hyperSpec Introduction

Claudia Beleites (cbeleites@units.it)
CENMAT, DMRN, University of Trieste

August 4, 2009

Contents

1 Introduction

hyperSpec is a R package that allows convenient handling of (hyper)spectroscopic data sets, i. e. data sets comprising spectra together with further data on a per-spectrum basis. Likewise, the spectra can be anything that is recorded over a common discretized axis, the wavelength axis. Throughout the documentation of the package, the terms intensity and wavelength refer to the spectral ordinate and abscissa, respectively.

However, *hyperSpec* works perfectly fine with any data that fits in that general scheme, so that the three terms may also be used for:

wavelength: frequency, wavenumbers, chemical shift, Raman shift, $\frac{m}{z}$, etc.

intensity: transmission, absorbance, $\frac{e^-}{s}$, ...

extra data: spatial information (spectral images, maps, or profiles), temporal information (kinetics, time series), concentrations (calibration series), class membership information, etc. Note that there is no restriction on the number of extra data columns.

This vignette gives an introduction on basic working techniques using the R package *hyperSpec*. It comes with three data sets,

`chondro` a Raman map of chondrocytes in cartilage,

`flu` a set of fluorescence spectra of a calibration series, and

`laser` a time series of an unstable laser emission

In this vignette, all three data sets are used in an intermixed way to illustrate appropriate procedures for different tasks.

This document describes how to accomplish spectroscopic tasks. It does not give a complete reference on particular functions. Therefore recommend to look up the used methods in R's help system using `? command`.

1.1 Notation

This vignette demonstrates working techniques mostly from a spectroscopic point of view: rather than going through the functions provided by *hyperSpec*, it is organized more closely on spectroscopic tasks. However, the functions discussed are printed on the margin for a fast overview.

In R, slots of a S4 class can be accessed directly by the `@` operator. In this vignette, the notation `@xxx` will thus mean *"slot xxx of an object"* see figure 1 on page 5).

Likewise, named elements of a *list*, like the columns of a *data.frame*, are accessed by the `$` operator, and `$xxx` will be used for *"column xxx"*, and as an abbreviation for *"column xxx of the data.frame in slot data of the object"* see figure 1 on page 5) .

2 Remarks on R

2.1 Generic Functions

Generic Functions are functions that apply to a wide range of data types or classes, e. g. *plot*, *print*, mathematical operators, etc. These functions can be implemented in a specialized way by each class.

hyperSpec implements with a variety of such functions, see table 1.

2.2 S4 Classes Can be Extended at Runtime

The concept of S4 classes offers more flexibility than the class concepts in many other programming languages. Functions may be added or changed by the user in his *workspace* at any time. Neither restart of R nor reloading of the package or anything the like is needed. At the same time, the original function is not deleted, it is just masked by the user's new function but stays accessible if the change should be reverted.

This offers the opportunity of easily writing specialized functions that are adapted to specific tasks.

2.3 Validity

S4 classes have a mechanism to define and enforce that the data actually stored in the object is appropriate for this class. In other words, there is a mechanism of *validity checking*.

The functions provided by *hyperSpec* check the validity of *hyperSpec* objects at the beginning, and — if the validity could be broken by inappropriate arguments — also before leaving the function.

3 Loading the package

To load *hyperSpec*, use

```
> library (hyperSpec)
```

Table 1: Generic methods implemented by *hyperSpec* and some related functions. *Emphasized* names indicate non-generic functions that are closely related to the generic functions in the row.

Function	**Explanation**
`print`, `show`, `summary`	print information about the object
`plot`	plotting
`[`,`[[`, `$`	extract parts of an object
`[<-`, `[[<-`, `$<-`	assign parts of an object
`dim`, `ncol`, `nrow`, *nwl*	the dimensions of the object
`colnames`, `rownames`, `dimnames`	names of the spectra, data columns, and both plus the names of the wavelengths
`labels`, *`labels<-`*	labels for axis annotations etc.
`+ - * ^ %% %/% / %*%`	arithmetical operators work on `$spc`
`> < == >= <=`	comparison operators work on `$spc`
`log`, `log10`, `exp`, etc.	basic math functions work on `$spc`, see also: `? "hyperSpec Math"`
`min`, `max`, `range`	minimum, maximum, and range of the intensities in `$spc`
`cbind`, `rbind`	combine two objects by columns or rows
`split`	split an object into a list of objects
`apply`	apply a function row- or column-wise, calculate e. g. the mean spectrum or normalization factors
`aggregate` (`ave`)	calculate sum characteristics for groups of spectra, e. g. cluster mean spectra. *hyperSpec*'s `aggregate` method covers also the functionality of `ave`.
`sweep`	"sweep" a sum characteristic over rows or columns, e. g. centre the data by subtracting the mean spectrum.
`as.character`, `as.matrix`, `as.data.frame`	type conversion functions
`initialize`	create an object
`validObject`	validity checking

