Scatterplot3d –
an R package for Visualizing Multivariate Data

Uwe Ligges and Martin Mächler

Fachbereich Statistik
Universität Dortmund
44221 Dortmund
Germany

Seminar für Statistik
ETH Zürich
CH-8092 Zürich
Switzerland

Abstract *Scatterplot3d* is an R package for the visualization of multivariate data in a three dimensional space. R is a "language for data analysis and graphics". In this paper we discuss the features of the package. It is designed by exclusively making use of already existing functions of R and its graphics system and thus shows the extensibility of the R graphics system. Additionally some examples on generated and real world data are provided, as well as the source code and the help page of *scatterplot3d*.

1 Introduction

Scatterplot3d is an R package for the visualization of multivariate data in a three dimensional space. R itself is a "language for data analysis and graphics" (Ihaka and Gentleman, 1996) and a freely available statistical software package implementing that language, see `http://www.R-project.org/`.

Basically *scatterplot3d* generates a scatter plot in the 3D space using a parallel projection. Higher dimensions (fourth, fifth, etc.) of the data can be visualized to some extent using, e.g. different colors, symbol types or symbol sizes.

The following properties of *scatterplot3d* will be further described and discussed in the present paper: A plot is generated entirely by using interpreted R graphics functions, so the appearance of the plot is consistent with other R graphics. Such a behavior is important for publications. Most features of the R graphics system can be applied in *scatterplot3d*, among them are vectorizing of colors or plotting symbols and mathematical annotation (Murrell and Ihaka, 2000). The latter means whole formulas with e.g. greek letters and mathematical symbols inside can be added into plots using a LaTeX like syntax. *Scatterplot3d* can be easily extended e.g., by adding additional points or drawing regression lines or planes into an already generated plot (via function closures, see below). The package is platform independent and can easily be installed, because it only requires an installed version of R.

This paper is structured as follows: In Section 2 the design of *scatterplot3d* will be described, followed by remarks on the extensibility of the function in Section 3. Some examples (including code and results) on generated and real world data are provided in Section 4. We present other R related 3D "tools" in Section 5, followed by the conclusion in Section 6. In the Appendix the source code as well as the help page of *scatterplot3d* are printed.

A colored version of this paper can be downloaded from `http://www.statistik.uni-dortmund.de/sfb475/en/index-e.html`. R and *scatterplot3d* are available from *CRAN* (Common R Archive Network), i.e. `http://CRAN.R-Project.org` or one of its mirrors.

2 Design

Scatterplot3d is designed to plot three dimensional point clouds by exclusive usage of functions in the R base package. Advantages of this "R *code only*" design are the well known generality and extensibility of the R graphics system, the similar behavior of arguments and the similar look and feel with respect to common R graphics, as well as the quality of the graphics, which is extremely important for publications. Drawbacks are the lack of interactivity, and the missing 3D support (2D design).

While the function `persp` for plotting surfaces (cf. Section 5) applies a perspective projection, in *scatterplot3d* a parallel projection for a better comparison of distances between different points is used.

The final implementation of the function and the building of the package was done according to the "R Language definition" and "Writing R Extensions" manuals of the R Development Core Team (in short, '*R core*'), 2002b and 2002c.

2.1 Arguments

The *scatterplot3d* function has been designed to accept as many common arguments to R graphics functions as possible, particularly those mentioned in the help pages of the function `par` and `plot.default` (R core, 2002a). In principle, arguments of `par` with a particular 2D design are replaced by new arguments in *scatterplot3d*. Regularly, values of the corresponding arguments in `par` for the first two dimensions are read out, and *scatterplot3d* either "guesses" the value for the third dimension or has an appropriate default.

A few graphical parameters can only be set as arguments in *scatterplot3d* but not in `par`. For details on which arguments have got a non common default with respect to other R graphics functions see the "Usage" and "Arguments" sections of the help page in appendix C. Other arguments of `par` may be split into several arguments in *scatterplot3d*, e.g. for specification of the line type. Finally, some of the arguments in `par` do not work, e.g. some of those for axis calculation. As common in R, additional arguments that are not mentioned on the help page can be passed through to underlying low level graphics functions by making use of the general '`...`' argument.

2.2 xyz.coords()

As well known from other R functions, vectors x, y and z (for the 3D case) are used to specify the locations of points.

If x has got an appropriate structure, it can be provided as a single argument. In this case, an attempt has to be made to interpret the argument in a way suitable for plotting. For that purpose, we added the function `xyz.coords` (R core, 2002a) into the R base package that accept various combinations of x and optionally y and z arguments. It is a "utility for obtaining consistent x, y and z coordinates and labels for three dimensional plots" (R core, 2002a). Many ideas used in this function are taken from the function `xy.coords` already existing for the 2D case. Even though `xyz.coords` was introduced to support *scatterplot3d*, it is designed to be used by *any* 3D plot functions making use of (x_i, y_i, z_i) triples[1].

If the argument is a formula of type `zvar ~ xvar + yvar` (cf. R core (2002b) for details on formulas), `xvar`, `yvar` and `zvar` are used as x, y and z variables. If the argument is a list with components x, y and z, these are assumed to define plotting coordinates. If it is a matrix with three columns, the first is assumed to contain the x values, etc. Alternatively, two arguments x and y can be be provided, one may be real, the other complex. In any other case, the arguments are coerced to vectors and the values plotted against their indices. If no axis labels are given explicitly, `xyz.coords` attempts to extract appropriate axis labels `xlab`, `ylab` and `zlab` from the above mentioned data structures.

Additionally, color vectors contained in a matrix, data frame or list can be detected by *scatterplot3d* internally.

2.3 Structure

The R code of *scatterplot3d* (Appendix B) is structured into a few parts:
A quite long list of arguments in the first part of the function is followed by some

[1] The functions `persp`, `image` and `contour` are restricted to use a *grid* of x, y values and hence only need n $x-$ and m $y-$ values for $n \times m$ $z-$ values.

plausibility checks, extraction of characters, conversion of data structures (cf. Section 2.2), basic calculations of the angle for displaying the cube, and calculations regarding the data region limits, as well as data sorting for an optional "3D highlighting" feature.

In order to optimize the fit of the data into the plotting region, the second part of the function deals with optimal scaling of the three axis. This yields a high printout quality as well known from regular R graphics, but unfortunately it results also in a static plot, i.e. rotation is not possible. If *scatterplot3d*s with different viewing angles are put together as a "slide show" to imitate a rotation, each of these "slides" is *individually* optimally sized regarding the plotting region, so all in all such a "slide show" will not work.

After the graphics device is initialized in the third part, axis, tick marks, box, grid and labels are added to the plot, if it is required. In the last but one part, the data is plotted and overlayed by the front edges of the box.

Besides the primarily expected result, a drawn plot, four functions are generated and invisibly returned as *Values* in the last part of *scatterplot3d* (cf. Appendix C).

These functions, namely `xyz.convert`, `points3d`, `plane3d` and `box3d`, are required to provide extensibility of the three dimensional plot; details are described in Section 3.

3 Extensibility

Two kinds of extensibilities will be described in this section. On one hand, regarding the *scatterplot3d* design, the extensibility of the R graphics system will be discussed; it provides the tools and features enabling the programmer to write complex high level plot functions in a very general manner. On the other hand we describe the extensibility of *scatterplot3d* itself.

3.1 Extensibility of R graphics

R provides a huge collection of low level graphic functions like those for adding elements to an existing plot or for computations related to plotting. These functions are used to build very general high level functions, at least for the two dimensional case, and without them, the "R code only" design of *scatterplot3d* would be impossible.

A selection of these low level functions begins with the functions to obtain x, y (and z) coordinates for plotting, namely `xy.coords` and `xyz.coords` (for the 3D case, cf. Section 2.2). Further on, the functions `plot.new` and `plot.window` can be used to set up the plotting region appropriately, `pretty` to calculate pretty axis tick marks, `segments` to draw line segments between pairs of points, and functions like `title`, `axis`, `points`, `lines`, `text` etc. are self-explanatory.

A huge collection of graphical parameters for R is documented in the help pages for `par` and `plot.default` (cf. R core (2002a)). Almost all low level graphic functions make use of the argument '`...`' which allows specifying most of these parameters in a very general manner. If this argument, '`...`', is also used in a high level function, arguments which are not *explicitly* introduced in the arguments list, can be passed through to lower level graphic functions as well; this is a powerful feature of the S language.

Since the R graphics system is designed for two dimensional graphics, it lacks of some features for the three dimensional case. Unfortunately, the `axis` function works only for 2D graphics. Consequently a large amount of code (Appendix B) was required to enable oblique axes for displaying the 3D scatter plot in an arbitrary angle.

Locations in R graphics devices can be addressed with 2D coordinates, Thus the information on the projection has to be calculated by the 3D graphic functions internally. As described in Section 2, *scatterplot3d* uses a parallel projection. Since the R graphics device does not know anything about the projection, without any appropriate additional tools it is not possible to add elements into an existing *scatterplot3d*.

3.2 Extensibility of *scatterplot3d*

In Sections 1 and 2 it was emphasized that the *scatterplot3d* design was intended to be as general as possible. Some attempts to obtain this generality are described in Section 2 and its subsections. Because of the missing projection information, the ability of adding elements to an already existing *scatterplot3d* would be restricted, if only the already defined (and for the 2D case general) R functions could be used (cf. Section 3.1).

For this reason, *scatterplot3d* (invisibly) returns a list of *function closures* (cf. Section 2.3). A *function closure* is a function together with an *environment*, and an *environment* is a collection of symbols and associated values (i.e. R variables). Thus these properties of R's scoping rules, called *Lexical Scoping* (Gentleman and Ihaka, 2000), are extensively used in *scatterplot3d*. Notice that *Lexical Scoping* is a feature of R, not defined as such in the S language.

In other words, the values returned by *scatterplot3d* are functions together with the environment in which they (and the scatter plot) were created. The benefit of returning function closures is, that the function somehow "knows" the values of variables (in the environment) that were assigned to those variables at the time when the function was created. All in all, we made those functions know details about the axis scaling and the projection information that are required to add elements to an existing plot appropriately.

The following functions are returned by *scatterplot3d*, for details see Appendix C:

`xyz.convert`: A function which converts 3D coordinates to the 2D parallel projection of the existing *scatterplot3d*. It is useful to add arbitrary elements into the plot.

`points3d`: A function which draws points or lines into the existing plot.

`plane3d`: A function which draws a plane into the existing plot:
`plane3d(Intercept, x.coef=NULL, y.coef=NULL, lty="dashed", ...)`.
Instead of an intercept, a vector containing three elements or an (g)lm object can be specified.

`box3d`: This function draws a box (or "refreshes" an existing one) around the plot.

