hyperSpec Plotting functions

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

May 11, 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`.

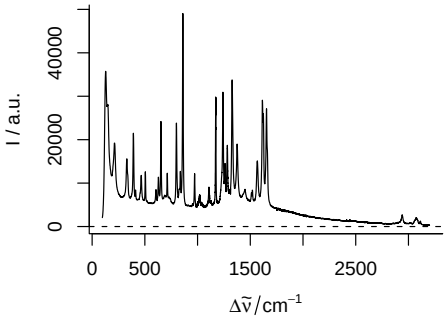
Contents

1 Predefined functions

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
plotmap	version of levelplot allowing some more preprocessing of the data.

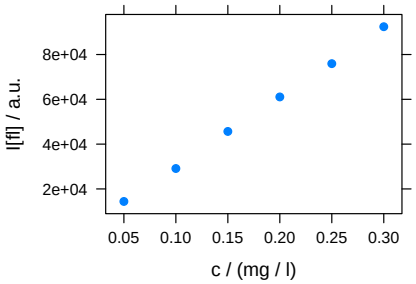
plotspc



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

```
> plotspc (paracetamol)
```

plotc

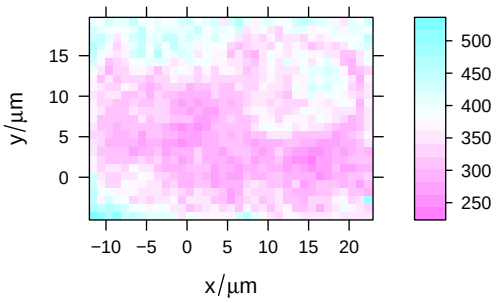


plots an intensity over a single other data column. e. g.

- time series
- calibration
- depth profile

```
> plotc (flu)
```

levelplot

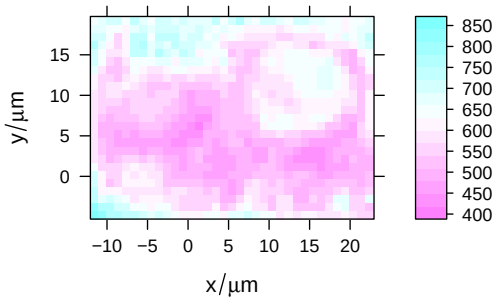


plots a false colour map, defined by a formula.

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

Warning: only the first wavelength is used.

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)
```

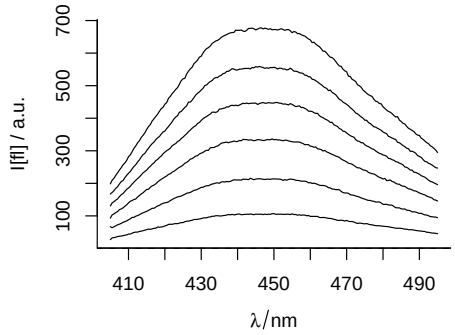
2 Arguments for plot

The three specialized functions are also accessible via *plot*:

hyperSpec's `plot` method uses the second argument to determine which of the three specialized plot functions to call. All further arguments are handed over to this function.

This allows a few more handy abbreviations.

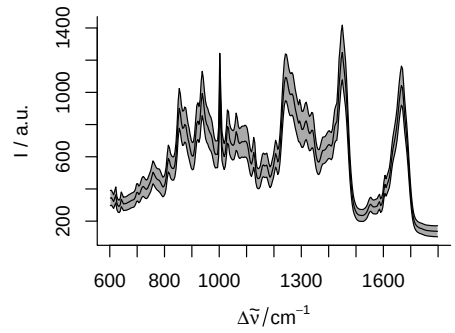
plot (x, "spc")



is equivalent to `plotspc (flu)`

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

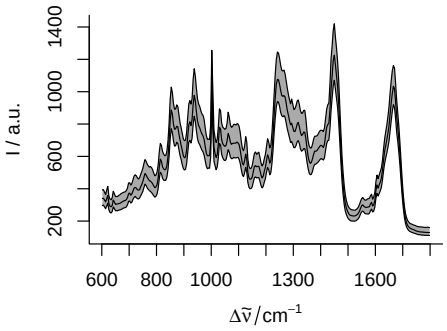
plot (x, "spcmeansd")



plots mean spectrum ± 1 standard deviation