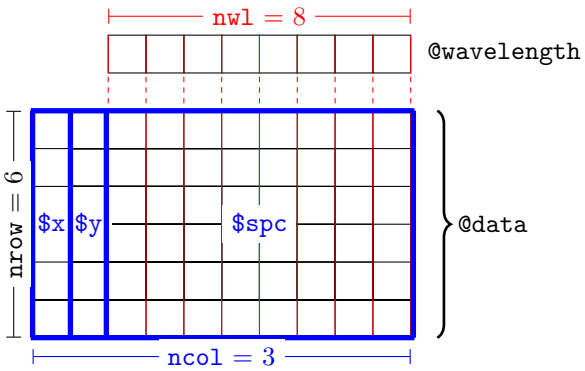


Figure 1: The structure of the data in a *hyperSpec* object.

```
Package hyperSpec, version 0.7

To get started, try
   help ("hyperSpec")
   help (package = "hyperSpec")
   vignette (package = "hyperSpec")

If you use this package please cite it appropriately.
   citation("hyperSpec")
will give you the correct reference.

The project is hosted on http://r-forge.r-project.org/projects/hyperspec/
```

4 The structure of hyperSpec objects

hyperSpec is a S4 (or new-style) class. It has four so-called *slots* that hold the data:

`@wavelength`	containing a numeric vector with the wavelength axis of the spectra.
`@data`	a *data.frame*with the spectra and all further information belonging to the spectra
`@label`	a list with appropriate labels (particularly for axis annotations)
`@log`	a *data.frame* keeping track of what is done with the object

However, it is good practice to use the functions provided by *hyperSpec* to handle the objects rather than accessing the slots directly. This also helps ensuring that proper (*valid*) objects are retained.

Most of the data is stored in `@data`. This *data.frame* has one special column, `$spc`. It is the column that actually contains the spectra. The spectra are stored in a matrix inside this column, as illustrated in figure 1. Even if there are no spectra, `$spc` must still be present but it can contain a matrix with zero columns.

Slot `@label` contains an element for each of the columns in `@data` plus one holding the label for the wavelength axis, `.wavelength`. The elements of the list may be anything suitable for axis annotations, i. e. they should be either character strings or expressions for prettier axis annotations (see figure 2 on page 12). To get familiar with expressions for axis annotation, see

```
> ? plotmath
```

and

```
> demo (plotmath)
```

5 Obtaining Basic Information about hyperSpec Objects

As usual, the *print* and *show* methods display information about the object, and *summary* yields some additional details about the data handling done so far:

```
> chondro
```

```
hyperSpec object
   875 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (875 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 875] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 875] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 875 x 300] range 80.04420 81.75761 ... 1858.881
```

```
> summary (chondro)
```

```
hyperSpec object
   875 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (875 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 875] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 875] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 875 x 300] range 80.04420 81.75761 ... 1858.881
log:
                   short       long                  date    user
   1   scan.txt.Renishaw   list(...   2009-07-07 12:02:48   cb@cb
   2             orderwl   list(...   2009-07-07 12:02:48   cb@cb
   3           spc.loess   list(...   2009-07-07 12:03:13   cb@cb
```

The data set `chondro` consists of 875 spectra with 300 data points each, and 3 data columns: two for the spatial information plus `$spc`. These informations can be directly obtained by

```
> nrow (chondro)
```

```
[1] 875
```

```
> nwl (chondro)
```

```
[1] 300
```

```
> ncol (chondro)
```

```
[1] 3
```

```
> dim (chondro)
```

```
nrow ncol  nwl
 875    3  300
```

The names of the columns in `@data` are accessed by

```
> colnames (chondro)
[1] "y"   "x"   "spc"
```

Likewise, *rownames* returns the names assigned to the spectra, and *dimnames* yields a list of these three vectors (including also the column names of `$spc`).

6 Creating a hyperSpec Object, Data Import and Export

6.1 ASCII Files

read.txt.long
read.txt.wide
write.txt.long
write.txt.wide

Currently, *hyperSpec* provides four functions for general ASCII data import and export:

`read.txt.long`	import long format ASCII files, i.e. one intensity value per row
`read.txt.wide`	import wide format ASCII files, i.e. one spectrum per row
`write.txt.long`	export long format ASCII files
`write.txt.wide`	export wide format ASCII files

The import functions immediately return a *hyperSpec* object.

6.2 Manufacturer Specific Import Functions

Many spectrometer manufacturers provide a function to export their spectra into ASCII files. The functions discussed in the previous section are written in a very general way, and are highly customizable. I recommend wrapping these calls with the appropriate settings for your spectra format in an import function. You may also consider contributing such import filters to *hyperSpec*: send me (cbeleites@units.it) the documented code (either .R + .Rd file or Roxygen commented .R).