`xyz.convert` is the most important function, because it does the parallel projection by converting the given 3D coordinates into the 2D coordinates needed for the R graphics devices. Examples how to use the mentioned function closures are given in Section 4.

4 Examples

Many examples presented in this sections are in color. If you have not got a colored version of this paper, you may get it from `http://www.statistik.uni-dortmund.de/sfb475/en/index-e.html`.

4.1 Feature demonstration

In this section some of the features of *scatterplot3d* will be demonstrated using artificially generated data, well known examples from other R functions and the (slightly modified) examples of *scatterplot3d*'s help file (cf. Appendix C). The presentation starts with the latter, each example printed on an individual page to obtain lucidity.

this space intentionally left blank

4.1.1 Helix

In Figure 1 points of a helix are calculated and plotted using the 3D highlighting mode (`highlight.3d = TRUE`) in a blue box with a light blue grid. We produce the solid look with the point symbol, `pch = 20`.

```
z <- seq(-10, 10, 0.01)
x <- cos(z)
y <- sin(z)
scatterplot3d(x, y, z, highlight.3d = TRUE, col.axis = "blue",
              col.grid = "lightblue", main = "Helix", pch = 20)
```

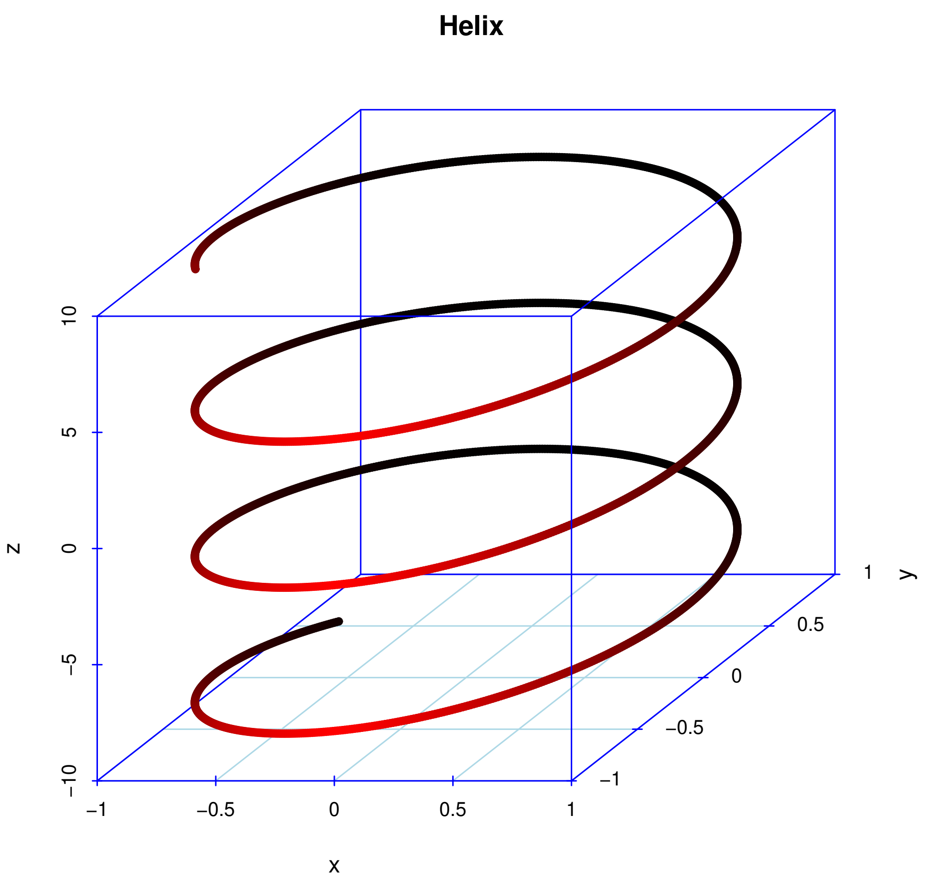

Figure 1: Helix

4.1.2 Hemisphere

Figure 2 shows points on a hemisphere. Except for angle and the size of axes annotation, this figure is generated analogously to Figure 1.

```
temp <- seq(-pi, 0, length = 50)
x <- c(rep(1, 50) %*% t(cos(temp)))
y <- c(cos(temp)  %*% t(sin(temp)))
z <- c(sin(temp)  %*% t(sin(temp)))
scatterplot3d(x, y, z, highlight.3d = TRUE,  angle = 120,
         col.axis = "blue", col.grid = "lightblue", cex.axis = 1.3,
         cex.lab = 1.1, main = "Hemisphere", pch = 20)
```

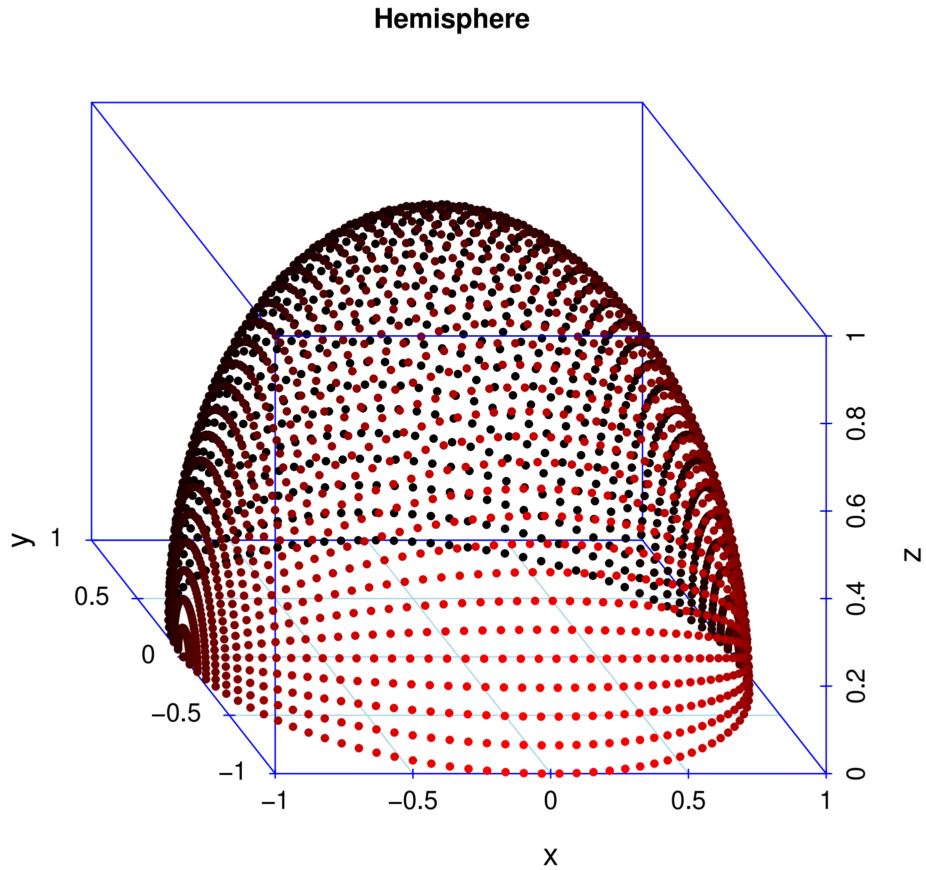

Figure 2: Hemisphere

4.1.3 3D barplot

With some simple modifications, it is possible to generate a 3D barplot, as shown in this example. To make the plot look like a barplot, `type = "h"` is set to draw vertical lines to the x–y plane, `pch = " "` to avoid plotting of point symbols and `lwd = 5` to make the lines looking like bars. Furthermore, instead of three vectors a data frame is given as the first argument to *scatterplot3d*.

```
my.mat <- matrix(runif(25), nrow = 5)
dimnames(my.mat) <- list(LETTERS[1:5], letters[11:15])
s3d.dat <- data.frame(columns = c(col(my.mat)),
                      rows    = c(row(my.mat)), value = c(my.mat))
scatterplot3d(s3d.dat, type = "h", lwd = 5, pch = " ",
      x.ticklabs = colnames(my.mat), y.ticklabs = rownames(my.mat),
      color = grey(25:1 / 40), main = "3D barplot")
```

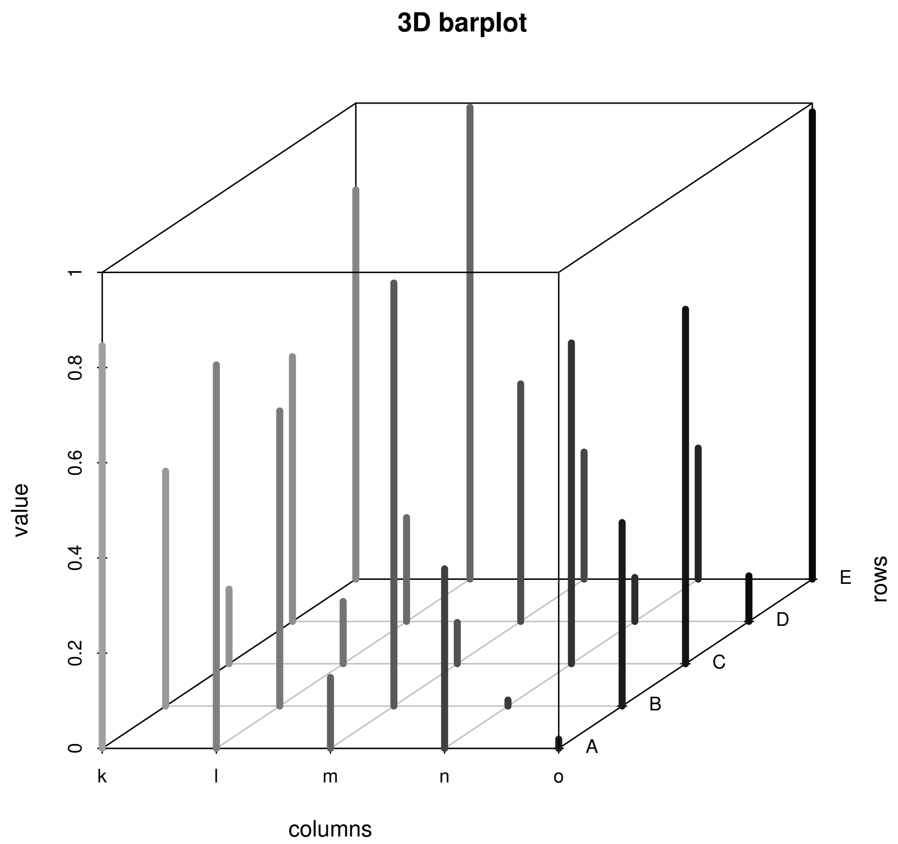

Figure 3: 3D barplot

4.1.4 Adding elements

The importance of *Lexical Scoping* to generate *function closures* to provide extensibility of *scatterplot3d* was discussed in Section 3. An example how to use the invisibly returned functions is given below on the famous (at least for S users) tree data.

