

hyperSpec Plotting functions

Claudia Beleites <cbeleites@units.it>
CENMAT, DMRN, University of Trieste

November 19, 2010

Vignette under Development

This file is currently undergoing a thorough revision. Changes may happen frequently. Even if the file is not yet nice to read, the shown code does work.

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`.

Contents

1	Predefined functions	2
2	Arguments for plot	3
3	Spectra	6
3.1	Stacked spectra	8
4	Calibration Plots, (Depth) Profiles, and Time Series Plots	10
4.1	Calibration plots	10
4.2	Time series and other Plots of the Type Intensity-over-Something	12
5	Levelplot	12
6	Spectra Matrix	13
7	False-Colour Maps: plotmap	14
8	3 D (with rgl)	17
9	Troubleshooting	17
9.1	No output is produced	17
10	Interactive Graphics	17
10.1	<code>spc.identify</code> : finding out wavelength, intensity and spectrum	18
10.2	<code>map.identify</code> : finding a spectrum in a map plot	18
10.3	Related functions provided by base graphics and lattice	18

For some plots of the `chondro` dataset, the pre-processed spectra are preferred, and their cluster averages \pm one standard deviation:

```
> chondro.preproc <- chondro - spc.fit.poly.below (chondro)

Fitting with npts.min = 15

> chondro.preproc <- sweep (chondro.preproc, 1, mean, "/")
> chondro.preproc <- sweep (chondro.preproc, 2, apply (chondro.preproc, 2, quantile, 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 vignette `vignette ("chondro")`, or the help `? chondro`.

1 Predefined functions

hyperSpec comes with 5 major predefined plotting functions.

`plot` main switchyard for most plotting tasks

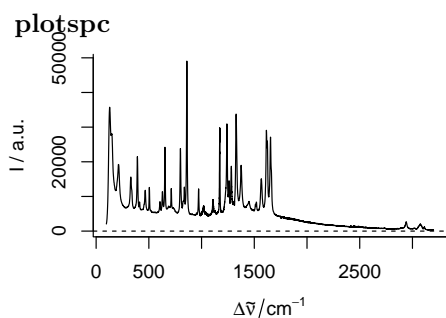
`levelplot` *hyperSpec* has a `levelplot` method

`plotspc` plots spectra

`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 tessellations.
`plotvoronoi` is a *lattice* function



plots the spectra, i.e. the intensities `$spc` over the wavelengths `@wavelength`.

```
> plotspc (paracetamol)
```

plotc

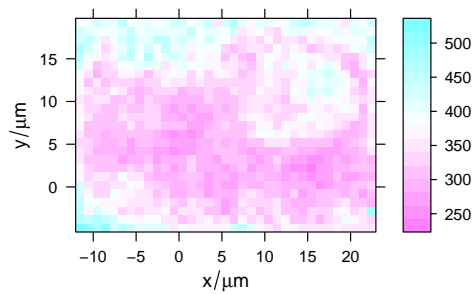


plots an intensity over a single other data column, e.g.

- calibration
- time series
- depth profile

```
> plotc (flu)
```

levelplot

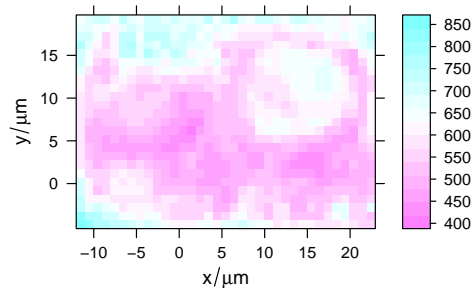


plots a false colour map, defined by a formula.

