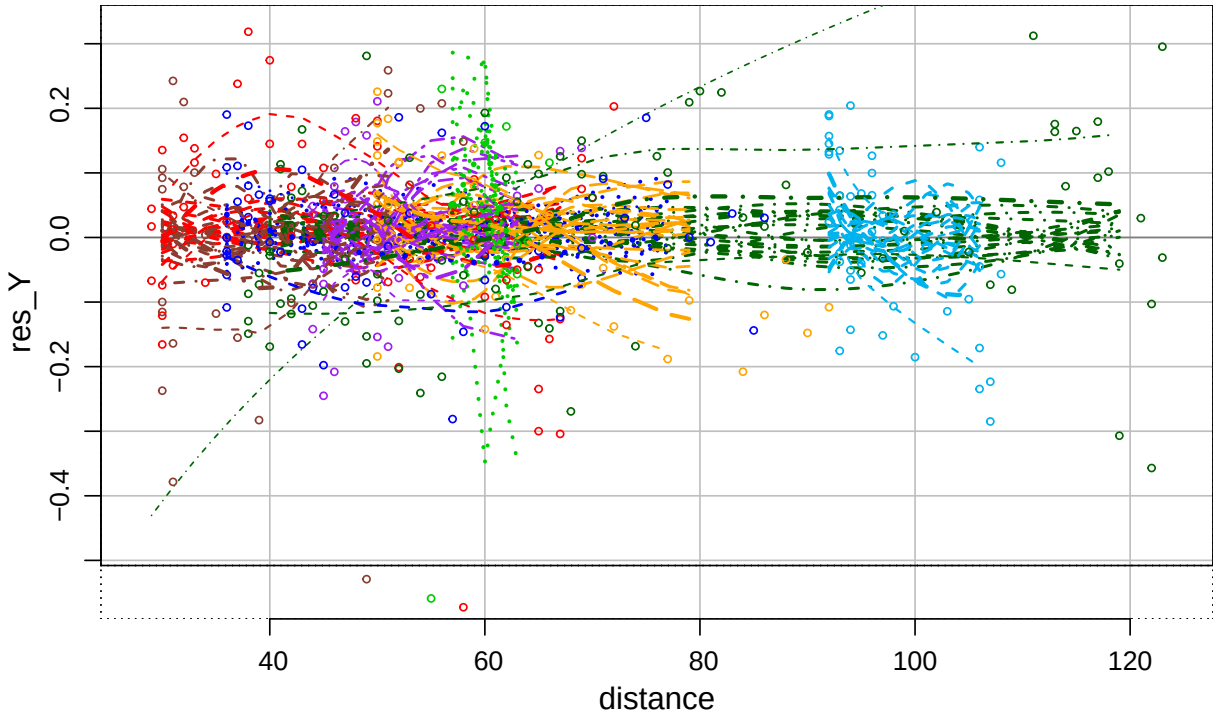
Plotting residuals against two regressors. A missing interaction term between `x1` and `x2` may be found when examining a plot or residuals against these two variables. This is achieved by the function `plrex2x`. It produces a scatterplot of `x1` against `x2` and represents the residuals as line segments with positive or negative slope, according to their sign. The absolute value determines the length of the segment.

```
data(d.blast)
dd <- plsubset(d.blast, as.numeric(location)%in%1:3)
rsubs <- lm(log10(tremor)~log10(distance)+log10(charge)+location, data=dd)
plres2x(~ log10(distance) + log10(charge), reg=rsubs, pcol=location)
```



An interaction between a factor and a continuous variable may be examined more effectively by plotting the residuals against the continuous variable and asking for smooths for each level of the factor.

```
plresx(r.blast, xvar=~ log10(distance), pcol=location, smooth.group=location)

## ..gensmooth:  too few observations for a 'high' smooth for group  7
## ..gensmooth:  too few observations for a 'low' smooth for group  7
```



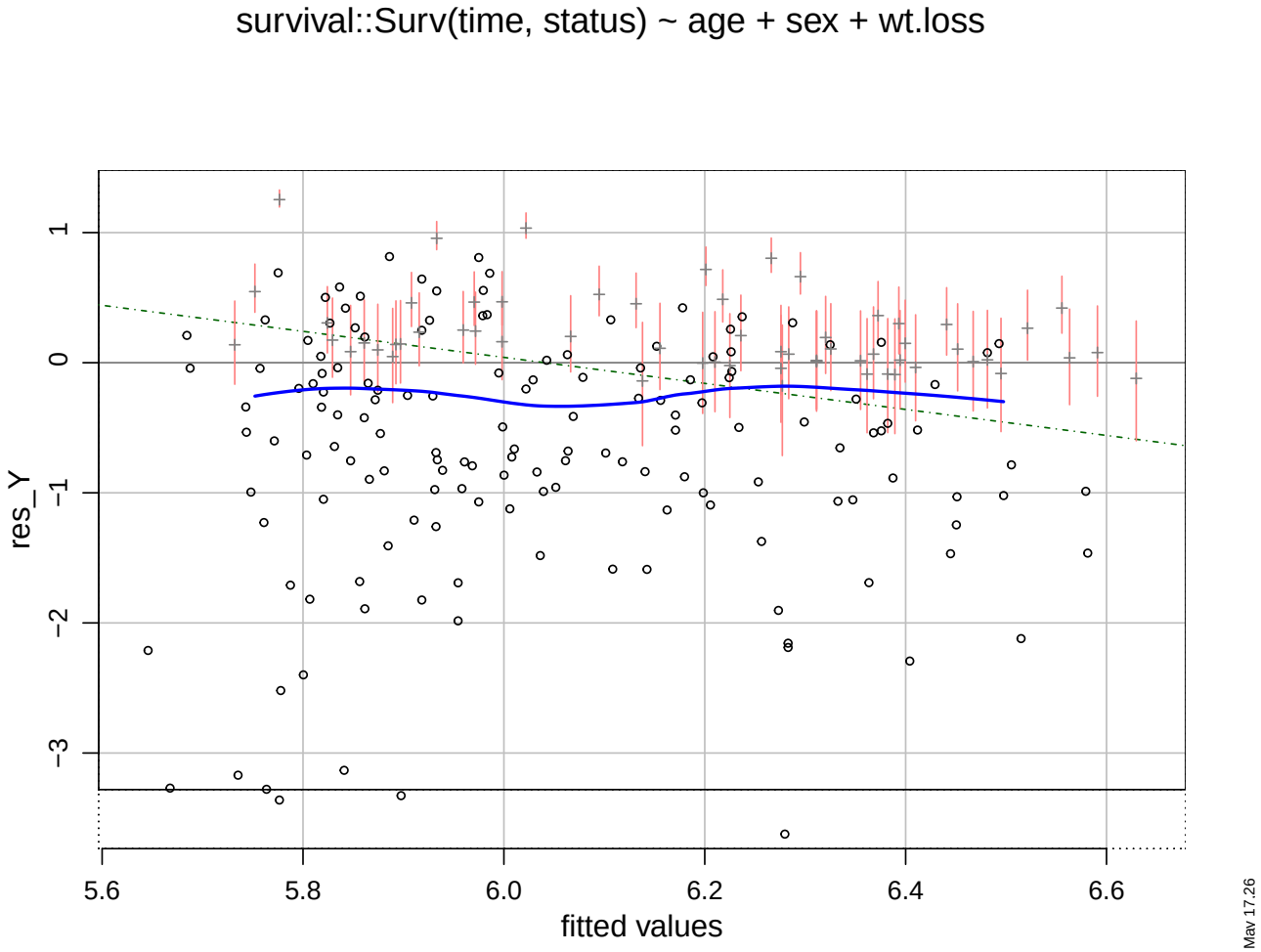
4.3 Censored residuals, conditional quantiles

For censored observations of a response variable, residuals are not clearly defined. In the case of right censoring, the underlying response value of a censored observation is known to exceed a given threshold. Therefore, the "true residual" exceeds a corresponding threshold. Then, the fitted model defines a conditional distribution for the true residual.

Conditional quantiles. Function `condquant` calculates the quartiles of the conditional distribution for each residual and, in addition, generates a corresponding random number. It also stores the probability of the condition.

These quantities are then used for plotting: the conditional median is shown together with segments connecting the conditional quartiles. This results in residual plots like those shown in the figure.

```
require(survival)
## data(lung) ## not needed temporarily, produces erroneous warning
r.lung <- survreg(survival::Surv(time,status) ~ age+sex+wt.loss, data=lung)
plregr(r.lung, plotselect=c(default=0, resfit=1), xvar=FALSE, smooth.sim=0)
```



4.4 Residuals for the Cox model

The Cox proportional hazards model, the most frequently used model in survival analysis, is a semi-parametric model. There is no obvious meaning of the notion residual in this context. The Cox-Snell residuals R_{CS} are defined in a way that they always follow an exponential distribution. Since this is an unusual law for residuals, it is convenient to transform them such that they then obey a standard normal distribution,

$$R = \Phi^{-1}(1 - \exp(-R_{CS})) \ .$$

Note that it is useless to draw a QQ-plot of these residuals, since they obey the normal law by construction. They should be plotted against the linear predictor values (Tukey-Anscombe plot) and against the input variables.

