hyperSpec Introduction

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

March 21, 2010

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. Introduction

hyperSpec is a R package that allows convenient handling of (hyper)spectral data sets, i. e. data sets comprising spectra together with further data on a per-spectrum basis. The spectra can be anything that is recorded over a common discretized axis, the *so-called* 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 five 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

`paracetamol` a Raman spectrum of paracetamol (acetaminophen) ranging from 100 to 3200 cm^{-1} with some overlapping wavelength ranges.
It is used mainly in the *plotting* vignette.

`barbituates` GC-MS spectra with differing wavelength axes as a list of 286 *hyperSpec* objects.

In this vignette, the data sets are used to illustrate appropriate procedures for different tasks and different spectra.

In addition, the first three data sets are accompanied by vignetted that show exemplary work flows for the respective data type.

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

After some remarks on the notation used in the document and on the general behaviour of R, sections 3 shows how to load the package. Section 4 describes how *hyperSpec* objects are organized internally.

A list of all functions available in *hyperSpec* is given in appendix A (23)

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 **??** on page **??**.

2.2. Functionality Can be Extended at Runtime

The concept of functions in `R` offers much flexibility. Functions may be added or changed by the user in his *workspace* at any time. This is also true for methods belonging to a certain class. Neither restart of R nor reloading of the package or anything the like is needed. If the original function resides in a namespace (as it is the case for all functions in *hyperSpec*), the original function is not deleted. It is just masked by the user's new function but stays accessible via the `::` operator.

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

For examples, see the setup of the lattice plotting functions in the `vignettes.defs` file accompanying all *hyperSpec* vignettes.

2.3. Validity Checking

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.

It is highly recommended to use validity checking also for user-defined functions. In addition, non-generic functions should first ensure that the argument actually is a *hyperSpec* object. The two tasks are accomplished by:

```
> .is.hy (object)
> validObject (object)
```

The first line checks whether `object` is a *hyperSpec* object, the second checks its validity. Both functions return `TRUE` if the checks succeed, otherwise they raise an error and stop.

3. Loading the package

To load *hyperSpec*, use

```
> library (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 (tab). This also ensures 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

slot	get	set
@wavelength	`wl`	wl<-
@data	`[`, `[[`, `$`, `as.data.frame`, `as.long.df`, ...	`[<-`, `[[<-`, `$<-`
@label	`labels`	`labels<-`
@log	`logbook`	`logentry`

Table 1: Get and set functions for the slots of *hyperSpec* objects

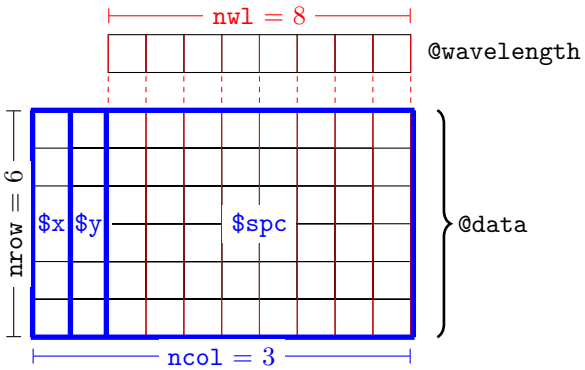


Figure 1: The structure of the data in a *hyperSpec* object.

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 "pretty" axis annotations (see figure 2 on page 13). To get familiar with expressions for axis annotation, see `? plotmath` and `demo (plotmath)`.

5. Functions provided by hyperSpec

Table A (p. 23) in the appendix gives an overview of the functions implemented by *hyperSpec*.

6. 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
   4 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (875 rows x 4 columns)
```

```
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  80.04420 81.75761 ... 1858.881
   4. clusters: clusters [factor] rng  cell   lacuna matrix + NA
```

```
> summary (chondro)
```

```
hyperSpec object
   875 spectra
   4 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (875 rows x 4 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  80.04420 81.75761 ... 1858.881
   4. clusters: clusters [factor] rng  cell   lacuna matrix + NA
log:
                   short       long                  date    user
   1   scan.txt.Renishaw   list(...   2010-02-24 10:04:06   cb@cb
   2             orderwl   list(...   2010-02-24 10:04:06   cb@cb
   3           spc.loess   list(...   2010-02-24 10:04:29   cb@cb
   4                 $<-   list(...   2010-02-24 10:04:47   cb@cb
   5                 $<-   list(...   2010-02-24 10:04:47   cb@cb
```