```
> levelplot ( spc ~ x * y, chondro, aspect = "iso")
```

Warning: Only first wavelength is used for plotting

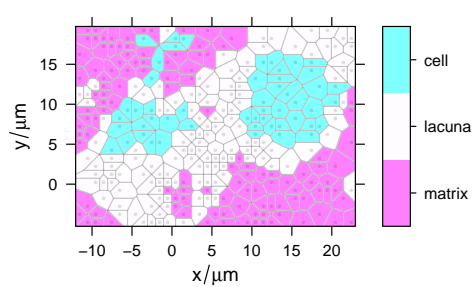
plotmap



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



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)

```
> plotvoronoi (sample (chondro, 300), clusters ~ x * y)
deldir 0.0-12
```

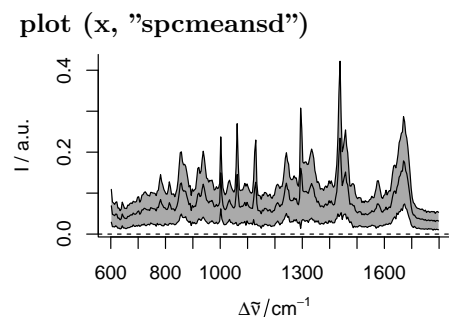
Please note: The process for determining duplicated points has changed from that used in version 0.0-9 (and previously).

2 Arguments for plot

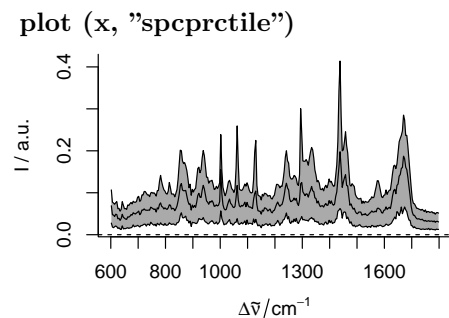
hyperSpec's `plot` method uses its second argument to determine which of the specialized plots to produce. This allows some handy abbreviations. All further arguments are handed over to the function actually producing the plot.



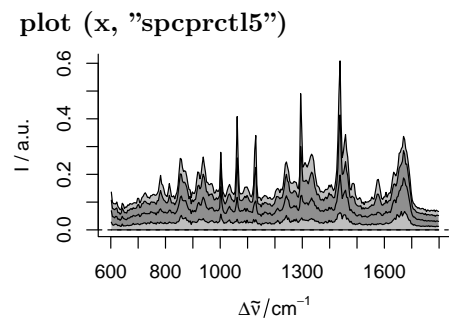
is equivalent to `plotspc (flu)`
`> plot (flu, "spc")`



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

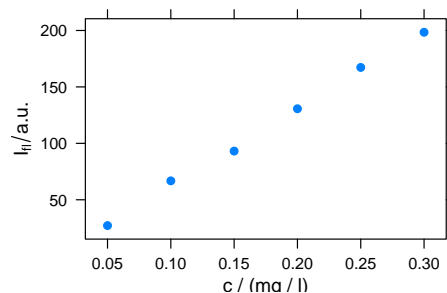


plots median, 16th and 84th percentile for each wavelength. For Gaussian distributed data, 16th, 50th and 84th percentile are equal to mean \pm standard deviation. Spectroscopic data frequently are not Gaussian distributed. The percentiles give a better idea of the true distribution. They are also less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



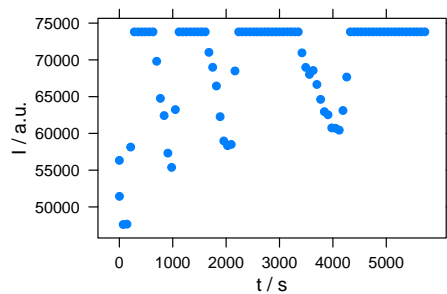
like "spcprctl" plus 5th and 95th percentile.
`> plot (chondro.preproc, "spcprctl5")`

plot (x, "c")



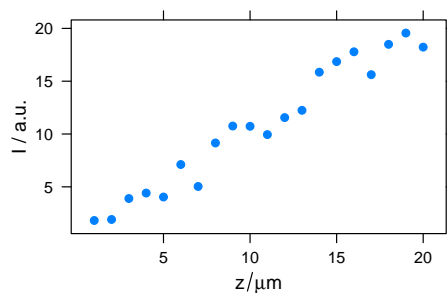
`> plot (flu, "c")`
is equivalent to `plotc (flu)`

plot (x, "ts")



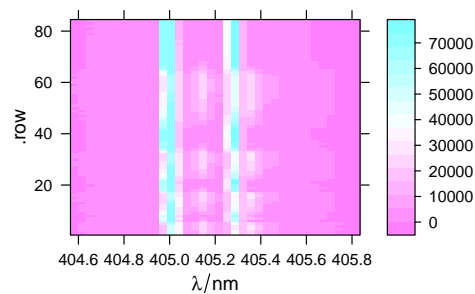
plots a time series plot
`> plot (laser [, 405], "ts")`
equivalent to `plotc (laser, spc ~ t)`

plot (x, "depth")

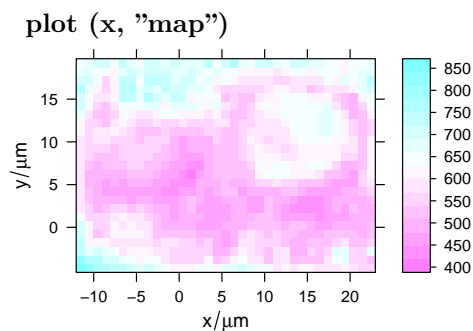


plots a depth profile plot
`> depth.profile <- new ("hyperSpec",`
+ `spc = as.matrix (rnorm (20) + 1:20),`
