

plot3D : Tools for plotting 3-D and 2-D data.

Karline Soetaert
NIOZ-Yerseke
The Netherlands

Abstract

R package **plot3D** (Soetaert 2021b) contains functions for plotting multi-dimensional data. Many functions are derived from the **persp** function, other functions start from the **image** or **contour** function.

Two related packages are:

- **plot3Drgl** (Soetaert 2021c), that plots multidimensional data using openGL graphics (and using package **rgl** (Adler, Murdoch *et al.* 2020)).
- **OceanView** (Soetaert 2021a) that contains functions for visualizing oceanographic data.

A graphical gallery using one of **plot3D**, **plot3Drgl** or **OceanView** is in <http://www.rforscience.com/rpackages/visualisation/oceanview/> and <http://www.rforscience.com/rpackages/visualisation/plot3d/>

Keywords: plot, persp, image, 2-D, 3-D, scatter plots, surface plots, slice plots, oceanographic data, R .

1. Introduction

R package **plot3D** provides functions for plotting 2-D and 3-D data, and that are either extensions of R's **persp** function or of R's **image** and **contour** function.

The main extensions to these functions are:

- In addition to the x, y (and z) values, an additional data dimension can be represented by a color variable (argument **colvar**).
- A color key (argument **colkey**) can be written next to the figure. It is possible to log-transform the color key, rescale it, adjust its position, ...
- The resolution of a figure can be increased (argument **resfac**).
- Either the **facets** can be colored, just the border, or both.

Package **plot3D** contains:

- Functions that are based on the **persp** function, for visualising 3-D data:
 - **persp3D**: an extended version of the **persp** function.

- `ribbon3D`: perspective plots as ribbons.
- `hist3D`: 3-D histograms.
- `scatter3D`, `points3D`, `lines3D`, `text3D`: scatter plots in 3-D, points, lines, labels.
- `surf3D`: 3-D shapes (or surfaces).
- `slice3D`, `slicecont3D`, `isosurf3D`, `voxel3D`: slices, isosurfaces and voxels from a full 3-D data set.
- `arrows3D`: arrows in 3D.
- `contour3D`, `image3D`: contours and images in 3D.
- `segments3D`, `polygon3D`, `rect3D`, `border3D`, `box3D`: line segments, polygons, rectangles, boxes in 3D.
- Functions defined on the `image` or `contour` function:
 - `image2D`, `contour2D`, for an extended version of these functions to visualise 2-D (or 3-D) data.
 - `ImageOcean`, for an image of the ocean's bathymetry.
- Other functions
- `scatter2D`: colored points, lines, ... in 2-D.
- `text2D`, `arrows2D`, `segments2D`, `rect2D`, `polygon2D` for other 2D functions, comparable to R's base graphics but that have a color key.
- Colors and colorkeys:
 - `colkey`: color legends.
 - `jet.col`, `jet2.col`, `gg.col`, `ramp.col`: suitable color palettes.
- Utility functions:
 - `mesh`: generating rectangular (2D) or (3D) meshes.
 - `plotdev`: plotting on the current device.
- Data sets:
 - `Oxsat`: a (rather large) 3-D data set with the ocean's oxygen saturation values.
 - `Hypsometry`: a 2-D data set with the world's elevation and the ocean's depth.

This vignette contains some examples; more can be found in the package's help files. To run all examples:

```
example(persp3D)
example(surf3D)
example(slice3D)
example(scatter3D)
example(segments3D)
example(image2D)
```

```

example(image3D)
example(contour3D)
example(colkey)
example(jet.col)
example(perspbox)
example(mesh)
example(trans3D)
example(plot.plist)
example(ImageOcean)
example(Oxsat)
example(legendplot)

```

2. Functions image2D and persp3D

`image2D` and `persp3D` are extensions of R's `image` and `persp` functions. The arguments of `persp3D` are (see the help file for what they mean):

args(persp3D)

```

function (x = seq(0, 1, length.out = nrow(z)), y = seq(0, 1,
  length.out = ncol(z)), z, ..., colvar = z, phi = 40, theta = 40,
  col = NULL, Nacol = "white", breaks = NULL, border = NA,
  facets = TRUE, colkey = NULL, resfac = 1, image = FALSE,
  contour = FALSE, panel.first = NULL, clim = NULL, clab = NULL,
  bty = "b", lighting = FALSE, shade = NA, ltheta = -135, lphi = 0,
  inttype = 1, curtain = FALSE, add = FALSE, plot = TRUE)
NULL

```

Many examples of the use of `image2D` and `persp3D` are in vignette `volcano`.

The `Hypsometry` data set is depicted first as an `image`, with 0 m contour lines added. Slight shading gives the plot a perspective view. The zoomed region (used in next figure) is then added.

```

image2D(Hypsometry, xlab = "longitude", ylab = "latitude",
  contour = list(levels = 0, col = "black", lwd = 2),
  shade = 0.1, main = "Hypsometry data set", clab = "m")
rect(-50, 10, -20, 40, lwd = 3)

```

```

ii <- which(Hypsometry$x > -50 & Hypsometry$x < -20)
jj <- which(Hypsometry$y > 10 & Hypsometry$y < 40)
zlim <- c(-10000, 0)

```

The perspective figure is made with black side-panels (`bty`). Grey contour lines are added on the bottom panel ("`zmin`") and on the `persp` plot itself ("`z`"). The resolution is increased (`resfac`) to make smoother images. A color key (`colkey`) is added on the first margin (`side`)