After the tree data is loaded, it is plotted by *scatterplot3d*, and the (invisibly returned) result is assigned to the variable `s3d`. The (blue colored) points are plotted using `type = "h"`, so one can see the x–y location of those points very clearly.

In the next step, a linear model (assumption: volume depends on girth and height of the trees) is calculated. Furthermore, this lm object is plotted by the returned `plain3d` function (was assigned to `s3d` before), and it results in a regression plane.

Just for demonstration purposes, in the last step some red colored points (on a imaginary line crossing the plot) are plotted with an asterisk as its point symbol.

```
data(trees)
s3d <- scatterplot3d(trees, type = "h", color = "blue",
    angle = 55, scale.y = 0.7, pch = 16, main = "Adding elements")
my.lm <- lm(trees$Volume ~ trees$Girth + trees$Height)
s3d$plain3d(my.lm)
s3d$points3d(seq(10, 20, 2), seq(85, 60, -5), seq(60, 10, -10),
    col = "red", type = "h", pch = 8)
```

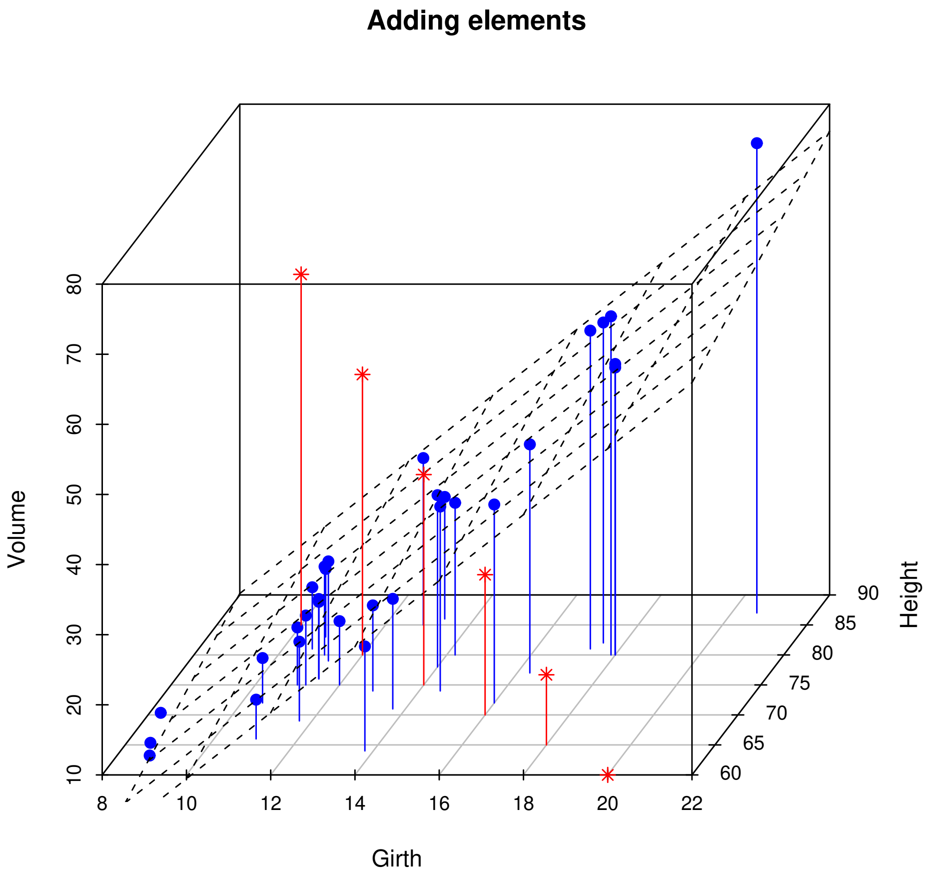

Figure 4: Adding elements

4.1.5 Bivariate normal distribution

In Figure 5 a surface of the density of a bivariate normal distribution is plotted. This example is a bit more sophisticated than the examples before and shows the extensibility of *scatterplot3d*. Note that *scatterplot3d* is designed to generate scatter plots, not to draw surfaces, is not really user friendly for this purpose, for which we'd typically rather use R's `persp` function.

In a first step a matrix containing the density is calculated. The call of *scatterplot3d* sets up the plot (axes, labels, etc.), but doesn't draw the surface itself which is accomplished by the two loops at the end of the code. Additionally, we give an example of quite sophisticated mathematical annotation.

```
library(mvtnorm)
x1 <- x2 <- seq(-10, 10, length = 51)
dens <- matrix(dmvnorm(expand.grid(x1, x2),
                  sigma = rbind(c(3, 2), c(2, 3))),
               ncol = length(x1))
s3d <- scatterplot3d(x1, x2,
                  seq(min(dens), max(dens), length = length(x1)),
                  type = "n", grid = FALSE, angle = 70,
                  zlab = expression(f(x[1], x[2])),
                  xlab = expression(x[1]), ylab = expression(x[2]),
                  main = "Bivariate normal distribution")
text(s3d$xyz.convert(-1, 10, 0.07),
      labels = expression(f(x) == frac(1, sqrt((2 * pi)^n *
          phantom(".") * det(Sigma[X]))) * phantom(".") * exp{
          bgroup("(", - scriptstyle(frac(1, 2) * phantom(".")) *
              (x - mu)^T * Sigma[X]^-1 * (x - mu), ")")}))
text(s3d$xyz.convert(1.5, 10, 0.05),
      labels = expression("with" * phantom("m") *
          mu == bgroup("(", atop(0, 0), ")") * phantom(".") * "," *
              phantom(0) *
          Sigma[X] ==  bgroup("(", atop(3 * phantom(0) * 2,
              2 * phantom(0) * 3), ")")))
for(i in length(x1):1)
    s3d$points3d(rep(x1[i], length(x2)), x2, dens[i,], type = "l")
for(i in length(x2):1)
    s3d$points3d(x1, rep(x2[i], length(x1)), dens[,i], type = "l")
```

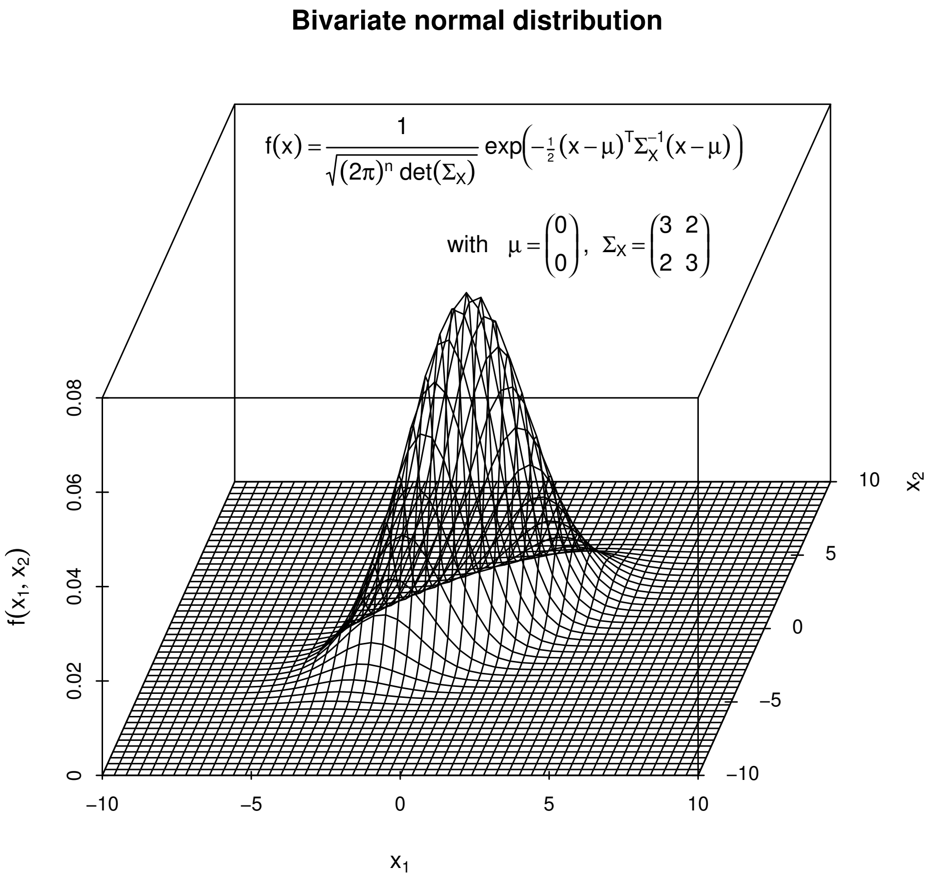

Figure 5: Density of a bivariate normal distribution

4.2 Real world examples

Three real world examples are presented in this section. The data are from the following recent projects of the collaborative research centre 475 (Deutsche Forschungsgemeinschaft, SFB 475: "Reduction of complexity in multivariate data structures"):

C3 (Biometrics) Meta–Analysis in Biometry and Epidemiology,

B3 (Econometrics) Multivariate Analysis of Business Cycles, and

C5 (Technometrics) Analysis and Modelling of the Deephole–Drilling–Process with Methods of Statistics and Neuronal Networks.

4.2.1 Meta–analysis of controlled clinical trials

In the first real world example the data from a project on "Meta–Analysis in Biometry and Epidemiology" is taken. The data set contains the results of 13 placebo–controlled clinical trials which evaluated the efficacy of the Bacillus Calmette–Guérin (BCG) vaccine for the prevention of tuberculosis (TB). An important task in combining the results of clinical trials is to detect possible sources of heterogeneity which may influence the true treatment effect. In the present example, a possible influential covariate is the distance of each trial from the equator, which may serve as a surrogate for the presence of environmental mycobacteria that provide a certain level of natural immunity against TB. Other covariates may be the year the trial was carried out and the allocation scheme of the vaccination (A = alternate, R = random, S = systematic). For more details, especially on the choice of the trials and the meta–analytical methods of combining the results, we refer to Knapp and Hartung (2002) and the references given therein.

In Figure 6 the estimated risks of TB disease are plotted for the vaccinated group and the non–vaccinated group, respectively, in the dependence of the year the trial was carried out, of the absolute distance from the equator and of the allocation scheme. The color represents the precisions of the estimated risks. Figure 6 clearly reveals a spatio–temporal trend in the realization of the trials. The former trials were carried out far away from the equator, and in all these trials one can observe an evident superiority of the BCG vaccine for the prevention of TB. Except one trial all the other later trials were realized closer to the equator. In these trials, it is apparently that the estimated risks in the non–vaccinated groups are even rather low and, consequently, cannot graphically separated from the estimated risks in the vaccinated groups. Finally, it is worthwhile to note that the later trial which was carried out far away from the equator has a relative small estimated risk in the non–vaccinated group compared to the former trials and, hence, does not yield such an evident superiority of the BCG vaccine.