```
> plot (chondro, "spcmeansd")
```

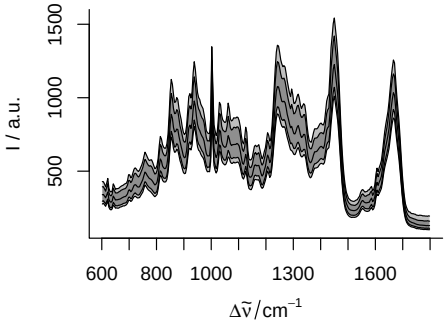
plot (x, "spcprctile")



plots median, 16^{th} and 84^{th} percentile for each wavelength. For Gaussian distributed data, 16^{th}, 50^{th} and 84^{th} 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, "spcprctile")
```

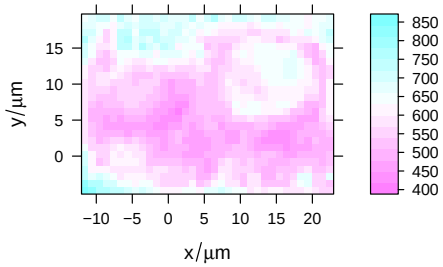
plot (x, "spcprctl5")



like `"spcprctl"` plus 5^{th} and 95^{th}percentile.

```
> plot (chondro, "spcprctl5")
```

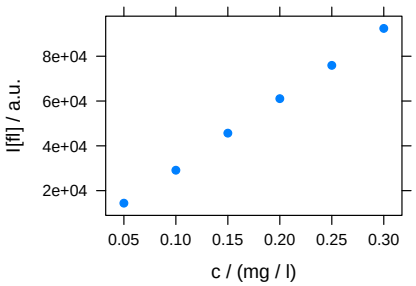
plot (x, "map")



is equivalent to `plotmap (chondro)`

```
> plot (chondro, "map")
```

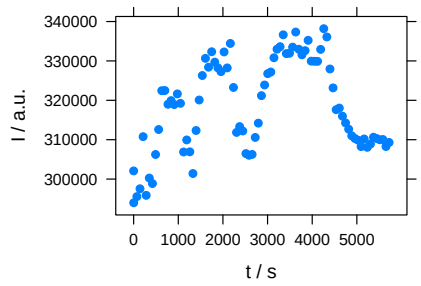
plot (x, "c")



is equivalent to `plotc (flu)`

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

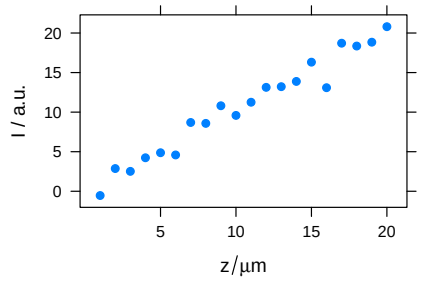
plot (x, "ts")



plots a time series plot, equivalent to `plotc (laser, spc ~ t)`

```
> plot (laser, "ts")
```

plot (x, "depth")

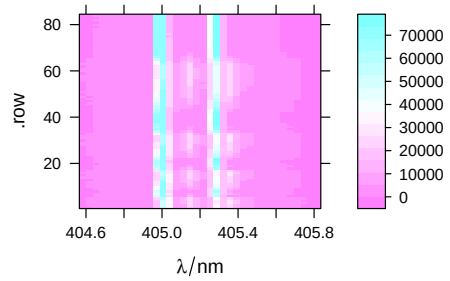


plots a depth profile plot, equivalent to `plotc (laser, spc ~ z)`

```
> 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")
```

plot (x, "mat")



plots the spectra matrix.

```
> plot (laser, "mat")
```

Equivalent to

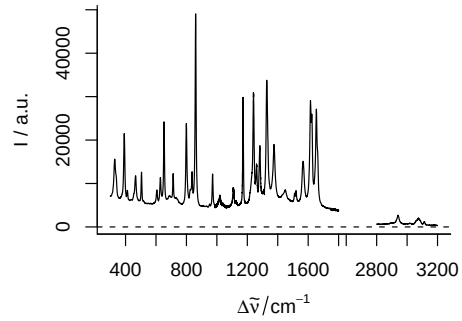
```
> levelplot (spc ~ .wavelength * .row, laser)
```