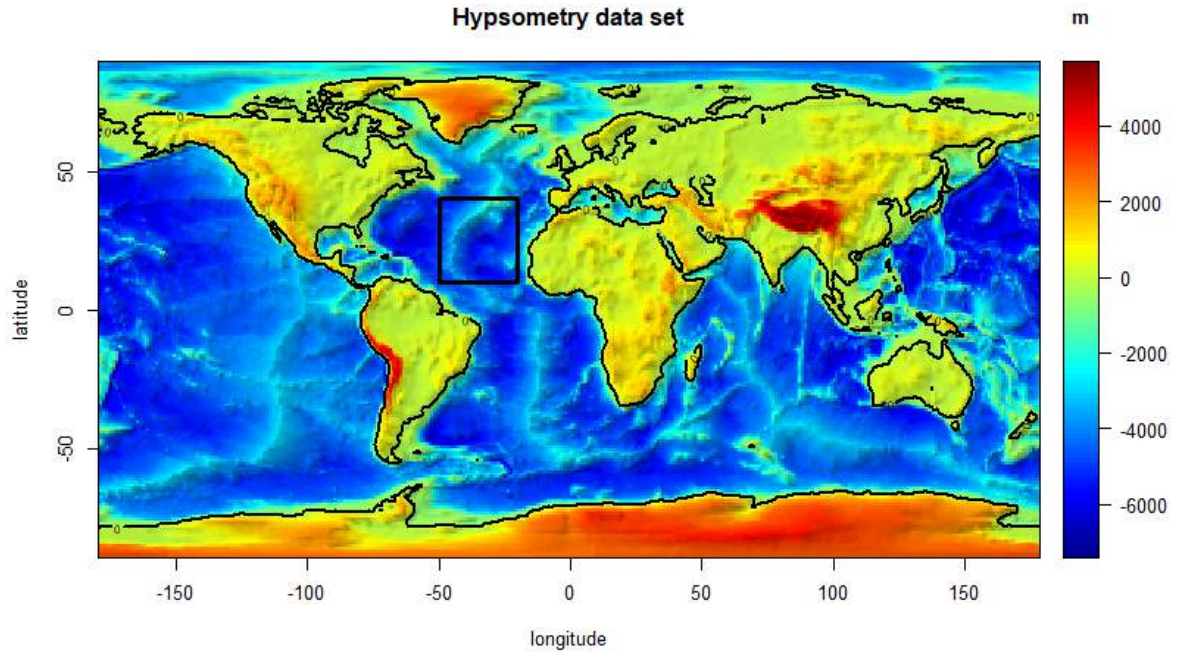


Figure 1: Hypsometry data set

```

par(mfrow = c(1, 1))
# Actual bathymetry, 4 times increased resolution, with contours
persp3D(z = Hypsometry$z[ii,jj], xlab = "longitude", bty = "bl2",
        ylab = "latitude", zlab = "depth", clab = "depth, m",
        expand = 0.5, d = 2, phi = 20, theta = 30, resfac = 2,
        contour = list(col = "grey", side = c("zmin", "z")),
        zlim = zlim, colkey = list(side = 1, length = 0.5))

```

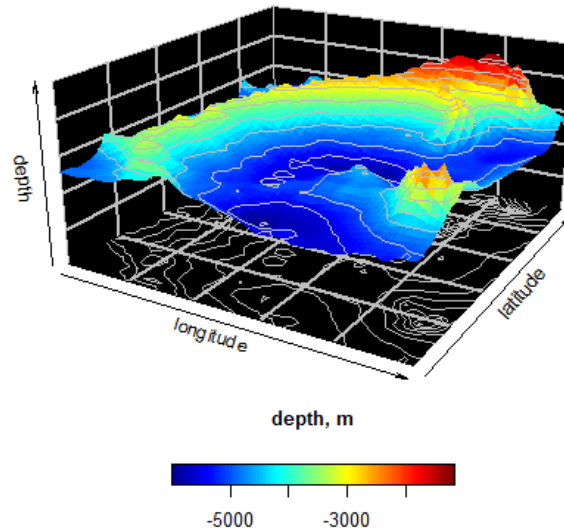


Figure 2: Bathymetry of a part of the ocean

3. slices and isosurfaces

Function `slice3D` draws slices from volumetric (3D) data, function `isosurf3D` creates and plots isosurfaces. It makes use of a function from package `misc3d` (Feng and Tierney 2008).

```
args(slice3D)

function (x, y, z, colvar, ..., phi = 40, theta = 40, xs = min(x),
  ys = max(y), zs = min(z), col = NULL, NAcot = "white", breaks = NULL,
  border = NA, facets = TRUE, colkey = NULL, panel.first = NULL,
  clim = NULL, clab = NULL, bty = "b", lighting = FALSE, shade = NA,
  ltheta = -135, lphi = 0, add = FALSE, plot = TRUE)
NULL
```

Function `mesh` is used to generate a full rectangular 3-D mesh. This is used to generate the volumetric data (`p`) that defines the coloration. The data are visualised by one slice in `x` (`xs`) and 3 slices in `y` direction (`ys`). Function `isosurf3D` plots the data for `p`-values that are equal to 0.

```
par(mfrow = c(1, 2))
x <- y <- z <- seq(-4, 4, by = 0.2)
M <- mesh(x, y, z)
R <- with (M, sqrt(x^2 + y^2 + z^2))
```