For the long ASCII format written by Renishaw's converter, a more optimized import function is already available: `scan.txt.Renishaw`.

scan.txt.Renishaw

```
> paracetamol <- scan.txt.Renishaw ("paracetamol.txt", "spc")
> paracetamol
hyperSpec object
   1 spectra
   1 data columns
   4064 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 4064]  96.7865 98.1432 ... 3200.07
data:  (1 rows x 1 columns)
   (1) spc: I / a.u. [AsIs matrix 1 x 4064] range 299.229 317.041 ... 49052.2
```

6.3 Matlab Files

Matlab files can be read and written using the package *R.matlab*[**?**], which is available at CRAN and can be installed by `install.packages ("R.matlab")`.

6.4 Creating a hyperSpec Object from Spectra Matrix and Wavelength Vector

Once the data is in R's workspace, a *hyperSpec* object is created by:
`spc <- new ("hyperSpec", spc = spectra.matrix, wavelength = wavelength.vector)`
You will usually give the following arguments:

`spc`	the spectra matrix
`wavelength`	the wavelength axis vector
`data`	the extra data
`label`	a list with the proper labels. Do not forget the wavelength axis label in `$.wavelength` and the spectral intensity axis label in `$spc`.

7 Combining hyperspec Objects

`cbind rbind`

hyperspec Objects can be bound together, either by rows to append a new spectral range or by columns to append new spectra

```
> cbind (chondro [, , 600 ~ 800], chondro [, , 1600 ~ 1800])

hyperSpec object
   875 spectra
   3 data columns
   101 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 101]  602 606 ... 1798
data:  (875 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 875] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 875] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 875 x 101] range 80.04420 81.75761 ... 1541.625

> rbind (chondro [, , 600 ~ 800], chondro [, , 600 ~ 800])

hyperSpec object
   1750 spectra
   3 data columns
   50 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 50]  602 606 ... 798
data:  (1750 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 1750] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 1750] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 1750 x 50] range 195.5281 212.0432 ... 729.5765
```

There is also a more general function, `bind`, taking the direction (`"r"` or `"c"`) as first argument and then all objects to bind either in separate arguments or in a list.

8 Access to the data

8.1 Selecting and Deleting Spectra

The extraction function `[]` (or `[[]]`, if the spectra *matrix* or the *data.frame* is needed rather than a *hyperSpec* object) takes the spectra as first argument (For detailed help: `? "["`). It may be a vector giving the indices of the spectra to extract (select), a vector with negative indices indicating which spectra should be deleted, or a logical

```
> flu [1 : 3]

hyperSpec object
   3 spectra
   2 data columns
   181 data points / spectrum
wavelength: lambda[fl]/nm [numeric 181]  405.0 405.5 ... 495
data:  (3 rows x 2 columns)
   (1) c: c / (mg/l) [numeric 3] range 0.05 0.10 0.15
   (2) spc: I / a.u. [AsIs matrix 3 x 181] range 27.15000 32.34467 ... 336.5057

> flu [-3]

hyperSpec object
   5 spectra
   2 data columns
   181 data points / spectrum
wavelength: lambda[fl]/nm [numeric 181]  405.0 405.5 ... 495
data:  (5 rows x 2 columns)
   (1) c: c / (mg/l) [numeric 5] range 0.05 0.10 0.20 0.25 0.30
   (2) spc: I / a.u. [AsIs matrix 5 x 181] range 27.15000 32.34467 ... 677.4947

> chondro  [chondro$y > 10]

hyperSpec object
   350 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (350 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 350] range 10.23 11.23 ... 19.23
   (2) x: x/(mu * m) [numeric 350] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 350 x 300] range 88.98556 89.99474 ... 1745.724
```

8.2 Accessing the Extra Data

The second argument of the extraction functions `[]` and `[[]]` specifies the (extra) data columns. They can be given like any column specification for a *data.frame*, i. e. numeric, logical, or by a vector of the column names:

```
> colnames (chondro)

[1] "y"   "x"   "spc"

> chondro [[1 : 3, 1]]

      y
1 -4.77
2 -4.77
3 -4.77

> chondro [[1 : 3, -3]]

      y      x
1 -4.77 -11.55
2 -4.77 -10.55
3 -4.77  -9.55
```

```
> chondro [[1 : 3, "x"]]

       x
1 -11.55
2 -10.55
3  -9.55

> chondro [[1 : 3, c (TRUE, FALSE, FALSE)]]

      y
1 -4.77
2 -4.77
3 -4.77
```

To select one column, the `$` operator is more convenient:

```
> flu$c

[1] 0.05 0.10 0.15 0.20 0.25 0.30
```

The extra data may also be set this way:

```
> flu$n <- list (1 : 6, label = "sample no.")
```

This function will append new columns, if necessary.

