hyperSpec Introduction

Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>
DIA Raman Spectroscopy Group, University of Trieste/Italy (2005 – 2008)
ÖPV, JKI, Berlin/Germany (2017 – 2019)
Arbeitskreis Lebensmittelmikrobiologie und Biotechnologie, Hamburg University, Hamburg/Germany
Chemometric Consulting and Chemometrix GmbH, Wölfersheim/Germany (since 2016)

March 30, 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

1. Introduction .................................................. 3
   1.1. Notation and Terms ........................................ 4

2. Remarks on R ................................................ 4
   2.1. Reporting an Issue with a package ...................... 4
   2.2. Generic Functions ........................................ 5
   2.3. Functionality Can be Extended at Runtime ............ 5
   2.4. Validity Checking ........................................ 5
   2.5. Special Function Names .................................. 5
       2.5.1. The Names of Operators ............................. 5
       2.5.2. Assignment Functions ............................... 6

3. Loading and the package and configuration .................. 6

4. The structure of hyperSpec objects ......................... 6

5. Functions provided by hyperSpec ........................... 7
6. Obtaining Basic Information about hyperSpec Objects

7. Creating a hyperSpec Object, Data Import and Export
   7.1. Creating a hyperSpec Object from Spectra Matrix and Wavelength Vector  
   7.2. Creating Random Spectra

8. Access to the data
   8.1. Access Functions and Abbreviations for Parts of the hyperSpec Object’s Data  
   8.2. Selecting and Deleting Spectra
      8.2.1. Random Samples  
      8.2.2. Sequences
   8.3. Selecting Extra Data Columns
   8.5. Wavelengths
      8.5.1. Converting Wavelengths to Indices and vice versa  
      8.5.2. Selecting Wavelength Ranges
      8.5.3. Deleting Wavelength Ranges
      8.5.4. Changing the Wavelength Axis
      8.5.5. Ordering the Wavelength Axis
   8.6. Conversion to Long- and Wide-Format data.frames
   8.7. Conversion to Matrix

9. Combining and Decomposing hyperSpec Objects
   9.1. Binding Objects together
   9.2. Binding Objects that do not Share the Same Extra Data and/or Wavelength Axis
   9.3. Binding Objects that do not Share the Same Spectra
   9.4. Merging extra data to objects that do not (necessarily) share the same spectra
   9.5. Matrix Multiplication
   9.6. Decomposition

10. Plotting

11. Spectral (Pre)processing
   11.1. Cutting the Spectral Range
   11.2. Shifting Spectra
      11.2.1. Calculating the Shift
   11.3. Removing Bad Data
      11.3.1. Bad Spectra  
      11.3.2. Removing Spectra outside mean ± n sd
      11.3.3. Bad Data Points
      11.3.4. Spikes in Raman Spectra
   11.4. Smoothing Interpolation
   11.5. Background Correction
   11.6. Offset Correction
   11.7. Baseline Correction
   11.8. Intensity Calibration
      11.8.1. Correcting by a constant, e.g. Readout Bias
      11.8.2. Correcting Wavelength Dependence
      11.8.3. Spectra Dependent Correction
   11.9. Normalization
   11.10. Centering and Variance Scaling the Spectra
11.11 Multiplicative Scatter Correction (MSC) ............................................. 33
11.12 Spectral Arithmetic ........................................................................... 33

12. Data Analysis ....................................................................................... 33
12.1. Data Analysis Methods using a data.frame
   e. g. Principal Component Analysis with prcomp .................................. 33
12.1.1. PCA as Noise Filter ...................................................................... 34
12.2. Data Analysis using long-format data.frame
   e. g. plotting with ggplot2 ..................................................................... 35
12.3. Data Analysis Methods using a matrix
   e. g. Hierarchical Cluster Analysis ........................................................ 36
12.4. Calculating group-wise Sum Characteristics, e. g. Cluster Mean Spectra .............................................................. 36
12.5. Factor columns in hyperSpec Objects: dropping factor levels that are not needed .......... 37
12.6. Splitting an Object, and Binding a List of hyperSpec Objects .................. 37

13. Speed and Memory Considerations ......................................................... 38

A. Overview of the functions provided by hyperSpec .................................... 41

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 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, \( \frac{m}{\lambda} \), etc.
- **intensity**: spectral ordinate
  - transmission, absorbance, \( \frac{1}{s} \), 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:

```r
> 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.

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

will take you to `hyperSpec`’s issue tracking page at [https://github.com/cbeleites/hyperSpec/issues](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 traveling 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.

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:

```r
> 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.:

```r
> sweep (flu, 2, mean, "-\-")
```

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

```r
> `^+` (3, 5)
[1] 8
```
### 2.5.2. Assignment Functions

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

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

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

### 3. Loading and the package and configuration

To load `hyperSpec`, use

```r
> 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).
Table 2  Get and set functions for the slots of `hyperSpec` objects

<table>
<thead>
<tr>
<th>slot</th>
<th>get</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>@wavelength</td>
<td><code>wl</code></td>
<td><code>wl&lt;-</code></td>
</tr>
<tr>
<td>@data</td>
<td><code>.$</code>, <code>as.data.frame</code>, <code>as.long.df</code>, ...</td>
<td><code>[&lt;-</code>, <code>[``, </code>&lt;-`</td>
</tr>
<tr>
<td>@label</td>
<td><code>labels</code></td>
<td><code>labels&lt;-</code></td>
</tr>
</tbody>
</table>

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:

```r
> chondro

`hyperSpec` object
875 spectra
```
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

```r
> nrow (chondro)
[1] 875
> nwl (chondro)
[1] 300
> ncol (chondro)
[1] 5
> dim (chondro)
875 5 300
```

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

```r
> 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.

### 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")`. 


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:

\[
> \text{spc} \leftarrow \text{new} \left( \text{"hyperSpec"}, \text{spc} = \text{spectra.matrix}, \text{wavelength} = \text{wavelength.vector}, \text{data} = \text{extra.data} \right)
\]

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 rmvnorm (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:

\[
> \text{pcov} \leftarrow \text{pooled.cov} (\text{chondro, chondro$clusters})
\]
\[
> \text{rnd} \leftarrow \text{rmvnorm} (\text{rep} (10, 3), \text{mean} = \text{pcov$mean}, \text{sigma} = \text{pcov$COV})
\]
\[
> \text{cluster.cols} \leftarrow \text{c} (\text{"dark blue", "orange", "#C02020")}
\]
\[
> \text{plot} (\text{rnd, col = cluster.cols [rnd$group]})
\]

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

\[
> \text{require ("MASS")}
\]
\[
> \text{rnd} \leftarrow \text{rmvnorm} (\text{rep} (200, 3), \text{mean} = \text{pcov$mean}, \text{sigma} = \text{pcov$COV})
\]
\[
> \text{lda} \leftarrow \text{lda} (\text{clusters ~ spc, rnd})
\]
\[
> \text{pred.chondro} \leftarrow \text{predict} (\text{lda, chondro})
\]
\[
> \text{pred.sim} \leftarrow \text{predict} (\text{lda})
\]
> 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)
8.2.1. Random Samples

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

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

hyperSpec object
- 3 spectra
- 5 data columns
- 300 data points / spectrum

```
data: (3 rows x 5 columns)
  1. y: y [numeric] 11.23 18.23 -1.77
  2. x: x [numeric] -7.55 18.45 -0.55
  3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt rawdata/chondro.txt
  4. clusters: clusters [factor] matrix lacuna lacuna
  5. spc: I / a.u. [matrix300] 355.55 390.34 ... 134.79
```

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

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

```
[1] 453 134 336
```

8.2.2. Sequences

Sequences of every \( n \)th spectrum or the like can be retrieved with `seq`:

```r
> 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.

```r
> 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
```

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

```r
> 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
```

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.

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 \( \text{start} \sim \text{end} \). This yields a vector index of start : index of end.

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

- The colon operator for constructing vectors accepts only integer numbers, the tilde (for formulas) does not have this restriction.

- If the vector does not take into account the spectral resolution, one may get only every \( n \)th point or repetitions of the same index:

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

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

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

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

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

```r
> 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:

```r
> 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`.  

---
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`.

```r
> 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:

```r
> 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`,
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:

```r
> 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 ...

> 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 ...

