hyperSpec Plotting functions

Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>

DIA Raman Spectroscopy Group, University of Trieste/Italy (2005 – 2008)

Spectroscopy · Imaging, IPHT, Jena/Germany (2008 – 2017)

ÖPV, JKI, Berlin/Germany (2017 – 2019)

Arbeitskreis Lebensmittelmikrobiologie und Biotechnologie, Hamburg University, Hamburg/German

Chemometric Consulting and Chemometrix GmbH, Wölfersheim/Germany (since 201

May 27, 2020

Reproducing the Examples in this Vignette

All spectra used in this manual are installed automatically with *hyperSpec*.
Note that some definitions are executed in `vignette.defs`, and others invisibly at the beginning of the file in order to have the code as similar as possible to interactive sessions.

Contents

> ## Suggested Packages
>
> *latticeExtra*: available
>
> *deldir*: available
>
> *rgl*: available
>
> *ggplot2*: available
>
> In addition *tripack*, and *latticist* are mentioned, but not used in this vignette.

Preliminary Calculations

For some plots of the `chondro` dataset, the pre-processed spectra and their cluster averages ± one
standard deviation are more suitable:

```
> chondro.preproc <- chondro - spc.fit.poly.below (chondro)
> chondro.preproc <- chondro.preproc / rowMeans (chondro)
> chondro.preproc <- chondro.preproc - quantile (chondro.preproc, 0.05)
> cluster.cols <- c ("dark blue", "orange", "#C02020")
> cluster.meansd <- aggregate (chondro.preproc, chondro$clusters, mean_pm_sd)
> cluster.means  <- aggregate (chondro.preproc, chondro$clusters, mean)
```

For details about the pre-processing, please refer to the example work flow in `vignette ("chondro")`,
or the help `? chondro`.

1 Predefined functions

hyperSpec comes with 6 major predefined plotting functions.

`plot`   main switchyard for most plotting tasks

`levelplot`   *hyperSpec* has a method for *lattice*[**?**] function `levelplot`

`plotspc`   plots spectra

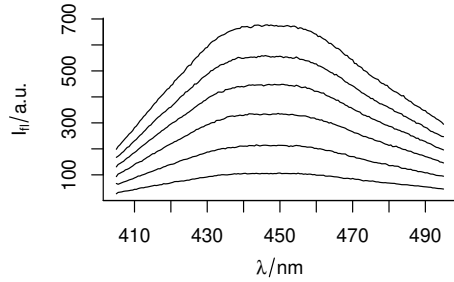`plotmat`   plots the spectra matrix

`plotc`   calibration plot, time series, depth profile
               `plotc` is a *lattice* function

`plotmap`   more specialized version of `levelplot` for map or image plots.
               `plotmap` is a *lattice* function

`plotvoronoi` more specialized version of `plotmap` that produces Voronoi tesselations.
               `plotvoronoi` is a *lattice* function

`plotmap`, `plotvoronoi`, and `levelplot` are *lattice* functions. Therefore, in loops, functions, Sweave chunks, etc. the lattice object needs to be printed explicitly by e. g. `print (plotmap (object))` (R FAQ: Why do lattice/trellis graphics not work?).
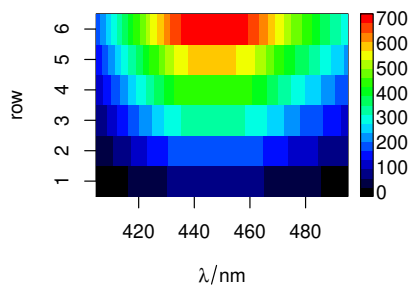
plotspc

plots the spectra, i. e. the intensities `$spc` over the wavelengths `@wavelength`.
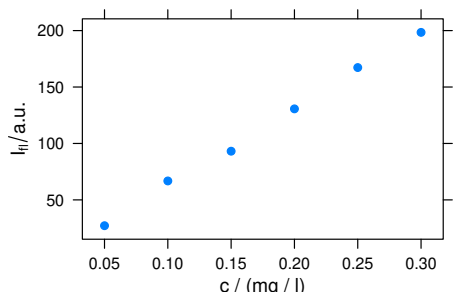```
> plotspc (flu)
```

**plotmat**



plots the spectra, i. e. the colour coded intensities `$spc` over the wavelengths `@wavelength` and the row number.
```
> plotmat (flu)
```
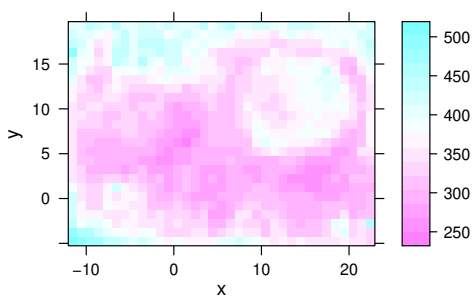
**plotc**



plots an intensity over a single other data column, e. g.

- calibration
- time series
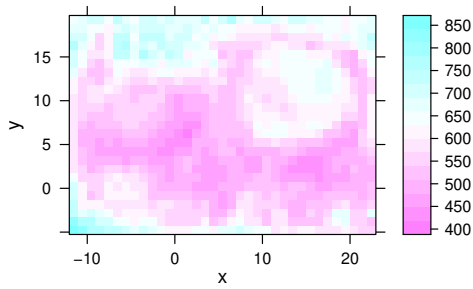- depth profile

```
> plotc (flu)
```

levelplot

plots a false colour map, defined by a formula.
```
> levelplot (spc ~ x * y, chondro, aspect = "iso")
```
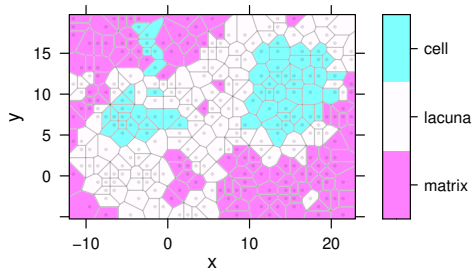Warning: `Only first wavelength is used for plotting`

plotmap

plotmap is a specialized version of levelplot. It uses a single value (e. g. average intensity or cluster membership) over two data columns (default $x and $y)

```
> plotmap (chondro)
```

plotvoronoi

Function plotvoronoi is a special version of plotmap that produces Voronoi diagram of the hyperSpec object.