8.3 Wavelengths and Spectral Axis

8.3.1 Wavelength Indices

wl2i i2wl

Spectra in *hyperSpec* have always discretized wavelength axes, they are stored in a matrix with column corresponding to one wavelength. *hyperSpec* provides two conversion functions: `i2wl` returns the wavelength corresponding to the given indices and `wl2i` calculates index vectors from wavelengths.

If the wavelengths are given as a numeric vector, they are each converted to the corresponding wavelength. In addition there is a more sophisticated possibility of specifying wavelength ranges using a *formula*. The basic syntax is *start* $\sim$ *end*. This yields a vector *index of start* `:` *index of end*.

The result of the formula conversion differs from the numeric vector conversion in three ways:

- The colon operator for constructing vectors accepts only integer numbers, the tilde (for formulas) does not have this restriction.
- If the vector does not take into account the spectral resolution, one may get only every n^{th} point or repetitions of the same index:

```
> wl2i (flu, 405 : 410)

[1]  1  3  5  7  9 11

> wl2i (flu, 405 ~ 410)

 [1]  1  2  3  4  5  6  7  8  9 10 11
```

```
> wl2i (chondro, 1000 : 1010)
 [1] 100 101 101 101 102 102 102 102 102 103 103
> wl2i (chondro, 1000 ~ 1010)
[1] 100 101 102 103
```

- If the object's wavelength axis is not ordered, the formula approach doesn't work. In that (rare) case, use `orderwl` first to obtain an object with ordered wavelength axis.

start and *end* may contain the special variables `min` and `max` that correspond to the lowest and highest wavelengths of the object:

```
> wl2i (flu, min ~ 410)
 [1]  1  2  3  4  5  6  7  8  9 10 11
```

Often, specifications like *wavelength* $\pm n$ *data points* are needed. They can be given using complex numbers in the formula. The imaginary part is added to the index calculated from the wavelength in the real part:

```
> wl2i (flu, 450 - 2i ~ 450 + 2i)
[1] 89 90 91 92 93
> wl2i (flu, max - 2i ~ max)
[1] 179 180 181
```

To specify several wavelength ranges, use a list containing the formulas and vectors[1]:

```
> wl2i (flu, 450 - 2i ~ 450 + 2i)
[1] 89 90 91 92 93
> wl2i (flu, c (min ~ 406.5, max - 2i ~ max))
[1]   1   2   3   4 179 180 181
```

This mechanism also works for the wavelength arguments of `[]`, `[[]]`, and `plotspc`.

8.3.2 Wavelength Axis Conversion

8.4 Fast Access to Parts of the hyperSpec Object

`[[]]` `$.` `$..`

hyperSpec comes with three abbreviation functions for easy access to the data:

`x [[]]` returns the spectra matrix (`x$spc`).

`x [[i, , l]]` the cut spectra matrix is returned if wavelengths are specified in l.

`x [[i, j, l]]` If data columns are selected (second index), the result is a *data.frame*.

`x [[i, , l]] <-` Also, parts of the spectra matrix can be set (only indices for spectra and wavelength are allowed for this function).

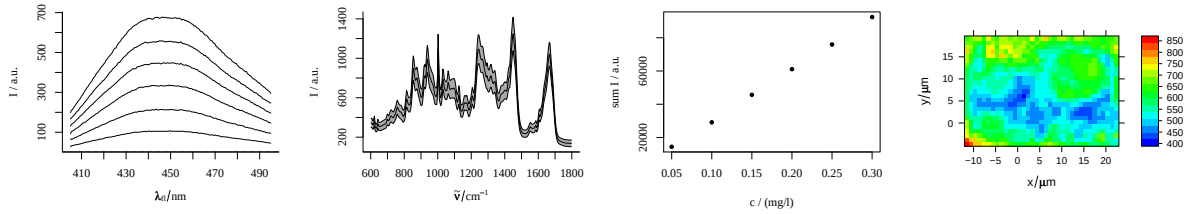
`x [i, j] <-` sets parts of `x@data`.

`x $.` returns the complete *data.frame* `x@data`, with the spectra in column `$spc`.

`x $..` returns the extra data (`x@data` without `x$spc`).

`x $.. <-` sets the extra data (`x@data` without `x$spc`). However, the columns must match exactly in this case.

[1] Formulas are combined to a list by `c`.



(a) (b) (c) (d)

Figure 2: Some example plots. (a)`plotspc (flu)`, (b)`plot (chondro, "spcmeansd")`, (c)`plotc (flu)`, and (d)`plotmap (chondro)`.

9 Plotting

hyperSpec comes with three predefined plotting functions.

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

`plotmap` plots a false colour map: a single value (e. g. average intensity or cluster membership) over two data columns (default `$x` and `$y`).

`plotc` plots a time series or calibration plot: e. g. an intensity over a single other data column (like concentration, depth, or time).