> rm (laser)
```

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

```r
> 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:

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

> barb (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 44.95 44.05 55.00 55.10 56.00 56.10 57.10 58.90 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 44.95 44.05 55.00 55.10 56.00 56.10 57.10 58.90 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
```

---

16
8.6. Conversion to Long- and Wide-Format data.frames

`as.data.frame()` extracts the @data slot as a `data.frame`:

```r
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
```

```r
colnames(as.data.frame(flu))
[1] "spc" "filename" "c" "n" ".row"
```

```r
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`:

```r
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
```

```r
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 columns needs to be extracted, use `[[ ]]`:

```r
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 as.t.df the discussion of ggplot2 in vignette ("plotting").

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

<table>
<thead>
<tr>
<th>.wavelength</th>
<th>mean.minus.sd</th>
<th>mean</th>
<th>mean.plus.sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>spc.405</td>
<td>405.0</td>
<td>49.958</td>
<td>113.91</td>
</tr>
<tr>
<td>spc.405.5</td>
<td>405.5</td>
<td>53.396</td>
<td>120.08</td>
</tr>
<tr>
<td>spc.406</td>
<td>406.0</td>
<td>55.352</td>
<td>123.89</td>
</tr>
<tr>
<td>spc.406.5</td>
<td>406.5</td>
<td>57.310</td>
<td>128.64</td>
</tr>
<tr>
<td>spc.407</td>
<td>407.0</td>
<td>59.513</td>
<td>132.79</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>.wavelength</th>
<th>spc</th>
<th>filename</th>
<th>c</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>405.0</td>
<td>27.150</td>
<td>rawdata/flu1.txt</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>405.0</td>
<td>66.801</td>
<td>rawdata/flu2.txt</td>
<td>0.10</td>
<td>2</td>
</tr>
<tr>
<td>405.0</td>
<td>93.144</td>
<td>rawdata/flu3.txt</td>
<td>0.15</td>
<td>3</td>
</tr>
<tr>
<td>405.0</td>
<td>130.664</td>
<td>rawdata/flu4.txt</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>405.0</td>
<td>167.267</td>
<td>rawdata/flu5.txt</td>
<td>0.25</td>
<td>5</td>
</tr>
<tr>
<td>405.0</td>
<td>198.430</td>
<td>rawdata/flu6.txt</td>
<td>0.30</td>
<td>6</td>
</tr>
<tr>
<td>405.5</td>
<td>32.345</td>
<td>rawdata/flu1.txt</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>405.5</td>
<td>63.715</td>
<td>rawdata/flu2.txt</td>
<td>0.10</td>
<td>2</td>
</tr>
<tr>
<td>405.5</td>
<td>103.068</td>
<td>rawdata/flu3.txt</td>
<td>0.15</td>
<td>3</td>
</tr>
<tr>
<td>405.5</td>
<td>139.998</td>
<td>rawdata/flu4.txt</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>405.5</td>
<td>171.898</td>
<td>rawdata/flu5.txt</td>
<td>0.25</td>
<td>5</td>
</tr>
<tr>
<td>405.5</td>
<td>209.458</td>
<td>rawdata/flu6.txt</td>
<td>0.30</td>
<td>6</td>
</tr>
<tr>
<td>406.0</td>
<td>33.379</td>
<td>rawdata/flu1.txt</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>406.0</td>
<td>66.712</td>
<td>rawdata/flu2.txt</td>
<td>0.10</td>
<td>2</td>
</tr>
<tr>
<td>406.0</td>
<td>106.194</td>
<td>rawdata/flu3.txt</td>
<td>0.15</td>
<td>3</td>
</tr>
<tr>
<td>406.0</td>
<td>143.798</td>
<td>rawdata/flu4.txt</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>406.0</td>
<td>177.471</td>
<td>rawdata/flu5.txt</td>
<td>0.25</td>
<td>5</td>
</tr>
<tr>
<td>406.0</td>
<td>215.785</td>
<td>rawdata/flu6.txt</td>
<td>0.30</td>
<td>6</td>
</tr>
<tr>
<td>406.5</td>
<td>34.419</td>
<td>rawdata/flu1.txt</td>
<td>0.05</td>
<td>1</td>
</tr>
<tr>
<td>406.5</td>
<td>69.582</td>
<td>rawdata/flu2.txt</td>
<td>0.10</td>
<td>2</td>
</tr>
</tbody>
</table>

8.7. Conversion to Matrix

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

```r
> 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
```

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

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

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

```r
> flu [[1:3, 406 ~ 407]]
```
If indices for the columns to extract are given, a `data.frame` is returned instead of a matrix:

```r
> 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
```

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:

```r
> dim (flu)
nrow ncol nwl
   6   3 181
```

```r
> dim (cbind (flu, flu))
nrow ncol nwl
   6   3 362
```

```r
> 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.

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`.

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.

