

# Package ‘hyperSpec’

January 2, 2012

**Maintainer** Claudia Beleites <cbeleites@units.it>

**License** GPL (>= 3)

**Title** Interface for hyperspectral data, i.e. spectra + meta info (spatial, time, concentration, ...)

**LazyData** yes

**Type** Package

**LazyLoad** yes

**Author** Claudia Beleites

**Description** This package is an interface to handle hyperspectral data sets in R. I.e. spatially or time-resolved spectra, or spectra with any other kind of information associated with each of the spectra. The spectra can be data as obtained in XRF, UV/VIS, Fluorescence, AES, NIR, IR, Raman, NMR, MS, etc. More generally, any data that is recorded over a discretized variable, e.g. absorbance =  $f(\text{wavelength})$ , stored as a vector of absorbance values for discrete wavelengths is suitable.

**Version** 0.98-20110927

**URL** <http://hyperSpec.r-forge.r-project.org/>

**Date** 2011-09-27

**Depends** R (>= 2.11), methods, utils, lattice, grid

**Suggests** R.matlab, latticeExtra, tripack, deldir, rgl, plotrix, testthat, ggplot2, pls, playwith, latticist, svUnit, sp, baseline, compiler, inline, Rcpp

**Collate** ‘validate.R’ ‘hyperSpec-class.R’ ‘aggregate.R’ ‘all.equal.R’ ‘apply.R’ ‘paste.row.R’ ‘Arith.R’ ‘as.data.frame.R’ ‘barbiturates.R’ ‘bind.R’ ‘call.list.R’ ‘package.R’ ‘utils.R’ ‘initialize.R’ ‘labels.R’ ‘laser.R’ ‘plotmap.R’ ‘levelplot.R’ ‘logbook.R’ ‘logentry.R’ ‘log.R’ ‘map.identi

**Repository** CRAN

**Date/Publication** 2011-09-28 07:02:20

**R topics documented:**

hyperSpec-package	3
aggregate	4
apply	6
Arith	7
asdataframe	9
barbiturates	11
baselines	11
bind	13
chk-dot-hy	15
chondro	15
collapse	16
Comparison	18
decomposition	19
dim	21
dimnames	22
dot-DollarNames-dot-hyperSpec	24
empty	24
extractreplace	25
flu	29
hyperSpec-class	30
initialize	32
labels	34
laser	35
levelplot	36
logbook	39
logentry	40
map-sel-poly	41
map.sel.poly	42
mark.dendrogram	42
math	44
mean_sd	45
merge	47
options	48
orderwl	49
palettes	50
paracetamol	51
pearson-dot-dist	52
plot	53
plotc	54
plotspc	56
qplotc	59
qplotmap	60
qplotspc	61
rbind-fill	62
read-spc	64
readENVI	66

sample . . . . .	68
scan-dot-txt-dot-Witec . . . . .	70
scan-txt-Renishaw . . . . .	71
seq . . . . .	72
show . . . . .	73
spc-bin . . . . .	75
spc-identify . . . . .	76
spc-loess . . . . .	79
spc.NA.linapprox . . . . .	80
split . . . . .	81
summary . . . . .	82
sweep . . . . .	83
textio . . . . .	84
trellis-dot-factor-dot-key . . . . .	87
unittests . . . . .	88
wc . . . . .	89
wl . . . . .	89
wl2i . . . . .	91

<b>Index</b>	<b>93</b>
--------------	-----------

---

hyperSpec-package	<i>Package hyperSpec</i>
-------------------	--------------------------

---

## Description

Interface for hyperspectral data sets This package gives an interface to handle hyperspectral data sets in R. Hyperspectral data are spatially or time-resolved spectra, or spectra with any other kind of information associated with the spectra. E.g. spectral maps or images, time series, calibration series, etc.

## Details

The spectra can be data as obtained in XRF, UV/VIS, Fluorescence, AES, NIR, IR, Raman, NMR, MS, etc.

More generally, any data that is recorded over a discretized variable, e.g. absorbance = f (wavelength), stored as a vector of absorbance values for discrete wavelengths is suitable.

## Author(s)

C. Beleites

Maintainer: Claudia Beleites <cbeleites@units.it>

## See Also

`citation ("hyperSpec")` produces the correct citation.

`package?hyperSpec` for information about the package

`class?hyperSpec` for details on the S4 class provided by this package.

---

 aggregate

 aggregate hyperSpec objects
 

---

### Description

aggregate Computes summary statistics for subsets of a hyperSpec object.

### Usage

```
## S4 method for signature 'hyperSpec'
aggregate(x, by=stop("by is needed"), FUN=stop("FUN is needed."), ...,
  out.rows=NULL, append.rows=NULL, by.isindex=FALSE,
  short="aggregate", date=NULL, user=NULL)
```

### Arguments

x	a hyperSpec object
by	grouping for the rows of x@data. Either a list containing an index vector for each of the subgroups or a vector that can be split in such a list.
FUN	function to compute the summary statistics
out.rows	number of rows in the resulting hyperSpec object, for memory preallocation.
append.rows	If more rows are needed, how many should be appended? Defaults to 100 or an estimate based on the percentage of groups that are still to be done, whatever is larger.
by.isindex	If a list is given in by: does the list already contain the row indices of the groups? If FALSE, the list in by is computed first (as in <a href="#">aggregate</a> ).
...	further arguments passed to FUN
short,date,user	arguments passed to logentry

### Details

aggregate applies FUN to each of the subgroups given by by. It combines the functionality of [aggregate](#), [tapply](#), and [ave](#) for hyperSpec objects.

aggregate avoids splitting x@data.

FUN does not need to return exactly one value. The number of returned values needs to be the same for all wavelengths (otherwise the result could not be a matrix), see the examples.

If the initially preallocated data.frame turns out to be too small, more rows are appended and a warning is issued.

### Value

A hyperSpec object with an additional column @data\$.aggregate tracing which group the rows belong to.

**Author(s)**

C. Beleites

**See Also**[tapply](#), [aggregate](#), [ave](#)**Examples**

```
cluster.means <- aggregate (chondro, chondro$clusters, mean_pm_sd)
plot(cluster.means, stacked = ".aggregate", fill = ".aggregate",
col = matlab.dark.palette (3))

## make some "spectra"
spc <- new ("hyperSpec", spc = sweep (matrix (rnorm (10*20), ncol = 20), 1, (1:10)*5, "+"))

## 3 groups
color <- c("red", "blue", "black")
by <- as.factor (c (1, 1, 1, 1, 1, 1, 5, 1, 2, 2))
by
plot (spc, "spc", col = color[by])

## Example 1: plot the mean of the groups
plot (aggregate (spc, by, mean), "spc", col = color, add = TRUE,
lines.args = list(lwd = 3, lty = 2))

## Example 2: FUN may return more than one value (here: 3)
plot (aggregate (spc, by, mean_pm_sd), "spc",
col = rep(color, each = 3), lines.args = list(lwd = 3, lty = 2))

## Example 3: aggregate even takes FUN that return different numbers of
##          values for different groups
plot (spc, "spc", col = color[by])

weird.function <- function (x){
  if (length (x) == 1)
    x + 1 : 10
  else if (length (x) == 2)
    NULL
  else
    x [1]
}

agg <- aggregate (spc, by, weird.function)
agg$.aggregate
plot (agg, "spc", add = TRUE, col = color[agg$.aggregate],
lines.args = list (lwd = 3, lty = 2))
```

---

apply	<i>apply...</i>
-------	-----------------

---

### Description

apply Computes summary statistics for the spectra of a hyperSpec object.

### Usage

```
## S4 method for signature 'hyperSpec'
apply(X, MARGIN, FUN, ..., label.wl=NULL, label.spc=NULL,
      new.wavelength=NULL, short="apply", long=NULL, user=NULL, date=NULL)
```

### Arguments

X, spc	a hyperSpec object
MARGIN	The subscript which the function will be applied over. 1 indicates rows (FUN is applied to each spectrum), 2 indicates columns (FUN is applied to each wavelength), 1 : 2 indicates that FUN should be applied to each single element of the spectra matrix. Note that many basic mathematical functions are already defined for hyperSpec objects (see <a href="#">Math</a> ). If MARGIN is missing, the whole spectra matrix is handed to FUN, see also the examples.
FUN	function to compute the summary statistics
...	further arguments passed to FUN
label.wl, label.spc	new labels for wavelength and spectral intensity axes
new.wavelength	for MARGIN = 2: numeric vector or name of the argument in ... that is to be used (character) as wavelength axis of the resulting object.
short, long, user, date	arguments passed to logentry

### Details

apply gives the functionality of [apply](#) for hyperSpec objects.

The generic functions of group [Math](#) are not defined for hyperSpec objects. Instead, apply can be used. For functions like log that work on scalars, MARGIN = 1 : 2 gives the appropriate behaviour.

spcapply does the same as apply with MARGIN = 1, but additionally allows to set a new wavelength axis and adjust the labels.

wlapply does the same as apply with MARGIN = 2, but additionally allows to set a new wavelength axis and adjust the labels.

**Value**

A hyperSpec object

**Author(s)**

C. Beleites

**See Also**

[apply](#), for applying FUN to subgroups of the hyperSpec object: [aggregate](#).

**Examples**

```
plotspc (apply (chondro, 2, range))

avgflu <- apply (flu, 1, mean,
  label.spc = expression (bar (I)),
  new.wavelength = mean (wl (flu)))
avgflu

flu[[,405:407]]
apply (flu, 1:2, "*", -1)[[,405:407]]

## without MARGIN the whole matrix is handed to FUN
apply (flu [,405:407], , print) [[]]

## whereas MARGIN = 1 : 2 leads to FUN being called for each element separately
apply (flu [,405:407], 1 : 2, print) [[]]
```

---

Arith

*Arithmetical Operators: +, -, \*, /, ^, %%, %/%, %\*%*


---

**Description**

The arithmetical operators +, -, \*, /, ^, %%, %/%, and %\*% for hyperSpec objects.

**Usage**

```
## S4 method for signature 'hyperSpec,hyperSpec'
Arith(e1, e2)
## S4 method for signature 'hyperSpec,matrix'
Arith(e1, e2)
## S4 method for signature 'matrix,hyperSpec'
Arith(e1, e2)
## S4 method for signature 'hyperSpec,numeric'
Arith(e1, e2)
## S4 method for signature 'numeric,hyperSpec'
Arith(e1, e2)
```

```
## S4 method for signature 'hyperSpec,missing'
Arith(e1, e2)
x %**% y
```

### Arguments

e1, e2	Either two hyperSpec objects or one hyperSpec object and matrix of same size as hyperSpec[[]] or a scalar (numeric of length 1).
x, y	Either two hyperSpec objects or one hyperSpec object and one matrix of appropriate size.

### Details

You can use these operators in different ways:

```
e1 + e2
'+ ' (e1, e2)
```

```
x %**% y
'%**%' (x, y)
```

```
-x
```

The arithmetical operators `+`, `-`, `*`, `/`, `^`, `%%`, `%/ %`, and `%**%` work on the spectra matrix of the hyperSpec object. They have their usual meaning (see [Arithmetic](#)). The operators work also with one hyperSpec object and a numeric object or a matrices of the same size as the spectra matrix of the hyperSpec object.

With numeric vectors [sweep](#) is most probably more appropriate.

If you want to calculate on the data.frame hyperSpec@data, you have to do this directly on hyperSpec@data.

### Value

hyperSpec object with the new spectra matrix.

### Author(s)

C. Beleites

### See Also

[sweep-methods](#) for calculations involving a vector and the spectral matrix.

[S4groupGeneric](#) for group generic methods.

[Arithmetic](#) for the base arithmetic functions.

[matmult](#) for matrix multiplications with `%**%`.

[Comparison](#) for comparison operators, [Math](#) for mathematical group generic functions (Math and Math2 groups) working on hyperSpec objects.

**Examples**

```
chondro + chondro
1 / chondro
all((chondro + chondro - 2 * chondro)[[]] == 0)
-flu
```

---

asdataframe

*Conversion of a hyperSpec object into a data...*


---

**Description**

Conversion of a hyperSpec object into a data.frame or matrix as.data.frame returns x@data (as data.frame) as.matrix returns the spectra matrix x@data\$spc as matrix

**Usage**

```
## S4 method for signature 'hyperSpec,missing,missing'
as.data.frame(x, row.names=stop("hyperSpec method does not support row.names"),
  optional=stop("hyperSpec method does not support row.names"), ...)
```

```
## S4 method for signature 'hyperSpec'
as.matrix(x, ...)
```

```
as.wide.df(x)
```

```
as.long.df(x, rownames=FALSE, wl.factor=FALSE, na.rm=TRUE)
```

```
as.t.df(x)
```

**Arguments**

x	a hyperSpec object
row.names, optional	must be missing: they are not supported for hyperSpec objects.
...	ignored
rownames	should the rownames be in column .rownames of the long-format data.frame?
wl.factor	should the wavelengths be returned as a factor (instead of numeric)?
na.rm	if TRUE, rows where spc is not NA are deleted.

**Details**

as.long.df: The data.frame returned by as.long.df is guaranteed to have columns spc and .wavelength. If nwl(x) == 0 these columns will be NA.

**Value**

`as.data.frame`, `hyperSpec`, `missing`, `missing-method`: `x@data` and `x@data$spc` (`== x$spc == x [ [ ] ]`), respectively.

`as.wide.df`: `as.wide.df` returns a data.frame that consists of the extra data and the spectra matrix converted to a data.frame. The spectra matrix is expanded *in place*.

`as.long.df`: `as.long.df` returns the stacked or molten version of `x@data`. The wavelengths are in column `.wavelength`.

`as.t.df`: `as.t.df` returns a data.frame similar to `as.long.df`, but each spectrum in its own column. This is useful for exporting summary spectra, see the example.

**Author(s)**

C. Beleites

**See Also**

[as.data.frame](#) and [as.matrix](#)

[ for a shortcut to [as.matrixstack](#) and [melt](#) for other functions producing long-format data.frames.

**Examples**

```
as.data.frame (chondro [1:3,, 600:620])
as.matrix (chondro [1:3,, 600:620])

as.wide.df (chondro [1:5,, 600 ~ 610])
summary (as.wide.df (chondro [1:5,, 600 ~ 610]))

as.long.df (flu [, , 405 ~ 410])
summary (as.long.df (flu [, , 405 ~ 410]))
summary (as.long.df (flu [, , 405 ~ 410], rownames = TRUE))
summary (as.long.df (flu [, , 405 ~ 410], wl.factor = TRUE))

df <- as.t.df (apply (chondro, 2, mean_pm_sd))
head (df)

if (require (ggplot2)){
  ggplot (df, aes (x = .wavelength)) +
  geom_ribbon (aes (ymin = mean.minus.sd, ymax = mean.plus.sd),
  fill = "#00000040") +
  geom_line (aes (y = mean))
}
```

---

barbiturates	<i>Barbiturates Spectra from...</i>
--------------	-------------------------------------

---

**Description**

Barbiturates Spectra from .spc example files A time series of mass spectra in a list of hyperSpec objects.

**Author(s)**

C. Beleites and Thermo Galactic

**References**

The raw data is available at <http://hyperspec.r-forge.r-project.org/fileio.zip>

**Examples**

```
barbiturates [1:3]
length (barbiturates)

barb <- collapse (barbiturates)
barb <- orderwl (barb)

plot (barb [1:3], lines.args = list (type = "h"),
      col = matlab.dark.palette (3), stacked = TRUE,
      stacked.args = list (add.factor = .2))

if (require (latticeExtra)){
  plot (log (barb), "mat", panel = panel.levelplot.points, cex = 0.3,
        col = "#00000000", col.regions = matlab.palette (20))
}

plotc (apply (barb [, , 42.9~43.2], 1, sum, na.rm = TRUE), spc ~ z,
       panel = panel.lines, ylab = expression (I[m/z == 43] / "a.u."))
```

---

baselines	<i>Polynomial Baseline Fitting...</i>
-----------	---------------------------------------

---

**Description**

Polynomial Baseline Fitting These functions fit polynomial baselines.

**Usage**

```
spc.fit.poly(fit.to, apply.to=NULL, poly.order=1, short="spc.fit.poly", user=NULL,
            date=NULL)
```

```
spc.fit.poly.below(fit.to, apply.to=fit.to, poly.order=1, npts.min=NULL, noise=0,
                  short="spc.fit.poly.below", user=NULL, date=NULL)
```

**Arguments**

<code>fit.to</code>	hyperSpec object on which the baselines are fitted
<code>apply.to</code>	hyperSpec object on which the baselines are evaluated. If NULL, a hyperSpec object containing the polynomial coefficients rather than evaluated baselines is returned.
<code>poly.order</code>	order of the polynomial to be used
<code>short, user, date</code>	handed to logentry
<code>npts.min</code>	minimal number of points used for fitting the polynomial
<code>noise</code>	noise level to be considered during the fit. It may be given as one value for all the spectra, or for each spectrum separately.

**Details**

`spc.fit.poly`: Both functions fit polynomials to be used as baselines. If `apply.to` is NULL, a hyperSpec object with the polynomial coefficients is returned, otherwise the polynomials are evaluated on the spectral range of `apply.to`.

`spc.fit.poly` calculates the least squares fit of order `poly.order` to the *complete* spectra given in `fit.to`. Thus `fit.to` needs to be cut appropriately.

**Value**

`spc.fit.poly`: hyperspec object containing the baselines in the spectra matrix, either as polynomial coefficients or as polynomials evaluated on the spectral range of `apply.to`

**Author(s)**

C. Beleites

**See Also**

`vignette("baseline", package = "hyperSpec")`

**Examples**

```
## Not run: vignette("baseline", package = "hyperSpec")

baselines <- spc.fit.poly(chondro[, , c(625 ~ 640, 1785 ~ 1800)], chondro)
plot(chondro - baselines, "spcprct15")
```

```
baselines <- spc.fit.poly.below(chondro)
plot(chondro - baselines, "spcprct15")
```

---

bind

*Binding hyperSpec Objects...*


---

### Description

Binding hyperSpec Objects Two S3 functions `cbind.hyperSpec` and `rbind.hyperSpec` act as an interfaces to `cbind2` and `rbind2` because neither `rBind` and `cBind` nor S4 versions of `cbind` and `rbind` do work at the moment.