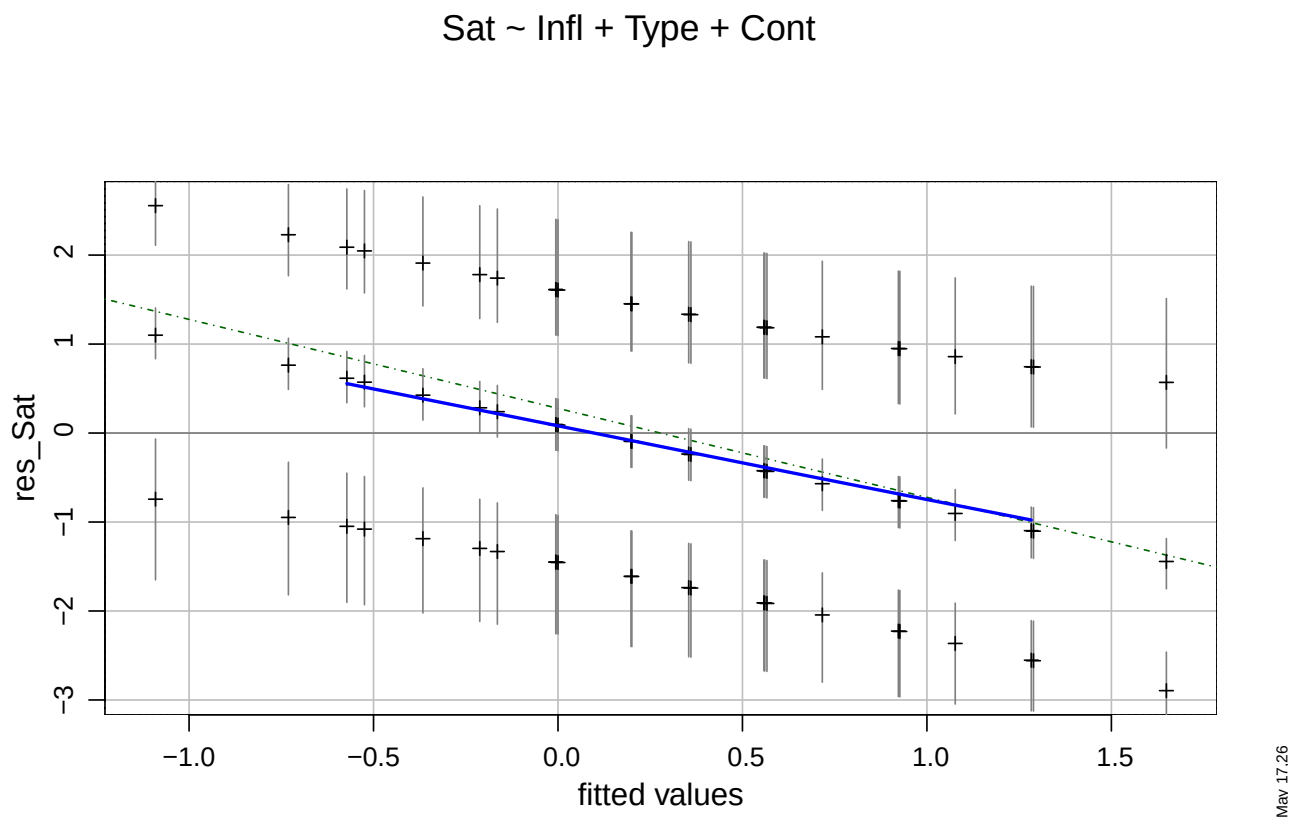
* The censored observations are shown with lighter color than the uncensored ones: their `pcol` is paled by applying `colorpale` with a `pale` of `ploptions("condquant.pale")[1]` . The color of the bars representing the quartiles is the paled `pcol` , paled again by `pale=ploptions("condquant.pale")[2]` If all observations

are censored, no paling is applied to the symbols, and `ploptions("condquant.pale")[1]` is used for the bars.

4.5 Ordinal and binary (logistic) regression

In ordinal regression, the response variable is modeled as a classified version of a continuous latent variable, which in turn follows a linear model with logistic (or normal) distribution of the random errors. According to this construction, the latent variable $\widetilde{Y}_i$, given the observed ordered variable Y_i and the linear predictor η_i, follows a truncated logistic (or normal) distribution. As with censored variables (Section 2.5), this yields conditional quantiles, which can be represented as described above.

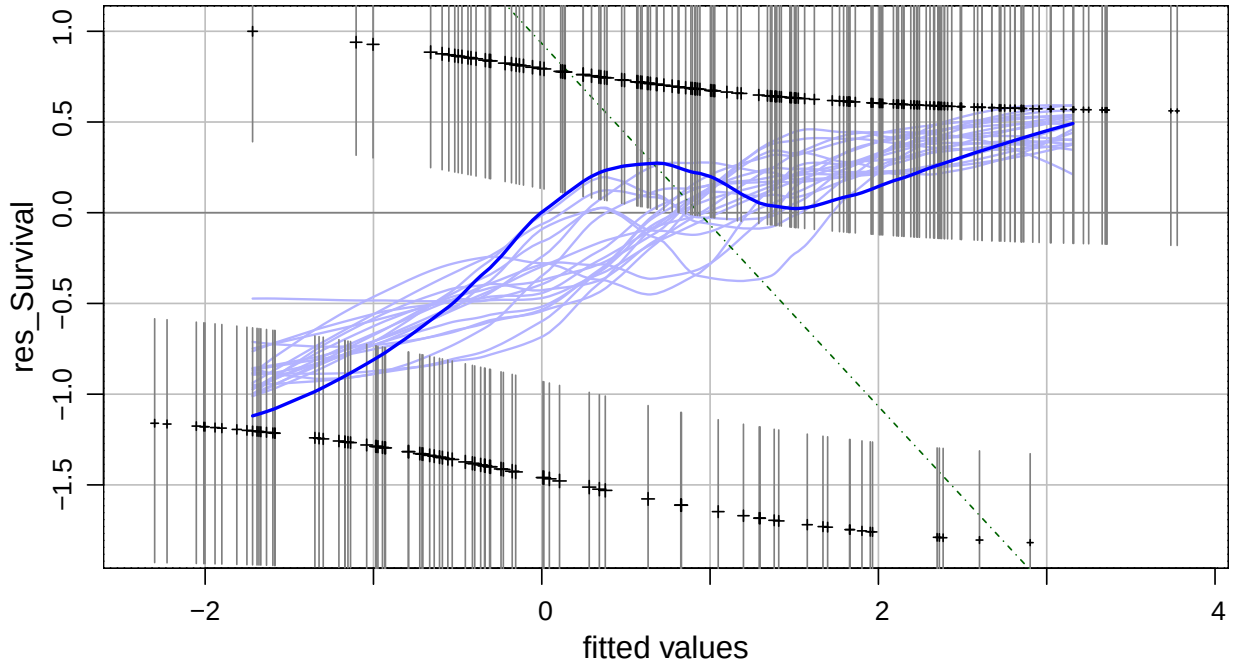
```
require(MASS)

## Loading required package:  MASS

data(housing, package="MASS")
rr <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
plregr(rr, plotselect=c(resfit=2, default=0), xvar=FALSE)

##
## Re-fitting to get Hessian
```

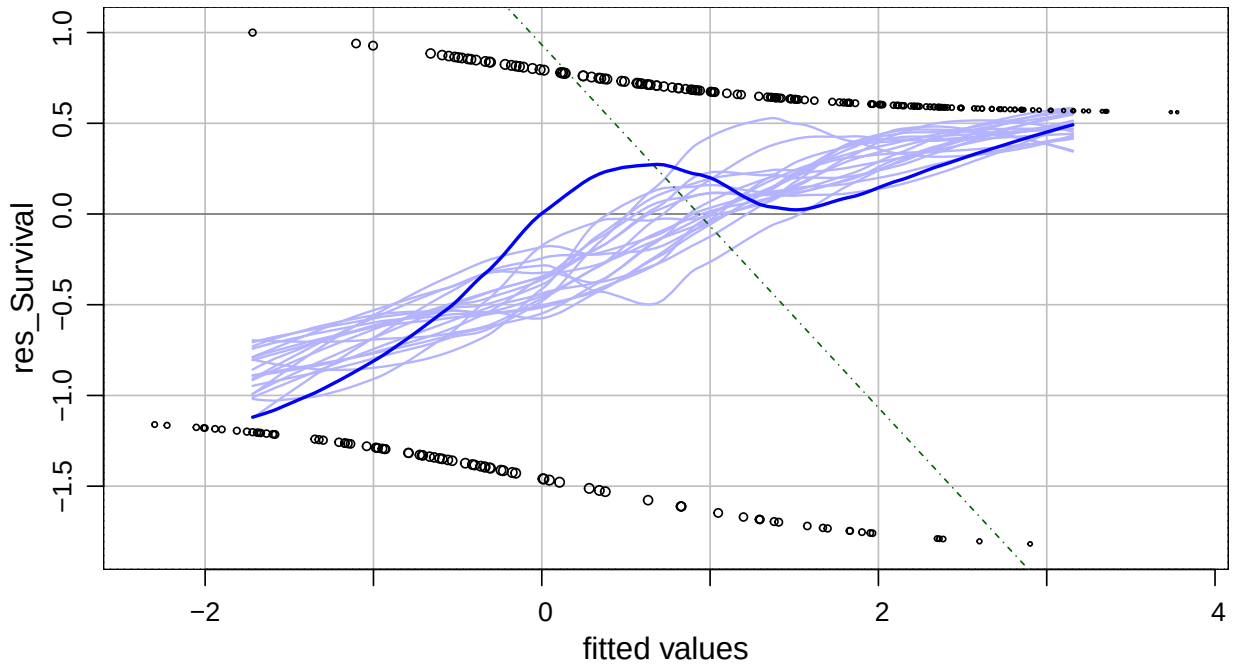


A logistic regression can be seen as a special case of an ordinal regression. Therefore, conditional quantiles can also be obtained for the residuals, and the respective displays can be generated. The following figure shows the residuals against linear predictor values, first with conditional quantiles, then with "working" residuals, one of the usual choices. Note that the smooth lines usually are increasing. As the simulated lines show, this does not means that there in any deficiency. The upward bump of the solid line, however, may be seen as a deviation from model assumptions.

```
data(d.babysurvival)
r.babys <- glm(Survival~Weight+Age+Apgar1,data=d.babysurvival,family=binomial)
plmframes(2,1, mar=c(3,3,3,1))
plregr(r.babys, plotselect=c(resfit=2, default=0), xvar=FALSE, mf=FALSE)
plregr(r.babys, plotselect=c(resfit=2, default=0), condquant=FALSE,
          xvar=FALSE, mf=FALSE)
```