The data set `chondro` consists of 875 spectra with 300 data points each, and 4 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] 4
```

```
> dim (chondro)
```

```
nrow ncol  nwl
 875    4  300
```

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

```
> colnames (chondro)
```

```
[1] "y"        "x"        "spc"      "clusters"
```

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`).

7. Creating a hyperSpec Object, Data Import and Export

hyperSpec comes with filters for a variety of file formats. ASCII files are supported for import and export, Matlab files can be read and written with the help of package *R.matlab*. ENVI files can be read (for writing, see e. g. package *caTools*). Also reading of Thermo Galactic's .spc file format is supported.

E. g. to read a spectrum saved in Renishaw's ASCII file format, use:

```
> paracetamol <- scan.txt.Renishaw ("rawdata/paracetamol.txt", "spc")
> paracetamol

hyperSpec object
   1 spectra
   1 data columns
   4064 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 96.7865 98.1432 ... 3200.07
data:  (1 rows x 1 columns)
   1. spc: I / a.u. [AsIs matrix x 4064] rng  299.229 317.041 ... 49052.2

> save (paracetamol, file = "paracetamol.rda")
```

Vignette `FileIO` discusses the available filters and also how to derive own filters for importing manufacturer specific data.

7.1. Creating a hyperSpec Object from Spectra Matrix and Wavelength Vector

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

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

8. Combining and Decomposing hyperspec Objects

8.1. Binding Objects together

`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
   4 data columns
   101 data points / spectrum
```

```
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (875 rows x 4 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix101] rng  80.04420 81.75761 ... 1541.625
   4. clusters: clusters [factor] rng  cell   lacuna matrix + NA
```

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

```
hyperSpec object
   1750 spectra
   4 data columns
   50 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 798
data:  (1750 rows x 4 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix50] rng  195.5281 212.0432 ... 729.5765
   4. clusters: clusters [factor] rng  cell   lacuna matrix + NA
```

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.

collapse

8.2. Binding Objects that do not share the same extra data and/or wavelength axis

%*%

8.3. Matrix Multiplication

Two *hyperSpec* objects can be matrix multiplied by `%*%`. For an example, see the principal component analysis below (section 12.1 on page 19).

decomposition

8.4. Decomposition

Matrix decompositions are common operations during chemometric data analysis. The results, e. g. of a principal component analysis are two matrices, the so-called scores and loadings. The results can have either the same number of rows as the spectra matrix they were calculated from (scores-like), or they have as many wavelengths as the spectra (loadings-like).

Both types of result objects can be "re-imported" into *hyperSpec* objects with function `decomposition`. A scores-like object retains all per-spectrum information (i. e. the extra data) while the spectra matrix and wavelength vector are replaced. A loadings-like object retains the wavelength information, while extra data is deleted (set to `NA`) unless the value is constant for all spectra.

A demonstration can be found in the principal component analysis example (section 12.1) on page 19.

9. Access to the data

9.1. 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
   3 data columns
   181 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 495
data:  (3 rows x 3 columns)
   1. file:  [factor] rng  rawdata/flu1.txt rawdata/flu2.txt rawdata/flu3.txt
   2. spc: I[fl] / a.u. [AsIs matrix x 181] rng  27.15000 32.34467 ... 336.5057
   3. c: c / (mg / l) [numeric] rng  0.05 0.10 0.15

> flu [-3]

hyperSpec object
   5 spectra
   3 data columns
   181 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 495
data:  (5 rows x 3 columns)
   1. file:  [factor] rng  rawdata/flu1.txt rawdata/flu2.txt rawdata/flu4.txt rawdata/flu5.txt rawdata/flu6.tx
   2. spc: I[fl] / a.u. [AsIs matrix x 181] rng  27.15000 32.34467 ... 677.4947
   3. c: c / (mg / l) [numeric] rng  0.05 0.10 0.20 0.25 0.30