All three plus some more handy abbreviations are also accessible via *plot*: `plot`

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

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

`plot (chondro, "spcprctl")` plots median, 16^{th} and 84^{th} percentile. This is similar to `"spcmeansd"`. 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, "spcprctl5")` like `"spcprctl"` plus 5^{th} and 95^{th}percentile.

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

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

`plot (laser, "ts")` plots a time series plot, equivalent to `plotc (laser, use.c = "t")`

`plot (x, "depth")` plots a depth profile plot, equivalent to `plotc (laser, use.c = "z")`

Figure 2 shows some example plots.

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

9.1 Plotting Spectra

`plotspc`

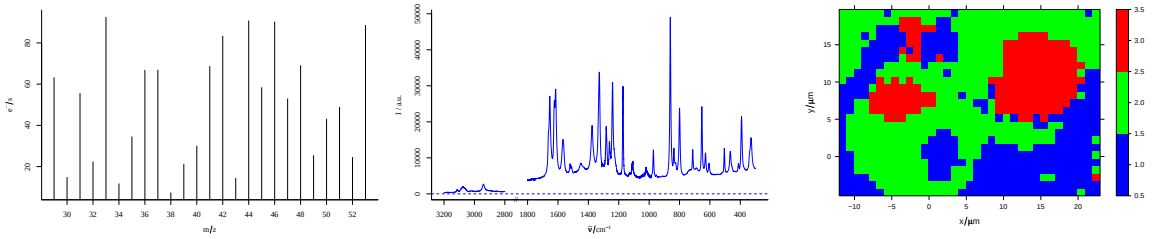
`plotspc` offers a variety of parameters for customized plots. To plot ...

with reversed abscissa use `wl.reverse = TRUE`

in different colours colours use `col = vector.of.colours`

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

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



(a) (b) (c)

Figure 3: Arguments to `plotspc`. (a) `plot (fake.mass.spec, lines.args = list (type = "h"))` (b) `plotspc (paracetamol, wl.range = c (300 ~ 1800, 2800 ~ max), xoffset = 850, wl.reverse = TRUE)` (c) `plotmap` with a factor, see section 11.2.

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 = 800`

stacked spectra use `stacked = TRUE`

more spectra into an existing plot use `add = TRUE`

with different line at I = 0 use `zeroline =` *`list.of.arguments.to.abline`*. `NULL` suppresses the line.

9.2 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 *use.c* `= name`. The ordinate can be calculated as a sum characteristic (with parameter *func*`= function`, defaulting to *sum*). If parameter z is given, these values are used instead. z may be the name of an extra data column, or a *numeric* that should be used directly.

To customize the plot, give any arguments that you would usually supply to *plot* as a list using argument *plot.args*.

9.3 Plotting 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`).

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

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.

10 Spectral (Pre)processing

10.1 Cutting the Spectral Range

`[] [[]]`

The extraction functions `[]` and `[[]]` can be used to cut the spectra: Their third argument takes wavelength specifications as discussed above and also logicals (i.e. vectors specifying with `TRUE`/`FALSE` for each column of `$spc` whether it should be included or not.
`[]` returns a *hyperSpec* object, `[[]]` the spectra *matrix*`$spc` (or the *data.frame*`@data` if data columns were specified, too) only.

```
> flu [,, min ~ 408.5]

hyperSpec object
   6 spectra
   3 data columns
   8 data points / spectrum
wavelength: lambda[fl]/nm [numeric 8]  405.0 405.5 ... 408.5
data:  (6 rows x 3 columns)
   (1) c: c / (mg/l) [numeric 6] range 0.05 0.10 ... 0.3
   (2) spc: I / a.u. [AsIs matrix 6 x 8] range 27.15000 32.34467 ... 256.8913
   (3) n: sample no. [integer 6] range 1 2 ... 6

> flu [[,, c (min ~ min + 2i, max - 2i ~ max)]]

           405     405.5       406       494     494.5       495
[1,]  27.15000  32.34467  33.37867  47.16267  46.41233  45.25633
[2,]  66.80133  63.71533  66.71200  96.60167  96.20600  94.61033
[3,]  93.14433 103.06767 106.19367 149.53900 148.52667 145.79333
[4,] 130.66367 139.99833 143.79767 201.48433 198.86733 195.86733
[5,] 167.26667 171.89833 177.47067 252.06567 248.06700 246.95200
[6,] 198.43033 209.45800 215.78500 307.51850 302.32550 294.64950
```

10.2 Spectral Interpolation and Smoothing

`spc.bin`
`spc.loess`

Frequently, a *hyperSpec* object needs to be interpolated onto a new wavelength axis. e. g. because measurements resulted in slightly shifted wavelength axes. Or data from a grating spectrometer with unequal data point spacing should be interpolated onto an evenly spaced wavelength axis. Also, the spectra can be smoothed: reducing the spectral resolution allows to increase the signal to noise ratio. For chemometric data analysis reducing the number of data points per spectrum may be crucial as it reduces the dimensionality of the data.

hyperSpec provides two functions to change the wavelength axis of *hyperSpec* objects: `spc.bin` and `spc.loess`.