Three variables are represented by the three dimensions of the cube, while variable "Precision" is represented by color. To realize color representation for metric variables, some manual tuning is necessary, though.

```
layout(cbind(1:2, 1:2), heights = c(7, 1))
prc <- hsv((prc <- 0.7 * Prec / diff(range(Prec))) - min(prc) + 0.3)
s3d <- scatterplot3d(Year, Latitude, Risk, mar = c(5, 3, 4, 3),
    type = "h", pch = " ", main = "Estimated TB risks")
s3d$points(Year, Latitude, Risk, pch = ifelse(vac, 22, 21), bg = prc,
    cex = ifelse(vac, 2, 1.5))
s3d.coords <- s3d$xyz.convert(Year, Latitude, Risk)
al.char <- toupper(substr(as.character(Allocation), 1, 1))
text(s3d.coords$x[!vac], s3d.coords$y[!vac], labels = al.char[!vac],
    pos = 2, offset = 0.5)
legend(s3d$xyz.convert(80, 15, 0.21), pch = c("A", "R", "S"), yjust=0,
    legend = c("alternate", "random", "systematic"), cex = 1.1)
legend(s3d$xyz.convert(47, 60, 0.24), pch = 22:21, yjust = 0,
    legend = c("vaccinated", "not vaccinated"), cex = 1.1)

par(mar=c(5, 3, 0, 3))
plot(seq(min(Prec), max(Prec), length = 100), rep(0, 100), pch = 15,
    axes = FALSE, xlab = "color code of variable \"Precision\"",
    ylab = "", col = hsv(seq(0.3, 1, length = 100)))
axis(1, at = 4:7, labels = expression(10^4, 10^5, 10^6, 10^7))
```

Two kinds of point symbols stand for the "Vaccinated" variable, and for a sixth variable, "Allocation", an appropriate letter is printed additionally close to the "not vaccinated" symbol.

For each of the latter three variables a legend is desirable. Thus the smaller two legends are plotted into the *scatterplot3d*, while the legend for the color coding gets a single plot. The function `layout` arranges the two plots suitably on the same device.

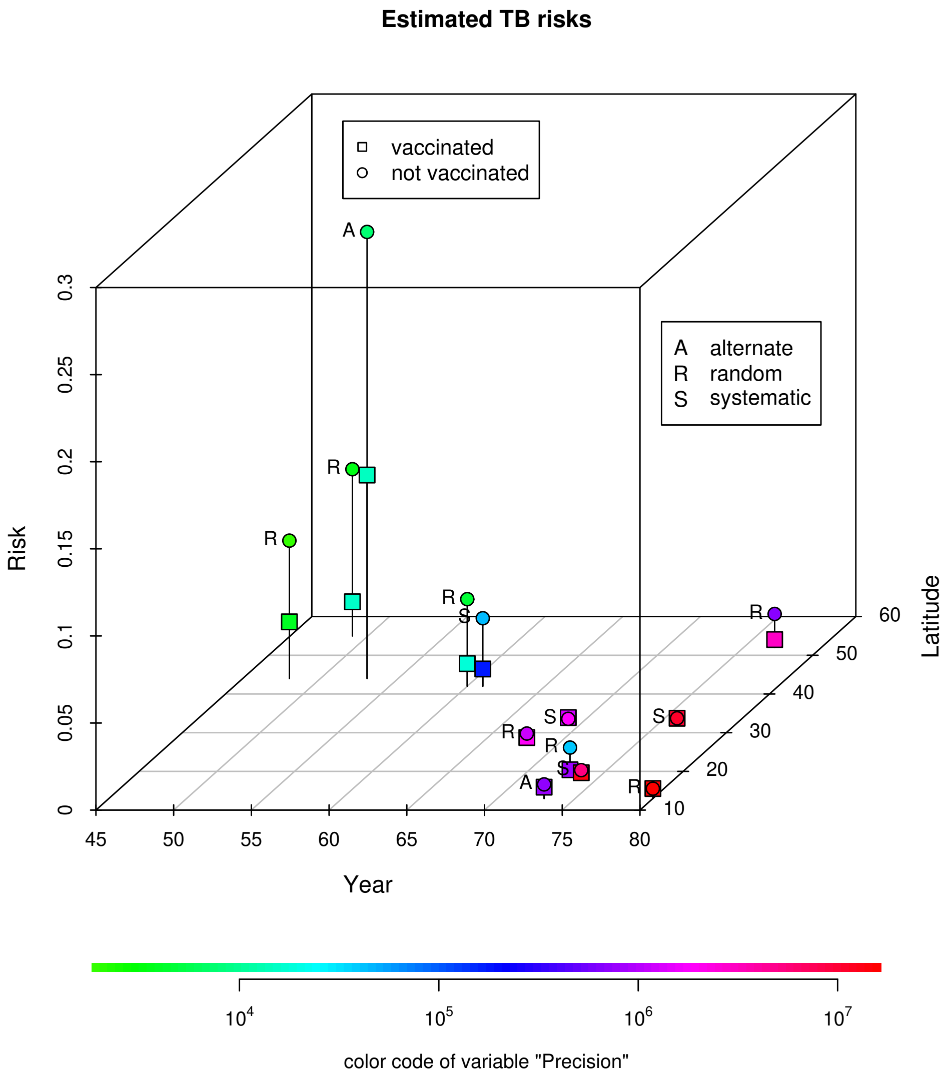

Figure 6: Estimated TB risks

4.2.2 Business cycle data

The example in this section shows the plotting of data from a project on "Multivariate Analysis of Business Cycles". One of the main interests of the project is the prediction of business cycle phases. An extraction of available relevant (concerning the purposes of this section) variables and its abbreviations is given in Table 1. The abbreviation 'gr' stands for growth rates with respect to last year's corresponding quarter.

abbr	description
IE	real investment in equipment (gr)
C	real private consumption (gr)
Y	real gross national product (gr)
L	wage and salary earners (gr)

Table 1: Abbreviations

The experts' classification of the data into business cycle phases ("PH") was done by Heilemann and Münch (1996) using a 4-phase scheme. These phases are called *lower turning points*, *upswing*, *upper turning points*, and *downswing*.

In Figure 7 the three variables C, Y, and L are represented by the three dimensions of the cube. The variable IE is represented by color, while four different point symbols stand for the four business cycle phases (PH).

For each of the latter two variables, a legend is desirable. Thus the smaller one (for PH) is plotted into the *scatterplot3d*, while the legend for the color coding of IE got a single plot, analogously to the example in Section 4.2.1.

A regression plane is added to the plot to support the visual impression. Obviously all the plotted variables are highly correlated, with the exception of the class variable which does not appear to be well predictable by the other variables. Details are discussed in Theis et al. (1999). In order to provide a correct impression of the fit, the residuals, i.e. the projection lines to the plane, are drawn in Figure 8 where different color and line types are used for positive and negative residuals respectively.

```
layout(cbind(1:2, 1:2), heights = c(7, 1))
temp <- hsv((temp <- 0.7 * IE / diff(range(IE))) - min(temp) + 0.3)
s3d <- scatterplot3d(L, C, Y, pch = Phase, color = temp,
    mar = c(5, 3, 4, 3), main = "Business cycle phases")
legend(s3d$xyz.convert(-2, 0, 16), pch = 1:4, yjust = 0,
    legend = c("upswing", "upper turning points",
        "downswing", "lower turning points"))
s3d$plain3d(my.lm <- lm(Y ~ L + C), lty = "dotted")
par(mar=c(5, 3, 0, 3))
plot(seq(min(IE), max(IE), length = 100), rep(0, 100), pch = 15,
    axes = FALSE, xlab = "color code of variable \"IE\"", ylab = "",
    col = hsv(seq(0.3, 1, length = 100)))
axis(1, at = seq(-20, 25, 5))
```

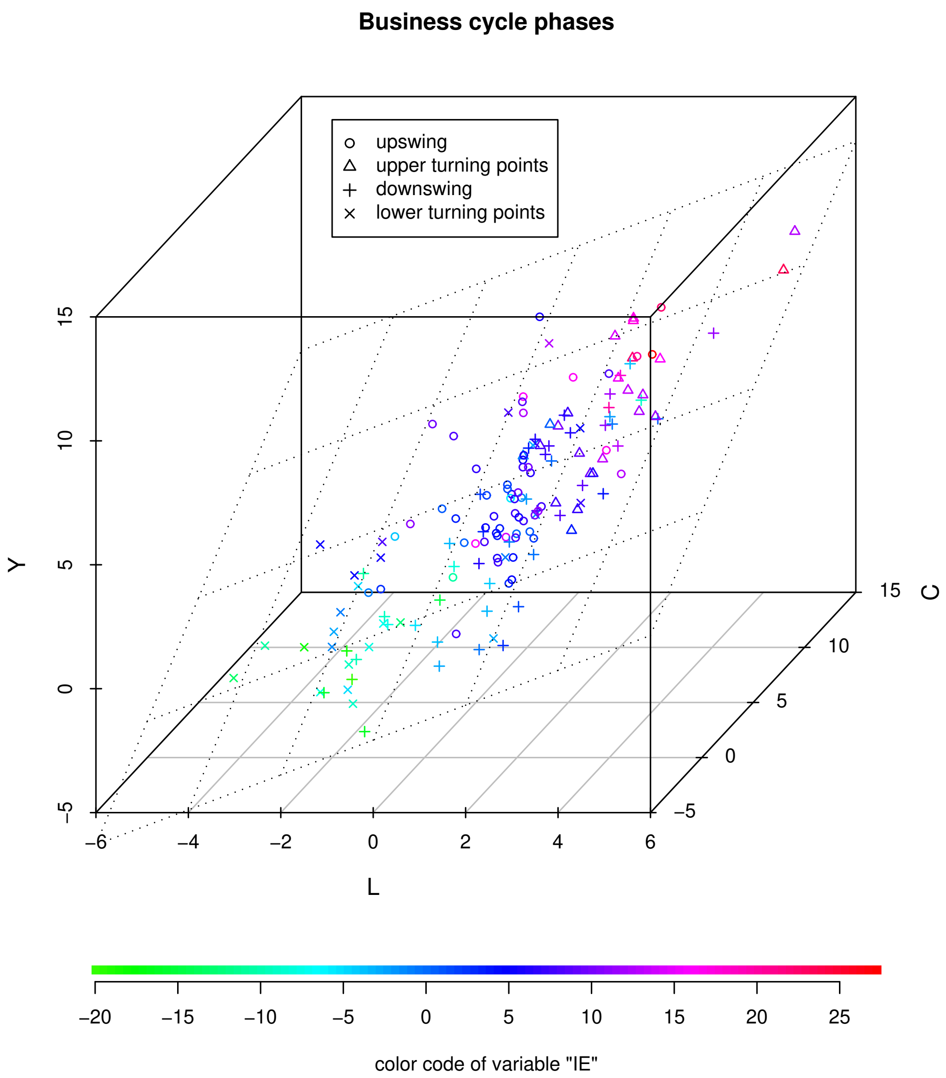

Figure 7: Business cycle phases

```
s3d <- scatterplot3d(L, C, Y, pch = 20, mar = c(5, 3, 4, 3),
          main = "Residuals")
s3d$plain3d(my.lm, lty = "dotted")
orig <- s3d$xyz.convert(L, C, Y)
plane <- s3d$xyz.convert(L, C,  fitted(my.lm))
i.negpos <- 1 + (resid(my.lm) > 0)
segments(orig$x, orig$y, plane$x, plane$y,
          col = c("blue", "red")[i.negpos], lty = (2:1)[i.negpos])
```