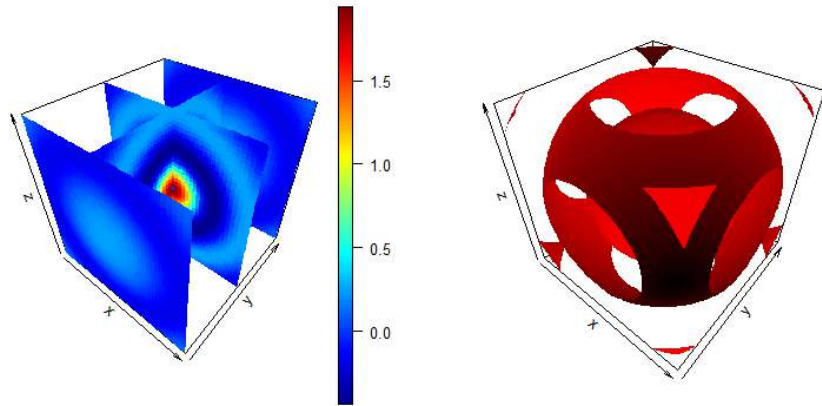


Figure 3: Slices and isosurfaces from volumetric data

```

p <- sin(2*R)/(R+1e-3)
slice3D(x, y, z, colvar = p,
        xs = 0, ys = c(-4, 0, 4), zs = NULL)
isosurf3D(x, y, z, colvar = p, level = 0, col = "red")

```

4. surf3D

Function `surf3D` creates 3-D surface plots.

`args(surf3D)`

```
function (x, y, z, ..., colvar = z, phi = 40, theta = 40, col = NULL,
  NAcol = "white", breaks = NULL, border = NA, facets = TRUE,
  colkey = NULL, panel.first = NULL, clim = NULL, clab = NULL,
  bty = "n", lighting = FALSE, shade = NA, ltheta = -135, lphi = 0,
  inttype = 1, add = FALSE, plot = TRUE)
NULL
```

Here are 4 applications, showing the different options of coloration.

```
par(mfrow = c(2, 2), mar = c(0, 0, 0, 0))
# Shape 1
M <- mesh(seq(0, 6*pi, length.out = 80),
  seq(pi/3, pi, length.out = 80))
u <- M$x ; v <- M$y
x <- u/2 * sin(v) * cos(u)
y <- u/2 * sin(v) * sin(u)
z <- u/2 * cos(v)
surf3D(x, y, z, colvar = z, colkey = FALSE, box = FALSE)
# Shape 2: add border
M <- mesh(seq(0, 2*pi, length.out = 80),
  seq(0, 2*pi, length.out = 80))
u <- M$x ; v <- M$y
x <- sin(u)
y <- sin(v)
z <- sin(u + v)
surf3D(x, y, z, colvar = z, border = "black", colkey = FALSE)
# shape 3: uses same mesh, white facets
x <- (3 + cos(v/2)*sin(u) - sin(v/2)*sin(2*u))*cos(v)
y <- (3 + cos(v/2)*sin(u) - sin(v/2)*sin(2*u))*sin(v)
z <- sin(v/2)*sin(u) + cos(v/2)*sin(2*u)
surf3D(x, y, z, colvar = z, colkey = FALSE, facets = FALSE)
# shape 4: more complex colvar
M <- mesh(seq(-13.2, 13.2, length.out = 50),
  seq(-37.4, 37.4, length.out = 50))
u <- M$x ; v <- M$y
b <- 0.4; r <- 1 - b^2; w <- sqrt(r)
D <- b*((w*cosh(b*u))^2 + (b*sin(w*v))^2)
x <- -u + (2*r*cosh(b*u)*sinh(b*u)) / D
y <- (2*w*cosh(b*u)*(-(w*cos(v)*cos(w*v)) - sin(v)*sin(w*v))) / D
z <- (2*w*cosh(b*u)*(-(w*sin(v)*cos(w*v)) + cos(v)*sin(w*v))) / D
surf3D(x, y, z, colvar = sqrt(x + 8.3), colkey = FALSE,
  border = "black", box = FALSE)
```