3 Spectra

plotspc

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

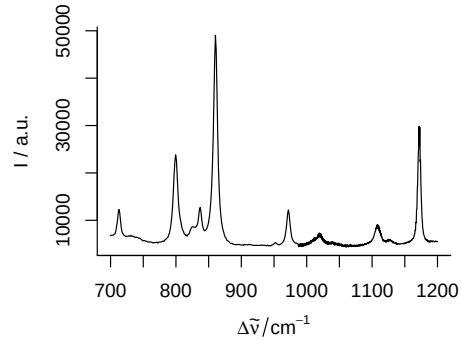
particular wavelength ranges



use `wl.range = list (600 ~ 1800, 2800 ~ 3100)`. If `wl.range` already contains indices: use `wl.index = TRUE`. Cut the wavelength axis appropriately with `xoffset = 750`

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

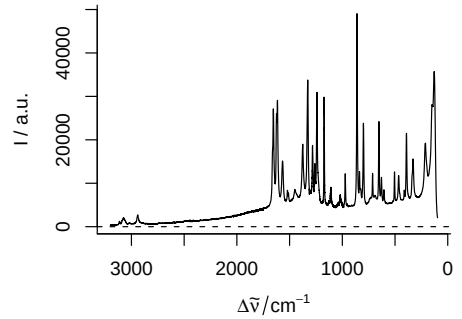
particular wavelength ranges II



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

```
> plotspc (paracetamol[,, 700 ~ 1200])
```

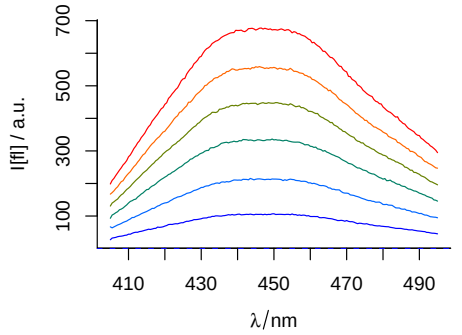
with reversed abscissa



use `wl.reverse = TRUE`

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

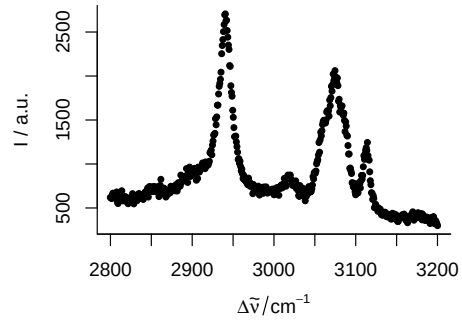
in different colours



use `col = vector.of.colours`

```
> plotspc (flu, col = matlab.dark.palette(6) )
```