+ `data = data.frame (z = 1 : 20),`
+ `label = list (spc = "I / a.u.",`
+ `z = expression (z, mu*m),`
+ `.wavelength = expression (lambda)))`
`> plot (depth.profile, "depth")`
the same as `plotc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

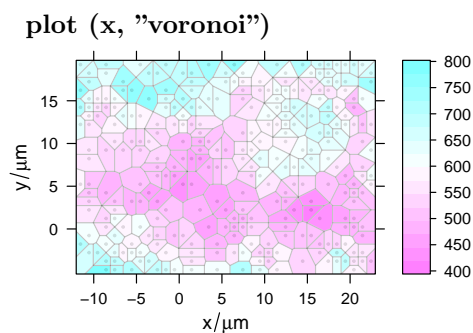
plot (x, "mat")



plots the spectra matrix.
`> plot (laser, "mat")`
Equivalent to
`> levelplot (spc ~ .wavelength * .row, laser)`



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

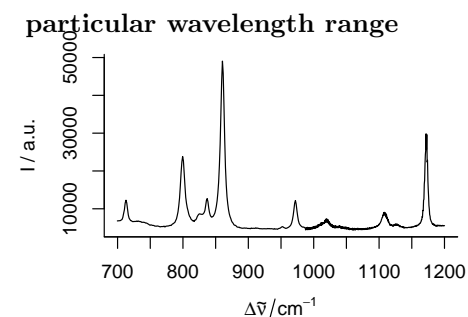


`> plot (sample (chondro, 300), "voronoi")`
 See `plotvoronoi`

3 Spectra

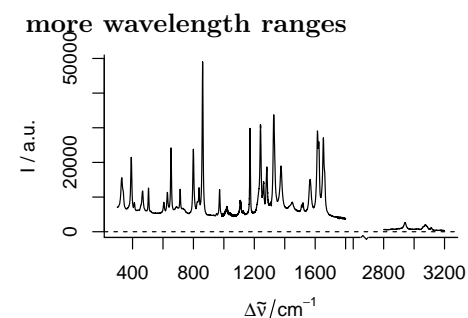
`plotspc`

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



if only one wavelength range is needed, the `extract` command is handiest:

`> plotspc (paracetamol [, , 700 ~ 1200])`
 If `wl.range` already contains indices: use `wl.index = TRUE`.

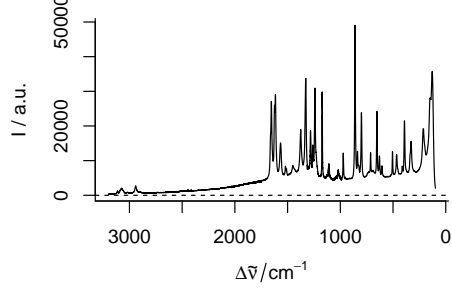


use `wl.range = list (600 ~ 1800, 2800 ~ 3100)`. Cut the wavelength axis appropriately with `xoffset = 750`

`> plotspc (paracetamol,`
`+ wl.range = c (300 ~ 1800, 2800 ~ max),`
`+ xoffset = 750)`

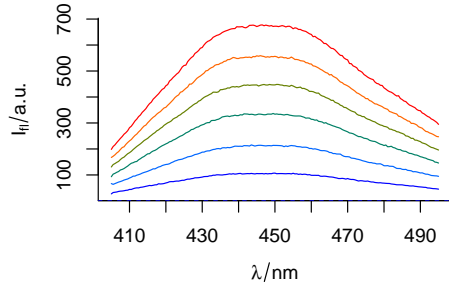
If available, the package *plotrix* is used to produce the cut mark.

with reversed abscissa



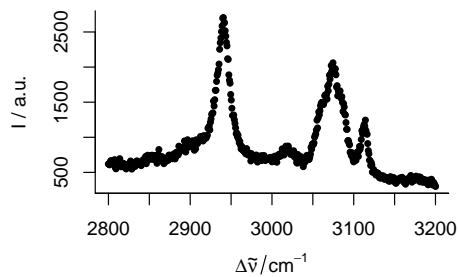
```
use wl.reverse = TRUE
> plotspc (paracetamol, wl.reverse = TRUE )
```

in different colours



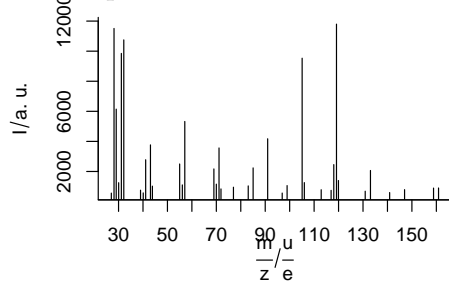
```
use col = vector.of.colours
> plotspc (flu, col = matlab.dark.palette (6))
```

dots instead of lines



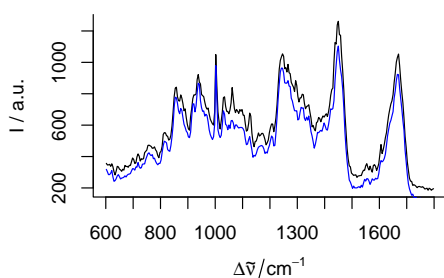
```
use lines.args = list (pch = 20, type = "p")
> plotspc (paracetamol [, 2800 ~ 3200],
+         lines.args = list (pch = 20, type = "p"))
```

mass spectra



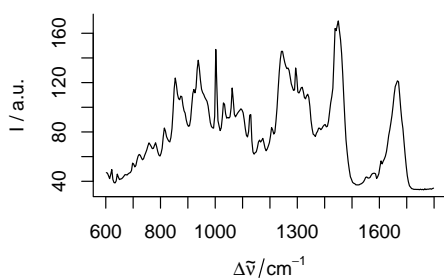
```
use lines.args = list (type = "h")
> plot (barbituates [[1]], lines.args = list (type = "h"))
```

more spectra into an existing plot