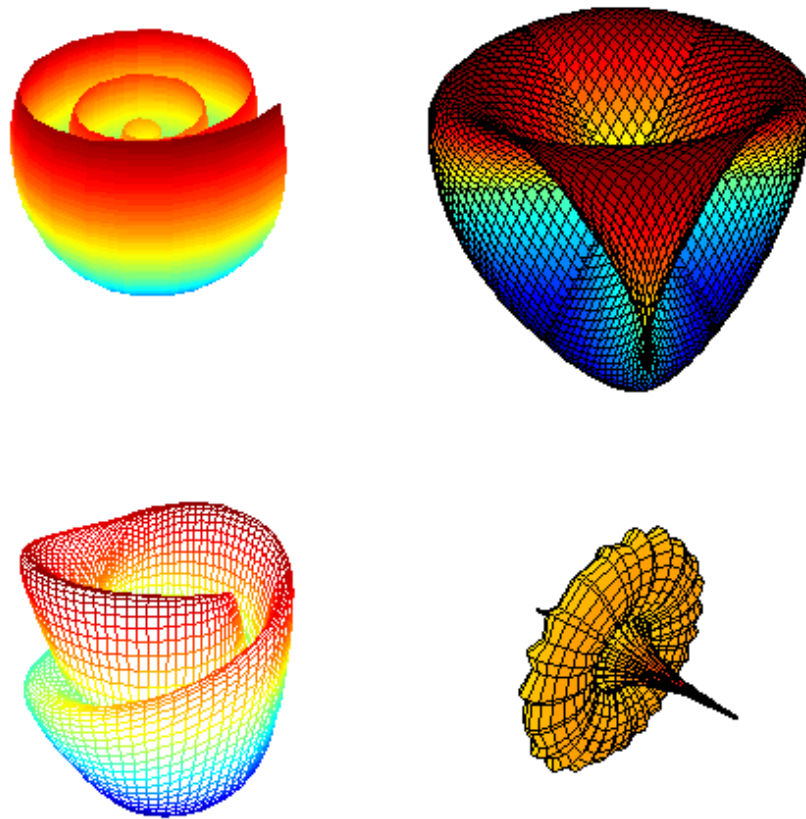


Figure 4: Surface plots

4.1. scatter2D and scatter3D

Functions `scatter2D` and `scatter3D` draw scatterplots.

```
args(scatter2D)

function (x, y, ..., colvar = NULL, col = NULL, NAcot = "white",
  breaks = NULL, colkey = NULL, clim = NULL, clab = NULL, CI = NULL,
  add = FALSE, plot = TRUE)
NULL

args(scatter3D)

function (x, y, z, ..., colvar = z, phi = 40, theta = 40, col = NULL,
  NAcot = "white", breaks = NULL, colkey = NULL, panel.first = NULL,
  clim = NULL, clab = NULL, bty = "b", CI = NULL, surf = NULL,
  add = FALSE, plot = TRUE)
NULL
```

The dataset `quakes` is plotted using function `scatter3D`. Before the 3-D quakes data are drawn, small dots are added on the bottom and on the depth plane (`panel.first`).

```
par(mfrow = c(1, 1))
panel.first <- function(pmat) {
  zmin <- min(-quakes$depth)
  XY <- trans3D(quakes$long, quakes$lat,
    z = rep(zmin, nrow(quakes)), pmat = pmat)
  scatter2D(XY$x, XY$y, colvar = quakes$mag, pch = ".",
    cex = 2, add = TRUE, colkey = FALSE)

  xmin <- min(quakes$long)
  XY <- trans3D(x = rep(xmin, nrow(quakes)), y = quakes$lat,
    z = -quakes$depth, pmat = pmat)
  scatter2D(XY$x, XY$y, colvar = quakes$mag, pch = ".",
    cex = 2, add = TRUE, colkey = FALSE)
}
with(quakes, scatter3D(x = long, y = lat, z = -depth, colvar = mag,
  pch = 16, cex = 1.5, xlab = "longitude", ylab = "latitude",
  zlab = "depth, km", clab = c("Richter", "Magnitude"),
  main = "Earthquakes off Fiji", ticktype = "detailed",
  panel.first = panel.first, theta = 10, d = 2,
  colkey = list(length = 0.5, width = 0.5, cex.clab = 0.75))
)
```