### Usage

```
bind(direction=stop("direction ('c' or 'r') required"), ..., short="bind",
      user=NULL, date=NULL)
```

```
cbind.hyperSpec(..., short="cbind", deparse.level)
```

```
rbind.hyperSpec(..., short="rbind", deparse.level)
```

```
## S4 method for signature 'hyperSpec,hyperSpec'
cbind2(x, y)
```

```
## S4 method for signature 'hyperSpec,missing'
cbind2(x, y)
```

```
## S4 method for signature 'hyperSpec,hyperSpec'
rbind2(x, y)
```

```
## S4 method for signature 'hyperSpec,missing'
rbind2(x, y)
```

### Arguments

```
...           The hyperSpec objects to be combined.
              Alternatively, one list of hyperSpec objects can be given to bind.
deparse.level ignored.
short,user,date for the log
```

direction "r" or "c" to bind rows or columns  
 x,y hyperSpec objects

### Details

bind: While it is now possible to do S4 dispatch on `...{}`, defining such S4 methods for `cbind` and `rbind` breaks the binding of `Matrix` objects. Therefore, two S3 methods `rbind.hyperSpec` and `cbind.hyperSpec` are defined.

bind does the common work for both column- and row-wise binding.

### Value

bind: a hyperSpec object, possibly with different row order (for `bind ("c", ...{ })` and `cbind2`).

### Note

You might have to make sure that the objects either all have or all do not have rownames and/or colnames.

### Author(s)

C. Beleites

### See Also

[rBind](#), [cBind](#) [rbind2](#), [cbind2](#) [rbind](#), [cbind](#)

[merge](#) and [collapse](#) for combining objects that do not share spectra or wavelengths, respectively.

### Examples

```
chondro
bind ("r", chondro, chondro)
rbind (chondro, chondro)
cbind (chondro, chondro)
bind ("r", list (chondro, chondro, chondro))

x <- chondro[, , 600 : 605]
x$a <- 1
x@data <- x@data[, sample (ncol (x), ncol (x))] # reorder columns

y <- chondro [nrow (chondro) : 1, , 1730 : 1750] # reorder rows
y$b <- 2

cbind2 (x, y) # works

y$y[3] <- 5
try (cbind2 (x, y)) # error
```

---

chk-dot-hy	<i>Validation of hyperSpec objects</i>
------------	--

---

**Description**

Check whether an object is a hyperSpec object and validate the object

**Usage**

```
chk.hy(object)
```

**Arguments**

object            the object to check

**Value**

TRUE if the check passes, otherwise stop with an error.

**Author(s)**

C. Beleites

**See Also**

[validObject](#)

**Examples**

```
chk.hy (chondro)
validObject (chondro)
```

---

chondro	<i>Raman spectra of 2 Chondrocytes in Cartilage...</i>
---------	--

---

**Description**

Raman spectra of 2 Chondrocytes in Cartilage A Raman-map (laterally resolved Raman spectra) of chondrocytes in cartilage.

**Details**

See the vignette.

**Author(s)**

A. Bonifacio and C. Beleites

## References

The raw data is available at <http://hyperspec.r-forge.r-project.org/chondro.zip>

## Examples

```
chondro

## do baseline correction
baselines <- spc.fit.poly.below (chondro)
chondro <- chondro - baselines

## area normalization
chondro <- sweep (chondro, 1, apply (chondro, 1, mean), "/")

## subtract common composition
chondro <- sweep (chondro, 2, apply (chondro, 2, quantile, 0.05), "-")

## PCA
pca <- prcomp (~ spc, data = chondro$, center = TRUE)
scores <- decomposition (chondro, pca$x, label.wavelength = "PC", label.spc = "score / a.u.")
loadings <- decomposition (chondro, t(pca$rotation), scores = FALSE, label.spc = "loading I / a.u.")

# remove outliers
out <- c(105, 140, 216, 289, 75, 69)
chondro <- chondro [- out]

# Hierarchical cluster analysis
dist <- dist (chondro [[]])
dendrogram <- hclust (dist, method = "ward")

plot (dendrogram)
clusters <- as.factor (cutree (dendrogram, k = 3))

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

cluster.means <- aggregate (chondro, chondro$clusters, mean_pm_sd)
plot(cluster.means, stacked = ".aggregate", fill = ".aggregate", col = cols)

## plot nucleic acids
plotmap (chondro[, , c( 728, 782, 1098, 1240, 1482, 1577)],
col.regions = colorRampPalette (c("white", "gold", "dark green"), space = "Lab") (20))

## Not run: vignette ("chondro", package = "hyperSpec")
```

**Description**

collapse/bind several hyperSpec objects into one object

**Usage**

```
collapse(..., short.log=TRUE, short="collapse", user=NULL, date=NULL)
```

**Arguments**

... hyperSpec objects to be collapsed into one object. Instead of giving several arguments, a list with all objects to be collapsed may be given.

short.log if TRUE, only the dimensions of the hyperSpec objects are logged by collapse.

short, user, date handed over to logentry

**Details**

The spectra from all objects will be put into one object. The resulting object has all wavelengths that occur in the input objects. Data points corresponding to wavelengths not in the original spectrum will be set to NA. Extra data is combined in the same manner.

**Value**

a hyperSpec object

**Author(s)**

C. Beleites

**See Also**

[merge](#) to merge hyperSpec objects that share wavelengths but contain different spectra, [rbind](#), and [rbind.fill](#) for

**Examples**

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

a <- barbiturates [[1]]
b <- barbiturates [[2]]
c <- barbiturates [[3]]

a
b
c
collapse (a, b, c)
```

---

Comparison	<i>all.equal,-method</i>
------------	--------------------------

---

### Description

Comparison Operators: <, >, <=, >=, ==, and != The comparison operators >, <, >=, <=, ==, and != for hyperSpec objects.

### Arguments

target, current	two hyperSpec objects that are tested for equality
...	handed to <a href="#">all.equal</a> when testing the slots of the hyperSpec objects
check.column.order	If two objects have the same data, but the order of the columns (determined by the names) differs, should they be regarded as different?
check.label	Should the slot label be checked? If the labels differ only in the order of their entries, they are considered equal.
check.log	Should the slot label be checked?
e1, e2	Either two hyperSpec objects or one hyperSpec object and matrix of same size as hyperSpec[[ ]] or a scalar (numeric of length 1). As hyperSpec objects must have numeric spectra matrices, the resulting matrix of the comparison is returned directly.

### Details

Comparison, -method: `all.equal` checks the equality of two hyperSpec objects.

The comparison operators >, <, >=, <=, ==, and != work on the spectra matrix of the hyperSpec object. They have their usual meaning (see [Comparison](#)). The operators work also with one hyperSpec object and a numeric (scalar) object or a matrices of the same size as the spectra matrix of the hyperSpec object.

With numeric vectors [sweep](#) might be more appropriate.

If you want to calculate on the data.frame hyperSpec@data, you have to do this directly on hyperSpec@data.

### Value

`all.equal, -method`: `all.equal` returns either TRUE, or a character vector describing the differences. In conditions, the result must therefore be tested with `isTRUE`.

Comparison, -method: a logical matrix for the comparison operators.

### Author(s)

C. Beleites

**See Also**

[all.equal](#) and [isTRUEsweep-methods](#) for calculations involving a vector and the spectral matrix.

[S4groupGeneric](#) for group generic methods.

[Comparison](#) for the base comparison functions.

[Arith](#) for arithmetic operators, [Math](#) for mathematical group generic functions (groups [Math](#) and [Math2](#)) working on hyperSpec objects.

**Examples**

```
all.equal (flu, --flu);
```

```
flu [, ,445 ~ 450] > 300
```

```
all (flu == flu[[ ]])
```

---

decomposition	<i>Convert Principal Component Decomposition or the like into a hyperSpec Object...</i>
---------------	---

---

**Description**

Convert Principal Component Decomposition or the like into a hyperSpec Object Decomposition of the spectra matrix is a common procedure in chemometrix data analysis. `decomposition` converts the result matrices into new hyperSpec objects.

**Usage**

```
decomposition(object, x, wavelength=seq_len(ncol(x)), label.wavelength, label.spc,
  scores=TRUE, retain.columns=FALSE, short="decomposition",
  user=NULL, date=NULL, ...)
```

**Arguments**

<code>object</code>	A hyperSpec object.
<code>x</code>	matrix with the new content for <code>object@data\$spc</code> . May correspond to rows (like a scores matrix) or columns (like a loadings matrix) of <code>object</code> .
<code>wavelength</code>	for a scores-like <code>x</code> : the new <code>object@wavelength</code> .
<code>label.wavelength</code>	The new label for the wavelength axis (if <code>x</code> is scores-like)
<code>label.spc</code>	The new label for the spectra matrix
<code>scores</code>	is <code>x</code> a scores-like matrix?

`retain.columns` for loading-like decomposition (i.e. `x` holds loadings, pure component spectra or the like), the data columns need special attention. Columns with different values across the rows will be set to NA if `retain.columns` is TRUE, otherwise they will be deleted.

`short, user, date` handed over to `logentry`

`...` ignored.

## Details

Multivariate data are frequently decomposed by methods like principal component analysis, partial least squares, linear discriminant analysis, and the like. These methods yield loadings (or latent variables, components, ...) that are linear combination coefficients along the wavelength axis and scores for each spectrum and loading.

The loadings matrix gives a coordinate transformation, and the scores are values in that new coordinate system.

The obtained loadings are spectra-like objects: a loading has a coefficient for each wavelength. If such a matrix (with the same number of columns as `object` has wavelengths) is given to `decomposition`, the spectra matrix is replaced by `x`. Moreover, all columns of `object@data` that did not contain the same value for all spectra are set to NA. Thus, for the resulting `hyperSpec` object, `plotspc` and related functions are meaningful. `plotmap` cannot be applied as the loadings are not laterally resolved.

The scores-matrix needs to have the same number of rows as `object` has spectra. If such a matrix is given, the spectra matrix is replaced by `x` and `object@wavelength` is replaced by `wavelength`. The information related to each of the spectra is retained. For such a `hyperSpec` object, `plotmap` and `plotc` and the like can be applied. Of course, it is also possible to use the spectra plotting, but the interpretation is not that of the spectrum any longer.

## Value

A `hyperSpec` object, updated according to `x`

## Author(s)

C. Beleites

## See Also

See `%%` for matrix multiplication of `hyperSpec` objects.

See e.g. `prcomp` and `princomp` for principal component analysis, and package `pls` for Partial Least Squares Regression.

## Examples

```
pca <- prcomp (~ spc, data = flu$. , center = FALSE)

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

```

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

plotspc (loadings, stacked = TRUE, col = matlab.palette(6))

plotc (scores, groups = .wavelength,col = matlab.palette(6), type = "b")
pca$sdev

## everything besides the first component is just noise
## Reconstructing the data using only the first PC results in a noise
## filtered data set.

flu.filtered <- scores[, ,1]

## example 2
pca <- prcomp (~ spc, data = chondro$. , tol = 0.1)

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

plotmap (scores[, ,1])

```

---

dim	<i>The Number of Rows (Spectra), Columns, and Data Points per Spectrum of an...</i>
-----	---

---

### Description

The Number of Rows (Spectra), Columns, and Data Points per Spectrum of an hyperSpec Object)

### Usage

```

## S4 method for signature 'hyperSpec'
ncol(x)

## S4 method for signature 'hyperSpec'
nrow(x)

nwl(x)

## S4 method for signature 'hyperSpec'
dim(x)

## S4 method for signature 'hyperSpec'

```

length(x)

### Arguments

x                    a hyperSpec object

### Details

ncol,hyperSpec-method: ncol returns the number of columns in x@data. I.e. the number of columns with additional information to each spectrum (e.g. "x", "y", ...) + 1 (for column spc containing the spectra).

### Value

ncol,hyperSpec-method: nrow, ncol, nwl, and length, return an integer.

dim,hyperSpec-method: dim returns a vector of length 3.

### Author(s)

C. Beleites

### See Also

[ncolnrowdimlength](#)

### Examples

```
ncol (chondro)
```

```
nrow (chondro)
```

```
nwl (chondro)
```

```
dim (chondro)
```

```
length (chondro)
```

---

dimnames

*Dimnames for hyperSpec objects...*

---

### Description

Dimnames for hyperSpec objects

**Usage**

```
## S4 method for signature 'hyperSpec'  
dimnames(x)  
  
## S4 method for signature 'hyperSpec'  
rownames(x, do.NULL=TRUE, prefix="row")  
  
## S4 replacement method for signature 'hyperSpec'  
rownames(x) <- value  
  
## S4 method for signature 'hyperSpec'  
colnames(x, do.NULL=TRUE, prefix="col")  
  
## S4 replacement method for signature 'hyperSpec'  
colnames(x) <- value
```

**Arguments**

x	the hyperSpec object
do.NULL	handed to <a href="#">rownames</a> or <a href="#">colnames</a> : logical. Should this create names if they are NULL?
prefix	handed to <a href="#">rownames</a> or <a href="#">colnames</a>
value	the new names

**Details**

`dimnames`, `hyperSpec`-method: `hyperSpec` objects can have row- and column names like `data.frames`. The "names" of the wavelengths are treated separately: see [w1](#)

**Author(s)**

C. Beleites

**See Also**

[w1](#) for the wavelength dimension  
[dimnamesrownamescolnames](#)

**Examples**

```
dimnames (flu)  
  
rownames (flu)  
  
colnames (chondro)
```

---

dot-DollarNames-dot-hyperSpec  
*command line completion for \$*

---

**Description**

command line completion for \$

**Usage**

`.DollarNames.hyperSpec(x, pattern)`

**Arguments**

x	the hyperSpecobject
pattern	pattern to look for

**Value**

the name of the extra data slot

**Author(s)**

C. Beleites

**See Also**

[.DollarNames](#)

---

empty *Empty hyperSpec object...*

---

**Description**

Empty hyperSpec object

**Usage**

`empty(x, nrow=0, spc=NA, extra=NA, short="empty", user=NULL, date=NULL)`

**Arguments**

x	hyperSpec object
nrow	number of rows the new object should have
spc	value to initialize the new spectra matrix with
extra	value to initialize the new extra data with
short	handed to logentry
user	handed to logentry
date	handed to logentry

**Details**

Empty produces an hyperSpec object with the same columns and wavelengths as x. The new object will either contain no rows at all (default), or the given number of rows with all data initialized to spc and extra, respectively.

**Author(s)**

C. Beleites

**Examples**

```
empty (chondro, nrow = 2, spc = 0)
```

---

extractreplace      *Extract and Replace parts of hyperSpec objects*

---

**Description**

These Methods allow to extract and replace parts of the hyperSpec object.

**Usage**

```
## S4 method for signature 'hyperSpec'
x[i, j, l, ..., wl.index = FALSE,
short = "[]", date = NULL, user = NULL, drop = FALSE]
## S4 method for signature 'hyperSpec'
x[[i, j, l, ..., wl.index = FALSE, drop = FALSE]]

## S4 method for signature 'hyperSpec'
x$name

## S4 replacement method for signature 'hyperSpec'
x[i, j, short = "[<-", date = NULL, user = NULL, ...] <- value

## S4 replacement method for signature 'hyperSpec'
```

```
x[[i, j, l, wl.index = FALSE, short = "[<-", date = NULL, user = NULL, ...]] <- value
## S4 replacement method for signature 'hyperSpec'
x$name <- value
```

## Arguments

<code>x</code>	a hyperSpec Object
<code>i</code>	row index: selects spectra [[ and code[[<- accept indexing with logical matrix or a n by 2 integer index matrix. In this case the indexing is done inside the spectra matrix. See the examples below.
<code>j</code>	selecting columns of <code>x@data</code>
<code>l</code>	selecting columns of the spectra matrix. If <code>l</code> is numeric, the default behaviour is treating <code>l</code> as wavelengths, <i>not</i> as indices.
<code>wl.index</code>	If TRUE (default), the value(s) in <code>l</code> are treated as column indices for the spectral matrix. Otherwise, the numbers in <code>l</code> are treated as wavelengths and the corresponding column indices are looked up first via <code>wl2i</code> .
<code>short, user, date</code>	handed to <a href="#">logentry</a> .
<code>drop</code>	For <code>[[</code> : drop unnecessary dimensions, see <a href="#">drop</a> and <a href="#">Extract</a> . Ignored for <code>[</code> , as otherwise invalid hyperSpec objects might result.
<code>...</code>	ignored
<code>name</code>	name of the data column to extract. <code>\$spc</code> yields the spectra matrix.
<code>value</code>	the replacement value

## Details

`[`, `hyperSpec`-method: They work with respect to the spectra (rows of `x`), the columns of the data matrix, and the wavelengths (columns of the spectra matrix).

Thus, they can be used for selecting/deleting spectra, cutting the spectral range, and extracting or setting the data belonging to the spectra.

Convenient shortcuts for access of the spectra matrix and the `data.frame` in slot `data` are provided.

*Extracting:* `[`, `[[`, and `$`.

The version with single square brackets (`[`) returns the resulting hyperSpec object.

`[[` yields `data.frame` of slot `@data` of that corresponding hyperSpec object returned with the same arguments by `[` if columns were selected (i.e. `j` is given), otherwise the spectra matrix `x@data$spc`.

`$` returns the selected column of the `data.frame` in slot `@data`.

*Shortcuts.* Three shortcuts to conveniently extract much needed parts of the object are defined:

`x[[[]]` returns the spectra matrix.

`x$.` returns the complete slot `@data`, including the spectra matrix in column `$spc`, as a `data.frame`.

`x$. .` returns a `data.frame` like `x$.` but without the spectra matrix.

*Replacing:* `[<-`, `[[<-`, and `$<-`.

```
## S4 method for signature 'hyperSpec':
x [i, j, l, short = NULL, ...] <- value

## S4 method for signature 'hyperSpec':
x [[i, j, l, wl.index = FALSE, short = NULL, ...]] <- value

## S4 method for signature 'hyperSpec':
x$name <- value
```

value gives the values to be assigned.

For \$, this can also be a list of the form list (value = value, label = label), with label containing the label for data column name.

[[<- replaces parts of the spectra matrix.