Survival ~ Weight + Age + Apgar1





May 17.26

* If `condquant` is false, the type of residuals is selected by the argument or ploption `glm.restype`. Its default is `"working"`, because the linear approximation of the model corresponds to a weighted linear regression with the linear predictor values and the working residuals as fitted values and residuals. The weights are shown by the sizes of the plotting symbols.

4.6 Residual plots for multivariate regression.

For multivariate regression, the plots corresponding to each target variable should be examined. They are arranged such that the same type of plot for the different response variables are grouped together, and a matrix of plots is produced for residuals against input variables. In addition, the qq-plot of Mahalanobis norms of residuals is shown as well as the scatterplot matrix of residuals.

4.7 Residual differences and Distance in x-space

If there are groups of observations with identical input variables (or identical x_i vectors), then there is the possibility to perform a kind of overall goodness-of-fit test by comparing the variability of the residuals within these groups to the estimated variance of the random deviations. This is performed by a simple Analysis of Variance of the standardized residuals with the grouping just mentioned. (Note that this method can be applied even if some or many x_i vectors are unique.)

If such groups are not available, an old idea, described by Daniel and Wood (1971/80) is to use "near replicates" for this purpose. They introduced a distance in x-space that they called WSSD and suggested to calculate all distances between observations along with the corresponding differences of residuals. If the differences of residuals are smaller for short distances than for long ones, then this points to a lack of fit of the model, which might be avoided by introducing additional terms or modelling nonlinear dependencies through transformations of variables. The procedure however does not suggest which modifications may help.

As to the distance measure d_{hi}, Daniel & Wood (1971) introduced their WSSD (weighted Squared Standard Distance (?)) with the intention to weigh variables according to their "importance" for the model. Since this feature seems too much dependent on the parametrization of the model, I prefer to use the Mahalanobis distance in the design space,

$$d_{hi}^{(X)} = n(\underline{x}_i - \underline{x}_h)^T (\boldsymbol{X}^T \boldsymbol{X})^{-1} (\underline{x}_i - \underline{x}_h)$$

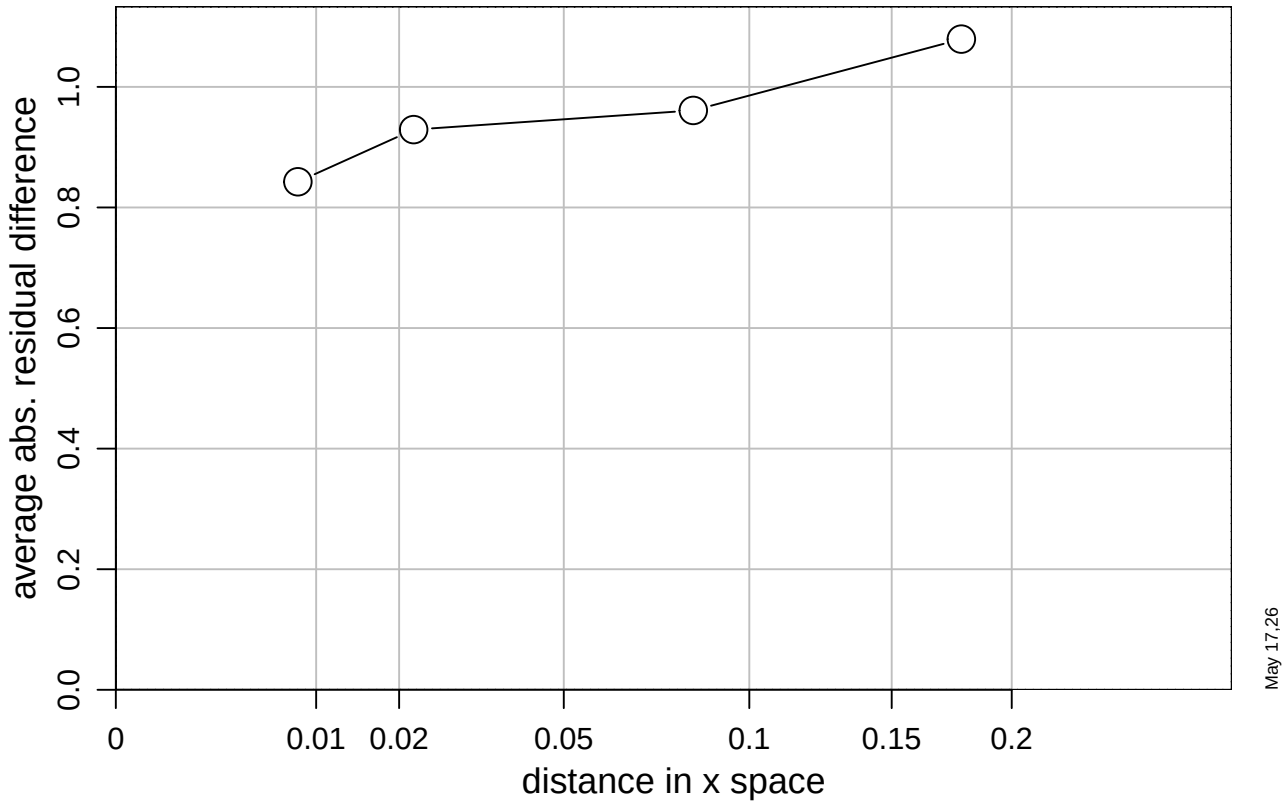
(which can be calculated efficiently from the QR decomposition of $\boldsymbol{X}$).

The distance pairs and the respective differences of residuals are calculated by the function `xdistResdiff`. They are grouped and the corresponding group means of absolute residual differences are then produced by `xdistResscale`. The distances are then classified, and the residual differences are averaged over these classes. This yields a pair of mean distance $\bar{d}_k^{(X)}$ and mean residual differences $\bar{d}_k^{(R)}$ for each class k. (Trimmed means ($\alpha = 1/6$) are used by default to obtain some robustness against outliers.) These pairs are then plotted.

The class limits are chosen by fixing the percentages of distance values that they should contain. By default, the first class is quite small – because plausible deviations from the null hypothesis will lead to lower mean residual differences for short distances –, followed by a somewhat larger class, then a really big class for an intermediate range and finally a class (again somewhat smaller) for the upper end. The current default limits are therefore at 3%,10%, and 90%.

In order to assess the variability of the mean residual differences, we resort to simulation. The residuals are permuted among the observations, and then the calculation of differences is repeated. (In fact, the assignment of residual differences to the observation pairs is re-determined.) This leads to simulated mean differences for each class, and their standard deviations can be used as standard errors of the observed mean differences. Corresponding error bars are shown in the plot.

```
data(d.blast)
rr <- lm(tremor~distance+charge, data=d.blast)
## an inadequate model!
xdrd <- xdistResdiff(rr)
plot(xdrd, zeroline.col="darkgreen")
```



This idea can also be used for a test of goodness of fit: The test statistic

$$T = \sum_{k=1}^{K} \left((\bar{d}_k^{(R)} - \bar{d}) / \mathrm{se}_k \right)^2$$

should have a Chisquared distribution with $K - 1$ degrees of freedom. The test is calculated by the function `xdistResscale` along with the mean distances and mean residual differences.