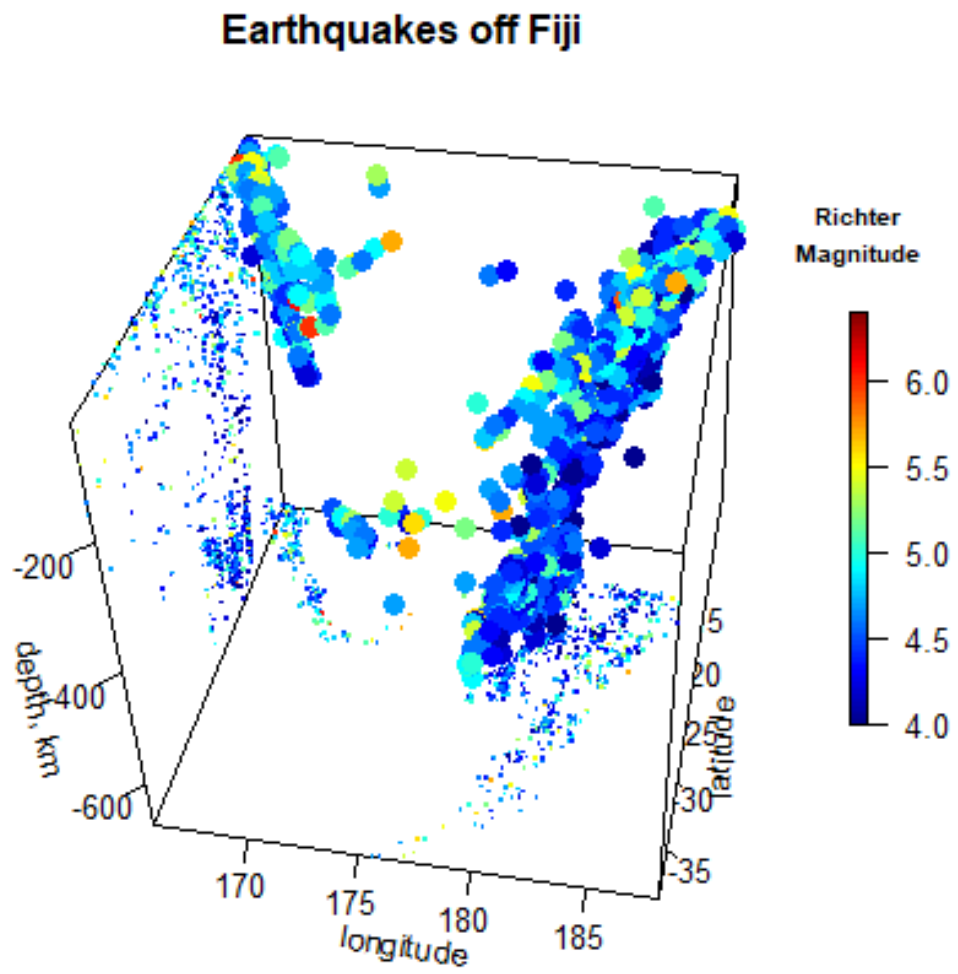


Figure 5: Scatter plot

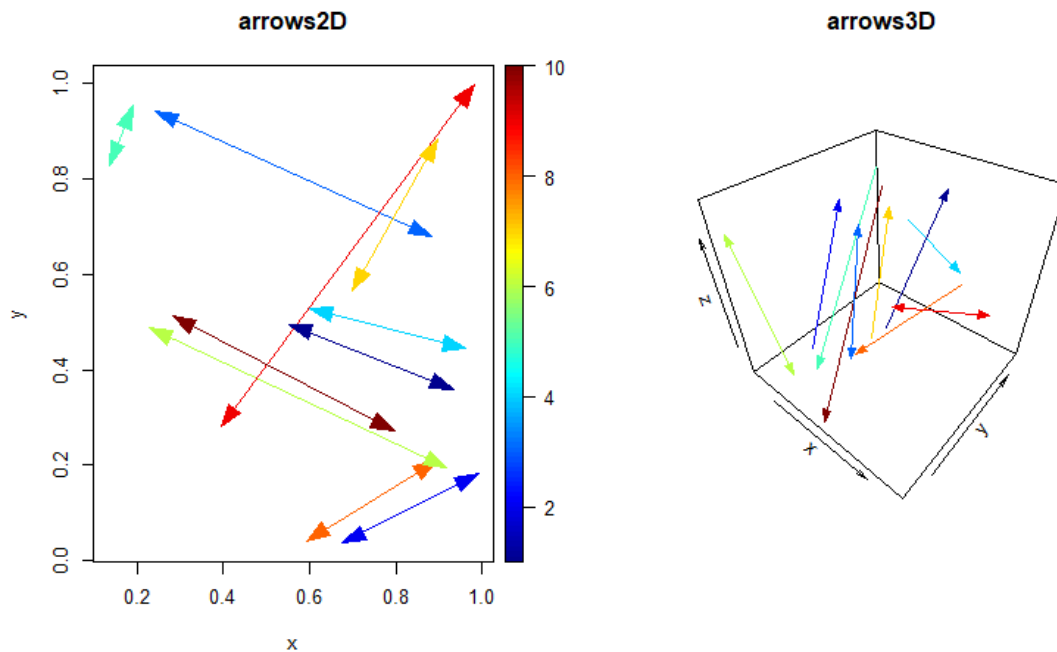


Figure 6: arrows

4.2. arrows3D, arrows2D

Functions `arrows2D` and `arrows3D` extend R function `arrows` with a color variable.

```
par (mfrow = c(1, 2))
arrows2D(x0 = runif(10), y0 = runif(10),
         x1 = runif(10), y1 = runif(10), colvar = 1:10,
         code = 3, main = "arrows2D")
arrows3D(x0 = runif(10), y0 = runif(10), z0 = runif(10),
         x1 = runif(10), y1 = runif(10), z1 = runif(10),
         colvar = 1:10, code = 1:3, main = "arrows3D", colkey = FALSE)
```

5. Functions based on image

The `image2D` function is an extended version of `image`. It has two S3 methods:

```
image2D(z =, ...)

image2D.matrix(z, x = NULL, y = NULL, ...,
               col = jet.col(100), NAcol = "white", facets = TRUE,
               contour = FALSE, colkey = NULL, resfac = 1,
               clab = NULL, theta = 0, border = NA)
image2D.array(z, margin = c(1, 2), subset, ask = NULL, ...)
```

The data set `Oxsat` has oxygen saturation values in the ocean, at 2dg horizontal resolution, and for 33 depth intervals.

```
names(Oxsat)

[1] "lon"  "lat"  "depth" "val"  "name" "units"

dim(Oxsat$val)

[1] 180  90  33
```

Function `image2D.array` plots several depth intervals at once, looping over the first and second margin. The color key is added in a separate figure.

```
sub <- c(1, 5, 9)
image2D(z = Oxsat$val, subset = sub,
        x = Oxsat$lon, y = Oxsat$lat,
        margin = c(1, 2), NAcol = "black", colkey = FALSE,
        xlab = "longitude", ylab = "latitude",
        main = paste("depth ", Oxsat$depth[sub], " m"),
        clim = c(0, 115), mfrow = c(2, 2))
colkey(clim = c(0, 115), clab = c("O2 saturation", "percent"))
```

