# hyperSpec Introduction

Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>
DIA Raman Spectroscopy Group, University of Trieste/Italy (2005 – 2008)
Spectroscopy · Imaging, IPHT, Jena/Germany (2008 – 2017)
ÖPV, JKI, Berlin/Germany (2017 – 2019)
Arbeitskreis Lebensmittelmikrobiologie und Biotechnologie, Hamburg University, Hamburg/Germa
Chemometric Consulting and Chemometrix GmbH, Wölfersheim/Germany (since 201

February 13, 2020

**Reproducing the Examples in this Vignette**

All spectra used in this manual are installed automatically with *hyperSpec*.
Note that some definitions are executed in `vignette.defs`.

**Reporting Issues and Suggesting Enhancements**

`bug.report (package = "hyperSpec")` will take you to *hyperSpec*'s issue tracking page at https://github.com/cbeleites/hyperSpec/issues where you can report issues you encounter, suggest features and comment on issues or suggested features.

**Contents**

**Suggested Packages**

To build this vignette, some packages are suggested but not strictly needed:

| | |
|---|---|
| *pls*: | available |
| *baseline*: | available |
| *ggplot2*: | available |
| *compiler*: | available |
| *inline*: | available |

## 1. Introduction

*hyperSpec* is a R package that allows convenient handling of hyperspectral data sets, i. e. data sets combining spectra with further data on a per-spectrum basis. The spectra can be anything that is recorded over a common discretized axis.

This vignette gives an introduction on basic working techniques using the R package *hyperSpec*. This is done mostly from a spectroscopic point of view: rather than going through the functions provided by *hyperSpec*, it is organized in spectroscopic tasks. However, the functions discussed are printed on the margin for a quick overview.

*hyperSpec* 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 (acetaminophene) ranging from 100 to 3200 $\mathrm{cm}^{-1}$ with overlapping wavelength ranges.

`barbiturates` 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 their own vignettes showing exemplary work flows for the respective data type.

This document describes how to accomplish typical tasks in the analysis of spectra. 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`.

A complete list of the functions available in *hyperSpec* is given in appendix A (p. 41).

### 1.1. Notation and Terms

Throughout the documentation of the package, the following terms are used:

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

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

extra data: further information/data belonging to each spectrum
spatial information (spectral images, maps, or profiles), temporal information (kinetics, time series), concentrations (calibration series), class membership information, etc.
*hyperSpec* object may contain arbitrary numbers of extra data columns.

In R, slots of a S4 class are accessed by the `@` operator. In this vignette, the notation `@xxx` will thus mean *"slot xxx of an object"*. Likewise, named elements of a *list* and 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"* (the structure of *hyperSpec* objects is discussed in section 4, p. 6).

## 2. Remarks on R

### 2.1. Reporting an Issue with a package

`R` packages include contact information of the package maintainer, which you can access e. g. by:

```
> maintainer ("hyperSpec")
```

```
[1] "Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>"
```

In case you want to report an issue, `R` provides a function to do so. `bug.report` will either open an email to the package maintainer or the issue tracker URL given in the package `DESCRIPTION`.

```
> bug.report (package = "hyperSpec")
```

will take you to *hyperSpec*'s issue tracking page at https://github.com/cbeleites/hyperSpec/issues. It also displays essential information about your installation which can help in tracking down the bug.

We're always happy about contributions and tag issues that may be tackled immediately by "help wanted". Please note that I (Claudia, the official maintainer) may be rather slow in answering pull requests: at the moment I'm travelig a lot professionally so it may take several weeks until I can find some calm chunk of time to do more for hyperSpec than emergency fixes. However, this does not mean that I won't do so: I can tell quickly if a pull request won't fit at all into *hyperSpec*.

## 2.2. 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 appendix A (p. 41).

## 2.3. Functionality Can be Extended at Runtime

R's concept of functions 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.

The same is true for "normal" variables: You may create changed copies of the example data sets, work with these and then "reset" to *hyperSpec*'s version of the data set by removing the object in your workspace.

This offers the opportunity of easily writing specialized functions that are adapted to specific tasks. *hyperSpec*'s vignettes use this to set up special versions of the lattice graphics functions that are already wrapped in `print` (see also R FAQ: Why do lattice/trellis graphics not work?) and allow the code in the code chunks of the vignettes to be exactly what one would type during an interactive R session. For the code, check the `vignettes.defs` file accompanying all *hyperSpec* vignettes.

## 2.4. 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.

`validObject`, `chk.hy`

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:

```
> chk.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.

## 2.5. Special Function Names

### 2.5.1. The Names of Operators

Operators such as `+`, `-`,`*`, `%%`, etc. are in fact functions in R. Thus they can be handed over as arguments to other functions (particularly to the vectorization functions `*apply`, `sweep`, etc.). In this case the name of the function must be quoted: `` `-` `` is the recommended style (although `"-"` will often work as well), e. g.:

```
> sweep (flu, 2, mean, `-`)
```

These functions can also be called in a more function-like style (prefix notation):

```
> `+` (3, 5)
[1] 8
```

| name | default value (range) | description | used by |
|---|---|---|---|
| debuglevel | 0 (1L 2L) | amount of debugging information produced | `spc.identify`, `map.identify`, `spc.rubberband`, various file import functions |
| gc | FALSE | triggers frequent calling of `gc ()` | `read.ENVI`, `new ("hyperSpec")` |
| tolerance | $\sqrt{$.Machine$double.eps}$ | tolerance for numerical comparisons | file import functions (removing empty spectra), `normalize01` |
| wl.tolerance | $\sqrt{$.Machine$double.eps}$ | tolerance for comparisons of the wavelength axis | `rbind`, `rbind2`, `bind ("r", ...)`, `all.equal`, `collapse` |
| file.remove.emptyspc | TRUE | automatic removing of empty spectra | file import functions, see `vignette ("fileio")` |
| file.keep.name | TRUE | automatic recording of file name in column `$filename` | file import functions, see `vignette ("fileio")` |
| plot.spc.nmax | 25 | number of spectra to be plotted by default | `plotspc` |
| ggplot.spc.nmax | 10 | | `qplotspc` |

**Table 1** *hyperSpec* options. Please refer to the documentation of the respective functions for details about the effect of the options.

### 2.5.2. Assignment Functions

R allows the definition of functions that do an assignment (set some part of the object), such as:

```
> wl (flu) <- new.wavelength.values
```

an assignment to variable `wl`: `` `wl<-` ``.

## 3. Loading and the package and configuration

To load *hyperSpec*, use

```
> library ("hyperSpec")
```

The global behaviour of *hyperSpec* can be configured via options. The values of the options are retrieved with `hy.getOptions` and `hy.getOption`, and changed with `hy.setOptions`. Table 1 gives an overview.

## 4. The structure of hyperSpec objects

*hyperSpec* is a S4 (or new-style) class. Four slots contain the parts of the object:

`@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)

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

**Table 2** Get and set functions for the slots of *hyperSpec* objects

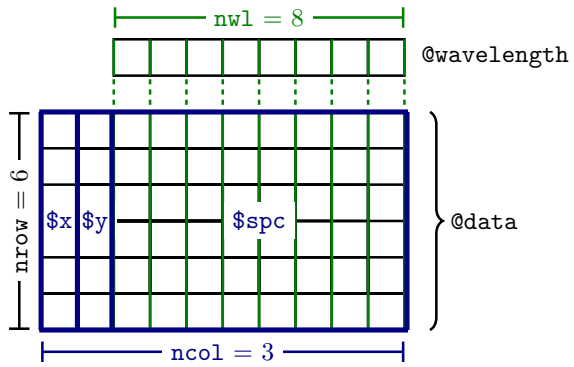



**Figure 1** The structure of the data in a *hyperSpec* object.

While the parts of the *hyperSpec* object can be accessed directly, it is good practice to use the functions provided by *hyperSpec* to handle the objects rather than accessing the slots directly (tab. 2). This also ensures that proper (*valid*) objects are returned. In some cases, however, direct access to the slots can considerably speed up calculations, see section 13 (p. 38).

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. It is then 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`. They are accessed by their names which must be the same for columns of `@data` and the list elements. 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 e. g. figure 7 on page 29). To get familiar with expressions for axis annotation, see `? plotmath` and `demo (plotmath)`.

## 5. Functions provided by hyperSpec

Table A (p. 41) 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:

`print, show, summary`

```
> chondro

hyperSpec object
   875 spectra
```

```
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (875 rows x 5 columns)
   1. y: y [numeric] -4.77 -4.77 ... 19.23
   2. x: x [numeric] -11.55 -10.55 ... 22.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] matrix matrix ... lacuna + NA
   5. spc: I / a.u. [matrix300] 501.82 500.46 ... 169.29

> summary (chondro)

hyperSpec object
   875 spectra
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (875 rows x 5 columns)
   1. y: y [numeric] -4.77 -4.77 ... 19.23
   2. x: x [numeric] -11.55 -10.55 ... 22.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] matrix matrix ... lacuna + NA
   5. spc: I / a.u. [matrix300] 501.82 500.46 ... 169.29
```

The data set `chondro` consists of 875 spectra with 300 data points each, and 5 data columns: two for the spatial information, one factor with the results of a cluster analysis plus `$spc`. These information can be directly obtained by

`nrow, ncol, nwl, dim`

```
> nrow (chondro)

[1] 875

> nwl (chondro)

[1] 300

> ncol (chondro)

[1] 5

> dim (chondro)

nrow ncol  nwl
 875    5  300
```

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

`colnames, rownames, dimnames, wl`

```
> colnames (chondro)

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

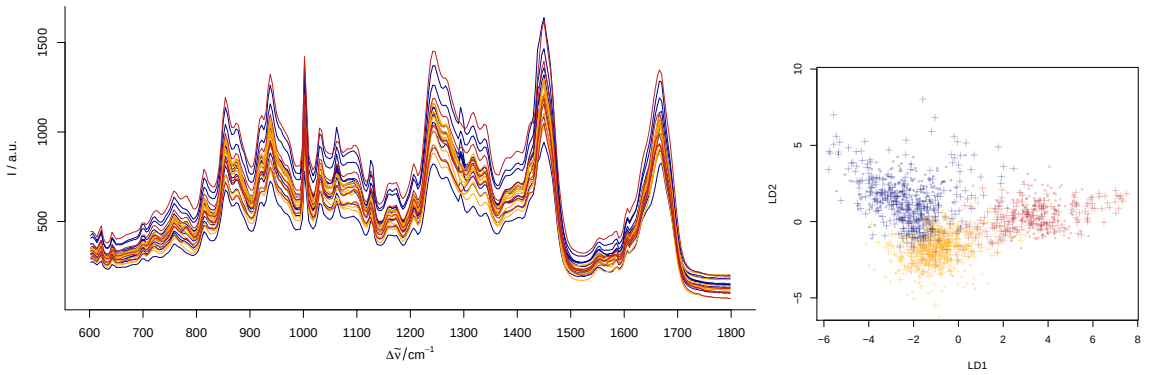
Likewise, *rownames* returns the names assigned to the spectra, and *dimnames* yields a list of these three vectors (including also the column names of `$spc`). The column names of the spectra matrix contain the wavelengths as character, while `wl` (see section 8.5.4, p. 15) yields the numeric vector of wavelengths.

Extra data column names and rownames of the object may be set by `colnames<-` and `rownames<-`, respectively. `colnames<-` renames the labels as well.

`colnames<-, rownames<-`

## 7. Creating a hyperSpec Object, Data Import and Export

*hyperSpec* comes with filters for a variety of file formats. These are discussed in detail in a separate vignette accessible via `vignette ("fileio")`.