[<- replaces parts of the data.frame in slot x@data.

\$<- replaces a column of the data.frame in slot x@data. The value may be a list with two elements, value and label. In this case the label of the data column is changed accordingly.

\$. <- is again an abbreviation for the data.frame without the spectra matrix.

## Value

[, hyperSpec-method: For [, [<-, [[<-, and \$<- a hyperSpec object,

for [[ a matrix or data.frame, and

for \$ the column of the data.frame @data.

x[[[]] returns the complete spectra matrix.

x\$. returns the complete slot @data,

x\$. . returns the data.frame in @data but without the column @data\$spc containing the spectra matrix.

## See Also

[wl2i](#) on conversion of wavelength ranges to indices.

[drop](#) and [Extract](#) on drop.

## Examples

```
## index into the rows (spectra) -----
## make some "spectra"

## numeric index
plot (flu, "spc", lines.args = list (lty = 2))
plot (flu[1:3], "spc", add = TRUE, col = "red") # select spectra
plot (flu[-(1:3)], "spc", add = TRUE, col = "blue") # delete spectra

## logic index
plot (flu, "spc", lines.args = list (lty = 2))
index <- rnorm (6) > 0
```

```

index
plot (flu[index], "spc", add = TRUE, col = "red") # select spectra
plot (flu[!index], "spc", add = TRUE, col = "blue") # select spectra

## index into the data columns -----
range (chondro[["x"]])
colnames (chondro[[,1]])
dim (chondro[[,c(TRUE, FALSE, FALSE)]])
chondro$x

## the shortcut functions -----

## extract the spectra matrix
flu[[]]

## indexing via logical matrix
summary (flu [[flu < 125]])

## indexing the spectra matrix with index matrix n by 2
ind <- matrix (c (1, 2, 4, 406, 405.5, 409), ncol = 2)
ind
flu [[ind]]

ind <- matrix (c (1, 2, 4, 4:6), ncol = 2)
ind
flu [[ind, wl.index = TRUE]]

pca <- prcomp (flu[[]])

## result is data.frame, if j is given:
result <- flu [[, 1:2, 405 ~ 410]]
result
class (result)
colnames (result)

## extract the data.frame including the spectra matrix
flu$.
dim(flu$.)
colnames (flu$.)
flu$. $spc

calibration <- lm (spc ~ c, data = flu[, ,450]$.)
calibration

flu$.
colnames (flu$.)

## replacement functions
spc <- flu
spc$.
spc[, "c"] <- 16 : 11
## be careful:

```

```

plot (spc)
spc [] <- 6 : 1
spc$.
plot (spc)

spc <- flu [, , 405 ~ 410]
spc [[]]
spc [[3]] <- -spc[[3]]
spc [[]]
spc [[,405 : 410]] <- -spc[[,405 : 410]]
spc [[]]
spc [[,405 ~ 410]] <- -spc[[,405 ~ 410]]

## indexing with logical matrix
spc <- flu [, , min ~ 410]
spc < 125
spc [[spc < 125]] <- NA
spc [[]]

## indexing with n by 2 matrix
ind <- matrix (c (1, 2, 4, 406, 405.5, 409), ncol = 2)
ind
spc [[ind]] <- 3
spc [[]]

ind <- matrix (c (1, 2, 4, 4:6), ncol = 2)
ind
spc [[ind, wl.index = TRUE]] <- 9999
spc [[]]

spc$.
spc$z <- 1 : 6
spc
spc$z <- list (1 : 6, "z / a.u.")

```

---

flu

*Quinine Fluorescence Spectra...*


---

### Description

Quinine Fluorescence Spectra Fluorescence spectra of different dilutions of quinine forming a calibration set.

### Details

See the vignette.

### Author(s)

M. Kammer and C. Beleites

**Examples**

```

flu

plot (flu)

plotc (flu)

## Not run: vignette ("flu", package = "hyperSpec")

```

---

hyperSpec-class      *Class "hyperSpec"*

---

**Description**

This class handles hyperspectral data sets, i.e. spatially or time-resolved spectra, or spectra with any other kind of information associated with the spectra.

The spectra can be data as obtained in XRF, UV/VIS, Fluorescence, AES, NIR, IR, Raman, NMR, MS, etc.

More generally, any data that is recorded over a discretized variable, e.g. absorbance = f (wavelength), stored as a vector of absorbance values for discrete wavelengths is suitable.

**Objects from the Class**

Objects can be created by calls of the form `new("hyperSpec", spc, data, wavelength, label, log)`, for details see [initialize](#).

**Slots**

**wavelength:** Numeric vector giving the wavelengths (or wavenumbers, frequencies, m/z, ...) for each data point of the spectrum.

wavelength wavelength axis of the spectra.

**data:** data.frame with the spectra and further data for each spectrum (e.g. x/y/z coordinates, times, sample numbers, concentrations, etc.).

The spectra are stored in `data$spc`, preferably as a matrix.

**label:** List with the labels (character vectors or expressions) that should be used to describe the columns of data.

The label for wavelength is in `label$.wavelength`.

**log:** A data.frame with the columns `short.description`, `long.description`, `date`, and `name` tracking what is done with the object.

**Methods**

**aggregate** signature(x = "hyperSpec"): ...  
**all.equal** signature(target = "hyperSpec", current = "hyperSpec"): ...  
**apply** signature(X = "hyperSpec"): ...  
**Arith** signature(e1 = "hyperSpec", e2 = "hyperSpec"): ...  
**Arith** signature(e1 = "hyperSpec", e2 = "numeric"): ...  
**Arith** signature(e1 = "hyperSpec", e2 = "matrix"): ...  
**Arith** signature(e1 = "hyperSpec", e2 = "missing"): ...  
**Arith** signature(e1 = "numeric", e2 = "hyperSpec"): ...  
**Arith** signature(e1 = "matrix", e2 = "hyperSpec"): ...  
**as.character** signature(x = "hyperSpec"): ...  
**as.data.frame** signature(x = "hyperSpec", row.names = "missing", optional = "missing"): ...  
 ...  
**as.matrix** signature(x = "hyperSpec"): ...  
**[<-** signature(x = "hyperSpec"): ...  
**[** signature(x = "hyperSpec"): ...  
**[[<-** signature(x = "hyperSpec"): ...  
**[[** signature(x = "hyperSpec"): ...  
**\$<-** signature(x = "hyperSpec"): ...  
**\$** signature(x = "hyperSpec"): ...  
**%\*%** signature(x = "hyperSpec", y = "hyperSpec"): ...  
**%\*%** signature(x = "hyperSpec", y = "matrix"): ...  
**%\*%** signature(x = "matrix", y = "hyperSpec"): ...  
**cbind2** signature(x = "hyperSpec", y = "hyperSpec"): ...  
**cbind2** signature(x = "hyperSpec", y = "missing"): ...  
**colnames** signature(x = "hyperSpec"): ...  
**Compare** signature(e1 = "hyperSpec", e2 = "hyperSpec"): ...  
**Compare** signature(e1 = "hyperSpec", e2 = "numeric"): ...  
**Compare** signature(e1 = "hyperSpec", e2 = "matrix"): ...  
**Compare** signature(e1 = "numeric", e2 = "hyperSpec"): ...  
**Compare** signature(e1 = "matrix", e2 = "hyperSpec"): ...  
**dim** signature(x = "hyperSpec"): ...  
**dimnames** signature(x = "hyperSpec"): ...  
**initialize** signature(.Object = "hyperSpec"): ...  
**labels** signature(object = "hyperSpec"): ...  
**log** signature(x = "hyperSpec"): ...  
**Math2** signature(x = "hyperSpec"): ...

```

Math signature(x = "hyperSpec"): ...
ncol signature(x = "hyperSpec"): ...
nrow signature(x = "hyperSpec"): ...
plot signature(x = "hyperSpec", y = "missing"): ...
plot signature(x = "hyperSpec", y = "character"): ...
print signature(x = "hyperSpec"): ...
rbind2 signature(x = "hyperSpec", y = "hyperSpec"): ...
rbind2 signature(x = "hyperSpec", y = "missing"): ...
rownames signature(x = "hyperSpec"): ...
show signature(object = "hyperSpec"): ...
split signature(x = "hyperSpec"): ...
summary signature(object = "hyperSpec"): ...
Summary signature(x = "hyperSpec"): ...
sweep signature(x = "hyperSpec"): ...

```

### Author(s)

C. Beleites

### See Also

See the vignette "introduction" for an introduction to hyperSpec from a spectroscopic point of view.

### Examples

```

showClass("hyperSpec")
## Not run: vignette("introduction")

```

---

initialize

*Creating a hyperSpec Object...*

---

### Description

Creating a hyperSpec Object Like other S4 objects, a hyperSpec object can be created by new. The hyperSpec object is then initialized using the given parameters.

**Arguments**

<code>.Object</code>	the new <code>hyperSpec</code> object.
<code>data</code>	<code>data.frame</code> , possibly with the spectra in <code>data\$spc</code> , and further variates in more columns. A matrix can be entered as <i>one</i> column of a data frame by: <code>data.frame (spc = I (as.matrix (spc)))</code> . However, it will usually be more convenient if the spectra are given in <code>spc</code>
<code>spc</code>	the spectra matrix. <code>spc</code> does not need to be a matrix, it is converted explicitly by <code>I (as.matrix (spc))</code> .
<code>wavelength</code>	The wavelengths corresponding to the columns of <code>data</code> . If no wavelengths are given, an appropriate vector is derived from the column names of <code>data\$spc</code> . If this is not possible, <code>1 : ncol (data\$spc)</code> is used instead.
<code>label</code>	A list containing the labels for the columns of the data slot of the <code>hyperSpec</code> object and for the wavelength (in <code>label\$wavelength</code> ). The labels should be given in a form ready for the text-drawing functions (see <a href="#">plotmath</a> ). If <code>label</code> is not given, a list containing <code>NULL</code> for each of the columns of <code>data</code> and <code>wavelength</code> is used.
<code>log</code>	A list used to fill into the first entry of <code>.Object@log</code> . The elements <code>log\$short</code> , <code>log\$long</code> , <code>log\$date</code> , and <code>log\$user</code> are handed down to <a href="#">logentry</a> .

**Details**

If option `gc` is `TRUE`, the initialization will have frequent calls to `gc ()` which can help to avoid swapping or running out of memory.

**Author(s)**

C.Beleites

**See Also**

[new](#) for more information on creating and initializing S4 objects.  
[plotmath](#) on expressions for math annotations as for slot `label`.  
[hy.setOptions](#)

**Examples**

```
new ("hyperSpec")

spc <- matrix (rnorm (12), ncol = 4)
new ("hyperSpec", spc = spc)
new ("hyperSpec", data = data.frame (x = letters[1:3]),
    spc = spc)

colnames (spc) <- 600:603
new ("hyperSpec", spc = spc) # wavelength taken from colnames (spc)
```

```

# given wavelengths precede over colnames of spc
new ("hyperSpec", spc = spc, wavelength = 700:703)

# specifying labels
h <- new ("hyperSpec", spc = spc, data = data.frame (pos = 1 : 3),
label = list (spc = "I / a.u.",
.wavelength = expression (tilde (nu) / cm^-1),
pos = expression ("/" (x, mu*m)))
)

plot (h)
plotc (h, spc ~ pos)

# giving a log entry
new ("hyperSpec", log = list (short = "measurement parameters",
long = list ("exposure / s" = 10, sample = "327a")))

```

---

labels	<i>labels&lt;-</i>
--------	--------------------

---

## Description

Get and Set Labels of a hyperSpec Object value may be a list or vector of labels giving the new label for each of the entries specified by which.

## Usage

```

labels (object, which = NULL, ..., short = "labels<-", user = NULL, date = NULL) <- value
## S4 method for signature 'hyperSpec'
labels(object, which = bquote (), drop = TRUE, ...,
use.colnames = TRUE)

```

## Arguments

value	the new label(s)
short,user,date	handed to <a href="#">logentry</a>
object	a hyperSpec object
which	numeric or character to specify the label(s)
...	ignored
drop	if the result would be a list with only one element, should the element be returned instead?
use.colnames	should missing labels be replaced by column names of the extra data?

**Details**

labels, hyperSpec-method: The names of the labels are the same as the colnames of the data.frame. The label for the wavelength axis has the name .wavelength.

The labels should be given in a form ready for the text-drawing functions (see [plotmath](#)), e.g. as expression or a character.

**Value**

labels<-: labels<- returns a hyperSpec object.

labels, hyperSpec-method: labels returns a list of labels. If drop is TRUE and the list contains only one element, the element is returned instead.

**Author(s)**

C. Beleites

**See Also**

[labels](#)

**Examples**

```
labels (flu, "c") <- expression ("/" ("c", "mg / l"))
```

```
labels (chondro)
```

---

laser

*Laser Emission...*

---

**Description**

Laser Emission A time series of an unstable laser emission.

**Details**

see the Vignette

**Author(s)**

C. Beleites

**Examples**

```
laser

cols <- c ("black", "blue", "darkgreen", "red")
wl <- c (405.0, 405.1, 405.3, 405.4)
plotspc (laser, axis.args=list (x = list (at = seq (404.5, 405.8, .1))))
for (i in seq_along (wl))
  abline (v = wl[i], col = cols[i], lwd = 2, lty = 2)

plotc (laser [, , wl], spc ~ t, groups = .wavelength, type = "b",
  col = cols)

## Not run: vignette ("laser", package="hyperSpec")
```

levelplot

*levelplot,formula,hyperSpec-method***Description**

Plot a Map and Identify/Select Spectra in the Map [levelplot](#) functions for hyperSpec objects. An image or map of a summary value of each spectrum is plotted. Spectra may be identified by mouse click.

**Usage**

```
## S4 method for signature 'formula,hyperSpec'
levelplot(x, data, transform.factor = TRUE,
  panel = panel.levelplot.raster, ...)
## S4 method for signature 'hyperSpec,missing'
levelplot(x, data, ...)

map.identify(object, model=spc ~ x * y, voronoi=FALSE, ..., tol=0.02, warn=TRUE)

plotmap(object, model=spc ~ x * y, func=mean, func.args=list(), ...)

plotvoronoi(object, model=spc ~ x * y, use.tripack=FALSE, mix=FALSE, ...)
```

**Arguments**

transform.factor	If the color-coded variable is a factor, should <a href="#">trellis.factor.key</a> be used to compute the color coding and legend?
panel	panel function
tol	tolerance for <code>map.identify</code> as fraction of the viewport (i.e. in "npc" <a href="#">units</a> )

warn	should a warning be issued if no point is within the specified tolerance? See also details.
object, data	the hyperSpec object
model, x	formula specifying the columns of object that are to be displayed by <a href="#">levelplot</a>
func, func.args	Before plotting, <code>plotmap</code> applies function <code>func</code> with the arguments given in the list <code>func.args</code> to each of the spectra. Thus a single summary value is displayed for each of the spectra. This can be suppressed manually by setting <code>func</code> to <code>NULL</code> . It is automatically suppressed if <code>.wavelength</code> appears in the formula.
voronoi	Should the plot for identifying spectra by mouse click be produced by <code>plotmap</code> (default) or <code>plotvoronoi</code> ?
...	further arguments are passed down the call chain, and finally to <a href="#">levelplot</a>
use.tripack	Whether package <code>tripack</code> should be used for calculating the voronoi polygons. If <code>FALSE</code> , package <code>deldir</code> is used instead. See details.
mix	For Voronoi plots using package <code>tripack</code> , I experienced errors if the data was spatially ordered. Randomly rearranging the rows of the hyperSpec object circumvents this problem.

## Details

`plotmap`: The model can contain the special column name `.wavelength` to specify the wavelength axis.

`plotmap`, `map.identify`, and the `levelplot` methods internally use the same gateway function to [levelplot](#). Thus `transform.factor` can be used with all of them and the panel function defaults to [panel.levelplot.raster](#) for all three. Two special column names, `.rownames` and `.wavelength` may be used.

`levelplot` plots the spectra matrix.

`plotvoronoi` calls `plotmap` with different default settings, namely the panel function defaults to [panel.voronoi](#). [panel.voronoi](#) depends on either of the packages 'tripack' or 'deldir' being installed. For further information, please consult the help page of [panel.voronoi](#). On the [chondro](#) data set, `plotmap` is roughly 5 times faster than `plotvoronoi` using `tripack`, and ca. 15 times faster than `plotvoronoi` using `deldir`. Package `tripack`, however, is free only for non-commercial use. Also, it seems that `tripack` version hang (R running at full CPU power, but not responding nor finishing the calculation) for certain data sets. In this case, `mix = TRUE` may help.

`map.identify` calls `plotmap` and `plotvoronoi`, respectively and waits for (left) mouse clicks on points. Other mouse clicks end the input.

Unlike [panel.identify](#), the indices returned by `map.identify` are in the same order as the points were clicked. Also, multiple clicks on the same point are returned as multiple entries with the same index.

`map.identify` uses option `debuglevel` similar to [spc.identify](#): `debuglevel == 1` will plot the tolerance window if no data point was inside (and additionally labels the point) while `debuglevel == 2` will always plot the tolerance window.

The `map.sel.*` functions offer further interactive selection, see [map.sel.poly](#).

**Value**

plotmap: map.identify returns a vector of row indices into object of the clicked points.  
The other functions return a lattice object.

**Author(s)**

C. Beleites

**See Also**

[levelplot](#)

[trellis.factor.key](#) for improved color coding of factors  
[hyperSpec](#) options [spc.identify](#)  
[map.sel.poly](#)[vignette \(plotting\)](#), [vignette \(introduction\)](#)

[plotpanel.voronoi](#)

**Examples**

```
## Not run:
vignette (plotting)
vignette (introduction)

## End(Not run)

levelplot (chondro) # throws warning: only first wl is used
levelplot (spc ~ y * x, chondro [, ,1003]) # properly rotated
levelplot (chondro [, ,1003], aspect = "iso")
plotmap (chondro [, ,1003])

# plot spectra matrix
levelplot (spc ~ .wavelength * t, laser, contour = TRUE, col = "#00000080")
plot (flu, "mat")

# applying a function before plotting
plotmap (chondro, func = max, col.regions = gray (seq (0, 1, 0.05)))

plotmap (chondro, clusters ~ x * y)
plotmap (chondro, clusters ~ x * y, transform.factor = FALSE)
plotmap (chondro, clusters ~ x * y, col.regions = gray (seq (0, 1, 0.05)))

# Voronoi plots
smpl <- sample (chondro, 300)
plotmap (smpl, clusters ~ x * y)
if (require (tripack))
plotvoronoi (smpl, clusters ~ x * y)
if (require (deldir))
plotvoronoi (smpl, clusters ~ x * y,
use.tripack = FALSE)
```

---

`logbook`*Logging the processing of a hyperSpec Object...*

---

**Description**

Logging the processing of a hyperSpec Object Extracts the slot @log of a hyperSpec object.