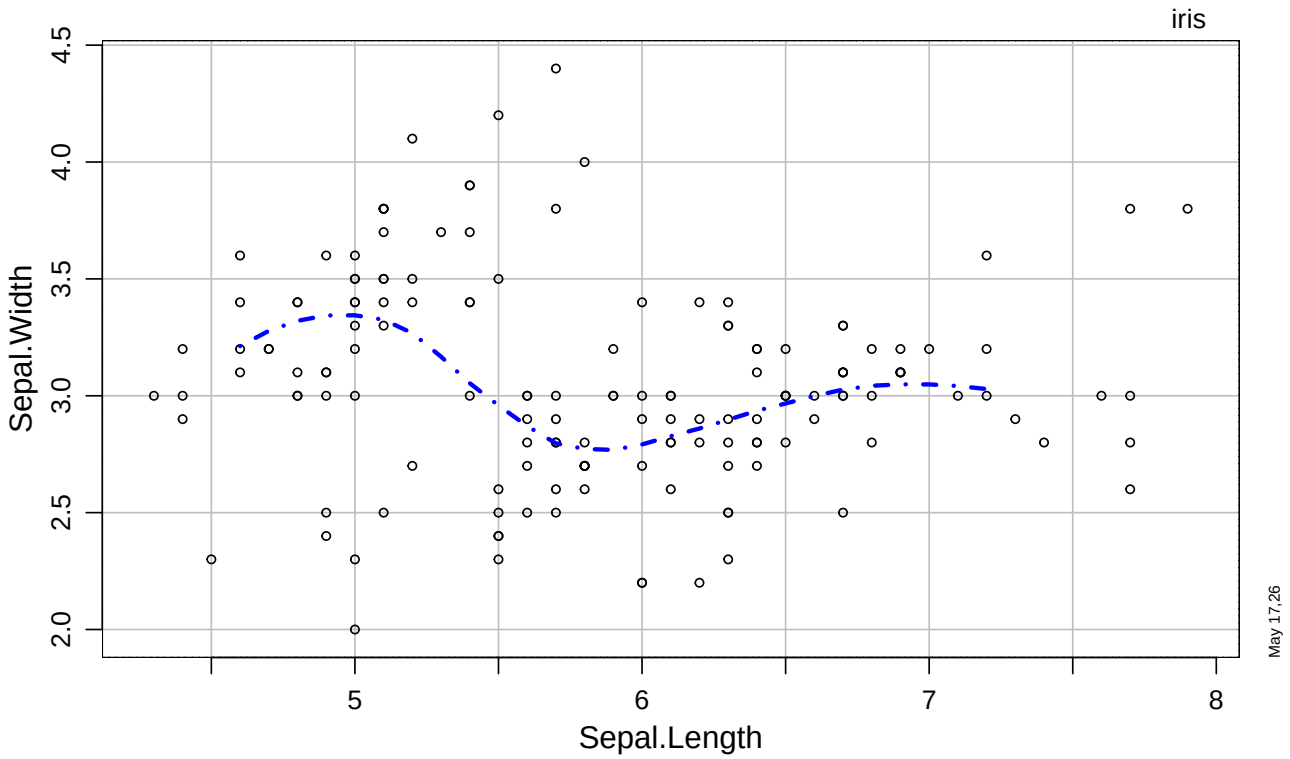
5 Specification of graphical elements

A central motivation underlying the `plgraphics` package consists of allowing for using graphical elements very flexibly and implementing an easy way to specify and maintain such options. They are set either explicitly by calling `ploptions` or generated by high level pl functions and stored for further use.

5.1 Pl options

The graphical elements, like plotting character, color, line types, etc. to be used in pl graphics are specified by the function `ploptions` in the same way as R's graphical "parameters" and other options are determined by the functions `par` and `options`. In addition to such general options, elements determining how variables will be plotted are stored along with the data as "properties" of variables.

```
t.plopt <- ploptions(col="magenta", pch=3, smooth.lty=4)
plyx(Sepal.Width~Sepal.Length, data=iris)
```




```
## restore the old options
ploptions(list=attr(t.plopt, "old"))
```

There are some differences between the behavior of `ploptions` and `par`:

- The pl options are stored in a list `.ploptions` in the environment `pl.envir`.
- There is a list `default.ploptions` in the package. It collects the package's default settings and is used as a backup if some components should not be contained in `.ploptions`.
- Both of these lists can be overridden by objects with the same name that appear earlier in the search list than the `plgraphics` package.
- The value returned by `ploptions` is the entire, modified list of pl options. The old values are stored in the attribute `attr( ,"old")` to be used for restoring them, see above.

Remark. The concept of the ploptions list is a version of a more general idea, suggesting that the default values of any "high level" R function should have an associated list of default arguments, which is not contained in the function definition, but stored separately. This would allow the user more generally to specify his own style by changing these defaults and storing them in a kind of style file to be loaded at the start of each session. Here, there is only one list because the various pl functions need the same graphical elements.

Thus, a graphical element like a plotting character is generally searched in

1. the argument list of the calling function,
2. the `ploptions` argument of the calling function,
3. the `.ploptions` list in environment `pl.envir`,
4. the list `default.ploptions` in the package `plgraphics` or in an environment hiding it.

The components of these lists include

- `colors`, the palette of colors to be used;
- `linewidth`, the linewidths used for the different line types. If the line types are shown with the same `lwd`, they are perceived with different intensity. `linewidth` intends to compensate this effect.
- `csize`, a factor applied to the current value of `par("cex")` to determine the general character expansion used by the pl functions.
- `csize.pch`, the median character expansion. The default is the function `charSize`, with argument `n`, defined as `min(1.5/log10(n), 2)`, that is called when the number `n` of observations is available. Alternatively, a fixed scalar can be given.
- a group of basic components: `pch, csize, csize.pch, csize.plab, lty, lwd, col`.
  `csize` is a factor which will be applied to `csize.pch` above for showing points by a single symbol (`pch`),
  `csize.plab` is an additional factor applied for the points that are shown by `plab`.
- a group of components named `group` (`group.pch, ...`). They characterize how different groups will be displayed. Thus, `group.pch` should be a vector defining the plotting symbol for the first, second, ... group (when there are groups in the data).
- a group named `variables`, defining the elements to be used when different variables should be distinguishable;
- `censored.pch` and `censored.csize` used to show censored observations;
- `mar, oma, mgp, thickintervals`, specifying the number of lines in the figure's margins and outer margins, the "margin parameters" as in `par`, and the targeted number of tick intervals for axis labelling. The latter usually consists of 2 numbers, specifying the number of intervals for all ticks and for labelled ones, respectively.
- `stamp`, logical value determining if a stamp should be added in the bottom right corner of each plotting page;

- a group `innerrange`, determining if and how an inner plotting range should be used and generated;
- `plext`: factor by which the range of the data should be extended unless an inner range is active (in which case the extension is determined by `innerrange.ext`), and
  `plsymbolext`: further extension to allow for large symbols near the limits of the plotting range;
- `markextremes` sets the proportion of extreme points that are shown with labels to help identify them. If set to `TRUE`, the proportion depends on the number `n` of observations through `ceiling(sqrt(n)/2)/n`.
- `title.csize` determines the character expansion of the plot title. By default, it adapts to the length of the title. For long titles, it will however never be smaller than `title.csizemin`. If `title.csize` has 2 elements, the second refers to the subtitle.
- a group `grid`. If `grid` is a list of two vectors, it contains the values where vertical and horizontal thin lines are drawn. If it is `TRUE`, the gridlines correspond to the tickmarks of the two axes. `grid.lty, grid.lwd, grid.col` set the respective properties for the gridlines.
- a group `zeroline`, which is analogous to `grid`, but the default of `zeroline` is `TRUE`, which leads to using 0 for numerical variables and none for factors. The properties are independent of those for `grid`;
- a group `refline`, where the component `refline` can be logical or `==2`, the latter indicating the a band should also be shown if available. Alternatively, `refline` can be a vector of length 2, which will be interpreted as intercept and slope of a straight line, or be a function that yields such a vector and takes as the first two arguments `x` and `y` or a first argument named formula which will be set to `y~x`. The properties `lty, lwd, col` to be used in `plrefline` are again defined as above;
- a group `smooth`, containing items needed for generating and drawing smooth lines;
- a group `bar` needed to show reference values for levels of factors used as input variables in regression diagnostic plots;
- `factor.show` indicates the way in which plots with factors are shown, presently only if one of the two variables (x or y) is a factor and the other, quantitative. Then, the factor can be jittered and then used as a quantitative variable, or a box plot or a "multibox plot" can be chosen.
- `jitter`: logical indicating if factors should be shown with jittering. A named vector may be given that defines the jittering for each variable.
  `jitter.factor` is the jittering factor used, see `?jitter`.
- `condprobrange` is used to determine which bars should be shown in the case of censored data.
- `functionxvalues` contains the number of argument values for which a smooth function or fitting component is evaluated in diagnostic plots.

- `leveragelimit` determines the range used for leverage values when plotting residuals against leverages.
- A group `plcond` controlling features within the function `plcond`.

If these options modify any setting of R's `par` list and they are set as arguments to high level pl functions (`plyx, plmatrix, plmboxes, plregr, plresx`), as well as `plframe` and `plaxis`, they are restored after leaving the function (please report failures!). When `plmframes` is called by the user, the information to restore the old settings is contained in its (invisible) value. Thus,

```
op <- plmframes(1,3)
## plyx(Sepal.Width~Sepal.Length, data=iris, group=Species, mar=c(3,1,3,0))
par(attr(op, "oldpar"))
par("mfg")

## [1] 1 1 1 1
```

will restore the old settings. Here, we have commented out the plotting statement to save space.

The options that are finally used in high level pl displays are saved as the list `ploptions` in an environment called `pl.envir`. They will be retrieved by subsequent low level pl functions, like `plpoints`. They can be inspected and changed by, e.g., `pl.envir$mar` or `pl.envir$mar<-c(5,3,2,1)`. (The list `.ploption`, with a leading dot, stores the options set by the function `ploptions`.)

5.2 Graphical metadata, attributes of variables used for plotting

Pl graphics rely on generating and maintaining metadata that guide the details of creating the plots. They are stored as an enriched dataset that contains the variables needed for the displays called `pldata`. We first describe them and then show how `pldata` is generated and made available.