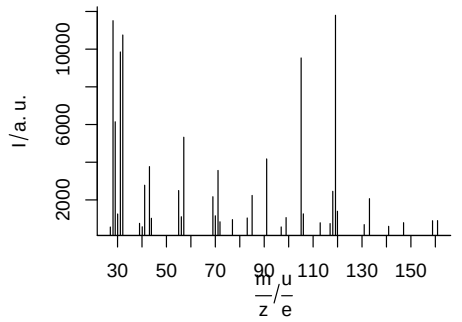
dots instead of lines



use `lines.args = list (pch = 20, type = "p")`

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

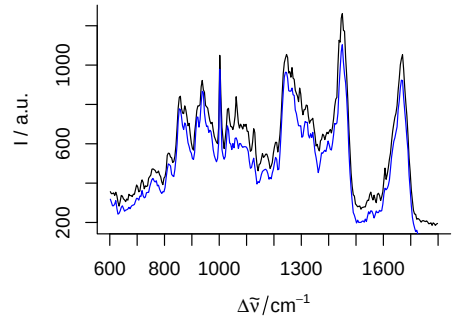
mass spectra



use `lines.args = list (type = "h")`

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

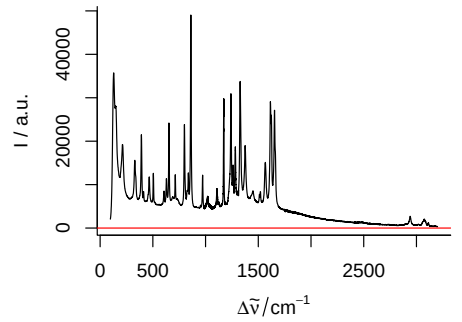
more spectra into an exsisting plot



use *add = TRUE*

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

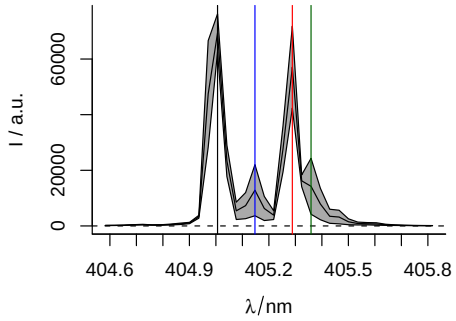
with different line at $I = 0$



use `zeroline = list.of.arguments.to.abline`. `NULL` suppresses the line.

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

adding lines

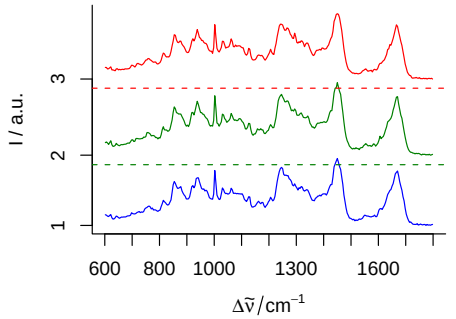


use `abline` for adding lines

```
> plot (laser, "spcmeansd")
> abline (v = wl (laser)[c (13, 17, 21, 23)],
+         col = c("black", "blue", "red", "darkgreen") )
```

3.1 Stacked spectra

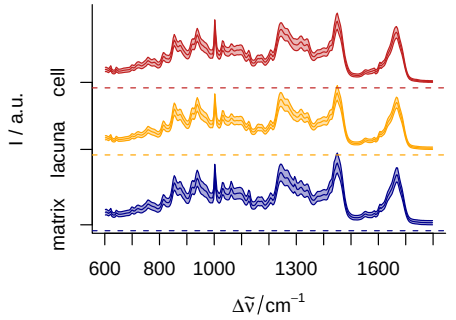
stacked = TRUE



use *stacked = TRUE*

```
> plotspc (chondro[1:3,,],
+          col = matlab.dark.palette (3),
+          stacked = TRUE)
```

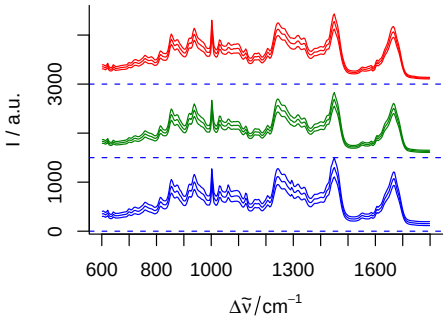
Stacking groups of spectra



if you want to plot mean spectrum ± 1 standard deviation of each of the clusters, use *stacked = grouping.column ".name"*

```
> cols <- c ("dark blue", "orange", "#C02020")
> cluster.means <- aggregate (chondro,
+                             chondro$clusters,
+                             mean_pm_sd)
> plot (cluster.means,
+       stacked = ".aggregate",
+       fill = ".aggregate",
+       col = cols)
```

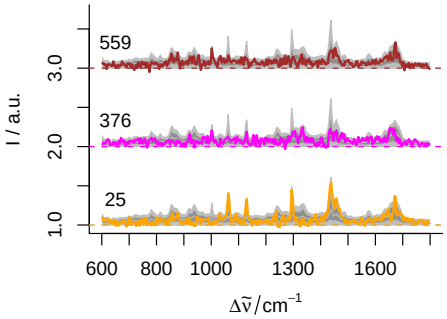
yoffset



or use *yoffset* =

```
> cluster.means <- aggregate (chondro,
+                             chondro$clusters,
+                             mean_pm_sd)
> plotspc (cluster.means,
+          yoffset = rep (c (0,1500,3000), each = 3),
+          col = rep (matlab.dark.palette (3), each = 3))
```