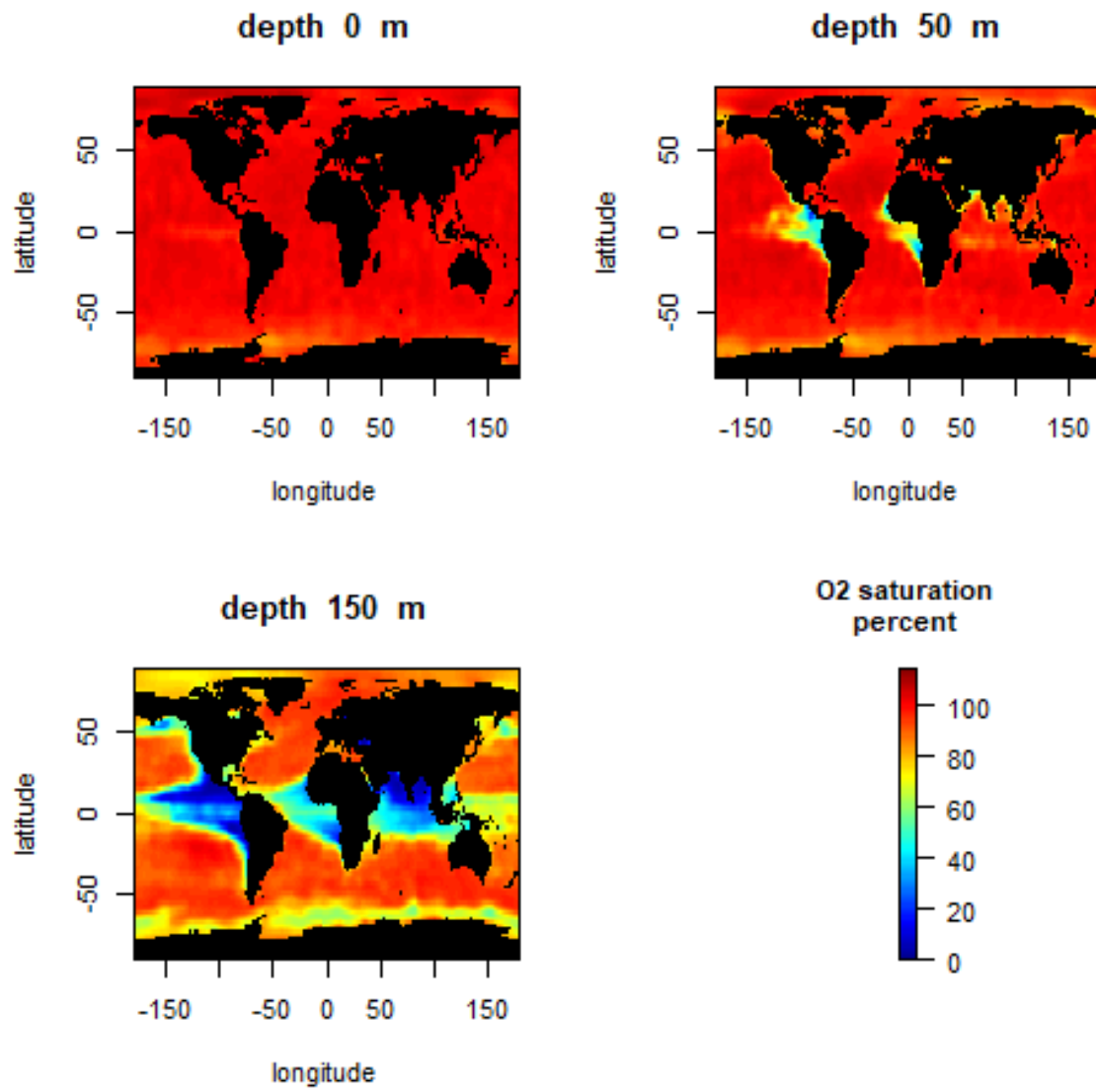


Figure 7: image2D function

6. Composite figures

It is also possible to make a composite figure combining several functions.

```
persp3D(z = volcano, zlim = c(-60, 200), phi = 20,
        colkey = list(length = 0.2, width = 0.4, shift = 0.15,
                      cex.axis = 0.8, cex.clab = 0.85), lighting = TRUE, lphi = 90,
        clab = c("", "height", "m"), bty = "f", plot = FALSE)
# create gradient in x-direction
Vx <- volcano[-1, ] - volcano[-nrow(volcano), ]
# add as image with own color key, at bottom
image3D(z = -60, colvar = Vx/10, add = TRUE,
        colkey = list(length = 0.2, width = 0.4, shift = -0.15,
                      cex.axis = 0.8, cex.clab = 0.85),
        clab = c("", "gradient", "m/m"), plot = FALSE)
# add contour
contour3D(z = -60+0.01, colvar = Vx/10, add = TRUE,
          col = "black", plot = TRUE)
```