`spc.bin` bins the spectral axis by averaging every *by* data points.

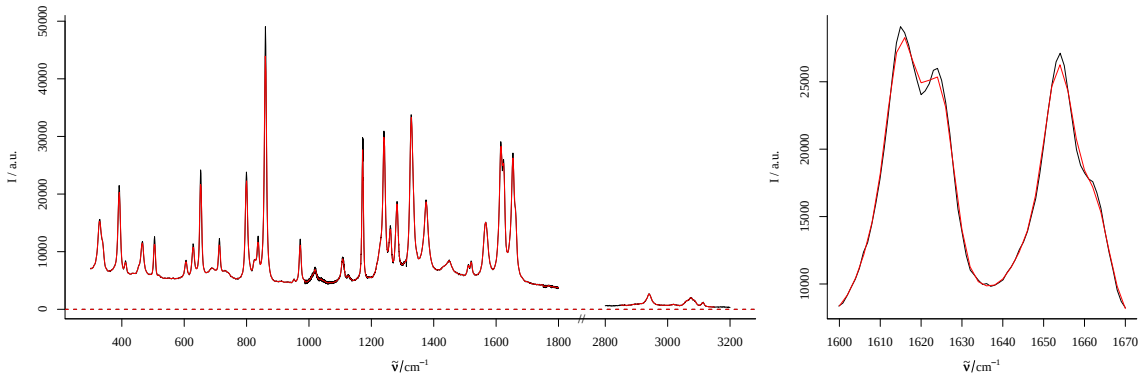


Figure 4: Smoothing interpolation by `spc.loess` with new data point spacing of 2 cm^{-1}. The magnification on the right shows how interpolation may cause a loss in signal.

```
> plot (paracetamol, wl.range = c (300 ~ 1800, 2800 ~ max), xoffset = 850)
> p <- spc.loess (paracetamol, c(seq (300, 1800, 2), seq (2850, 3150, 2)))
> plot (p, wl.range = c (300 ~ 1800, 2800 ~ max), xoffset = 850, col = "red", add = TRUE)
```

`spc.loess` applies R's `loess` function for spectral interpolation. Figure 4 shows the result of interpolating from 300 to 1800 and 2850 to 3150 cm^{-1} with 2 cm^{-1} data point distance. This corresponds to a spectral resolution of about 4 cm^{-1}, and the decrease in spectral resolution can be seen at the sharp bands where the maxima are not reached (due to the fact that the interpolation wavelength axis does not necessarily hit the maxima. The original spectrum had 4064 data points with unequal data point spacing (between 0 and 1.4 cm^{-1}). The interpolated spectrum has 902 data points.

10.3 Background Correction

To subtract a background spectrum of each of the spectra in an object, use `sweep (spectra, 2, background.spectrum, "-")`.

10.4 Offset Correction

Calculate the offsets and sweep them off the spectra:

```
> offsets <- apply (chondro, 1, min)
> chondro.offset.corrected <- sweep (chondro, 1, offsets, "-")
```

10.5 Baseline Correction

hyperSpec comes with two functions to fit polynomial baselines.

`spc.fit.poly` fits a polynomial baseline of the given order. A least-squares fit is done so that the function may be used on rather noisy spectra. However, the user must supply an object that is cut appropriately. Particularly, the supplied wavelength ranges are not weighted.

`spc.fit.poly.below` tries to find appropriate support points for the baseline iteratively.

Both functions return a *hyperSpec* object containing the fitted baselines. They need to be subtracted afterwards:

```
> bl <- spc.fit.poly.below (chondro)
Fitting with npts.min =  15
> chondro <- chondro - bl
```

For details, see `vignette (baselinebelow)`.

10.6 Intensity Calibration

10.6.1 Correcting by a constant, e. g. Readout Bias

CCD cameras often operate with a bias, causing a constant value for each pixel. Such a constant can be immediately subtracted:
`spectra - constant`

10.6.2 Correcting Wavelength Dependence

This means that for each of the wavelengths the same correction needs to be applied to all spectra.