```r
> barb <- collapse (barbiturates)
> wl (barb) [1 : 25]
[1] 25.95 26.05 26.15 26.95 27.05 27.15 28.05 28.15 28.95 29.05 29.15 29.95 30.05 30.15 30.25 31.05 31.15
```

```r
[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:

```r
> 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
[2,] NA NA NA NA 618 10151 NA 5040 NA
[3,] NA NA NA NA 638 NA NA 10722 5253 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`.

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.

```r
> 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:

```r
> chondro.merged <- merge(chondro.low, chondro.high)
> nrow(chondro.merged)
[1] 560
```

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):

```r
> chondro.merged <- merge(chondro.low, chondro.high, all = TRUE)
> nrow(chondro.merged)
[1] 840
```

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

```r
   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

20
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

```r
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:

```r
> 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 |
|----|----|-------------------------|----------------|----------------|----------------|----------------|----------------|
| -4.77 | -11.55 | rawdata/chondro.txt matrix 1 NA 488.63 466.18 492.00 NA NA |
| -4.77 | -10.55 | rawdata/chondro.txt matrix 2 NA 489.48 465.05 490.53 NA NA |
| -4.77 | -9.55  | rawdata/chondro.txt matrix 3 NA 456.03 436.62 458.06 NA NA |
| -4.77 | -8.55  | rawdata/chondro.txt matrix 4 NA 464.82 444.85 470.02 NA NA |
| -4.77 | -7.55  | rawdata/chondro.txt matrix 5 1 428.66 410.80 433.12 461.19 397.38 |
| -4.77 | -6.55  | rawdata/chondro.txt matrix 6 2 426.07 407.86 431.21 458.15 394.18 |
| -4.77 | -5.55  | rawdata/chondro.txt lacuna 7 3 412.37 396.50 421.27 445.54 382.72 |
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`:

```r
> flu.ref <- data.frame(filename = rep(flu$filename[1:2], each = 2), cref = rep(flu$c [1:2], each = 2) + rnorm(2))
> flu.ref
   filename cref
1  rawdata/flu1.txt 0.041930
2  rawdata/flu1.txt 0.057315
3  rawdata/flu2.txt 0.098126
4  rawdata/flu2.txt 0.100434
```

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

```r
> flu.merged <- merge(flu, flu.ref)
> flu.merged$..
 filename   c  cref
1  rawdata/flu1.txt 0.05 0.041930
2  rawdata/flu1.txt 0.05 0.057315
3  rawdata/flu2.txt 0.10 0.098126
4  rawdata/flu2.txt 0.10 0.100434
```

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

```r
> flu.merged <- merge(flu, flu.ref, all.x = TRUE)
> flu.merged$..
 filename   c  cref
1  rawdata/flu1.txt 0.05 0.041930
2  rawdata/flu1.txt 0.05 0.057315
3  rawdata/flu2.txt 0.10 0.098126
4  rawdata/flu2.txt 0.10 0.100434
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:

```r
> 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] 0.041930 0.057315 0.098126 0.100434
```

```r
> 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 0.041930 27.150 32.345 33.379 34.419 36.531 37.648 38.137 39.177
2  rawdata/flu1.txt 0.057315 27.150 32.345 33.379 34.419 36.531 37.648 38.137 39.177
3  rawdata/flu2.txt 0.098126 66.801 63.715 66.712 69.582 72.530 74.558 77.048 80.260
```
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.