> chondro  [chondro$y > 10]

hyperSpec object
   350 spectra
   4 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (350 rows x 4 columns)
   1. y: y/(mu * m) [numeric] rng  10.23 11.23 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  88.98556 89.99474 ... 1745.724
   4. clusters: clusters [factor] rng  cell   lacuna matrix
```

9.2. 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"      "clusters"

> chondro [[1 : 3, 1]]

      y
1 -4.77
2 -4.77
3 -4.77

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

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

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

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

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

```
      y clusters
1 -4.77   matrix
2 -4.77   matrix
3 -4.77   matrix
```

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.

9.3. Wavelengths and Spectral Axis

9.3.1. 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] 100 101 101 101 101 102 102 102 102 103 103
```

```
> 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 ±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`.

9.3.2. Wavelength Axis Conversion

9.4. 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`).

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

`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.

10. 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 $\pm$ 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.

10.1. Plotting Spectra

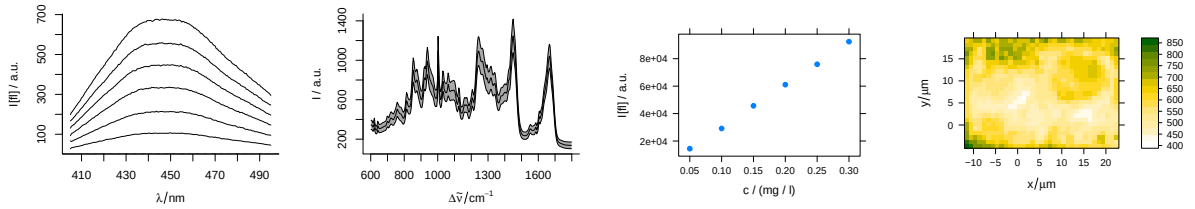
`plotspc`

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

with reversed abscissa use `wl.reverse = TRUE`

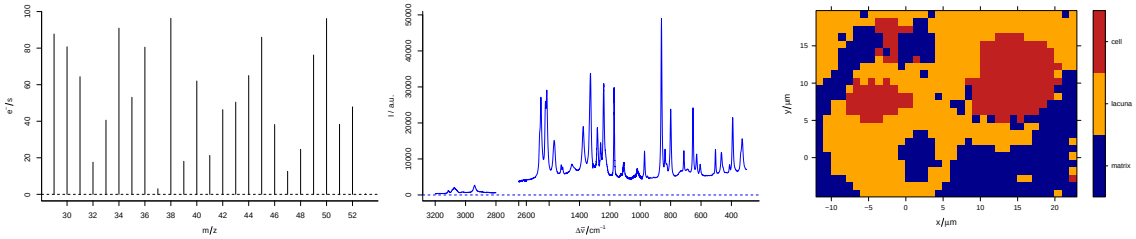
in different colours colours use `col = vector.of.colours`

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



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

Figure 2: Some example plots. (a)`plotspc (flu)`, (b)`plot (chondro, "spcmeansd")`, (c)`plotc (flu)`, and (d)`plotmap (chondro, col.regions = YG (20))` (using a yellow-green palette).



(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 12.2.

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

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*

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.

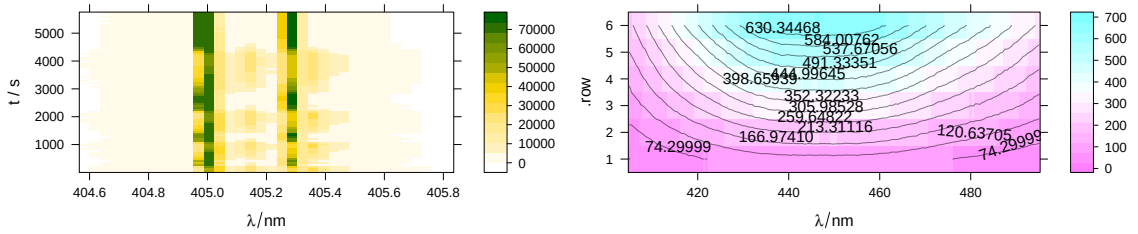
10.2. Plotting the Spectra Matrix

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 is done by `levelplot`, or by using `plot (object, "mat", model = spc ~ .wavelength * other.data.column)`. The actual plotting is done by `levelplot`, so the plots can be grouped or conditioned. See figure 4