Properties of observations. The first type of such properties relates to the observations and includes the plotting symbol, color and size for each of them. These specifications are

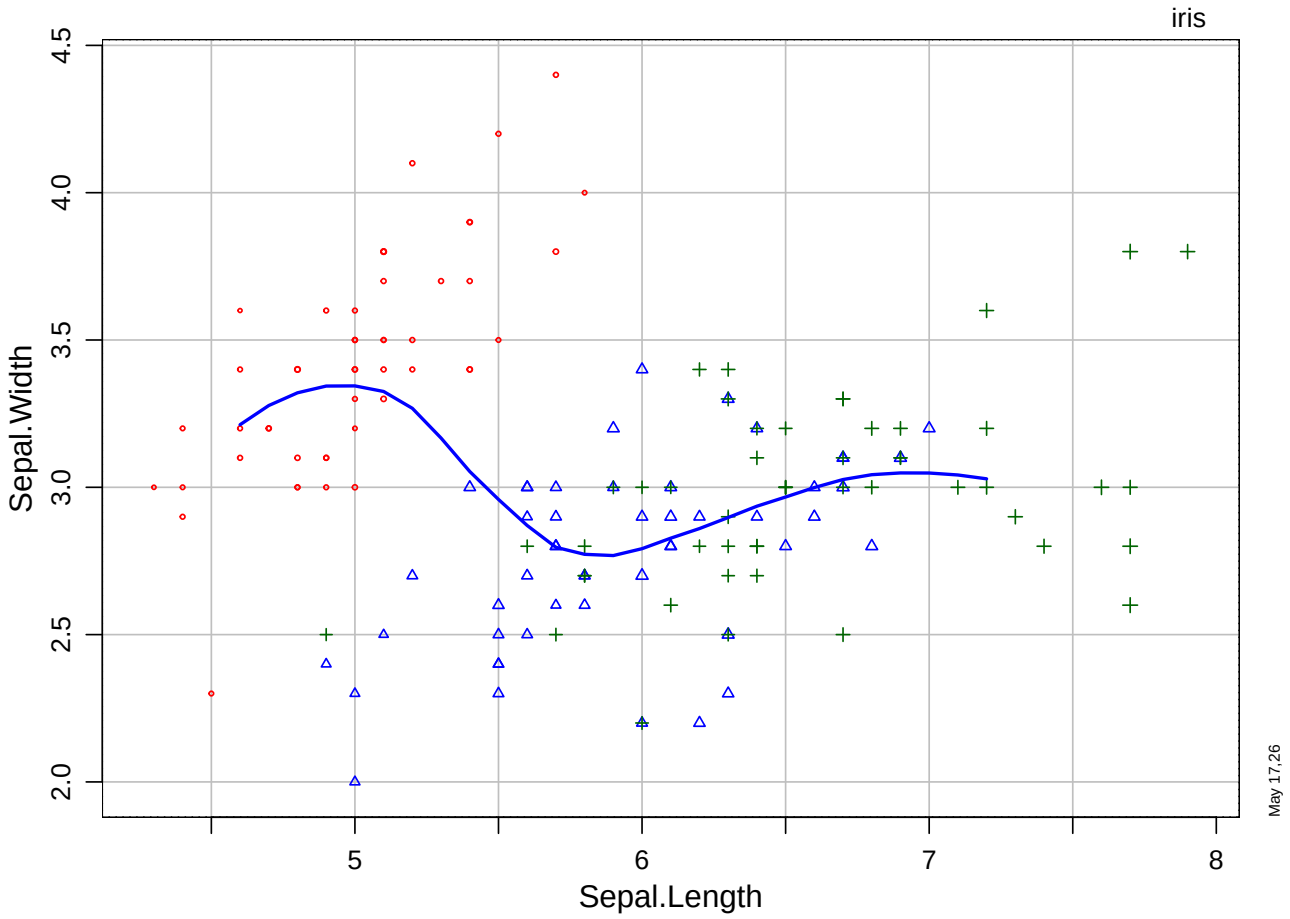
- `pch`: plotting symbol (character);
- `plab`: plotting label, an extension of `pch` to more than one symbol, often used to identify observations;
- `psize`: plotting size, scaled by the pl option `csize`;
- `pcol`: color of the symbol. This can be a factor. Then, the colors in `ploptions("group.color")` are used according to the `levels` of the factor.
  It can be numeric and should then contain low integer values. Values > 0 are used again to select from `ploptions("group.color")`; A value 0 selects `ploptions("color")[1]` which is black unless changed. Negative values lead to a color of `NA` and will suppress plotting of the

point.

If `pcol` is logical, it is converted to `pcol+1` and selects the first two group colors.

These elements are stored in `pldata` as columns with names `(pch)`, `(plab)`, `(psize)`, `(pcol)`. They are specified by the user usually by setting the corresponding arguments `pch, plab, psize, pcol` in high level pl functions. These arguments are not listed explicitly in the help files of the high level functions. Instead, these functions call `pl.control` to set them. Alternatively, they may already be contained in the dataset given by the argument `data`.

```
plyx(Sepal.Width~Sepal.Length, data=iris,
     pch=Species, psize=Petal.Length, pcol=Species)
```




```
table(pl.envir$pldata[,".pch."])

##
##  1  2  3
## 50 50 50
```

As the last statement shows, the `pldata` dataset is available after calling the plotting function in the environment `pl.envir`.

Properties of variables. A second type of properties determines how variables should be displayed. These are

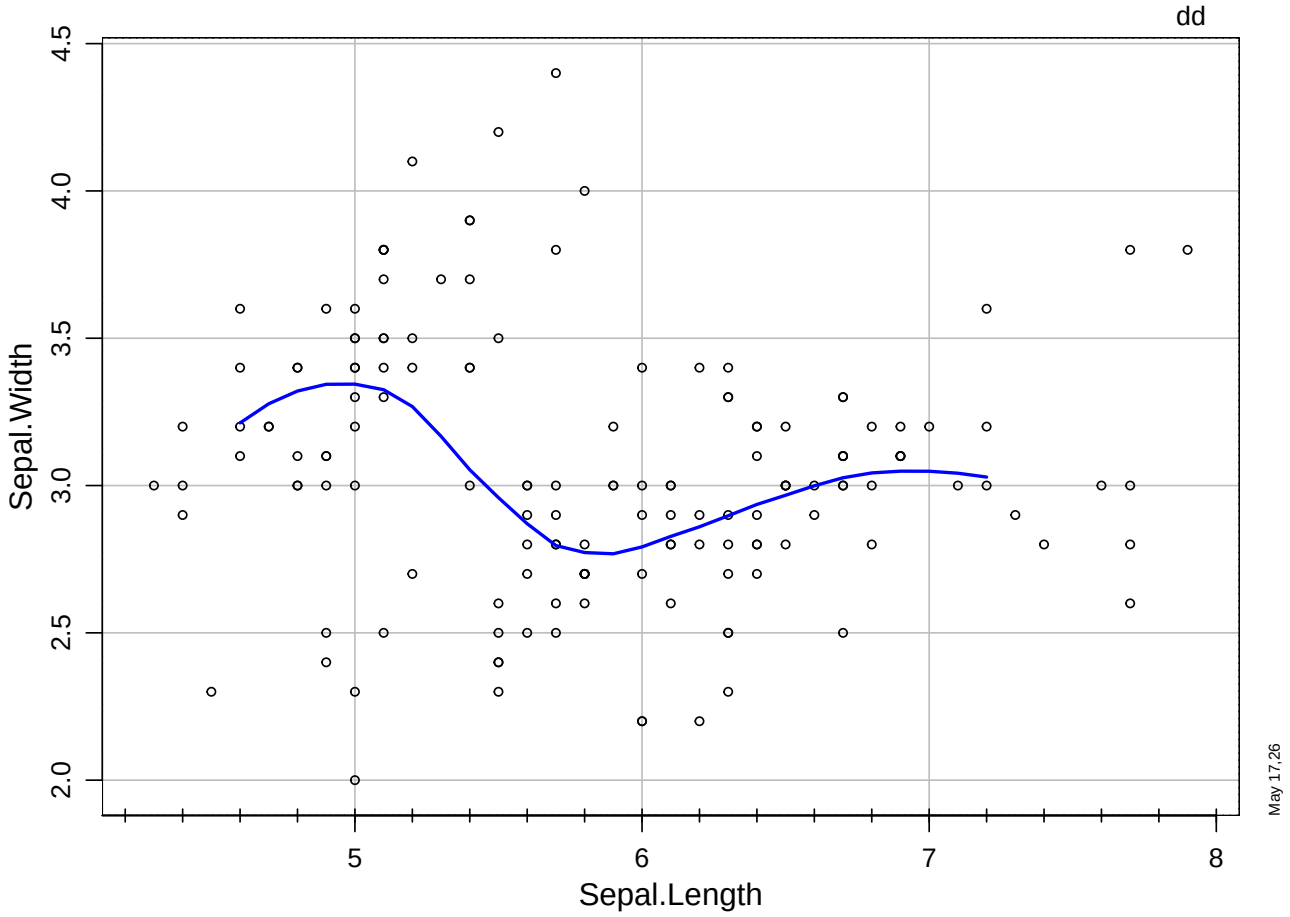
- `varname` and `varlabel`: variable name and label – to be used for labelling the axis on which the variable is shown –, typically identical to the name of the variable in the data.frame it comes from;
- `numvalues`: numerical values to represent the given data values in case these are not numeric or for other reasons, see `gendateaxis` below;
- `plscale`: transformation function to generate the "plotting scale";
- `plvalues`: values used for plotting if different from `numvalues`, typically transformed values;
- `plcoord`: plotting coordinates, if different from `plvalues`, typically when an inner plotting range is active.