Manually with normalized spectra



it's also possible to normalize and stack the spectra like this:

```
> out <- c (25, 376, 559)
> outcols <- c ("orange", "magenta", "brown")
```

Preprocessing of the spectra:

```
> bl <- spc.fit.poly.below (chondro)
Fitting with npts.min =  15
> chondro <- chondro - bl
> chondro <- sweep (chondro, 1, mean, "/")
> chondro <- sweep (chondro,
+                   2,
+                   apply (chondro,
+                          2,
+                          quantile,
+                          0.05),
+                   "-")
```

The figure:

```
> ## coordinate system:
> plot(chondro[1],
+      plot.args = list (ylim = c (1, length (out) + .7)),
+      lines.args = list( type = "n")
+      )
> ## stacked spectra:
> for (i in seq (along = out)){
+   plot(chondro,
+        "spcprctl5",
+        yoffset = i,
+        col = "gray",
+        add = TRUE)
+   plot (chondro [out[i]],
+         yoffset = i,
+         col = outcols[i],
+         add = TRUE,
+         lines.args = list (lwd = 2))
+   text (650, i + .33, out [i]) }
```

4 Levelplot

Levelplot can use two special column names:

.wavelength for the wavelengths

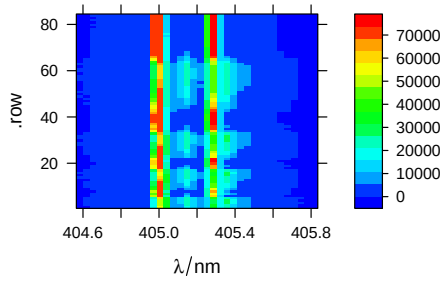
.row for the row index (i. e. spectrum number) in the data

5 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")` or `levelplot (model = spc ~.wavelength * other.data.column, object)`. The actual plotting is done by `levelplot`, so the plots can be grouped or conditioned.

different palette

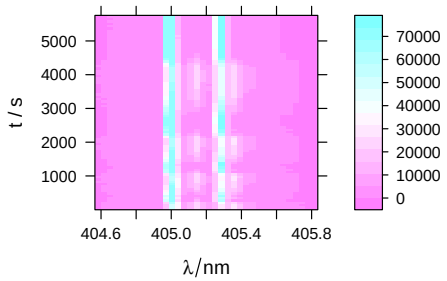



```
> plot (laser, "mat",
+       col.regions = matlab.palette (20) )
```

is the same as

```
> levelplot (spc ~ .wavelength * .row,
+            laser,
+            col.regions = matlab.palette (20))
```

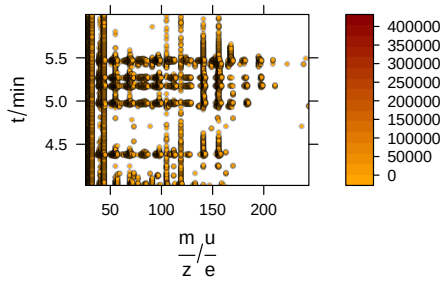
different y axis



Changing the y axis is only possible with levelplot:

```
> levelplot (spc ~ .wavelength * t,
+            laser)
```

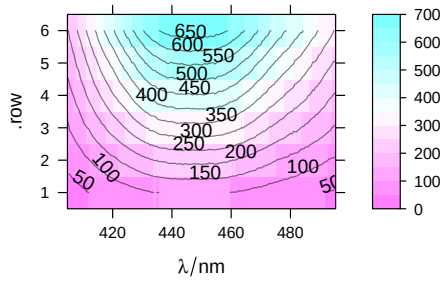
different panel




```
> class (barbituates)
[1] "list"
> barbituates <- do.call (collapse, barbituates)
> barbituates <- orderwl (barbituates)


> levelplot (spc ~ .wavelength * z, barbituates,
+            panel = panel.levelplot.points,
+            cex = .5,
+            col.regions = colorRampPalette (c ("orange", "darkred
+            )
```