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.

```r
> 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 ...
3. c: c / (mg / l) [numeric] 0.05 0.10 ...

> 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

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):

```r
> 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:

```r
> 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):
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 \(\text{wl\leftarrow}\) (red) does not affect the intensities, while the spectrum is slightly changed by interpolations (blue).

\[
\text{tmp <- apply (chondro, 1, interpolate, shift = -10, wl = wl (chondro))}
\]

If different spectra need to be offset by different shift, use a loop\(^2\)

\[
\text{shifts <- rnorm (nrow (chondro))}
\]
\[
\text{tmp <- chondro [1]}\]
\[
\text{for (i in seq_len (nrow (chondro)))}
\]
\[
\text{+ tmp [i, ] <- interpolate (tmp [i, ], shifts [i], wl = wl (chondro))}
\]
\[
\text{chondro [1] <- tmp}
\]

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 \text{chondro} data, let’s use the maximum of the phenylalanine band between 990 and 1020 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.

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

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.

\[
\text{chondro <- chondro - spc.fit.poly.below (chondro [,min+3i ~ max - 3i], chondro)}
\]
\[
\text{chondro <- sweep (chondro, 1, rowMeans (chondro [,], na.rm = TRUE), "}")}
\]

\(^2\text{sweep cannot be used here, and while there is the possibility to use sapply or mapply, they are not faster than the for loop in this case. Make sure to work on a copy of the spectra matrix, as that is much faster than row-wise extracting and changing the spectra by [ and [\leftarrow.\)
Figure 5  The shifts used to disturb the chondrocyte data (original), and the remaining shift after correction with the two methods discussed here.

```r
> 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:

```r
> 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 ± n sd

```r
> 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)
```
11.3.3. Bad Data Points

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

```r
> 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,] 0.109779 0.09619 0.043026 0.048578 0.147560 0.074874 -0.039309 -0.050153 -0.065479 0.039308
[2,] 0.000000 0.10474 0.079726 0.025398 0.055468 0.149382 0.0277173 -0.052785 -0.062668 -0.043988
[3,] 0.087767 0.10945 0.072054 0.029876 0.000000 0.150299 0.0079205 -0.053727 -0.061925 -0.030075
```

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

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

```
602 606 610 614 618 622 626 630 634 638 642
[1,] 0.109779 0.09619 0.043026 0.048578 0.147560 0.074874 NA NA NA 0.039308 NA
[2,] NA 0.10474 0.079726 0.025398 0.055468 0.149382 NA NA 0.052828 NA
[3,] 0.087767 0.10945 0.072054 0.029876 NA 0.150299 NA NA 0.065073 NA
```

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

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

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

```
602 606 610 614 618 622 626 630 634 638 642
[1,] 0.109779 0.09619 0.043026 0.048578 0.147560 0.074874 0.0659828 0.057091 0.048200 0.039308
[2,] 0.104743 0.10474 0.079726 0.025398 0.055468 0.149382 0.0277173 0.033995 0.040273 0.046550
[3,] 0.087767 0.10945 0.072054 0.029876 0.000000 0.150299 0.0079205 0.033995 0.040273 0.046550
```

Figure 6 filtering data
11.4. Smoothing Interpolation

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.

```r
> 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 = 1, 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.

---

**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.

[Hyperlink to Figure 7]
11.5. Background Correction

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

11.6. Offset Correction

Calculate the offsets and sweep them off the spectra:

```r
> 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:

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

11.7. Baseline Correction

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

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

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

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

```r
> 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:

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

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

```r
> 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:

```r
spectra - constant
```
11.8.2. Correcting Wavelength Dependence

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 sub-
   tracted:
   
   ```
   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

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

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

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 5th percentile spectrum is more suitable:
> chondro <- scale (chondro, center = quantile (chondro, 0.05), scale = FALSE)
> plot (chondro, "spcprct15")

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

### 11.11. Multiplicative Scatter Correction (MSC)

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

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

### 11.12. Spectral Arithmetic

Basic mathematical functions are defined for `hyperSpec` objects. You may convert spectra:

```r
absorbance.spectra = - log10 (transmission.spectra)
```

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

```r
> 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:

```r
> 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:

```r
> 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.

```r
> 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 ... -1.249e-16
```

The loadings can be similarly re-imported:

```r
> 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 ... -0.17923
```

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 NAs:

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

   y x filename clusters
PC1 NA NA 1 <NA>
```

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

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

   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.
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:

\[ \text{spectra}^{(nrow x nwl)} = \text{scores}^{(nrow x p)} \text{loadings}^{(p x nwl)} \]

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

\[ A_n(\lambda) = c_n \epsilon(i, \lambda) + \text{error} \]

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:

```r
> 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 x 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.

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

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

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

```r
> 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:

```r
> 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.

```r
> 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:

```r
> 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:

```r
> 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

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

```r
> 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).

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

```r
> 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:

```r
> 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
0.032 0.000 0.032
```

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.frames or matrices are handed over directly.

```r
> 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
+ })
```

38
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\text{'s development repository at \url{http://hyperSpec.r-forge.r-project.org/}}.