```
use add = TRUE
> plotspc (chondro [ 30,,])
> plotspc (chondro [300,,], add = TRUE, col = "blue")
```

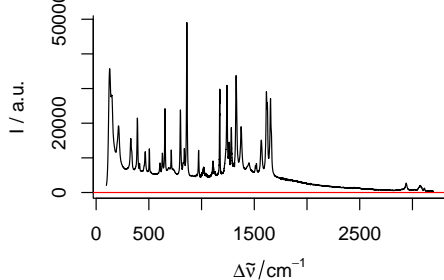
Summary characteristics



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

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

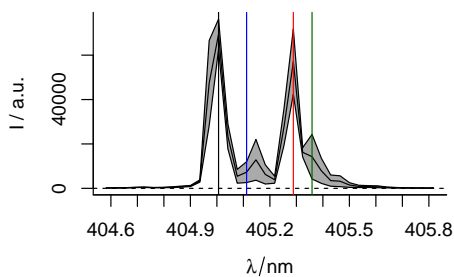
with different line at $I = 0$



zeroline takes a list with parameters to *abline*, NULL suppresses the line.

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

adding to a spectra plot

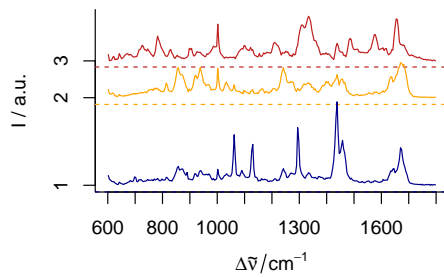


plotspc uses base graphics. After plotting the spectra, more content may be added to the graphic by *abline*, *lines*, *points*, etc.

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

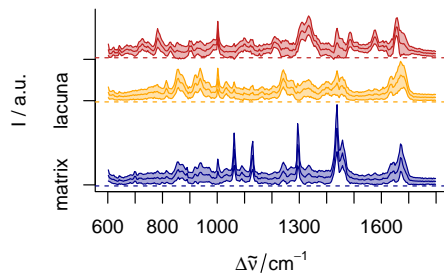
3.1 Stacked spectra

stacked = TRUE



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

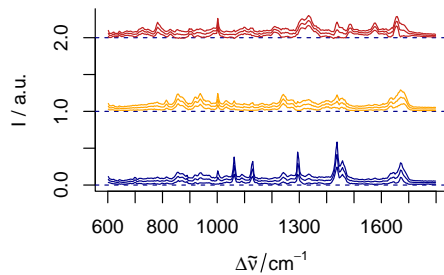
Stacking groups of spectra



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)
```

