

# Package ‘hsdar’

October 6, 2015

**Type** Package

**Title** Manage, Analyse and Simulate Hyperspectral Data

**Version** 0.3.1

**Date** 2015-10-06

**Author** Lukas W. Lehnert [cre, aut],  
Hanna Meyer [aut],  
Joerg Bendix [aut]

**Maintainer** Lukas W. Lehnert <lukaslehnert@googlemail.com>

**Depends** R (>= 2.10.0), sp, raster, rgdal, rootSolve, signal, methods

**Suggests** rgl

## Description

Transformation of reflectance spectra, calculation of vegetation indices and red edge parameters, spectral resampling for hyperspectral remote sensing, simulation of reflectance and transmittance using the leaf reflectance model PROSPECT and the canopy reflectance model PROSAIL.

**License** GPL

**LazyLoad** yes

**BuildVignettes** yes

**Copyright** see file COPYRIGHTS

**URL** <http://lcrs.geographie.uni-marburg.de>,  
<http://teledetection.ipgp.jussieu.fr/prosail/>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2015-10-06 17:58:02

## R topics documented:

hsdar-package . . . . .	3
addep . . . . .	4
apply.DistMat3D . . . . .	5
apply.Speclib . . . . .	7

attribute . . . . .	8
bandnames . . . . .	9
bdri . . . . .	10
checkhull . . . . .	11
Clman . . . . .	12
clman . . . . .	14
Clman-class . . . . .	15
cor.test . . . . .	15
cubePlot . . . . .	16
cut_specfeat . . . . .	17
define.features . . . . .	18
deletcp . . . . .	20
derivative.speclib . . . . .	21
dim.speclib . . . . .	22
dist.speclib . . . . .	23
distMat3D . . . . .	25
DistMat3D-class . . . . .	27
endmember_spectra . . . . .	27
feature_properties . . . . .	28
get.gaussian.response . . . . .	30
get.response . . . . .	31
get.sensor.characteristics . . . . .	32
get.sensor.name . . . . .	33
getcp . . . . .	34
getNRI . . . . .	35
get_reflectance . . . . .	36
glm.nri . . . . .	37
hsdardocs . . . . .	38
HyperSpecRaster . . . . .	39
HyperSpecRaster-class . . . . .	41
idSpeclib . . . . .	42
makehull . . . . .	43
mask . . . . .	44
meanfilter . . . . .	46
merge . . . . .	47
nri . . . . .	48
Nri-class . . . . .	49
Nri-methods . . . . .	50
nri_best_performance . . . . .	51
plot.Nri . . . . .	52
plot.Specfeat . . . . .	54
plot.Speclib . . . . .	55
PROSAIL . . . . .	56
PROSPECT . . . . .	59
Raster-methods . . . . .	61
rastermeta . . . . .	62
rededge . . . . .	63
response_functions . . . . .	64

smgm . . . . .	65
smoothSpeclib . . . . .	66
soilindex . . . . .	68
specfeat . . . . .	70
Specfeat-class . . . . .	71
speclib . . . . .	72
Speclib-class . . . . .	74
spectra . . . . .	76
spectral.resampling . . . . .	77
spectral_data . . . . .	78
subset.speclib . . . . .	79
t.test . . . . .	80
transform_speclib . . . . .	81
unmix . . . . .	83
updatecl . . . . .	84
usagehistory . . . . .	86
vegindex . . . . .	87
wavelength . . . . .	92

<b>Index</b>	<b>94</b>
--------------	-----------

---

hsdar-package	<i>Manage, analyse and simulate hyperspectral data in R</i>
---------------	---

---

## Description

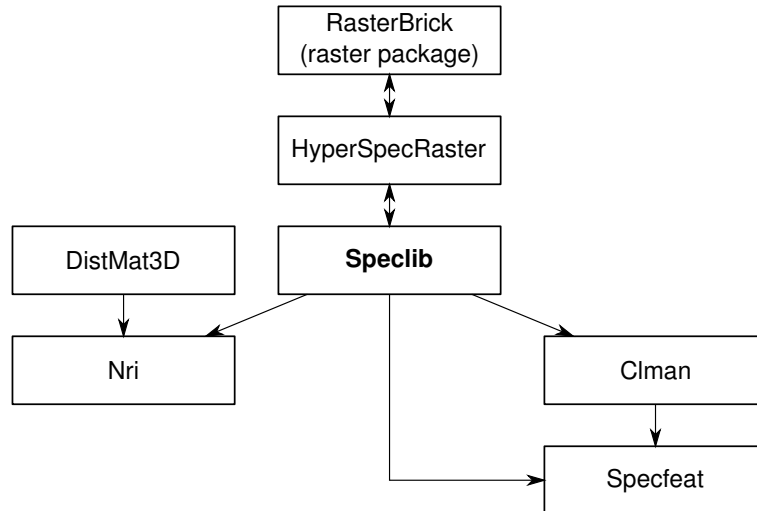
The **hsdar** package contains classes and functions to manage, analyse and simulate hyperspectral data. These might be either spectrometer measurements or hyperspectral images through the interface of **raster**.

## Details

**hsdar** provides amongst others the following functionality.

- Data handling: **hsdar** is designed to handle even large sets of spectra. Spectra are stored in a [Speclib](#) containing, amongst other details, the wavelength and reflectance for each spectrum. **hsdar** further contains functions for [plotting](#) spectral data and [applying](#) functions to spectra.
- Data manipulation: A variety of established methods for data manipulation such as filter functions ([smoothSpeclib](#)), resampling of bands to various satellite sensors ([spectral.resampling](#)), continuum removal ([transform\\_speclib](#)), calculations of derivations ([derivative.speclib](#)) and extraction of absorption features ([cut\\_specfeat](#)) are implemented.
- Data analysis: Supported methods to analyse vegetation spectra are the calculation of red edge parameters ([rededge](#)), vegetation ([vegindex](#)) and soil ([soilindex](#)) indices as well as ndvi-like narrow band indices ([nri](#)). **hsdar** further enables to perform spectral unmixing of spectra ([unmix](#)) by use of endmember spectra.
- Data simulation: **hsdar** has implemented the models PROSAIL 5B ([PROSAIL](#), Jacquemoud et al. 2009) and PROSPECT 5 ([PROSPECT](#), Jacquemoud and Baret 1990) to simulate spectra of canopy and plants.

Several classes are defined and used in **hsdar**. Most of the classes are used and respective objects are created internally. However, the following figure gives an overview which class is used at which stage of processing.



To see the preferable citation of the package, type `citation("hsdar")`.

### Acknowledgements

Development initially funded by German Federal Ministry of Education and Research (03G0808C) in the scope of the project PaDeMoS as precondition to develop a space-based Pasture Degradation Monitoring System for the Tibetan Plateau.

### Author(s)

Lukas Lehnert, Hanna Meyer, Joerg Bendix

---

addcp

*Add fix point*

---

### Description

Add fix point to continuum line.

### Usage

```
addcp(x, ispec, cpadd)
```

### Arguments

x	Object of class Clman.
ispec	ID or index of spectrum to be modified.
cpadd	Single value or vector of wavelength containing new fix points.

**Value**

Object of class Clman containing the updated version of x.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [deletecp](#), [getcpc](#), [checkhull](#), [makehull](#), [updatecl](#),  
[idSpeclib](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Add fix point at 4595 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2495)

## Plot new line
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error
```

---

apply.DistMat3D

*Apply function for class DistMat3D*

---

**Description**

Apply function to values in a 3-D distance matrix

**Usage**

```
## S4 method for signature 'DistMat3D'
apply(X, MARGIN, FUN, ...)
```

**Arguments**

X	Object of class 'DistMat3D'.
MARGIN	A vector giving the subscripts (dimensions) of the DistMat3D-object which the function will be applied over (see details).
FUN	Function to be applied. Matched with <a href="#">match.fun</a> .
...	Further arguments passed to FUN.

**Details**

The specified function is either applied to the distances of all samples (MARGIN = 1) or to all distances for each sample (MARGIN = 3). In the first case, if X would be replaced by an array of same dimensions the return value would be equal if the following code is applied:

```
apply(X, MARGIN = c(1,2), FUN),
```

where X is an array (see examples).

**Value**

Depending on the length of the return value of the specified function, objects of classes numeric or matrix are returned.

**Author(s)**

Lukas Lehnert

**See Also**

[apply](#), [match.fun](#), [DistMat3D](#)

**Examples**

```
data(spectral_data)

## Calculate NDVI
ndvi <- nri(spectral_data, b1=800, b2=680)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)
class(nri_WV@nri)

## Calculate mean value of all samples for all indices
meanIndexVals <- apply(nri_WV@nri, MARGIN = 1, FUN = mean)
meanIndexVals

## Same but for array
nri_WV_dat <- as.array(nri_WV@nri)
meanIndexVals_arr <- apply(nri_WV_dat, MARGIN = c(1, 2), FUN = mean)
```

```
meanSampleVals <- apply(nri_WV@nri, MARGIN = 3, FUN = mean)
meanSampleVals_arr <- apply(nri_WV_dat, MARGIN = 3, FUN = mean)
```

---

apply.Speclib      *Apply function for class Speclib*

---

## Description

Apply function over all spectra or a subset of spectra

## Usage

```
## S4 method for signature 'Speclib'
apply(X, FUN, byattributes = NULL, ...)
```

## Arguments

X	Object of class Speclib
FUN	Function to be applied. Matched with <a href="#">match.fun</a> .
byattributes	Character string giving the name of the column in the attributes to be used as subsets to apply function FUN on.
...	Further arguments passed to FUN.

## Value

Object of class Speclib.

## Author(s)

Lukas Lehnert

## See Also

[apply](#), [match.fun](#), [Speclib](#)

## Examples

```
data(spectral_data)

mean_spectrum <- apply(spectral_data, FUN = mean)
plot(mean_spectrum)

## Same as above but seperately for both sites
mean_spectra <- apply(spectral_data, FUN = mean, byattributes = "Site")
plot(mean_spectra, FUN = 1, ylim = c(0,50))
plot(mean_spectra, FUN = 2, new = FALSE)
attribute(mean_spectra)
```

---

attribute

*Handling attributes of spectra*

---

## Description

Returning and setting attributes of spectra in speclib.

## Usage

```
## S4 method for signature 'Speclib'  
attribute(object)  
  
## S4 replacement method for signature 'Speclib,data.frame'  
attribute(object) <- value  
  
## S4 replacement method for signature 'Speclib,matrix'  
attribute(object) <- value
```

## Arguments

object	Object of class Speclib.
value	Data frame with <code>nrow(value) == nspectra(object)</code> , or NULL.

## Value

For `attribute<-`, the updated object. `attribute` returns a data frame with attribute data.

## Author(s)

Lukas Lehnert

## See Also

[Speclib](#)

## Examples

```
data(spectral_data)  
  
## Returning attributes  
attribute(spectral_data)
```



---

bandnames	<i>Handling names of bands</i>
-----------	--------------------------------

---

**Description**

Returning and setting names of bands in Speclib

**Usage**

```
bandnames(x)  
bandnames(x) <- value
```

**Arguments**

x	Object of class Speclib.
value	Character vector of the same length as nbands(x), or NULL.

**Value**

For bandnames<-, the updated object. Otherwise a vector giving the name of each band in Speclib is returned.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#)

**Examples**

```
data(spectral_data)  
bandnames(spectral_data)
```

---

bdri *Band depth ratio indices*

---

### Description

Calculate band depth ratio indices for objects of class Specfeat.

### Usage

```
bdri(x, fnumber, index = "ndbi")
```

### Arguments

x	Object of class Specfeat.
fnumber	Integer. Index of feature to modify.
index	Method to be applied. Currently, "bdr", "ndbi" and "bna" are available.

### Details

Method "bdr" calculates the normalised band depth ratio as

$$bdr = \frac{BD}{Dc},$$

with  $BD$  is the band depth calculated by [transform\\_speclib](#) and  $Dc$  is the maximum band depth called band centre. Method "ndbi" calculates the the normalised band depth index as

$$ndbi = \frac{BD - Dc}{BD + Dc}.$$

Method "bna" calculates the band depth normalised to band area as

$$bna = \frac{BD}{Da},$$

where  $Da$  is the area of the absorption feature (see [feature\\_properties](#)). For further information see Mutanga and Skidmore (2004).

### Value

Object of class specfeat containing the updated version of x.

### Author(s)

Lukas Lehnert and Hanna Meyer

### References

Mutanga, O. and Skidmore, A. (2004): Hyperspectral band depth analysis for a better estimation of grass biomass (*Cenchrus ciliaris*) measured under controlled laboratory conditions. International Journal of applied Earth Observation and Geoinformation, 5, 87-96

**See Also**

[transform\\_speclib](#), [define.features](#), [specfeat](#)

**Examples**

```
data(spectral_data)

## Transform speclib
bd <- transform_speclib(subset(spectral_data, Site == "Namco"),
                        method = "sh", out = "bd")

## Define features automatically
features <- define.features(bd)

## Isolate the features around 450nm, 700nm, 1200nm and 1500nm and
## convert to specfeat.
featureSelection <- specfeat(features, c(450,700,1200,1500))

## Plot features
plot(featureSelection,1:4)

## Calculate normalized band depth index for first feature
featureSelection_bdri <- bdri(featureSelection, 1, index = "ndbi")

## Plot result
plot(featureSelection_bdri)
```

---

checkhull

*Check continuum line*

---

**Description**

Check if continuum line is intersecting the reflectance curve.

**Usage**

```
checkhull(x, ispec)
```

**Arguments**

x	Object of class clman.
ispec	ID or index of spectrum to be checked.

**Value**

Object of class list.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [addcp](#), [deletecp](#), [makehull](#), [updatecl](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Add fix point at 4595 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2495)

## Plot new line
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error

## Add fix point at 4596 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2496)

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error
```

---

Clman

\* *Clman class*

---

**Description**

Class to store and handle manual continuum lines

**Usage**

```
Clman(wavelength, cp, hull, spectra, outdatedhull = NULL, mask = NULL)
```

```
## S4 method for signature 'Clman'
plot(x, ispec, subset = NULL, numeratepoints = TRUE, ...)
```

**Arguments**

wavelength	Vector with corresponding wavelength for each band.
cp	Data frame or matrix containing fix points. Fix points have numbers greater than 0, all other bands are 0.
hull	Data frame or matrix containing linear hull.
spectra	Data frame, matrix of raster object of class 'SpatialGridDataFrame' with spectral data.
outdatedhull	Data frame or matrix containing hull of step before for undo porposes.
mask	Data frame with masked parts in the spectra. See <a href="#">mask</a> .
x	Object of class <code>clman</code> .
ispec	Name or index of spectrum to be plotted.
subset	Lower and upper spectral limits used for plot.
numeratepoints	Flag if points should be numerated in plot.
...	Further arguments passed to <code>plot.default</code> .