Compiled code. R provides interfaces to Fortran and C code, see the manual “Writing R Extensions”. Rcpp\cite{Rcpp, Rcpp:1999, Rcpp:2001} allows to conveniently integrate C++ code. inline\cite{inline} 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\cite{compiler}.

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.

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

see assignment functions, 6
@ operator, 4
$ operator, 4

assignment functions, 6

cchk.hy, 5

data sets
   barbiturates, 3
   chondro, 3
   flu, 3
   laser, 3
   paracetamol, 3

extra data, 4

Generic Functions, 5

hyperspectral data sets, 3

intensity, 4

loading, 6

operators, 5
options, 6
   debuglevel, 6
   gc, 6, 39

validity checking, 5
validObject, 5

wavelength, 4
   conversion, 15
   conversion to index, 13
   formula notation, 13

wavelength indices
   conversion to wavelength, 13
References


A. Overview of the functions provided by hyperSpec

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[]</code></td>
<td>Select / extract / delete spectra, wavelength ranges or extra data</td>
</tr>
<tr>
<td><code>&lt;-</code></td>
<td>Set parts of spectra or extra data</td>
</tr>
<tr>
<td><code>[[</code></td>
<td>Select / extract / delete spectra, wavelength ranges or extra data, get the result as matrix or data.frame</td>
</tr>
<tr>
<td><code>[[&lt;-</code></td>
<td>Set parts of spectra matrix</td>
</tr>
<tr>
<td><code>$</code></td>
<td>extract a data column (including <code>$spc</code>)</td>
</tr>
<tr>
<td><code>$&lt;-</code></td>
<td>replace a data column (including <code>$spc</code>)</td>
</tr>
<tr>
<td><code>i2wl</code></td>
<td>convert spectra matrix column indices to wavelengths</td>
</tr>
<tr>
<td><code>isample</code></td>
<td>get a random sample of the spectra as index vector</td>
</tr>
<tr>
<td><code>labels</code></td>
<td>get column labels</td>
</tr>
<tr>
<td><code>labels&lt;-</code></td>
<td>set column labels</td>
</tr>
<tr>
<td><code>logbook</code></td>
<td>logging the data treatment</td>
</tr>
<tr>
<td><code>logentry</code></td>
<td>make a logbook entry</td>
</tr>
<tr>
<td><code>rownames&lt;-</code></td>
<td></td>
</tr>
<tr>
<td><code>sample</code></td>
<td>generate random sample of the spectra</td>
</tr>
<tr>
<td><code>seq.hyperSpec</code></td>
<td>sequence along the spectra, either as <code>hyperSpec</code> object or index vector</td>
</tr>
<tr>
<td>Function</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wl</td>
<td>extract the wavelengths</td>
</tr>
<tr>
<td>wl&lt;-</td>
<td>replace the wavelengths</td>
</tr>
<tr>
<td>wl2i</td>
<td>convert wavelengths to spectra matrix column indices</td>
</tr>
<tr>
<td>Maths</td>
<td>matrix multiplication</td>
</tr>
<tr>
<td>Vectorization</td>
<td></td>
</tr>
<tr>
<td>aggregate</td>
<td></td>
</tr>
<tr>
<td>apply</td>
<td></td>
</tr>
<tr>
<td>sweep</td>
<td></td>
</tr>
<tr>
<td>Comparison</td>
<td></td>
</tr>
<tr>
<td>all.equal</td>
<td></td>
</tr>
<tr>
<td>Plotting</td>
<td></td>
</tr>
<tr>
<td>alois.palette</td>
<td>another palette</td>
</tr>
<tr>
<td>levelplot</td>
<td></td>
</tr>
<tr>
<td>map.identify</td>
<td>identify spectra in map plot</td>
</tr>
<tr>
<td>map.sel.poly</td>
<td>identify spectra in map plot: select polygon</td>
</tr>
<tr>
<td>mark.dendrogram</td>
<td>mark samples in hclust dendrogram</td>
</tr>
<tr>
<td>matlab.dark.palette</td>
<td>darker version of matlab.palette</td>
</tr>
<tr>
<td>matlab.palette</td>
<td>palette resembling Matlab's jet colors</td>
</tr>
<tr>
<td>plot</td>
<td>main switchyard for plotting</td>
</tr>
<tr>
<td>plotc</td>
<td>intensity over one other dimension: calibration plots, time series, depth</td>
</tr>
<tr>
<td>plotmap</td>
<td>false-colour intensity over two other dimensions: spectral images, maps, etc.</td>
</tr>
<tr>
<td>plotspc</td>
<td>spectra plots: intensity over wavelength</td>