**(a)** `rmmvnorm`

**(b)** LDA of simulated spectra. Crosses mark real spectra.

**Figure 2** Multivariate normally distributed random spectra.

### 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)
```

The most frequently needed arguments are:

| | |
|---|---|
| `spc` | the spectra matrix |
| `wavelength` | the wavelength axis vector |
| `data` | the extra data (can already contain 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`. |

More information about converting existing data into *hyperSpec* objects can be found in `vignette ("fileio")`.

### 7.2. Creating Random Spectra

If *mvtnorm* is available, multivariate normally distributed spectra can be generated from mean and covariance matrix using `rmmvnorm` (fig. 2a). Note that the *hyperSpec* function's name has an additional "m": it already takes care of multiple groups. Mean spectra and pooled covariance matrix can be calculated using `pooled.cov`:

`rmmvnorm`

`pooled.cov`

```
> pcov <- pooled.cov (chondro, chondro$clusters)
> rnd <- rmmvnorm (rep (10, 3), mean = pcov$mean, sigma = pcov$COV)

> cluster.cols <- c ("dark blue", "orange", "#C02020")
> plot (rnd, col = cluster.cols [rnd$.group])
```

fig. 2b shows the linear discriminant analysis (LDA) scores of such simulated specta in comparison to the real spectra in the `chondro` object:

```
> require ("MASS")
> rnd <- rmmvnorm (rep (200, 3), mean = pcov$mean, sigma = pcov$COV)
> lda <- lda (clusters ~ spc, rnd)
> pred.chondro <- predict (lda, chondro)
> pred.sim     <- predict (lda)
```

```
> colors <- c("#00008040", "#FFA50040", "#C0202040")
> plot (pred.chondro$x, col = colors [chondro$clusters], pch = 3)
> points (pred.sim$x, col = colors [rnd$clusters], pch = 20, cex = 0.5)
```

If individual covariance matrices should be used for each group, *sigma* should be an array with the 3rd dimension corresponding to the group.

## 8. Access to the data

`[], [[]]`

The main functions to retrieve the data of a *hyperSpec* object are `[]` and `[[]]`.

`$`

The difference between these functions is that `[]` returns a *hyperSpec* object, whereas the result of `[[]]` is a *data.frame* if extra data columns were selected or otherwise the spectra matrix. Single extra data columns may be retrieved by `$`.

`[]<-, [[]]<-, $<-`

In order to change data, use `[]<-`, `[[]]<-`, and `$<-` (see 8.4 and 8.3).

### 8.1. Access Functions and Abbreviations for Parts of the hyperSpec Object's Data

`[] [[]] $. $.. []<- [[]]<- $<-`

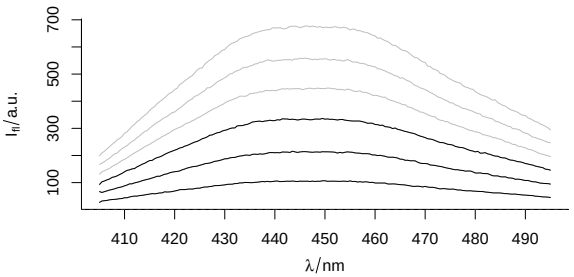
*hyperSpec* comes with three abbreviation functions for easy access to the data:

| | |
|---|---|
| `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`). The columns must match exactly in this case. |

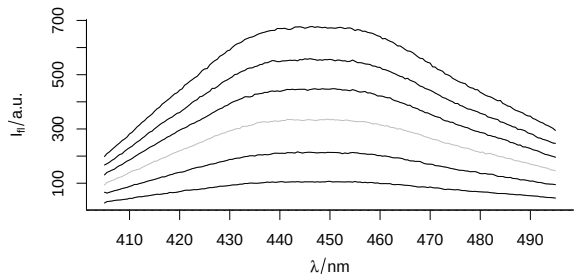
### 8.2. Selecting and Deleting Spectra

The extraction function `[]` takes the spectra as first argument (For detailed help: see `? `[`). 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. Note that a matrix given to `[]` will be treated as a vector.

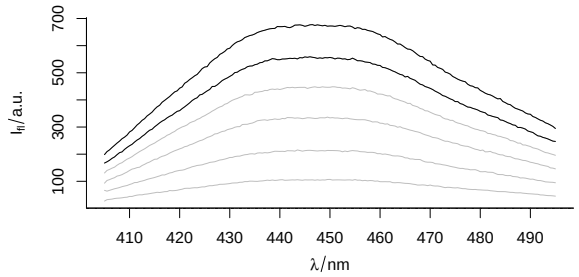
```
> plot (flu, col = "gray")
> plot (flu [1 : 3], add = TRUE)
```



```
> plot (flu, col = "gray")
> plot (flu [-3], add = TRUE)
```




```
> plot (flu, col = "gray")
> plot (flu [flu$c > 0.2], add = TRUE)
```




### 8.2.1. Random Samples

A random subset of spectra is conveniently selected by `sample` : `sample`

```
> sample (chondro, 3)
```

```
hyperSpec object
   3 spectra
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (3 rows x 5 columns)
   1. y: y [numeric] 11.23 10.23 12.23
   2. x: x [numeric] 6.45 1.45 13.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt rawdata/chondro.txt
   4. clusters: clusters [factor] matrix lacuna cell
   5. spc: I / a.u. [matrix300] 318.93 291.92 ... 108.3
```

If appropriate indices into the spectra are needed instead, use `isample`: `isample`

```
> isample (chondro, 3)
```

```
[1] 412 504 193
```

### 8.2.2. Sequences

Sequences of every $n^{th}$ spectrum or the like can be retrieved with `seq`: `seq`

```
> seq (chondro, length.out = 3, index = TRUE)
```

```
[1]   1 438 875
```

```
> seq (chondro, by = 100)
```

```
hyperSpec object
   9 spectra
   5 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (9 rows x 5 columns)
   1. y: y [numeric] -4.77 -2.77 ... 17.23
   2. x: x [numeric] -11.55 18.45 ... 18.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] matrix matrix ... lacuna
   5. spc: I / a.u. [matrix300] 501.82 400.94 ... 124.64
```

Here, indices may be requested using `index = TRUE`.

### 8.3. Selecting Extra Data Columns

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"        "filename" "clusters" "spc"
```

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

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

```
> chondro [[1 : 3, -5]]
```

```
      y      x            filename clusters
1 -4.77 -11.55 rawdata/chondro.txt   matrix
2 -4.77 -10.55 rawdata/chondro.txt   matrix
3 -4.77  -9.55 rawdata/chondro.txt   matrix
```

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

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

```
> chondro [[1 : 3, c (FALSE, TRUE)]]      # note the recycling!
```

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

*hyperSpec* supports command line completion for the `$` operator.

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.

### 8.4. More on the [[]] and [[]]<- Operators: Accessing Single Elements of the Spectra Matrix

[[]] works mostly analogous to []. In addition, however, these two functions also accept index matrices of size $n \times 2$. In this case, a vector of values from the spectra matrix is returned.

```
> indexmatrix <- matrix (c (1 : 3, 1 : 3), ncol = 2)
> indexmatrix

     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3

> chondro [[indexmatrix, wl.index = TRUE]]

[1] 501.82 507.81 456.03

> diag (chondro [[1 : 3, , min ~ min + 2i]])

[1] 501.82 507.81 456.03
```

[[]]<- also accepts index matrices of size $n \times 2$.

```
> indexmatrix <- matrix (c (1 : 3, 1 : 3), ncol = 2)
> indexmatrix

     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3

> chondro [[indexmatrix, wl.index = TRUE]]

[1] 501.82 507.81 456.03

> diag (chondro [[1 : 3, , min ~ min + 2i]])

[1] 501.82 507.81 456.03
```

### 8.5. Wavelengths

#### 8.5.1. Converting Wavelengths to Indices and vice versa

Spectra in *hyperSpec* have always discretized wavelength axes, they are stored in a matrix with each column corresponding to one wavelength. *hyperSpec* provides two functions to convert the respective column indices into wavelengths and vice versa: i2wl and wl2i. wl2i i2wl

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* ~ *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 $n^{\text{th}}$ 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 will give weird results. In that (probably 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]:

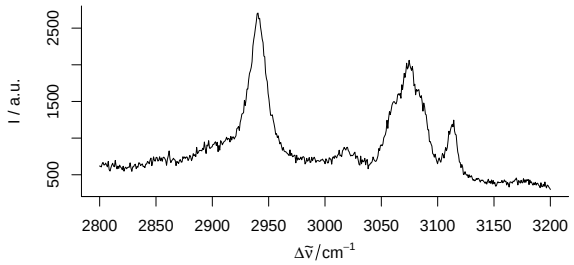
```
> wl2i (flu, c (min ~ 406.5, max - 2i ~ max))

[1]   1   2   3   4 179 180 181
```

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

### 8.5.2. Selecting Wavelength Ranges

Wavelength ranges can easily be selected using `[]`'s third argument:

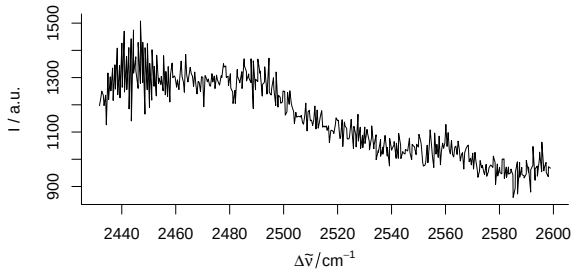
```
> plot (paracetamol [,, 2800 ~ 3200])
```




By default, the values given are treated as wavelengths. If they are indices into the columns of the spectra matrix, use `wl.index = TRUE`:

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

```
> plot (paracetamol [,, 2800 : 3200, wl.index = TRUE])
```

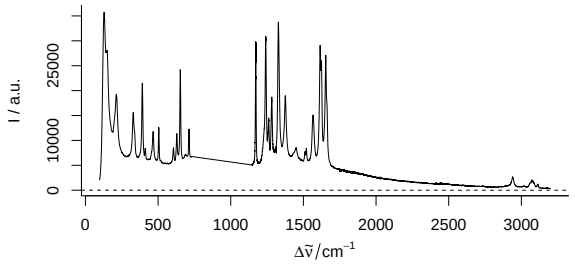



Section 8.5.1 (p. 13) details into the different possibilities of specifying wavelengths.

### 8.5.3. Deleting Wavelength Ranges

Deleting wavelength ranges may be accomplished using negative index vectors together with `wl.index = TRUE`.

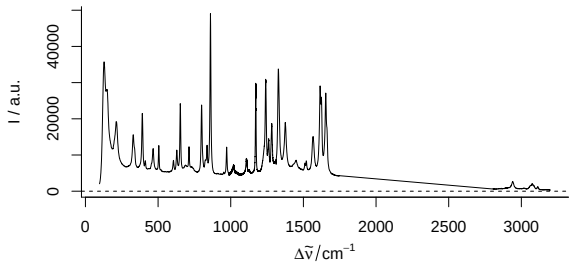
```
> plot (paracetamol [,, -(500 : 1000), wl.index = TRUE])
```




However, this mechanism works only if the proper indices are known.

If the range to be cut out is rather known in the units of the wavelength axis, it is easier to select the remainder of the spectrum instead. To delete the spectral range from 1750 to 2800 $cm^{-1}$ of the paracetamol spectrum one can thus use:

```
> plot (paracetamol [,, c (min ~ 1750, 2800 ~ max)])
```




(It is possible to produce a plot of this data where the cut range is actually omitted and the wavelength axis is optionally cut in order to save space. For details see the "plotting" vignette).

### 8.5.4. Changing the Wavelength Axis

Sometimes wavelength axes need to be transformed, e. g. converting from wavelengths to frequencies. In this case, retrieve the wavelength axis vector with `wl`, convert each value of the resulting vector `wl, wl<-`

and assign the result with `wl<-`. Also the label of the wavelength axis may need to be adjusted.