**Value**

Object of class `Clman`.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot clman
plot(spec_clman, ispec = 1, subset = c(400, 1000))
```

---

`clman`*Methods to create, manipulate and query objects of class 'Clman'.*

---

**Description**

Methods to create, manipulate and query objects of class 'Clman'.

**Usage**

```
## Creation of objects
## S4 method for signature 'Clman'
initialize(.Object, ...)

## S4 method for signature 'Clman'
spectra(object, ...)

## S4 replacement method for signature 'Clman,data.frame'
spectra(object) <- value

## S4 replacement method for signature 'Clman,matrix'
spectra(object) <- value

## S4 replacement method for signature 'Clman,numeric'
spectra(object) <- value
```

**Arguments**

<code>.Object, object</code>	Matrix, numeric or array in cases of creation of 'Clman' objects otherwise object of class 'Clman'.
<code>value</code>	Object of class numeric, matrix or array which is used for replacement of the values in x.
<code>...</code>	Arguments passed to <a href="#">createspeclib</a> .

**Value**

For `spectra<-`, the updated object. Otherwise a matrix returning the spectra in the Clman object.

**Note**

The functions to create objects of class Clman are mainly internally needed by [transform\\_speclib](#).

**Author(s)**

Lukas Lehnert

**See Also**

[dist.speclib](#), [Clman](#), [transform\\_speclib](#)

---

Clman-class	* <i>Clman class</i>
-------------	----------------------

---

**Description**

Class to handle continuum removal objects (extends [Speclib](#) class).

**Details**

The class extends [Speclibs](#) and adds two additional slots:

- cp: Object of class `matrix` containing continuum points for all spectra (rows) and bands (columns).
- hull: Object of class `matrix` containing hull lines for all spectra (rows) and bands (columns).

**Note**

See figure in [hsdar-package](#) for an overview of classes in `hsdar`.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#)

---

<code>cor.test</code>	<i>Test for association/correlation between nri values and vector of samples</i>
-----------------------	--

---

**Description**

Test for association between paired samples (with one variable being nri-values), using one of Pearson's product moment correlation coefficient, Kendall's tau or Spearman's rho.

**Usage**

```
## S4 method for signature 'Nri'
cor.test(x, y, ...)
```

**Arguments**

x	Object of class <code>Nri</code> or numerical vector
y	Object of class <code>Nri</code> or numerical vector
...	Further arguments passed to <code>cor.test</code>

**Details**

NRI-values may be used as x and/or as y variable. If x and y are NRI-values the number of samples in both datasets must be equal. For additional information on correlation tests see details in [cor.test](#).

**Value**

Object of class `Nri`

**Author(s)**

Lukas Lehnert

**See Also**

[plot](#), [cor.test](#), [glm.nri](#), [lm.nri](#), [getNRI](#)

**Examples**

```
data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

cortestnri <- cor.test(nri_WV, attribute(spec_WV)$Soil.moisture)

cortestnri
```

---

cubePlot

*cubePlot*

---

**Description**

Plotting 3D cube of hyperspectral data using **rgl**-package

**Usage**

```
cubePlot(x, r, g, b, ncol = 1, nrow = 1,
         sidecol = colorRamp(palette(heat.colors(100))), ...)
```



**Arguments**

x	Object of class HyperSpecRaster.
r	Integer. Index of band used as red channel. If omitted, the band closest to 680 nm is selected.
g	Integer. Index of band used as green channel. If omitted, the band closest to 540 nm is selected.
b	Integer. Index of band used as blue channel. If omitted, the band closest to 470 nm is selected.
ncol	Integer giving the column(s) in x which is/are used to plot the spectral dimension.
nrow	Integer giving the row(s) in x which is/are used to plot the spectral dimension.
sidecol	ColorRamp used to illustrate spectral dimension.
...	Further arguments (currently ignored)

**Author(s)**

Lukas Lehnert

**See Also**

[HyperSpecRaster](#)

**Examples**

```
## Not run:  
data(spectral_data)  
ras <- HyperSpecRaster(spectral_data, nrow = 9, ncol = 9)  
  
cubePlot(ras)  
  
## End(Not run)
```

---

cut\_specfeat

*Cut absorption features*

---

**Description**

Function cuts absorption features to a user-specified range.

**Usage**

```
cut_specfeat(x, ..., fnumber, limits)
```

**Arguments**

x	An object of class "Specfeat" containing isolated features determined by <a href="#">specfeat</a> .
fnumber	A vector of the positions of the features in x to be cut.
limits	A vector containing the start and end wavelength for each fnumber. The corresponding feature will be cut to this specified range.
...	Further arguments passed to generic functions. Currently ignored.

**Value**

An object of class [Specfeat](#) containing the cut features.

**Author(s)**

Hanna Meyer and Lukas Lehnert

**See Also**

[define.features](#), [specfeat](#), [Specfeat](#)

**Examples**

```
data(spectral_data)

##Example to cut the features around 450nm and 700nm to a specific range
## Transform speclib
bd <- transform_speclib(subset(spectral_data, Site == "Namco"),
                        method = "sh", out = "bd")

## Define features
features <- define.features(bd)

## Convert speclib to specfeat giving center wavelength of features
featureSelection <- specfeat(features, c(450,700,1200,1500))

## Cut 1st and 2nd feature to [310 nm, 560 nm] and [589 nm, 800 nm]
featuresCut <- cut_specfeat(x = featureSelection, fnumber = c(1,2),
                           limits = c(c(310, 560), c(589, 800)))

## Plot result (1st and 2nd feature)
plot(featuresCut, fnumber = 1:2)
```

---

define.features

*Definition of absorption features*

---

**Description**

Function sets the spectral range of absorption features.

## Usage

```
define.features(x, tol = 1.0e-7, FWL = NULL)
```

## Arguments

x	Object of class <code>Speclib</code> containing the band depth or ratio transformed reflectance spectra.
tol	The tolerance of the band depth which defines a wavelength as a start or end point of a feature. Usually a band depth of 0 or a ratio of 1 indicates feature limits, however, better results are achieved if slightly deviating values are tolerated.
FWL	Optional. If passed, result is directly converted into <code>Specfeat</code> . A vector containing one wavelength per feature to be isolated, e.g. the major absorption features. Features which include these specified wavelengths will be isolated.

## Details

Absorption features are defined as the area between local maxima in the reflectance spectra. This function adds the information of the feature limits to the `Speclib`. Thus, it is a pre-processing step to isolate features.

## Value

The updated `Speclib` containing additional information about the feature limits. If `FWL` is not `NULL`, result will be of class `Specfeat`.

## Author(s)

Hanna Meyer and Lukas Lehnert

## See Also

[transform\\_speclib](#), [specfeat](#), [Specfeat](#)

## Examples

```
data(spectral_data)

##Example to define feature limits
bd <- transform_speclib(subset(spectral_data, Site == "Namco"),
                        method = "sh", out = "bd")

## Define features
features <- define.features(bd)
```

---

deletecp	<i>Delete fix point</i>
----------	-------------------------

---

**Description**

Delete fix point from continuum line.

**Usage**

```
deletecp(x, ispec, cpdelete)
```

**Arguments**

x	Object of class <code>Clman</code> .
ispec	ID or index of spectrum to be modified.
cpdelete	Single value or vector of wavelength containing fix point(s) to be deleted.

**Value**

Object of class `Clman` containing the updated version of x.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [addcp](#), [getcp](#), [checkhull](#), [makehull](#), [updatecl](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)
## Mask parts not necessary for the example
mask(spec) <- c(1600, 2600)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, subset = c(1100, 1300))

## Find wavelength of fix point to be deleted
getcp(spec_clman, 1, subset = c(1100, 1300))

## Delete all fix points between 1200 and 1240 nm
spec_clman <- deletecp(spec_clman, 1, c(1200:1240))
```

```
## Plot new line
plot(spec_clman, ispec = 1, subset = c(1100, 1300))

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error
```

---

derivative.speclib      *Derivation*

---

## Description

Calculate derivations of spectra

## Usage

```
derivative.speclib(x, m = 1, method = "sgolay", ...)
```

## Arguments

x	Object of class <code>Speclib</code> .
m	Return the m-th derivative of the spectra.
method	Character string giving the method to be used. Valid options are "finApprox" or "sgolay".
...	Further arguments passed to <code>sgolayfilt</code> .

## Details

Two different methods are available:

- Finite approximation (finApprox):

$$\frac{dr}{d\lambda} = \frac{r(\lambda_i) - r(\lambda_{i+1})}{\Delta\lambda},$$

where  $r_i$  is the reflection in band  $i$  and  $\Delta\lambda$  the spectral difference between adjacent bands.

- Savitzky-Golay derivative computation (sgolay)

## Value

Object of class `Speclib`.

## Author(s)

Lukas Lehnert

## References

Tsai, F. & Philpot, W. (1998): Derivative analysis of hyperspectral data. Remote Sensing of Environment 66/1. 41-51.

## See Also

[sgolayfilt](#), [vegindex](#)

## Examples

```
data(spectral_data)

## Calculate 1st derivation
d1 <- derivative.speclib(spectral_data)

## Calculate 2nd derivation
d2 <- derivative.speclib(spectral_data, m = 2)

## Calculate 3rd derivation
d3 <- derivative.speclib(spectral_data, m = 3)

par(mfrow=c(2,2))
plot(spectral_data)
plot(d1)
plot(d2)
plot(d3)
```

---

dim.speclib

*Dimensions of Speclib*

---

## Description

Get dimension(s) of Speclib

## Usage

```
## S4 method for signature 'Speclib'
dim(x)

nspectra(x)
nbands(x)
```

## Arguments

x                    Object of class Speclib.

## Value

Vector of length = 2 or single integer value.

**Author(s)**

Lukas Lehnert

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

```
data(spectral_data)
```

```
dim(spectral_data)
```

---

dist.speclib	<i>Distance between spectra</i>
--------------	---------------------------------

---

**Description**

Calculation of distance matrices by using one of the various distance measure to compute the distances between the spectra in Speclib. Spectral Angle Mapper (SAM) is calculated with `sam` giving reference spectra or with `sam_distance` taking all combinations between spectra in single Speclib into account.

**Usage**

```
dist.speclib(x, method = "sam", ...)
```

```
## Direct call to Spectral Angle Mapper function
```

```
sam(x, ref)
```

```
sam_distance(x)
```

**Arguments**

<code>x</code>	Object of class <code>Speclib</code> .
<code>method</code>	The distance measure to be used. This must be one of "sam", "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
<code>ref</code>	Object of class <code>Speclib</code> containing reference spectra.
<code>...</code>	Further arguments, passed to other methods.

**Details**

Available distance measures are "spectral angle mapper" (`sam`) and all distance measures available in `dist`. Spectral angle mapper is calculated with the following formula:

$$sam = \cos^{-1} \left( \frac{\sum_{i=1}^{nb} t_i r_i}{\sqrt{\sum_{i=1}^{nb} t_i^2} \sqrt{\sum_{i=1}^{nb} r_i^2}} \right)$$

$nb$  is the number of bands in Speclib.  $t_i$  and  $r_i$  are the reflectances of target and reference spectrum in band  $i$ , respectively.

### Value

The `dist`-method for Speclibs returns an object of class "dist". See [dist](#) for further information on class "dist". Both other functions return an object of class matrix.

### Author(s)

Lukas Lehnert

### References

Kruse, F. A.; Lefkoff, A. B.; Boardman, J. W.; Heidebrecht, K. B.; Shapiro, A. T.; Barloon, P. J. & Goetz, A. F. H. (1993). The spectral image processing system (SIPS) – interactive visualization and analysis of imaging spectrometer data. *Remote Sensing of Environment*, 44, 145-163.

### See Also

[dist](#), [Speclib](#)

### Examples

```
data(spectral_data)

## Mask channel crossing part (arround 1050 nm) and strong
## water absorption part (above 1350 nm)
mask(spectral_data) <- c(1045, 1055, 1350, 1706)

## Calculate distance between all spectra from Namco-site
## using spectral angle mapper
dist.speclib(subset(spectral_data, Site == "Namco"))

## Calculate spectral angle mapper between reference spectrum
## and spectral_data
## 1. Use pure endmember spectrum from Namco-site
data(endmember_spectra)

## Calculate distance between vegetation reference spectrum
## and spectra from Namco-site using spectral angle mapper
distance <- sam(subset(spectral_data, Site == "Namco"),
                subset(endmember_spectra,
                      Site == "Namco" & Endmember == "PV"))

## Plot result
plot(distance ~ attribute(subset(spectral_data, Site == "Namco"))$PV,
      xlab = "Observed vegetation cover", ylab = "SAM")

## 2. Use PROSPECT as reference
ref <- PROSPECT(N = 1.3)
```



```

## Mask spectra to equalize spectral ranges
mask(ref) <- c(1045, 1055, 1350, 2501)
mask(spectral_data) <- c(300, 400)

## Calculate distance between spectrum from PROSPECT
## and spectra from Namco-site using spectral angle mapper
distance <- sam(subset(spectral_data, Site == "Namco"), ref)

## Plot result
plot(distance ~ attribute(subset(spectral_data, Site == "Namco"))$PV,
      xlab = "Observed vegetation cover", ylab = "SAM")

```

---

distMat3D	<i>Methods to create, manipulate and query objects of class 'DistMat3D'.</i>
-----------	--

---

### Description

Methods to create, manipulate and query objects of class 'DistMat3D'. The following relational operators are defined to compare values between 'DistMat3D'-object(s): <, <=, ==, >, >=

### Usage

```

## Creation of objects
## S4 method for signature 'numeric'
distMat3D(x, ncol, nlyr)

## S4 method for signature 'matrix'
distMat3D(x, lower_tri = TRUE)

## S4 method for signature 'array'
distMat3D(x, lower_tri = TRUE)

## Conversion methods
## S4 method for signature 'DistMat3D'
as.array(x)

## S4 method for signature 'DistMat3D'
as.matrix(x, lyr = 1)

## Query of properties
## S4 method for signature 'DistMat3D'
dim(x)

## S4 method for signature 'DistMat3D'
ncol(x)

## S4 method for signature 'DistMat3D'

```

```
nrow(x)

## Manipulate and query data in objects
## S4 method for signature 'DistMat3D,ANY,ANY'
x[i, j, n]

## S4 replacement method for signature 'DistMat3D,ANY,ANY'
x[i, j, n] <- value

## S4 method for signature 'DistMat3D'
show(object)
```

### Arguments

x, object	Matrix, numeric or array in cases of creation of 'DistMat3D' objects otherwise object of class 'DistMat3D'.
ncol	Number of columns in the new 'DistMat3D' object.
nlyr	Number of layer in the new 'DistMat3D' object.
lower_tri	Flag if only the lower triangle is used.
lyr	Layer in the 'DistMat3D' object to be transformed into matrix.
value	Object of class numeric, matrix or array which is used for replacement of the values in x.
i, j, n	Subscripts to access data.