Thus, plotting coordinates are defined as those first found searching for attributes `plcoord, plvalues, numvalues` and finally the original values of the variable. Further properties are:

- `innerrange, plrange`: inner and outer plotting range;
- `innerrange.ext`: extension used to calculate `plrange` from `innerrange`, if the latter does not cover all points;
- `nouter`: the number of points modified at each end of the inner range;
- `vlim, vlimsc`: range within which the plotting should occur, that is, `xlim` or `ylim`, if the variable is used as the horizontal and vertical axis, respectively, either on original or transformed scale;
- `ticksat`: the values for which tick marks will be shown in plots. This item may carry an attribute `small` that leads to an additional set of smaller tickmarks;
- `ticklabelsat` the possibly tick labels `ticklabels`: positions of labels to indicate values of the variable, and the labels themselves;
- `vpch, vlty, vcol`: plotting symbol, line type and color to be used if multiple y's are shown in a plot. These elements are obtained by default from `ploptions("variables.xxx")`, where `xxx` is `pch`, `lty`, or `col`.

These elements are stored as attributes of the variables, e.g., `attr(var, "thicksat")`. They can be set (or generated by the function `genvarattributes` and then modified) before calling the high level pl function. Those that are needed and have not been stored beforehand will be generated by `pl.control` when calling such a function.

```
dd <- iris ## (avoid a modified version of  iris  in .GlobalEnv)
attr(dd$Sepal.Length, "grid") <- seq(4,8,0.5)
attr(dd$Sepal.Length, "ticksat") <- structure(seq(4, 8, 1), small=seq(4,8,0.2))

plyx(Sepal.Width~Sepal.Length, data=dd)
```

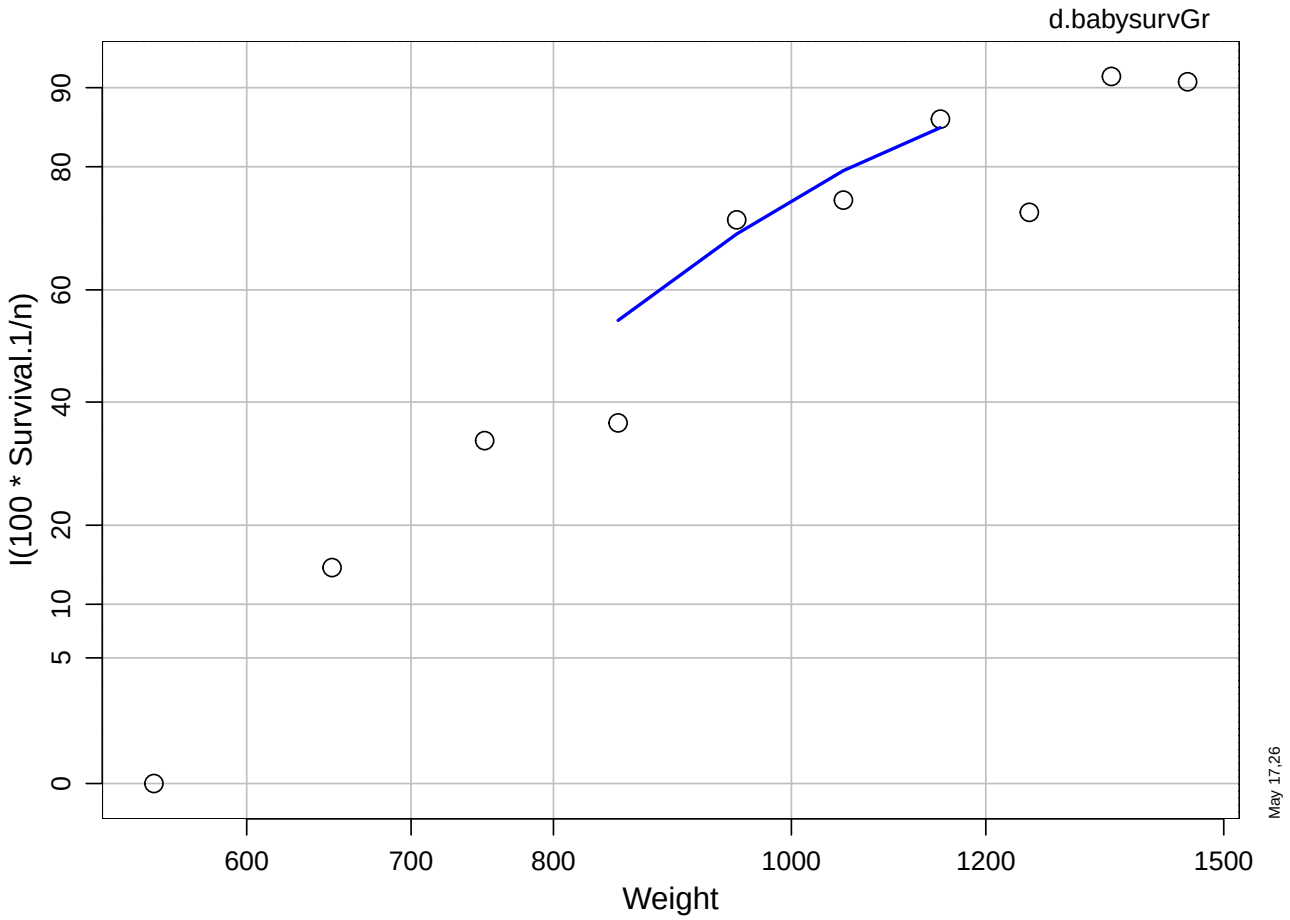


`pl.control`. High level pl functions call the function `pl.control` first. It generates the "plotting dataset" `pldata`, which collects data dependent information needed for plotting in an enriched, standardized form. It also takes any further arguments to be passed on to `ploptions`. The result is stored in the `pl.envir` environment. This allows for inspection of the plotting data `pl.envir$pldata` and the active `ploptions` (`pl.envir$ploptions`) and thereby helps debugging.

Plot scale. Showing a variable with a "plot scale", like "log scale", means to transform the values with a given function (`log`), storing them in the attribute `plvalues`, and setting `ticksat`, `ticklabelsat` and `ticklabels` corresponding to "pretty" values of the original scale. The function that generates these attributes is `plscale`. It can be called explicitly in order to fix the scale as a

property of the variable in the dataset it comes from, or implicitly by setting the argument `plscale` when calling a high level function, see 2.2.

```
dd <- d.babysurvGr
plyx(I(100*Survival.1/n) ~ Weight, data=d.babysurvGr, plscale=c("log","asinp"))
```




```
## or, in order to fix the plotting scale for further use
t.survp <- with(dd, 100*Survival.1/n)
dd$SurvPerc <- plscale(t.survp, "asinp")
dd$Weight <- plscale(dd$Weight, "log")
attr(dd$Weight, "ticklabels")
```

```
## [1] "600"  "700"  "800"  "1000" "1200" "1500"
```

```
str(dd$SurvPerc)
```

```
##  num [1:10] 0 14.3 33.3 36.4 71.9 ...
```

```
##  - attr(*, "plvalues")= num [1:10] 0 0.247 0.392 0.412 0.644 ...
##  - attr(*, "ticksat")= num [1:8] 0 0.144 0.205 0.295 0.436 ...
##  - attr(*, "ticklabelsat")= num [1:8] 0 0.144 0.205 0.295 0.436 ...
##  - attr(*, "ticklabels")= chr [1:8] "0" "5" "10" "20" ...
##  - attr(*, "plscale")= chr "asinp"

  ## or  dd <- genvarattributes(dd, plscale=c(Weight="log",SurvPerc="asinp"))
  ## plyx(SurvPerc~Weight, data=dd) ## now produces the same plot as above
```

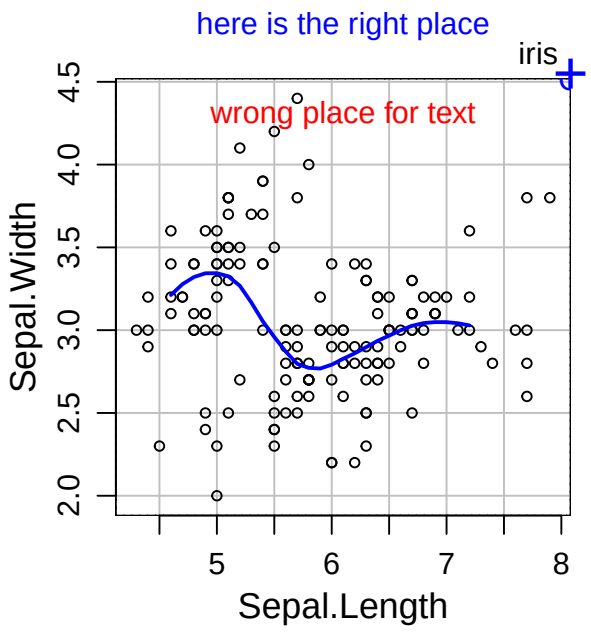
Date variable. Similarly, the function `gendateaxis` provides nice tick values and labels for variables expressing a date or time. The function `gendate` helps to generate such a date. If a variable inherits from class `Date` or `times`, it will be shown automatically in this way.