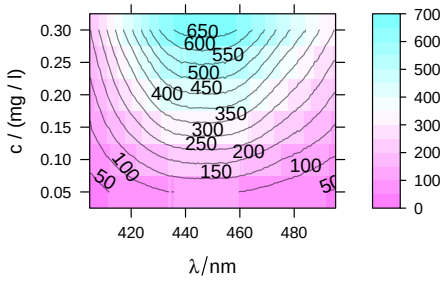
contour lines with plot



Contourplots are possible with plot and levelplot:

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

contour lines with levelplot



Contourplot with a different y axis:

```
> levelplot (spc ~ .wavelength * c,
+            flu,
+            contour = TRUE,
+            labels = TRUE,
+            col = "#00000080",
+            at = seq (0, 700, by = 50))
```

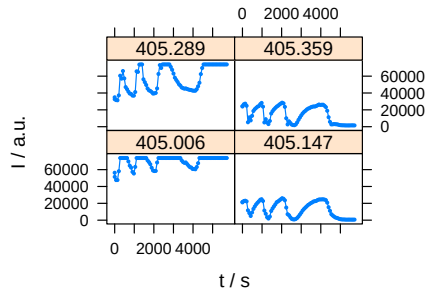
6 Calibration Plots, (Depth) Profiles, and Time Series Plots

plotc

`plotc` plots an intensity over one of the extra data columns. The abscissa uses column `$c` by default, another column can be specified using a proper formula. The ordinate can be calculated as a sum characteristic (with parameter *func*= `function`, defaulting to *sum*).

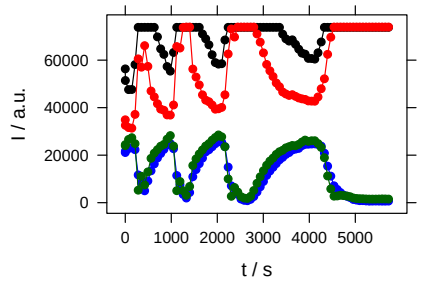
6.1 Time series

Grouping with `plotc`




```
> plotc (laser [,, c (13, 17, 21, 23),
+               wl.index = TRUE],
+        spc ~ t | .wavelength,
+        type = "b",
+        cex = .3)
```

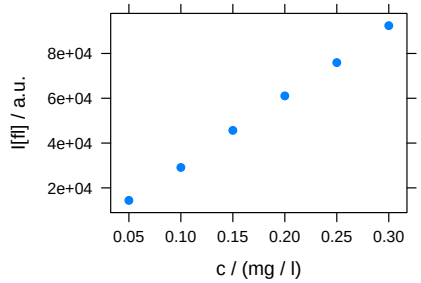
In one panel




```
>  plotc (laser [,, c (13, 17, 21, 23),
+                wl.index = TRUE],
+         spc ~ t,
+         groups = .wavelength,
+         type = "b",
+         col = c ("black", "blue", "red", "darkgreen"))
```

6.2 Calibration plots

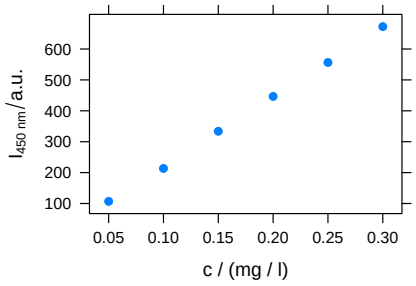
Intensities over concentration



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

```
> plotc (flu)
```

Changing Axis Labels (and other parameters)



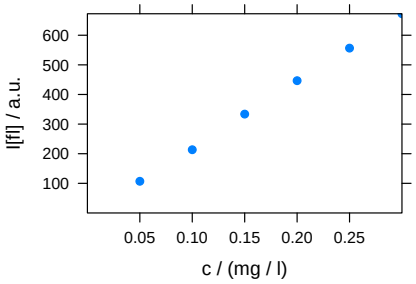
Alternative with different label for y-axis:

```
> flu <- flu[,,450]
```

```
> plotc (flu, ylab = expression (I ["450 nm"] / a.u.))
```

Arguments for `xyplot` can be given to `plotc`.