### Author(s)

Lukas Lehnert

### See Also

[DistMat3D](#), [apply](#), [Nri](#)

### Examples

```
data(spectral_data)

## Mask channel crossing part (arround 1050 nm) and strong
## water absorption part (above 1350 nm)
mask(spectral_data) <- c(1045, 1055, 1350, 1706)

## Calculate SAM distances (object of class 'dist')
sam_dist <- dist.speclib(subset(spectral_data, Site == "Namco"))

## Convert to class 'distMat3D'
sam_dist <- distMat3D(as.matrix(sam_dist))

sam_dist
```

---

DistMat3D-class      \* *DistMat3D* class

---

**Description**

Class to store effectively (large) distance matrices (up to 3D).

**Details**

Object with 3 slots:

- values: Numerical vector containing distance values
- ncol: Number of columns in the 3D-matrix. Number of columns equals always the number of rows
- nlyr: Number of layers in the 3D-matrix

**Note**

See figure in [hsdar-package](#) for an overview of classes in hsdar.

**Author(s)**

Lukas Lehnert

**See Also**

[distMat3D](#)

---

endmember\_spectra      *Hyperspectral samples from pure endmembers*

---

**Description**

Hyperspectral samples from pure endmembers of two ecosystems on the Tibetan Plateau

**Usage**

```
data(endmember_spectra)
```

**Format**

An object of class `Speclib`

**Details**

Data has been sampled during vegetation period 2012 from two of the most wide-spread vegetation types on the Tibetan Plateau: *Kobresia* meadows and dry alpine steppes. Measurements were taken with a HandySpec Field portable spectrometer (tec5 AG Oberursel, Germany). This device has two channels measuring incoming and reflected radiation simultaneously between 305 and 1705 nm in 1 nm steps. Note that example data only contains spectral ranges between 305 and 1350 nm.

The [Speclib](#) contains 8 spectra, which are mean values of several 10th of single samples. The 8 spectra encompass soil (dry/wet), photosynthetically active vegetation, open *Kobresia* turf, stones, and photosynthetically non-active vegetation.

**Author(s)**

Lukas Lehnert, Hanna Meyer

**References**

Lehnert, L. W.; Meyer, H.; Meyer, N.; Reudenbach, C. & Bendix, J. (2014): A hyperspectral indicator system for rangeland degradation on the Tibetan Plateau: A case study towards spaceborne monitoring. *Ecological Indicators*, 39, 54 - 64. DOI: 10.1016/j.ecolind.2013.12.005

---

feature\_properties      *Calculation of properties of features*

---

**Description**

Function to calculate feature properties such as the area, the position of the maximum and several other parameters.

**Usage**

```
feature_properties(x)
```

**Arguments**

x                      Object of class Specfeat

**Details**

The function calculates several parameters:

- *area*: The feature area is calculated by

$$area_{F_i} = \sum_{k=min(\lambda)}^{max(\lambda)} BD\lambda,$$

with  $area_{F_i}$  is the area of the feature  $i$ ,  $min(\lambda)$  is the minimum wavelength of the spectrum,  $max(\lambda)$  is the maximum wavelength of the spectrum and  $BD$  is the band depth.

- *max*: Wavelength position of the maximum value observed in the feature.
- Parameters based on half-max values:
  - *lo* and *up*: Wavelength position of the *lower* and *upper* half-max value.
  - *width*: Difference between wavelength positions of *upper* and *lower* half-max values.
  - *gauss\_lo*: Similarity of the Gauss distribution function and the feature values between the *lower* half-max and the *maximum* position. As similarity measurement, the root mean square error is calculated.
  - *gauss\_up*: Same as above but for feature values between the *maximum* position and the *upper* half-max.

### Value

A matrix containing feature area values.

### Author(s)

Hanna Meyer & Lukas Lehnert

### See Also

[define.features](#), [specfeat](#)

### Examples

```
data(spectral_data)

## Example calculating the areas of the features around 450nm,
## 700nm, 1200nm and 1500nm.
bd <- transform_speclib(subset(spectral_data, Site == "Namco"),
                        method = "sh", out = "bd")

## Define features
features <- define.features(bd)

## Convert speclib to specfeat giving center wavelength of features
featureSelection <- specfeat(features, c(450,700,1200,1500))

## Calculate properties of features
featureProp <- feature_properties(featureSelection)
```

---

get.gaussian.response *Gaussian response function*

---

### Description

Simulate Gaussian response function for satellite sensor

### Usage

```
get.gaussian.response(fwhm)
```

### Arguments

fwhm                    Object of class `data.frame` with three columns. See details and examples sections.

### Details

The characteristics of the sensor must be passed as a `data.frame` with three columns: first column is used as name for bands, second with lower bounds of channels and third column with upper bounds. Alternatively, the `data.frame` may encompass band centre wavelength and full-width-half-maximum values of the sensor. Function will check the kind of data passed by partially matching the names of the data frame: If any column is named "fwhm" or "center", it is assumed that data are band centre and full-width-half-maximum values.

### Value

Data frame with response values for all bands covering the entire spectral range of satellite sensor.

### Author(s)

Lukas Lehnert

### See Also

[get.sensor.characteristics](#), [get.gaussian.response](#)

### Examples

```
par(mfrow=c(1,2))
## Plot response function of RapidEye
plot(c(0,1)~c(330,1200), type = "n", xlab = "Wavelength [nm]",
     ylab = "Spectral response")
data_RE <- get.gaussian.response(get.sensor.characteristics("RapidEye"))
xwl_response <- seq.int(attr(data_RE, "minwl"),
                       attr(data_RE, "maxwl"),
                       attr(data_RE, "stepsize"))
for (i in 1:ncol(data_RE))
  lines(xwl_response, data_RE[,i], col = i)
```

```

## Plot original response function
data_RE <- get.sensor.characteristics("RapidEye", TRUE)

plot(c(0,1)~c(330,1200), type = "n", xlab = "Wavelength [nm]",
      ylab = "Spectral response")
xwl_response <- seq.int(attr(data_RE$response, "minwl"),
                       attr(data_RE$response, "maxwl"),
                       attr(data_RE$response, "stepsize"))
for (i in 1:nrow(data_RE$characteristics))
  lines(xwl_response, data_RE$response[,i], col = i)

## Simulate gaussian response for arbitrary sensor with 3 bands
sensor <- data.frame(Name = paste("Band_", c(1:3), sep = ""),
                    center = c(450, 570, 680),
                    fwhm = c(30, 40, 30))

## Plot response function
par(mfrow=c(1,1))
plot(c(0,1)~c(330,800), type = "n", xlab = "Wavelength [nm]",
      ylab = "Spectral response")
data_as <- get.gaussian.response(sensor)
xwl_response <- seq.int(attr(data_as, "minwl"),
                       attr(data_as, "maxwl"),
                       attr(data_as, "stepsize"))
for (i in 1:3)
  lines(xwl_response, data_as[,i], col = i)

```

---

get.response

*Satellite response functions*


---

## Description

Handling satellite sensor response functions

## Usage

```
get.response(sensor, range = NULL, response_function = TRUE,
            continuousdata = "auto")
```

## Arguments

sensor	Name or integer value of satellite sensor. Matched with <a href="#">get.sensor.name</a> .
range	Vector of length = 2 containing maximum and minimum wavelength to be considered.
response_function	If TRUE, spectral response function else wise Gaussian response function will be returned.
continuousdata	Definition if returned <a href="#">Speclib</a> is containing continuous data or not.

**Value**

Object of class [Speclib](#) containing spectral response values instead of reflectance value. Spectral response values may be accessed with [spectra](#).

**Author(s)**

Lukas Lehnert

**See Also**

[get.sensor.name](#), [get.sensor.characteristics](#), [get.gaussian.response](#)

**Examples**

```
## See example in get.sensor.characteristics
```

---

```
get.sensor.characteristics
```

*Sensor characteristics*

---

**Description**

Get channel wavelength of satellite sensor

**Usage**

```
get.sensor.characteristics(sensor, response_function = FALSE)
```

**Arguments**

sensor	Character or integer. Name or numerical abbreviation of sensor. See 'sensor="help"' or 'sensor=0' for an overview of available sensors.
response_function	If TRUE, the spectral response function is returned

**Author(s)**

Lukas Lehnert

**See Also**

[spectral.resampling](#)



## Examples

```
## Return implemented sensors
get.sensor.characteristics(0)

## RapidEye
data_wv <- get.sensor.characteristics("RapidEye", TRUE)

## Plot response functions
plot(c(0,1)~c(330,1200), type = "n", xlab = "Wavelength [nm]",
      ylab = "Spectral response")
xwl_response <- seq.int(attr(data_wv$response, "minwl"),
                       attr(data_wv$response, "maxwl"),
                       attr(data_wv$response, "stepsize"))
for (i in 1:nrow(data_wv$characteristics))
  lines(xwl_response, data_wv$response[,i], col = i)
```

---

get.sensor.name	<i>Satellite sensor name</i>
-----------------	------------------------------

---

## Description

Get satellite sensor name by integer value

## Usage

```
get.sensor.name(sensor)
```

## Arguments

sensor            Integer value to match against predefined satellite sensors.

## Details

See [get.sensor.characteristics](#) to get overview on available satellite sensors.

## Value

Name of satellite sensor as character string.

## Author(s)

Lukas Lehnert

## See Also

[get.sensor.characteristics](#)

## Examples

```
get.sensor.name(1)
```

---

`getcp`*Get fix points*

---

**Description**

Get fix points of continuum line within spectral range.

**Usage**

```
getcp(x, ispec, subset = NULL)
```

**Arguments**

<code>x</code>	Object of class <code>Clman</code> .
<code>ispec</code>	ID or index of spectrum to be analysed.
<code>subset</code>	Vector of length = 2 giving the lower and upper limit of spectral range.

**Value**

Object of class `list` containing two elements:

- `ptscon`: Data frame with wavelength and reflectance of fix points
- `ispec`: Index of analysed spectrum within passed `Clman`-object.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [deletecp](#), [addcp](#), [Clman](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Fix points
spec_cp <- getcp(spec_clman, 1, c(400, 800))
spec_cp
```

---

getNRI	<i>Return nri-values</i>
--------	--------------------------

---

### Description

Return normalized ratio index values giving the wavelength

### Usage

```
getNRI(nri, wavelength)
```

### Arguments

nri	Object of class 'Nri'
wavelength	Wavelength values where nri is returned. See details section.

### Details

Wavelength can be passed in three ways. As the result of [nri\\_best\\_performance](#), as a data frame with two columns or as a vector of length 2. In the first two cases, the result will be a data frame (if data frames contain more than one row) with the nri-values of each pair of wavelengths. In the latter case it will be a vector.

### Author(s)

Lukas Lehnert

### See Also

[nri](#), [Nri](#)

### Examples

```
data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Build glm-models
glmnri <- glm.nri(nri_WV ~ Soil.moisture, predata = spec_WV)

## Return best 5 models
BM <- nri_best_performance(glmnri, n = 5, coefficient = "p.value")

## Get nri values for the 5 models
nri_BM <- getNRI(nri_WV, BM)
```

---

get_reflectance	<i>Get reflectance values</i>
-----------------	-------------------------------

---

**Description**

Returns weighted or unweighted reflectance values at wavelength position.

**Usage**

```
## S4 method for signature 'Speclib'  
get_reflectance(spectra, wavelength, position, weighted = FALSE,  
                ...)
```

**Arguments**

spectra	Object of class <code>Speclib</code> or <code>data.frame</code> with reflectance values.
wavelength	Vector with wavelength values.
position	Numeric value passing the position of reflectance values to be returned in dimensions of the wavelength values.
weighted	Logical indicating if reflectance values should be interpolated to fit wavelength position. If <code>FALSE</code> the reflectance values of nearest neighbour to passed position are returned.
...	Arguments to be passed to specific functions. For <code>get_reflectance.default</code> ignored.

**Value**

A vector with reflectance values for each spectrum is returned. If position falls outside of spectral range of input values, NA values are returned.

**Author(s)**

Lukas Lehnert & Hanna Meyer

**See Also**

[spectra](#)

**Examples**

```
data(spectral_data)
```

---

glm.nri *(Generalised) Linear models from normalised ratio indices*

---

## Description

Build (generalised) linear models of normalised ratio indices as response and predictor variables

## Usage