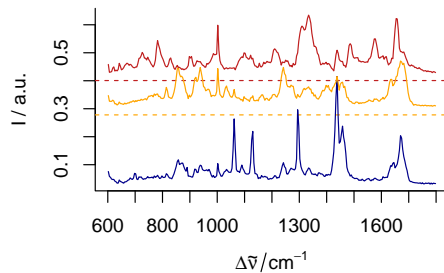
Manually giving yoffset



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))
```

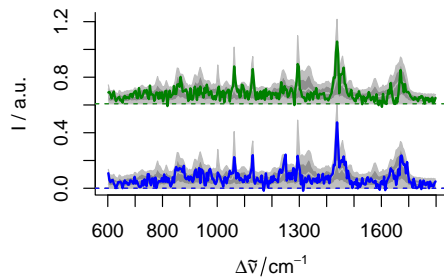
Dense stacking



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



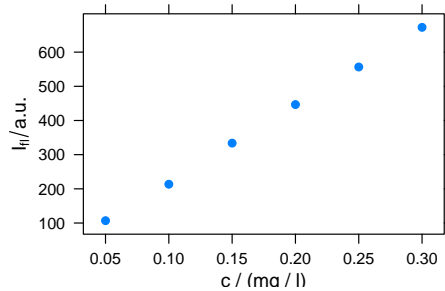
```
> yoffset <- apply (chondro.preproc, 2, quantile, c(0.05, 0.95))
> yoffset <- range (yoffset)
> plot(chondro.preproc[1],
+       plot.args = list (ylim = c (0, 2) * yoffset),
+       lines.args = list( type = "n"))
> yoffset <- (0:1) * diff (yoffset)
> for (i in 1 : 3){
+   plot(chondro.preproc, "spcprct15", 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))
+ }
```

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

plotc

4.1 Calibration plots

Intensities over concentration

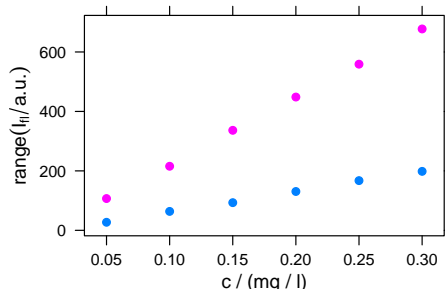


Plotting the Intensities of one wavelength over the concentration for univariate calibration:

```
> 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



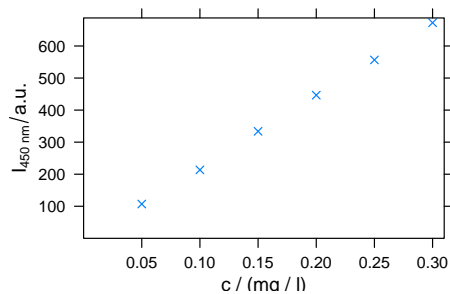
```
> plotc (flu [, c (405, 445)], spc ~ c | .wavelength,
+       cex = .3, scales = list (alternating = c(1, 1)))
```