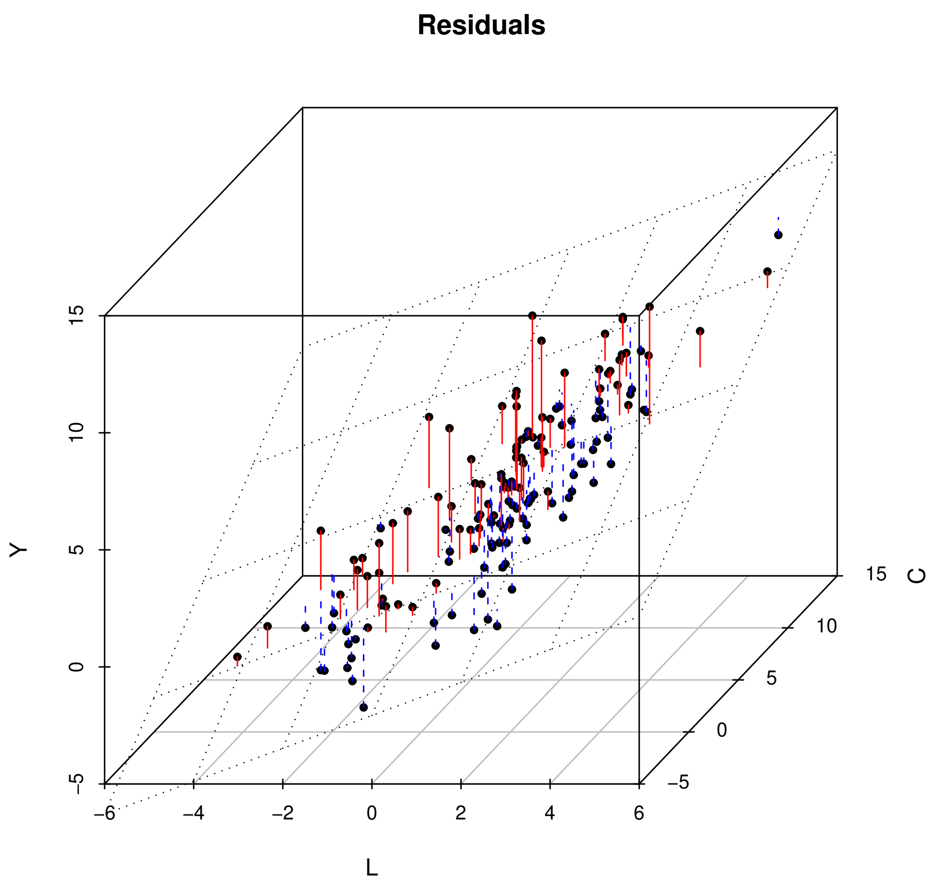

Figure 8: Residuals (cf. Figure 7)

4.2.3 Deep hole drilling

Our last real world example shows phase spaces (Tong, 1993) of the drilling torque of a deep hole drilling process. The data is taken from a project on ”Analysis and Modelling of the Deephole–Drilling–Process with Methods of Statistics and Neuronal Networks”. More detailed analysis on the data than provided in the following example was done by, e.g., Busse et al. (2001) and Weinert et al. (2001).

Figure 9 visualizes the phase spaces of the drilling torques of two deep hole drilling processes, a regular and a chattering one. Obviously the points in the phase space of the chattering process are very systematically scattered, and the range of the data is very different for the two processes. The magnification of the regular process in Figure 10 shows that the points of the regular process are scattered unsystematically. Note that other lags like 10, 20, 100 would produce a similar plot. This indicates a sine wave like relationship in the chattering case.

```
s3d <- scatterplot3d(drill1[1:400], drill1[7:406], drill1[32:431],
    color = "red", type = "l", angle = 120, xlab = "drilling torque",
    ylab = "drilling torque, lag 6", zlab = "drilling torque, lag 31",
    main = "Two deep hole drilling processes")
s3d$points3d(drill2[1:400], drill2[7:406], drill2[32:431],
    col = "blue", type = "l")
legend(s3d$xyz.convert(-400, 1000, 950), col= c("blue", "red"),
    legend = c("regular process", "chattering process"), lwd = 2,
    bg = "white")

scatterplot3d(drill2[1:400], drill2[7:406], drill2[32:431],
    color = "blue", type = "l", angle = 120, xlab = "drilling torque",
    ylab = "drilling torque, lag 6", zlab = "drilling torque, lag 31",
    main = "Magnification of the regular process")
```

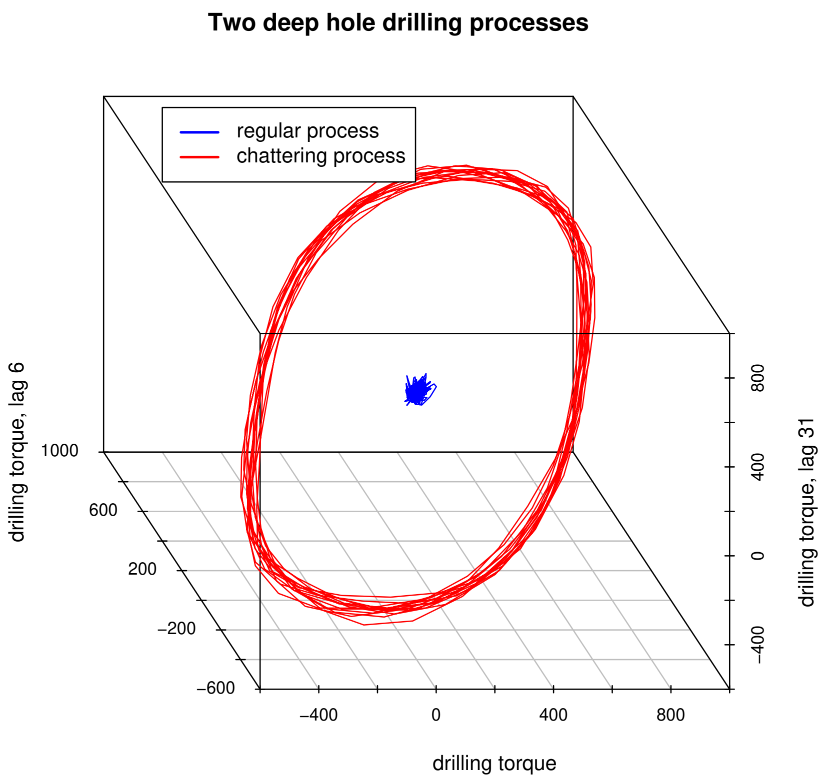

Figure 9: Phase spaces of the drilling torques of two deep hole drilling processes

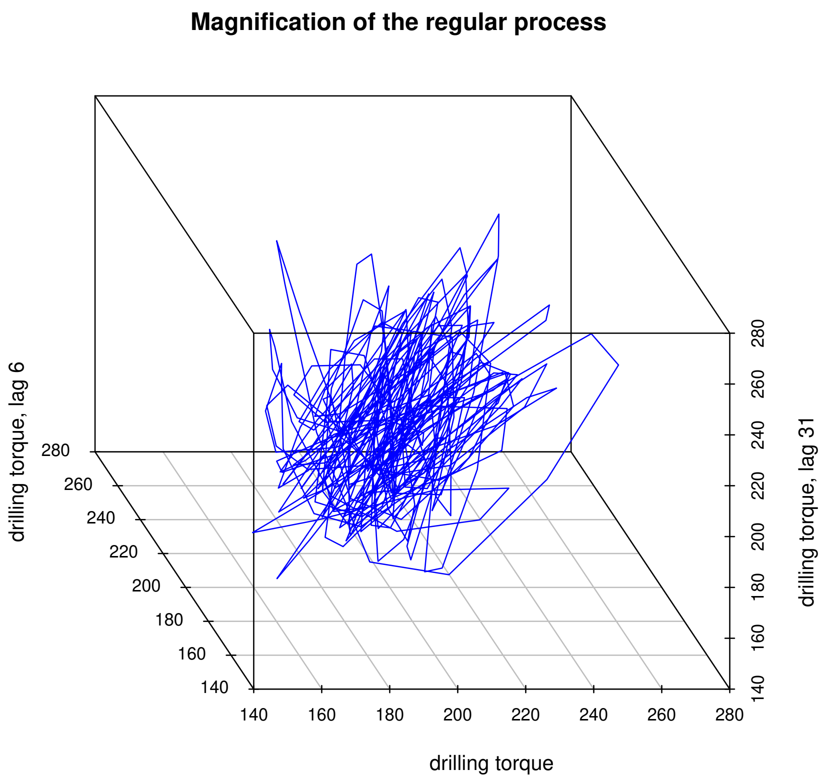

Figure 10: Magnification of the regular process (Figure 9)

5 Other 3D tools in R

At the time of writing *scatterplot3d*, the function `persp()` in the base package of R for three dimensional surface plots was available, but there was no way to generate 3D scatter plots in R itself.

The data visualization system *xgobi* (Swayne et al., 1998) provides interactive visualization of multidimensional data, e.g. brush and spin, higher-dimensional rotation, grand tour, etc. The R package *xgobi* (Swayne et al., 1991; we have to distinguish the visualization system and the package) provides an Interface to *xgobi* and launches a *xgobi* process appropriately. *ggobi* (Swayne et al., 2002) is the next edition of *xgobi* and available at `http://www.ggobi.org`.

Analogously to *xgobi* a R package *Rggobi* (Temple Lang and Swayne, 2001) exists in the *Omegahat* project (Temple Lang, 2000) `http://www.omegahat.org` that allows one to embed *ggobi* within R and to both set and query the *ggobi* contents. All in all, *ggobi* can be loaded dynamically into R (as well as into other software products, in principle), and R into *ggobi*. This provides interactive, direct manipulation, linked, high-dimensional graphics within R.