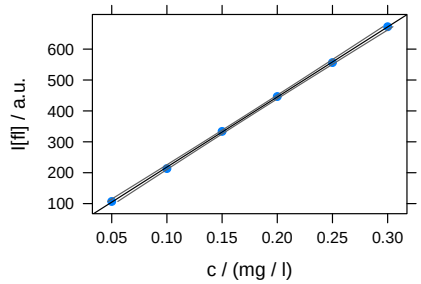
Axes starting at origin



Use `xlim` and `ylim` for setting the origin

```
> plotc (flu,
+        xlim = range (0, flu$c),
+        ylim = range (0, flu$spc))
```

Calibration - customized panel function



The calibration model:

```
> calibration <- lm (c ~ spc, data = flu$.)
```

Prediction for new measurements with e. g. an intensity of 125 or 400 units:

```
> I <- c (125, 400)
> conc <- predict (calibration,
+                  newdata = list (spc = as.matrix (I)),
+                  interval = "prediction",
+                  level = .99)
```

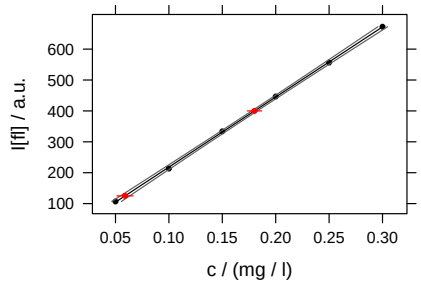
Calibration function and its 99% confidence interval:

```
> int <- list (spc = as.matrix (seq (min (flu),
+               max (flu), length.out = 25) ) )
> ci <- predict (calibration,
+                newdata = int,
+                interval = "confidence",
+                level = .99)
> panel.ci <- function (x, y, ...,
+                       intensity, ci.lwr, ci.upr,
+                       ci.col = "#606060") {
+   panel.xyplot (x, y, ...)
+   panel.lmline (x, y, ...)
+   panel.lines (ci.lwr, intensity, col = ci.col)
+   panel.lines (ci.upr, intensity, col = ci.col)
+ }
```

And now the plot:

```
> plotc (flu,
+        panel = panel.ci,
+        intensity = int$spc,
+        ci.lwr = ci [, 2],
+        ci.upr = ci [, 3])
```

Calibration - alternative




```
> flu$type <- "data points"
```

Calculated confidence intervals are added to the hyperSpec object:

```
> tmp <- new ("hyperSpec",
+             spc = as.matrix(seq (min (flu),
+               max(flu), length.out = 25)),
+             wavelength = 450)
> ci <-  predict (calibration, newdata = tmp$.,
+                 interval = "confidence",
+                 level = 0.99)
> tmp <- tmp [rep (seq (tmp, index = TRUE), 3)]
> tmp$c <- as.numeric (ci)
> tmp$type <- rep (colnames (ci), each = 25)
> flu$file <- NULL
> flu <- rbind (flu, tmp)
```

Another panel function:

```
> panel.predict <- function (x, y, ...,
+                  intensity, ci, pred.col = "red",
+                            pred.pch = 19,
+                            pred.cex = 1) {
+    panel.xyplot (x, y, ...)
+    mapply (function (i, lwr, upr, ...) {
+                  panel.lines (c (lwr, upr),
+                               rep (i, 2), ...)
+              },
+            intensity, ci [, 2], ci [, 3],
+            MoreArgs = list (col = pred.col))
+    panel.xyplot (ci [, 1], intensity,
+                  col = pred.col, pch = pred.pch,
+                  cex = pred.cex, type = "p")
+ }
```

The final plot:

```
> plotc (flu, groups = type, type = c("l", "p"),
+        col = c ("black", "black", "#606060",
+          "#606060"),
+        pch = c (19, NA, NA, NA), cex = 0.5,
+        lty = c (0, 1, 1, 1),
+        panel = panel.predict,
+        intensity = I,
+        ci = conc,
+        pred.cex = 0.5)
```

7 False-Colour Maps

plotmap

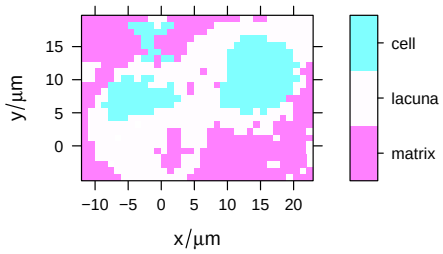
`plotmap` uses `levelplot`, a *lattice* function. Therefore, in loops, functions, Sweave chunks, etc. the lattice object needs to be printed explicitly by `print (plotmap (object))` (R FAQ: Why do

lattice/trellis graphics not work?).