```
lm.nri(formula, preddata = NULL, ...)  
glm.nri(formula, preddata = NULL, ...)
```

## Arguments

formula	Formula for (generalized) linear model
preddata	Data frame or speclib containing predictor variables
...	Further arguments passed to <code>lm</code> , <code>glm</code> and generic <code>print.default</code>

## Details

NRI-values may be used as predictor or response variable. If NRI-values are predictors, the models are build only with one index as predictor instead of all available indices. In this case, only one predictor and one response variable is currently allowed. See help pages for `lm` and `glm` for any additional information. Note that this function does not store the entire information returned from a normal (g)lm-model. To get full (g)lm-models use either the function `nri_best_performance` to return best performing model(s) or extract nri-values with `getNRI` and build directly the model from respective index.

See details in `Nri-plot`-method for information about plotting.

## Value

The function returns an object of class `Nri`. The list in the slot `multivariate` contains the new (g)lm information which depends on the kind of model which is applied:

1. `lm.nri`: The list contains the following items:
  - Estimate: Coefficient estimates for each index and term
  - Std.Error: Standard errors
  - t.value: T-values
  - p.value: P-values
  - r.squared:  $R^2$  values
2. `glm.nri`: The list contains the following items (depending on formula used):
  - Estimate: Coefficient estimates for each index and term
  - Std.Error: Standard errors
  - t.value/z.value: T-values or Z-values
  - p.value: P-values

**Author(s)**

Lukas Lehnert

**See Also**

[plot](#), [lm](#), [glm](#), [getNRI](#)

**Examples**

```
data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

glmnri <- glm.nri(nri_WV ~ Soil.moisture, preddata = spec_WV)
glmnri

plot(glmnri)
```

---

hsdardocs

*Load additional documents*

---

**Description**

Access help documents and references for different methods.

**Usage**

```
hsdardocs(doc)
```

**Arguments**

doc	Name of document to load. Currently, only "References.pdf" and "Copyright" are available
-----	--

**Author(s)**

Lukas Lehnert

**Examples**

```
## Not run:
## Open references of hyperspectral vegetation indices (PDF-file)
hsdardocs("References.pdf")

## See copyrights of routines and data used in hsdar-package (ascii-file)
hsdardocs("Copyright")
```

```
## End(Not run)
```

---

HyperSpecRaster      *Handle hyperspectral cubes using raster package*

---

## Description

Methods to create and handle objects of class HyperSpecRaster

## Usage

```
## S4 method for signature 'character,numeric'  
HyperSpecRaster(x, wavelength, fwhm = NULL, attributes = NULL, ...)
```

```
## S4 method for signature 'RasterLayer,numeric'  
HyperSpecRaster(x, wavelength, fwhm = NULL, attributes = NULL)
```

```
## S4 method for signature 'RasterBrick,numeric'  
HyperSpecRaster(x, wavelength, fwhm = NULL, attributes = NULL)
```

```
## S4 method for signature 'HyperSpecRaster'  
HyperSpecRaster(x, wavelength)
```

```
## S4 method for signature 'Speclib,ANY'  
brick(x, nrow, ncol, xmn, xmx, ymn, ymx, crs)
```

```
## S4 method for signature 'Speclib,ANY'  
HyperSpecRaster(x, nrow, ncol, xmn, xmx, ymn, ymx, crs)
```

```
## S4 method for signature 'HyperSpecRaster,character'  
writeStart(x, filename, ...)
```

```
## S4 method for signature 'HyperSpecRaster'  
getValuesBlock(x, ...)
```

```
## S4 method for signature 'RasterLayer,Speclib'  
writeValues(x, v, start)
```

```
## S4 method for signature 'RasterBrick,Speclib'  
writeValues(x, v, start)
```

```
## S4 method for signature 'HyperSpecRaster,Speclib'  
writeValues(x, v, start)
```

**Arguments**

x	Raster* object
wavelength	Vector containing wavelength for each band
fwhm	Optional vector containing full-width-half-max values. If length == 1 the same value is assumed for each band. Note that function does not check the integrity of the values
attributes	Optional data.frame containing attributes data
nrow	Optional. Number of rows in HyperSpecRaster. If omitted, function will try to get the information from the attributes in Speclib (attr(x, "rastermeta"))
ncol	Optional. Number of columns in HyperSpecRaster. See nrow above.
xmn	Optional. Minimum coordiante in x-dimension. See nrow above.
xmx	Optional. Maximum coordiante in x-dimension. See nrow above.
ymn	Optional. Minimum coordiante in y-dimension. See nrow above.
ymx	Optional. Maximum coordiante in y-dimension. See nrow above.
crs	Optional. Object of class 'CRS' giving the coordinate system for HyperSpecRaster. See nrow above.
...	Additional arguments as for <a href="#">brick</a>
filename	Name of file to create
v	Speclib or matrix of values
start	Integer. Row number (counting starts at 1) from where to start writing v

**Value**

HyperSpecRaster or RasterBrick

**Author(s)**

Lukas Lehnert

**Examples**

```
## Not run:
## Create raster file using PROSAIL
## Run PROSAIL
parameter <- data.frame(N = c(rep.int(seq(0.5, 1.4, 0.1), 6)),
                        LAI = c(rep.int(0.5, 10), rep.int(1, 10),
                                rep.int(1.5, 10), rep.int(2, 10),
                                rep.int(2.5, 10), rep.int(3, 10)))
spectra <- PROSAIL(parameterList = parameter)

## Create SpatialPixelsDataFrame and fill data with spectra from PROSAIL
rows <- round(nspectra(spectra)/10, 0)
cols <- ceiling(nspectra(spectra)/rows)
grd <- SpatialGrid(GridTopology(cellcentre.offset = c(1,1,1),
```



```

                                cellsize = c(1,1,1),
                                cells.dim = c(cols, rows, 1)))
x <- SpatialPixelsDataFrame(grd, data = as.data.frame(spectra(spectra)))

## Write data to example file (example_in.tif) in workingdirectory
writeGDAL(x, fname = "example_in.tif", drivename = "GTiff")

## Examples for HyperSpecRaster using file example_in.tif
## Example 1:
## smoothing spectra
infile <- "example_in.tif"
outfile <- "example_result_1.tif"
wavelength <- spectra$wavelength

ra <- HyperSpecRaster(infile, wavelength)
tr <- blockSize(ra)

res <- writeStart(ra, outfile, overwrite = TRUE)
for (i in 1:tr$n)
{
  v <- getValuesBlock(ra, row=tr$row[i], nrows=tr$nrows[i])
  v <- smoothSpeclib(v, method="sgolay", n=25)
  res <- writeValues(res, v, tr$row[i])
}
res <- writeStop(res)

## Example 2:
## masking spectra and calculating vegetation indices
outfile <- "example_result_2.tif"
n_veg <- as.numeric(length(vegindex()))
res <- writeStart(ra, outfile, overwrite = TRUE, nl = n_veg)
for (i in 1:tr$n)
{
  v <- getValuesBlock(ra, row=tr$row[i], nrows=tr$nrows[i])
  mask(v) <- c(1350, 1450)
  v <- as.matrix(vegindex(v, index=vegindex()))
  res <- writeValues(res, v, tr$row[i])
}
res <- writeStop(res)

## End(Not run)

```

---

HyperSpecRaster-class *HyperSpecRaster\** class

---

## Description

Extension of *\*RasterBrick*-class to handle hyperspectral data

**Details**

Extension of `*RasterBrick`-class with three additional slots:

`wavelength`: A numeric vector giving the center wavelength for each band.

`fwhm` (**optional**): A numeric vector giving the full-width-half-max values for each band.

`attributes` (**optional**): A `data.frame` containing additional information for each pixel.

The information in the three slots are used for the conversion to [Speclib](#).

**Author(s)**

Lukas Lehnert

**See Also**

[brick](#), [Speclib](#)

---

idSpeclib

*Handling IDs of spectra*

---

**Description**

Returning and setting ID of spectra in `Speclib`

**Usage**

```
idSpeclib(x)
idSpeclib(x) <- value
```

**Arguments**

<code>x</code>	Object of class <code>Speclib</code> .
<code>value</code>	Character vector of the same length as <code>nspectra(x)</code> , or <code>NULL</code> .

**Value**

For `idSpeclib<-`, the updated object. Otherwise a vector giving the ID of each spectrum in `Speclib` is returned.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#)

**Examples**

```
data(spectral_data)

idSpeclib(spectral_data)
```

---

**makehull***Check continuum line*

---

**Description**

Check if continuum line is intersecting the reflectance curve.

**Usage**

```
makehull(x, ispec)
```

**Arguments**

x	Object of class Clman.
ispec	Name or index of spectrum to be checked.

**Value**

Object of class list.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [addcp](#), [deletecp](#), [makehull](#), [updatecl](#)  
[Clman](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Add fix point at 4595 nm to continuum line of first spectrum
```

```

spec_clman <- addcp(spec_clman, 1, 2495)

## Plot new line
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error

## Add fix point at 4596 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2496)

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error

hull <- makehull(spec_clman, 1)

## Transform spectra using band depth
spec_bd <- transform_speclib(spec, method = "sh", out = "bd")

## Update continuum line of first spectrum
spec_bd <- updatecl(spec_bd, hull)

## Plot modified transformed spectrum
plot(spec_bd, FUN = 1)

```

---

mask

*Mask spectra*


---

## Description

Returning and setting mask of spectra in Speclib. `interpolate.mask` linearly interpolates masked parts in spectra.

## Usage

```

## S4 method for signature 'Speclib'
mask(object)
## S4 replacement method for signature 'Speclib,data.frame'
mask(object) <- value
## S4 replacement method for signature 'Speclib,list'
mask(object) <- value
## S4 replacement method for signature 'Speclib,numeric'
mask(object) <- value

## Linear interpolation of masked parts
interpolate.mask(object)

```

## Arguments

object	Object of class <code>Speclib</code> .
value	Numeric vector, data frame or list giving the mask boundaries in wavelength units. See details section.

## Details

Value may be an object of class vector, data frame or list. Data frames must contain 2 columns with the first column giving the lower and the second the upper boundary values of the mask. List must have two items consisting of vectors of length = 2. The first entry is used as lower and the second as upper boundary values. Vectors must contain corresponding lower and upper boundary values consecutively.

Interpolation of masked parts is mainly intended for internal use. Interpolation is only possible if mask does not exceed spectral range of `Speclib`.

## Value

For `mask<-`, the updated object. Otherwise a data frame giving the mask boundaries.  
`interpolate.mask` returns a new object of class `Speclib`.

## Author(s)

Lukas Lehnert and Hanna Meyer

## See Also

[Speclib](#)

## Examples

```
data(spectral_data)

mask(spectral_data) ## NULL

## Mask from vector
spectral_data_ve <- spectral_data
mask(spectral_data_ve) <- c(1040,1060,1300,1450)
mask(spectral_data_ve)

## Mask from data frame
spectral_data_df <- spectral_data
mask(spectral_data_df) <- data.frame(lb=c(1040,1300),ub=c(1060,1450))
mask(spectral_data_df)

## Mask from list
spectral_data_li <- spectral_data
mask(spectral_data_li) <- list(lb=c(1040,1300),ub=c(1060,1450))
```

```
mask(spectral_data_li)

## Linear interpolation
plot(spectral_data)
plot(interpolate.mask(spectral_data_li), new=FALSE)
```

---

meanfilter

*Apply mean filter*

---

### Description

Apply mean filter to data frame with spectra as rows and bands as columns. Filter size is passed as number of bands averaged at both sites of the respective band value.

### Usage

```
meanfilter(spectra, p = 5)
```

### Arguments

spectra	Data frame containing spectra
p	Filter size.

### Value

Filtered data frame of same dimension as input data frame

### Author(s)

Lukas Lehnert

### See Also

[smoothSpeclib](#)

### Examples

```
data(spectral_data)

spectra_filtered <- meanfilter(spectra(spectral_data), p = 10)
spectra(spectral_data) <- spectra_filtered
```

---

merge	<i>Merge speclibs</i>
-------	-----------------------

---

**Description**

Merge 2 Speclibs and their attributes data

**Usage**

```
## S4 method for signature 'Speclib,Speclib'  
merge(x, y, ...)
```

**Arguments**

x	1st Object of class Speclib to be merged.
y	2nd Object of class Speclib to be merged.
...	Further arguments passed to generic functions. Currently ignored.

**Value**

Object of class Speclib.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#)

**Examples**

```
data(spectral_data)  
sp1 <- spectral_data[c(1:10),]  
sp2 <- spectral_data[c(11:20),]  
  
speclib_merged <- merge(sp1, sp2)
```

---

nri *Normalised ratio index*

---

### Description

Calculate normalised ratio index for a single given band combination or for all possible band combinations

### Usage

```
nri(x, b1, b2, recursive = FALSE, bywavelength = TRUE)
```

### Arguments

x	List of class <code>Speclib</code> or of class <code>Nri</code> for print and <code>as.matrix</code> methods
b1	Band 1 given as index or wavelength
b2	Band 2 given as index or wavelength
recursive	If TRUE indices for all possible band combinations are calculated
bywavelength	Flag to determine if b1 and b2 are indices ( <code>bywavelength = FALSE</code> ) or wavelength ( <code>bywavelength = TRUE</code> )
...	Further arguments passed to generic functions. Currently ignored.

### Details

Function performs the following calculation:

$$nri_{B1, B2} = \frac{R_{B1} - R_{B2}}{R_{B1} + R_{B2}}$$

with  $R$  being reflectance values at wavelength  $B1$  and  $B2$ , respectively.

If `recursive = TRUE`, all possible band combinations are calculated.

### Value

If `recursive = FALSE`, a data frame with index values is returned. Otherwise result is an object of class `nri`. See `glm.nri` for applying a generalised linear model to an array of normalised ratio indices.

### Author(s)

Lukas Lehnert



## References

Sims, D.A.; Gamon, J.A. (2002). Relationships between leaf pigment content and spectral reflectance across a wide range of species, leaf structures and developmental stages. *Remote Sensing of Environment*: 81/2, 337 - 354.

Thenkabail, P.S.; Smith, R.B.; Pauw, E.D. (2000). Hyperspectral vegetation indices and their relationships with agricultural crop characteristics. *Remote Sensing of Environment*: 71/2, 158 - 182.

## See Also

[glm.nri](#), [glm](#), [Speclib](#), [Nri](#)

## Examples

```
data(spectral_data)

## Calculate NDVI
ndvi <- nri(spectral_data, b1=800, b2=680)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)
nri_WV
```

---

Nri-class

\* *Nri class*

---

## Description

Class to handle datasets containing normalized ratio indices of spectra.