As an example, convert the wavelength axis of `laser` to frequencies. As the wavelengths are in nanometers, and the frequencies are easiest expressed in terahertz, an additional conversion factor of 1000 is needed:

```
> laser
```

```
hyperSpec object
   84 spectra
   3 data columns
   36 data points / spectrum
wavelength: lambda/nm [numeric] 404.58 404.62 ... 405.82
data:  (84 rows x 3 columns)
   1. t: t / s [numeric] 0 2 ... 5722
   2. spc: I / a.u. [matrix36] 164.65 179.72 ... 112.09
   3. filename: filename [character] rawdata/laser.txt.gz rawdata/laser.txt.gz ... rawdata/laser.txt.gz
```

```
> wavelengths <- wl (laser)
> frequencies <- 2.998e8 / wavelengths / 1000
> wl (laser) <- frequencies
> labels (laser, ".wavelength") <- "f / THz"
> laser
```

```
hyperSpec object
   84 spectra
   3 data columns
   36 data points / spectrum
wavelength: f / THz [numeric] 741.01 740.95 ... 738.76
data:  (84 rows x 3 columns)
   1. t: t / s [numeric] 0 2 ... 5722
   2. spc: I / a.u. [matrix36] 164.65 179.72 ... 112.09
   3. filename: filename [character] rawdata/laser.txt.gz rawdata/laser.txt.gz ... rawdata/laser.txt.gz
```

```
> rm (laser)
```

There are other possibilities of invoking `wl<-` including the new label, e. g.

```
> wl (laser, "f / THz") <- frequencies
```

and

```
> wl (laser) <- list (wl = frequencies, label = "f / THz")
```

see `?`wl<-`` for more information.

### 8.5.5. Ordering the Wavelength Axis

If the wavelength axis of an object needs reordering (e. g. after `collapse`), `orderwl` can be used: `orderwl`

```
> barb <- collapse (barbiturates [1 : 3])
> wl (barb)
```

```
 [1]  27.05  27.15  28.05  28.15  29.05  30.05  30.15  31.15  32.15  39.00  40.00  40.10  41.10
[14]  43.05  43.85  43.95  44.05  55.00  55.10  56.00  56.10  57.10  68.90  69.00  69.10  70.00
[27]  71.10  71.90  72.00  77.00  82.95  83.05  84.15  85.05  91.00  96.95  98.95 105.10 105.90
[40] 106.00 112.90 113.00 116.95 117.95 118.05 119.05 119.15 119.95 120.05 130.90 131.00 132.95
[53] 133.05 140.90 147.00 158.85 160.90
```

```
> barb <- orderwl (barb)
> wl (barb)
```

```
 [1]  27.05  27.15  28.05  28.15  29.05  30.05  30.15  31.15  32.15  39.00  40.00  40.10  41.10
[14]  43.05  43.85  43.95  44.05  55.00  55.10  56.00  56.10  57.10  68.90  69.00  69.10  70.00
[27]  71.10  71.90  72.00  77.00  82.95  83.05  84.15  85.05  91.00  96.95  98.95 105.10 105.90
[40] 106.00 112.90 113.00 116.95 117.95 118.05 119.05 119.15 119.95 120.05 130.90 131.00 132.95
[53] 133.05 140.90 147.00 158.85 160.90
```

### 8.6. Conversion to Long- and Wide-Format data.frames

as.data.frame

`as.data.frame` extracts the `@data` slot as a *data.frame*:

```
> flu <- flu [,,400 ~ 407] # make a small and handy version of the flu data set
> as.data.frame (flu)

  spc.405 spc.405.5 spc.406 spc.406.5 spc.407         filename    c n .row
1  27.150    32.345  33.379    34.419  36.531 rawdata/flu1.txt 0.05 1    1
2  66.801    63.715  66.712    69.582  72.530 rawdata/flu2.txt 0.10 2    2
3  93.144   103.068 106.194   110.186 113.249 rawdata/flu3.txt 0.15 3    3
4 130.664   139.998 143.798   148.420 152.133 rawdata/flu4.txt 0.20 4    4
5 167.267   171.898 177.471   184.625 189.752 rawdata/flu5.txt 0.25 5    5
6 198.430   209.458 215.785   224.587 232.528 rawdata/flu6.txt 0.30 6    6

> colnames (as.data.frame (flu))

[1] "spc"      "filename" "c"        "n"        ".row"

> as.data.frame (flu) $ spc

         405   405.5     406   406.5     407
[1,]  27.150  32.345  33.379  34.419  36.531
[2,]  66.801  63.715  66.712  69.582  72.530
[3,]  93.144 103.068 106.194 110.186 113.249
[4,] 130.664 139.998 143.798 148.420 152.133
[5,] 167.267 171.898 177.471 184.625 189.752
[6,] 198.430 209.458 215.785 224.587 232.528
```

Note that the spectra matrix is still a matrix inside column `$spc`.

`as.data.frame` and the abbreviations `$.` and `$..` retrieve the usual wide format *data.frames*: $., $..

```
> flu$.

  spc.405 spc.405.5 spc.406 spc.406.5 spc.407         filename    c n
1  27.150    32.345  33.379    34.419  36.531 rawdata/flu1.txt 0.05 1
2  66.801    63.715  66.712    69.582  72.530 rawdata/flu2.txt 0.10 2
3  93.144   103.068 106.194   110.186 113.249 rawdata/flu3.txt 0.15 3
4 130.664   139.998 143.798   148.420 152.133 rawdata/flu4.txt 0.20 4
5 167.267   171.898 177.471   184.625 189.752 rawdata/flu5.txt 0.25 5
6 198.430   209.458 215.785   224.587 232.528 rawdata/flu6.txt 0.30 6

> flu$..

          filename    c n
1 rawdata/flu1.txt 0.05 1
2 rawdata/flu2.txt 0.10 2
3 rawdata/flu3.txt 0.15 3
4 rawdata/flu4.txt 0.20 4
5 rawdata/flu5.txt 0.25 5
6 rawdata/flu6.txt 0.30 6
```

If another subset of colums needs to be extracted, use `[[]]`: [[]]

```
> flu [[, c ("c", "spc")]]

     c spc.405 spc.405.5 spc.406 spc.406.5 spc.407
1 0.05  27.150    32.345  33.379    34.419  36.531
2 0.10  66.801    63.715  66.712    69.582  72.530
3 0.15  93.144   103.068 106.194   110.186 113.249
4 0.20 130.664   139.998 143.798   148.420 152.133
5 0.25 167.267   171.898 177.471   184.625 189.752
6 0.30 198.430   209.458 215.785   224.587 232.528
```

This can be combined with extracting certain spectra and wavelengths, see below in subsection "Conversion to Matrix" on page 18.

The transpose of a wide format *data.frame* can be obtained by `as.t.df`. For further examples, see the discussion of *ggplot2* in `vignette ("plotting")`. `as.t.df`

```
> as.t.df (apply (flu, 2, mean_pm_sd))
```

```
          .wavelength mean.minus.sd   mean mean.plus.sd
spc.405         405.0        49.958 113.91       177.86
spc.405.5       405.5        53.396 120.08       186.77
spc.406         406.0        55.352 123.89       192.43
spc.406.5       406.5        57.310 128.64       199.96
spc.407         407.0        59.513 132.79       206.06
```

Some functions need the data being an *unstacked* or *long-format data.frame*. `as.long.df` is the appropriate conversion function. `as.long.df`

```
> head (as.long.df (flu), 20)
```

```
    .wavelength     spc         filename    c n
1         405.0  27.150 rawdata/flu1.txt 0.05 1
2         405.0  66.801 rawdata/flu2.txt 0.10 2
3         405.0  93.144 rawdata/flu3.txt 0.15 3
4         405.0 130.664 rawdata/flu4.txt 0.20 4
5         405.0 167.267 rawdata/flu5.txt 0.25 5
6         405.0 198.430 rawdata/flu6.txt 0.30 6
1.1       405.5  32.345 rawdata/flu1.txt 0.05 1
2.1       405.5  63.715 rawdata/flu2.txt 0.10 2
3.1       405.5 103.068 rawdata/flu3.txt 0.15 3
4.1       405.5 139.998 rawdata/flu4.txt 0.20 4
5.1       405.5 171.898 rawdata/flu5.txt 0.25 5
6.1       405.5 209.458 rawdata/flu6.txt 0.30 6
1.2       406.0  33.379 rawdata/flu1.txt 0.05 1
2.2       406.0  66.712 rawdata/flu2.txt 0.10 2
3.2       406.0 106.194 rawdata/flu3.txt 0.15 3
4.2       406.0 143.798 rawdata/flu4.txt 0.20 4
5.2       406.0 177.471 rawdata/flu5.txt 0.25 5
6.2       406.0 215.785 rawdata/flu6.txt 0.30 6
1.3       406.5  34.419 rawdata/flu1.txt 0.05 1
2.3       406.5  69.582 rawdata/flu2.txt 0.10 2
```

### 8.7. Conversion to Matrix

`as.matrix`, `[[]]`

The spectra matrix is extracted by `as.matrix`, the convenient abbreviation is `[[]]`:

```
> flu [[]]
```

```
         405   405.5     406   406.5     407
[1,]  27.150  32.345  33.379  34.419  36.531
[2,]  66.801  63.715  66.712  69.582  72.530
[3,]  93.144 103.068 106.194 110.186 113.249
[4,] 130.664 139.998 143.798 148.420 152.133
[5,] 167.267 171.898 177.471 184.625 189.752
[6,] 198.430 209.458 215.785 224.587 232.528
```

```
> class (flu [[]])
```

```
[1] "matrix"
```

`[[]]` takes the same arguments as `[]`, and can be used to extract a matrix containing parts of the spectra matrix:

```
> flu [[1:3,, 406 ~ 407]]
```

```
        406   406.5     407
[1,]  33.379  34.419  36.531
[2,]  66.712  69.582  72.530
[3,] 106.194 110.186 113.249
```

If indices for the columns to extract are given, a *data.frame* is returned instead of a matrix:

```
> flu [[1:3, c ("filename", "spc"), 406 ~ 407]]

         filename spc.406 spc.406.5 spc.407
1 rawdata/flu1.txt  33.379    34.419  36.531
2 rawdata/flu2.txt  66.712    69.582  72.530
3 rawdata/flu3.txt 106.194   110.186 113.249

> rm (flu)
```

## 9. Combining and Decomposing hyperSpec Objects

### 9.1. Binding Objects together

*hyperspec* Objects can be bound together, either by columns (`cbind`) to append a new spectral range or by row (`rbind`) to append new spectra: `cbind rbind`

```
> dim (flu)

nrow ncol  nwl
   6    3  181

> dim (cbind (flu, flu))

nrow ncol  nwl
   6    3  362

> dim (rbind (flu, flu))

nrow ncol  nwl
  12    3  181
```

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

As usual for `rbind` and `cbind`, the objects that should be bound together must have the same rows and columns, respectively.