1. There might be wavelength dependent offsets (background or dark spectra). They are subtracted:
   `sweep (spectra, 2, offset.spectrum, "-")`
2. A multiplicative dependency such as a CCD's photon efficiency:
   `sweep (spectra, 2, photon.efficiency, "/")`

10.6.3 Spectra Dependent Correction

If the correction depends on the spectra (e. g. due to inhomogeneous illumination while collecting imaging data[2]), the *MARGIN* of the `sweep` function needs to be 1:

1. Pixel dependent offsets are subtracted:
   `sweep (spectra, 2, pixel.offsets, "-")`
2. A multiplicative dependency:
   `sweep (spectra, 2, illumination.factors, "*")`

10.7 Normalization

1. Calculate appropriate normalization factors:
   `factors <- 1 / apply (spectra, 1, sum)` for area normalization. `mean` gives equal results, just that the Intensities are on the same scale as before.
   For minimum-maximum-normalization, first do an offset- or baseline correction, then calculate the *`factors`* using `max`.
   You may calculate the factors using only a certain wavelength range, thereby normalizing on a particular band or peak.
2. Again, sweep the factor off the spectra:
   `normalized <- sweep (spectra, 1, factors, "*")`

```
> factors <- 1 / apply (chondro, 1, mean)
> chondro <- sweep (chondro, 1, factors, "*")
```

[2] imaging (as opposed to mapping) refers to simultaneously collecting spatially resolved spectra, either 2d images or line imaging.

10.8 Centering the Data

Centering means that the mean spectrum is subtracted from each of the spectra. Many data analysis techniques, like principal component analysis, partial least squares, etc., work much better on centered data.

However, from a spectroscopic point of view it depends on the particular data set whether centering does make sense or not.

It is perfectly fine to centre the `flu` data set: the interpretation is that centering the data cancels the offset (background spectrum etc.) of the calibration:

```
> flu.centered <- sweep (flu, 2, apply (flu, 2, mean), "-")
```

```
> plot (flu.centered)
```

On the other hand, the `chondro` data set consists of Raman spectra, so the spectroscopic interpretation of centering is getting rid of the the average chemical composition of the sample. But: what is the meaning of the "average spectrum" of an inhomogeneous sample? In this case it is better to subtract the minimum spectrum (which will hopefully have almost the same benefit on the data analysis) as it is the spectrum of that chemical composition that is underlying the whole sample.

One more point to consider is that the actual minimum spectrum will pick up lots of the negative noise. In order to avoid that, using e. g. the 5^{th} percentile spectrum is more suitable:

```
> chondro <- sweep (chondro, 2, apply (chondro, 2, quantile, 0.05), "-")
```

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

10.9 Variance Scaling

Variance scaling is often used in multivariate analysis to adjust the influence and scaling of the variates (that are typically different physical values). However, it is hardly appropriate for spectra that do have the same scale of the same physical value.

10.10 Multiplicative Scatter Correction (MSC)

MSC can be done using `msc` from package *pls*[**?**]. It operates on the spectra matrix:

```
> library (pls)
> chondro.msc <- chondro
> chondro.msc [[]] <- msc (chondro [[]])
```

10.11 Spectral Arithmetic

`+ - * / ^ log log10`

Basic mathematical functions are defined for *hyperSpec* objects. You may convert spectra:
`absorbance.spectra = - log10 (transmission.spectra)`

In this case, do not forget to adapt the label:

`labels`

```
> labels (absorbance.spectra)$spc <- "A"
```

Be careful: R's `log` function calculates the natural logarithm if no base is given.

The basic arithmetic operators work element-wise in R. Thus they all need either a scalar, or a matrix (or *hyperSpec* object) of the correct size.

Matrix multiplication is done by **%*%** , again each of the operands may be a matrix or a *hyperSpec* object, and must have the correct dimensions.

`%*%`

11 Data Analysis

11.1 Data Analysis Methods using a data.frame e. g. Principal Component Analysis with `prcomp`

The `$.` notation is handy, if a data analysis function expects a *data.frame*. The column names can then be used in the formula:

```
> pca <- prcomp (~ spc, data = chondro$., centre = FALSE)
```

Results of such a decomposition can be put again into *hyperSpec* objects. This allows to plot e.g. the loading like spectra, or score maps, see figure 5.

```
> scores <- decomposition (chondro, pca$x, label.wavelength = "PC", label.spc = "score / a.u.")
> loadings <- decomposition (chondro, t(pca$rotation), scores = FALSE, label.spc = "loading I / a.u.")
```

11.2 Data Analysis Methods using a matrix e. g. Hierarchical Cluster Analysis

```
> dist <- pearson.dist (chondro [[]])
> dendrogram <- hclust (dist, method = "ward")
```