</tr>
<tr>
<td>plotvoronoi</td>
<td>false-colour intensity over two other dimensions: spectral images, maps, etc.</td>
</tr>
<tr>
<td>sel.poly</td>
<td>polygon selection in lattice plot</td>
</tr>
<tr>
<td>spc.identify</td>
<td>identify spectra and wavelengths in spectra plot</td>
</tr>
<tr>
<td>spc.label.default</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>spc.label.wlonly</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>spc.point.default</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>spc.point.max</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>spc.point.min</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>spc.point.sqr</td>
<td>helper for spc.identify</td>
</tr>
<tr>
<td>stacked.offsets</td>
<td>calculate intensity axis offsets for stacked spectral plots</td>
</tr>
<tr>
<td>trellis.factor.key</td>
<td>modify list of levelplot arguments according to factor levels</td>
</tr>
<tr>
<td>Function</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Type conversion</strong></td>
<td></td>
</tr>
<tr>
<td>as.data.frame</td>
<td>convert to a long-format data.frame.</td>
</tr>
<tr>
<td>as.long.df</td>
<td>convert to a transposed data.frame (spectra in columns)</td>
</tr>
<tr>
<td>as.matrix</td>
<td>convert to a wide-format data.frame with each wavelength one column</td>
</tr>
<tr>
<td>decomposition</td>
<td>re-import results of spectral matrix decomposition (or the like) into hyperSpec object</td>
</tr>
<tr>
<td><strong>Combine/split</strong></td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td>common interface for rbind and cbind</td>
</tr>
<tr>
<td>cbind.hyperSpec</td>
<td></td>
</tr>
<tr>
<td>collapse</td>
<td>combine objects by adding columns if necessary. See plyr::rbind.fill.</td>
</tr>
<tr>
<td>merge</td>
<td>combines spectral ranges. works if spectra are in only one of the data sets</td>
</tr>
<tr>
<td>rbind.hyperSpec</td>
<td>bind objects by row, i.e. add wavelength ranges or extra data</td>
</tr>
<tr>
<td>split</td>
<td></td>
</tr>
<tr>
<td><strong>Basic information</strong></td>
<td></td>
</tr>
<tr>
<td>chk.hy</td>
<td>checks whether the object is a hyperSpec object</td>
</tr>
<tr>
<td>colnames</td>
<td>number of data columns (extra data plus spectra matrix)</td>
</tr>
<tr>
<td>colnames&lt;-</td>
<td>number of spectra</td>
</tr>
<tr>
<td>ncol</td>
<td>number of data points per spectrum</td>
</tr>
<tr>
<td>nrow</td>
<td>summary information</td>
</tr>
<tr>
<td>nwl</td>
<td>summary information including the log</td>
</tr>
<tr>
<td>print</td>
<td></td>
</tr>
<tr>
<td>ronames</td>
<td></td>
</tr>
<tr>
<td>summary</td>
<td></td>
</tr>
<tr>
<td><strong>Create and initialize an object</strong></td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>creates an hyperSpec object with 0 rows, but the same wavelengths as another object</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td></td>
</tr>
<tr>
<td>hy.getOption</td>
<td>get an option</td>
</tr>
<tr>
<td>hy.getOptions</td>
<td>get more options</td>
</tr>
<tr>
<td>hy.setOptions</td>
<td>set options</td>
</tr>
<tr>
<td><strong>Tests</strong></td>
<td></td>
</tr>
<tr>
<td>hy.unittest</td>
<td>run all unit tests</td>
</tr>
<tr>
<td><strong>Utility functions</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mean</td>
<td>mean spectrum</td>
</tr>
<tr>
<td>mean_pm_sd</td>
<td>mean ± one standard deviation of a vector</td>
</tr>
<tr>
<td>mean_sd</td>
<td>mean and standard deviation of a vector</td>
</tr>
<tr>
<td>pearson.dist</td>
<td>distance measure based on Pearson’s $R^2$</td>
</tr>
<tr>
<td>quantile</td>
<td>quantile spectra</td>
</tr>
<tr>
<td>rbind.fill.matrix</td>
<td>transitional until plyr::rbind.fill.matrix is out</td>
</tr>
<tr>
<td>wc</td>
<td>word count using wc if available on the system</td>
</tr>
</tbody>
</table>