For binding row-wise (`rbind`), `collapse` is more flexible but also faster. `collapse`

### 9.2. Binding Objects that do not Share the Same Extra Data and/or Wavelength Axis

`collapse` combines objects that should be bound together by row, but they do not share the columns and/or spectral range. The resulting object has all columns from all input objects, and all wavelengths from the input objects. If an input object does not have a particular column or wavelength, its value in the resulting object is `NA`. `collapse`

The `barbiturates` data is a list of 286 *hyperSpec* objects, each containing one mass spectrum. The spectra have between 4 and 101 data points each.

```
> barb <- collapse (barbiturates)
> wl (barb) [1 : 25]

 [1] 25.95 26.05 26.15 26.95 27.05 27.15 28.05 28.15 29.05 29.15 29.95 30.05 30.15 30.25 31.05 31.15
[17] 32.05 32.15 36.90 37.00 38.00 38.10 38.90 39.00 39.10
```

The resulting object does not have an ordered wavelength axis. This can be obtained in a second step:

```
> barb <- orderwl (barb)
> barb [[1:3, , min ~ min + 10i]]
```

```
     25.95 26.05 26.15 26.95 27.05 27.15 28.05 28.15 29.05 29.15 29.95
[1,]    NA    NA    NA    NA   562    NA    NA 11511  6146    NA    NA
[2,]    NA    NA    NA    NA    NA   618 10151    NA  5040    NA    NA
[3,]    NA    NA    NA    NA   638    NA    NA 10722  5253    NA    NA
```

### 9.3. Binding Objects that do not Share the Same Spectra

`merge` adds a new spectral range (like `cbind`), but works also if spectra are missing in one of the objects. The arguments *by*, *by.x*, and *by.y* specify which columns should be used to decide which spectra are the same. The arguments *all*, *all.x*, and *all.y* determine whether spectra should be kept for the result set if they appear in only one of the objects. For details, see also the help on the base function *merge*. `merge`

As an example, let's construct a version of the `chondro` data like being taken as two maps with different spectral ranges. In each data set, some spectra are missing.

```
> chondro.low <- sample (chondro [,, 600 ~ 1200], 700)
> nrow (chondro.low)
```

```
[1] 700
```

```
> chondro.high <- sample (chondro [,, 1400 ~ 1800], 700)
> nrow (chondro.high)
```

```
[1] 700
```

As all extra data columns are the same, no special declarations are needed for merging the data:

```
> chondro.merged <- merge (chondro.low, chondro.high)
> nrow (chondro.merged)
```

```
[1] 557
```

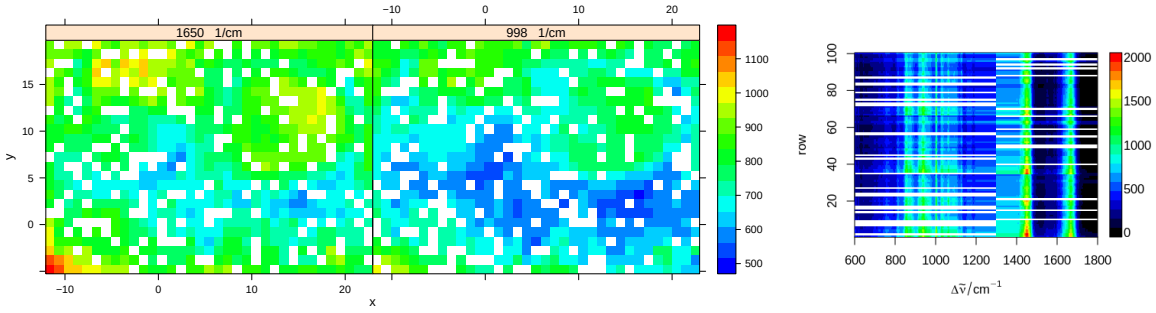
By default, the result consists of only those spectra, where *both* spectral ranges were available. To keep all spectra replacing missing parts by `NA` (see fig. 3):

```
> chondro.merged <- merge (chondro.low, chondro.high, all = TRUE)
> nrow (chondro.merged)
```

```
[1] 843
```

```
> merged <- merge (chondro [1:7,, 610 ~ 620], chondro [5:10,, 615 ~ 625], all = TRUE)
> merged$.
```

```
       y      x            filename clusters .nx .ny spc.610 spc.614 spc.618 spc.614 spc.618
1  -4.77 -11.55 rawdata/chondro.txt   matrix   1  NA  488.63  466.18  492.00      NA      NA
2  -4.77 -10.55 rawdata/chondro.txt   matrix   2  NA  489.48  465.05  490.53      NA      NA
3  -4.77  -9.55 rawdata/chondro.txt   matrix   3  NA  456.03  436.62  458.06      NA      NA
4  -4.77  -8.55 rawdata/chondro.txt   matrix   4  NA  464.82  444.85  470.02      NA      NA
5  -4.77  -7.55 rawdata/chondro.txt   matrix   5   1  428.66  410.80  433.12  410.80  433.12
6  -4.77  -6.55 rawdata/chondro.txt   matrix   6   2  426.07  407.86  431.21  407.86  431.21
7  -4.77  -5.55 rawdata/chondro.txt   lacuna   7   3  412.37  396.50  421.27  396.50  421.27
8  -4.77  -4.55 rawdata/chondro.txt   lacuna  NA   4      NA      NA      NA  381.95  406.25
9  -4.77  -3.55 rawdata/chondro.txt   lacuna  NA   5      NA      NA      NA  397.51  423.30
10 -4.77  -2.55 rawdata/chondro.txt   lacuna  NA   6      NA      NA      NA  377.39  402.23
   spc.622 spc.626
1       NA      NA
```




**(a)** **(b)**

**Figure 3** (a) For both spectral ranges some spectra are missing. (b) The missing parts of the spectra are filled with `NA`.

```
2       NA      NA
3       NA      NA
4       NA      NA
5   461.19  397.38
6   458.15  394.18
7   445.54  382.72
8   429.67  368.46
9   446.15  381.87
10  424.19  362.43
```

If the spectra overlap, the result will have both data points. In the example here one could easily delete duplicate wavelengths. For real data, however, the duplicated wavelength will hardly ever contain the same values. The appropriate method to deal with this situation depends on the data at hand, but it will usually be some kind of spectral interpolation.

One possibility is removing duplicated wavelengths by using the mean intensity. This can conveniently be done by using `approx` using `method = "constant"`. For duplicated wavelengths, the intensities will be combined by the tie function. This already defaults to the mean, but we need `na.rm = TRUE`.

Thus, the function to calculate the new spectral intensities is

```
> approxfun <- function (y, wl, new.wl){
+   approx (wl, y, new.wl, method = "constant",
+           ties = function (x) mean (x, na.rm = TRUE)
+           )$y
+ }
```

which can be applied to the spectra:

```
> merged <- apply (merged, 1, approxfun,
+                  wl = wl (merged), new.wl = unique (wl (merged)),
+                  new.wavelength = "new.wl")
> merged$.
```

```
      y      x            filename clusters .nx .ny spc.610 spc.614 spc.618 spc.622 spc.626
1  -4.77 -11.55 rawdata/chondro.txt   matrix   1   NA  488.63  466.18  492.00      NA      NA
2  -4.77 -10.55 rawdata/chondro.txt   matrix   2   NA  489.48  465.05  490.53      NA      NA
3  -4.77  -9.55 rawdata/chondro.txt   matrix   3   NA  456.03  436.62  458.06      NA      NA
4  -4.77  -8.55 rawdata/chondro.txt   matrix   4   NA  464.82  444.85  470.02      NA      NA
5  -4.77  -7.55 rawdata/chondro.txt   matrix   5    1  428.66  410.80  433.12  461.19  397.38
6  -4.77  -6.55 rawdata/chondro.txt   matrix   6    2  426.07  407.86  431.21  458.15  394.18
7  -4.77  -5.55 rawdata/chondro.txt   lacuna   7    3  412.37  396.50  421.27  445.54  382.72
```

```
8  -4.77  -4.55 rawdata/chondro.txt   lacuna  NA   4      NA  381.95  406.25  429.67  368.46
9  -4.77  -3.55 rawdata/chondro.txt   lacuna  NA   5      NA  397.51  423.30  446.15  381.87
10 -4.77  -2.55 rawdata/chondro.txt   lacuna  NA   6      NA  377.39  402.23  424.19  362.43
```

### 9.4. Merging extra data to objects that do not (necessarily) share the same spectra

Assume we obtained duplicate reference measurements for some of the concentrations in `flu`:

```
> flu.ref <- data.frame (filename = rep (flu$filename[1:2], each = 2), cref = rep (flu$c [1:2], each = 2) + rn
> flu.ref

         filename     cref
1 rawdata/flu1.txt 0.047697
2 rawdata/flu1.txt 0.081835
3 rawdata/flu2.txt 0.091149
4 rawdata/flu2.txt 0.104366
```

This information can be `merge`d into the extra data of `flu` by:

```
> flu.merged <- merge (flu, flu.ref)
> flu.merged$..

         filename    c     cref
1 rawdata/flu1.txt 0.05 0.047697
2 rawdata/flu1.txt 0.05 0.081835
3 rawdata/flu2.txt 0.10 0.091149
4 rawdata/flu2.txt 0.10 0.104366
```

The usual rules for `merge` apply. E. g., if to preserver all spectra of flu, use `all.x = TRUE`:

```
> flu.merged <- merge (flu, flu.ref, all.x = TRUE)
> flu.merged$..

         filename    c     cref
1 rawdata/flu1.txt 0.05 0.047697
2 rawdata/flu1.txt 0.05 0.081835
3 rawdata/flu2.txt 0.10 0.091149
4 rawdata/flu2.txt 0.10 0.104366
5 rawdata/flu3.txt 0.15       NA
6 rawdata/flu4.txt 0.20       NA
7 rawdata/flu5.txt 0.25       NA
8 rawdata/flu6.txt 0.30       NA
```