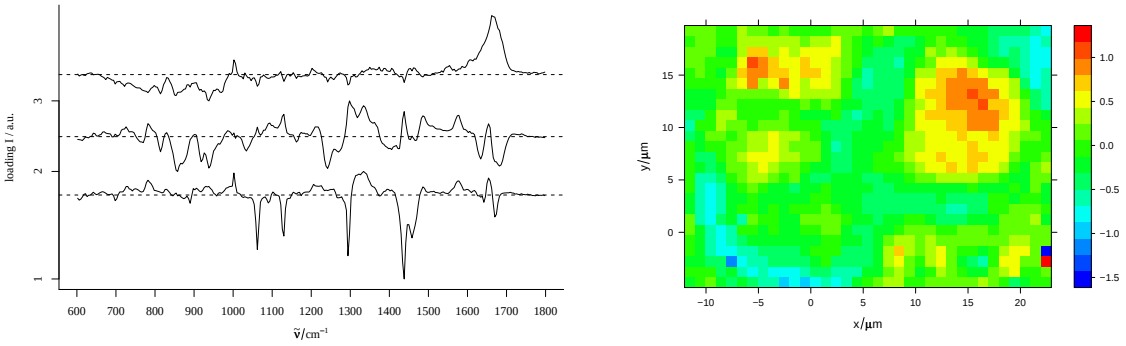
```
> plot (dendrogram)
```

In order to plot a cluster map, the cluster membership needs to be calculated from the dendrogram.

First, cut the dendrogam so that three clusters result:

```
> clusters <- cutree (dendrogram, k = 3)
```

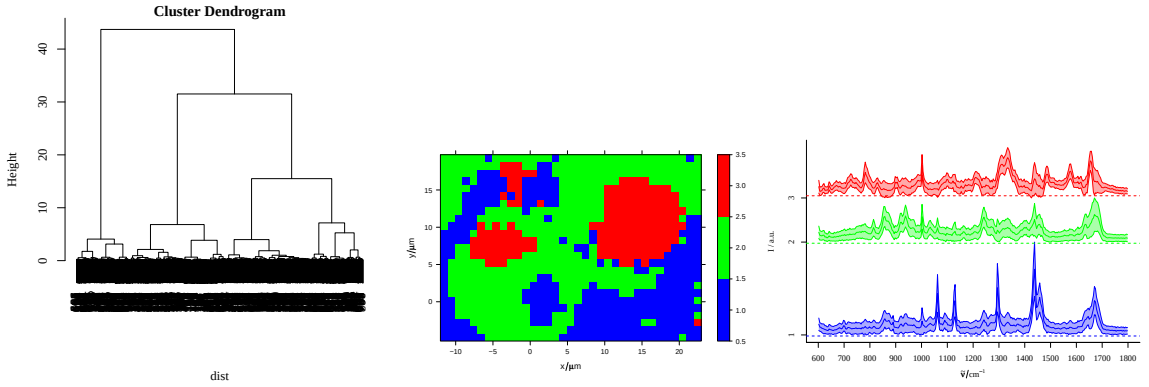
Then the result may be plotted:



(a) (b)

Figure 5: (a) The first three loadings: `plot (loadings [1 : 3], stacked = TRUE)`. (b) The second score map: `plotmap (scores [, , 2])`

.



(a) (b) (c)

Figure 6: The results of the cluster analysis: (a) the dendrogram (b) the map of the 3 clusters (c) the mean spectra.

11.3 Calculating group-wise Sum Characteristics e. g. Cluster Mean Spectra

`aggregate` applies the function given in *FUN* to each of the groups of spectra specified in *by*. `aggregate`

So we may plot the cluster mean spectra:

```
> means <- aggregate (chondro, by = clusters, mean_pm_sd)
> means
```

```
hyperSpec object
   9 spectra
   4 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (9 rows x 4 columns)
   (1) y: y/(mu * m) [numeric 9] range -4.77 -2.77
   (2) x: x/(mu * m) [numeric 9] range -11.55 -5.55 22.45
   (3) spc: I / a.u. [matrix 9 x 300] range -0.01384176 -0.01258208 ... 0.5974339
   (4) .aggregate: [factor 9] range 1 2 3
```

```
> plot (means, col = matlab.palette (3), stacked = ".aggregate", fill = ".aggregate")
```

11.4 Splitting an Object

A *hyperSpec* object may also be split into a list of *hyperSpec* objects:

```
> clusters <- split (chondro, clusters)
> clusters
```

```
$`1`
hyperSpec object
   292 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (292 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 292] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 292] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 292 x 300] range -0.1372477 -0.1306328 ... 0.9382884

$`2`
hyperSpec object
   417 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (417 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 417] range -4.77 -3.77 ... 19.23
   (2) x: x/(mu * m) [numeric 417] range -11.55 -10.55 ... 22.45
   (3) spc: I / a.u. [matrix 417 x 300] range -0.2540939 -0.2373205 ... 1.043128

$`3`
hyperSpec object
```

```
   166 spectra
   3 data columns
   300 data points / spectrum
wavelength: tilde(nu)/cm^-1 [numeric 300]  602 606 ... 1798
data:  (166 rows x 3 columns)
   (1) y: y/(mu * m) [numeric 166] range -2.77 5.23 ... 18.23
   (2) x: x/(mu * m) [numeric 166] range -7.55 -6.55 ... 22.45
   (3) spc: I / a.u. [matrix 166 x 300] range -0.2713962 -0.2156510 ... 0.4380194


> rm (list = ls () )
```