*Spectra-specific transformations*

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderwl</td>
<td>sort columns of spectra matrix according to the wavelengths</td>
</tr>
<tr>
<td>spc.bin</td>
<td>spectral binning</td>
</tr>
<tr>
<td>spc.fit.poly</td>
<td>least squares fit of a polynomial</td>
</tr>
<tr>
<td>spc.fit.poly.below</td>
<td>least squares fit of a polynomial with automatic support point determination</td>
</tr>
<tr>
<td>spc.loess</td>
<td>loess smoothing interpolation</td>
</tr>
</tbody>
</table>

*File import/export*

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>read.ENVI</td>
<td>import ENVI file</td>
</tr>
<tr>
<td>read.ENVI.Nicolet</td>
<td>import ENVI files written by Nicolet spectrometers</td>
</tr>
<tr>
<td>read.spc</td>
<td>import .spc file</td>
</tr>
<tr>
<td>read.spc.KaiserMap</td>
<td>import a Raman map saved by Kaiser Optical Systems’ Hologram software as multiple .spc files</td>
</tr>
<tr>
<td>read.txt.long</td>
<td>import long-type ASCII file</td>
</tr>
<tr>
<td>read.txt.wide</td>
<td>import wide-type ASCII file</td>
</tr>
<tr>
<td>scan.txt.Renishaw</td>
<td>import ASCII files produced by Renishaw (InVia) spectrometers</td>
</tr>
<tr>
<td>scan.txt.Witec</td>
<td>import ASCII files produced by Witec Raman spectrometers</td>
</tr>
<tr>
<td>scan.zip.Renishaw</td>
<td>directly read zip packed ASCII files produced by Renishaw spectrometers</td>
</tr>
<tr>
<td>write.txt.long</td>
<td>export as long-type ASCII file</td>
</tr>
<tr>
<td>write.txt.wide</td>
<td>export as wide-type ASCII file</td>
</tr>
</tbody>
</table>

**Session Info**

```r
[[1]]
sysname  "Linux"
release   "5.4.0-4-amd64"
version   "#1 SMP Debian 5.4.19-1 (2020-02-13)"
nodename  "aragorn"
machine   "x86_64"
login     "hornik"
user      "hornik"
effective_user "hornik"
```

R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Debian GNU/Linux bullseye/sid

Matrix products: default
BLAS: /home/Hornik/tmp/R-r/lib/libRblas.so
LAPACK: /home/Hornik/tmp/R-r/lib/libRlapack.so

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

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

other attached packages:
[1] baseline_1.2-3 MASS_7.3-51.5 hyperSpec_0.99-20200213.1
[4] xml2_1.2.5 ggplot2_3.3.0 lattice_0.20-38

loaded via a namespace (and not attached):
[1] Rcpp_1.0.4 pillar_1.4.3 compiler_3.6.3 RColorBrewer_1.1-2
[5] R.methodsS3_1.8.0 R.utils_2.3.2 testthat_2.3.2 digest_0.6.25
[9] lifecycle_0.2.0 tidyselect_1.0.0 tibble_2.1.3 tibble_3.3.0
[13] pkgconfig_2.0.3 png_0.1-7 rlang_0.4.5 SparseM_1.78
[17] mvtnorm_1.1-0 R.rsp_0.43.2 withr_2.1.2 dplyr_0.8.5
[21] tidyselect_1.0.0 limSolve_1.5.6 glue_1.3.2 R6_2.4.1
[25] jpeg_0.1-8.1 plotrix_3.7-7 latticeExtra_0.6-29 farver_2.0.3
[29] purrr_0.3.3 magrittr_1.5 scales_1.1.0 assertthat_0.2.1
[33] lpSolve_5.6.15 colorspace_1.4-1 labeling_0.3 quadprog_1.5-8
[37] lazyeval_0.2.2 munsell_0.5.0 crayon_1.3.4 R.oo_1.23.0