**Usage**

```
logbook(x)
```

**Arguments**

`x` a hyperSpec object

**Details**

A data.frame in slot `x@log` keeps track of the changes done to the hyperSpec object.

If option `log` is TRUE, entries will be generated automatically by hyperSpec functions that do assignments (see [hy.setOptions](#)). Entries can also be created manually via [logentry](#).

**Value**

a data.frame containing `x@log`

**Author(s)**

C. Beleites

**See Also**

[hy.setOptions](#), [logentry](#).

**Examples**

```
logbook (flu)
```

logentry

*Append a Row to the logbook of a hyperSpec Object...*

---

**Description**

Append a Row to the logbook of a hyperSpec Object A log entry is generated and appended to the log of x.

**Usage**

```
logentry(x, short=NULL, long=NULL, date=NULL, user=NULL)
```

**Arguments**

x	a hyperSpec object
short	short description
long	long description, e.g. list with function arguments
date	time stamp
user	username

**Details**

The arguments (besides x) go into the respective columns of `x@log`.

The following values are used for any arguments that are NULL:

The name of the function that called `logentry` is used for `short`, and its call for `long`.

For `date`, the current `Sys.time()` is used, and `user` is constructed from `Sys.info()`'s `user` and `nodename`.

**Value**

`x@log (data.frame)` including the new row.

**Author(s)**

C. Beleites

**Examples**

```
logentry (chondro, short = "test")
```

---

`map-sel-poly`*Interactively select a polygon (grid graphics) and highlight points...*

---

**Description**

Interactively select a polygon (grid graphics) and highlight points

**Usage**

```
map.sel.poly(data, pch=19, size=0.3, ...)
```

**Arguments**

<code>data</code>	hyperSpec object for plotting map
<code>pch</code>	symbol to display the points of the polygon for <code>sel.poly</code>
<code>size</code>	size for polygon point symbol for <code>sel.poly</code>
<code>...</code>	further arguments for <code>grid.points</code> and <code>grid.lines</code>

**Details**

Click the points that should be connected as polygon. Input ends with right click (see `grid.locator`). Polygon will be drawn closed. Wrapper for `plotmap`, `sel.poly`, and `point.in.polygon`.

**Value**

array of indices for points within selected polygon

**Author(s)**

Claudia Beleites, Sebastian Mellor

**See Also**

`grid.locator`, `sel.poly`, `map.identify`

**Examples**

```
map.sel.poly (chondro)
```

---

map.sel.poly	<i>Interactively select a polygon (grid graphics)...</i>
--------------	--

---

**Description**

Interactively select a polygon (grid graphics)

**Usage**

```
sel.poly(pch=19, size=0.3, ...)
```

**Arguments**

pch	symbol to display the points of the polygon
size	size for polygon point symbol
...	further arguments for <a href="#">grid.points</a> and <a href="#">grid.lines</a>

**Details**

Click the points that should be connected as polygon. Input ends with right click (see [grid.locator](#)).

**Value**

n x 2 matrix with the points of the open polygon

**Author(s)**

Claudia Beleites

**See Also**

[grid.locator](#)

---

mark.dendrogram	<i>Mark groups in hclust dendrograms</i>
-----------------	--

---

**Description**

Groups are marked by colored rectangles as well as by their levels.

**Usage**

```
mark.dendrogram(dendrogram, groups, col=seq_along(unique(groups)), pos.marker=0,
  height=0.025 * max(dendrogram$height), pos.text=-2.5 * height,
  border=NA, text.col="black", label, label.right=TRUE, ...)
```

**Arguments**

dendrogram	the dendrogram
groups	factor giving the the groups to mark
col	vector with colors for each group
pos.marker	top of the marker rectangle
height	height of the marker rectangle
pos.text	position of the text label
border	see <a href="#">text</a>
text.col	color (vector) giving the color for the text markers
label	side label see example
label.right	should the side labels be at the right side?
...	handed to <a href="#">rect</a> and <a href="#">text</a>

**Details**

The dendrogram should be plotted separately, see the example.

**Author(s)**

Claudia Beleites

**Examples**

```
dend <- hclust (pearson.dist (laser[[ ]]))
par (xpd = TRUE, mar = c (5.1, 4, 4, 3)) # allows plotting into the margin
plot (dend, hang = -1, labels = FALSE)

## mark clusters
clusters <- as.factor (cutree (dend, k = 4))
levels (clusters) <- LETTERS [1 : 4]
mark.dendrogram (dend, clusters, label = "cluster")

## mark independent factor
mark.dendrogram (dend, as.factor (laser [, ,405.36] > 11000),
pos.marker = -0.02, pos.text = - 0.03)

## mark continuous variable: convert it to a factor and omit labels
mark.dendrogram (dend, cut (laser [, , 405.36]), 100), alois.palette (100),
pos.marker = -.015, text.col = NA,
label = expression (I [lambda == 405.36~nm]), label.right = FALSE)
```

---

math	<i>log,-method</i>
------	--------------------

---

### Description

Math Functions for hyperSpec Objects

### Arguments

x	the hyperSpec object
digits	integer stating the rounding precision

### Details

Math2, -method: The functions abs, sign, sqrt, floor, ceiling, trunc, round, signif, exp, log, expm1, log1p, cos, sin, tan, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh, lgamma, gamma, digamma, trigamma, cumsum, cumprod, cummax, cummin for hyperSpec objects.

### Value

Math2, -method: a hyperSpec object

### Author(s)

C. Beleites

### See Also

[S4groupGeneric](#) for group generic methods.

[Math](#) for the base math functions.

[Arith](#) for arithmetic operators, [Comparison](#) for comparison operators, and [Summary](#) for group generic functions working on hyperSpec objects.

### Examples

```
log (flu)
```

---

`mean_sd`*Mean and Standard Deviation...*

---

## Description

Mean and Standard Deviation Calculate mean and standard deviation, and mean, mean  $\pm$  one standard deviation, respectively.

## Usage

```
## S4 method for signature 'numeric'
mean_sd(x, na.rm=TRUE, ...)

## S4 method for signature 'matrix'
mean_sd(x, na.rm=TRUE, ...)

## S4 method for signature 'hyperSpec'
mean_sd(x, na.rm=TRUE, ..., short="mean_sd", user=NULL, date=NULL)

## S4 method for signature 'numeric'
mean_pm_sd(x, na.rm=TRUE, ...)

## S4 method for signature 'matrix'
mean_pm_sd(x, na.rm=TRUE, ...)

## S4 method for signature 'hyperSpec'
mean_pm_sd(x, na.rm=TRUE, ..., short="mean_sd", user=NULL, date=NULL)

## S4 method for signature 'hyperSpec'
mean(x, na.rm=TRUE, ..., short="mean", user=NULL, date=NULL)

## S4 method for signature 'hyperSpec'
quantile(x, probs=seq(0, 1, 0.25), na.rm=TRUE, names="num", ...,
  short="quantile", user=NULL, date=NULL)
```

## Arguments

<code>x</code>	a numeric vector
<code>na.rm</code>	handed to <a href="#">mean</a> and <a href="#">sd</a>

... ignored (needed to make function generic)  
 short, user, date handed to [logentry](#).  
 probs the quantiles, see [quantile](#)  
 names "pretty" results in percentages (like [quantile](#)'s names = TRUE), "num" results in the row names being as.character (probs) (good for ggplot2 getting the order of the quantiles right). Otherwise, no names are assigned.

### Details

mean\_sd,numeric-method: These functions are provided for convenience.

### Value

mean\_sd,numeric-method: mean\_sd returns a vector with two values (mean and standard deviation) of x.

mean\_sd,matrix-method: mean\_sd (matrix) returns a matrix with the mean spectrum in the first row and the standard deviation in the 2nd.

mean\_sd,hyperSpec-method: mean\_sd returns a hyperSpec object with the mean spectrum in the first row and the standard deviation in the 2nd.

mean\_pm\_sd,numeric-method: mean\_pm\_sd returns a vector with 3 values: mean - 1 sd, mean, mean + 1 sd

mean\_pm\_sd,matrix-method: mean\_pm\_sd (matrix) returns a matrix containing mean - sd, mean, and mean + sd rows.

mean\_pm\_sd,hyperSpec-method: For hyperSpec objects, mean\_pm\_sd returns a hyperSpec object containing mean - sd, mean, and mean + sd spectra.

mean,hyperSpec-method: For hyperSpec object, mean returns a hyperSpec object containing the mean spectrum.

quantile,hyperSpec-method: For hyperSpec object, quantile returns a hyperSpec object containing the respective quantile spectra.

### Author(s)

C. Beleites

### See Also

[mean](#), [sdmean](#), [sdquantile](#)

### Examples

```
mean_sd (flu [, , 405 ~ 410])
```

```
mean_sd (flu$spc)
```

```
mean_sd (flu)
```

```
mean_pm_sd (flu$c)
mean_pm_sd (flu$spc)
mean_pm_sd (flu)
plot (mean (chondro))
plot (quantile (chondro))
```

---

merge	<i>Merge hyperSpec objects...</i>
-------	-----------------------------------

---

## Description

Merge hyperSpec objects

## Usage

```
## S4 method for signature 'hyperSpec,hyperSpec'
merge(x, y, ..., short="merge", user=NULL, date=NULL)
```

## Arguments

x	a hyperSpec object
y	a hyperSpec object
...	handed to <a href="#">merge.data.frame</a>
short,user,date	handed to <a href="#">logentry</a>

## Details

Merges two hyperSpec objects and [cbinds](#) their spectra matrices.

After merging, the spectra matrix can contain duplicates, and is not ordered according to the wavelength.

If the wavelength axis should be ordered, use [orderwl](#).

## Author(s)

C. Beleites

## See Also

[merge](#).

[collapse](#) combines hyperSpec objects that do not share the wavelength axis. [rbind](#), and [cbind](#) for combining hyperSpec objects that.

**Examples**

```

merge (chondro [1:10,, 600], chondro [5:15,, 600], by = c("x", "y"))$.
tmp <- merge (chondro [1:10,, 610], chondro [5:15,, 610],
by = c("x", "y"), all = TRUE)
tmp$.
wl (tmp)

## remove duplicated wavelengths:
approxfun <- function (y, wl, new.wl){
  approx (wl, y, new.wl, method = "constant",
  ties = function (x) mean (x, na.rm = TRUE)
  )$y
}

merged <- merge (chondro [1:7,, 610 ~ 620], chondro [5:10,, 615 ~ 625], all = TRUE)
merged$.
merged <- apply (merged, 1, approxfun,
wl = wl (merged), new.wl = unique (wl (merged)),
new.wavelength = "new.wl")
merged$.

```

---

options

*Options for package hyperSpec...*


---

**Description**

Options for package hyperSpec Functions to access and set hyperSpec's options.

**Usage**

```
hy.getOptions(...)
```

```
hy.getOption(name)
```

```
hy.setOptions(...)
```

**Arguments**

...	hy.setOptions: pairs of argument names and values. hy.getOptions: indices (or names) of the options.
name	the name of the option

**Details**

hy.getOptions: Currently, the following options are defined:



**Details**

The wavelength vector is sorted and the columns of the spectra matrix are rearranged accordingly.

**Value**

A hyperSpec object.

**Author(s)**

C. Beleites

**See Also**

[order](#)

**Examples**

```
## Example 1: different drawing order in plotspc
spc <- new ("hyperSpec", spc = matrix (rnorm (5) + 1:5, ncol = 5))
spc <- cbind (spc, spc+.5)

plot (spc, "spc")
text (wl (spc), spc [[]], as.character (1:10))
spc <- orderwl (spc)
plot (spc, "spc")
text (wl (spc), spc [[]], as.character (1:10))

## Example 2
spc <- new ("hyperSpec", spc = matrix (rnorm (5)*2 + 1:5, ncol = 5))
spc <- cbind (spc, spc)

plot (seq_len(nwl(spc)), spc[[]], type = "b")
spc[[]]

spc <- orderwl (spc)
lines (seq_len(nwl(spc)), spc[[]], type = "l", col = "red")
spc[[]]
```

---

palettes

*Matlab-like Palettes...*

---

**Description**

Matlab-like Palettes Two palettes going from blue over green to red, approximately as the standard palette of Matlab does. The second one has darker green values and is better suited for plotting lines on white background.

**Usage**

```
matlab.palette(n=100, ...)
```

```
matlab.dark.palette(n=100, ...)
```

```
alois.palette(n=100, ...)
```

**Arguments**

`n` the number of colors to be in the palette.  
`...` further arguments are handed to [rainbow](#) (alois.palette: [colorRampPalette](#))

**Value**

`matlab.palette`: A vector containing the color values in the form "#rrbbggaa".

**Author(s)**

C. Beleites and A. Bonifacio

**See Also**

[rainbow](#)

**Examples**

```
plotmap (chondro [, , 778], col.regions = matlab.palette ())
```

```
plot (flu, col = matlab.dark.palette (nrow (flu)))
```

```
plotmap (chondro, col = alois.palette)
```

---

paracetamol

*Paracetamol Spectrum...*

---

**Description**

Paracetamol Spectrum A Raman spectrum of a paracetamol tablet.

**Author(s)**

C. Beleites

**Examples**

```
paracetamol

plot (paracetamol)
plotspc (paracetamol, c (min ~ 1750, 2800 ~ max), xoffset = 800,
wl.reverse = TRUE)
```

---

pearson-dot-dist      *Distance based on Pearson's  $R^2$  squared...*

---

**Description**

Distance based on Pearson's  $R^2$

**Usage**

```
pearson.dist(x)
```

**Arguments**

x                    a matrix

**Details**

The calculated distance is  $D^2 = \frac{1-COR(x')}{2}$

The distance between the rows of x is calculated. The possible values range from 0 (perfectly correlated) over 0.5 (uncorrelated) to 1 (perfectly anti-correlated).

**Value**

distance matrix (distance object)

**Author(s)**

C. Beleites

**References**

S. Theodoridis and K. Koutroumbas: Pattern Recognition, 3rd ed., p. 495

**See Also**

[as.dist](#)

**Examples**

```
dist <- pearson.dist (flu[[[]]])
```

---

plot

*Plotting hyperSpec Objects...*

---

## Description

Plotting hyperSpec Objects

## Arguments

x	the hyperSpec object
y	selects what plot should be produced
...	arguments passed to the respective plot function

## Details

plot-methods, -method: Plotting hyperSpec objects. The plot method for hyperSpec objects is a switchyard to [plotspc](#), [plotmap](#), and [plotc](#).

It also supplies some convenient abbreviations for much used plots.

If y is missing, plot behaves like plot (x, y = "spc").

Supported values for y are:

"**spc**" calls [plotspc](#) to produce a spectra plot.

"**spcmeansd**" plots mean spectrum +/- one standard deviation

"**spcprctile**" plots 16th, 50th, and 84th percentile spectre. If the distributions of the intensities at all wavelengths were normal, this would correspond to "spcmeansd". However, this is frequently not the case. Then "spcprctile" gives a better impression of the spectral data set.

"**spcprctil5**" like "spcprctile", but additionally the 5th and 95th percentile spectra are plotted.

"**map**" calls [plotmap](#) to produce a map plot.

"**c**" calls [plotc](#) to produce a calibration (or time series, depth-profile, or the like)

"**ts**" plots a time series: abbreviation for [plotc](#) (x, use.c = "t")

"**depth**" plots a depth profile: abbreviation for [plotc](#) (x, use.c = "z")

## Author(s)

C. Beleites

## See Also

[plotspc](#) for spectra plots (intensity over wavelength),

[plotmap](#) for plotting maps, i.e. color coded summary value on two (usually spatial) dimensions.

[plotc](#)

[plot](#)

**Examples**

```

plot (flu)

plot (flu, "c")

plot (laser, "ts")

spc <- apply (chondro, 2, quantile, probs = 0.05)
spc <- sweep (chondro, 2, spc, "-")
plot (spc, "spcprctl5")
plot (spc, "spcprctile")
plot (spc, "spcmeansd")

```

---

plotc

*Calibration- and Timeseries Plots, Depth-Profiles and the like...*


---

**Description**

Calibration- and Timeseries Plots, Depth-Profiles and the like plotc plots intensities of a hyperSpec object over another dimension such as concentration, time, or a spatial coordinate.

**Usage**

```
plotc(object, model=spc ~ c, groups=NULL, func=NULL, func.args=list(), ...)
```

**Arguments**

object	the hyperSpec object
model	the lattice model specifying the plot
func	function to compute a summary value from the spectra to be plotted instead of single intensities
func.args	further arguments to func
groups	grouping variable, e.g. .wavelength if intensities of more than one wavelength should be plotted
...	further arguments to <a href="#">xyplot</a> .

**Details**

If func is not NULL, the summary characteristic is calculated first by applying func with the respective arguments (in func.args) to each of the spectra. If func returns more than one value (for each spectrum), the different values end up as different wavelengths.

If the wavelength is not used in the model specification nor in groups, nor for specifying subsets, and neither is func given, then only the first wavelength's intensities are plotted and a warning is issued.

The special column names .rownames and .wavelength may be used.

The actual plotting is done by [xyplot](#).