```
> levelplot (spc ~ .wavelength * t, laser, col.regions = YG (20))
```

```
> print (plot (flu, "mat", contour = TRUE, labels = TRUE, col = "#00000080"))
```



(a) `levelplot (spc .wavelength * t, laser, col.regions = YG (20))`

(b) `plot (flu, "mat", contour = TRUE, labels = TRUE, col = "#00000080")`

Figure 4: plotting the spectra matrix.

10.3. 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*.

10.4. 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.

11. Spectral (Pre)processing

11.1. 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
   4 data columns
   8 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 408.5
data:  (6 rows x 4 columns)
   1. file:  [factor] rng  rawdata/flu1.txt rawdata/flu2.txt ... rawdata/flu6.txt
   2. spc: I[fl] / a.u. [AsIs matrix x 8] rng  27.15000 32.34467 ... 256.8913
   3. c: c / (mg / l) [numeric] rng  0.05 0.10 ... 0.3
   4. n: sample no. [integer] rng  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
```

11.2. 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.

```
> 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 5 shows the result of interpolating from 300 to 1800 and 2850 to 3150 cm^{-1} with 2 cm^{-1} data point distance. This corresponds

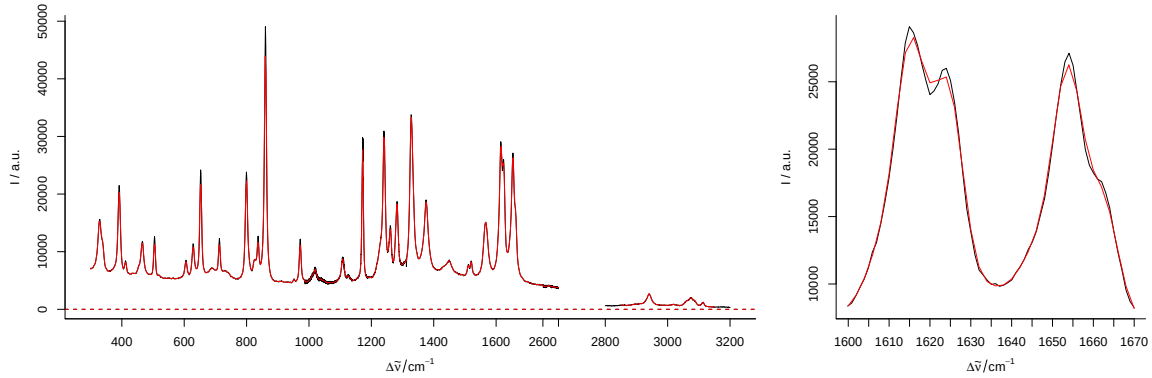


Figure 5: Smoothing interpolation by `spc.loess` with new data point spacing of 2 cm^{-1}. The magnification on the right shows how interpolation may cause a loss in signal.

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.

11.3. Background Correction

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

11.4. Offset Correction

Calculate the offsets and sweep them off the spectra:

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

11.5. 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)`.

11.6. Intensity Calibration

11.6.1. 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`

11.6.2. 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, "/")`

11.6.3. 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, "*")`

11.7. 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.

11.8. 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")
```

11.9. 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.

11.10. Multiplicative Scatter Correction (MSC)

MSC can be done using `msc` from package *pls*[1]. It operates on the spectra matrix:

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

11.11. 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. `%*%`

12. Data Analysis

12.1. 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$., center = FALSE)
```

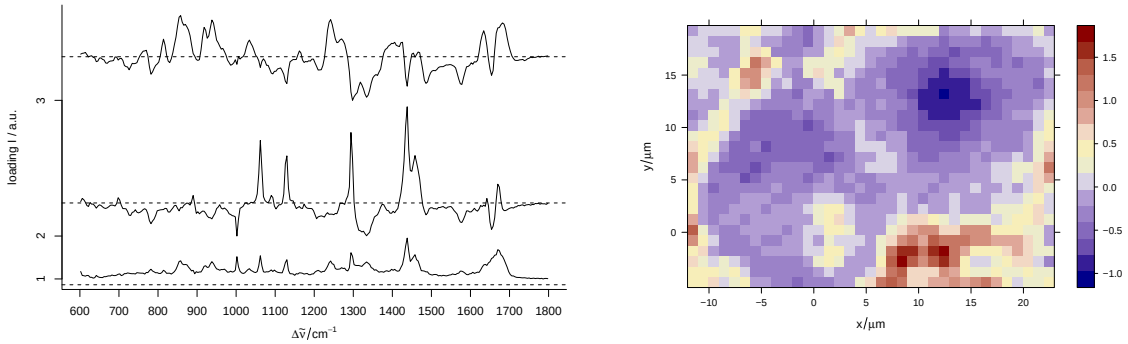
Results of such a decomposition can be put again into *hyperSpec* objects. This allows to plot e. g. the loading like spectra, or score maps, see figure 6.

```
> scores <- decomposition (chondro, pca$x, label.wavelength = "PC",
+                          label.spc = "score / a.u.")
> scores
```

```
hyperSpec object
   875 spectra
   4 data columns
   300 data points / spectrum
wavelength: PC [integer] 1 2 ... 300
data:  (875 rows x 4 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: score / a.u. [AsIs matrix x 300] rng  -2.003297 -1.173165 ... 1.918811
   4. clusters: clusters [factor] rng  cell   lacuna matrix + NA
```

The loadings can be similarly re-imported:

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



(a) (b)

Figure 6: (a) The first three loadings: `plot (loadings [1 : 3], stacked = TRUE)`. (b) The second score map: `plotmap (scores [, , 2])`

.

```
hyperSpec object
   300 spectra
   1 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (300 rows x 1 columns)
   1. spc: loading I / a.u. [AsIs matrix x 300] rng  -0.4384899 -0.3787975 ... 0.3816652
```

There is, however, one important difference. The loadings are thought of as values computed from all spectra togehter. Thus no meaningful extra data can be assigned for the loadings object (at least not if the column consists of different values). Therefore, the loadings object lost all extra data (see above).

retain.columns triggers whether columns that contain different values should be dropped. If it is set to `TRUE`, the columns are retained, but contain `NA`s:

```
> loadings <- decomposition (chondro, t(pca$rotation), scores = FALSE,
+                            retain.columns = TRUE, label.spc = "loading I / a.u.")
> loadings[1]$..

   y  x clusters
1 NA NA     <NA>
```

If an extra data column does contain only one unique value, it is retained anyways:

```
> chondro$measurement <- 1
> loadings <- decomposition (chondro, t(pca$rotation), scores = FALSE,
+                            label.spc = "loading I / a.u.")
> loadings[1]$..

  measurement
1           1
```

12.1.1. PCA as Noise Filter

Principal component analysis is sometimes used as a noise filtering technique. The idea is that the relevant differences are captured in the first components while the higher components contain noise only. Thus the spectra are reconstructed using only the first p components.

This reconstruction is in fact a matrix multiplication:

$$spectra^{(nrow \times nwl)} = scores^{(nrow \times p)} loadings^{(p \times nwl)}$$

Note that this corresponds to a model based on the Beer-Lambert law:

$$A_n(\lambda) = c_{n,i}\epsilon(i,\lambda) + error$$

The matrix formulation puts the n spectra into the rows of A and c, while the i pure components appear in the columns of c and rows of the absorbance coefficients ϵ.

For an ideal data set (constituents varying independently, sufficient signal to noise ratio) one would expect the principal component analysis to extract something like the concentrations and pure component spectra.

If we decide that only the first 10 components actually carry spectroscopic information, we can reconstruct spectra with better signal to noise ratio:

```
> smoothed <- scores [,, 1:10] %*% loadings [1:10]
```

Keep in mind, though, that we cannot be sure how much *useful* information was discarded with the higher components. This kind of noise reduction may influence further modeling of the data. Mathematically speaking, the rank of the 875 × 300 spectra matrix was reduced to 10.

12.2. 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.

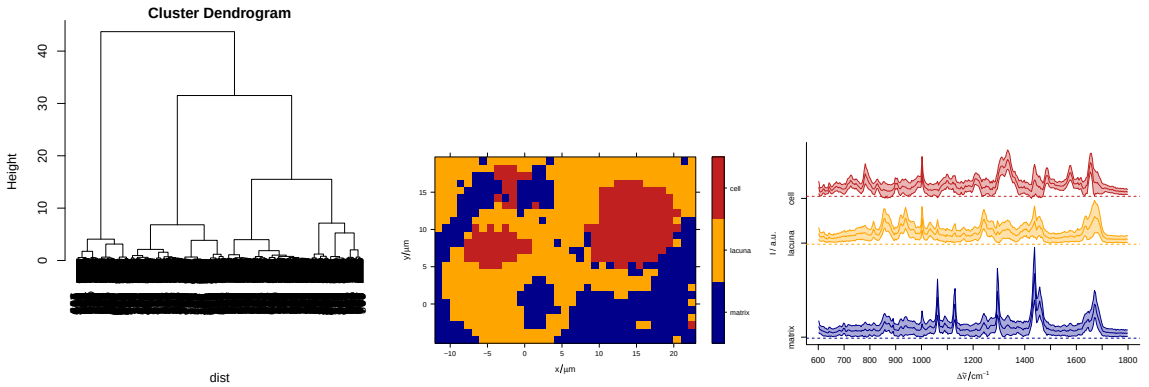
First, cut the dendrogam so that three clusters result:

```
> chondro$clusters <- as.factor (cutree (dendrogram, k = 3))
```

As the cluster membership was stored as factor, the levels can be meaningful names, which are displayed in the color legend.

```
> levels (chondro$clusters) <- c ("matrix", "lacuna", "cell")
```

Then the result may be plotted (figure 7b):



(a) (b) (c)

Figure 7: The results of the cluster analysis: (a) the dendrogram (b) the map of the 3 clusters (c) the mean spectra.

12.3. 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 = chondro$clusters, mean_pm_sd)
> plot (means, col = cluster.cols, stacked = ".aggregate", fill = ".aggregate")
```

12.4. Splitting an Object

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

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

$matrix
hyperSpec object
   292 spectra
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (292 rows x 5 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  -0.1372477 -0.1306328 ... 0.9382884
   4. clusters: clusters [factor] rng  matrix
   5. measurement: measurement [numeric] rng  1

$lacuna
hyperSpec object
   417 spectra
   5 data columns
   300 data points / spectrum
```

```
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (417 rows x 5 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  -0.2540939 -0.2373205 ... 1.043128
   4. clusters: clusters [factor] rng  lacuna
   5. measurement: measurement [numeric] rng  1