The class of the first object (`x`) determines the resulting class:

```
> merge (flu, flu.ref)

hyperSpec object
   4 spectra
   4 data columns
   181 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 495
data:  (4 rows x 4 columns)
   1. filename: filename [character] rawdata/flu1.txt rawdata/flu1.txt rawdata/flu2.txt rawdata/flu2.txt
   2. spc: I[fl]/"a.u." [matrix181] 27.15 27.15 ... 94.61
   3. c: c / (mg / l) [numeric] 0.05 0.05 0.10 0.10
   4. cref:  [numeric] 0.047697 0.081835 0.091149 0.104366

> merge (flu.ref, flu)

         filename     cref spc.405 spc.405.5 spc.406 spc.406.5 spc.407 spc.407.5 spc.408 spc.408.5
1 rawdata/flu1.txt 0.047697  27.150    32.345  33.379    34.419  36.531    37.648  38.137    39.177
2 rawdata/flu1.txt 0.081835  27.150    32.345  33.379    34.419  36.531    37.648  38.137    39.177
3 rawdata/flu2.txt 0.091149  66.801    63.715  66.712    69.582  72.530    74.558  77.048    80.260
```

```
4 rawdata/flu2.txt 0.104366  66.801    63.715  66.712    69.582  72.530    74.558  77.048    80.260
  spc.409 spc.409.5 spc.410 spc.410.5 spc.411 spc.411.5 spc.412 spc.412.5 spc.413 spc.413.5 spc.414
1  40.736    41.381  44.251    44.126  46.981    49.082  50.274    50.110  52.232    53.040  54.519
2  40.736    41.381  44.251    44.126  46.981    49.082  50.274    50.110  52.232    53.040  54.519
3  82.539    84.492  88.152    91.085  95.372    95.530  98.995   101.034 103.558   107.027 109.545
4  82.539    84.492  88.152    91.085  95.372    95.530  98.995   101.034 103.558   107.027 109.545
  spc.414.5 spc.415 spc.415.5 spc.416 spc.416.5 spc.417 spc.417.5 spc.418 spc.418.5 spc.419
1    56.220  57.719    59.514  58.745    60.095  61.841    62.169  66.049    65.245  65.799
2    56.220  57.719    59.514  58.745    60.095  61.841    62.169  66.049    65.245  65.799
3   111.672 113.987   117.033 119.835   123.494 123.324   127.002 130.681   132.358 136.165
4   111.672 113.987   117.033 119.835   123.494 123.324   127.002 130.681   132.358 136.165
  spc.419.5 spc.420 spc.420.5 spc.421 spc.421.5 spc.422 spc.422.5 spc.423 spc.423.5 spc.424
1    66.943  69.136    70.925  73.834    74.386  74.173    75.531  76.430    76.132  77.891
2    66.943  69.136    70.925  73.834    74.386  74.173    75.531  76.430    76.132  77.891
3   138.399 140.224   141.043 143.131   147.180 150.012   153.693 155.728   158.472 159.501
4   138.399 140.224   141.043 143.131   147.180 150.012   153.693 155.728   158.472 159.501
  spc.424.5 spc.425 spc.425.5 spc.426 spc.426.5 spc.427 spc.427.5 spc.428 spc.428.5 spc.429
1    79.369  79.427    82.028  83.878    83.814  85.507    86.502  88.937    88.995  89.515
2    79.369  79.427    82.028  83.878    83.814  85.507    86.502  88.937    88.995  89.515
3   160.816 163.910   166.255 169.592   170.828 172.088   175.211 178.471   179.681 181.829
4   160.816 163.910   166.255 169.592   170.828 172.088   175.211 178.471   179.681 181.829
  spc.429.5 spc.430 spc.430.5 spc.431 spc.431.5 spc.432 spc.432.5 spc.433 spc.433.5 spc.434
1    90.557  91.706    93.579  94.013    94.210  96.442    96.627  98.962    98.848 100.568
2    90.557  91.706    93.579  94.013    94.210  96.442    96.627  98.962    98.848 100.568
3   184.429 187.756   187.938 191.057   192.509 192.801   195.556 197.891   198.782 200.995
4   184.429 187.756   187.938 191.057   192.509 192.801   195.556 197.891   198.782 200.995
  spc.434.5 spc.435 spc.435.5 spc.436 spc.436.5 spc.437 spc.437.5 spc.438 spc.438.5 spc.439
1   100.449  99.916    99.719 100.886   101.445 102.643   104.230 103.111   104.758 103.158
2   100.449  99.916    99.719 100.886   101.445 102.643   104.230 103.111   104.758 103.158
3   201.522 200.206   204.132 204.904   203.988 205.756   204.817 206.091   208.607 210.797
4   201.522 200.206   204.132 204.904   203.988 205.756   204.817 206.091   208.607 210.797
  spc.439.5 spc.440 spc.440.5 spc.441 spc.441.5 spc.442 spc.442.5 spc.443 spc.443.5 spc.444
1   103.943 105.371   105.863 103.680   104.090 104.654   105.226 104.804   105.810 104.758
2   103.943 105.371   105.863 103.680   104.090 104.654   105.226 104.804   105.810 104.758
3   210.584 210.190   210.542 209.859   212.827 212.521   212.420 211.222   215.564 213.425
4   210.584 210.190   210.542 209.859   212.827 212.521   212.420 211.222   215.564 213.425
  spc.444.5 spc.445 spc.445.5 spc.446 spc.446.5 spc.447 spc.447.5 spc.448 spc.448.5 spc.449
1   103.883 105.362   105.015 105.059   105.349 103.987   105.439 104.197   105.098 104.723
2   103.883 105.362   105.015 105.059   105.349 103.987   105.439 104.197   105.098 104.723
3   211.159 212.975   213.989 214.172   213.476 212.489   211.741 212.705   212.940 212.129
4   211.159 212.975   213.989 214.172   213.476 212.489   211.741 212.705   212.940 212.129
  spc.449.5 spc.450 spc.450.5 spc.451 spc.451.5 spc.452 spc.452.5 spc.453 spc.453.5 spc.454
1   106.667 106.950   104.755 105.083   105.300 105.213   104.781 104.539   105.133 105.170
2   106.667 106.950   104.755 105.083   105.300 105.213   104.781 104.539   105.133 105.170
3   213.773 213.497   213.464 213.171   212.836 211.963   208.799 211.506   209.477 211.860
4   213.773 213.497   213.464 213.171   212.836 211.963   208.799 211.506   209.477 211.860
  spc.454.5 spc.455 spc.455.5 spc.456 spc.456.5 spc.457 spc.457.5 spc.458 spc.458.5 spc.459
1   104.057 106.385   104.080 104.401   102.181 103.442   101.797 102.872   102.389 100.419
2   104.057 106.385   104.080 104.401   102.181 103.442   101.797 102.872   102.389 100.419
3   213.262 212.284   211.773 209.391   208.856 208.340   206.506 206.777   206.645 205.255
4   213.262 212.284   211.773 209.391   208.856 208.340   206.506 206.777   206.645 205.255
  spc.459.5 spc.460 spc.460.5 spc.461 spc.461.5 spc.462 spc.462.5 spc.463 spc.463.5 spc.464
1   101.162  98.611    98.429  98.576    98.341  98.467    95.149  94.711    95.274  94.884
2   101.162  98.611    98.429  98.576    98.341  98.467    95.149  94.711    95.274  94.884
3   201.747 201.988   199.570 200.551   198.768 197.033   194.145 193.383   193.665 192.513
4   201.747 201.988   199.570 200.551   198.768 197.033   194.145 193.383   193.665 192.513
  spc.464.5 spc.465 spc.465.5 spc.466 spc.466.5 spc.467 spc.467.5 spc.468 spc.468.5 spc.469
1    93.621  92.466    92.115  91.309    89.539  88.281    86.136  86.162    86.817  85.958
2    93.621  92.466    92.115  91.309    89.539  88.281    86.136  86.162    86.817  85.958
3   190.633 186.456   186.054 182.305   182.334 180.569   180.506 180.213   175.211 171.650
4   190.633 186.456   186.054 182.305   182.334 180.569   180.506 180.213   175.211 171.650
  spc.469.5 spc.470 spc.470.5 spc.471 spc.471.5 spc.472 spc.472.5 spc.473 spc.473.5 spc.474
1    84.067  86.212    83.683  82.569    82.705  81.260    78.752  76.675    77.953  77.507
2    84.067  86.212    83.683  82.569    82.705  81.260    78.752  76.675    77.953  77.507
3   170.417 170.374   169.407 164.890   164.141 163.173   160.989 159.665   157.746 155.135
```

```
4   170.417 170.374   169.407 164.890   164.141 163.173   160.989 159.665   157.746 155.135
  spc.474.5 spc.475 spc.475.5 spc.476 spc.476.5 spc.477 spc.477.5 spc.478 spc.478.5 spc.479
1    76.164  75.646    76.437  74.570    72.879  72.803    71.052  70.214    69.605  69.990
2    76.164  75.646    76.437  74.570    72.879  72.803    71.052  70.214    69.605  69.990
3   153.239 151.523   149.197 147.495   147.490 145.434   144.819 142.439   142.690 142.881
4   153.239 151.523   149.197 147.495   147.490 145.434   144.819 142.439   142.690 142.881
  spc.479.5 spc.480 spc.480.5 spc.481 spc.481.5 spc.482 spc.482.5 spc.483 spc.483.5 spc.484
1    68.780  68.342    67.685  67.277    67.048  65.313    64.509  63.506    62.184  62.045
2    68.780  68.342    67.685  67.277    67.048  65.313    64.509  63.506    62.184  62.045
3   139.838 135.479   135.253 135.846   133.559 133.318   132.824 128.930   125.203 124.740
4   139.838 135.479   135.253 135.846   133.559 133.318   132.824 128.930   125.203 124.740
  spc.484.5 spc.485 spc.485.5 spc.486 spc.486.5 spc.487 spc.487.5 spc.488 spc.488.5 spc.489
1    62.027  61.799    60.528  59.342    59.125  57.748    57.447  57.743    56.275  55.492
2    62.027  61.799    60.528  59.342    59.125  57.748    57.447  57.743    56.275  55.492
3   124.717 121.678   120.902 121.406   117.668 117.041   115.941 112.229   112.724 111.673
4   124.717 121.678   120.902 121.406   117.668 117.041   115.941 112.229   112.724 111.673
  spc.489.5 spc.490 spc.490.5 spc.491 spc.491.5 spc.492 spc.492.5 spc.493 spc.493.5 spc.494
1    54.409  53.833    53.263  52.457    52.140  49.784    49.623  48.338    47.304  47.163
2    54.409  53.833    53.263  52.457    52.140  49.784    49.623  48.338    47.304  47.163
3   109.698 107.641   108.051 105.220   102.953 102.758   100.628  97.965    97.353  96.602
4   109.698 107.641   108.051 105.220   102.953 102.758   100.628  97.965    97.353  96.602
  spc.494.5 spc.495    c
1    46.412  45.256 0.05
2    46.412  45.256 0.05
3    96.206  94.610 0.10
4    96.206  94.610 0.10
```

### 9.5. Matrix Multiplication

Two *hyperSpec* objects can be matrix multiplied by `%*%`. For an example, see the principal component analysis below (section 12.1 on page 33). `%*%`

### 9.6. 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. `decomposition`

A demonstration can be found in the principal component analysis example (section 12.1) on page 33.

## 10. Plotting

*hyperSpec* offers a variety of possibilities to plot spectra, spectral maps, the spectra matrix, time series, depth profiles, etc.. This all is discussed in a separate document: see `vignette ("plotting")`.

## 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 in addition data columns were specified) only.

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

hyperSpec object
   6 spectra
   3 data columns
   8 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 408.5
data:  (6 rows x 3 columns)
   1. spc: I[fl]/"a.u." [matrix8] 27.150 66.801 ... 256.89
   2. filename: filename [character] rawdata/flu1.txt rawdata/flu2.txt ... rawdata/flu6.txt
   3. c: c / (mg / l) [numeric] 0.05 0.10 ... 0.3

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

         405   405.5     406     494   494.5     495
[1,]  27.150  32.345  33.379  47.163  46.412  45.256
[2,]  66.801  63.715  66.712  96.602  96.206  94.610
[3,]  93.144 103.068 106.194 149.539 148.527 145.793
[4,] 130.664 139.998 143.798 201.484 198.867 195.867
[5,] 167.267 171.898 177.471 252.066 248.067 246.952
[6,] 198.430 209.458 215.785 307.519 302.325 294.649
```

### 11.2. Shifting Spectra

Sometimes, spectra need to be aligned along the spectral axis.

In general, two options are available for shifting spectra along the wavelength axis.

1. The wavelength axis can be shifted, while the intensities stay unaffected.
2. the spectra are interpolated onto a new wavelength axis, while the nominal wavelengths stay.

The first method is very straightforward (see fig 4a):

```
> tmp <- chondro
> wl (tmp) <- wl (tmp) - 10
```

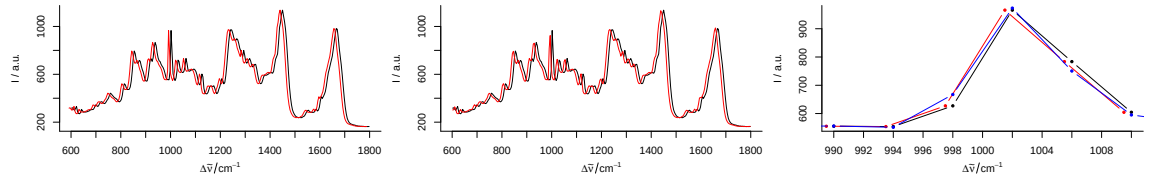
but it cannot be used if each spectrum (or groups of spectra) are shifted individually.

In that case, interpolation is needed. R offers many possibilities to interpolate (e. g. `approx` for constant / linear approximation, `spline` for spline interpolation, `loess` can be used to obtain smoothed approximations, etc.). The appropriate interpolation strategy will depend on the spectra, and *hyperSpec* therefore leaves it up to the user to select a sensible interpolation function.

As an example, we will use natural splines to do the interpolation. It is convenient to set it up as a function:

```
> interpolate <- function (spc, shift, wl){
+   spline (wl + shift, spc, xout = wl, method = "natural")$y
+ }
```

This function can now be applied to a set of spectra (see fig 4b):




**(a)** `wl<-` **(b)** interpolation **(c)**

**Figure 4** Shifting the Spectra along the Wavelength Axis. (a) Changing the wavelength values. (b) Interpolation. (c) Detail view of the phenylalanine band: shifting by `wl<-` (red) does not affect the intensities, while the spectrum is slightly changed by interpolations (blue).