```
> plotvoronoi (sample (chondro, 300), clusters ~ x * y)
```

2 Arguments for plot

hyperSpec's plot method uses its second argument to determine which of the specialized plots to produce. This allows some handy abbreviations. All further arguments are handed over to the function actually producing the plot.
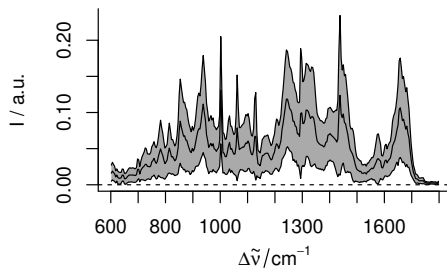
plot (x, "spc")
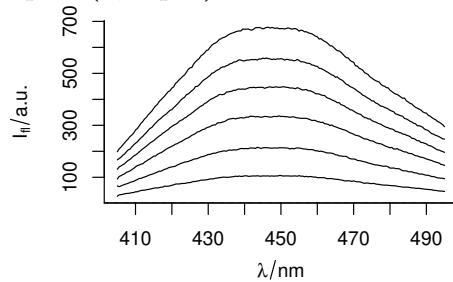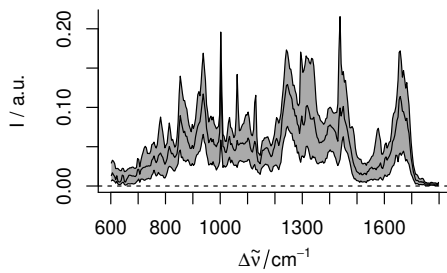

is equivalent to plotspc (flu)

```
> plot (flu, "spc")
```

plot (x, "spcmeansd")


plots mean spectrum ± 1 standard deviation
```
> plot (chondro.preproc, "spcmeansd")
```

plot (x, "spcprctile")

plots median, 16th and 84th percentile for each wavelength. For Gaussian distributed data, 16th, 50th and 84th percentile are equal to mean ± standard deviation. Spectroscopic data frequently are not Gaussian distributed. The percentiles give a better idea of the true distribution. They are also less sensitive to outliers.
```
> plot (chondro.preproc, "spcprctile")
```

plot (x, "spcprctl5")
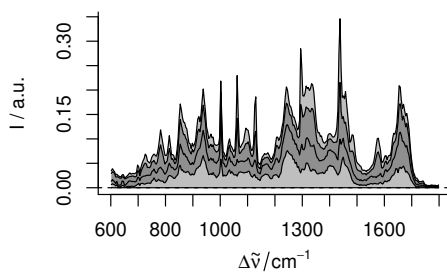
like `"spcprctl"` plus 5th and 95th percentile.
```
> plot (chondro.preproc, "spcprctl5")
```
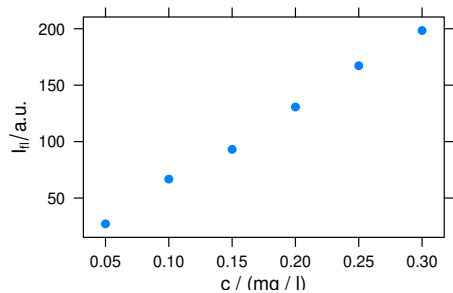
plot (x, "c")


```
> plot (flu, "c")
```
is equivalent to `plotc (flu)`

plot (x, "ts")

plots a time series plot
```
> plot (laser [,, 405], "ts")
```
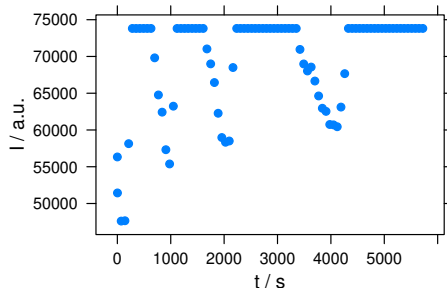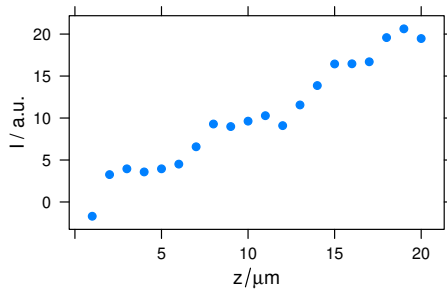equivalent to `plotc (laser, spc ~ t)`

plot (x, "depth")

plots a depth profile plot
```
> depth.profile <- new ("hyperSpec",
+     spc = as.matrix (rnorm (20) + 1:20),
+     data = data.frame (z = 1 : 20),
+     labels = list (spc = "I / a.u.",
+         z = expression (`/` (z, mu*m)),
+         .wavelength = expression (lambda)))
> plot (depth.profile, "depth")
```
the same as `plotc (laser, spc ~ z)`

plot (x, "mat")
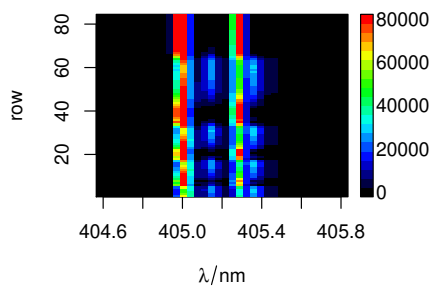
plots the spectra matrix.
```
> plot (laser, "mat")
```
Equivalent to
```
> plotmat (laser)
```
A lattice alternative is:
```
> levelplot (spc ~ .wavelength * .row, laser)
```

plot (x, "map")
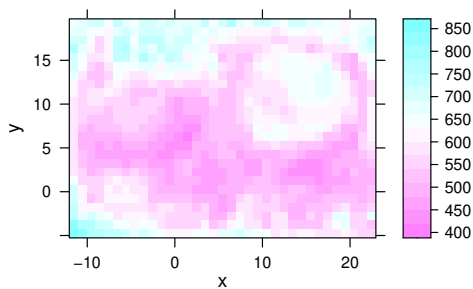
is equivalent to `plotmap (chondro)`
```
> plot (chondro, "map")
```

7

plot (x, "voronoi")