## Details

Object with slots:

- nri: Object of class [DistMat3D](#) containing nri values.
- fwhm: Vector or single numerical value giving the full-width-half-max value(s) for each band.
- wavelength: Vector with wavelength information.
- dimnames: Character vector containing band names used to calculate nri-values.
- multivariate: List defining the kind of test/model applied to the data and the model data. Only used after object has passed e.g. [\(g\)lm.nri](#).

## Note

See figure in [hsdar-package](#) for an overview of classes in hsdar.

**Author(s)**

Lukas Lehnert

**See Also**[Speclib](#)

---

**Nri-methods***Methods for \*Nri-class*

---

**Description**

Methods to handle data in objects of class Nri.

**Usage**

```
## S4 method for signature 'Nri'  
as.matrix(x, ..., named_matrix = TRUE)
```

```
## S4 method for signature 'Nri'  
as.data.frame(x, ...)
```

```
## S4 method for signature 'Nri'  
wavelength(object)
```

**Arguments**

<code>x, object</code>	List of class 'Nri'
<code>named_matrix</code>	Flag if column and row names are set to band indices used for the calculation of the nri-values.
<code>...</code>	Further arguments passed to generic functions. Currently ignored.

**Author(s)**

Lukas Lehnert

**See Also**[glm.nri](#), [glm](#)

---

nri\_best\_performance *Best performing model(s) with NRI*

---

### Description

Get or mark best performing model(s) between narrow band indices and environmental variables

### Usage

```
nri_best_performance(nri, n = 1, coefficient = "p.value",
                    predictor = 2, abs = FALSE, findMax = FALSE,
                    ...)
mark_nri_best_performance(best, glmnri, n = nrow(best$Indices),
                        upperdiag = FALSE, ...)
```

### Arguments

nri	Object of class nri
glmnri	Object of class glmnri
n	Number of models to return or mark
coefficient	Name or index of coefficient to plot
predictor	Name or index of term to plot
abs	Use absolute value (e.g. for t-values)
findMax	Find maximum or minimum values
best	Output from nri_best_performance
upperdiag	Flag to return the upper diagonal
...	Further arguments passed to <a href="#">glm</a> function. These must be the same as used for initial creation of <a href="#">glm.nri</a> . For mark_nri_best_performance arguments are passed to <a href="#">polygon</a> .

### Details

See details in [glm.nri](#) and [glm](#).

### Author(s)

Lukas Lehnert

### See Also

[glm.nri](#), [glm](#)

**Examples**

```

data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Build glm-models
glmnri <- glm.nri(nri_WV ~ Soil.moisture, predata = spec_WV)

## Return best 5 models
BM <- nri_best_performance(glmnri, n = 5, coefficient = "p.value")

## Get nri values for the 5 models
nri_BM <- getNRI(nri_WV, BM)

```

---

plot.Nri

*Plot function for (g)lm.nri and cor.test.nri*


---

**Description**

Plot values in (generalised) linear modes and correlation tests from narrow band indices

**Usage**

```

## S4 method for signature 'Nri'
plot(x, coefficient = "p.value", predictor = 2,
     xlab = "Wavelength band 1 (nm)",
     ylab = "Wavelength band 2 (nm)", legend = TRUE,
     colspace = "hcl", col = c(10, 90, 60, 60, 10, 80),
     digits = 2, range = "auto", constraint = NULL,
     upperdiag = FALSE, ...)

```

**Arguments**

x	Object to be plotted.
coefficient	Name or index of coefficient to plot.
predictor	Name or index of term to plot.
xlab	Label for x-axis.
ylab	Label for y-axis.
legend	Flag if legend is plotted. If legend == "outer" the legend is plotted in the outer margins of the figure. This is useful if both diagonals are used.
colspace	Either "hcl" or "rgb". Colour space to be used for the plots.

col	If colspace == "hcl", the vector is giving the minimum and maximum values of hue (element 1 & 2), chroma (element 3 & 4) and luminance (element 5 & 6). The optional element 7 is used as alpha value. See <a href="#">hcl</a> for further explanation. If colspace == "rgb", a vector of length >=2 giving the colours to be interpolated using <a href="#">colorRamp</a> .
digits	Precision of labels in legend.
range	"auto" or a vector of length = 2 giving the range of values to be plotted.
constraint	A character string giving a constraint which values should be plotted. See examples section.
upperdiag	Flag if upper diagonal is used for the plot. Note that if TRUE the current plot is used instead of starting a new plot
...	Further arguments passed to plot.default.

### Details

See details in [glm.nri](#) and [glm](#).

### Value

An invisible vector with minimum and maximum values plotted.

### Author(s)

Lukas Lehnert

### See Also

[nri](#), [glm.nri](#), [glm](#), [cor.test](#), [t.test](#)

### Examples

```
## Not run:
data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Fit generalised linear models between NRI-values and soil
## moisture
glmnri <- glm.nri(nri_WV ~ Soil.moisture, predata = spec_WV)

## Plot p-values
plot(glmnri, range = c(0, 0.05))
## Plot t-values
plot(glmnri, coefficient = "t.value")
## Plot only t-values where p-values < 0.001
plot(glmnri, coefficient = "t.value",
      constraint = "p.value < 0.001")
```

```

## Fit linear models between NRI-values and soil moisture
lmnri <- lm.nri(nri_WV ~ Soil.moisture, preddata = spec_WV)

## Plot r.squared
plot(lmnri)

## Example for EnMAP (Attention: Calculation time may be long!)
spec_EM <- spectral.resampling(spectral_data, "EnMAP",
                              response_function = FALSE)
mask(spec_EM) <- c(300, 550, 800, 2500)
nri_EM <- nri(spec_EM, recursive = TRUE)
glmri <- glm.nri(nri_EM ~ Soil.moisture, preddata = spec_EM)

## Plot T values in lower and p-values in upper diagonal
## of the plot
## Enlarge margins for legends
par(mar = c(5.1, 4.1, 4.1, 5))
plot(glmri, coefficient = "t.value", legend = "outer")
plot(glmri, coefficient = "p.value", upperdiag = TRUE)
lines(c(400,1705),c(400,1705))

## End(Not run)

```

---

plot.Specfeat

*Plot function for class Specfeat*


---

## Description

Plot spectra in Specfeat objects

## Usage

```

## S4 method for signature 'Specfeat,ANY'
plot(x, fnumber = 1, stylebysubset = NULL, changecol = TRUE,
      changetype = FALSE, autolegend = TRUE, new = TRUE, ...)

```

## Arguments

x	Object to be plotted
fnumber	Subscript of feature(s) to be plotted
stylebysubset	Name of column in attributes to be used for colour.
changecol	Flag indicating if line colours change according to values in coloumn defined by stylebysubset
changetype	Flag indicating if line types change according to values in coloumn defined by stylebysubset
autolegend	Flag if legend is plotted.
new	Flag if a new plot is started.
...	Further arguments passed to plot.default

**Author(s)**

Lukas Lehnert

**See Also**[nri](#), [glm.nri](#), [glm](#), [cor.test](#), [Nri-method](#), [t.test](#), [Nri-method](#)**Examples**

```
## See examples in specfeat
```

---

`plot.Speclib`*Plot speclib*

---

**Description**

Plot Speclib in a new plot or adding it to an existing plot.

**Usage**

```
## S4 method for signature 'Speclib,ANY'
plot(x, FUN = NULL, new = TRUE, ...)

## S4 method for signature 'Clman,ANY'
plot(x, ispec, subset = NULL, numeratepoints = TRUE, ...)
```

**Arguments**

<code>x</code>	Object of class <code>Speclib</code> .
<code>FUN</code>	Name of a function (character) or index or ID of single spectrum to plot (integer).
<code>new</code>	If <code>FALSE</code> the plot is added to active existing plot.
<code>ispec</code>	Subscript of spectrum to be plotted.
<code>subset</code>	Vector of length = 2 containing minimum and maximum wavelength to plot.
<code>numeratepoints</code>	Flag if continuum points are numerated and labeled.
<code>...</code>	Further arguments passed to internal plot functions or to <a href="#">plot</a> for objects of class <code>Speclib</code> and <code>Clman</code> .

## Details

The function may work in a couple of modes. The default way is to plot mean values (solid line) of all spectra and the standard deviations within bands. If data is assumed to be continuous the standard deviations are plotted as dashed lines otherwise error bars will indicate standard deviations.

The user has various options to change the way things are looking: With argument FUN the name of a function, the ID or the index of a certain spectrum may be specified. Note that if FUN is a function, this function will be applied to all spectra. If function should be applied to a subset of spectra, use function [subset](#) to define rules excluding certain spectra.

By passing a subset, the user may specify a spectral range to plot. Limits for x- and y-axis will be found automatically or may be passed separately.

## Author(s)

Lukas Lehnert

## See Also

[Speclib](#)

## Examples

```
data(spectral_data)

## Set mask for channel crossing and water absorption bands
mask(spectral_data) <- c(1040, 1060, 1350, 1450)

## Simple example
plot(spectral_data, legend = list(x = "topleft"))

## Example with groups
plot(spectral_data, bygroups = TRUE, legend = list(x = "topleft"))

## Example with function
par(mfrow = c(2,3))
plot(spectral_data, FUN = "min", main = "Minimum of speclib")
plot(spectral_data, FUN = "max", main = "Maximum of speclib")
plot(spectral_data, FUN = "median", main = "Median of speclib")
plot(spectral_data, FUN = "mean", main = "Mean of speclib")
plot(spectral_data, FUN = "var", main = "Variance of speclib")
```

---

PROSAIL

*Simulate canopy spectrum*

---

## Description

Simulate a canopy spectrum using PROSAIL 5B



**Usage**

```
PROSAIL(N = 1.5, Cab = 40, Car = 8, Cbrown = 0.0,
        Cw = 0.01, Cm = 0.009, psoil = 0, LAI = 1,
        TypeLidf = 1, lidfa = -0.35, lidfb = -0.15,
        hspot = 0.01, tts = 30, tto = 10, psi = 0,
        parameterList = NULL)
```

**Arguments**

N	Structure parameter
Cab	Chlorophyll content
Car	Carotenoid content
Cbrown	Brown pigment content
Cw	Equivalent water thickness
Cm	Dry matter content
psoil	Dry/Wet soil factor
LAI	Leaf area index
TypeLidf	Type of leaf angle distribution. See details section
lidfa	Leaf angle distribution. See details section
lidfb	Leaf angle distribution. See details section
hspot	Hotspot parameter
tts	Solar zenith angle
tto	Observer zenith angle
psi	Relative azimuth angle
parameterList	An optional object of class 'data.frame'. Function will iterate over rows of parameterList setting missing entries to default values. See examples section.

**Details**

This function uses the FORTRAN code of PROSAIL model (Version 5B). For a general introduction see following web page and the links to articles provided there:

<http://teledetection.ipgp.jussieu.fr/prosail/>

The following table summarises the abbreviations of parameters and gives their units as used in PROSAIL. Please note that default values of all parameters were included with the intention to provide an easy access to the model and should be used with care in any scientific approach!

Parameter	Description of parameter	Units
N	Leaf structure parameter	NA
Cab	Chlorophyll a+b concentration	$\mu\text{g}/\text{cm}^2$
Car	Carotenoid concentration	$\mu\text{g}/\text{cm}^2$
Caw	Equivalent water thickness	cm

Cbrown	Brown pigment	NA
Cm	Dry matter content	g/cm <sup>2</sup>
LAI	Leaf Area Index	NA
psoil	Dry/Wet soil factor	NA
hspot	Hotspot parameter	NA
tts	Solar zenith angle	deg
tto	Observer zenith angle	deg
psi	Relative azimuth angle	deg

Functions for distribution of leaf angles within the canopy may work in two modes, which is controlled via TypeLidf:

1. TypeLidf == 1 (default): lidfa is the average leaf slope and lidfb describes bimodality of leaf distribution. The following list gives an overview on typical settings:

LIDF type	lidfa	lidfb
Planophile	1	0
Erectophile	-1	0
Plagiophile	0	-1
Extremophile	0	1
Spherical (default)	-0.35	-0.15

2. TypeLidf != 1: lidfa is the average leaf angle in degree (0 = planophile / 90 = erectophile); lidfb is 0

### Value

An object of class SpecLib. If parameterList is used, the parameter are stored in attributes table of SpecLib.

### Note

The function is based on the FORTRAN version of the PROSAIL-code initially developed by Stephane JACQUEMOUD, Jean-Baptiste FERET, Christophe FRANCOIS and Eben BROADBENT. SAIL component has been developed by Wout VERHOEF.

### Author(s)

Lukas Lehnert

### References

Jacquemoud, S., Verhoef, W., Baret, F., Bacour, C., Zarco-Tejada, P.J., Asner, G.P., Francois, C., and Ustin, S.L. (2009): PROSPECT + SAIL models: a review of use for vegetation characterization, Remote Sensing of Environment, 113, S56-S66.

**See Also**

[PROSPECT](#), [SpecLib](#)

**Examples**

```
## Single spectrum
spectrum <- PROSAIL(N = 1.3)
plot(spectrum)

## Example using parameterList
## Test effect of leaf structure and LAI on spectra
parameter <- data.frame(N = c(rep.int(seq(0.5, 1.5, 0.5), 2)),
  LAI = c(rep.int(0.5, 3), rep.int(1, 3)))
spectra <- PROSAIL(parameterList = parameter)

## Print attributes table
attribute(spectra)

## Plot spectra
#plot(subset(spectra, LAI == 0.5), col = "red", ylim = c(0, 0.3))
#plot(subset(spectra, LAI == 1), col = "green", new = FALSE)
```

---

PROSPECT

*Simulate plant spectrum*

---

**Description**

Simulate plant spectrum using PROSPECT 5

**Usage**

```
PROSPECT(N = 1.5, Cab = 40, Car = 8, Cbrown = 0.0,
  Cw = 0.01, Cm = 0.009, transmittance = FALSE,
  parameterList = NULL)
```

**Arguments**

N	Structure parameter
Cab	Chlorophyll content
Car	Carotenoid content
Cbrown	Brown pigment content
Cw	Equivalent water thickness
Cm	Dry matter content
transmittance	Logical flag, if transmittance instead of reflectance values are returned.
parameterList	An optional object of class 'data.frame'. Function will iterate over rows of parameterList setting missing entries to default values. See examples section.