```
> tmp <- apply (chondro, 1, interpolate, shift = -10, wl = wl (chondro))
```

If different spectra need to be offset by different shift, use a loop[2]

```
> shifts <- rnorm (nrow (chondro))
> tmp <- chondro [[]]
> for (i in seq_len (nrow (chondro)))
+   tmp [i, ] <- interpolate (tmp [i, ], shifts [i], wl = wl (chondro))
> chondro [[]] <- tmp
```

### 11.2.1. Calculating the Shift

Often, the shift in the spectra is determined by aligning a particular signal. This strategy works best with spectrally oversampled data that allows accurate determination of the signal position.

For the `chondro` data, let's use the maximum of the phenylalanine band between 990 and $1020\,\mathrm{cm}^{-1}$. As just the very maximum is too coarse, we'll use the maximum of a square polynomial fitted to the maximum and its two neighbours.

```
> find.max <- function (y, x){
+   pos <- which.max (y) + (-1:1)
+   X <- x [pos] - x [pos [2]]
+   Y <- y [pos] - y [pos [2]]
+
+   X <- cbind (1, X, X^2)
+   coef <- qr.solve (X, Y)
+
+   - coef [2] / coef [3] / 2 + x [pos [2]]
+ }
> bandpos <- apply (chondro [[,, 990 ~ 1020]], 1, find.max,  wl (chondro [,, 990 ~ 1020]))
> refpos <- find.max (colMeans (chondro[[,, 990 ~ 1020]]),  wl (chondro [,, 990 ~ 1020]))
> shift1 <- refpos - bandpos
```

A second possibility is to optimize the shift. For this strategy, the spectra must be sufficiently similar, while low spectral resolution is compensated by using larger spectral windows.

```
> chondro <- chondro - spc.fit.poly.below (chondro [,,min+3i ~ max - 3i], chondro)
> chondro <- sweep (chondro, 1, rowMeans (chondro [[]], na.rm = TRUE), "/")
```

[2] `sweep` cannot be used here, and while there is the possibility to use sapply or mapply, they are not faster than the for loop in this case. Make sure to work on a copy of the spectra matrix, as that is much faster than row-wise extracting and changing the spectra by `[[` and `[[<-`.

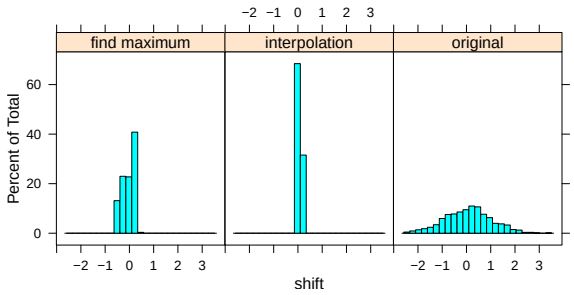



**Figure 5** The shifts used to disturb the chondrocyte data (original), and the remaining shift after correction with the two methods discussed here.

```
> targetfn <- function (shift, wl, spc, targetspc){
+   error <- spline (wl + shift, spc, xout = wl)$y - targetspc
+   sum (error^2)
+ }
> shift2 <- numeric (nrow (chondro))
> tmp <- chondro [[]]
> target <- colMeans (chondro [[]])
> for (i in 1 : nrow (chondro))
+   shift2 [i] <- unlist (optimize (targetfn, interval = c (-5, 5), wl = chondro@wavelength,
+                                   spc = tmp[i,], targetspc = target)$minimum)
```

Figure 5 shows that the second correction method works better for the chondrocyte data. This was expected, as the spectra are hardly or not oversampled, but are very similar to each other.

## 11.3. Removing Bad Data

### 11.3.1. Bad Spectra

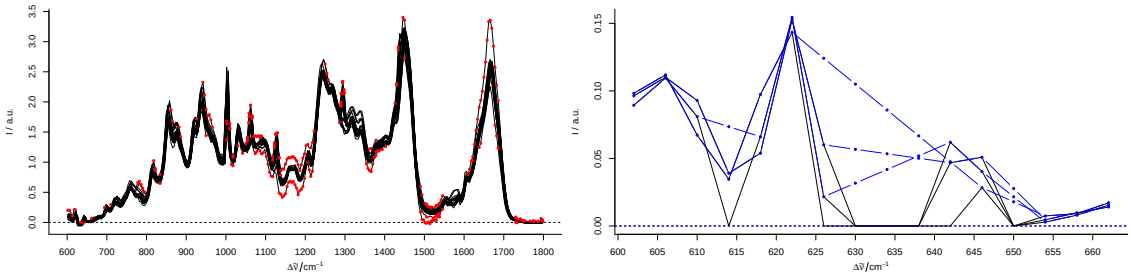
Occasionally, one may want to remove spectra because of too low or too high signal.

E.g. for infrared spectra one may state that the absorbance maximum should be, say, between 0.1 and 1. *hyperSpec*'s comparison operators return a logical matrix of the size of the spectra that is suitable for later indexing:

```
> ir.spc <- chondro / 1500 ## fake IR data
> high.int <- apply (ir.spc > 1, 1, any) # any point above 1 is bad
> low.int <- apply (ir.spc, 1, max) < 0.1 # the maximum should be at least 0.1
> ir.spc <- ir.spc [! high.int & ! low.int]
```

### 11.3.2. Removing Spectra outside mean $\pm$ $n$ sd

```
> mean_sd_filter <- function (x, n  = 5) {
+   x <- x - mean (x)
+   s <- n * sd (x)
+   (x <= s) & (x > -s)
+ }
> OK <- apply (chondro [[]], 2, mean_sd_filter, n = 4) # logical matrix
> spc.OK <- chondro [apply (OK, 1, all)]

> plot (chondro [! apply (OK, 1, all)])
> i <- which (! OK, arr.ind = TRUE)
> points (wl (chondro) [i [,2]], chondro[[!OK]], pch = 19, col = "red", cex = 0.5)
```




**(a)** mean ± sd filter **(b)** remove bad points

**Figure 6** filtering data

### 11.3.3. Bad Data Points

Assume the data contains once in a while a detector readout of 0:

```
> spc <- chondro [1 : 3,, min ~ min + 15i]
> spc [[cbind (1:3, sample (nwl (spc), 3)), wl.index = TRUE]] <- 0
> spc [[]]

          602      606       610       614       618      622         626         630         634         638
[1,] 0.096345 0.11011 0.092900 0.039016 0.053884 0.15268  0.0600764 -0.041852 -0.051207 -0.0431862
[2,] 0.089393 0.10953 0.080946 0.000000 0.066067 0.15438  0.0216284 -0.048903 -0.058006 -0.0350570
[3,] 0.098244 0.11182 0.067290 0.034626 0.097359 0.14334 -0.0082548 -0.051172 -0.056180 -0.0095044
          642       646          650       654       658       662
[1,] 0.046817 0.050862  0.0000e+00 0.0045712 0.0098066 0.014035
[2,] 0.061956 0.040337 -1.6737e-03 0.0028803 0.0080017 0.015405
[3,] 0.000000 0.028361  7.3094e-05 0.0074266 0.0091003 0.017158
```

We can set these points to `NA`, again using that the comparison returns a suitable logical matrix:

```
> spc [[spc < 1e-4]] <- NA
> spc [[]]

          602      606       610       614       618      622       626 630 634 638       642       646 650
[1,] 0.096345 0.11011 0.092900 0.039016 0.053884 0.15268 0.060076  NA  NA  NA 0.046817 0.050862  NA
[2,] 0.089393 0.10953 0.080946       NA 0.066067 0.15438 0.021628  NA  NA  NA 0.061956 0.040337  NA
[3,] 0.098244 0.11182 0.067290 0.034626 0.097359 0.14334       NA  NA  NA  NA        NA 0.028361  NA
           654       658       662
[1,] 0.0045712 0.0098066 0.014035
[2,] 0.0028803 0.0080017 0.015405
[3,] 0.0074266 0.0091003 0.017158
```

Depending on the type of analysis, one may wants to replace the `NA`s by interpolating the neighbour values. So far, *hyperSpec* provides three functions that can interpolate the `NA`s: : `spc.NA.approx`, `spc.loess`, and `spc.bin` with `na.rm = TRUE` (the latter two are discussed below).

`spc.NA.approx`, `spc.loess`, `spc.bin`

```
> if (!exists("spc.NA.approx")){
+   spc.NA.approx <- spc.NA.linapprox
+ }

> spc.corrected <- spc.NA.approx (spc)
> spc.corrected [[]]

          602      606       610       614       618      622       626       630       634       638
[1,] 0.096345 0.11011 0.092900 0.039016 0.053884 0.15268 0.060076 0.056761 0.053447 0.050132
[2,] 0.089393 0.10953 0.080946 0.073506 0.066067 0.15438 0.021628 0.031710 0.041792 0.051874
[3,] 0.098244 0.11182 0.067290 0.034626 0.097359 0.14334 0.124173 0.105011 0.085848 0.066686
```

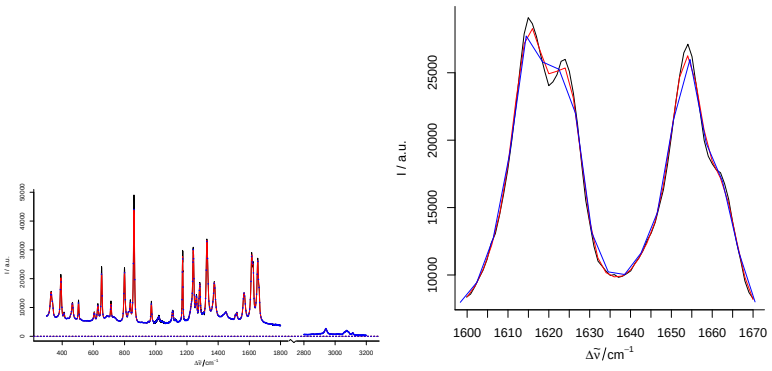



**Figure 7** Smoothing interpolation by `spc.loess` with new data point spacing of 2 $cm^{-1}$ (red) and `spc.bin` (blue). The magnification on the right shows how interpolation may cause a loss in signal height.

```
          642       646       650       654       658       662
[1,] 0.046817 0.050862 0.027717 0.0045712 0.0098066 0.014035
[2,] 0.061956 0.040337 0.021609 0.0028803 0.0080017 0.015405
[3,] 0.047523 0.028361 0.017894 0.0074266 0.0091003 0.017158
```

### 11.3.4. Spikes in Raman Spectra

...coming soon...

## 11.4. Smoothing Interpolation

`spc.bin`
`spc.loess`

Spectra acquired by grating instruments are frequently interpolated onto a new wavelength axis, e. g. because the unequal data point spacing should be removed. 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 do so: `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)
> b <- spc.bin (paracetamol, 4)
> plot (b, wl.range = c (300 ~ 1800, 2800 ~ max), xoffset = 850,
+       lines.args = list (pch = 20, cex = .3, type = "p"), col = "blue", add = TRUE)
```

`spc.loess` applies R's `loess` function for spectral interpolation. Figure 7 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.