```
> plot (sample (chondro, 300), "voronoi")
```
See `plotvoronoi`

3 Spectra

`plotspc` offers a variety of parameters for customized plots. To plot ...

particular wavelength range
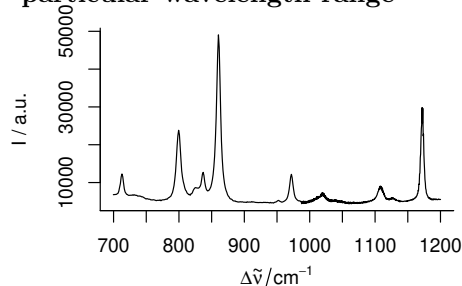
if only one wavelength range is needed, the extract command (see `vignette ("introduction")`) is handiest:
```
> plotspc (paracetamol [,, 700 ~ 1200])
```
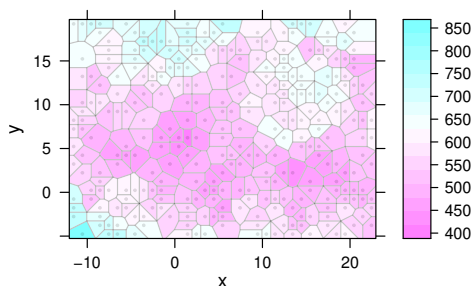wavelengths. If `wl.range` already contains indices use `wl.index = TRUE`.

more wavelength ranges
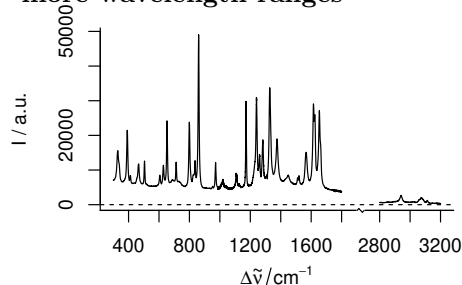
use `wl.range = list (600 ~ 1800, 2800 ~ 3100)`. Cut the wavelength axis appropriately with `xoffset = 750`
```
> plotspc (paracetamol,
+          wl.range = c (300 ~ 1800, 2800 ~ max),
+          xoffset = 750)
```
If available, the package *plotrix*[1] is used to produce the cut mark.

with reversed abscissa
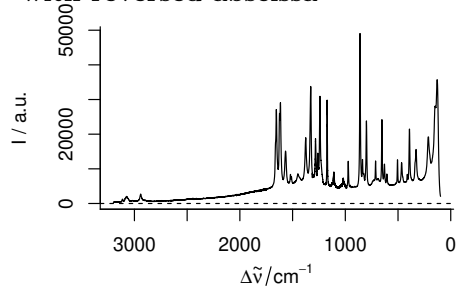
use `wl.reverse = TRUE`
```
> plotspc (paracetamol, wl.reverse = TRUE )
```

8

in different colours
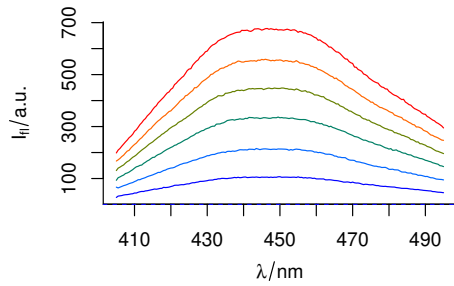
use `col = vector.of.colours`
```
> plotspc (flu, col = matlab.dark.palette (6))
```

dots instead of lines
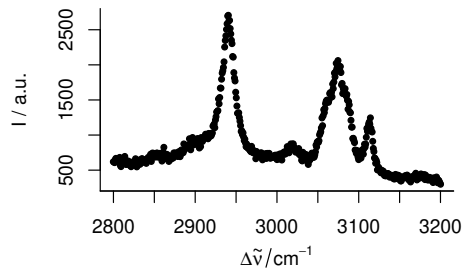
use `lines.args = list (pch = 20, type = "p")`
```
> plotspc (paracetamol [,, 2800 ~ 3200],
+          lines.args = list (pch = 20, type = "p"))
```

mass spectra
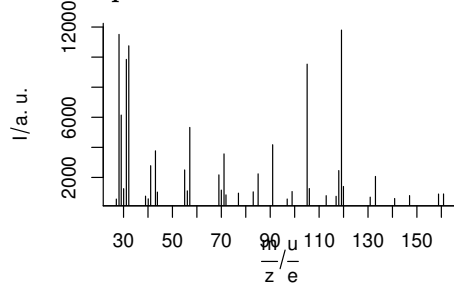
use `lines.args = list (type = "h")`
```
> plot (barbiturates [[1]], lines.args = list (type = "h"))
```

more spectra into an existing plot
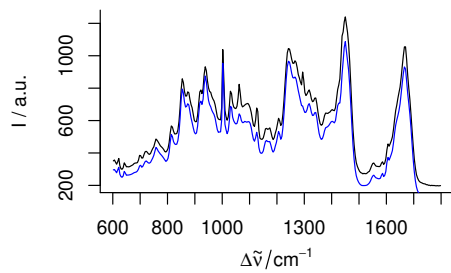
use `add = TRUE`
```
> plotspc (chondro [ 30,,])
> plotspc (chondro [300,,], add = TRUE, col = "blue")
```

Summary characteristics
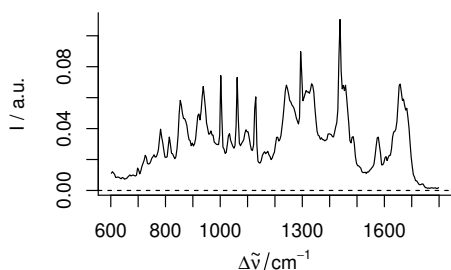
func may be used to calculate summary characteristics prior to plotting. To plot e.g. the standard deviation of the spectra, use:

```
> plotspc (chondro.preproc, func = sd)
```

with different line at $I = 0$
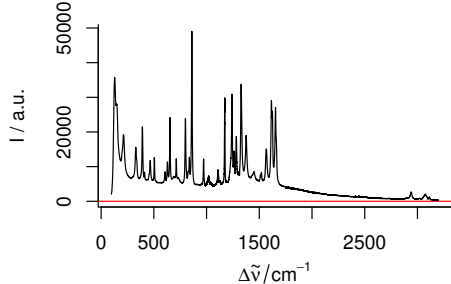
zeroline takes a list with parameters to `abline`, NA suppresses the line.

```
> plotspc (paracetamol,
+          zeroline = list (col = "red"))
```

adding to a spectra plot
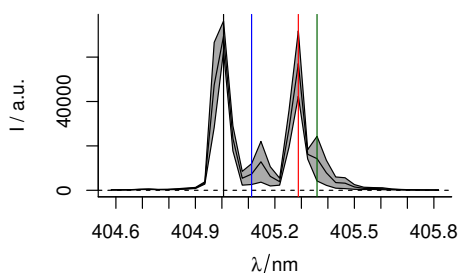
`plotspc` uses base graphics. After plotting the spectra, more content may be added to the graphic by `abline`, `lines`, `points`, etc.

```
> plot (laser, "spcmeansd")
> abline (v = c(405.0063, 405.1121, 405.2885, 405.3591),
+         col = c("black", "blue", "red", "darkgreen"))
```

3.1 Stacked spectra

stacked
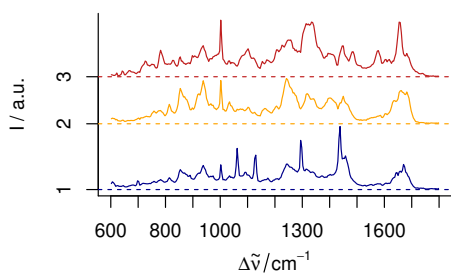
use `stacked = TRUE`

```
> plotspc (cluster.means,
+          col = cluster.cols,
+          stacked = TRUE)
```

Stacking groups of spectra
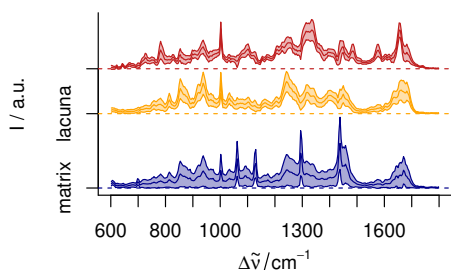
The spectra to be stacked can be grouped: `stacked = factor`. Alternatively, the name of the grouping extra data column can be used:

```
> plot (cluster.meansd,
+         stacked = ".aggregate",
+         fill = ".aggregate",
+         col = cluster.cols)
```

Manually giving yoffset
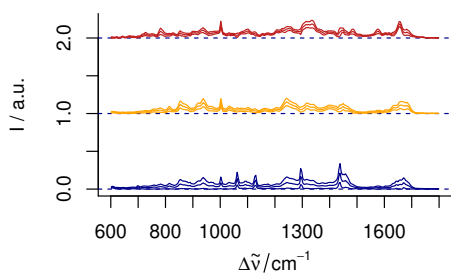
Stacking values can also be given manually as numeric values in *yoffset*:

```
> plotspc (cluster.meansd,
+           yoffset = rep (0:2, each = 3),
+           col = rep (cluster.cols, each = 3))
```

Dense stacking
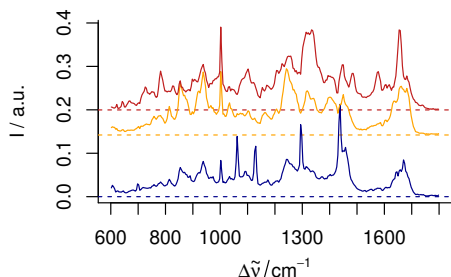
To obtain a denser stacking:

```
> yoffsets <- apply (cluster.means [[]], 2, diff)
> yoffsets <- - apply (yoffsets, 1, min)
> plot (cluster.means, yoffset = c (0, cumsum (yoffsets)),
+         col = cluster.cols)
```

Elaborate example
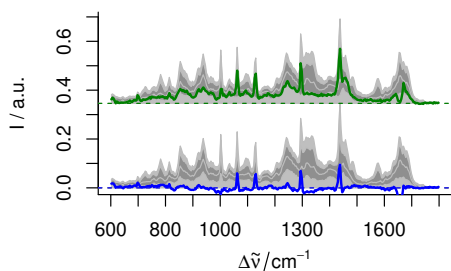

```
> yoffset <- apply (chondro.preproc, 2, quantile, c(0.05, 0.95))
> yoffset <- range (yoffset)
> plot(chondro.preproc[1],
+       plot.args = list (ylim = c (0, 2) * yoffset),
+       lines.args = list( type = "n"))
> yoffset <- (0:1) * diff (yoffset)
> for (i in 1 : 3){
+   plot(chondro.preproc, "spcprctl5", yoffset = yoffset [i],
+         col = "gray", add = TRUE)
+   plot (chondro.preproc [i], yoffset = yoffset [i],
+          col = matlab.dark.palette (3) [i], add = TRUE,
+          lines.args = list (lwd = 2))
+ }
```

`plotspc` allows fine grained customization of almost all aspects of the plot. This is possible by

11

giving arguments to the functions that actually perform the plotting `plot` for setting up the plot area, `lines` for the plotting of the lines, `axis` for the axes, etc. The arguments for these functions should be given in lists as *plot.args*, *lines.args*, *axis.args*, etc.

4 Calibration Plots, (Depth) Profiles, and Time Series Plots

4.1 Calibration plots

Intensities over concentration
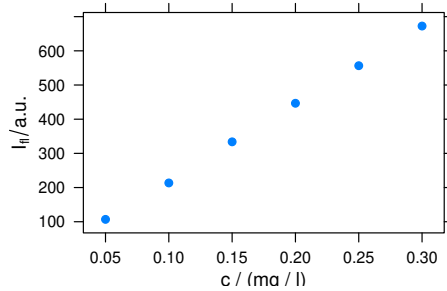
Plotting the Intensities of one wavelength over the concentration for univariate calibration:
```
> plotc (flu [,, 450])
```
The default is to use the first intensity only.

Summary Intensities over concentration
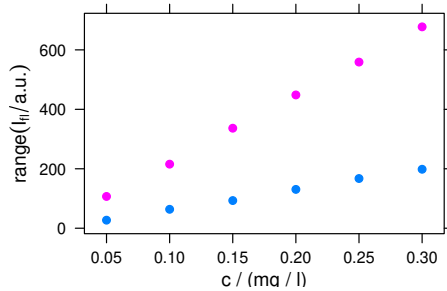
A function to compute a summary of the intensities before drawing can be used:
```
> plotc (flu, func = range, groups = .wavelength)
```
If `func` returns more than one value, the different results are accessible by `.wavelength`.

Conditioning: plotting more traces separately
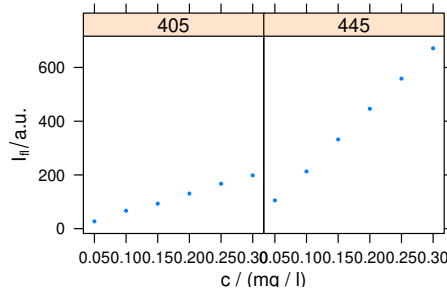

```
> plotc (flu [,, c (405, 445)], spc ~ c | .wavelength,
+        cex = .3, scales = list (alternating = c(1, 1)))
```

Grouping: plot more traces in one panel
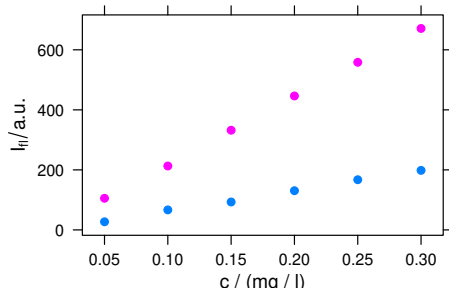

```
> plotc (flu [,, c (405, 445)], groups = .wavelength)
```

Changing Axis Labels (and other parameters)
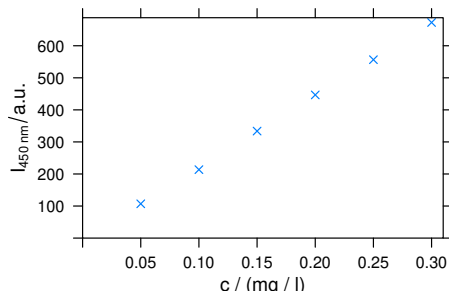
Arguments for `xyplot` can be given to `plotc`:

```
> plotc (flu [,, 450],
+          ylab = expression (I ["450 nm"] / a.u.),
+          xlim = range (0, flu$c + .01),
+          ylim = range (0, flu$spc + 10),
+          pch = 4)
```

Adding things to the plot: customized panel function
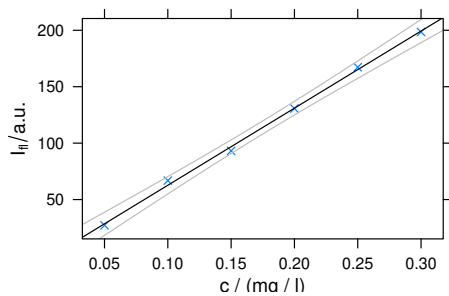
As `plotc` uses the *lattice* function `xyplot`, additions to the plot must be made via the panel function:

```
> panelcalibration <- function (x, y, ..., clim = range (x), level = .
+    panel.xyplot (x, y, ...)
+    lm <- lm (y ~ x)
+    panel.abline (coef (lm), ...)
+    cx <- seq (clim [1], clim [2], length.out = 50)
+    cy <- predict (lm, data.frame (x = cx),
+                   interval = "confidence",
+                   level = level)
+    panel.lines (cx, cy [,2], col = "gray")
+    panel.lines (cx, cy [,3], col = "gray")
+ }
> plotc (flu [,,405], panel = panelcalibration,
+          pch = 4, clim = c (0, 0.35), level = .99)
```

4.2 Time series and other Plots of the Type Intensity-over-Something

13

Abscissae other than c
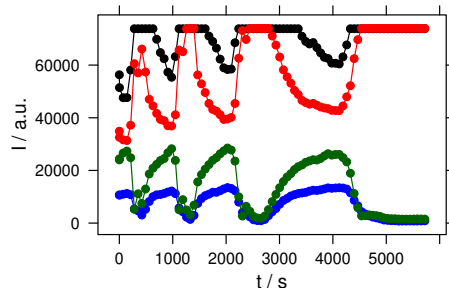
Other abscissae may be specified by explicitly giving the model formula:

```
> plotc (laser [,, c(405.0063, 405.1121, 405.2885, 405.3591)],
+       spc ~ t,
+       groups = .wavelength,
+       type = "b",
+       col = c ("black", "blue", "red", "darkgreen"))
```

5 Levelplot

hyperSpec's *levelplot* can use two special column names:

.wavelength for the wavelengths

.row   for the row index (i.e. spectrum number) in the data

Besides that, it behaves exactly like *levelplot*. Particularly, the data is given as the *second* argument:

levelplot
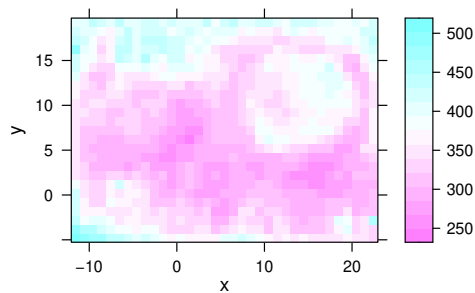

```
> levelplot (spc ~ x * y, chondro)
```

factors as z
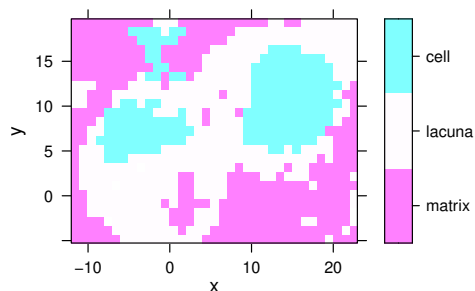
If the colour-coded value is a factor, the display is adjusted to this fact:

```
> levelplot (clusters ~ x * y, chondro)
```

6 Spectra Matrix

It is often useful to plot the spectra against an additional coordinate, e.g. the time for time series, the depth for depth profiles, etc.

This can be done by `plot (object, "mat")`. The actual plotting is done by `image`, but `levelplot`

can produce spectra matrix plots as well and these plots can be grouped or conditioned.

different palette
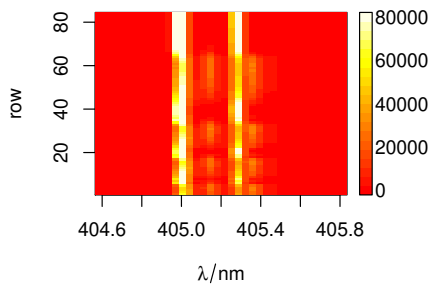

```
> plot (laser, "mat", col = heat.colors (20))
```
is the same as
```
> plotmat (laser, col = heat.colors (20))
```

different y axis
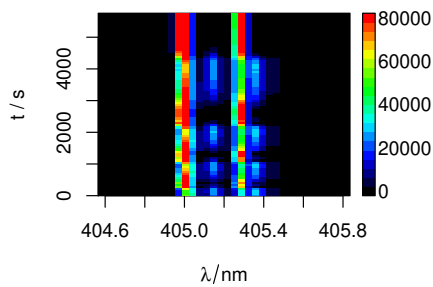
Using a different extra data column for the y axis:
```
> plotmat (laser, y = "t")
```
alternatively, y values and axis label can be given separately.

```
> plotmat (laser, y = laser$t, ylab = labels (laser, "t"))
```

contour lines
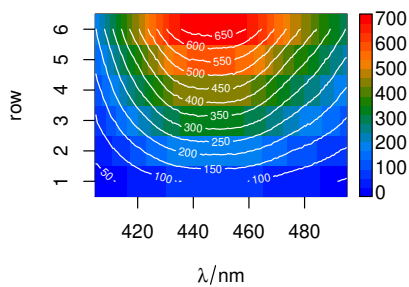
Contour lines may be added:
```
> plotmat (flu, col = matlab.dark.palette (20))
> plotmat (flu, col = "white",
+          contour = TRUE, add = TRUE)
```

colour-coded points: `levelplot` with special panel function
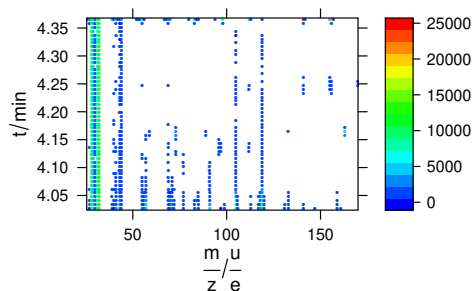

```
> require ("latticeExtra")
> barb <- do.call (collapse, barbiturates[1:50])
> barb <- orderwl (barb)
> levelplot (spc ~ .wavelength * z, barb,
+            panel = panel.levelplot.points,
+            cex = .33, col.symbol = NA,
+            col.regions = matlab.palette)
```

15

7 False-Colour Maps: `plotmap`

`plotmap` is a specialized version of `levelplot`. The spectral intensities may be summarized by a function before plotting (default: `mean`). The same scale is used for x and y axes (*aspect = "iso"*).

plotting map
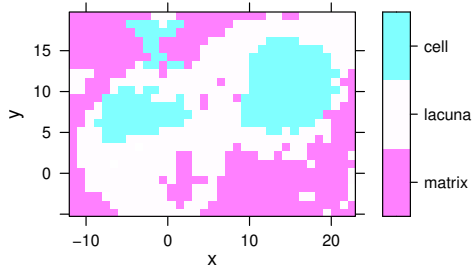

```
> plotmap (chondro)
```

plotting maps with other than x and y
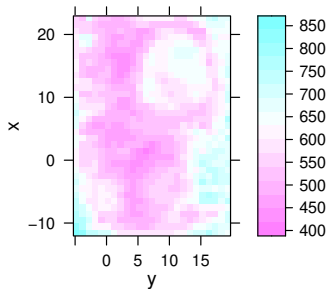
specify the colour-coded variable, abscissa and ordinate as formula: `colour.coded ~ abscissa * ordinate`
```
> plotmap (chondro, spc ~ y * x)
```

colour-coded factors
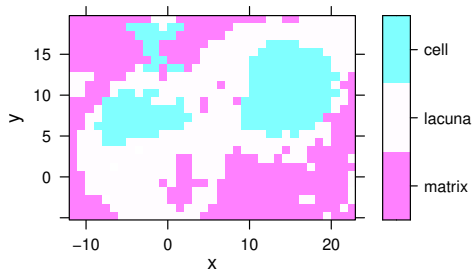

```
> plotmap (chondro, clusters ~ x * y)
```
If the colour-coded variable is a factor, each level gets its own colour, and the legend is labeled accordingly.

different palette
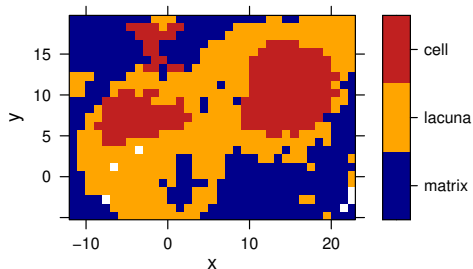
To plot with a different palette, use argument `col.regions`.