Grouping: plot more traces in one panel



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

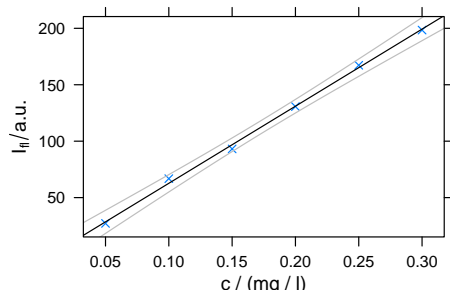
Changing Axis Labels (and other parameters)



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

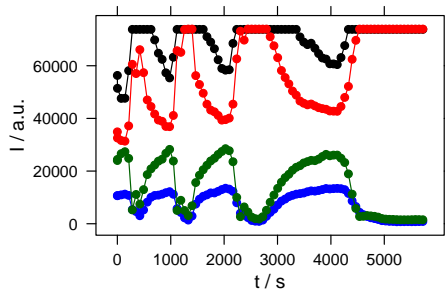


As `plotc` uses the *lattice* function `xyplot`, additions to the plot must be made via the panel function:

```
> panelcalibration <- function (x, y, ..., clim = range (x), level = .99) {
+   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

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

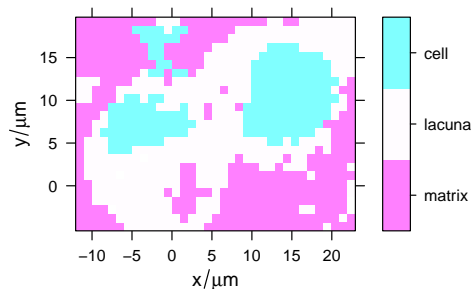
Besides that, it behaves exactly like *levelplot*. Particularly, the data is given as the *second* argument:

levelplot



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

factors as z



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

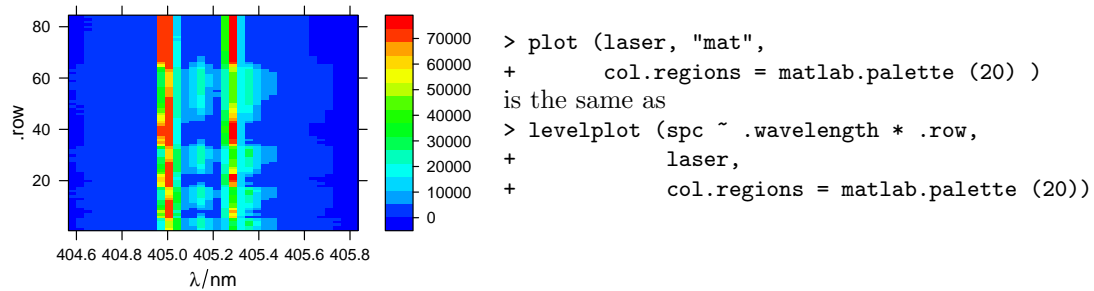
```
> levelplot (clusters ~ x * y, chondro)
```

6 Spectra Matrix

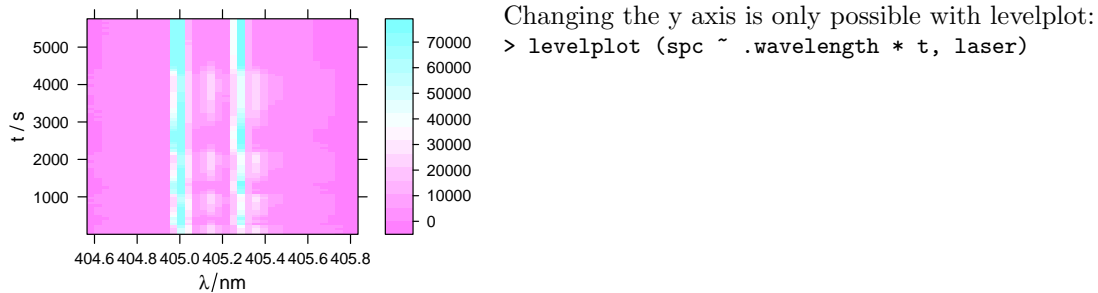
It is often useful to plot the spectra against an additional coordinate, e.g. the time for time series, the depth for depth profiles, etc.

This can be done by `plot (object, "mat")`. The actual plotting is done by `levelplot`, so the plots can be grouped or conditioned.

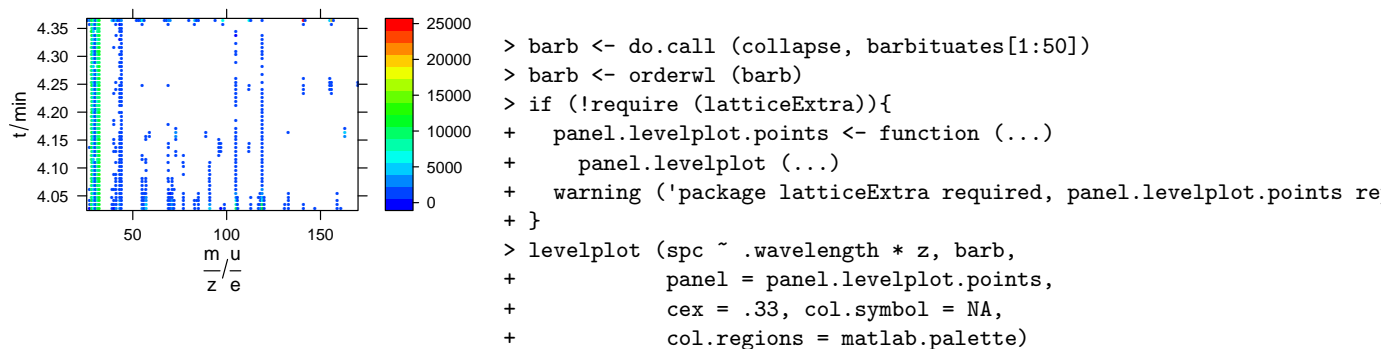
different palette