### 11.5. Background Correction

sweep

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

### 11.6. Offset Correction

apply sweep

Calculate the offsets and sweep them off the spectra:

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

If the offset is calculated by a function, as here with the `min`, *hyperSpec*'s `sweep` method offers a shortcut: `sweep`'s *STATS* argument may be the function instead of a numeric vector:

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

### 11.7. Baseline Correction

*hyperSpec* comes with two functions to fit polynomial baselines.

spc.fit.poly
spc.fit.poly.below

`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)
> chondro <- chondro - bl
```

For details, see `vignette (baselinebelow)`.

Package *baseline* [1] offers many more functions for baseline correction. The `baseline` function works on the spectra matrix, which is extracted by `[[]]`. The result is a *baseline* object, but can easily be re-imported into the *hyperSpec* object:

```
> corrected <- hyperSpec::chondro [1] # start with the unchanged data set
> require ("baseline")
> bl <- baseline (corrected [[]], method = "modpolyfit", degree = 4)
> corrected [[]] <- getCorrected (bl)
```

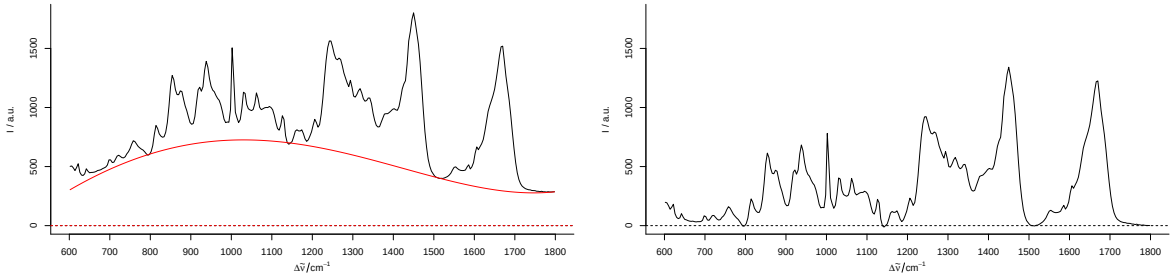
Fig. 8 shows the result for the first spectrum of `chondro`.

```
> rm (bl, chondro)
```

### 11.8. Intensity Calibration

#### 11.8.1. 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`




**(a)** Raw data with baseline

**(b)** Baseline corrected spectrum

**Figure 8** Baseline correction using the *baseline* package: the first spectrum of `chondro` with baseline (left) and after baseline correction (right) with method "modpolyfit".

### 11.8.2. Correcting Wavelength Dependence

`sweep`

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, "/")`

### 11.8.3. Spectra Dependent Correction

`sweep`

If the correction depends on the spectra (e. g. due to inhomogeneous illumination while collecting imaging data, differing optical path length, etc.), the *MARGIN*of the `sweep` function needs to be 1 or `SPC`:

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

## 11.9. Normalization

`apply sweep`

Again, `sweep` is the function of choice. E. g. for area normalization, use:

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

(using the mean instead of the sum results in conveniently scaled spectra with intensities around 1.)

If the calculation of the normalization factors is more elaborate, use a two step procedure:

1. Calculate appropriate normalization factors
   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 [, , 1600 ~ 1700], 1, mean)
> chondro <- sweep (chondro, 1, factors, "*")
```

For the special case of area normalization using the `mean` spectra, the factors can be more conveniently calculated by

```
> factors <- 1 / rowMeans (chondro [, , 1600 ~ 1700])
```

and instead of `sweep` the arithmetic operators (here `*`) can be used directly with the normalization factor:

```
> chondro <- chondro * factors
```

Put together, this results in:

```
> chondro <- chondro / rowMeans (chondro [, , 1600 ~ 1700])
```

For minimum-maximum-normalization, first do an offset- or baseline correction, then normalize using `max`.

## 11.10. Centering and Variance Scaling the Spectra

`scale`

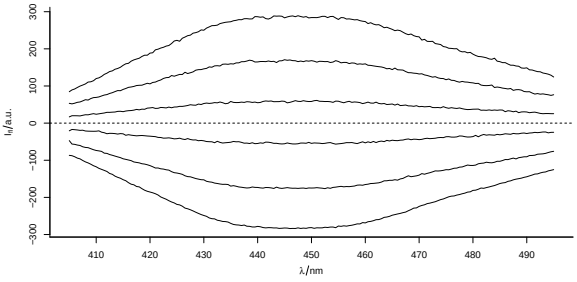
Centering means that the mean spectrum is subtracted from each of the spectra. Many data analysis techniques, like principal component analysis, partial least squares, etc., work much better on centered data. From a spectroscopic point of view it depends on the particular data set whether centering does make sense or not.

Variance scaling is often used in multivariate analysis to adjust the influence and scaling of the variates (that are typically different physical values). However, spectra already do have the same scale of the same physical value. Thus one has to trade off the the expected numeric benefit with the fact that for wavelengths with low signal the noise level will "explode" by variance scaling. Scaling usually makes sense only for centered data.

Both tasks are carried out by the same method in `R`, `scale`, which will by default both mean center and variance scale the spectra matrix.

To center the `flu` data set, use:

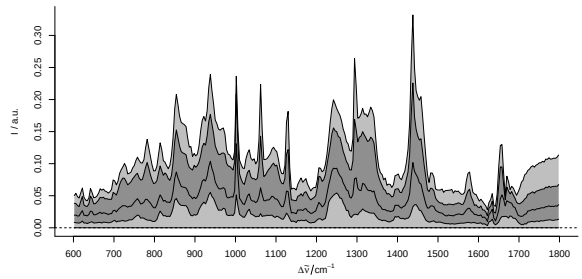
```
> flu.centered <- scale (flu, scale = FALSE)
> plot (flu.centered)
```




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 may be 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 (negative) noise. In order to avoid that, using e. g. the $5^{\text{th}}$ percentile spectrum is more suitable:

```
> chondro <- scale (chondro, center = quantile (chondro, 0.05), scale = FALSE)
> plot (chondro, "spcprctl5")
```




See section 13 (p. 13) for some tips to speed up these calculations.

### 11.11. Multiplicative Scatter Correction (MSC)

pls::msc

MSC can be done using `msc` from package *pls*[2]. It operates on the spectra matrix:

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

### 11.12. 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)
```

Many modeling functions call `as.data.frame` on their *data* argument. In that case, the conversion is done automatically:

```
> 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 9. decomposition

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

```
hyperSpec object
   875 spectra
   5 data columns
   300 data points / spectrum
wavelength: PC [integer] 1 2 ... 300
data:  (875 rows x 5 columns)
   1. y: y [numeric] -4.77 -4.77 ... 19.23
   2. x: x [numeric] -11.55 -10.55 ... 22.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] matrix matrix ... lacuna + NA
   5. spc: score / a.u. [matrix300] -0.43543 -0.92192 ... -2.7756e-17
```

The loadings can be similarly re-imported:

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

```
hyperSpec object
   300 spectra
   2 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (300 rows x 2 columns)
   1. filename: filename [character] 1 1 ... 1
   2. spc: loading I / a.u. [matrix300] -0.0258979 -0.0014762 ... 0.15463
```

There is, however, one important difference. The loadings are thought of as values computed from all spectra together. 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 filename clusters
PC1 NA NA        1     <NA>
```

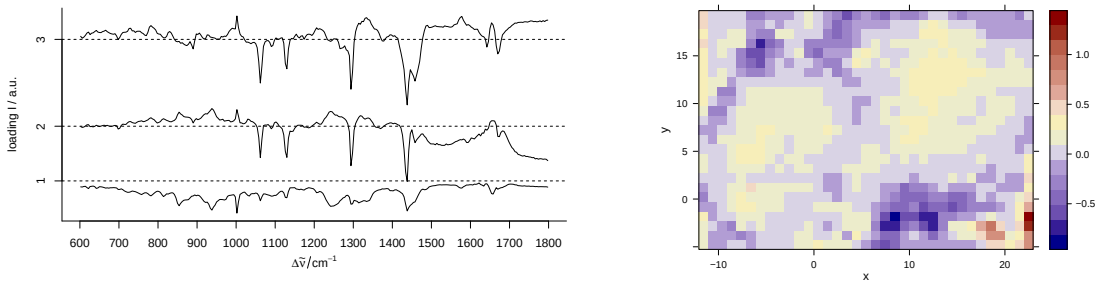
If an extra data column does contain only one unique value, it is retained anyways:

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

```
    filename measurement
PC1        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.




**(a)** **(b)**

**Figure 9** (a) The first three loadings: `plot (loadings [1 : 3], stacked = TRUE)`. (b) The third score map: `plotmap (scores [, , 3])`.

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 $\epsilon$.

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: `%*%`

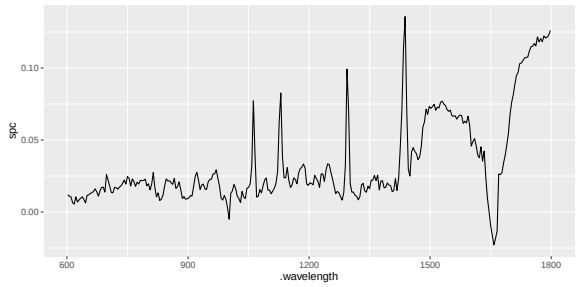
```
> smoothed <- scores [,, 1:10] %*% loadings [1:10]
```

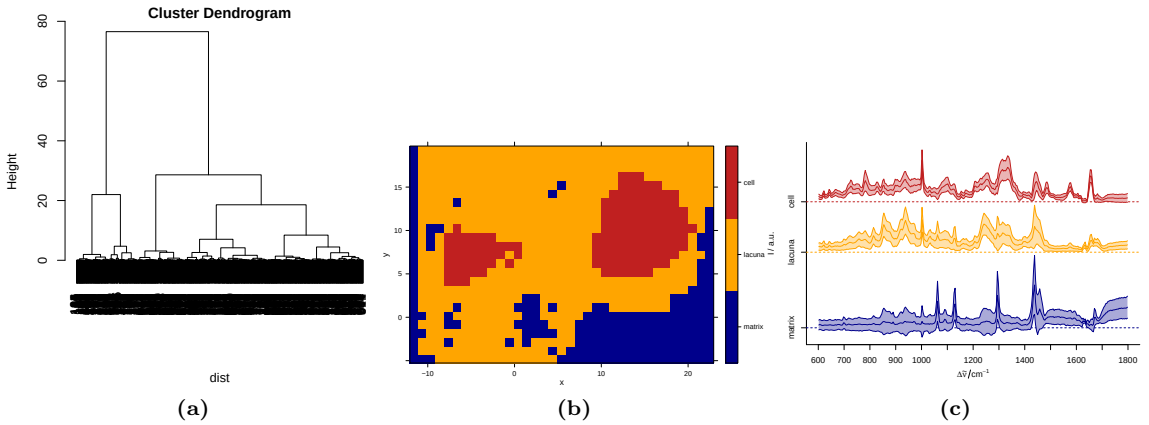
Keep in mind, though, that we cannot be sure how much *useful* information was discarded with the higher components. This kind of noise reduction may influence further modeling of the data. Mathematically speaking, the rank of the new 875 × 300 spectra matrix is only 10.

## 12.2. Data Analysis using long-format data.frame e. g. plotting with ggplot2

Some functions need the data being an *unstacked* or *long-format data.frame*. `as.long.df` is the appropriate conversion function. `as.long.df`

```
> require (ggplot2)
> ggplot (as.long.df (chondro [1]), aes (x = .wavelength, y = spc)) + geom_line ()
```






(a) (b) (c)

**Figure 10** The results of the cluster analysis: (a) the dendrogram (b) the map of the 3 clusters (c) the mean spectra.