$cell
hyperSpec object
   166 spectra
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (166 rows x 5 columns)
   1. y: y/(mu * m) [numeric] rng  -2.77 5.23 ... 18.23
   2. x: x/(mu * m) [numeric] rng  -7.55 -6.55 ... 22.45
   3. spc: I / a.u. [matrix300] rng  -0.2713962 -0.2156510 ... 0.4380194
   4. clusters: clusters [factor] rng  cell
   5. measurement: measurement [numeric] rng  1
```

Splitting can be reversed by `rbind` (see section 8.1, page 7).

References

[1] Ron Wehrens and Bjørn-Helge Mevik. *pls: Partial Least Squares Regression (PLSR) and Principal Component Regression (PCR)*, 2007. URL http://mevik.net/work/software/pls.html. R package version 2.1-0.

A. Overview of the functions provided by hyperSpec

Function	Explanation
Create and initialize an object	
`initialize`	
Basic information	
`colnames`	
`colnames<-`	
`dim`	
`dimnames`	
`length`	
`ncol`	number of data columns (extra data plus spectra matrix)
`nrow`	number of spectra
`nwl`	number of data points per spectrum
`print`	summary information
`rownames`	

Function	Explanation
`show`	
`summary`	summary information including the log
Access parts of the object	
`[`	Select / extract / delete spectra, wavelength ranges or extra data
`[<-`	Set parts of spectra or extra data
`[[`	Select / extract / delete spectra, wavelength ranges or extra data, get the result as matrix or data.frame
`[[<-`	Set parts of spectra matrix
`$`	extract a data column (including `$spc`)
`$<-`	replace a data column (including `$spc`)
`i2wl`	convert spectra matrix column indices to wavelengths
`isample`	get a random sample of the spectra as index vector
`labels`	get column labels
`labels<-`	set column labels
`logbook`	logging the data treatment
`logentry`	make a logbook entry
`rownames<-`	
`sample`	generate random sample of the spectra
`seq.hyperSpec`	sequence along the spectra, either as *hyperSpec* object or index vector
`wl`	extract the wavelengths
`wl<-`	replace the wavelengths
`wl2i`	convert wavelengths to spectra matrix column indices
Type conversion	
`as.character`	
`as.data.frame`	
`as.long.df`	convert to a long-format data.frame.
`as.matrix`	
`as.wide.df`	convert to a wide-format data.frame with each wavelength one column
`decomposition`	re-import results of spectral matrix decomposition (or the like) into *hyperSpec* object
File import/export	
`\emph{R.matlab::readMat}`	import matlab file
`\emph{R.matlab::writeMat}`	export as matlab file
`read.ENVI`	import ENVI file
`read.ENVI.Nicolet`	import ENVI files writen by Nicolet spectrometers
`read.spc`	import .spc file

Function	Explanation
`read.spc.KaiserMap`	import a Raman map saved by Kaiser Optical Systems' Hologram software as multiple .spc files
`read.txt.long`	import long-type ASCII file
`read.txt.wide`	imort wide-type ASCII file
`scan.txt.Renishaw`	import ASCII files produced by Renishaw (InVia) spectrometers
`write.txt.long`	export as long-type ASCII file
`write.txt.wide`	export as wide-type ASCII file
Combine/split	
`bind`	commom interface for `rbind` and `cbind`
`cbind2`	bind two *hyperSpec* objects by column
`cbind.hyperSpec`	
`collapse`	combine objects by adding columns if necessary. See `plyr::rbind.fill`.
`rbind2`	bind two *hyperSpec* objects by row, i. e. add wavelength ranges or extra data
`rbind.hyperSpec`	bind objects by row, i. e. add wavelength ranges or extra data
`split`	
Vectorization	
`aggregate`	
`apply`	
`sweep`	
Maths	
`%*%`	matrix multiplication
`Arith`	
`log`	
`Math`	mathematical functions. See (`help (■Math■)`)
`Math2`	rounding
`Summary`	summary measures such as `min`, `max`, etc.
Comparison	
`all.equal`	
`Compare`	`> < == >= <=` return a logical matrix
`is.na`	
Plotting	
`levelplot`	
`map.identify`	identify spectra in map plot
`matlab.dark.palette`	darker version of `matlab.palette`
`matlab.palette`	palette resembling Matlab's jet colors

Function	Explanation
`plot`	main switchyard for plotting
`plotc`	intensity over one other dimension: calibration plots, time series, depth series, etc.
`plotmap`	false-colour intensity over two other dimensions: spectral images, maps, etc. (rectangular tesselation)
`plotspc`	spectra plots: intensity over wavelength
`plotvoronoi`	false-colour intensity over two other dimensions: spectral images, maps, etc. (Voronoi tesselation)
`spc.identify`	identify spectra and wavelengths in spectra plot
`stacked.offsets`	calculate intensity axis offsets for stacked spectral plots
`trellis.factor.key`	modify list of `levelplot` arguments according to factor levels
Spectra-specific transformations	
`orderwl`	sort columns of spectra matrix according to the wavelengths
`spc.bin`	spectral binning
`spc.fit.poly`	least sqares fit of a polynomial
`spc.fit.poly.below`	least sqares fit of a polynomial with automatic support point determination
`spc.loess`	`loess` smoothing interpolation
Utility functions	
`array2df`	convert array into a matrix or data.frame
`array2vec`	convert array indices (n element vector) into vector indices
`factor2num`	convert a factor with numeric levels into the numeric
`mean_pm_sd`	mean $\pm$ one standard deviation of a vector
`mean_sd`	mean and standard deviation of a vector
`pearson.dist`	distance measure based on Pearson's R^2
`rbind.fill`	transitional patch of plyr::rbind.fill working with matrices
`rbind.fill.matrix`	transitional until plyr::rbind.fill.matrix is out
`vec2array`	convert vector (one element) index into an array into an n element array index
`wc`	word count using `wc` if available on the system