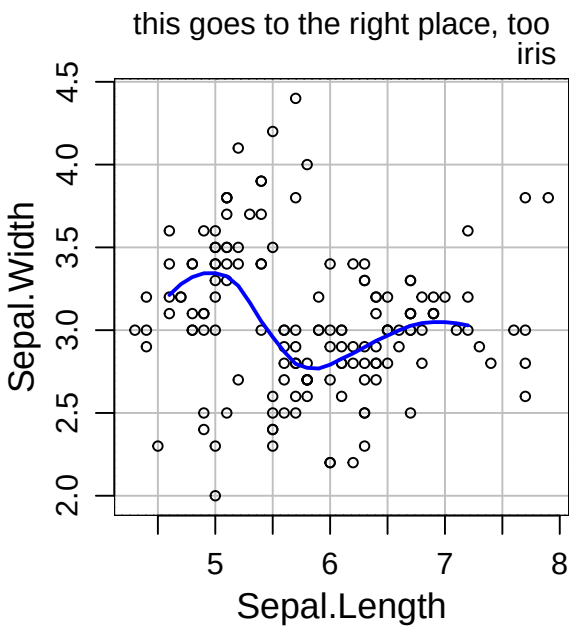
5.3 Graphical parameters inside and outside high level pl functions

The high level pl functions set graphical parameters (by calling `par`) according to their needs and they often set pl options. When they end, they reset the `par` parameters to the previous values unless the argument `keeppar` is `TRUE`. Since quite often, the parameters determining the width of the plot margins are changed, subsequent calls to low level R functions would not work properly, that is, points within the plotting area as well as elements put to the margin by `mtext(...)` would be wrongly positioned. As an alternative to setting `keeppar=TRUE`, the function `plmarginpar` can be called to set the "margin parameters" `mar, oma, mgp, cex` to the values used in the last call to a high level pl function. These are available from `pl.envir$ploptions` (unless `assign` was `FALSE` in this call).

```
plmframes(1,2)

par(mar=c(2,2,5,2))
plyx(Sepal.Width~Sepal.Length, data=iris) ## margins according to ploptions
par("mar") ## parameters have been recovered

## [1] 2 2 5 2

mtext("wrong place for text",3,1, col="red")  ## margins not appropriate for active plot
t.usr <- par("usr")
points(t.usr[2],t.usr[4], pch="X", col="red")
plpoints(t.usr[2],t.usr[4], pch="+", col="blue")

t.plo <- plmarginpar() ## get margin parameters from pl.envir
  ## generated by the last pl graphics call
par("mar")
```

```
## [1] 3.20 2.96 2.60 1.66

mtext("here is the right place",3,1, col="blue")
points(t.usr[2],t.usr[4], pch="O", col="blue")

par(attr(t.plo, "oldpar"))  ## restores old 'margin parameters'

## named list()

par("mar")

## [1] 3.20 2.96 2.60 1.66

plyx(Sepal.Width~Sepal.Length, data=iris, keeppar=TRUE)
par("mar")

## [1] 3.20 2.96 2.60 1.66

mtext("this goes to the right place, too",3,1)
```

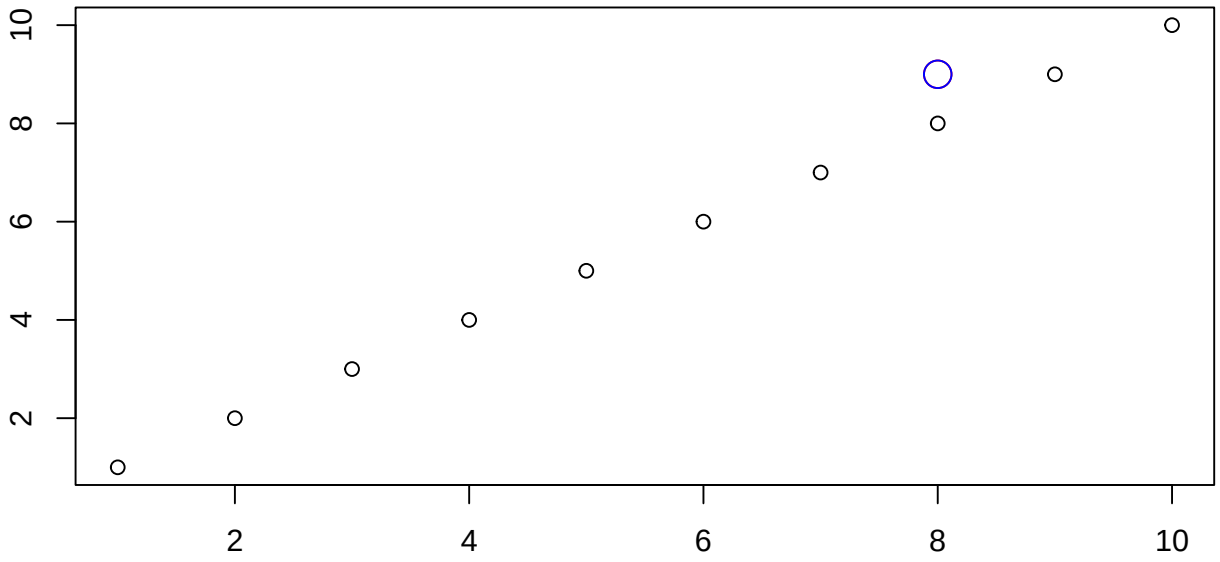




May 17.26

The low level pl functions to be presented next avoid this problem. However, if you want to use these for adding to a figure started with a standard plotting function (and you have different margin parameters from those in `usr.ploptions`), then you need to set `marpar=FALSE` in their call.

```
par(mar=c(2,2,5,2))
plot(1:10)
plpoints(8,9, col="red") ## wrong
plpoints(8,9, col="blue", setpar=FALSE) ## correct
```



6 Low level graphics

Like in basic R, there are "low level" graphical functions that add to an existing plot, whereas "high level" functions are designed to generate a full plot. Low level plotting functions include:

- `plframe` generates a new frame, frames the inner and outer plotting ranges and draws gridlines and axes, the latter by calling `plaxis`.
- `plaxis` draws an axis based on the attributes of the variable given as the second argument.
- `plpoints` draws points and lines.
  In the simplest case, this function places the plotting symbol at the given coordinates. As

the basic `points` function, it draws lines if the argument `type` is set to `"l"` or `"b"`, and the argument `pch` (or the column `".pch."` in `plargs$pldata`) can provide different plotting symbols for the different points.

`plpoints` also includes the capabilities of `text`: If the argument `plab` is set (or `plargs$pldata` contains a column named `".plab."`), it should be a character vector and is reproduced at the (x,y) locations. Values `NA` or `""` being replaced by the plotting symbol in `pch`.
The size of the plotting symbols or strings is determined by `plargs$pldata[,"psize"]` if available and by the ploptions `csize` and `csize.pch`.

If one or both coordinate variables contain censored values (that is, if they inherit from the class `"Surv"`), the censored observations are shown by the appropriate symbol from `ploptions("censored.pch")` and with paled color (according to `ploptions("censored.pale")`).

- `plmark` can be used to mark extreme points by labels and leave the non-extreme ones to be shown by the plotting symbol.
- `plsmooth` and `plsmoothline` generates a smooth line and draws it in the plot, respectively.
- `plrefline` adds reference lines (straight lines or curves) to a plot. It is used by `plregr` and `plresx`.
- `pltitle` adds a title. By default, the character size (given by the ploption `"title.csize"` is decreased for long titles (`main` or `sub`) to fit it onto one line.
- `pllimits` and `plcoord` determine inner plotting range (see above) and the respective coordinates where the points outside of it will appear on the plot.
- `stamp` adds a time stamp and, if available, a project and analysis step title to the right bottom corner of the plotting page. This is avoided by setting `ploptions(stamp=FALSE)`.

`plpanel` is a "medium level" function. It calls all of the above functions. The user can re-program this function to modify and expand the actions that are taken, store the modified function, e.g. under `my.panel` and then set the argument `panel = my.panel` in `plyx` and `plmatrix`, or change the ploption, `ploptions(panel=my.panel)`.

Coordinates from last high level pl function. The low level functions that act on `x` and `y` coordinates can be used with empty `x` and `y` arguments. They will then obtain these values from `pl.envir$pldata`, which is generated and stored by high level pl functions and will thus reflect the coordinates that are likely the bases of the current plot. The pl options are also recovered from `pl.envir` (unless the argument `ploptions` is specified in the call to the low level function).

7 Auxiliary functions

These functions do calculations needed for generating graphical elements (like generating a smooth) or are useful additional functions, like `showd`, which displays a kind of summary of data. Let us start with the latter category.

Displaying data by `showd`. This function is useful for inspecting a data set or another object, avoiding overwhelming output. If the number of rows or list components is larger than 4, the first 3 of them and the last one are shown only. For rows in data.frames or matrices, two of the rows between the forth and the last one are also printed.

Subset of a data.frame. The function `plsubset` extracts a subset of a data.frame in a similar way as the `subset` function of plain R. The essential enhancement is that it keeps the graphical metadata and adjusts it where necessary.

Counting `NA`'s. `sumNA` counts the number of `NA`'s in vectors, matrices, and lists, including data.frames. For numerical objects, infinite values are counted as `NA`'s (unless the argument `inf` is `FALSE`).

Started logarithm. Variables that contain only positive numbers usually call for a log transformation according to the "first aid transformation" principle of data analysis introduced by John Tukey. If they also contain zeros, the `log` function turns them into `-Inf`. This is rarely desirable. A common way to avoid it is to add 1 to the variable before taking logs. This is unreasonable, since adding 1 has a very different effect on the result, depending on the magnitude of the range of non-zero values of the variable. At least, the constant should adapt to this magnitude. Another solution of the problem is given by the rule that below some value c, the function should be linear, with a smooth derivative in c. This is implemented in the function `logst`, defining the value c as a function of the quartiles of the non-zero values.