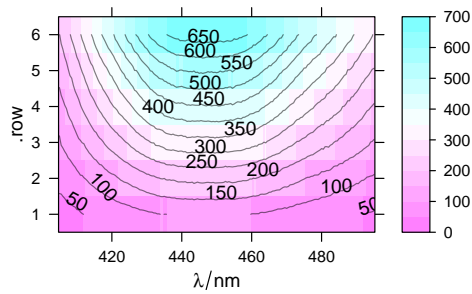
different y axis



colour-coded points: different panel function



contour lines



Contour lines may be added to all `levelplot` based plots:

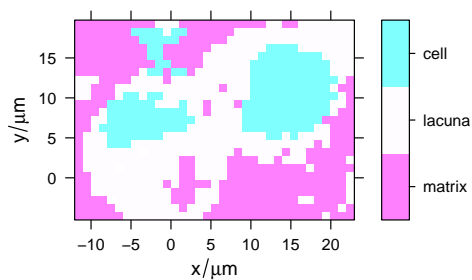
```
> plot (flu, "mat",
+       contour = TRUE,
+       labels = TRUE,
+       col = "#00000080",
+       at = seq (0, 700, by = 50))
```

7 False-Colour Maps: plotmap

`plotmap` uses `levelplot`, a *lattice* function. 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?](#)).

`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 `trellis.args= list (col.regions = palette)`.

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

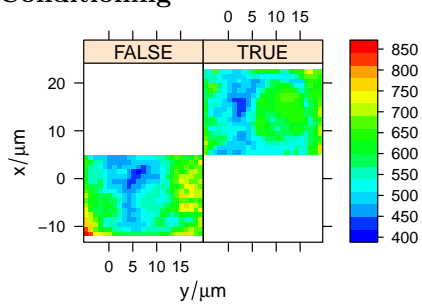
defined wavelengths



To plot a map with particular wavelengths use this:

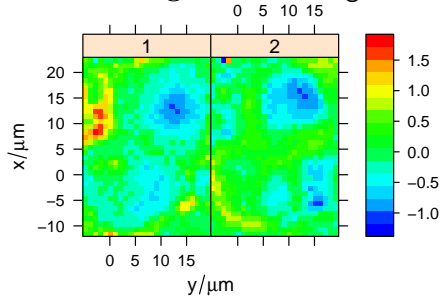
```
> plotmap (chondro.preproc [, , c(728, 782, 1098,
+                               1240, 1482, 1577)],
+         col.regions = matlab.palette)
```

Conditioning



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

Conditioning on .wavelength

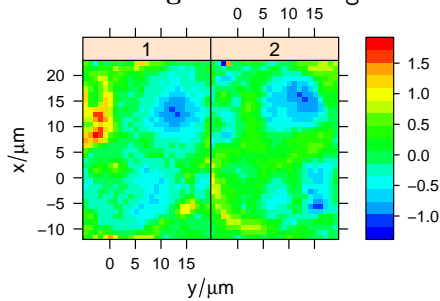


plotmap automatically applies the function in *func* before plotting. This defaults to the *mean*. In order to suppress this, use *func* = *NULL*. This allows conditioning on the wavelengths.

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))
```