## 12.3. Data Analysis Methods using a matrix e. g. Hierarchical Cluster Analysis

`[[]]`

Some functions expect their input data in a matrix, so either `as.matrix (object)` or the abbreviation `object [[]]` can be used:

```
> dist <- pearson.dist (chondro [[]])
```

Again, many such functions coerce the data to a matrix automatically, so the *hyperSpec* object can be handed over:

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

> plot (dendrogram)
```

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

First, cut the dendrogram 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 10b):

## 12.4. 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")
```

## 12.5. Factor columns in hyperSpec Objects: dropping factor levels that are not needed

For subsections of *hyperSpec* objects that do not contain all levels of a factor column, `droplevels` drops the "unpopulated" levels: droplevels

```
> tmp <- chondro [1 : 50]
> table (tmp$clusters)

matrix lacuna   cell
    22     28      0

> tmp <- droplevels (tmp)
> table (tmp$clusters)

matrix lacuna
    22     28
```

## 12.6. Splitting an Object, and Binding a List of hyperSpec Objects

split

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

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

$matrix
hyperSpec object
   187 spectra
   6 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (187 rows x 6 columns)
   1. y: y [numeric] -4.77 -4.77 ... 19.23
   2. x: x [numeric] -11.55 -10.55 ... -11.55
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] matrix matrix ... matrix
   5. spc: I / a.u. [matrix300] 0.011964 0.022204 ... 0.13706
   6. measurement: measurement [numeric] 1 1 ... 1

$lacuna
hyperSpec object
   546 spectra
   6 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (546 rows x 6 columns)
   1. y: y [numeric] -4.77 -4.77 ... 19.23
   2. x: x [numeric] -8.55 -7.55 ... 22.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] lacuna lacuna ... lacuna
   5. spc: I / a.u. [matrix300] 0.038900 0.031386 ... 0.049803
   6. measurement: measurement [numeric] 1 1 ... 1

$cell
hyperSpec object
   142 spectra
   6 data columns
   300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data:  (142 rows x 6 columns)
   1. y: y [numeric] 4.23 4.23 ... 16.23
   2. x: x [numeric] -7.55 -6.55 ... 14.45
   3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
   4. clusters: clusters [factor] cell cell ... cell
```

```
   5. spc: I / a.u. [matrix300] 0.024574 0.027541 ... 0.017377
   6. measurement: measurement [numeric] 1 1 ... 1
```

Splitting can be reversed by `rbind` (see section 9.1, page 19). Another, similar way to combine a number of *hyperSpec* objects with different wavelength axes or extra data columns is `collapse` (see section 9.2, page 19). rbind collapse

Both `rbind` and `collapse` take care that factor levels are expanded as necessary:

```
> lacunae <- droplevels (chondro [chondro$clusters == "lacuna" & ! is.na (chondro$clusters)])
> summary (lacunae$clusters)

lacuna
   546

> cells <- droplevels (chondro [chondro$clusters == "cell" & ! is.na (chondro$clusters)])
> summary (cells$clusters)

cell
 142

> summary (rbind (cells, lacunae)$clusters)

  cell lacuna
   142    546

> summary (collapse (cells, lacunae)$clusters)

  cell lacuna
   142    546
```

## 13. Speed and Memory Considerations

While most of *hyperSpec*'s functions work at a decent speed for interactive sessions (of course depending on the size of the object), iterated (repeated) calculations as for bootstrapping or iterated cross validation may ask for special speed considerations.

As an example, let's again consider the code for shifting the spectra:

```
> tmp <- chondro [1 : 50]
> shifts <- rnorm (nrow (tmp))
> system.time ({
+   for (i in seq_len (nrow (tmp)))
+     tmp [[i]] <- interpolate (tmp [[i]], shifts [i], wl = wl (tmp))
+ })

   user  system elapsed
  0.028   0.000   0.028
```

Calculations that involve a lot of subsetting (i. e. extracting or changing the spectra matrix or extra data) can be sped up considerably if the required parts of the *hyperSpec* object are extracted beforehand. This is somewhat similar to model fitting in R in general: many model fitting functions in R are much faster if the formula interface is avoided and the appropriate *data.frame*s or matrices are handed over directly.

```
> tmp <- chondro [1 : 50]
> system.time ({
+   tmp.matrix <- tmp [[]]
+   wl <- wl (tmp)
+   for (i in seq_len (nrow (tmp)))
+     tmp.matrix [i, ] <- interpolate (tmp.matrix [i, ], shifts [i], wl = wl)
+   tmp [[]] <- tmp.matrix
+ })
```

```
   user  system elapsed
  0.004   0.000   0.005
```

**Additional packages.**

**matrixStats** implements fast functions to calculate summary statistics for each row or each column of a matrix. This functionality can be enabled for *hyperSpec* by installing package *hyperSpec.matrixStats* which is available in *hyperSpec*'s development repository at http://hyperSpec.r-forge.r-project.org/

**Compiled code.** R provides interfaces to Fortran and C code, see the manual "Writing R Extensions". *Rcpp*[3, 4, 5] allows to conveniently integrate C++ code. *inline*[6] adds another layer of convenience: inline definition of functions in C, C++, or Fortran.

An intermediate level is byte compilation of R code, which is done by *compiler*[7].

**Memory use.** In general, it is recommended not to work with variables that are more than approximately a third of the available RAM in size. Particularly the import of raw spectroscopic data can consume large amounts of memory. At certain points, *hyperSpec* provides switches that allow working with data sets that are actually close to this memory limit.

new ("hyperSpec"), read.ENVI*, read.txt.Renishaw

The initialization method `new ("hyperSpec", ...)` takes particular care to avoid unneccessary copies of the spectra matrix. In addition, frequent calls to `gc ()` can be requested by `hy.setOption (gc = TRUE)`. The same behaviour is triggered in `read.ENVI` and its derivatives (`read.ENVI`, `read.ENVI.HySpex`, and `read.ENVI.Nicolet`). The memory consumption of `read.txt.Renishaw` can be lowered by importing the data in chunks (argument *nlines*).

## Index

## References


[1] Kristian Hovde Liland, Trygve Almøy, and Bjørn-Helge Mevik. Optimal choice of baseline correction for multivariate calibration of spectra. *Applied Spectroscopy*, 64:1007–1016, 2010.

[2] 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.

[3] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL http://www.jstatsoft.org/v40/i08/.

[4] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, New York, 2013. doi: 10.1007/978-1-4614-6868-4. ISBN 978-1-4614-6867-7.

[5] Dirk Eddelbuettel and James Joseph Balamuta. Extending extitR with extitC++: A Brief Introduction to extitRcpp. *PeerJ Preprints*, 5:e3188v1, aug 2017. ISSN 2167-9843. doi: 10.7287/peerj.preprints.3188v1. URL https://doi.org/10.7287/peerj.preprints.3188v1.

[6] Oleg Sklyar, Duncan Murdoch, Mike Smith, Dirk Eddelbuettel, Romain Francois, and Karline Soetaert. *inline: Functions to Inline C, C++, Fortran Function Calls from R*, 2018. URL https://CRAN.R-project.org/package=inline. R package version 0.3.15.

[7] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL https://www.R-project.org/.


## A. Overview of the functions provided by hyperSpec

| Function | Explanation |
|---|---|
| *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 |

| Function | Explanation |
|---|---|
| `wl` | extract the wavelengths |
| `wl<-` | replace the wavelengths |
| `wl2i` | convert wavelengths to spectra matrix column indices |
| *Maths* | |
| `%*%` | matrix multiplication |
| *Vectorization* | |
| `aggregate` | |
| `apply` | |
| `sweep` | |
| *Comparison* | |
| `all.equal` | |
| *Plotting* | |
| `alois.palette` | another palette |
| `levelplot` | |
| `map.identify` | identify spectra in map plot |
| `map.sel.poly` | identify spectra in map plot: select polygon |
| `mark.dendrogram` | mark samples in hclust dendrogram |
| `matlab.dark.palette` | darker version of `matlab.palette` |
| `matlab.palette` | palette resembling Matlab's jet colors |
| `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) |
| `sel.poly` | polygon selection in lattice plot |
| `spc.identify` | identify spectra and wavelengths in spectra plot |
| `spc.label.default` | helper for spc.identify |
| `spc.label.wlonly` | helper for spc.identify |
| `spc.point.default` | helper for spc.identify |
| `spc.point.max` | helper for spc.identify |
| `spc.point.min` | helper for spc.identify |
| `spc.point.sqr` | helper for spc.identify |
| `stacked.offsets` | calculate intensity axis offsets for stacked spectral plots |
| `trellis.factor.key` | modify list of `levelplot` arguments according to factor levels |

| Function | Explanation |
|---|---|
| *Type conversion* | |
| `as.data.frame` | |
| `as.long.df` | convert to a long-format data.frame. |
| `as.matrix` | |
| `as.t.df` | convert to a transposed data.frame (spectra in columns) |
| `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 |
| *Combine/split* | |
| `bind` | commom interface for `rbind` and `cbind` |
| `cbind.hyperSpec` | |
| `collapse` | combine objects by adding columns if necessary. See `plyr::rbind.fill`. |
| `merge` | combines spectral ranges. works if spectra are in only one of the data sets |
| `rbind.hyperSpec` | bind objects by row, i. e. add wavelength ranges or extra data |
| `split` | |
| *Basic information* | |
| `chk.hy` | checks whether the object is a hyperSpec object |
| `colnames` | |
| `colnames<-` | |
| `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` | |
| `summary` | summary information including the log |
| *Create and initialize an object* | |
| `empty` | creates an hyperSpec object with 0 rows, but the same wavelengths as another object |
| *Options* | |
| `hy.getOption` | get an option |
| `hy.getOptions` | get more options |
| `hy.setOptions` | set options |
| *Tests* | |
| `hy.unittest` | run all unit tests |
| *Utility functions* | |

| Function | Explanation |
|---|---|
| `mean` | mean spectrum |
| `mean_pm_sd` | mean ± one standard deviation of a vector |
| `mean_sd` | mean and standard deviation of a vector |
| `pearson.dist` | distance measure based on Pearson's $R^2$ |
| `quantile` | quantile spectra |
| `rbind.fill.matrix` | transitional until plyr::rbind.fill.matrix is out |
| `wc` | word count using `wc` if available on the system |
| *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 |
| *File import/export* | |
| `read.ENVI` | import ENVI file |
| `read.ENVI.Nicolet` | import ENVI files writen by Nicolet spectrometers |
| `read.spc` | import .spc file |
| `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 |
| `scan.txt.Witec` | import ASCII files produced by Witec Raman spectrometers |
| `scan.zip.Renishaw` | directly read zip packed ASCII files produced by Renishaw spectrometers |
| `write.txt.long` | export as long-type ASCII file |
| `write.txt.wide` | export as wide-type ASCII file |

## Session Info

```
              [,1]
sysname        "Linux"
release        "4.15.0-76-generic"
version        "#86-Ubuntu SMP Fri Jan 17 17:24:28 UTC 2020"
nodename       "cx17007"
machine        "x86_64"
login          "unknown"
user           "cb"
effective_user "cb"
```

```
R version 3.6.2 (2019-12-12)
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
Running under: Ubuntu 18.04.4 LTS

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

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

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

other attached packages:
[1] baseline_1.2-2          MASS_7.3-51.5           hyperSpec_0.99-20200213 xml2_1.2.2
[5] ggplot2_3.2.1           lattice_0.20-38

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.3          pillar_1.4.2        compiler_3.6.2      RColorBrewer_1.1-2
 [5] R.methodsS3_1.7.1   R.utils_2.9.2       testthat_2.3.1      digest_0.6.23
 [9] lifecycle_0.1.0     tibble_2.1.3        gtable_0.3.0        R.cache_0.14.0
[13] pkgconfig_2.0.3     rlang_0.4.2         mvtnorm_1.0-11      SparseM_1.78
[17] R.rsp_0.43.2        withr_2.1.2         dplyr_0.8.3         tidyselect_0.2.5
[21] glue_1.3.1          R6_2.4.1            plotrix_3.7-7       latticeExtra_0.6-28
[25] farver_2.0.1        purrr_0.3.3         magrittr_1.5        scales_1.1.0
[29] assertthat_0.2.1    colorspace_1.4-1    labeling_0.3        lazyeval_0.2.2
[33] munsell_0.5.0       crayon_1.3.4        R.oo_1.23.0
```