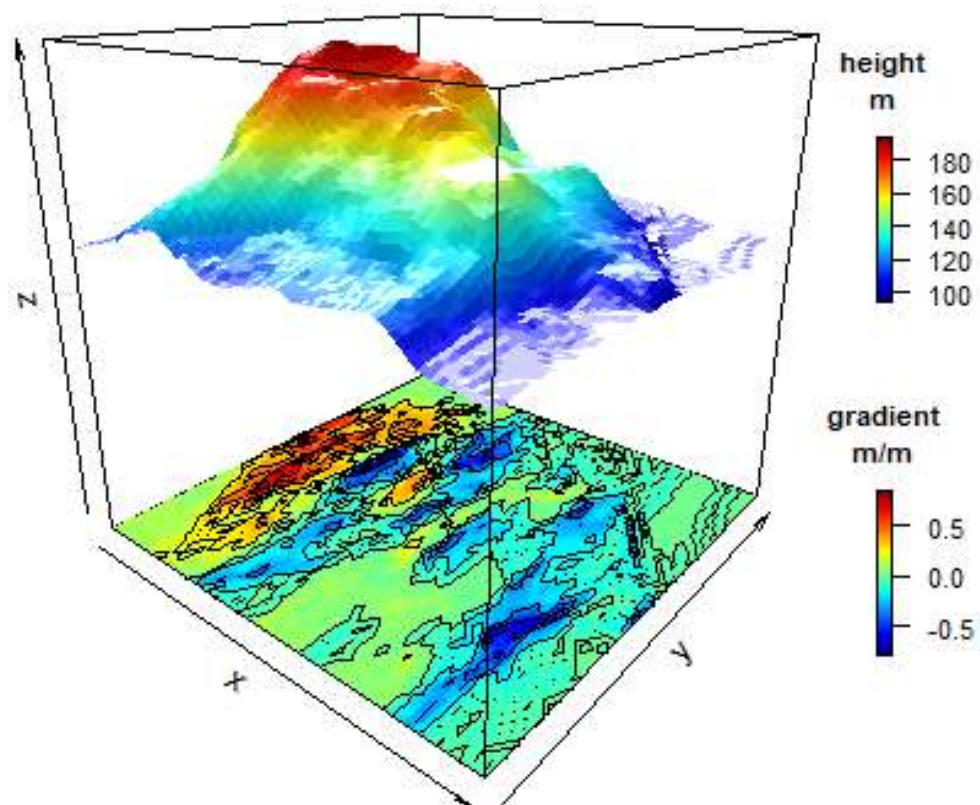


Figure 8: Several color keys in composite figure

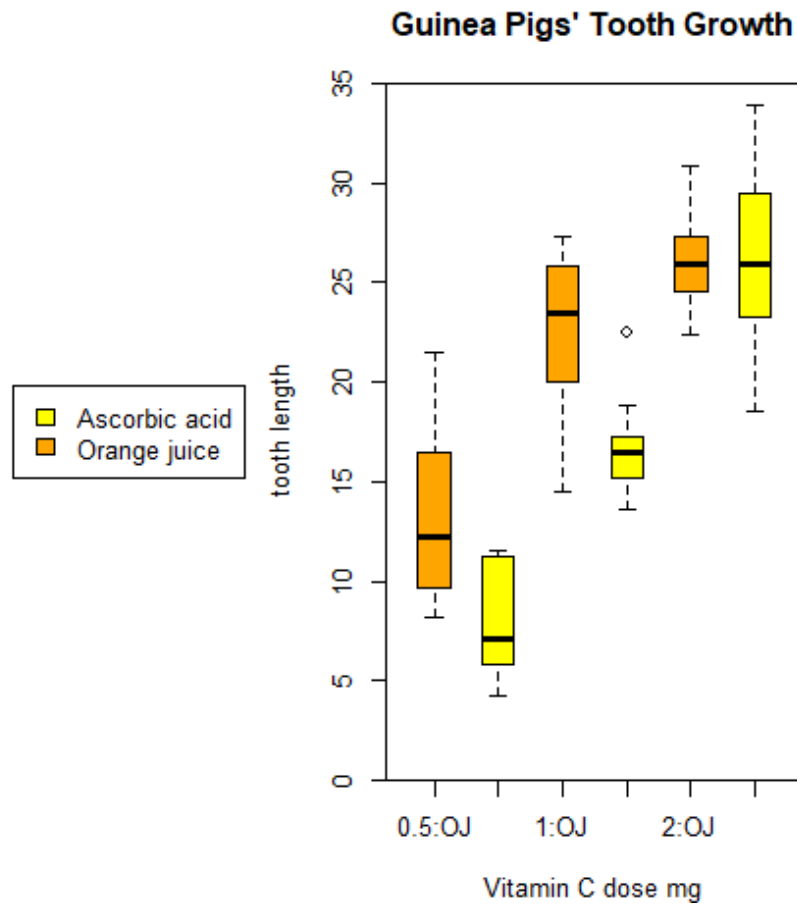


Figure 9: adding a legend to boxplot

7. plotting with legends and colorkeys

As from version 1.4, a new feature has been added, allowing plotting functions to have colorkeys, or legends that are positioned outside of the plotting region. See `?legendplot`.

For instance, to put a legend outside of a boxplot:

```
pm <- par(mar = c(4,3,4,2))
legend.plt(formula = len ~ dose:supp, data = ToothGrowth,
  boxwex = 0.5, col = c("orange", "yellow"),
  main = "Guinea Pigs' Tooth Growth",
  xlab = "Vitamin C dose mg", ylab = "tooth length",
  sep = ":", lex.order = TRUE, ylim = c(0, 35), yaxs = "i",
  method = "boxplot", legend.side = 2,
  legend = list(legend = c("Ascorbic acid", "Orange juice"),
    fill = c("yellow", "orange")))
```

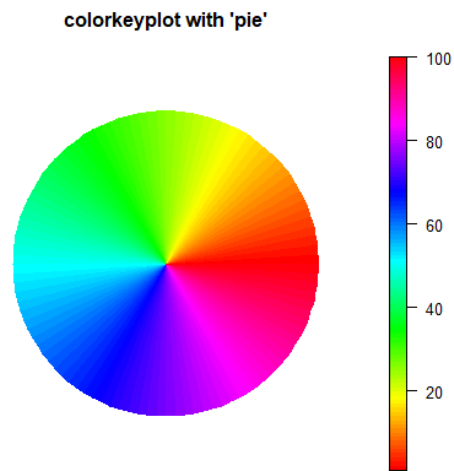



Figure 10: adding a colorkey

To add a colorkey

```
n <- 100
colorkey.plt(method = "pie", x = rep(1, n), labels = "",
             col = rainbow(n), border = NA,
             main = "colorkeyplot with 'pie'",
             colorkey = list(col = rainbow(n), clim = c(1,n)))
```

8. Issues

8.1. Specifying axes limits

There are two ways in which the axes limits can be set, either allowing the 3D data to overflow the axes, or to be clipped. Consider the following code, which is based on a demo from the **rgl** package (Adler *et al.* 2020). In this code, the **iris** data set is fitted with two models, and the data plotted using **scatter3D** and the fitted surfaces using **persp3D**.