The package *RGL* (Murdoch, 2001) provides an R interface to *OpenGL*. A huge collection of useful functions to generate, manipulate and interactively rotate 3D objects is available. Unfortunately, at the time of writing the package is only available for the Windows operating system.

Very recently, the function `cloud` in the lattice package has been introduced. It is a 3D scatter plot function that works in the *lattice* (Sarkar, 2002) (and *grid* (Murrell, 2001)) environment of R, but it is still under development (as well as parts of *lattice* and *grid*) and therefore contains some bugs and lacks of some features (compare the help pages for details). One of these features, mathematical annotation, will be added presumably in R–1.6.0. *Lattice* is an implementation of *Trellis Graphics*, which is a framework for data visualization developed at the Bell Labs by Becker et al. (1996), extending ideas presented in Cleveland (1993).

6 Conclusion

In the design (Section 2) of the scatter plot function *scatterplot3d* emphasis is placed on generality and extensibility (Section 3). These two properties are demonstrated in Section 4, as well as the high printout quality. A high printout quality and a homogeneous appearance with respect of any other R (2D) graphics is extremely important for publications and presentations. Thus we recommend to use *scatterplot3d* particularly for these purposes.

Other R related 3D "tools" (Section 5) are focused on different properties, such as surface plotting (e.g. function `persp`), interactivity and online analysis (e.g. *ggobi* or *RGL*).

Acknowledgements. The financial support of the Deutsche Forschungsgemeinschaft (SFB 475, "Reduction of complexity in multivariate data structures") is gratefully acknowledged.

We express our sincere thanks to the following people (in alphabetical order) for their extensive comments on the features and bugs during the time of development, as well as for the discussion of the example data:
Ben Bolker, Anja Busse, Ursula Garczarek, Joachim Hartung, Guido Knapp, Winfried Theis, Brigitta Voß, and Claus Weihs.

References

Becker, R. A., W. S. Cleveland, and M. Shyu (1996). The visual design and control of trellis display. *Journal of Computational and Graphical Statistics 5*(2), 123–155.

Busse, A. M., M. Hüsken, and P. Stagge (2001). Offline-Analyse eines BTA-Tiefbohrprozesses. Technical Report 16/2001, SFB 475, Department of Statistics, University of Dortmund, Germany. See also: `http://www.statistik.uni-dortmund.de/sfb475/en/tr-e.html`.

Cleveland, W. S. (1993). *Visualizing Data*. Summit, NJ: Hobart Press.

Gentleman, R. and R. Ihaka (2000). Lexical Scope and Statistical Computing. *Journal of Computational and Graphical Statistics 9*(3), 491–508.

Heilemann, U. and H. J. Münch (1996). West German Business Cycles 1963–1994: A Multivariate Discriminant Analysis. In *CIRET–Conference in Singapore, CIRET–Studien 50*.

Ihaka, R. and R. Gentleman (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics 5*(3), 299–314.

Knapp, G. and J. Hartung (2002). Improved tests for random effects meta-regression with a single covariate. *Statistics in Medicine*. revised version submitted.

Murdoch, D. (2001). RGL: An R Interface to OpenGL. In K. Hornik and F. Leisch (Eds.), *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15–17*, Vienna. Technische Universität Wien. ISSN 1609-395X, `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/`.

Murrell, P. (2001). R Lattice Graphics. In K. Hornik and F. Leisch (Eds.), *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15–17*, Vienna. Technische Universität Wien. ISSN 1609-395X, `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/`.

Murrell, P. and R. Ihaka (2000). An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics 9*(3), 582–599.

R Development Core Team (2002a). *The R Environment for Statistical Computing and Graphics,* Version 1.5.0. R–Project. ISBN 3-901167-50-1, `http://CRAN.R-project.org/manuals.html`.

R Development Core Team (2002b). *R Language Definition,* Version 1.5.0. R–Project. ISBN 3-901167-56-0, `http://CRAN.R-project.org/manuals.html`.

R Development Core Team (2002c). *Writing R Extensions,* Version 1.5.0. R–Project. ISBN 3-901167-54-4, `http://CRAN.R-project.org/manuals.html`.

Sarkar, D. (2002). Lattice: An implementation of Trellis graphics in R. *R News 2*(2). ISSN 1609-3631, `http://CRAN.R-project.org/doc/Rnews/`, to be published.

Swayne, D. F., A. Buja, and N. Hubbell (1991). XGobi meets S: Integrating software for data analysis. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, Fairfax Station, VA, pp. 430–434. Interface Foundation of North America, Inc.

Swayne, D. F., D. Cook, and A. Buja (1998). Xgobi: Interactive dynamic graphics in the X window system. *Journal of Computational and Graphical Statistics 7*(1), 113–130. See also `http://www.research.att.com/areas/stat/xgobi/`.

Swayne, D. F., D. Temple Lang, A. Buja, and D. Cook (2002). GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Journal of Computational and Graphical Statistics*. (To appear).

Temple Lang, D. (2000). The Omegahat Environment: New Possibilities for Statistical Computing. *Journal of Computational and Graphical Statistics 9*(3), 423–451.

Temple Lang, D. and D. F. Swayne (2001). GGobi meets R: an extensible environment for interactive dynamic data visualization. In K. Hornik and F. Leisch (Eds.), *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, Vienna. Technische Universität Wien. ISSN 1609-395X, `http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/`.

Theis, W., K. Vogtländer, and C. Weihs (1999). Descriptive studies on stylized facts of the german business cycle. Technical Report 45/1999, SFB

475, Department of Statistics, University of Dortmund, Germany. See also: `http://www.statistik.uni-dortmund.de/sfb475/en/tr-e.html`.

Tong, H. (1993). *Non-linear Time Series, A Dynamical System Approach.* Oxford Statistical Science Series. New York: Oxford University Press.

Weinert, K., O. Webber, A. M. Busse, M. Hüsken, J. Mehnen, and S. P. (2001). In die Tiefe: Koordinierter Einsatz von Sensorik und Statistik zur Analyse und Modellierung von BTA-Tiefbohrprozessen. *Spur, G. (ed.): ZWF, Zeitschrift für wirtschaftlichen Fabrikbetrieb 5*, 299–314.

Appendix

A Information files

A.1 Description

```
Package: scatterplot3d
Version: 0.3-11
Title: 3D Scatter Plot
Author: Uwe Ligges <ligges@statistik.uni-dortmund.de>
Maintainer: Uwe Ligges <ligges@statistik.uni-dortmund.de>
Description: Plots a three dimensional (3D) point cloud.
Depends: R (>= 1.1.0)
License: GPL (version 2)
```

A.2 Changes

```
Changes in 0.3-x releases of scatterplot3d:
=======
 0.3.0:  New design: box, pretty() for ticks, ...
 0.3.1:  par("las") bug patched, scale.y is changed (code and default)
 0.3.2:  all angles will work again (default: 40)
         tick mark labeling changed (using mtext)
         par("mar") is set in the first line, not very general!
 0.3.3:  new argument ‘‘mar’’, more details in the help files
 0.3.4:  new arguments x/y/z.ticklabs, thanks to Ben Bolker!
         bug fix: adj for tick.mark.labels corrected
 0.3-5:  new argument y.margin.add for manual fixing scaling problems
         (e.g. some y-tickmarks dissapear after rescaling the window)
 0.3-6:  cex.symbols introduced to solve magnification errors
 0.3-7:  added function plane3d, which will be returned,
         (e.g. for overlaying a regression plane)
 0.3-8:  bugfix: some magnification errors for y.ticklabs
 0.3-9:  bugfix: pch works vectorized again (error with y-sorting)
 0.3-10: added function box3d(), which will be returned,
         to draw the box surrounding the plot again after additions
         * added a function s3d.persp() - somehow joining s3d and persp() *
 0.3-11: * s3d.persp() deleted again, because of various reasons *
         Created this file to reduce the size of the R code.

known UNfixed bug: xlim, ylim, zlim don’t work *exactly* for enlarged areas
```