```
> plotmap (chondro, clusters ~ x * y,
+                   col.regions = cluster.cols)
```

16

Fine tuning `lattice` parameters

The plotting of color maps is done via R package `lattice` (aka Trellis graphic approach), which is highly customizable. Use `trellis.par.get` and `trellis.par.set` to get/set the settings for the current graphics device.

```
> my.theme = trellis.par.get()
> names(my.theme)        # note how many parameters are tunable

 [1] "grid.pars"          "fontsize"           "background"         "panel.background"
 [5] "clip"               "add.line"           "add.text"           "plot.polygon"
 [9] "box.dot"            "box.rectangle"      "box.umbrella"       "dot.line"
[13] "dot.symbol"         "plot.line"          "plot.symbol"        "reference.line"
[17] "strip.background"   "strip.shingle"      "strip.border"       "superpose.line"
[21] "superpose.symbol"   "superpose.polygon"  "regions"            "shade.colors"
[25] "axis.line"          "axis.text"          "axis.components"    "layout.heights"
[29] "layout.widths"      "box.3d"             "par.xlab.text"      "par.ylab.text"
[33] "par.zlab.text"      "par.main.text"      "par.sub.text"
```

Any of these parameters can be fine-tuned to produce the desired output. For example, parameter `my.theme$region` is responsible for the appearance of color maps, and it contains elements `$alpha` and `$col`. By changing this parameters you can create your own theme for plotting and pass it to the plotting function via `par.settings`.

changed palette
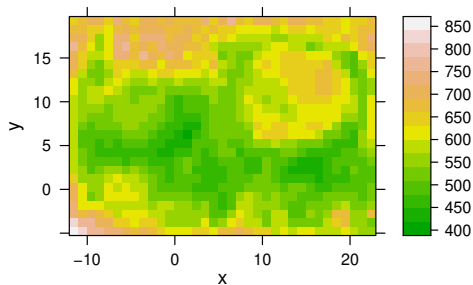
This plot uses a customized lattice theme.
```
> my.theme$regions$col = grDevices::terrain.colors
> plotmap (chondro, par.settings = my.theme)
```

It is possible to **persistently** (i.e. inside of the current R session) set `lattice` parameters, so they would apply to all further plots. This is done via a call to `trellis.par.set`, for example `trellis.par.set(my.theme)`.

The current settings can be visualized via a call to `show.settings()`

show current lattice settings
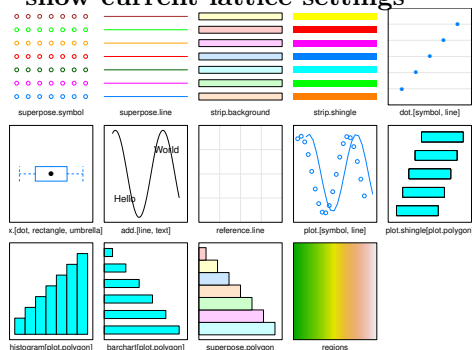

```
> # Display current trellis parameters
> show.settings()
```

An overview of different color palettes, and ways to create your own, can be found in the **R color cheatsheet**.

defined wavelengths
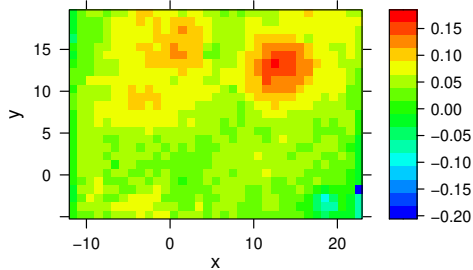
To plot a map of the average intensity at particular wavelengths use extraction:

```
> plotmap (chondro.preproc [, , c(728, 782, 1098,
+                                 1240, 1482, 1577)],
+          col.regions = matlab.palette)
```

Conditioning
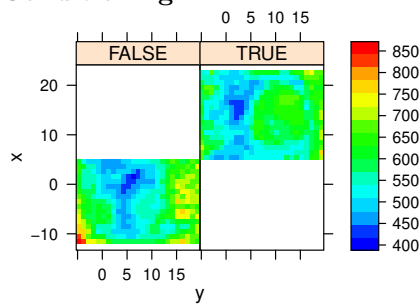

```
> plotmap (chondro,
+          spc ~ y * x | x > 5,
+          col.regions = matlab.palette(20))
```

Conditioning on `.wavelength`
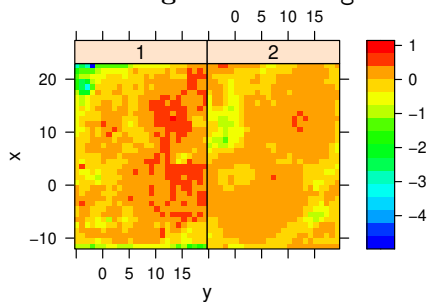
`plotmap` automatically applies the function in *func* before plotting. This defaults to the `mean`. In order to suppress this, use `func = NULL`. This allows conditioning on the wavelengths.

To plot e.g. the first two score maps of a principal component analysis:

```
> pca <- prcomp (~ spc, data = chondro.preproc$.)
> scores <- decomposition (chondro, pca$x,
+                          label.wavelength = "PC",
+                          label.spc = "score /  a.u.")
> plotmap (scores [,,1:2],
+          spc ~ y * x | as.factor(.wavelength),
+          func = NULL,
+          col.regions = matlab.palette(20))
```

Conditioning on .wavelength II
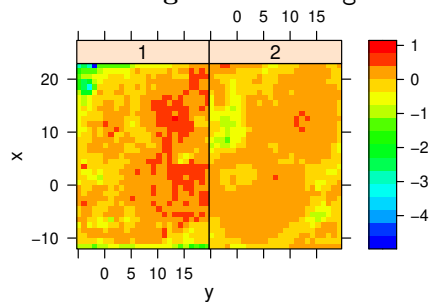
Alternatively, use `levelplot` directly:

```
> levelplot (spc ~ y * x | as.factor(.wavelength),
+            scores [,,1:2],
+            aspect = "iso",
+            col.regions = matlab.palette(20))
```

Voronoi plot
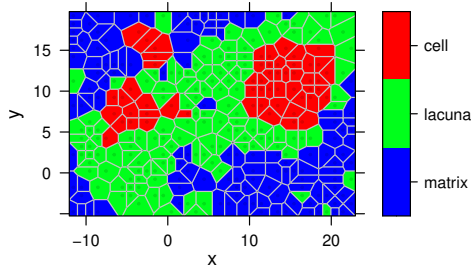

```
> plotvoronoi (sample (chondro, 300), clusters ~ x * y,
+              col.regions = matlab.palette(20))
```
Voronoi uses `panel.voronoi` from *latticeExtra*[2]. The tesselation is calculated by default using *deldir*[3], but *tripack*[4] can also be used. *tripack* seems to faster in general, but may "hang" with certain data sets (particularly regular grids with missing spectra as in this example). Furthermore, it is not FOSS (free and open source software), so users are kindly asked to review *tripack*'s license before using it.

Mark missing spectra
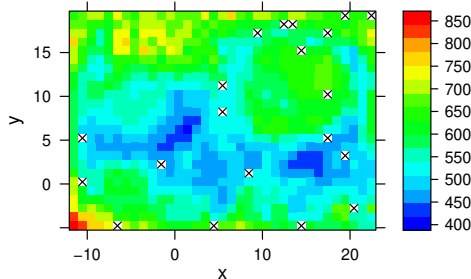
If the spectra come from a rectangular grid, missing positions can be marked with this panel function:

```
> mark.missing <- function (x, y, z, ...){
+    panel.levelplot (x, y, z, ...)
+
+    miss <- expand.grid (x = unique (x), y = unique (y))
+    miss <- merge (miss, data.frame (x, y, TRUE),
+                   all.x = TRUE)
+    miss <- miss [is.na (miss[, 3]),]
+    panel.xyplot (miss [, 1], miss [, 2], pch = 4, ...)
+ }
> plotmap (sample (chondro, length(chondro) - 20),
+          col.regions = matlab.palette(20),
+          col = "black",
+          panel = mark.missing)
```

Unevenly spaced measurement grid

The panel function used by `plotmap` defaults to `panel.levelplot.raster` which assumes an evenly spaced measurement grid. Even if the spectra are measured on a nominally evenly spaced grid, the actual stage position may be slightly varying due to positioning inaccuracy and some manufacturers (e.g. Kaiser) record the position reported by the stage rather than the position requested by the stage control.
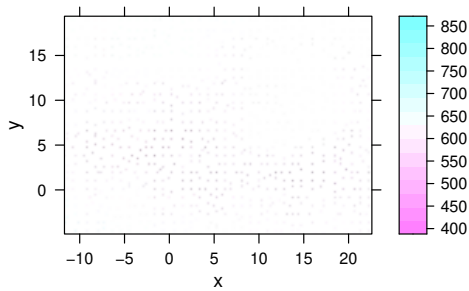
This leads to weird looking output with holes, and possibly wrong columns:

```
> plotmap (uneven)
```

The symptom of this situation are warnings about values in x and/or y not being equispaced; and that the output therefore may be wrong.
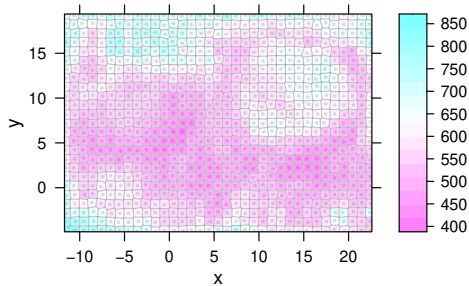
One possibility to obtain a correct map is using `plotvoronoi` instead which will construct a mosaic-like image with the respective "pixel" areas being centered around the actually recorded `$x` and `$y` position:

```
> plotvoronoi (uneven)
```
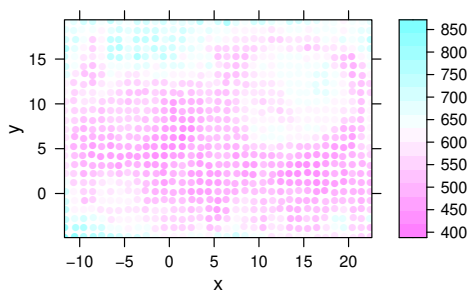
Another possibility that underlines a point shape of the measurements is switching to `latticeExtra::panel.levelplot.points`:

```
> plotmap (uneven, panel = panel.levelplot.points,
+                                cex = 0.75, col.symbol=NA)
```
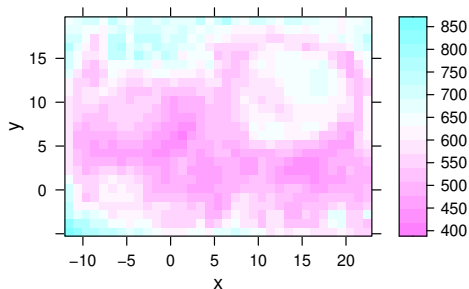
Alternatively, the measurement raster positions can be rounded to their nominal raster, e.g.:

```
> rx <- makeraster (uneven$x, start = -11.55, d = 1, tol = 0.3)
> uneven$x <- rx$x
> ry <- makeraster (uneven$y, start = -4.77, d = 1, tol = 0.3)
> uneven$y <- ry$x
> plotmap (uneven)
```

8 3D plots (with rgl)

rgl[5] offers fast 3d plotting in R. As rgl's axis annotations
are sometimes awkward, they may better be set manually:

```
> laser <- laser [,,404.8 ~ 405.6] / 10000
> laser$t <- laser$t / 3600
> cols <- rep (matlab.palette (nrow (laser)), nwl (laser))
> surface3d (y = wl (laser), x = laser$t,
+            z = laser$spc, col =  cols)
> aspect3d (c (1, 1, 0.25))
> axes3d (c ('x+-', 'y--', 'z--'))
> axes3d ('y--', nticks = 25, labels= FALSE)
> mtext3d ("t / h", 'x+-', line = 2.5)
> mtext3d ("lambda / nm", 'y--', line = 2.5)
> mtext3d ("I / a.u.", edge = 'z--', line = 2.5)
```

Function not available:
package rgl needed.

9 Using ggplot2 with hyperSpec objects

hyperSpec objects do not yet directly support plotting with ggplot2 [6]. Nevertheless, ggplot2 graphics
can easily be obtained, and qplot* equivalents to plotspc and plotmap are defined:

plot spectra with as.long.df
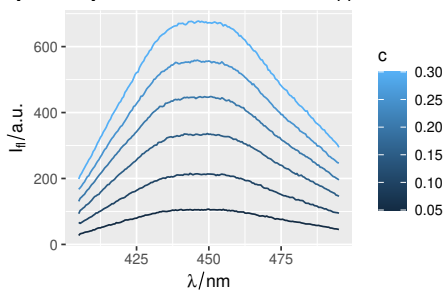

```
> qplotspc (flu) + aes (colour = c)
```

Map with ggplot2
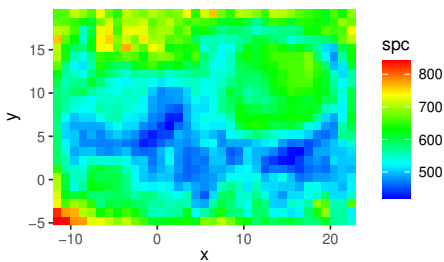

```
> qplotmap (chondro) +
+   scale_fill_gradientn ("spc", colours = matlab.palette ())
```

The two special columns .wavelength
and .rownames contain the wavelength
axis and allow to distinguish the spectra.

For more general plotting, as.long.df transforms a hyperSpec object into a long-form data.frame
that is suitable for qplot, while as.t.df produces a data.frame where each spectrum is one column,
and an additional first column gives the wavelength (see "plotting mean ± sd" below for an example).

21

Long *data.frame*s can be *very* memory consuming as they are of size $nrow \cdot nwl \times (ncol + 2)$ with respect to the dimensions of the *hyperSpec* object. Thus, e.g. the `chondro` data set (2 MB) as *hyperSpec* object) needs 28 MB as long-format *data.frame*. It is therefore highly recommended to calculate the particular data to be plotted beforehand.

Mean ± standard deviation with *ggplot2*
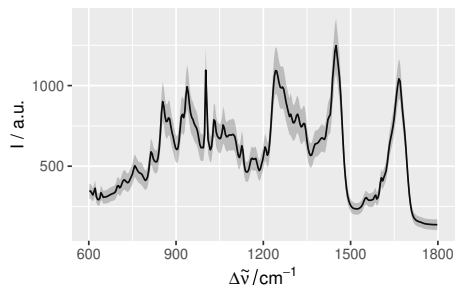