**Author(s)**

C. Beleites

**See Also**[xyplot](#)**Examples**

```
## example 1: calibration of fluorescence
plotc (flu) ## gives a warning

plotc (flu, func = mean)
plotc (flu, func = range, groups = .wavelength)

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

calibration <- lm (spc ~ c, data = flu[, ,450]$.)
summary (calibration)
plotc (flu [, , 450], type = c("p", "r"))

conc <- list (c = seq (from = 0.04, to = 0.31, by = 0.01))
ci <- predict (calibration, newdata = conc, interval = "confidence", level = 0.999)

panel.ci <- function (x, y, ...,
conc, ci.lwr, ci.upr, ci.col = "#606060") {
panel.xyplot (x, y, ...)
panel.lmline (x, y,...)
panel.lines (conc, ci.lwr, col = ci.col)
panel.lines (conc, ci.upr, col = ci.col)
}

plotc (flu [, , 450], panel = panel.ci,
conc = conc$c, ci.lwr = ci [, 2], ci.upr = ci [, 3])

## example 2: time-trace of laser emission modes
cols <- c ("black", "blue", "#008000", "red")
wl <- i2wl (laser, c(13, 17, 21, 23))

plotspc (laser, axis.args=list (x = list (at = seq (404.5, 405.8, .1))))
for (i in seq_along (wl))
abline (v = wl[i], col = cols[i], lwd = 2)

plotc (laser [, , wl], spc ~ t, groups = .wavelength, type = "b",
col = cols)
```

plotspc

*Plotting Spectra...***Description**

Plotting Spectra Plot the spectra of a hyperSpec object, i.e. intensity over wavelength. Instead of the intensity values of the spectra matrix summary values calculated from these may be used.

**Usage**

```
plotspc(object, wl.range=NULL, wl.index=FALSE, wl.reverse=FALSE, spc.nmax=50,
        func=NULL, func.args=list(), stacked=NULL, stacked.args=list(),
        add=FALSE, bty="l", plot.args=list(), col="black",
        lines.args=list(), xoffset=0, yoffset=0, nxticks=10,
        axis.args=list(), break.args=list(), title.args=list(), fill=NULL,
        fill.col=NULL, border=NA, polygon.args=list(), zeroline=list(lty =
        2, col = col))
```

```
stacked.offsets(x, stacked=TRUE, min.zero=FALSE, add.factor=0.05, add.sum=0, .spc=NULL)
```

**Arguments**

object	the hyperSpec object
wl.range	the wavelength range to be plotted. Either a numeric vector or a list of vectors with different wavelength ranges to be plotted separately. The values can be either wavelengths or wavelength indices (according to <code>wl.index</code> ).
wl.index	if TRUE, <code>wl.range</code> is considered to give column indices into the spectra matrix. Defaults to specifying wavelength values rather than indices.
wl.reverse	if TRUE, the wavelength axis is plotted backwards.
spc.nmax	maximal number of spectra to be plotted (to avoid accidentally plotting of large numbers of spectra).
func	a function to apply to each wavelength in order to calculate summary spectra such as mean, min, max, etc.
func.args	list with further arguments for <code>func</code>
add	if TRUE, the output is added to the existing plot
bty	see <a href="#">par</a>
col	see <a href="#">par</a> . <code>col</code> might be a vector giving individual colors for the spectra.
xoffset	vector with abscissa offsets for each of the <code>wl.ranges</code> . If it has one element less than there are <code>wl.ranges</code> , 0 is padded at the beginning. The values are interpreted as the distance along the wavelength axis that the following parts of the spectra are shifted towards the origin. E.g. if <code>wl.range = list(600 ~ 1800, 2800 ~ 3200)</code> , <code>xoffset = 750</code> would result in a reasonable plot. See also the examples.

yoffset	ordinate offset values for the spectra. May be offsets to stack the spectra ( <a href="#">stacked.offsets</a> ). Either one for all spectra, one per spectrum or one per group in stacked.
nxticks	hint how many tick marks the abscissa should have.
stacked	if not NULL, a "stacked" plot is produced, see the example. stacked may be TRUE to stack single spectra. A numeric or factor is interpreted as giving the grouping, character is interpreted as the name of the extra data column that holds the groups.
stacked.args	a list with further arguments to <a href="#">stacked.offsets</a> .
fill	if not NULL, the area between the specified spectra is filled with color col. The grouping can be given as factor or numeric, or as a character with the name of the extra data column to use. If a group contains more than 2 spectra, the first and the last are used. If TRUE spectra n and nrow (spc) - n build a group.
fill.col	character vector with fill color. Defaults to brightened colors from col.
border	character vector with border color. You will need to set the line color col to NA in order see the effect.
plot.args	list with further arguments to <a href="#">plot</a>
axis.args	list with further arguments for <a href="#">axis</a> . axis.args\$x should contain arguments for plotting the abscissa, axis.args\$y those for the ordinate (again as lists).
title.args	list with further arguments to <a href="#">title</a> . title.args may contain two lists, \$x, and \$y to set parameters individually for each axis.
lines.args	list with further arguments to <a href="#">lines</a> . lines.args\$type defaults to "l".
break.args	list with arguments for <a href="#">axis.break</a> .
polygon.args	list with further arguments to <a href="#">polygon</a> which draws the filled areas.
zeroline	NA or a list with arguments <a href="#">abline</a> , used to plot line (s) marking I = 0. NA suppresses plotting of the line. The line is by default turned off if yoffset is not 0.
x	a hyperSpec object
min.zero	if TRUE, the lesser of zero and the minimum intensity of the spectrum is used as minimum.
add.factor, add.sum	proportion and absolute amount of space that should be added.
.spc	for internal use. If given, the ranges are evaluated on .spc. However, this may change in future.

## Details

plotspc: This is hyperSpec's main plotting function for spectra plots.

New plots are created by [plot](#), but the abscissa and ordinate are drawn separately by [axis](#). Also, [title](#) is called explicitly to set up titles and axis labels. This allows fine-grained customization of the plots.

If package `plotrix` is available, its function `axis.break` is used to produce break marks for cut wavelength axes.

`stacked.offsets`: Usually, the `stacked` argument of `plotspc` will do fine, but if you need fine control over the stacking, you may calculate the y offsets yourself.

Empty levels of the stacking factor are dropped (as no stacking offset can be calculated in that case.)

### Value

`plotspc`: `plotspc` invisibly returns a list with

<code>x</code>	the abscissa coordinates of the plotted spectral data points
<code>y</code>	a matrix the ordinate coordinates of the plotted spectral data points
<code>wavelengths</code>	the wavelengths of the plotted spectral data points

This can be used together with `spc.identify`.

`stacked.offsets`: a list containing

<code>offsets</code>	numeric with the yoffset for each group in <code>stacked</code>
<code>groups</code>	numeric with the group number for each spectrum
<code>levels</code>	if <code>stacked</code> is a factor, the levels of the groups

### Author(s)

C. Beleites

### See Also

`plot`, `axis`, `title`, `lines`, `polygon`, `par` for the description of the respective arguments.

`axis.break` for cut marks

See `plot` for some predefined spectra plots such as mean spectrum +/- one standard deviation and the like.

`identify` and `locator` about interaction with `plots.plotspc`

### Examples

```
plotspc (flu)

## artificial example to show wavelength axis cutting
plotspc (chondro [sample (nrow (chondro), 50)],
        wl.range = list (600 ~ 650, 1000 ~ 1100, 1600 ~ 1700),
        xoffset = c (0, 300, 450))

plotspc (chondro [sample (nrow (chondro), 50)],
        wl.range = list (600 ~ 650, 1000 ~ 1100, 1600 ~ 1700),
        xoffset = c (300, 450))

## some journals publish Raman spectra backwards
plotspc (chondro [sample (nrow (chondro), 50)], wl.reverse = TRUE)
```

```

plotspc (laser[(0:4)*20+1,,], stacked = TRUE)

plotspc (laser, func = mean_pm_sd,
col = c(NA, "red", "black"), lines.args = list (lwd = 2),
fill = c (1, NA, 1),
fill.col = "yellow", border = "blue",
polygon.args = list (lty = 2, lwd = 4),
title.args = list (xlab = expression (lambda[emission] / nm),
y = list(line = 3.4),
col.lab = "darkgreen"),
axis.args = list (x = list (col = "magenta"), y = list (las = 1))
)

mean.pm.sd <- aggregate (chondro, chondro$clusters, mean_pm_sd)
plot (mean.pm.sd, col = matlab.palette (3), fill = ".aggregate", stacked = ".aggregate")

mean.pm.sd <- aggregate (chondro, chondro$clusters, mean_pm_sd)

offset <- stacked.offsets (mean.pm.sd, ".aggregate")
plot (mean.pm.sd, fill.col = matlab.palette (3), fill = ".aggregate",
stacked = ".aggregate")

plot (aggregate (chondro, chondro$clusters, mean), yoffset = offset$offsets,
lines.args = list (lty = 2, lwd = 2), add = TRUE)

barb <- do.call (collapse, barbiturates [1:3])
plot (barb, lines.args = list (type = "h"), stacked = TRUE,
stacked.args = list (add.factor = .2))

```

---

qplotc

*Spectra plotting with ggplot2*


---

## Description

Spectra plotting with ggplot2

## Usage

```
qplotc(object, mapping=aes_string(x = "c", y = "spc"), ..., func=NULL,
func.args=list())
```

## Arguments

object	hyperSpec object
mapping	see <a href="#">geom_point</a>
...	handed to <a href="#">geom_point</a>
func	function to summarize the wavelengths, if NULL, only the first wavelength is used
func.args	arguments to func

**Details**

These functions are still experimental and may change substantially in future.

**Value**

a `ggplot` object

**Author(s)**

Claudia Beleites

**See Also**

[plotc](#)

[ggplotgeom\\_point](#)

**Examples**

```
qplotc (flu)
qplotc (flu) + geom_smooth (method = "lm")
```

---

qplotmap

*Spectra plotting with ggplot2*

---

**Description**

Spectra plotting with ggplot2

**Usage**

```
qplotmap(object, mapping=aes_string(x = "x", y = "y", fill = "spc"), ...,
         func=mean, func.args=list())
```

**Arguments**

object	hyperSpec object
mapping	see <a href="#">geom_tile</a>
...	handed to <a href="#">geom_tile</a>
func	function to summarize the wavelengths
func.args	arguments to func

**Details**

These functions are still experimental and may change substantially in future.

**Value**

a [ggplot](#) object

**Author(s)**

Claudia Beleites

**See Also**

[plotmap](#)

[ggplotgeom\\_tile](#)

**Examples**

```
qplotmap (chondro)
```

---

qplotspc

*Spectra plotting with ggplot2*

---

**Description**

Spectra plotting with ggplot2

**Usage**

```
qplotspc(x, wl.range, ..., mapping=aes_string(x = ".wavelength", y = "spc",  
group = ".rownames"), spc.nmax=10)
```

**Arguments**

x	hyperSpec object
wl.range	wavelength ranges to plot
...	handed to <a href="#">geom_line</a>
mapping	see <a href="#">geom_line</a>
spc.nmax	maximum number of spectra to plot

**Details**

These functions are still experimental and may change substantially in future.

**Value**

a [ggplot](#) object

**Author(s)**

Claudia Beleites

**See Also**

[plotspc](#)  
[ggplotgeom\\_line](#)

**Examples**

```
plotspc (chondro)
plotspc (paracetamol, c (2800 ~ max, min ~ 1800)) + scale_x_reverse (breaks = seq (0, 3200, 400))

plotspc (aggregate (chondro, chondro$clusters, mean),
mapping = aes (x = .wavelength, y = spc, colour = clusters)) +
facet_grid (clusters ~ .)

plotspc (aggregate (chondro, chondro$clusters, mean_pm_sd),
mapping = aes (x = .wavelength, y = spc, colour = clusters, group = .rownames)) +
facet_grid (clusters ~ .)
```

---

rbind-fill

*Bind matrices by row, and fill missing columns with NA...*


---

**Description**

Bind matrices by row, and fill missing columns with NA

**Usage**

```
rbind.fill.matrix(...)
```

```
rbind.fill.matrix(...)
```

```
rbind.fill(...)
```

**Arguments**

... data frames to row bind together

**Details**

`rbind.fill.matrix`: The matrices are bound together using their column names or the column indices (in that order of precedence.) Numeric columns may be converted to character beforehand, e.g. using `format`. If a matrix doesn't have `colnames`, the column number is used (via `make.names(unique = TRUE)`).

Note that this means that a column with name "X1" is merged with the first column of a matrix without name and so on.

Vectors are converted to 1-column matrices prior to `rbind`.

Matrices of factors are not supported. (They are anyways quite inconvenient.) You may convert them first to either numeric or character matrices. If a character matrix is merged with a numeric, the result will be character.

Row names are ignored.

The return matrix will always have column names.

`rbind.fill.matrix`: The matrices are bound together using their column names or the column indices (in that order of precedence.) Numeric columns may be converted to character beforehand, e.g. using `format`. If a matrix doesn't have `colnames`, the column number is used (via `make.names(unique = TRUE)`).

Note that this means that a column with name "X1" is merged with the first column of a matrix without name and so on.

Vectors are converted to 1-column matrices prior to `rbind`.

Matrices of factors are not supported. (They are anyways quite inconvenient.) You may convert them first to either numeric or character matrices. If a character matrix is merged with a numeric, the result will be character.

Row names are ignored.

The return matrix will always have column names.

`rbind.fill`: This is an enhancement to `rbind` which adds in columns that are not present in all inputs, accepts a list of data frames, and operates substantially faster

## Value

`rbind.fill.matrix`: a matrix

`rbind.fill.matrix`: a matrix

## Author(s)

C. Beleites

## See Also

[rbind](#), [cbind](#), [rbind.fill](#), [rbind](#), [cbind](#), [rbind.fill](#)

## Examples

```
A <- matrix (1:4, 2)
B <- matrix (6:11, 2)
A
B
rbind.fill.matrix (A, B)

colnames (A) <- c (3, 1)
A
rbind.fill.matrix (A, B)

rbind.fill.matrix (A, 99)
```

```

A <- matrix (1:4, 2)
B <- matrix (6:11, 2)
A
B
rbind.fill.matrix (A, B)

colnames (A) <- c (3, 1)
A
rbind.fill.matrix (A, B)

rbind.fill.matrix (A, 99)

#' rbind.fill(mtcars[c("mpg", "wt")], mtcars[c("wt", "cyl")])

```

---

read-spc

*read.spc.KaiserMap*


---

## Description

`read.spc.KaiserMap` imports sets of `.spc` files of Raman maps written by Kaiser Optical Systems' Hologram software. It may also serve as an example how to write wrapper functions for `read.spc` to conveniently import specialized sets of `.spc` files.

## Usage

```
read.spc.KaiserMap(files, keys.hdr2data=FALSE, keys.hdr2log=TRUE, keys.log2data=NULL,
  keys.log2log=TRUE, glob=TRUE, ...)
```

```
read.spc(filename, keys.hdr2data=c("fexper", "fres", "fsource"),
  keys.hdr2log=c("fdate", "fpeakpt"), keys.log2data=FALSE,
  keys.log2log=TRUE, log.txt=TRUE, log.bin=FALSE, log.disk=FALSE,
  hdr=list(), no.object=FALSE)
```

## Arguments

<code>glob</code>	If <code>TRUE</code> the filename is interpreted as a wildcard containing file name pattern and expanded to all matching file names.
<code>...</code>	All further arguments to <code>read.spc.KaiserMap</code> are handed over directly to <code>read.spc</code> .
<code>filename</code>	The complete file name of the <code>.spc</code> file.
<code>files</code>	If <code>glob = TRUE</code> , <code>filename</code> can contain wildcards. Thus all files matching the name pattern in <code>filename</code> can be specified.
<code>keys.hdr2data</code> , <code>keys.hdr2log</code> , <code>keys.log2data</code> , <code>keys.log2log</code>	character vectors with the names of parameters in the <code>.spc</code> file's log block ( <code>log2xxx</code> ) or header ( <code>hdr2xxx</code> ) that should go into the extra data ( <code>yyy2data</code> ) or into the long.description field of the returned <code>hyperSpec</code> object's log ( <code>yyy2log</code> ). All header fields specified in the <code>.spc</code> file format specification (see below) are imported and can be referred to by their de-capitalized names.

log.txt	Should the text part of the .spc file's log block be read?
log.bin, log.disk	Should the normal and on-disk binary parts of the .spc file's log block be read? If so, they will be put as raw vectors into the hyperSpec object's log.
hdr	A list with fileheader fields that overwrite the settings of actual file's header. Use with care, and look into the source code for detailed insight on the elements of this list.
no.object	If TRUE, a list with wavelengths, spectra, labels, log and data are returned instead of a hyperSpec object. This parameter will likely be subject to change in future - use with care.

### Value

read.spc: If the file contains multiple spectra with individual wavelength axes, read.spc returns a list of hyperSpec objects. Otherwise the result is a hyperSpec object.

read.spc.KaiserMap returns a hyperSpec object with data columns x, y, and z containing the stage position as recorded in the .spc files' log.

### Note

Only a restricted set of test files was available for development. Particularly, the w-planes feature could not be tested.

If you have .spc files that cannot be read with these function, don't hesitate to contact the package maintainer with your code patch or asking advice.

### Author(s)

C. Beleites

### References

Source development kit and file format specification of .spc files: <https://ftirsearch.com/features/converters/SPCfileFormat.htm>.

### See Also

[textio](#)

### Examples

```
## get the sample .spc files from ftirsearch.com (see above)
## Not run:
# single spectrum
spc <- read.spc ("BENZENE.SPC")
plot (spc)

# multi-spectra .spc file with common wavelength axis
spc <- read.spc ('IG_MULTI.SPC')
spc
```

```
# multi-spectra .spc file with individual wavelength axes
spc <- read.spc ("BARBITUATES.SPC")
plot (spc [[1]], lines.args = list (type = "h"))

## End(Not run)
```

---

readENVI

*read.ENVI.Nicolet*


---

## Description

Nicolet uses some more keywords in their header file. `read.ENVI.Nicolet` therefore appends "description", "z plot titles", and "pixel size" to `keys.hdr2log` before calling `read.ENVI`. They are then interpreted as follows:

description	giving the position of the first spectrum
z plot titles	wavelength and intensity axis units, comma separated
pixel size	interpreted as x and y step size

## Usage

```
read.ENVI.Nicolet(..., x=NA, y=NA, log=list(), keys.hdr2log=TRUE,
  nicolet.correction=FALSE)
```

```
read.ENVI(file=stop("read.ENVI: file name needed"), headerfile=NULL,
  header=list(), keys.hdr2data=FALSE, keys.hdr2log=TRUE, x=0:1, y=x,
  wavelength=NULL, label=list(), log=list())
```

## Arguments

<code>nicolet.correction</code>	see details
<code>...</code>	handed to <code>read.ENVI</code>
<code>file</code>	complete name of the binary file
<code>headerfile</code>	name of the ASCII header file. If NULL, the name of the header file is guessed by looking for a second file with the same basename but different suffix as <code>file</code> .
<code>header</code>	list with the respective information, see details.
<code>x,y</code>	vectors of form <code>c(offset, step size)</code> for the position vectors, see details.
<code>wavelength,label,log</code>	lists that overwrite the respective information from the ENVI header file. These data is then handed to <code>initialize</code>
<code>keys.hdr2data,keys.hdr2log</code>	determine which fields of the header file should be put into the extra data and which are to be stored in <code>long.description\$header</code> in the <code>hyperSpec</code> object's

log. Defaults to putting all header information into the log, and none as extra data.