`plotmap` produces a 3d plot, with the z axis colour-coded. `plotmap`'s arguments x and y take the name of extra data columns.

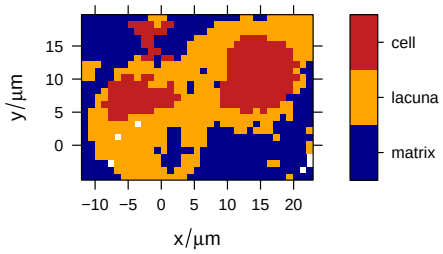
The colour-coded axis. Also z can be used to select one column of the extra data by name. Alternatively, it may be a numeric or factor directly giving the values to be used. Each level of a factor will have one colour. It is also possible to plot a sum characteristic of the spectra: supply the function in argument *func*. The default setting is to plot the average intensity (no z and *func*= `mean`).

plotting clusters




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

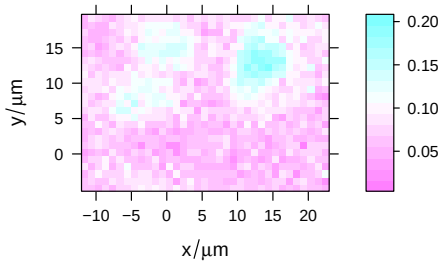
different palette



To plot with a different palette, use *trellis.args*= `list (col.regions = palette)`.

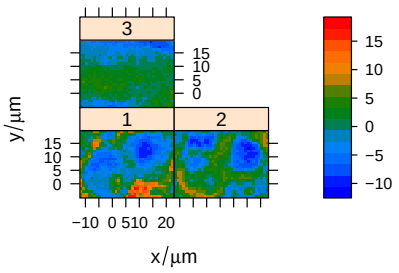
```
> cols <- c ("dark blue", "orange", "#C02020")
> print (plotmap (chondro,
+                 clusters ~ x * y,
+                 col.regions = cols))
```

defined wavelengths



To plot a map with particular wavelengths use this:

```
> plotmap (chondro[, , c( 728, 782, 1098, 1240,
+                         1482, 1577)])
```



Preprocessing of the data:

```
> baselines <- spc.fit.poly.below (chondro)
Fitting with npts.min =  15
> chondro <- chondro - baselines
> chondro <- sweep (chondro, 1, apply (chondro, 1, mean), "/")
> chondro <- sweep (chondro, 2, apply (chondro, 2, quantile,
+                                      0.05), "-")
> pca <- prcomp (~ spc, data = chondro$., center = TRUE)
> scores <- decomposition (chondro, pca$x,
+                          label.wavelength = "PC",
+                          label.spc = "score /  a.u.")
```

The plot:

```
> levelplot (spc ~ x * y | as.factor(.wavelength),
+            scores [,,1:3],
+            aspect = "iso",
+            col.regions = matlab.dark.palette(20))
```

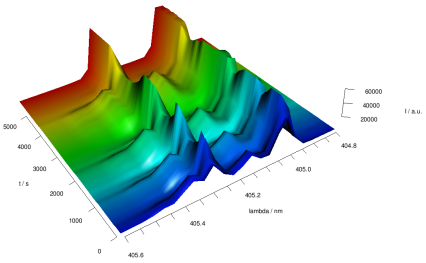
Examples

Conditioning. Lattice graphics have a concept of conditioning a plot. Instead of plotting all data in one diagram, a diagram is produced for each of the groups specified by the condition. `plotmap`'s argument *cond* takes he name of the extra data column used for conditioning. This could e. g. be a column containing the sample number of a *hyperSpec* object that contains several samples.

Beispiele: voronoi

8 3 D

3D figures are possible with *rgl*



```
> 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 preceeding 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 *latticist*. These packages allow easy customization of the plots and also identification of points.