## Details

This function uses the FORTRAN code of PROSPECT model (Version 5). For a general introduction see following web page and the links to articles provided there:

<http://teledetection.ipgp.jussieu.fr/prosail/>

The following table summarises the abbreviations of parameters and gives their units as used in PROSPECT. Please note that default values of all parameters were included with the intention to provide an easy access to the model and should be used with care in any scientific approach!

Parameter	Description of parameter	Units
N	Leaf structure parameter	NA
Cab	Chlorophyll a+b concentration	$\mu\text{g}/\text{cm}^2$
Car	Carotenoid concentration	$\mu\text{g}/\text{cm}^2$
Cw	Equivalent water thickness	cm
Cbrown	Brown pigment	NA
Cm	Dry matter content	$\text{g}/\text{cm}^2$

## Value

An object of class `SpecLib`.

## Note

The function is based on the FORTRAN version of the PROSPECT-code initially developed by Jean-Baptiste FERET, Stephane JACQUEMOUD and Christophe FRANCOIS.

## Author(s)

Lukas Lehnert

## References

Jacquemoud, S. and Baret, F. (1990). PROSPECT: A model of leaf optical properties spectra, *Remote Sensing of Environment* 34: 75 - 91.

## See Also

[PROSAIL](#), [SpecLib](#)

## Examples

```
## Single spectrum
spectrum <- PROSPECT(N = 1.3, Cab = 30, Car = 10, Cbrown = 0,
                    Cw = 0.01, Cm = 0.01)
plot(spectrum)

## Example using parameterList
```

```
## Test effect of leaf structure and chlorophyll content on
## spectra
parameter <- data.frame(N = c(rep.int(seq(0.5, 1.5, 0.5), 2)),
                        Cab = c(rep.int(40, 3), rep.int(20, 3)))
spectra <- PROSPECT(parameterList = parameter)

## Print attributes table
attribute(spectra)

## Plot spectra for range from 400 to 800 nm
#plot(subset(spectra, Cab == 20), col = "red", ylim = c(0, 0.5),
#      subset = c(400, 800))
#plot(subset(spectra, Cab == 40), col = "green", new = FALSE,
#      subset = c(400, 800))
```

---

Raster-methods

*Rasterbased methods for spectra*

---

## Description

Methods to manipulate and save spectra in Speclibs stored as RasterBrick

## Usage

```
## S4 method for signature 'Speclib,ANY'
extract(x, y, ...)

## S4 method for signature 'Speclib,character'
writeRaster(x, filename, ...)
```

## Arguments

x	Speclib with RasterBrick-object for spectra
y	Object of any valid type to define area to extract
filename	Output filename
...	Additionally arguments passed to basic funtions in the raster-package

## Value

Speclib

## Author(s)

Lukas Lehnert

---

rastermeta	<i>Create list containing rastermeta-information</i>
------------	--

---

**Description**

Create valid objects for slot rastermeta in [Speclib](#).

**Usage**

```
rastermeta(x, dim, ext, crs)
```

**Arguments**

x	Optional. Object of one of the following classes: "Raster", "RasterBrick", "RasterStack", "HyperSpecRaster".
dim	Optional. Vector with length == 2. The first and second elements give the number of rows and columns, respectively.
ext	Optional. Object of class extent.
crs	Optional. Object of class CRS.

**Value**

List with following elements (in exactly this order!):

- dim: Vector with length == 2. The first and second elements give the number of rows and columns, respectively.
- ext: Object of class extent.
- crs: Object of class CRS.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#), [HyperSpecRaster](#)

---

rededge	<i>Red edge parameter</i>
---------	---------------------------

---

### Description

Derive red edge parameters from hyperspectral data

### Usage

```
rededge(x, smooth = TRUE, round = FALSE, ...)
```

### Arguments

x	List of class <code>Speclib</code>
smooth	Logical indicating if spectral data should be smoothed. See details section.
round	Logical indicating if resulting wavelength position should be rounded.
...	Further arguments passed to <code>derivative.speclib</code>

### Details

Shape and location of the red edge are commonly described by four parameters:

- $R_0$ : minimum reflectance in the red spectrum
- $\lambda_0$ : wavelength of the minimum reflectance
- $\lambda_p$ : inflection point
- $R_s$ : shoulder wavelength

The red edge parameters are calculated as proposed in Bach (1995) from the spectral area between 550 and 900 nm.  $\lambda_0$  is calculated as the last root before the maximum value of the 2nd derivation. The minimum reflectance is the reflectance at ( $\lambda_0$ ). The inflection point is the root of the 2nd derivative function between the maximum value and the minimum value. The shoulder wavelength is the first root beyond the minimum value of the 2nd derivation.

Optional smoothing is performed with

```
smoothSpeclib(x, method = "spline", n = round(nbands(x)/10,0))
```

prior to all calculations. Note that reflectance values returned by the rededge-function are original values and not the smoothed reflectances. This would not be the case, if already smoothed reflectance values are passed to rededge-function.

### Value

A data frame containing parameters for each spectrum.

### Author(s)

Hanna Meyer

## References

Bach, H. (1995): Die Bestimmung hydrologischer und landwirtschaftlicher Oberflaechenparameter aus hyperspektralen Fernerkundungsdaten. Muenchner Geographische Abhandlungen Reihe B, Band B21.

## See Also

[vegindex](#), [derivative.speclib](#), [smoothSpeclib](#)

## Examples

```
data(spectral_data)
rd <- reledge(spectral_data)
boxplot(rd$R0 ~ attribute(spectral_data)$Site, ylab = "R0")
```

---

response\_functions      *Satellite sensor response functions*

---

## Description

Satellite sensor response functions for all sensor channels

## Format

An object of class 'data.frame'

## Details

Please do not access this data directly, since it contains only the response values without any spectral information. To get response functions use function [get.response](#), instead.

## Note

This data is kindly provided by operators of satellites. See [hsdardocs\("Copyright"\)](#) for copyright information on spectral response functions.

- Quickbird: Copyright by DigitalGlobe, Inc. All Rights Reserved
- RapidEye: Copyright by RapidEye AG
- WorldView-2: Copyright by DigitalGlobe, Inc. All Rights Reserved



---

`smgm`*SMGM*

---

**Description**

Calculate Gaussian model on soil spectra

**Usage**

```
smgm(x, percentage = TRUE, gridsize = 50)
```

**Arguments**

<code>x</code>	Object of class <code>Speclib</code> .
<code>percentage</code>	Flag if spectra in <code>x</code> are in range [0, 100]. If <code>FALSE</code> , the spectra are scaled to [0,100].
<code>gridsize</code>	Size of the grid used to perform least squares approximation.

**Details**

The algorithm fits a Gaussian function to the continuum points of the spectra in the spectral region between approx. 1500 to 2500 nm. The continuum points are derived constructing the convex hull of the spectra (see [transform\\_speclib](#)). The Gaussian function requires three parameter: (1) the mean values which is set to the water fundamental of 2800 nm, (2) the absorption depth at 2800 nm, and (3) the distance to the inflection point of the function. The latter two parameters are iteratively chosen using a grid search. The mesh size of the grid can be adjusted with the `gridsize` parameter. Note that the function requires the spectral reflectance values to be in interval [0, 100].

**Value**

Object of class `Speclib` containing the fitted Gaussian spectra and the parameters derived from the Gaussian curve. The three parameters (absorption depth, `R0`; distance to the inflection point, `sigma`; area between the curve and 100 % reflectance, `area`) are stored in the attributes of the new `Speclib`. Additionally, the function returns the final root mean square error of the Gaussian fit.

**Note**

The code is based on the IDL functions written by Michael L. Whiting.

**Author(s)**

Lukas Lehnert

**References**

Whiting, M. L., Li, L. and Ustin, S. L. (2004): Predicting water content using Gaussian model on soil spectra. *Remote Sensing of Environment*, 89, 535-552.

**See Also**

[soilindex](#), [Speclib](#)

**Examples**

```
## Use PROSAIL to simulate spectra with different soil moisture content
Spektr.lib <- smoothSpeclib(PROSAIL(parameterList = data.frame(psoil = seq(0,1,0.1), LAI = 0)))

smgm_val <- smgm(Spektr.lib)

for (i in 1:nspectra(smgm_val))
  plot(smgm_val, FUN = i, new = i==1, col = i)

attribute(smgm_val)
```

---

smoothSpeclib

*Smooth spectra*

---

**Description**

Smooth spectra using Savitzky-Golay filtering, lowess-, spline-functions or mean filter.

**Usage**

```
smoothSpeclib(x, method = "mean", ...)
```

**Arguments**

x	Object of class <a href="#">Speclib</a> .
method	Character string giving the method to be used. Predefined valid options are "sgolay", "lowess", "spline" and "mean". However, method can also be the (character) name of any other filter function (see examples).
...	Further arguments passed to filter functions. See examples.

**Details**

This function allows filtering using four different methods:

- Savitzky-Golay: Smoothing applying Savitzky-Golay-Filter. See [sgolayfilt](#) for details.
- Lowess: Smoothing applying lowess-Filter. See [lowess](#) for details.
- Spline: Smoothing applying spline-Filter. See [spline](#) for details.
- Mean: Smoothing applying mean-Filter. See [meanfilter](#) for details.

**Value**

Object of class [Speclib](#).

**Author(s)**

Lukas Lehnert

**References**

Tsai, F. & Philpot, W. (1998): Derivative analysis of hyperspectral data. Remote Sensing of Environment 66/1. 41-51.

**See Also**

[sgolayfilt](#), [lowess](#), [spline](#), [meanfilter](#)

**Examples**

```
data(spectral_data)

## Example of predefined filter functions
## Savitzky-Golay
sgolay <- smoothSpeclib(spectral_data, method="sgolay", n=25)

## Spline
spline <- smoothSpeclib(spectral_data, method="spline",
                       n=round(nbands(spectral_data)/10,0))

## Lowess
lowess <- smoothSpeclib(spectral_data, method="lowess", f=.01)

## Mean
meanflt <- smoothSpeclib(spectral_data, method="mean", p=5)

par(mfrow=c(2,2))
plot(spectral_data, FUN=1, main="Savitzky-Golay")
plot(sgolay, FUN=1, new=FALSE, col="red", lty="dotted")
plot(spectral_data, FUN=1, main="Spline")
plot(spline, FUN=1, new=FALSE, col="red", lty="dotted")
plot(spectral_data, FUN=1, main="Lowess")
plot(lowess, FUN=1, new=FALSE, col="red", lty="dotted")
plot(spectral_data, FUN=1, main="Mean")
plot(meanflt, FUN=1, new=FALSE, col="red", lty="dotted")

## Example of a not predefined filter function (Butterworth filter)
bf <- butter(3, 0.1)
bf_spec <- smoothSpeclib(spectral_data, method="filter", filt=bf)
plot(spectral_data, FUN=1, main="Butterworth filter")
plot(bf_spec, FUN=1, new=FALSE, col="red", lty="dotted")
```

soilindex

*soilindex***Description**

Function calculates a variety of hyperspectral soil indices

**Usage**

```
soilindex(x, index, returnHCR = "auto", weighted = TRUE, ...)
```

**Arguments**

x	Object of class Speclib
index	Character string. Name or definition of index or vector with names/definitions of indices to calculate. See Details section for further information.
returnHCR	If TRUE, the result will be of class HyperSpecRaster, otherwise it is a data frame. If "auto", the class is automatically determined by passed Speclib.
weighted	Logical indicating if reflectance values should be interpolated to fit wavelength position. If FALSE the reflectance values of nearest neighbour to passed position are returned. See <a href="#">get_reflectance</a> for further explanation.
...	Further arguments passed to derivative functions. Only used for indices requiring derivations.

**Details**

Index must be a character vector containing pre-defined indices (selected by their name) or self defined indices or any combination of pre- and self-defined indices.

**Pre-defined indices:** The following indices are available:

Name	Formula	Reference*
BI_TM	$((TM_1^2 + TM_2^2 + TM_3^2)/3)^{0.5}$ **	Mathieu et al. (1998)
CI_TM	$(TM_3 - TM_2)/(TM_3 + TM_2)$ **	Escadafal and Huete (1991)
HI_TM	$(2 \cdot TM_3 - TM_2 - TM_1)/(TM_2 - TM_1)$ **	Escadafal et al. (1994)
NDI	$(R_{840} - R_{1650})/(R_{840} + R_{1650})$	McNairn, H. and Protz, R. (1993)
NSMI	$(R_{1800} - R_{2119})/(R_{1800} + R_{2119})$	Haubrock et al. (2008)
RI	$R_{693}^2/(R_{447} \cdot R_{556}^3)$	Ben-Dor et al. (2006)
RI_TM	$TM_3^2/(TM_1 \cdot TM_2^3)$ **	Madeira et al. (1997), Mathieu et al. (1998)
SI_TM	$(TM_3 - TM_1)/(TM_3 + TM_1)$ **	Escadafal et al. (1994)
SWIR SI	$-41.59 \cdot (R_{2210} - R_{2090}) + 1.24 \cdot (R_{2280} - R_{2090}) + 0.64$	Lobell et al. (2001)

\* For references please type: `hsdardocs("References.pdf")`.  
\*\* `TM_1` denotes the first band of Landsat Thematic Mapper. Consequently, the hyperspectral data is resampled to Landsat TM using [spectral.resampling](#) prior to the calculation of the index. For resampling, the spectral response function is used.

### Self-defining indices:

Self-defined indices may be passed using the following syntax:

- `Rxxx`: Reflectance at wavelength 'xxx'. Note that R must be upper case.
- `Dxxx`: First derivation of reflectance values at wavelength 'xxx'. Note that D must be upper case.

Using this syntax, complex indices can be easily defined. Note that the entire definition of the index must be passed as one character string. Consequently, the NSMI would be written as `"(R1800-R2119)/(R1800+R2119)"`.

### Value

A vector containing indices values. If `index` is a vector with `length > 1`, a data frame with `ncol = length(index)` and `nrow = number of spectra in x` is returned.

If function is called without any arguments, return value will be a vector containing all available indices in alphabetical order.