To specify certain entries, give character vectors containing the lowercase names of the header file entries.

## Details

`read.ENVI.Nicolet`: The values in header line description seem to be microns while the pixel size seems to be in microns. If `nicolet.correction` is true, the pixel size values (i.e. the step sizes) are multiplied by 1000.

`read.ENVI`: `read.ENVI.Nicolet` should be a good starting point for writing custom wrappers for `read.ENVI` that take into account your manufacturer's special entries in the header file.

ENVI data usually consists of two files, an ASCII header and a binary data file. The header contains all information necessary for correctly reading the binary file.

I experienced missing header files (or rather: header files without any contents) produced by Bruker Opus' ENVI export.

In this case the necessary information can be given as a list in parameter header instead. The elements of header are then:

header\$	values	meaning
samples	integer	no of columns / spectra in x direction
lines	integer	no of lines / spectra in y direction
bands	integer	no of wavelengths / data points per spectrum
'data type'		format of the binary file
	1	1 byte unsigned integer
	2	2 byte signed integer
	3	4 byte signed integer
	4	4 byte float
	5	8 byte double
	9	16 (2 x 8) byte complex double
	12	2 byte unsigned integer
'header offset'	integer	number of bytes to skip before binary data starts
interleave		directions of the data cube
	"BSQ"	band sequential (indexing: [sample, line, band])
	"BIL"	band interleave by line (indexing: [sample, line, band])
	"BIP"	band interleave by pixel (indexing: [band, line, sample])
'byte order'	0 or "little"	little endian
	1 or "big"	big endian
	"swap"	swap byte order

Some more information that is not provided by the ENVI files may be given:

Wavelength axis and axis labels in the respective parameters. For more information, see [initialize](#).

The spatial information is by default a sequence from 0 to `header$samples - 1` and `header$lines - 1`, respectively. `x` and `y` give offset of the first spectrum and step size.

Thus, the object's `$x` column is:  $(0 : \text{header}\$samples - 1) * x [2] + x [1]$ . The `$y` column is calculated analogously.

**Value**

read.ENVI: a hyperSpec object

**Author(s)**

C. Beleites, testing for the Nicolet files C. Dicko

**References**

This function was adapted from [read.ENVI](#):

Jarek Tuszynski (2008). caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc.. R package version 1.9.

**See Also**

[read.ENVI](#)

[textio](#)

---

sample

*Random Samples and Permutations...*

---

**Description**

Random Samples and Permutations Take a sample of the specified size from the elements of x with or without replacement.

**Usage**

```
## S4 method for signature 'hyperSpec'  
sample(x, size=nrow(x), replace=FALSE, prob=NULL, short="sample", user=NULL,  
       date=NULL)
```

```
isample(x, size=nrow(x), replace=FALSE, prob=NULL)
```

```
## S4 method for signature 'data.frame'  
sample(x, size=nrow(x), replace=FALSE, prob=NULL, drop=FALSE)
```

```
## S4 method for signature 'matrix'  
sample(x, size=nrow(x), replace=FALSE, prob=NULL, drop=FALSE)
```

**Arguments**

x	The hyperSpec object, data.frame or matrix to sample from
size	positive integer giving the number of spectra (rows) to choose.
replace	Should sampling be with replacement?
prob	A vector of probability weights for obtaining the elements of the vector being sampled.
short	date are handed to <a href="#">logentry</a>
user	date are handed to <a href="#">logentry</a>
date	are handed to <a href="#">logentry</a>
drop	see <a href="#">drop</a> : by default, do not drop dimensions of the result

**Value**

sample, hyperSpec-method: a hyperSpec object, data.frame or matrix with size rows for sample, and an integer vector for isample that is suitable for indexing (into the spectra) of x.

isample: vector with indices suitable for row-indexing x

**Author(s)**

C. Beleites

**See Also**

[sample](#)

**Examples**

```
sample (flu, 3)

plot (flu, col = "darkgray")
plot (sample (flu, 3), col = "red", add = TRUE)

plot (flu, col = "darkgray")
plot (sample (flu, 3, replace = TRUE), col = "#0000FF80", add = TRUE,
lines.args = list (lwd = 2));

isample (flu, 3)
isample (flu, 3, replace = TRUE)
isample (flu, 8, replace = TRUE)

sample (cars, 2)

sample (matrix (1:24, 6), 2)
```

---

scan-dot-txt-dot-Witec

*File Import Witec Raman*

---

## Description

Import Raman Spectra/Maps from Witec Instrument via ASCII files

## Usage

```
scan.txt.Witec(filex=stop("filename needed"), filey=sub("-x", "-y", filex),
  points.per.line=NULL, lines.per.image=NULL, short="scan.txt.Witec",
  user=NULL, date=NULL)
```

## Arguments

filex	filename wavelength axis file
filey	filename intensity file
points.per.line	number of spectra in x direction of the map
lines.per.image	number of spectra in y direction
short,user,date	handed to <a href="#">logentry</a>

## Value

a hyperSpec object

## Author(s)

Claudia Beleites

## See Also

vignette ("fileio") for more information on file import.

---

```
scan-txt-Renishaw    import Raman measurements from Renishaw ASCII-files...
```

---

## Description

import Raman measurements from Renishaw ASCII-files import Raman measurements from Renishaw (possibly compressed) .txt file.

## Usage

```
scan.txt.Renishaw(file=stop("file is required"), data="xyspc", nlines=0, nspc=NULL,
  short="scan.txt.Renishaw", user=NULL, date=NULL)
```

```
scan.zip.Renishaw(file=stop("filename is required"), txt.file=sub("[.]zip", ".txt",
  basename(file)), ...)
```

## Arguments

file	file name or connection
data	type of file, one of "spc", "xyspc", "zspc", "depth", "ts", see details.
nlines	number of lines to read in each chunk, if 0 or less read whole file at once. nlines must cover at least one complete spectrum,i.e. nlines must be at least the number of data points per spectrum. Reasonable values start at 1e6.
nspc	number of spectra in the file
...	Arguments for scan.txt.Renishaw
short,user,date	passed to logentry
txt.file	name of the .txt file in the .zip archive. Defaults to zip file's name with suffix .txt instead of .zip

## Details

scan.txt.Renishaw: The file may be of any file type that can be read by [gzfile](#) (i.e. text, or zipped by gzip, bzip2, xz or lzma). .zip zipped files need to be read using scan.zip.Renishaw.

Renishaw .wxd files are converted to .txt ASCII files by their batch converter. They come in a "long" format with columns (y x | time | z)? wavelength intensity. The first columns depend on the data type.

The corresponding possibilities for the data argument are:

data	columns	
"spc"	wl int	single spectrum
"zspc", "depth"	z wl int	depth profile
"ts"	t wl int	time series
"xyspc"	y x wl int	2d map

This function allows reading very large ASCII files, but it does not work on files with missing values (NAs are allowed).

If the file is so large that it could be read in chunks and `nspc` is not given, `scan.txt.Renishaw` tries to guess it by using `wc` (if installed).

### Value

`scan.txt.Renishaw`: the `hyperSpec` object

### Author(s)

C. Beleites

### See Also

[read.txt.long](#), [read.txt.wide](#), [scan](#)

---

seq

*Sequence generation along spectra or wavelengths...*

---

### Description

Sequence generation along spectra or wavelengths This function generates sequences along the spectra (rows) or wavelengths of `hyperSpec` objects.

### Usage

```
seq.hyperSpec(x, from=1, to=nrow(x), ..., index=FALSE, short="seq", user=NULL,
             date=NULL)
```

### Arguments

<code>x</code>	the <code>hyperSpec</code> object
<code>from, to</code>	arguments handed to <a href="#">seq.int</a>
<code>...</code>	arguments for <a href="#">seq</a> , namely <code>by</code> , <code>length.out</code>
<code>index</code>	should a vector with indices be returned rather than a <code>hyperSpec</code> object?
<code>short, user, date</code>	handed to <a href="#">logentry</a> .

### Details

Note that [wl2i](#) generates sequences of indices along the wavelength axis.

`seq` had to be implemented as S3 method as the generic has only `...` arguments (on which no dispatch with differing types is possible).

[seq\\_along](#) is not generic, but returns a sequence of the length of the object. As `hyperSpec` provides a Method [length](#), it can be used. The result is a sequence of indices for the spectra.

**Value**

a numeric or hyperSpec object, depending on index.

**Author(s)**

C. Beleites

**See Also**

[wl2i](#) to construct sequences of wavelength indices.

[seq](#)

**Examples**

```
seq (flu, index = TRUE)
seq_along (flu)
seq (flu, length.out = 3, index = TRUE) # return value is numeric, not integer!
seq (flu, by = 2, index = TRUE)      # return value is numeric, not integer!

plot (flu, col = "darkgray")
plot (seq (flu, by = 2), add = TRUE, col= "red")
plot (seq (flu, length.out = 2), add = TRUE, col= "blue")
```

---

show

*as.character,hyperSpec-method*

---

**Description**

Convert a hyperSpec object to character strings for Display print, show, and summary show the result of as.character.

**Usage**

```
## S4 method for signature 'hyperSpec'
as.character(x, digits=getOption("digits"), range=TRUE, max.print=5, shorten.to=c(2,
  1), log=TRUE)

## S4 method for signature 'hyperSpec'
show(object)

## S4 method for signature 'hyperSpec'
print(x, log=FALSE, range=FALSE, ...)

## S4 method for signature 'hyperSpec'
summary(object, log=TRUE, ...)
```

**Arguments**

<code>digits</code>	number of digits handed over to format
<code>range</code>	should the values be indicated as range rather than first and last elements?
<code>max.print</code>	maximum number of elements to be printed (of a variable)
<code>shorten.to</code>	if a vector is longer than <code>max.print</code> , only the first <code>shorten.to[1]</code> and the last <code>shorten.to[2]</code> elements are printed
<code>object</code>	a hyperSpec object
<code>log</code>	should the log be printed?
<code>x</code>	a hyperSpec object
<code>...</code>	print and summary hand further arguments to <code>as.character</code>

**Details**

`show`, hyperSpec-method: `print`, `show`, and `summary` differ only in the defaults. `show` displays the range of values instead,

**Value**

`as.character`, hyperSpec-method: `as.character` returns a character vector fit to be printed by `cat` with `sep = "\n"`.

`print`, hyperSpec-method: `print invisibly` returns `x` after printing, `show` returns an invisible `NULL`.

**See Also**

[as.charactershowprintsummary](#)

**Examples**

```
chondro
```

```
show (chondro)
```

```
summary (chondro)
```

```
print (chondro, log = TRUE)
print (chondro, range = TRUE)
```

```
logbook (chondro)
```

---

spc-bin                      *Wavelength Binning...*

---

### Description

Wavelength Binning In order to reduce the spectral resolution and thus gain signal to noise ratio or to reduce the dimensionality of the spectral data set, the spectral resolution can be reduced.

### Usage

```
spc.bin(spc, by=stop("reduction factor needed"), na.rm=TRUE, ...,
        short="spc.bin", user=NULL, date=NULL)
```

### Arguments

spc	the hyperSpec object
by	reduction factor
na.rm	decides about the treatment of NAs: if FALSE or 0, the binning is done using na.rm = FALSE if TRUE or 1, the binning is done using na.rm = TRUE if 2, the binning is done using na.rm = FALSE, and resulting NAs are corrected with <code>mean(...{ }, na.rm = TRUE)</code> .
...	ignored
short,user,date	handed to <a href="#">logentry</a> .

### Details

The mean of every by data points in the spectra is calculated.

Using na.rm = TRUE always takes about twice as long as na.rm = FALSE.

If the spectra matrix does not contain too many NAs, na.rm = 2 is faster than na.rm = TRUE.

### Value

A hyperSpec object with `ceiling (nwl (spc) / by)` data points per spectrum.

### Author(s)

C. Beleites

## Examples

```
spc <- spc.bin (flu, 5)

plot (flu[1,,425:475])
plot (spc[1,,425:475], add = TRUE, col = "blue")

nwl (flu)
nwl (spc)
```

---

spc-identify

*Identifying Spectra and Spectral Data Points...*

---

## Description

Identifying Spectra and Spectral Data Points This function allows to identify the spectrum and the wavelength of a point in a plot produced by [plotspc](#).

## Usage

```
spc.identify(x, y=NULL, wavelengths=NULL, ispc=NULL, tol.wl=diff(range(x))/200,
  tol.spc=diff(range(y))/50, point.fn=spc.point.max,
  formatter=spc.label.default, ..., cex=0.7, adj=c(0, 0.5), srt=90,
  warn=TRUE)
```

```
spc.point.max(wl, spc, wlclick)
```

```
spc.point.default(wl, spc, wlclick)
```

```
spc.point.min(wl, spc, wlclick)
```

```
spc.point.sqr(wl, spc, wlclick, delta=1)
```

```
spc.label.default(ispc, wl, spc, digits=3)
```

```
spc.label.wlonly(ispc, wl, spc, digits=3)
```

## Arguments

x either the abscissa coordinates or the list returned by [plotspc](#)  
y the ordinate values. Giving y will override any values from x\$y.

wavelengths	the wavelengths for the data points. Giving wavelengths will override any values from x\$wavelengths.
tol.wl, tol.spc	tolerance in wavelength and spectral intensity to search around the clicked point. See details.
point.fn	function (wl, spc, wlclick) to determine the actual point to label, see details.
formatter	function (i, wl, spc) that produces the labels. If NULL, no labels are displayed.
...	passed to <code>text</code> in order to produce the labels
cex, adj, srt	see <code>par</code>
warn	Should the user be warned if no point is in the considered window? In addition, see the discussion of option <code>debugLevel</code> in the details. If FALSE, the resulting data.frame will have a row of NAs instead.
delta	<code>spc.point.sqr</code> fits the parabola in the window <code>wlclick ± delta</code> points.
wl	the wavelength to label
spc	the intensity to label
wlclick	the clicked wavelength
ispc	if a selection of spectra was plotted, their indices can be given in <code>ispc</code> . In this case <code>ispc [i]</code> is returned rather than <code>i</code> .
digits	how many digits of the wavelength should be displayed?

## Details

`spc.identify`: This function first finds the spectrum with a point closest to the clicked position (see `locator`). The distance to the clicked point is evaluated relative to the size of the tolerance window.

In a second step, `max.fn` searches for the actual point to label within the specified wavelength window of that spectrum. This allows to label maxima (or minima) without demanding too precise clicks. Currently, the following functions to determine the precise point:

<code>spc.point.default</code>	uses the clicked wavelength together with its spectral intensity
<code>spc.point.max</code>	the point with the highest intensity in the wavelength window
<code>spc.point.min</code>	the point with the lowest intensity in the wavelength window
<code>spc.point.sqr</code>	maximum of a parabola fit through the point with highest intensity and the two surrounding points

`point.fn` is called with the arguments `wl` containing the considered wavelength window, `spc` the respective intensities of the closest spectrum, and `wlclick` the wavelength that was clicked. They return a vector of two elements (wavelength and intensity).

As a last step, a label for the point produced by `formatter` and plotted using `text`. Currently, the following formatters are available:

<code>spc.label.default</code>	spectrum number, wavelength
<code>spc.label.wlonly</code>	wavelength

formatter functions receive the number of the spectrum `ispc`, the wavelength `wl`, and the spectral intensity `spc` and produce a character variable suitable for labelling. The predefined formatters surround the label text by spaces in order to easily have an appropriate offset from the point of the spectrum.

The warning issued if no spectral point is inside the tolerance window may be switched off by `warn = FALSE`. In that case, the click will produce a row of NAs in the resulting `data.frame`.

`spc.identify` uses option `debuglevel` to determine whether debugging output should be produced. `debuglevel == 2` will plot the tolerance window for every clicked point, `debuglevel == 1` will plot the tolerance window only if no data point was inside. See [hyperSpec options](#) for details about retrieving and setting options.

You may want to adjust the plot's `ylim` to ensure that the labels are not clipped. As a dirty shortcut, `xpd = NA` may help.

### Value

`spc.identify`: a `data.frame` with columns

<code>ispc</code>	spectra indices of the identified points, i.e. the rows of the <code>hyperSpec</code> object that was plotted. If <code>ispc</code> is given, <code>ispc [i]</code> is returned rather than <code>i</code> .
<code>wavelengths</code>	the wavelengths of the identified points
<code>spc</code>	the intensities of the identified points

### Author(s)

C. Beleites

### See Also

[locator](#), [plotspc](#), [hyperSpec options](#)  
[map.identify](#) [map.sel.poly](#)

### Examples

```
ispc <- sample (nrow (laser), 10)
ispc

identified <- spc.identify (plotspc (laser[ispc]))
## convert to the "real" spectra indices
ispc [identified$ispc]
identified$wl
identified$spc

## allow the labels to be plotted into the plot margin
spc.identify (plotspc (laser[ispc]), ispc = ispc, xpd = NA)
```

```

spc.identify (plotspc (paracetamol, xoffset = 1100,
wl.range = c (600 ~ 1700, 2900 ~ 3150)),
formatter = spc.label.wlonly)

## looking for minima
spc.identify (plot (-paracetamol, wl.reverse = TRUE),
point.fn = spc.point.min, adj = c (1, 0.5))

```

---

spc-loess

*loess smoothing interpolation for spectra...*


---

### Description

loess smoothing interpolation for spectra Spectra can be smoothed and interpolated on a new wavelength axis using [loess](#).