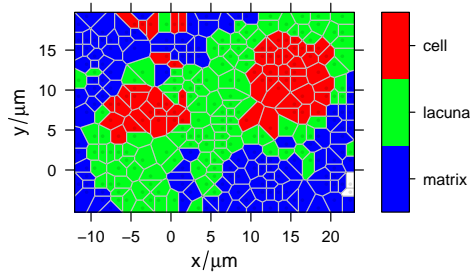
Conditioning on .wavelength II



Alternatively, use *levelplot* directly:

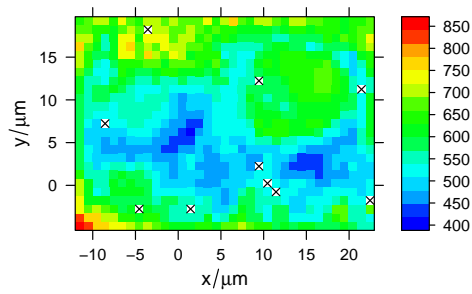
```
> levelplot (spc ~ y * x | as.factor(.wavelength),
+            scores [,1:2],
+            aspect = "iso",
+            col.regions = matlab.palette(20))
```

Voronoi plot



```
> plotvoronoi (sample (chondro, 300), clusters ~ x * y,
+              col.regions = matlab.palette(20))
```


Mark missing spectra I

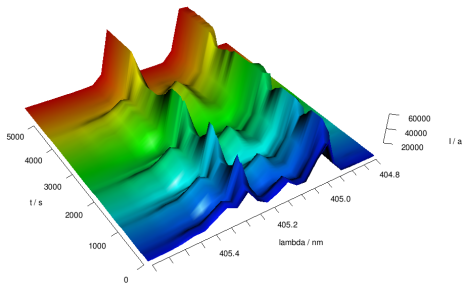


If the spectra come from a rectangular grid, missing positions can be marked with this panel function:

```
> 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, 865),
+          col.regions = matlab.palette(20),
+          col = "black",
+          panel = mark.missing)
```

8 3 D (with rgl)

3D plots with *rgl*



rgl offers fast 3d plotting in R. As *rgl*'s axis annotations are sometimes awkward. They may better be set manually:

```
> library (rgl)
> laser <- laser [,404.8 ~ 405.6]
> 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 / s", 'x+-', line = 2)
> mtext3d("lambda / nm", 'y--', line = 2)
> mtext3d("I / a.u.", edge = 'z--', line = 2.5)
```

9 Troubleshooting

9.1 No output is produced

`plotmap` and `plotc` use `levelplot`, a *lattice* function. 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?](#)).

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

10 Interactive Graphics

hyperSpec offers two basic interaction functions, `spc.identify`, and `map.identify`. They identify

points in spectra plots and map plots, respectively.

10.1 `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.

10.2 `map.identify`: finding a spectrum in a map plot

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

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

10.3 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.

10.4 Interactively changing graphics

hyperSpec's lattice functions work with *playwith* and *latticeist*. These packages allow easy customization of the plots and also identification of points.