B Code (scatterplot3d.R)

```
scatterplot3d <- function(x, y = NULL, z = NULL, color = par("col"),
  pch = NULL, main = NULL, sub = NULL, xlim = NULL, ylim = NULL,
  zlim = NULL, xlab = NULL, ylab = NULL, zlab = NULL, scale.y = 1,
  angle = 40, axis = TRUE, tick.marks = TRUE, label.tick.marks = TRUE,
  x.ticklabs = NULL, y.ticklabs = NULL, z.ticklabs = NULL,
  y.margin.add = 0, grid = TRUE, box = TRUE, lab = par("lab"),
  lab.z = mean(lab[1:2]), type = par("type"), highlight.3d = FALSE,
  mar = c(5, 3, 4, 3) + 0.1, col.axis = par("col.axis"), col.grid= "grey",
  col.lab= par("col.lab"), cex.symbols = par("cex"),
  cex.axis = par("cex.axis"), cex.lab = 0.8 * par("cex.lab"),
  font.axis = par("font.axis"), font.lab = par("font.lab"),
  lty.axis = par("lty"), lty.grid = par("lty"), log = "", ...)
  # log not yet implemented
{
  ## scatterplot3d, 0.3-11, 24.05.2002,
  ## Uwe Ligges <ligges@statistik.uni-dortmund.de>,
  ## http://www.statistik.uni-dortmund.de/leute/ligges.htm
  ##
  ## For MANY ideas and improvements thanks to Martin Maechler!!!
  ## Parts of the help files are stolen from the standard plotting functions in R.

  mem.par <- par(mar = mar)
  x.scal <- y.scal <- z.scal <- 1
  xlabel <- if (!missing(x)) deparse(substitute(x))
  ylabel <- if (!missing(y)) deparse(substitute(y))
  zlabel <- if (!missing(z)) deparse(substitute(z))
  ## verification, init, ...
  if(highlight.3d && !missing(color))
    warning(message = "color is ignored when  highlight.3d = TRUE")
  if(length(x) < 2) stop("Minimal required length of x is 2 !")

  ## color as part of 'x' (data.frame or list):
  if(!is.null(d <- dim(x)) && (length(d) == 2) && (d[2] >= 4))
    color <- x[,4]
  else if(is.list(x) && !is.null(x$color))
    color <- x$color

  ## convert 'anything' -> vector
  xyz <- xyz.coords(x = x, y = y, z = z, xlab = xlabel, ylab = ylabel,
    zlab=zlabel, log=log)
  if(is.null(xlab)) { xlab <- xyz$xlab; if(is.null(xlab)) xlab <- "" }
  if(is.null(ylab)) { ylab <- xyz$ylab; if(is.null(ylab)) ylab <- "" }
  if(is.null(zlab)) { zlab <- xyz$zlab; if(is.null(zlab)) zlab <- "" }
```

```
if(length(color) == 1)
  color <- rep(color, length(xyz$x))
else if(length(color) != length(xyz$x))
  stop("length(color) must be equal length(x) or 1 !")

angle <- (angle %% 360) / 90
yz.f <- scale.y * abs(if(angle < 1) angle else
  if(angle > 3) angle - 4 else 2 - angle)
yx.f <- scale.y * (if(angle < 2) 1 - angle else angle - 3)
if(angle > 2) { ## switch y and x axis to ensure righthand oriented coord.
  temp <- xyz$x; xyz$x <- xyz$y; xyz$y <- temp
  temp <- xlab; xlab <- ylab; ylab <- temp
  temp <- xlim; xlim <- ylim; ylim <- temp
}
angle.1 <- ifelse((2 > angle && angle > 1) || angle > 3, TRUE, FALSE)
angle.2 <- ifelse(1 > angle || angle > 3, FALSE, TRUE)
dat <- cbind(as.data.frame(xyz[c("x","y","z")]), col = color)

## xlim, ylim, zlim -- select the points inside the limits
if(!is.null(xlim)) {
  xlim <- range(xlim)
  dat <- dat[ xlim[1] <= dat$x & dat$x <= xlim[2] , , drop = FALSE]
}
if(!is.null(ylim)) {
  ylim <- range(ylim)
  dat <- dat[ ylim[1] <= dat$y & dat$y <= ylim[2] , , drop = FALSE]
}
if(!is.null(zlim)) {
  zlim <- range(zlim)
  dat <- dat[ zlim[1] <= dat$z & dat$z <= zlim[2] , , drop = FALSE]
}
n <- nrow(dat)
if(n < 1) stop("No data left within (x|y|z)lim")

y.range <- range(dat$y[is.finite(dat$y)])
if(all(diff(y.range) == 0))
  stop("All points have the same Y-value! Use 2D-plot!")

### 3D-highlighting / colors / sort by y
if(type == "p" || type == "h") {
  y.ord <- rev(order(dat$y))
  dat <- dat[y.ord, ]
  if(length(pch) > 1)
      if(length(pch) != length(y.ord))
          stop("length(pch) must be equal length(x) or 1 !")
```

```
    else pch <- pch[y.ord]
  if(highlight.3d)
    dat$col <- rgb((1:n / n) * (y.range[2] - dat$y) / diff(y.range),
      g = 0, b = 0)
}

### optim. axis scaling
p.lab <- par("lab")
## Y
y.range <- range(dat$y, ylim)
y.prty <- pretty(y.range, n = lab[2],
  min.n = max(1, min(.5 * lab[2], p.lab[2])))
y.scal <- round(diff(y.prty[1:2]), digits = 12)
y.add <- min(y.prty)
dat$y <- (dat$y - y.add) / y.scal
y.max <- (max(y.prty) - y.add) / y.scal
if(!is.null(ylim))
  y.max <- max(y.max, ceiling((ylim[2] - y.add) / y.scal))
if(angle > 2)
  dat$y <- y.max - dat$y ## turn y-values around
## X
x.range <- range(dat$x[is.finite(dat$x)], xlim)
if(all(diff(x.range) == 0))
  stop("All points have the same X-value! Use 2D-plot!")
x.prty <- pretty(x.range, n = lab[1],
  min.n = max(1, min(.5 * lab[1], p.lab[1])))
x.scal <- round(diff(x.prty[1:2]), digits = 12)
dat$x <- dat$x / x.scal
x.range <- range(x.prty) / x.scal
x.max <- ceiling(x.range[2])
x.min <-  floor(x.range[1])
if(!is.null(xlim)) {
  x.max <- max(x.max, ceiling(xlim[2] / x.scal))
  x.min <- min(x.min, floor(xlim[1] / x.scal))
}
x.range <- range(x.min, x.max)
## Z
z.range <- range(dat$z[is.finite(dat$z)], zlim)
if(all(diff(z.range) == 0))
  stop("All points have the same Z-value! Use 2D-plot!")
z.prty <- pretty(z.range, n = lab.z,
  min.n = max(1, min(.5 * lab.z, p.lab[2])))
z.scal <- round(diff(z.prty[1:2]), digits = 12)
dat$z <- dat$z / z.scal
z.range <- range(z.prty) / z.scal
z.max <- ceiling(z.range[2])
```

```
z.min <-  floor(z.range[1])
if(!is.null(zlim)) {
  z.max <- max(z.max, ceiling(zlim[2] / z.scal))
  z.min <- min(z.min, floor(zlim[1] / z.scal))
}
z.range <- range(z.min, z.max)

### init graphics
plot.new()
if(angle.2) {x1 <- x.min + yx.f * y.max; x2 <- x.max}
else {x1 <- x.min; x2 <- x.max + yx.f * y.max}
plot.window(c(x1, x2), c(z.min, z.max + yz.f * y.max))
temp <- strwidth(as.character(y.scal * y.max + round(y.add, 0)),
    cex = cex.lab/par("cex"))
if(angle.2) x1 <- x1 - temp - y.margin.add
else x2 <- x2 + temp + y.margin.add
plot.window(c(x1, x2), c(z.min, z.max + yz.f * y.max))
if(angle > 2) par("usr" = par("usr")[c(2, 1, 3:4)])
title(main, sub, ...)

### draw axis, tick marks, labels , grid , ...
if(grid) {
  ## X
  i <- x.min:x.max
  segments(i, z.min, i + (yx.f * y.max), yz.f * y.max + z.min,
      col = col.grid, lty = lty.grid)
  ## Y
  i <- 0:y.max
  segments(x.min + (i * yx.f), i * yz.f + z.min,
      x.max + (i * yx.f), i * yz.f + z.min,
      col = col.grid, lty = lty.grid)
}
if(tick.marks && axis) { ## tick marks
  xtl <- (z.max - z.min) * (tcl <- -par("tcl")) / 50
  ztl <- (x.max - x.min) * tcl / 50
  ## Y
  i <- 0:y.max
  temp <- ifelse(angle.2, x.min, x.max)
  segments(yx.f * i - ztl + temp, yz.f * i + z.min,
      yx.f * i + ztl + temp, yz.f * i + z.min,
      col=col.axis, lty=lty.axis)
  ## X
  i <- x.min:x.max
  segments(i, -xtl + z.min, i, xtl + z.min, col=col.axis, lty=lty.axis)
```

```
  ## Z
  i <- z.min:z.max
  temp <- ifelse(angle.2, x.max, x.min)
  segments(-ztl + temp, i, ztl + temp, i, col=col.axis, lty=lty.axis)

  if(label.tick.marks) { ## label tick marks
      las <- par("las")
      mytext <- function(labels, side, at, ...)
          mtext(text = labels, side = side, at = at, line = -.5,
              col=col.lab, cex=cex.lab, font=font.lab, ...)
      ## X
      j <- subset(temp <- pretty(x.range, n = lab[1]),
          temp <= x.range[2] & temp >= x.range[1])
      if(is.null(x.ticklabs))
          x.ticklabs <- j * x.scal
      mytext(x.ticklabs, side = 1, at = j)
      ## Z
      j <- subset(temp <- pretty(z.range, n = lab.z),
          temp <= z.range[2] & temp >= z.range[1])
      if(is.null(z.ticklabs))
          z.ticklabs <- j * z.scal
      mytext(z.ticklabs, side = ifelse(angle.1, 4, 2), at = j,
          adj = ifelse((0 < las) && (las < 3), 1, NA))
      ## Y
      j <- subset(temp <- pretty(c(0, y.max), n = lab[2]),
          temp <= y.max & temp >= 0)
      temp <- if(angle > 2) rev(j) else j ## turn y-labels around
      if(is.null(y.ticklabs))
          y.ticklabs <- y.scal * temp + round(y.add, 0)
      else if (angle > 2)
          y.ticklabs <- rev(y.ticklabs)
      text(j * yx.f + ifelse(angle.2, x.min, x.max),
           j * yz.f + z.min, y.ticklabs,
          pos = ifelse(angle.1, 2, 4), offset = 1,
          col=col.lab, cex = cex.lab/par("cex"), font=font.lab)
  }
}
if(axis) { ## axis and labels
  mytext <-
      function(lab, side = side, at = at, ...)
      mtext(lab, side = side, at = at, col = col.lab,
          cex = cex.axis, font = font.axis, las = 0, ...)
  ## X
  lines(c(x.min, x.max), c(z.min, z.min), col = col.axis, lty=lty.axis)
  mytext(xlab, line = 1.5, side = 1, at = mean(x.range))
```

```
  ## Y
  lines(c(x.max, x.max + y.max * yx.f), c(z.min, y.max * yz.f + z.min),
      col = col.axis, lty = lty.axis)
  mytext(ylab, side = ifelse(angle.1, 2, 4), at = z.min + y.max * yz.f,
      line = .5)
  ## Z
  lines(c(x.min, x.min), c(z.min, z.max), col = col.axis, lty=lty.axis)
  mytext(zlab, line = 1.5, side = ifelse(angle.1, 4, 2),
      at = mean(z.range))
  if(box) {
      ## X
      temp <- yx.f * y.max
      temp1 <- yz.f * y.max
      lines(c(x.min + temp, x.max + temp),
          c(z.min + temp1, z.min+temp1), col=col.axis, lty=lty.axis)
      lines(c(x.min + temp, x.max + temp),
          c(temp1 + z.max, temp1+z.max), col=col.axis, lty=lty.axis)
      ## Y
      temp <- c(0, y.max * yx.f)
      temp1 <- c(0, y.max * yz.f)
      lines(temp + x.min, temp1 + z.min, col=col.axis, lty=lty.axis)
      lines(temp + x.min, temp1 + z.max, col=col.axis, lty=lty.axis)
      ## Z
      temp <- yx.f * y.max
      temp1 <- yz.f * y.max
      lines(c(temp + x.min, temp + x.min),
          c(z.min + temp1, z.max+temp1), col=col.axis, lty=lty.axis)
      lines(c(x.max + temp, x.max + temp),
          c(z.min + temp1, z.max+temp1), col=col.axis, lty=lty.axis)
  }
}

### plot points
x <- dat$x + (dat$y * yx.f)
z <- dat$z + (dat$y * yz.f)
col <- as.character(dat$col)
if(type == "h") {
  z2 <- dat$y * yz.f + z.min
  segments(x, z, x, z2, col = col, cex = cex.symbols, ...)
  points(x, z, type = "p", col = col, pch = pch, cex = cex.symbols, ...)
}
else points(x, z, type = type, col = col, pch = pch,
  cex = cex.symbols, ...)


### box-lines in front of points (overlay)
```

```
if(axis && box) {
  lines(c(x.min, x.max), c(z.max, z.max), col=col.axis, lty=lty.axis)
  lines(c(0, y.max * yx.f) + x.max, c(0, y.max * yz.f) + z.max,
      col = col.axis, lty = lty.axis)
  lines(c(x.max, x.max), c(z.min, z.max), col=col.axis, lty=lty.axis)
}

par(mem.par)
### Return List of functions
ob <- ls() ## remove all unused objects from the result 's enviroment:
rm(list = ob[!ob %in% c("x.scal", "y.scal", "z.scal", "yx.f",
  "yz.f", "y.add", "z.min", "z.max", "x.min", "x.max", "y.max")])
rm(ob)
invisible(list(
  xyz.convert = function(x, y=NULL, z=NULL) {
      xyz <- xyz.coords(x, y, z)
      y <- (xyz$y - y.add) / y.scal
      return(x = xyz$x / x.scal + yx.f * y,
          y = xyz$z / z.scal + yz.f * y)
  },
  points3d = function(x, y = NULL, z = NULL, type = "p", ...) {
      xyz <- xyz.coords(x, y, z)
      y2 <- (xyz$y - y.add) / y.scal
      x <- xyz$x / x.scal + yx.f * y2
      y <- xyz$z / z.scal + yz.f * y2
      if(type == "h") {
          y2 <- z.min + yz.f * y2
          segments(x, y, x, y2, ...)
          points(x, y, type = "p", ...)
      }
      else points(x, y, type = type, ...)
  },
  plane3d = function(Intercept, x.coef = NULL, y.coef = NULL,
      lty = "dashed", ...){
      if(!is.null(coef(Intercept))) Intercept <- coef(Intercept)
      if(is.null(x.coef) && length(Intercept) == 3){
          x.coef <- Intercept[2]
          y.coef <- Intercept[3]
          Intercept <- Intercept[1]
      }
      x <- x.min:x.max
      x.coef <- x.coef * x.scal
      z1 <- (Intercept + x * x.coef + y.add * y.coef) / z.scal
      z2 <- (Intercept + x * x.coef +
          (y.max * y.scal + y.add) * y.coef) / z.scal
      segments(x, z1, x + y.max * yx.f, z2 + yz.f * y.max, lty=lty, ...)
```

```
        y <- 0:y.max
        y.coef <- (y * y.scal + y.add) * y.coef
        z1 <- (Intercept + x.min * x.coef + y.coef) / z.scal
        z2 <- (Intercept + x.max * x.coef + y.coef) / z.scal
        segments(x.min + y * yx.f, z1 + y * yz.f,
            x.max + y * yx.f, z2 + y * yz.f, lty = lty, ...)
    },
    box3d = function(...){
        lines(c(x.min, x.max), c(z.max, z.max), ...)
        lines(c(0, y.max * yx.f) + x.max, c(0, y.max * yz.f) + z.max, ...)
        lines(c(x.max, x.max), c(z.min, z.max), ...)
        lines(c(x.min, x.max), c(z.min, z.min), ...)
    }
  ))
}
```