### Usage

```

spc.loess(spc, newx, enp.target=nwl(spc)/4, surface="direct", ...,
short="spc.loess", user=NULL, date=NULL)

```

### Arguments

spc	the hyperSpec object
newx	wavelength axis to interpolate on
enp.target, surface, ...	parameters for <a href="#">loess</a> and <a href="#">loess.control</a> .
short, user, date	handed to <a href="#">logentry</a> .

### Details

Applying [loess](#) to each of the spectra, an interpolation onto a new wavelength axis is performed. At the same time, the spectra are smoothed in order to increase the signal : noise ratio. See [loess](#) and [loess.control](#) on the parameters that control the amount of smoothing.

### Value

a new hyperspec object.

### Author(s)

C. Beleites

**See Also**

[loess](#), [loess.control](#)

**Examples**

```
plot (flu, col = "darkgray")
plot (spc.loess(flu, seq (420, 470, 5)), add = TRUE, col = "red")

flu [[3, ]] <- NA_real_
smooth <- spc.loess(flu, seq (420, 470, 5))
smooth [[, ]]
plot (smooth, add = TRUE, col = "blue")
```

---

spc.NA.linapprox      *Impute missing data points...*

---

**Description**

Impute missing data points

**Usage**

```
spc.NA.linapprox(spc, neighbours=1, ..., short="spc.NA.linapprox", user=NULL, date=NULL)
```

**Arguments**

spc	hyperSpec object with spectra matrix containing NAs
neighbours	how many neighbour data points should be used to fit the line
...	ignored
short,user,date	handed to <a href="#">logentry</a>

**Details**

Replace NAs in the spectra matrix by linear interpolation.

**Value**

hyperSpec object

**Author(s)**

Claudia Beleites

---

split	<i>Split a hyperSpec object according to groups...</i>
-------	--

---

### Description

Split a hyperSpec object according to groups `split` divides the hyperSpec object into a list of hyperSpec objects according to the groups given by `f`.

### Usage

```
## S4 method for signature 'hyperSpec'  
split(x, f, drop=TRUE, short=NULL, user=NULL, date=NULL)
```

### Arguments

<code>x</code>	the hyperSpec object
<code>f</code>	a factor giving the grouping (or a variable that can be converted into a factor by <code>as.factor</code> )
<code>drop</code>	if TRUE, levels off that do not occur are dropped.
<code>short, user, date</code>	handed to <a href="#">logentry</a>

### Details

The hyperSpec objects in the list may be bound together again by `bind("r", list_of_hyperSpec_objects)`.

### Value

A list of hyperSpec objects.

### Author(s)

C. Beleites

### See Also

[split](#)

### Examples

```
dist <- pearson.dist (chondro[[ ]])  
dend <- hclust (dist, method = "ward")  
z <- cutree (dend, h = 0.15)  
  
clusters <- split (chondro, z)  
length (clusters)  
  
# difference in cluster mean spectra  
plot (apply (clusters[[2]], 2, mean) - apply (clusters[[1]], 2, mean))
```

---

summary

*The functions...*

---

## Description

The functions

## Arguments

...	further objects
na.rm	logical indicating whether missing values should be removed
digits	integer stating the rounding precision

## Details

Summary, -method: all, any,  
sum, prod,  
min, max,  
range, and  
is.na  
for hyperSpec objects.

All these functions work on the spectra matrix.

## Value

Summary, -method: sum, prod, min, max, and range return a numeric, all, any, and is.na a logical.

## See Also

[Summary](#) for the base summary functions [all.equal](#) and [isTRUE](#)

## Examples

```
range (flu)
is.na (flu [, , 405 ~ 410]);
```

---

sweep

*Sweep Summary Statistic out of an hyperSpec Object...*


---

**Description**

Sweep Summary Statistic out of an hyperSpec Object [sweep](#) for hyperSpec objects.

**Usage**

```
## S4 method for signature 'hyperSpec'
sweep(x, MARGIN, STATS, FUN="-", check.margin=TRUE, ..., short="sweep",
      user=NULL, date=NULL)
```

**Arguments**

x	a hyperSpec object.
MARGIN	direction of the spectra matrix that STATS goes along.
STATS	the summary statistic to sweep out. Either a vector or a hyperSpec object. hyperSpec offers a non-standard convenience function: if STATS is a function, this function is applied first (with the same MARGIN) to compute the statistic. However, no further arguments to the apply function can be given. See the examples.
FUN	the function to do the sweeping, e.g. '-' or '/'.
check.margin	If TRUE (the default), warn if the length or dimensions of STATS do not match the specified dimensions of x. Set to FALSE for a small speed gain when you <i>know</i> that dimensions match.
...	further arguments for FUN
short,user,date	handed over to <a href="#">logentry</a> .

**Details**

Calls [sweep](#) for the spectra matrix.

sweep is useful for some spectra preprocessing, like offset correction, subtraction of background spectra, and normalization of the spectra.

**Value**

A hyperSpec object.

**Author(s)**

C. Beleites

**See Also**[sweep](#)**Examples**

```
## Subtract the background / slide / blank spectrum
# the example data does not have spectra of the empty slide,
# so instead the overall composition of the sample is subtracted
background <- apply (chondro, 2, quantile, probs = 0.05)
corrected <- sweep (chondro, 2, background, "-")
plot (corrected, "spcprct15")

## Offset correction
offsets <- apply (chondro, 1, min)
corrected <- sweep (chondro, 1, offsets, "-")
plot (corrected, "spcprct15")

## Min-max normalization (on max amide I)
# the minimum is set to zero by the offset correction.
factor <- apply (corrected, 1, max)
mm.corrected <- sweep (corrected, 1, factor, "/")
plot (mm.corrected, "spcprct15")

## convenience: give function to compute STATS:
mm.corrected2 <- sweep (corrected, 1, max, "/")
plot (mm.corrected2)

## checking
stopifnot (all (mm.corrected2 == mm.corrected))
```

---

textio

---

*Import and Export of hyperSpec objects...*


---

**Description**

Import and Export of hyperSpec objects Besides [save](#) and [load](#), two general ways to import and export data into hyperSpec objects exist.

**Usage**

```
read.txt.long(file=stop("file is required"), cols=list(.wavelength =
  expression(lambda/nm), spc = "I / a.u."), header=TRUE, ...)

read.txt.wide(file=stop("file is required"), cols=list(spc = "I / a.u.", .wavelength
  = expression(lambda/nm)), check.names=FALSE, ...)

write.txt.long(object, file=stop("file is required"), order=c(".rownames",
  ".wavelength"), na.last=TRUE, decreasing=FALSE, quote=FALSE,
```

```
sep="\t", row.names=FALSE, cols=NULL, col.names=TRUE,
col.labels=FALSE, append=FALSE, ...)
```

```
write.txt.wide(object, file=stop("file is required"), cols=NULL, quote=FALSE,
  sep="\t", row.names=FALSE, col.names=TRUE, header.lines=1,
  col.labels=if (header.lines == 1) FALSE else TRUE, append=FALSE,
  ...)
```

## Arguments

file	filename or connection
cols	the column names specifying the column order. For data import, a list with elements <code>colname = label</code> ; for export a character vector with the colnames. Use <code>wavelength</code> to specify the wavelengths.
header	the file has (shall have) a header line
...	arguments handed to <code>read.table</code> and <code>write.table</code> , respectively.
decreasing	logical vector giving the sort order
check.names	handed to <code>read.table</code> . Make sure this is FALSE, if the column names of the spectra are the wavelength values.
object	the hyperSpec object
order	which columns should be ordered? order is used as index vector into a <code>data.frame</code> with columns given by <code>cols</code> .
na.last	handed to <code>order</code> by <code>write.txt.long</code> .
quote, sep, row.names, col.names	have their usual meaning (see <code>read.table</code> and <code>write.table</code> ), but different default values.
col.labels	Should the column labels be used rather than the colnames?
append	Should the output be appended to an existing file?
header.lines	Toggle one or two line header (wavelengths in the second header line) for <code>write.txt.wide</code>

## Details

`read.txt.long`: Firstly, hyperSpec objects can be imported and exported as ASCII files.

A second option is using the package `R.matlab` which provides the functions `readMat` and `writeMat`.

hyperSpec comes with a number of pre-defined functions to import manufacturer specific file formats. For details, see vignette ("file-io").

`read.spc` imports Thermo Galactic's .spc file format, and ENVI files may be read using `read.ENVI`.

These functions are very flexible and provide lots of arguments.

If you use them to read or write manufacturer specific ASCII formats, please consider writing a wrapper function and contributing this function to **hyperSpec**. An example is in the "flu" vignette (see vignette ("flu", package = "hyperSpec").

Note that R accepts many packed formats for ASCII files, see [connections](#). For .zip files, see [unzip](#).

For further information, see the examples below and the documentation of [R.matlab](#).

`read.txt.wide`: Besides [save](#) and [load](#), two general ways to import and export data into hyperSpec objects exist.

Firstly, hyperSpec objects can be imported and exported as ASCII files.

A second option is using the package [R.matlab](#) which provides the functions [readMat](#) and [writeMat](#). hyperSpec comes with a number of pre-defined functions to import manufacturer specific file formats. For details, see vignette (`"fileio"`).

[read.spc](#) imports Thermo Galactic's .spc file format, and ENVI files may be read using [read.ENVI](#).

These functions are very flexible and provide lots of arguments.

If you use them to read or write manufacturer specific ASCII formats, please consider writing a wrapper function and contributing this function to **hyperSpec**. An example is in the "flu" vignette (see vignette (`"flu"`, `package = "hyperSpec"`)).

Note that R accepts many packed formats for ASCII files, see [connections](#). For .zip files, see [unzip](#).

For further information, see the examples below, vignette (`"fileio"`) and the documentation of [R.matlab](#).

### Author(s)

C. Beleites

### See Also

[read.table](#) and [write.table](#)

[R.matlab](#) for .mat files

[read.ENVI](#) for ENVI data

[read.spc](#) for .spc files

Manufacturer specific file formats: [scan.txt](#), [Renishaw](#) vignette (`"fileio"`) and <http://hyperspec.r-forge.r-project.org/fileio.pdf>, respectively

### Examples

```
## Not run: vignette ("file-io")

## export & import matlab files
if (require (R.matlab)){
# export to matlab file
writeMat ("test.mat", x = flu[[[]], wavelength = flu@wavelength,
label = lapply (flu@label, as.character))

# reading a matlab file
data <- readMat ("test.mat")
print (data)
mat <- new ("hyperSpec", spc = data$x,
```

```

wavelength = as.numeric(data$wavelength),
label = data$label[, ,1])
}

## ascii export & import

write.txt.long (flu, file = "flu.txt", cols = c(".wavelength", "spc", "c"),
order = c("c", ".wavelength"),
decreasing = c(FALSE, TRUE))

read.txt.long (file = "flu.txt", cols = list (.wavelength = expression (lambda / nm),
spc= "I / a.u", c = expression ("/" (c, (mg/l))))))

write.txt.wide (flu, file = "flu.txt", cols = c("c", "spc"),
col.labels = TRUE, header.lines = 2, row.names = TRUE)

write.txt.wide (flu, file = "flu.txt", col.labels = FALSE, row.names = FALSE)

read.txt.wide (file = "flu.txt",
cols = list (c=expression ("/"("c", "mg/l")), spc="I / a.u", .wavelength = "lambda / nm"),
header = TRUE)

```

---

trellis-dot-factor-dot-key

*Color coding legend for factors...*


---

## Description

Color coding legend for factors Modifies a list of lattice arguments (as for `levelplot`, etc.) according to the factor levels. The colorkey will shows all levels (including unused), and the drawing colors will be set accordingly.

## Usage

```
trellis.factor.key(f, levelplot.args=list())
```

## Arguments

`f` the factor that will be color-coded  
`levelplot.args` a list with levelplot arguments

## Details

`trellis.factor.key` is used during levelplot-based plotting of factors (for hyperSpec objects) unless `transform.factor = FALSE` is specified.

## Value

the modified list with levelplot arguments.

**Author(s)**

C. Beleites

**See Also**

[levelplot](#)

**Examples**

```
chondro$z <- factor (rep (c("a", "a", "d", "c"),
length.out = nrow (chondro)),
levels = letters [1 : 4])

str (trellis.factor.key (chondro$z))

plotmap (chondro, z ~ x * y)

## switch off using trellis.factor.key:
## note that the factor levels are collapsed to c(1, 2, 3) rather than
## c (1, 3, 4)
plotmap (chondro, z ~ x * y, transform.factor = FALSE)

plotmap (chondro, z ~ x * y,
col.regions = c ("gray", "red", "blue", "dark green"))
```

---

unittests

*hyperSpec unit tests...*

---

**Description**

hyperSpec unit tests If [svUnit](#) is available, run the unit tests and display the results.

**Value**

NA if [svUnit](#) is not available, otherwise TRUE if all tests are passed successfully. If a test fails, `hy.unittest` stops with an error.

**Author(s)**

C. Beleites

**See Also**

[svUnit](#)

**Examples**

```
if (require (svUnit)){
hy.unittest ()
}
```

---

WC	WC...
----	-------

---

**Description**

wc word count of ASCII files

**Usage**

```
wc(file, flags=c("lines", "words", "bytes"))
```

**Arguments**

file	the file name or pattern
flags	the parameters to count, character vector with the long form of the parameters

**Details**

wc uses the system command wc

**Value**

data.frame with the counts and file names, or NULL if wc is not available

**Author(s)**

C. Beleites

---

wl	<i>Getting and Setting the Wavelength Axis...</i>
----	---

---

**Description**

Getting and Setting the Wavelength Axis wl returns the wavelength axis, wl<- sets it.

**Usage**

```
wl(x)
```

```
wl(x, label=NULL, digits=6, short="wl<-", user = NULL, date = NULL) <- value
```

**Arguments**

x	a hyperSpec object
value	either a numeric containing the new wavelength vector, or a list with value\$wl containing the new wavelength vector and value\$label holding the corresponding label.
label	The label for the new wavelength axis. See <a href="#">initialize</a> for details.
digits	handed to <a href="#">signif</a> . See details.
short,user,date	handed to <a href="#">logentry</a> .

**Details**

wl: The wavelength axis of a hyperSpec object can be retrieved and replaced with wl and wl<-, respectively.

When the wavelength axis is replaced, the colnames of x@data\$spc are replaced by the rounded new wavelengths. digits specifies the how many significant digits should be used.

There are two ways to set the label of the new wavelength axis, see the examples. If no label is given, a warning will be issued.

**Value**

wl: a numeric vector  
 wl<-: hyperSpec object

**Note**

wl<- always sets the complete wavelength axis, without changing the columns of the spectra matrix. If you rather want to cut the spectral range, use [\[](#), for interpolation along the spectral axis see [spc.loess](#) and for spectral binning [spc.bin](#).

**Author(s)**

C. Beleites

**See Also**

[signif](#)  
 cutting the spectral range: [\[](#)  
 interpolation along the spectral axis: [spc.loess](#)  
 spectral binning: [spc.bin](#)

**Examples**

```

w1 (laser)

# convert from wavelength to frequency
plot (laser)
w1 (laser, "f / Hz") <- 2.998e8 * w1 (laser) * 1e9
plot (laser)

# convert from Raman shift to wavelength
# excitation was at 785 nm
plot (chondro [1])
w1 (chondro) <- list (w1 = 1e7 / (1e7/785 - w1 (chondro)), label = expression (lambda / nm))
plot (chondro [1])

```

w12i

*Conversion between Wavelength and Spectra Matrix Column...***Description**

Conversion between Wavelength and Spectra Matrix Column Index w12i returns the column indices for the spectra matrix for the given wavelengths. i2w1 converts column indices into wavelengths.

**Usage**

```

w12i(x, wavelength=stop("wavelengths are required. "))
i2w1(x, i)

```

**Arguments**

x	a hyperSpec object
wavelength	the wavelengths to be converted into column indices, either numeric or a formula, see details.
i	the column indices into the spectra matrix for which the wavelength is to be computed

**Details**

w12i: If wavelength is numeric, each of its elements is converted to the respective index. Values outside the range of x@wavelength become NA.

If the range is given as a formula (i.e. start ~ end, a sequence

index corresponding to start : index corresponding to end

is returned. If the wavelengths are not ordered, that may lead to chaos. In this case, call [orderw1](#) first.

Two special variables can be used: `min` and `max`, corresponding to the lowest and highest wavelength of `x`, respectively.