```
> qplotspc (mean (chondro)) +
+ geom_ribbon (aes (ymin = mean + sd,
+                   ymax = mean - sd,
+                   y = 0, group = NA),
+              alpha = 0.25,
+              data = as.t.df (mean_sd (chondro)))
```

Note that `qpotspc` specifies aesthetics `y = spc` and `groups = .rownames`, which do not have corresponding columns in the `data.frame` returned by `as.t.df`. These aesthetics must therefore be set manually in the aesthetics definition in `geom_ribbon` (or any other `geom_` that uses `as.t.df`). Otherwise, errors occur that object `spc` (and/or `.rownames`) cannot be found.

Cut spectra with *ggplot2*

Cut axes can be implemented by faceting:
```
> qplotspc (paracetamol / 1e4,
+           wl.range = c( min ~ 1800, 2800 ~ max)) +
+           scale_x_continuous (breaks = seq (0, 3200, 400))
```

10 Troubleshooting

10.1 No output is produced

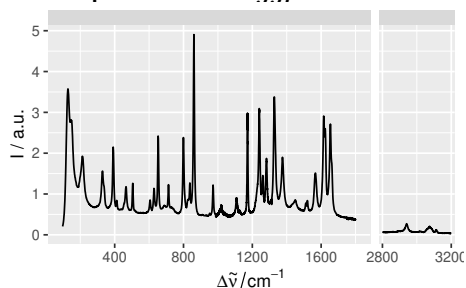`plotmap`, `plotvoronoi`, `levelplot`, and `plotc` use *lattice* functions. Therefore, in loops, functions, Sweave chunks, etc. the lattice object needs to be printed explicitly by `print (plotmap (object))` (R FAQ: Why do lattice/trellis graphics not work?). The same holds for *ggplot2* graphics.

For suggestions how the lattice functions can be redefined so that the result is printed without external print command, see the file `vignettes.defs`.

11 Interactive Graphics

hyperSpec offers basic interaction, `spc.identify` for spectra plots, and `map.identify` and `map.sel.poly` for maps. The first two identify points in spectra plots and map plots, respectively. `map.sel.poly` selects the part of a *hyperSpec* object that lies inside the user defined polygon.

11.1 `spc.identify`: **finding out wavelength, intensity and spectrum**

`spc.identify` allows to measure points in graphics produced by `plotspc`. It works correctly with reversed and cut wavelength axes.

```
> spc.identify (plotspc (paracetamol, wl.range = c (600 ~ 1800, 2800 ~ 3200), xoffset = 800))
```

The result is a data.frame with the indices of the spectra, the wavelength, and its intensity.

11.2 `map.identify`: **finding a spectrum in a map plot**

`map.identify` returns the spectra indices of the clicked points.

```
> map.identify (chondro)
```


11.3 `map.sel.poly`: **selecting spectra inside a polygon in a map plot**

`map.sel.poly` returns a logical indicating which spectra are inside the polygon drawn by the user:

```
> map.sel.poly (chondro)
```


11.4 Related functions provided by base graphics and lattice

For base graphics (as produced by `plotspc`), `locator` may be useful as well. It returns the clicked coordinates. Note that these are *not* transformed according to xoffset & Co.

For lattice graphics, `grid.locator` may be used instead. If it is not called in the panel function, a preceding call to `trellis.focus` is needed:

```
> plot (laser, "mat")
> trellis.focus ()
> grid.locator ()
```

`identify` (or `panel.identify` for lattice graphics) allows to identify points of the plot directly. Note that the returned indices correspond to the plotted object.

References

[1] Lemon J. Plotrix: a package in the red light district of r. *R-News*, 6(4):8–12, 2006.

[2] Deepayan Sarkar and Felix Andrews. *latticeExtra: Extra Graphical Utilities Based on Lattice*, 2019. URL https://CRAN.R-project.org/package=latticeExtra. R package version 0.6-29.

[3] Rolf Turner. *deldir: Delaunay Triangulation and Dirichlet (Voronoi) Tessellation*, 2020. URL https://CRAN.R-project.org/package=deldir. R package version 0.1-25.

[4] Fortran code by R. J. Renka. R functions by Albrecht Gebhardt. With contributions from Stephen Eglen <stephen@anc.ed.ac.uk>, Sergei Zuyev, and Denis White. *tripack: Triangulation of Irregularly Spaced Data*, 2020. URL https://CRAN.R-project.org/package=tripack. R package version 1.3-9.

[5] Daniel Adler, Duncan Murdoch, et al. *rgl: 3D Visualization Using OpenGL*, 2020. URL https://CRAN.R-project.org/package=rgl. R package version 0.100.54.

[6] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Session Info

```
             [,1]
sysname       "Linux"
release       "4.15.0-101-generic"
version       "#102-Ubuntu SMP Mon May 11 10:07:26 UTC 2020"
nodename      "cx17007"
machine       "x86_64"
login         "unknown"
user          "cb"
effective_user "cb"

R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so

locale:
 [1] LC_CTYPE=de_DE.UTF-8        LC_NUMERIC=C               LC_TIME=de_DE.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=de_DE.UTF-8
 [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] tools     grid      stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] latticeExtra_0.6-29    baseline_1.3-0         MASS_7.3-51.6          hyperSpec_0.99-20200527
[5] xml2_1.3.2             ggplot2_3.3.0          lattice_0.20-41

loaded via a namespace (and not attached):
 [1] tidyselect_1.0.0    xfun_0.13           lpSolve_5.6.15      purrr_0.3.4
 [5] limSolve_1.5.6      colorspace_1.4-1    vctrs_0.2.4         testthat_2.3.2
 [9] miniUI_0.1.1.1      htmltools_0.4.0     rlang_0.4.6         manipulateWidget_0.10.1
[13] R.oo_1.23.0         pillar_1.4.4        later_1.0.0         glue_1.4.0
[17] withr_2.2.0         R.utils_2.9.2       RColorBrewer_1.1-2  jpeg_0.1-8.1
[21] R.cache_0.14.0      lifecycle_0.2.0     munsell_0.5.0       gtable_0.3.0
[25] R.methodsS3_1.8.0   htmlwidgets_1.5.1   mvtnorm_1.1-0       labeling_0.3
[29] knitr_1.28          fastmap_1.0.1       SparseM_1.78        httpuv_1.5.2
[33] crosstalk_1.1.0.1   Rcpp_1.0.4.6        xtable_1.8-4        promises_1.1.0
[37] scales_1.1.0        plotrix_3.7-8       webshot_0.5.2       jsonlite_1.6.1
[41] deldir_0.1-25       mime_0.9            farver_2.0.3        R.rsp_0.43.2
[45] png_0.1-7           digest_0.6.25       dplyr_0.8.5         shiny_1.4.0.2
[49] quadprog_1.5-8      magrittr_1.5        lazyeval_0.2.2      tibble_3.0.1
[53] crayon_1.3.4        pkgconfig_2.0.3     ellipsis_0.3.0      assertthat_0.2.1
[57] R6_2.4.1            compiler_3.6.3
```