### Author(s)

Lukas Lehnert

### References

See `hsdardocs("References.pdf")`

### See Also

[vegindex](#), [get\\_reflectance](#)

### Examples

```
data(spectral_data)
## Example calculating all available indices
## Get available indices

avl <- soilindex()
vi <- soilindex(spectral_data, avl)
```

specfeat

*Function to isolate spectral features***Description**

Function isolates specified absorption features previously identified by [define.features](#).

**Usage**

```
specfeat(x, FWL)

## S4 method for signature 'Specfeat'
plot(x, fnumber = 1, stylebysubset = NULL, changecol = TRUE,
     changetype = FALSE, autolegend = TRUE, new = TRUE, ...)
```

**Arguments**

x	Object of class <code>SpecLib</code> containing the band depth or ratio transformed reflectance spectra with additional information on feature limits calculated by <a href="#">define.features</a> . For plot this must be object of class <code>specfeat</code> .
FWL	A vector containing one wavelength per feature to be isolated, e.g. the major absorption features. Features which include these specified wavelengths will be isolated.
fnumber	Index of feature(s) to be plotted.
stylebysubset	Name of variable to be used as grouping factor. May be selected from attributes table, groups or subgroups and must be convertible to factors.
changecol	Flag, if line colour should be varied among groups
changetype	Flag, if line styles should be varied among groups
autolegend	Flag if, legend is printed automatically.
new	Create new plot or add data to existing one.
...	Further arguments passed to plot function.

**Value**

An object of class `Specfeat` containing the isolated features.

**Author(s)**

Hanna Meyer and Lukas Lehnert

**See Also**

[define.features](#), [cut\\_specfeat](#), [Specfeat](#)

## Examples

```
data(spectral_data)

## Transform speclib
bd <- transform_speclib(spectral_data, method = "sh", out = "bd")

## Define features automatically
features <- define.features(bd)

##Example to isolate the features around 450nm, 700nm, 1200nm and 1500nm.
featureSelection <- specfeat(features, c(450,700,1200,1500))

## Plot features
plot(featureSelection, 1:4)

## Advanced plotting example
plot(featureSelection, 1:4, stylebysubset = "Site")

plot(featureSelection, 1:4, stylebysubset = "Site", changecol = FALSE,
      changetype = TRUE)
```

---

Specfeat-class      \* *Specfeat class*

---

## Description

Class to handle spectral feature data.

## Details

Class extends `Speclib`-class and adds two additional slots:

- `features`: List containing the spectra according to the features.
- `featureLimits`: List containing limits of features defined by `define.features`.

## Note

See figure in [hsdar-package](#) for an overview of classes in `hsdar`.

## Author(s)

Lukas Lehnert

## See Also

[Speclib](#)

---

 speclib

*Methods to create objects of class Speclib*


---

## Description

Methods to create objects of class Speclib from various data types

## Usage

```
## S4 method for signature 'matrix,numeric'
speclib(spectra, wavelength, ...)

## S4 method for signature 'SpatialGridDataFrame,numeric'
speclib(spectra, wavelength, ...)

## S4 method for signature 'numeric,numeric'
speclib(spectra, wavelength, ...)

## S4 method for signature 'matrix,data.frame'
speclib(spectra, wavelength, ...)

## S4 method for signature 'SpatialGridDataFrame,data.frame'
speclib(spectra, wavelength, ...)

## S4 method for signature 'numeric,data.frame'
speclib(spectra, wavelength, ...)

## S4 method for signature 'matrix,matrix'
speclib(spectra, wavelength, ...)

## S4 method for signature 'SpatialGridDataFrame,matrix'
speclib(spectra, wavelength, ...)

## S4 method for signature 'numeric,matrix'
speclib(spectra, wavelength, ...)

## S4 method for signature 'HyperSpecRaster,ANY'
speclib(spectra, wavelength, ...)

## S4 method for signature 'Speclib'
print(x)

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

createspeclib(spectra, wavelength, fwhm = NULL, attributes = NULL,
              usagelibrary = NULL, transformation = NULL,
```



```
continuousdata = "auto", wlunit = "nm",
xlabel = "Wavelength", ylabel = "Reflectance",
rastermeta = NULL)
```

```
is.speclib(x)
```

### Arguments

spectra	Data frame, matrix of raster object of class 'RasterBrick' or 'SpatialGridDataFrame' with spectral data
x,object	Object to be converted to or from Speclib. For conversion to Speclib it can be a of class 'data frame', 'matrix', 'list' or 'character string'. In the latter case x is interpreted as path to raster object and read by readGDAL. For conversion from Speclib the object must be of class Speclib.
wavelength	Vector with corresponding wavelength for each band. A matrix or data.frame may be passed giving the upper and lower limit of each band. In this case, the first column is used as lower band limit and the second as upper limit, respectively.
fwhm	Vector containing full-width-half-max values for each band
attributes	Data frame with additional attributes data.
transformation	Kind of transformation applied to spectral data (character)
usagehistory	Character string or vector used for history of usage
continuousdata	Flag indicating if spectra are quasi continuous or discrete sensor spectra
wlunit	Unit of wavelength in spectra
xlabel	Label of wavelength data to be used for plots etc.
ylabel	Label of spectral signal to be used for plots etc.
rastermeta	List of meta information for SpatialGridDataFrame or HyperSpecRaster. If missing, meta data in speclib is used. Use function <a href="#">rastermeta</a> to create valid objects.
...	Further arguments passed to specific (generic) functions or <code>createspeclib</code>

### Details

See details in [Speclib](#).

### Value

An object of class `Speclib` containing the following slots is returned:

- wavelength: Vector with wavelength information
- fwhm: Vector or single numerical value giving the full-width-half-max value(s) for each band.
- spectra: Object of class `'Spectra'` with three slots:
  - fromRaster: logical, indicating if spectral data is read from a RasterBrick-object.
  - spectra\_ma: Matrix with ncol = number of bands and nrow = number. Used if fromRaster == FALSE

– spectra\_ra: RasterBrick-object which is used if fromRaster == TRUE.

Contains reflectance, transmittance or absorbance values. Handle with function [spectra](#).

- attributes: Data frame containing additional data to each spectrum. May be used for linear regression etc. Handle with function [attribute](#).
- usagehistory: Vector giving information on history of usage of speclib. Handle with function [usagehistory](#).
- rastermeta: List containing meta information to create \*Raster objects from Speclib. Handle with function [rastermeta](#).

### Author(s)

Lukas Lehnert

### See Also

[Speclib](#), [plot](#), [readGDAL](#), [mask](#),  
[idSpeclib](#), [dim](#), [spectra](#),  
[attribute](#)

### Examples

```
data(spectral_data)
spectra <- spectra(spectral_data)
wavelength <- spectral_data$wavelength

spectra <- speclib(spectra,wavelength)
```

---

Speclib-class

\* *Speclib class*

---

### Description

Class to store and handle hyperspectral data in R

### Details

**Spectral data:** The spectral data (usually reflectance values) are stored in an object of class `'Speclib'`. This object may either contain the spectral data as a RasterBrick or as a matrix with columns indicating spectral bands and rows different samples, respectively. The Speclib-class provides converting routines to and from RasterBrick-class allowing to read and write geographic raster data via [brick](#) and its extension [HyperSpecRaster-class](#). Since R is in general not intended to be used for VERY large data sets, this functionality should be handled with care. If raster files are large, one should split them in multiple smaller ones and process each of the small files, separately. See the excellent tutorial 'Writing functions for large raster files' available on <https://cran.r-project.org/web/packages/raster/vignettes/functions.pdf> and section '2.2.2 Speclibs from raster files' in 'hsdar-intro.pdf'.

**Spectral information:** Speclib contains wavelength information for each band in spectral data. This information is used for spectral resampling, vegetation indices and plotting etc. Since spectra can be handled either as continuous lines or as discrete values like for satellite bands, spectral information is handled in two principle ways:

- Continuous spectra: Data of spectrometers or hyperspectral (satellite) sensors. This data is plotted as lines with dotted lines indicating standard deviations by default.
- Non-continuous spectra: Data of multispectral satellite sensors. Here, data is plotted as solid lines and error bars at the mean position of each waveband indicating standard deviations by default.

The kind of data may be chosen by the user by setting the attribute flag "continuousdata" (`attr(x, "continuousdata")`) or passing `continuousdata = TRUE/FALSE`, when initially converting data to `Speclib`-class. Take care of doing so, because some functions as `spectral.resampling` may only work correctly with continuous data!

The unit of spectral data must be set initially, when converting data to `speclib`. Note that the package currently supports only "nm" as unit. This is particularly important for function like `vegindex`, which need to get correct bands out of the spectral data.

**Technical description:** An object of class `Speclib` contains the following slots:

- wavelength: Vector with wavelength information.
- fwhm: Vector or single numerical value giving the full-width-half-max value(s) for each band.
- spectra: Object of class `'Spectra'` with three slots:
  - `fromRaster`: logical, indicating if spectral data is read from a `RasterBrick`-object.
  - `spectra_ma`: Matrix with `ncol = number of bands` and `nrow = number`. Used if `fromRaster == FALSE`
  - `spectra_ra`: `RasterBrick`-object which is used if `fromRaster == TRUE`.

Contains reflectance, transmittance or absorbance values. Handle with function `spectra`.

- attributes: Data frame containing additional data to each spectrum. May be used for linear regression etc. Handle with function `attribute`.
- usagehistory: Vector giving information on history of usage of `speclib`. Handle with function `usagehistory`.

#### Note

See figure in `hsdar-package` for an overview of classes in `hsdar`.

#### Author(s)

Lukas Lehnert

#### See Also

`plot`, `readGDAL`, `mask`, `idSpeclib`,  
`dim`, `spectra`, `attribute`

spectra

*Handling spectra***Description**

Returning and setting spectra in Speclib

**Usage**

```
## S4 method for signature 'Speclib'
spectra(object, ...)

## S4 replacement method for signature 'Speclib,data.frame'
spectra(object) <- value

## S4 replacement method for signature 'Speclib,matrix'
spectra(object) <- value

## S4 replacement method for signature 'Speclib,numeric'
spectra(object) <- value

## S4 replacement method for signature 'Speclib,RasterBrick'
spectra(object) <- value
```

**Arguments**

object	Object of class Speclib.
...	Passed to internal function. Currently only one parameter is accepted: return_names: Logical indicating, if names of columns and rows should be set to <a href="#">bandnames</a> and <a href="#">idSpeclib</a> .
value	Matrix or RasterBrick-object containing spectral values. If value is a matrix, columns are band values and rows are spectra.

**Details**

For `spectra<-`, the function does not check if dimensions of spectra match dimensions of Speclib. Additionally, no conversion into matrix is performed! If spectra are not correctly stored, errors in other functions may arise. Thus check always carefully, if spectra are modified by hand.

**Value**

For `spectra<-`, the updated object. Otherwise a matrix of the spectra in x is returned.

**Author(s)**

Lukas Lehnert

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

```
data(spectral_data)

## Manual plot of the first spectrum
plot(wavelength(spectral_data), spectra(spectral_data)[1,], type="l")
```

---

spectral.resampling    *Spectral resampling*

---

**Description**

Resample spectra to (satellite) sensors

**Usage**

```
spectral.resampling(x, sensor, rm.NA = TRUE, continuousdata = "auto",
                  response_function = TRUE)
```

**Arguments**

x	Object of class <code>Speclib</code> . Data to be spectrally resampled.
sensor	Character or <code>data.frame</code> containing definition of sensor characteristics. See details section for further information.
rm.NA	If TRUE, channels which are not covered by input data wavelength are removed
continuousdata	Definition if returned <code>Speclib</code> is containing continuous data or not.
response_function	If TRUE, the spectral response function of the sensor is used for integration, if FALSE a Gaussian distribution is assumed and if NA the mean value of <code>spectra[min(ch):max(ch)]</code> is calculated.

**Details**

The characteristics of (satellite) sensor to integrate spectra can be chosen from a list of already implemented sensors. See [get.sensor.characteristics](#) for available sensors.

Otherwise the characteristics can be passed as a `data.frame` with two columns: first column with lower bounds of channels and second column with upper bounds. Alternatively, the `data.frame` may encompass band centre wavelength and full-width-half-maximum values of the sensor. Function will check the kind of data passed by partially matching the names of the data frame: If any column is named "fwhm" or "center", it is assumed that data are band centre and full-width-half-maximum values.

If sensor characteristics are defined manually, a Gaussian response is always assumed.

**Value**

Object of class `Speclib`

**Author(s)**

Lukas Lehnert

**See Also**

[get.sensor.characteristics](#), [get.gaussian.response](#)

**Examples**

```
## Load example data
data(spectral_data)

## Resample to RapidEye
data_RE <- spectral.resampling(spectral_data, "RapidEye",
                              response_function = TRUE)

## Plot resampled spectra
plot(data_RE)

## Compare different methods of spectral resampling
par(mfrow=c(1,3))
ga <- spectral.resampling(spectral_data, "RapidEye",
                          response_function = FALSE)
plot(ga)
re <- spectral.resampling(spectral_data, "RapidEye",
                          response_function = TRUE)
plot(re)
no <- spectral.resampling(spectral_data, "RapidEye",
                          response_function = NA)
plot(no)
```

---

spectral\_data

*Hyperspectral samples*

---

**Description**

Hyperspectral samples from two ecosystems on the Tibetan Plateau

**Usage**

```
data(spectral_data)
```

**Format**

An object of class `Speclib`

**Details**

Data has been sampled during vegetation period 2012 from two of the most wide-spread vegetation types on the Tibetan Plateau: *Kobresia* meadows and dry alpine steppes. Measurements were taken with a HandySpec Field portable spectrometer (tec5 AG Oberursel, Germany). This device has two channels measuring incoming and reflected radiation simultaneously between 305 and 1705 nm in 1 nm steps.

**Author(s)**

Lukas Lehnert, Hanna Meyer

**References**

Lehnert, L. W.; Meyer, H.; Meyer, N.; Reudenbach, C. & Bendix, J. (2014): A hyperspectral indicator system for rangeland degradation on the Tibetan Plateau: A case study towards spaceborne monitoring. *Ecological Indicators*, 39, 54 - 64. DOI: 10.1016/j.ecolind.2013.12.005

---

subset.speclib	<i>Subsetting speclibs</i>
----------------	----------------------------

---

**Description**

Return subsets of `Speclibs` which meet conditions.