To be precise, `logst` is defined as

$$\begin{cases} \log_{10}(x) & \text{if } x \geq c \\ \log_{10}(c) - (c-x)/(c\log(10)) & \text{if } x < c \end{cases} .$$

The threshold c is set to $c = q_1^{1+r}/q_3^r$, where q_1 and q_3 are the quartiles of the positive data and r is a tuning constant. The default of r is 1 and leads to an expected percentage of about 2% of the data below c for log-normal data. It is certainly useful to inspect a graph of the function, as drawn in `example(logst)`.

The use of `logst` in regression will cause problems when calculating predicted values for new data, since its result depends on the whole dataset through the determination of c by default. Therefore, c can be given to `logst` by the argument `threshold`.

Other auxiliary functions include

- `getvariables` gets the variables contained in a formula.
- `clipat`, drops data outside a range or replaces them with `NA` or a suitable value.
- `warn` summarizes warnings, thus avoiding endless repetitions of the same warning.

Plotting properties. Auxiliary functions used for getting plotting properties are

- `gensmooth`, the wrapper function for getting smooths. It in turn calls `smoothRegr` by default.

- `robrange` determines a robust range of data to be used as inner plotting range.
- `simresiduals` simulates residuals used for generating additional smooths to which the actual one can be compared.
- `colorpale` determines a paled version of a color.

8 Details

8.1 Palette

Even though the palette that is active in the global environment is not affected, the pl functions use an implicit palette given by `ploptions("colors")`. By default, the list is

```
default.ploptions$colors

##  [1] "black"        "red"          "blue"         "darkgreen"    "orange"
##  [6] "purple"       "deepskyblue2" "green3"       "coral4"       "pink3"
## [11] "aquamarine3"  "brown2"
```

The selection of these colors has been driven by the following ideas:

- The first few should be readily distinguishable and reflect a common sequence: after the default "black", the most salient one is red, followed by blue, then "darkgreen" because "green" is lighter than the foregoing, and finally "orange".
- The first colors should be simply named. If the user explicitly sets one of them, e.g., by `attr(data$var, "col") <- "red"`, the pl functions have a chance to avoid them when making default decisions.
- After the first group of 4 alternative colors, another group of 4 follows. It is a modification of the first group. The third group is incomplete as no blueish color could be found that would be distinct enough from the foregoing colors.

8.2 Point labelling and plotting character

The properties of the symbols displaying observations are set by the following rules.

1. If they are specified by the respective argument to high level `pl` function (and evaluated by `pl.control`), this has priority (exception, see 2.).
2. In the case of multiple ys, colors are determined primarily by the argument `vcol` of the high level `pl` function, secondarily by the `vcol` attribute of the variables. Thirdly,

`ploptions("variables.col")` is used, avoiding colors that are already specified for some variables by the foregoing steps (see `i.getvarattribute`, called by `genvarattributes` in `pl.control`.)
If `pch` is not determined otherwise (argument, see 1., or group, see 3.) it is set in the same way. For plots of type `"l"` or `"b"`, the line type `lty` is determined in the same way as the color.

3. If there is grouping and only a single y, the group determines `pch` and its color by the `group` components of `ploptions` unless set by 1. above.

4. In other cases, the `default` group of `ploptions` is used.

8.3 Groups

Pl functions may use a grouping of observations, defined by an argument `group=...` in different ways. `plyx` splits the observations accordingly and produces a panel for each group.

Other types of groupings are:

- **Smooth groups.** The argument `smooth.group` allows for drawing a separate smooth line for each group.
- **Color.** Groups get different colors by specifying `pcol`. If the value is a factor, it will be converted into `ploptions("group.col")[as.numeric(pcol)]`. In order to give color by color names, make sure that `pcol` is a character variable.

8.4 Axes, plotting ranges

Setting pl ranges. The regular and inner plotting ranges can be set by specifying `vlim` and `innerrange` as attributes of variables using `setvarattributes` (or directly setting the attributes `attr(dd, "plrange")` or `attr(dd, "innerrange")`), or by giving named lists of vectors of length 2 as arguments of high level pl functions.

The "variable limits" `vlim` can also be set as `xlim` and `ylim` arguments of high level functions.

Set the `innerrange` attribute by calling `genvarattributes`. Otherwise, you need to call also `plcoord` in order to have a conforming `plcoord` attribute of the variable(s). Note that the resulting `innerrange` may differ from the required inner range at the end(s) where no data are modified (`nouter==0`).

Tick marks. The tick mark occur in three "degrees", the first one being labelled, the second (called "regular") being only shown by a mark, the third, by a short mark. (The lengths of the tick marks are determined by the ploption `ticklength`.)

Gridlines, zeroline. If the ploption `grid` is `TRUE`, grid lines will be drawn at each (regular) tick of both axes. Alternatively, the variable attribute `gridline` can be specified either directly or by `setvarattribute` (The plotting properties `col, lty, lwd` are determined by the ploptions `grid.col, grid.lty, grid.lwd`.)

Likewise, lines can be obtained by setting the variable attribute `zeroline` and the respective options. By default, a zero line is drawn at 0 for numerical variables.

Axis label. The plotting routine (`plframe`) will look for an attribute `varlabel` . If it is not available, the name of the variable will be used.

8.5 Smooths

Within the graphical functions, smooths are generated by calling `gensmooth` . It calls the function determined by `ploptions("smooth.function")` , which defaults to `smoothRegr` , and that function calls basic R's `loess` .

The smoothing parameter, which in the default option is the `span` argument of `loess` , depends on the number of observations by the function `smoothpar` (unless set differently by the user).

8.6 Standardized residuals

$$R_i^* = R_i \Big/ \left(\widehat{\sigma}\sqrt{w_i}\sqrt{1-H_{ii}}\right)$$

Standardization ratio: `stratio` $_i = R_i^*/R_i$

`i.stres` calculates leverages, standardized residuals, and `strratio` according to this formula. For binary and Poisson models, ...

Cook's distance:

$$d_i^{(C)} = \frac{R_i^2\, H_{ii}}{p\widehat{\sigma}^2\,(1-H_{ii})^2} = (1/p)\, R_i^{*2}\, H_{ii}/(1-H_{ii})\ ,$$

It is constant, $= d$, on the curve

$$R_i^{*2} = d\, p\, (1-H_{ii})/H_{ii}$$

A rule suggests $d = 4/(n-p)$ as a warning level. Curves are drawn for $d =$ `cookdistlines` $^2/(n-p)$.

8.7 Modified methods for `residuals, fitted, predict`

Some methods for the extractor functions `residuals, fitted, predict` needed modifications and extensions. For example, `type="link"` is needed for `polr` models. The package contains a method `predict.regrpolr` which is used within the package and provides the additional type, but is otherwise compatible with `predict.polr` of the package `MASS` . It is exported. Similarly, there are the methods `fitted.regrpolr, residuals.regrpolr, residuals.regrsurvreg, residuals.regrcoxph` . Since a method for residuals is not available in the `MASS` package for models fitted by `polr` , `residuals.regrpolr` is also made available as `residuals.polr` and thereby provides the missing method.

8.8 Missing values in residual analysis

Model fitting functions in R support two different ways of dealing with missing values. The default is determined by the value `na.omit` of the argument `na.action`. This leads to residuals as extracted by the `residuals` function and other analogous results that do not contain the observations with missing values in the data used for fitting. As a consequence, they cannot be readily combined with variables from the dataset. The alternative choice, `na.action=na.exclude`, corrects this flaw. Vectors of residuals will contain `NA`s where appropriate and have the same length as the variables in the data.

The pl functions `plregr, plresx, plres2x` work with the extended version. This is reflected in `pl.envir$pldata` and allows for combining that data.frame with the original dataset.

8.9 Documentation

For a data analysis, it is often useful to save graphical displays in documents or on paper. In an extended data analysis, one can easily lose control over the precise content of the different plots. `plgraphics` provides some help for keeping track.

- Every graphical display generated by a graphical function from the package gets a "stamp" in the lower right corner that indicates date and time of its creation and, if specified by the user before that time point, a project title and a step name (by writing `ploptions(project=projecttitle, step=stepname)`).
  This stamp can of course be suppressed, e.g., for producing publication graphics: set `ploptions(stamp=FALSE)`.
- Data sets may be documented by attaching two attributes, `tit` and `doc` – title and description –, which will be printed with numerical output if desired. Currently, this only works with the function `showd`. This feature is disabled by setting `ploptions(doc=FALSE)`.

```
tit(iris) <- "Fisher's iris data"
doc(iris) <- "The most famous statistical dataset ever"
showd(iris)
```

```
## Fisher's iris data
##    The most famous statistical dataset ever
## dim:  150 5
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## ...
## 40           5.1         3.4          1.5         0.2     setosa
```

```
## 76          6.6         3.0          4.4         1.4 versicolor
## 113         6.8         3.0          5.5         2.1  virginica
## 150         5.9         3.0          5.1         1.8  virginica
```

This is the end of the story for the time being. I hope that you will get into using `plgraphics` and have good success with your data analyses. Feedback is highly appreciated.

Werner Stahel, `stahel at stat.math.ethz.ch`

```
options(olddigits)
```