C Help page (generated from scatterplot3d.Rd)

scatterplot3d	*3D Scatter Plot*

Description

Plots a three dimensional (3D) point cloud.

Usage

```
scatterplot3d(x, y = NULL, z = NULL, color = par("col"), pch = NULL,
    main = NULL, sub = NULL, xlim = NULL, ylim = NULL, zlim = NULL,
    xlab = NULL, ylab = NULL, zlab = NULL, scale.y = 1, angle = 40,
    axis = TRUE, tick.marks = TRUE, label.tick.marks = TRUE,
    x.ticklabs = NULL, y.ticklabs = NULL, z.ticklabs = NULL,
    y.margin.add = 0, grid = TRUE, box = TRUE, lab = par("lab"),
    lab.z = mean(lab[1:2]), type = par("type"), highlight.3d = FALSE,
    mar = c(5,3,4,3) + 0.1, col.axis = par("col.axis"),
    col.grid = "grey", col.lab = par("col.lab"),
    cex.symbols = par("cex"), cex.axis = par("cex.axis"),
    cex.lab = 0.8 * par("cex.lab"), font.axis = par("font.axis"),
    font.lab = par("font.lab"), lty.axis = par("lty"),
    lty.grid = par("lty"), log = "", ...)
```

Arguments

`x` the coordinates of points in the plot.

`y` the y coordinates of points in the plot, optional if `x` is an appropriate structure.

`z` the z coordinates of points in the plot, optional if `x` is an appropriate structure.

`color` colors of points in the plot, optional if `x` is an appropriate structure. Will be ignored if `highlight.3d = TRUE`.

`pch` plotting “character”, i.e. symbol to use.

`main` an overall title for the plot.

`sub` sub-title.

`xlim, ylim, zlim`
the x, y and z limits (min, max) of the plot.

`xlab, ylab, zlab`
: titles for the x, y and z axis.

`scale.y`
: scale of y axis related to x- and z axis.

`angle`
: angle between x and y axis.
Attention: result depends on scaling. For 180 < angle < 360 the returned functions `xyz.convert` and `points3d` will not work properly.

`axis`
: a logical value indicating whether axes should be drawn on the plot.

`tick.marks`
: a logical value indicating whether tick marks should be drawn on the plot (only if `axis = TRUE`).

`label.tick.marks`
: a logical value indicating whether tick marks should be labeled on the plot (only if `axis = TRUE` and `tick.marks = TRUE`).

`x.ticklabs, y.ticklabs, z.ticklabs`
: vector of tick mark labels.

`y.margin.add`
: add additional space between tickmark labels and axis label of the y axis

`grid`
: a logical value indicating whether a grid should be drawn on the plot.

`box`
: a logical value indicating whether a box should be drawn around the plot.

`lab`
: a numerical vector of the form c(x, y, len). The values of x and y give the (approximate) number of tickmarks on the x and y axes.

`lab.z`
: the same as `lab`, but for z axis.

`type`
: character indicating the type of plot: "p" for points, "l" for lines, "h" for vertical lines to x-y-plane, etc.

`highlight.3d`
: points will be drawn in different colors related to y coordinates (only if `type = "p"` or `type = "h"`, else `color` will be used).
On some devices not all colors can be displayed. In this case try the postscript device or use `highlight.3d = FALSE`.

`mar`
: A numerical vector of the form c(bottom, left, top, right) which gives the lines of margin to be specified on the four sides of the plot.

`col.axis, col.grid, col.lab`
: the color to be used for axis / grid / axis labels.

`cex.symbols, cex.axis, cex.lab`
: the magnification to be used for point symbols, axis annotation, labels relative to the current.

`font.axis, font.lab`
: the font to be used for axis annotation / labels.

`lty.axis, lty.grid`
: the line type to be used for axis / grid.

`log`
: Not yet implemented! A character string which contains "x" (if the x axis is to be logarithmic), "y", "z", "xy", "xz", "yz", "xyz".

`...`
: more graphical parameters can be given as arguments, `pch = 16` or `pch = 20` may be nice.

Value

list with components

- `xyz.convert` — function which converts coordinates from 3D (x, y, z) to 2D-projection (x, y) of `scatterplot3d`. Usefull to plot objects into existing plot.
- `points3d` — function which draws points or lines into the existing plot.
- `plane3d` — function which draws a plane into the existing plot: `plane3d(Intercept, x.coef = NULL, y.coef = NULL, lty = "dashed", ...)`. Instead of `Intercept` a vector containing three elements or an (g)lm object can be specified.
- `box3d` — function which "refreshes" the box surrounding the plot.

Note

Some graphical parameters can only be set as arguments in `scatterplot3d` but not in `par`, e.g. `mar`. Other arguments in `par` may be split into several arguments in `scatterplot3d`, e.g. for specifying the line type. And finally some of the arguments in `par` do not work, e.g. many of those for axis calculation. So the recommended way is to try the specification of graphical parameters at first as arguments in `scatterplot3d` and at last as arguments in `par`.

Author(s)

Uwe Ligges ⟨ligges@statistik.uni-dortmund.de⟩;
`http://www.statistik.uni-dortmund.de/leute/ligges.htm`.

See Also

`persp`, `plot`, `par`.

Examples

```
## On some devices not all colors can be displayed.
## Try the postscript device or use highlight.3d = FALSE.

## example 1
z <- seq(-10, 10, 0.01)
x <- cos(z)
y <- sin(z)
scatterplot3d(x, y, z, highlight.3d = TRUE, col.axis = "blue",
  col.grid = "lightblue", main = "scatterplot3d - 1", pch = 20)
```

```
## example 2
temp <- seq(-pi, 0, length = 50)
x <- c(rep(1, 50) %*% t(cos(temp)))
y <- c(cos(temp) %*% t(sin(temp)))
z <- c(sin(temp) %*% t(sin(temp)))
scatterplot3d(x, y, z, highlight.3d = TRUE, col.axis = "blue",
  col.grid = "lightblue", main = "scatterplot3d - 2", pch = 20)

## example 3
temp <- seq(-pi, 0, length = 50)
x <- c(rep(1, 50) %*% t(cos(temp)))
y <- c(cos(temp) %*% t(sin(temp)))
z <- 10 * c(sin(temp) %*% t(sin(temp)))
color <- rep("green", length(x))
temp <- seq(-10, 10, 0.01)
x <- c(x, cos(temp))
y <- c(y, sin(temp))
z <- c(z, temp)
color <- c(color, rep("red", length(temp)))
scatterplot3d(x, y, z, color, pch = 20, zlim = c(-2, 10),
  main = "scatterplot3d - 3")

## example 4
my.mat <- matrix(runif(25), nrow = 5)
dimnames(my.mat) <- list(LETTERS[1:5], letters[11:15])
my.mat # the matrix we want to plot ...

s3d.dat <- data.frame(cols = c(col(my.mat)),
  rows = c(row(my.mat)),
  value = c(my.mat))
scatterplot3d(s3d.dat, type = "h", lwd = 5, pch = " ",
  x.ticklabs = colnames(my.mat), y.ticklabs = rownames(my.mat),
  color = grey(25:1/40), main = "scatterplot3d - 4")

## example 5
data(trees)
s3d <- scatterplot3d(trees, type = "h", highlight.3d = TRUE,
  angle = 55, scale.y = 0.7, pch = 16, main = "scatterplot3d - 5")
# Now adding some points to the "scatterplot3d"
s3d$points3d(seq(10,20,2), seq(85,60,-5), seq(60,10,-10),
  col = "blue", type = "h", pch = 16)
# Now adding a regression plane to the "scatterplot3d"
attach(trees)
my.lm <- lm(Volume ~ Girth + Height)
s3d$plane3d(my.lm)
```