The z-axis limits are set to `c(1, 9)`; plotting is postponed until all objects have been added (this avoids clogging - see next subsection). Then the result is plotted using **plotdev** (this plots the graph on any device that has been opened). The first time, without specifying the axes limits, the second time setting the axes limits.

```
nout <- 30
xout <- with(iris, seq(min(Sepal.Length), max(Sepal.Length), length = nout))
yout <- with(iris, seq(min(Sepal.Width), max(Sepal.Width), length = nout))
xy <- expand.grid(Sepal.Length = xout, Sepal.Width = yout)
# Fit two models, linear and quadratic
mod <- with(iris, lm(Petal.Length ~ Sepal.Length + Sepal.Width))
mod2 <- with(iris, lm(Petal.Length ~ Sepal.Length + Sepal.Width +
                      I(Sepal.Length^2) + I(Sepal.Width^2) +
                      I(Sepal.Length*Sepal.Width)))
# predict at new values
zpred.1 <- matrix(
  predict(mod, newdata = xy), nrow = nout, ncol = nout)
zpred.2 <- matrix(
  predict(mod2, newdata = xy), nrow = nout, ncol = nout)
# make graph, postpone plotting till the end
par(mfrow = c(1, 2))
with(iris,
  scatter3D(Sepal.Length, Sepal.Width, Petal.Length,
    colvar = as.numeric(Species), colkey = FALSE,
    col = c("blue", "red", "gold"), bty = "b",
    xlab = 'SL', ylab = 'PL', zlab = 'SW', zlim = c(1, 9),
    pch = 16, cex = 2, theta = 0, plot = FALSE))
persp3D(x = xout, y = yout, z = zpred.1, facets = NA,
  add = TRUE, col = "blue", plot = FALSE)
persp3D(x = xout, y = yout, z = zpred.2,
  add = TRUE, col = "green", plot = FALSE)
# plot using traditional device
plotdev(theta = -50, alpha = 0.5)
plotdev(theta = -50, alpha = 0.5, zlim = c(1, 9))

# if you want to see this in rgl:
# library(plot3Drgl)
# plotrgl(alpha = 0.5)
```

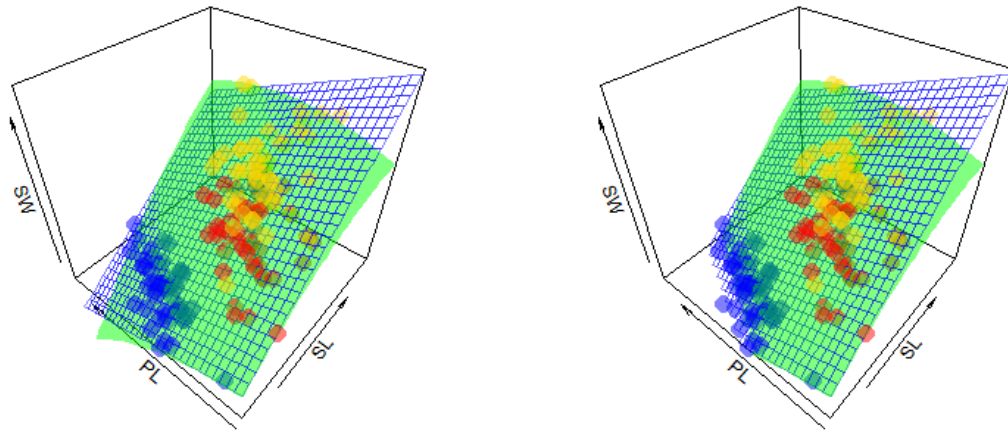


Figure 11: calling `plotdev()` with axes limits causes the figure to be clipped

8.2. Preventing clogging of the figures

When a lot of objects are added then the ultimate figure may appear clogged. This is because each time something is added, all the objects are redrawn on top of what was already there - the only thing that is not redrawn are the axes and titles. To create ‘slim’ figures, use `plotdev()` (e.g. see previous section).

9. Finally

This vignette was made with Sweave ([Leisch 2002](#)).

References

- Adler D, Murdoch D, *et al.* (2020). *rgl: 3D Visualization Using OpenGL*. R package version 0.100.54, URL <http://CRAN.R-project.org/package=rgl>.
- Feng D, Tierney L (2008). “Computing and Displaying Isosurfaces in R.” *Journal of Statistical Software*, **28**(1). URL <http://www.jstatsoft.org/v28/i01/>.
- Leisch F (2002). “Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis.” In W Härdle, B Rönz (eds.), *Compstat 2002 - Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL <http://www.stat.uni-muenchen.de/~leisch/Sweave>.
- Soetaert K (2021a). *OceanView: Visualisation of Oceanographic Data and Model Output*. R package version 1.0.6, URL <http://CRAN.R-project.org/package=OceanView>.
- Soetaert K (2021b). *plot3D: Plotting multi-dimensional data*. R package version 1.4, URL <http://CRAN.R-project.org/package=plot3D>.
- Soetaert K (2021c). *plot3Drgl: Plotting multi-dimensional data - using rgl*. R package version 1.0.2, URL <http://CRAN.R-project.org/package=plot3Drgl>.

Affiliation:

Karline Soetaert
Royal Netherlands Institute of Sea Research (NIOZ)
4401 NT Yerseke, Netherlands
E-mail: karline.soetaert@nioz.nl
URL: <http://http://www.nioz.nl/>