`start` and `end` may be complex numbers. The resulting index for a complex `x` is then

index ( $\text{Re}(x)$ ) +  $\text{Im}(x)$

### Value

`w12i`: A numeric containing the resulting indices for `w12i`

`i2wl`: `i2wl` returns a numeric with the wavelengths

### Author(s)

C. Beleites

### Examples

```
flu
w12i (flu, 405 : 407)
w12i (flu, 405 ~ 407)

## beginning of the spectrum to 407 nm
w12i (flu, min ~ 407)

## 2 data points from the beginning of the spectrum to 407 nm
w12i (flu, min + 2i ~ 407)

## the first 3 data points
w12i (flu, min ~ min + 2i)

## from 490 nm to end of the spectrum
w12i (flu, 490 ~ max)

## the last 8 data points
w12i (flu, max - 7i ~ max)

## get 450 nm +- 3 data points
w12i (flu, 450 - 3i ~ 450 + 3i)

w12i (flu, 300 : 400) ## all NA:
w12i (flu, 600 ~ 700) ## NULL: completely outside flu's wavelength range

i2wl (chondro, 17:20)
```

# Index

- !=, hyperSpec, hyperSpec-method  
(Comparison), 18
- \*Topic **IO**
  - read-spc, 64
  - readENVI, 66
  - scan-txt-Renishaw, 71
  - textio, 84
- \*Topic **aplot**
  - trellis-dot-factor-dot-key, 87
- \*Topic **arith**
  - Arith, 7
  - Comparison, 18
- \*Topic **array**
  - aggregate, 4
- \*Topic **category**
  - aggregate, 4
- \*Topic **classes**
  - hyperSpec-class, 30
- \*Topic **cluster**
  - pearson-dot-dist, 52
- \*Topic **color**
  - palettes, 50
- \*Topic **datagen**
  - baselines, 11
  - initialize, 32
  - spc-bin, 75
  - spc-loess, 79
- \*Topic **datasets**
  - barbiturates, 11
  - chondro, 15
  - flu, 29
  - laser, 35
  - paracetamol, 51
- \*Topic **distribution**
  - sample, 68
- \*Topic **file**
  - read-spc, 64
  - readENVI, 66
  - scan-txt-Renishaw, 71
  - textio, 84
- \*Topic **hplot**
  - levelplot, 36
  - plot, 53
  - plotc, 54
  - plotspc, 56
- \*Topic **iplot**
  - map-sel-poly, 41
  - map.sel.poly, 42
  - spc-identify, 76
- \*Topic **iteration**
  - apply, 6
- \*Topic **manip**
  - baselines, 11
  - bind, 13
  - collapse, 16
  - decomposition, 19
  - empty, 24
  - extractreplace, 25
  - merge, 47
  - rbind-fill, 62
  - seq, 72
  - spc-bin, 75
  - spc-loess, 79
- \*Topic **math**
  - math, 44
- \*Topic **methods**
  - aggregate, 4
  - apply, 6
  - Arith, 7
  - asdataframe, 9
  - bind, 13
  - chk-dot-hy, 15
  - Comparison, 18
  - decomposition, 19
  - dim, 21
  - dimnames, 22
  - extractreplace, 25
  - initialize, 32

- math, 44
- plot, 53
- sample, 68
- show, 73
- split, 81
- sweep, 83
- \*Topic **misc**
  - options, 48
- \*Topic **multivar**
  - mean\_sd, 45
- \*Topic **package**
  - hyperSpec-package, 3
- \*Topic **print**
  - show, 73
- \*Topic **programming**
  - unittests, 88
- \*Topic **univar**
  - mean\_sd, 45
- \*Topic **utilities**
  - dot-DollarNames-dot-hyperSpec, 24
  - unittests, 88
- \*, hyperSpec, hyperSpec-method (Arith), 7
- +, hyperSpec, hyperSpec-method (Arith), 7
- , hyperSpec, hyperSpec-method (Arith), 7
- .DollarNames, 24
- .DollarNames.hyperSpec
  - (dot-DollarNames-dot-hyperSpec), 24
- /, hyperSpec, hyperSpec-method (Arith), 7
- <, hyperSpec, hyperSpec-method (Comparison), 18
- <=, hyperSpec, hyperSpec-method (Comparison), 18
- ==, hyperSpec, hyperSpec-method (Comparison), 18
- >, hyperSpec, hyperSpec-method (Comparison), 18
- >=, hyperSpec, hyperSpec-method (Comparison), 18
- [, 10, 90
- [ (extractreplace), 25
- [, hyperSpec-method (extractreplace), 25
- [<- (extractreplace), 25
- [<-, hyperSpec-method (extractreplace), 25
- [[ (extractreplace), 25
- [[, hyperSpec-method (extractreplace), 25
- [[<- (extractreplace), 25
- [[<-, hyperSpec-method (extractreplace), 25
- \$ (extractreplace), 25
- \$, hyperSpec-method (extractreplace), 25
- \$<- (extractreplace), 25
- \$<-, hyperSpec-method (extractreplace), 25
- %\*% (Arith), 7
- %\*%, hyperSpec, hyperSpec-method (Arith), 7
- %\*%, hyperSpec, matrix-method (Arith), 7
- %\*%, matrix, hyperSpec-method (Arith), 7
- %/%, hyperSpec, hyperSpec-method (Arith), 7
- %, hyperSpec, hyperSpec-method (Arith), 7
- %\*%, 20
- ^, hyperSpec, hyperSpec-method (Arith), 7
- abline, 57
- abs, hyperSpec-method (math), 44
- acos, hyperSpec-method (math), 44
- acosh, hyperSpec-method (math), 44
- aggregate, 4, 4, 5, 7
- aggregate, hyperSpec-method (aggregate), 4
- all, hyperSpec-method (summary), 82
- all.equal, 18, 19, 82
- all.equal (Comparison), 18
- all.equal, -method (Comparison), 18
- all.equal, hyperSpec, hyperSpec-method (Comparison), 18
- alois.palette (palettes), 50
- any, hyperSpec-method (summary), 82
- apply, 6, 6, 7
- apply, hyperSpec-method (apply), 6
- Arith, 7, 19, 44
- Arith, hyperSpec, hyperSpec-method (Arith), 7
- Arith, hyperSpec, matrix-method (Arith), 7
- Arith, hyperSpec, missing-method (Arith), 7
- Arith, hyperSpec, numeric-method (Arith), 7
- Arith, hyperSpec-method (Arith), 7
- Arith, matrix, hyperSpec-method (Arith), 7
- Arith, numeric, hyperSpec-method (Arith), 7
- Arith-method (Arith), 7
- Arithmetic, 8

- as.character, 74
- as.character (show), 73
- as.character, hyperSpec-method (show), 73
- as.data.frame, 10
- as.data.frame (asdataframe), 9
- as.data.frame, hyperSpec, missing, missing-method (asdataframe), 9
- as.data.frame, hyperSpec-method (asdataframe), 9
- as.data.frame-methods (asdataframe), 9
- as.dist, 52
- as.long.df (asdataframe), 9
- as.matrix, 10
- as.matrix (asdataframe), 9
- as.matrix, ANY-method (asdataframe), 9
- as.matrix, hyperSpec-method (asdataframe), 9
- as.matrix-methods (asdataframe), 9
- as.t.df (asdataframe), 9
- as.wide.df (asdataframe), 9
- asdataframe, 9
- asin, hyperSpec-method (math), 44
- asinh, hyperSpec-method (math), 44
- atan, hyperSpec-method (math), 44
- atanh, hyperSpec-method (math), 44
- ave, 4, 5
- ave, hyperSpec-method (aggregate), 4
- axis, 57, 58
- axis.break, 57, 58
  
- barbiturates, 11
- baselines, 11
- bind, 13, 81
  
- cBind, 13, 14
- cbind, 14, 47, 63
- cbind.hyperSpec (bind), 13
- cbind2, 14
- cbind2 (bind), 13
- cbind2, hyperSpec, hyperSpec-method (bind), 13
- cbind2, hyperSpec, missing-method (bind), 13
- ceiling, hyperSpec-method (math), 44
- chk-dot-hy, 15
- chk.hy (chk-dot-hy), 15
- chondro, 15, 37
- collapse, 14, 16, 47
- colnames, 23
- colnames (dimnames), 22
- colnames, hyperSpec-method (dimnames), 22
- colnames<- (dimnames), 22
- colnames<-, hyperSpec-method (dimnames), 22
- colorRampPalette, 51
- Compare (Comparison), 18
- Compare, -method (Comparison), 18
- Compare, hyperSpec, hyperSpec-method (Comparison), 18
- Compare, hyperSpec, matrix-method (Comparison), 18
- Compare, hyperSpec, numeric-method (Comparison), 18
- Compare, hyperSpec-method (Comparison), 18
- Compare, matrix, hyperSpec-method (Comparison), 18
- Compare, numeric, hyperSpec-method (Comparison), 18
- Comparison, 8, 18, 18, 19, 44
- Comparison, -method (Comparison), 18
- connections, 86
- cos, hyperSpec-method (math), 44
- cosh, hyperSpec-method (math), 44
- create (initialize), 32
- create, hyperSpec-method (initialize), 32
- cummax, hyperSpec-method (math), 44
- cummin, hyperSpec-method (math), 44
- cumprod, hyperSpec-method (math), 44
- cumsum, hyperSpec-method (math), 44
  
- decomposition, 19
- digamma, hyperSpec-method (math), 44
- dim, 21, 22
- dim, hyperSpec-method (dim), 21
- dimnames, 22, 23
- dimnames, hyperSpec-method (dimnames), 22
- dot-DollarNames-dot-hyperSpec, 24
- drop, 26, 27, 69
  
- empty, 24
- exp, hyperSpec-method (math), 44
- expm1, hyperSpec-method (math), 44
- export (textio), 84
- Extract, 26, 27
- extractreplace, 25
  
- floor, hyperSpec-method (math), 44

- flu, 29
- gamma, hyperSpec-method (math), 44
- geom\_line, 61, 62
- geom\_point, 59, 60
- geom\_tile, 60, 61
- ggplot, 60–62
- grid.lines, 41, 42
- grid.locator, 41, 42
- grid.points, 41, 42
- gzfile, 71
- hclust, 42
- hy.getOption (options), 48
- hy.getOptions (options), 48
- hy.setOptions, 33, 39
- hy.setOptions (options), 48
- hy.unittest (unittests), 88
- hyperSpec Arith (Arith), 7
- hyperSpec options, 38, 78
- hyperSpec-class, 30
- hyperSpec-package, 3
- i2wl (wl2i), 91
- identify, 58
- import (textio), 84
- initialize, 30, 32, 66, 67, 90
- initialize, -method (initialize), 32
- initialize, hyperSpec-method (initialize), 32
- is.na (summary), 82
- is.na, -method (summary), 82
- is.na, hyperSpec-method (summary), 82
- isample (sample), 68
- isTRUE, 18, 19, 82
- labels, 34, 35
- labels, hyperSpec-method (labels), 34
- labels<- (labels), 34
- laser, 35
- length, 22, 72
- length (dim), 21
- length, hyperSpec-method (dim), 21
- levelplot, 36, 36–38, 87, 88
- levelplot, formula, hyperSpec-method (levelplot), 36
- levelplot, hyperSpec, missing-method (levelplot), 36
- lgamma, hyperSpec-method (math), 44
- lines, 57, 58
- load, 84, 86
- locator, 58, 77, 78
- loess, 79, 80
- loess.control, 79, 80
- log (math), 44
- log, -method (math), 44
- log, hyperSpec-method (math), 44
- log1p, hyperSpec-method (math), 44
- logbook, 39, 49
- logentry, 26, 33, 34, 39, 40, 46, 47, 49, 69, 70, 72, 75, 79–81, 83, 90
- make.names, 62, 63
- map.sel.poly, 41
- map.identify, 41, 49, 78
- map.identify (levelplot), 36
- map.sel.poly, 37, 38, 42, 78
- map.sel.poly (map.sel.poly), 41
- mark.dendrogram, 42
- Math, 6, 8, 19, 44
- Math (math), 44
- math, 44
- Math, -method (math), 44
- Math, hyperSpec-method (math), 44
- Math2 (math), 44
- Math2, -method (math), 44
- Math2, hyperSpec-method (math), 44
- matlab.dark.palette (palettes), 50
- matlab.palette (palettes), 50
- matmult, 8
- max, hyperSpec-method (summary), 82
- mean, 45, 46
- mean (mean\_sd), 45
- mean, hyperSpec-method (mean\_sd), 45
- mean\_pm\_sd (mean\_sd), 45
- mean\_pm\_sd, hyperSpec-method (mean\_sd), 45
- mean\_pm\_sd, matrix-method (mean\_sd), 45
- mean\_pm\_sd, numeric-method (mean\_sd), 45
- mean\_sd, 45
- mean\_sd, hyperSpec-method (mean\_sd), 45
- mean\_sd, matrix-method (mean\_sd), 45
- mean\_sd, numeric-method (mean\_sd), 45
- melt, 10
- merge, 14, 17, 47, 47
- merge, hyperSpec, hyperSpec-method (merge), 47
- merge.data.frame, 47

- min,hyperSpec-method (summary), 82
- ncol, 22
- ncol (dim), 21
- ncol,hyperSpec-method (dim), 21
- new, 33
- new (initialize), 32
- new,hyperSpec-method (initialize), 32
- nrow, 22
- nrow (dim), 21
- nrow,hyperSpec-method (dim), 21
- nwl (dim), 21
- Operators (Comparison), 18
- options, 48
- order, 49, 50, 85
- orderwl, 47, 49, 91
- palettes, 50
- panel.identify, 37
- panel.levelplot.raster, 37
- panel.voronoi, 37, 38
- par, 56, 58, 77
- paracetamol, 51
- pearson-dot-dist, 52
- pearson.dist (pearson-dot-dist), 52
- plot, 38, 53, 53, 57, 58
- plot, -method (plot), 53
- plot, ANY, ANY-method (plot), 53
- plot,hyperSpec, character-method (plot), 53
- plot,hyperSpec, missing-method (plot), 53
- plot-methods (plot), 53
- plot-methods, -method (plot), 53
- plotc, 20, 53, 54, 60
- plotmap, 20, 41, 53, 61
- plotmap (levelplot), 36
- plotmath, 33, 35
- plotspc, 20, 53, 56, 58, 62, 76, 78
- plotvoronoi (levelplot), 36
- point.in.polygon, 41
- polygon, 57, 58
- prcomp, 20
- princomp, 20
- print, 74
- print (show), 73
- print,hyperSpec-method (show), 73
- prod,hyperSpec-method (summary), 82
- qplotc, 59
- qplotmap, 60
- qplotspc, 61
- quantile, 46
- quantile (mean\_sd), 45
- quantile,hyperSpec-method (mean\_sd), 45
- R.matlab, 85, 86
- rainbow, 51
- range,hyperSpec-method (summary), 82
- rBind, 13, 14
- rbind, 14, 17, 47, 63
- rbind-fill, 62
- rbind.fill, 17, 63
- rbind.fill (rbind-fill), 62
- rbind.hyperSpec (bind), 13
- rbind2, 14
- rbind2 (bind), 13
- rbind2,hyperSpec, hyperSpec-method (bind), 13
- rbind2,hyperSpec, missing-method (bind), 13
- read-spc, 64
- read.ENVI, 49, 68, 85, 86
- read.ENVI (readENVI), 66
- read.spc, 85, 86
- read.spc (read-spc), 64
- read.table, 85, 86
- read.txt.long, 72
- read.txt.long (textio), 84
- read.txt.wide, 72
- read.txt.wide (textio), 84
- readENVI, 66
- readMat, 85, 86
- rect, 43
- round,hyperSpec-method (math), 44
- rownames, 23
- rownames (dimnames), 22
- rownames,hyperSpec-method (dimnames), 22
- rownames<- (dimnames), 22
- rownames<-,hyperSpec-method (dimnames), 22
- S4groupGeneric, 8, 19, 44
- sample, 68, 69
- sample, data.frame-method (sample), 68
- sample,hyperSpec-method (sample), 68
- sample, matrix-method (sample), 68
- save, 84, 86
- scan, 72

- scan-dot-txt-dot-Witec, [70](#)
- scan-txt-Renishaw, [71](#)
- scan.txt.Renishaw, [86](#)
- scan.txt.Renishaw (scan-txt-Renishaw), [71](#)
- scan.txt.Witec
  - (scan-dot-txt-dot-Witec), [70](#)
- scan.zip.Renishaw (scan-txt-Renishaw), [71](#)
- sd, [45, 46](#)
- sel.poly, [41](#)
- sel.poly (map.sel.poly), [42](#)
- seq, [72, 72, 73](#)
- seq, hyperSpec-method (seq), [72](#)
- seq.hyperSpec (seq), [72](#)
- seq.int, [72](#)
- seq\_along, [72](#)
- show, [73, 74](#)
- show, hyperSpec-method (show), [73](#)
- sign, hyperSpec-method (math), [44](#)
- signif, [90](#)
- signif, hyperSpec-method (math), [44](#)
- sin, hyperSpec-method (math), [44](#)
- sinh, hyperSpec-method (math), [44](#)
- spc-bin, [75](#)
- spc-identify, [76](#)
- spc-loess, [79](#)
- spc.bin, [90](#)
- spc.bin (spc-bin), [75](#)
- spc.fit.poly (baselines), [11](#)
- spc.identify, [37, 38, 49, 58](#)
- spc.identify (spc-identify), [76](#)
- spc.label.default (spc-identify), [76](#)
- spc.label.wlonly (spc-identify), [76](#)
- spc.loess, [90](#)
- spc.loess (spc-loess), [79](#)
- spc.NA.linapprox, [80](#)
- spc.point.default (spc-identify), [76](#)
- spc.point.max (spc-identify), [76](#)
- spc.point.min (spc-identify), [76](#)
- spc.point.sqr (spc-identify), [76](#)
- split, [81, 81](#)
- split, ANY-method (split), [81](#)
- split, hyperSpec-method (split), [81](#)
- split-methods (split), [81](#)
- sqrt, hyperSpec-method (math), [44](#)
- stack, [10](#)
- stacked.offsets, [57](#)
- stacked.offsets (plotspc), [56](#)
- sum, hyperSpec-method (summary), [82](#)
- Summary, [44, 82](#)
- Summary (summary), [82](#)
- summary, [74, 82](#)
- summary (show), [73](#)
- Summary, -method (summary), [82](#)
- Summary, hyperSpec-method (summary), [82](#)
- summary, hyperSpec-method (show), [73](#)
- svUnit, [88](#)
- sweep, [8, 18, 83, 83, 84](#)
- sweep, hyperSpec-method (sweep), [83](#)
- sweep-methods, [8, 19](#)
- sweep-methods (sweep), [83](#)
- Sys.info, [40](#)
- Sys.time, [40](#)
- tan, hyperSpec-method (math), [44](#)
- tanh, hyperSpec-method (math), [44](#)
- tapply, [4, 5](#)
- text, [43, 77](#)
- textio, [65, 68, 84](#)
- title, [57, 58](#)
- trellis-dot-factor-dot-key, [87](#)
- trellis.factor.key, [36, 38](#)
- trellis.factor.key
  - (trellis-dot-factor-dot-key), [87](#)
- trigamma, hyperSpec-method (math), [44](#)
- trunc, hyperSpec-method (math), [44](#)
- unit, [36](#)
- unittests, [88](#)
- unzip, [86](#)
- validObject, [15](#)
- validObject (chk-dot-hy), [15](#)
- validObject, hyperSpec-method (chk-dot-hy), [15](#)
- wc, [89](#)
- wl, [23, 89](#)
- wl2i, [26, 27, 72, 73, 91](#)
- wl<- (wl), [89](#)
- write.table, [85, 86](#)
- write.txt.long (textio), [84](#)
- write.txt.wide (textio), [84](#)
- writeMat, [85, 86](#)
- xyplot, [54, 55](#)