**Usage**

```
## S4 method for signature 'Speclib'
subset(x, subset, ...)
```

**Arguments**

<code>x</code>	Object of class <code>'Speclib'</code> .
<code>subset</code>	Logical expression indicating spectra to keep: missing values are taken as false. See details section.
<code>...</code>	Further arguments passed to <code>agrep</code> .

**Details**

Matchable objects are attributes data. Use column names to identify the respective attribute. See [attribute](#) to access attributes of a `Speclib`.

**Value**

Object of class `Speclib`.

**Author(s)**

Lukas Lehnert

**See Also**

[Speclib](#), [attribute](#)

**Examples**

```
data(spectral_data)

## Return names of attributes data
names(spectral_data$attributes)

## Devide into both locations according to groups
sp_namco <- subset(spectral_data, Site == "Namco")
sp_kailash <- subset(spectral_data, Site == "Kailash")

## Plot both speclibs
plot(sp_namco, col="darkgreen")
plot(sp_kailash, col="darkred", new=FALSE)
```

---

t.test

*t-test for nri values*

---

**Description**

Performs t-tests for nri values.

**Usage**

```
## S4 method for signature 'Nri'
t.test(x, ...)
```

**Arguments**

x                    Object of class 'nri'.  
...                   Arguments to be passed to [t.test](#).

**Value**

An object of class "data.frame"



**Author(s)**

Lukas Lehnert & Hanna Meyer

**See Also**

[t.test](#), [cor.test](#), [Nri-method](#)

**Examples**

```
data(spectral_data)

## Subset spectral_data to Namco and Kailash samples
spectral_data <- subset(spectral_data, Site != "")

## Calculate nri-values for WorldView-2-8
spec_WV <- spectral.resampling(spectral_data, "WorldView2-8",
                              response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Perform t.tests between nri-values of both sites
site <- spec_WV$attributes$Site
ttestres <- t.test(x = nri_WV, y = site, alternative = "two.sided")
ttestres

## Plot p.values of t.tests
plot(ttestres)
```

---

transform_speclib	<i>Transform spectra</i>
-------------------	--------------------------

---

**Description**

Transform spectra by using convex hull or segmented upper hull

**Usage**

```
transform_speclib(data, ..., method = "ch", out = "bd")
```

**Arguments**

data	Speclib to be transformed
method	Method to be used. See details section.
out	Kind of value to be returned. See details section.
...	Further arguments passed to generic functions. Currently ignored.

## Details

Function performs a continuum removal transformation by firstly establishing a continuum line/hull which connects the local maxima of the reflectance spectrum. Two kinds of this hull are well established in scientific community: the convex hull (e.g. Mutanga et al. 2004) and the segmented hull (e.g. Clark et al. 1987). Both hulls are established by connecting the local maxima, however, the precondition of the convex hull is that the resulting continuum line must be convex whereas considering the segmented hull it might be concave or convex but the algebraic sign of the slope is not allowed to change from the global maximum of the spectrum downwards to the sides. In contrast to a convex hull, the segmented hull is able to identify small absorption features.

Specify method = "ch" for the convex hull and method = "sh" for the segmented hull. The output might be "raw", "bd" or "ratio":

- "raw": the continuum line is returned
- "bd": the spectra are transformed to band depth by

$$BD_{\lambda} = 1 - \frac{R_{\lambda}}{CV_{\lambda}},$$

where  $BD$  is the band depth,  $R$  is the reflectance and  $CV$  is the continuum value at the wavelength  $\lambda$ .

- "ratio": the spectra are transformed by

$$BD_{\lambda} = \frac{R_{\lambda}}{CV_{\lambda}}.$$

In some cases it might be useful to apply [smoothSpecLib](#) before the transformation if too many small local maxima are present in the spectra. Anyway, a manual improvement of the continuum line is possible using [addcp](#) and [deletecp](#).

## Value

If method != "raw" an object of class [SpecLib](#) containing transformed spectra is returned. Otherwise the return object will be of class [Clman](#).

## Author(s)

Hanna Meyer and Lukas Lehnert

## References

Clark, R. N., King, T. V. V. and Gorelick, N. S. (1987): Automatic continuum analysis of reflectance spectra. Proceedings of the Third Airborne Imaging Spectrometer Data Analysis Workshop, 30. 138-142.

Mutanga, O. and Skidmore, A. K. (2004): Narrow band vegetation indices overcome the saturation problem in biomass estimation. International Journal of Remote Sensing, 25, 3999-4014.

## See Also

[Clman](#), [addcp](#), [deletecp](#), [checkhull](#)

**Examples**

```
data(spectral_data)

transformed_spectra <- transform_speclib(spectral_data)

par(mfrow=c(1,2))
plot(spectral_data)
plot(transformed_spectra)
```

unmix

*Unmix spectra***Description**

Unmix spectra or spectra resampled to satellite bands using endmember spectra

**Usage**

```
unmix(spectra, endmember, returnHCR = "auto", scale = FALSE, ...)
```

**Arguments**

spectra	Input spectra of class 'speclib'
endmember	Endmember spectra of class 'speclib'
returnHCR	Set class of value. If TRUE, value will be of class 'HyperSpecRaster', otherwise a list is returned. If auto, function will switch to mode depending on input data characteristics.
scale	Flag to scale spectra to [0,1] if necessary.
...	Further arguments passed to <a href="#">HyperSpecRaster</a> (ignored if returnHCR = FALSE).

**Details**

Linear spectral unmixing is a frequently used method to calculate fractions of land-cover classes (endmembers) within the footprint of pixels. This approach has originally been intended to be used for multispectral satellite images. The basic assumption is that the signal received at the sensor ( $\rho_{mix}$ ) is a linear combination of  $n$  pure endmember signals ( $\rho_i$ ) and their cover fractions ( $f_i$ ):

$$\rho_{mix} = \sum_{i=1}^n \rho_i f_i,$$

where  $f_1, f_2, \dots, f_n \geq 0$  and  $\sum_{i=1}^n f_i = 1$  to fulfill two constraints:

1. All fractions must be greater or equal 0
2. The sum of all fractions must be 1

Since this linear equation system is usually over-determined, a least square solution is performed. The error between the final approximation and the observed pixel vector is returned as vector (error) in list (returnSpatialGrid = FALSE) or as last band if returnSpatialGrid = TRUE.

**Value**

A list containing the fraction of each endmember in each spectrum and an error value giving the euclidean norm of the error vector after least square error minimisation.

**Note**

Unmixing code is based on "i.spec.unmix" for GRASS 5 written by Markus Neteler (1999).

**Author(s)**

Lukas Lehnert

**References**

Sohn, Y. S. & McCoy, R. M. (1997): Mapping desert shrub rangeland using spectral unmixing and modeling spectral mixtures with TM data. *Photogrammetric Engineering and Remote Sensing*, 63, 707-716

**Examples**

```
## Not run:
data(spectral_data)
data(endmember_spectra)

## Mask channel crossing part (arround 1050 nm) and strong
## water absorption part (above 1350 nm)
mask(spectral_data) <- c(1045, 1055, 1350, 1706)

## Convert range of reflectance values to [0, 1]
spectra(spectral_data) <- spectra(spectral_data)/100
spectra(endmember_spectra) <- spectra(endmember_spectra)/100

## Perform unmixing for spectra from Kailash-site
unmix_fractions <- unmix(subset(spectral_data, Site == "Kailash"),
                        subset(endmember_spectra, Site == "Kailash"))

## End(Not run)
```

---

updatecl

*Check continuum line*

---

**Description**

Check if continuum line is intersecting the reflectance curve.

**Usage**

```
updatecl(x, hull)
```

**Arguments**

x                    Object of class `Speclib` transformed by `transform_speclib`.  
hull                 Hull to be applied to x. Output of function `makehull`.

**Value**

Object of class `Speclib`.

**Author(s)**

Lukas Lehnert and Hanna Meyer

**See Also**

[transform\\_speclib](#), [makehull](#), [Speclib](#)

**Examples**

```
## Model spectra using PROSAIL
parameter <- data.frame(N = rep.int(c(1, 1.5),2), LAI = c(1,1,3,3))
spec <- PROSAIL(parameterList=parameter)

## Transform spectra
spec_clman <- transform_speclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Add fix point at 4595 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2495)

## Plot new line
plot(spec_clman, ispec = 1, subset = c(2480, 2500))

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error

## Add fix point at 4596 nm to continuum line of first spectrum
spec_clman <- addcp(spec_clman, 1, 2496)

## Check new hull
hull <- checkhull(spec_clman, 1)
hull$error

hull <- makehull(spec_clman, 1)

## Transform spectra using band depth
spec_bd <- transform_speclib(spec, method = "sh", out = "bd")
```

```
## Update continuum line of first spectrum
spec_bd <- updatecl(spec_bd, hull)

## Plot modified transformed spectrum
plot(spec_bd, FUN = 1)
```

---

usagehistory

*History of usage*

---

## Description

Handling history of usage of Speclibs

## Usage

```
usagehistory(x)
usagehistory(x) <- value
```

## Arguments

x	Object of class Speclib
value	Character string to be added to usagehistory or NULL, if usagehistory should be deleted.

## Value

For usagehistory<-, the updated object. Otherwise a vector giving the history of usage of Speclib is returned.

## Author(s)

Lukas Lehnert

## See Also

[Speclib](#)

## Examples

```
data(spectral_data)

## Return history of usage
usagehistory(spectral_data)

## Deleting history of usage
usagehistory(spectral_data) <- character()
spectral_data

## Adding entries
```

```
usagehistory(spectral_data) <- "New entry" ## Adding new entry
usagehistory(spectral_data) <- "New entry 2" ## Adding second entry
spectral_data
```

vegindex

*vegindex*

## Description

Function calculates a variety of hyperspectral vegetation indices

## Usage

```
vegindex(x, index, returnHCR = "auto", L = 0.5,
         weighted = TRUE, ...)
```

## Arguments

x	Object of class <code>Speclib</code>
index	Character string. Name or definition of index or vector with names/definitions of indices to calculate. See Details section for further information.
returnHCR	If TRUE, the result will be of class <code>HyperSpecRaster</code> , otherwise it is a data frame. If "auto", the class is automatically determined by passed <code>Speclib</code> .
L	Factor for SAVI index. Unused for other indices.
weighted	Logical indicating if reflectance values should be interpolated to fit wavelength position. If FALSE the reflectance values of nearest neighbour to passed position are returned. See <a href="#">get_reflectance</a> for further explanation.
...	Further arguments passed to derivative functions. Only used for indices requiring derivations.

## Details

Index must be a character vector containing pre-defined indices (selected by their name) or self defined indices or any combination of pre- and self-defined indices.

**Pre-defined indices:** The following indices are available:

Name	Formula	Reference*
Boochs	$D_{703}$	Boochs et al. (1990)
Boochs2	$D_{720}$	Boochs et al. (1990)
CAI	$0.5 \cdot (R_{2000} + R_{2200}) - R_{2100}$	Nagler et al. (2003)
CARI	$a = (R_{700} - R_{550})/150$ $b = R_{550} - (a \cdot 550)$ $R_{700} \cdot \text{abs}(a \cdot 670 + R_{670} + b)/R_{670}$	Kim et al. (1994)

	$(a^2 + 1)^{0.5}$	
Carter	$R_{695}/R_{420}$	Carter (1994)
Carter2	$R_{695}/R_{760}$	Carter (1994)
Carter3	$R_{605}/R_{760}$	Carter (1994)
Carter4	$R_{710}/R_{760}$	Carter (1994)
Carter5	$R_{695}/R_{670}$	Carter (1994)
Carter6	$R_{550}$	Carter (1994)
CI	$R_{675} \cdot R_{690}/R_{683}^2$	Zarco-Tejada et al. (2003)
CI2	$R_{760}/R_{700} - 1$	Gitelson et al. (2003)
CIAInt	$\int_{600nm}^{735nm} R$	Oppelt and Mauser (2004)
CRI1	$1/R_{515} - 1/R_{550}$	Gitelson et al. (2003)
CRI2	$1/R_{515} - 1/R_{770}$	Gitelson et al. (2003)
CRI3	$1/R_{515} - 1/R_{550} \cdot R_{770}$	Gitelson et al. (2003)
CRI4	$1/R_{515} - 1/R_{700} \cdot R_{770}$	Gitelson et al. (2003)
D1	$D_{730}/D_{706}$	Zarco-Tejada et al. (2003)
D2	$D_{705}/D_{722}$	Zarco-Tejada et al. (2003)
Datt	$(R_{850} - R_{710})/(R_{850} - R_{680})$	Datt (1999b)
Datt2	$R_{850}/R_{710}$	Datt (1999b)
Datt3	$D_{754}/D_{704}$	Datt (1999b)
Datt4	$R_{672}/(R_{550} \cdot R_{708})$	Datt (1998)
Datt5	$R_{672}/R_{550}$	Datt (1998)
Datt6	$(R_{860})/(R_{550} \cdot R_{708})$	Datt (1998)
Datt7	$(R_{860} - R_{2218})/(R_{860} - R_{1928})$	Datt (1999a)
Datt8	$(R_{860} - R_{1788})/(R_{860} - R_{1928})$	Datt (1999a)
DD	$(R_{749} - R_{720}) - (R_{701} - R_{672})$	le Maire et al. (2004)
DDn	$2 \cdot (R_{710} - R_{660} - R_{760})$	le Maire et al. (2008)
DPI	$(D_{688} \cdot D_{710})/D_{697}^2$	Zarco-Tejada et al. (2003)
DWSI1	$R_{800}/R_{1660}$	Apan et al. (2004)
DWSI2	$R_{1660}/R_{550}$	Apan et al. (2004)
DWSI3	$R_{1660}/R_{680}$	Apan et al. (2004)
DWSI4	$R_{550}/R_{680}$	Apan et al. (2004)
DWSI5	$(R_{800} + R_{550})/(R_{1660} + R_{680})$	Apan et al. (2004)
EGFN	$(\max(D_{650:750}) - \max(D_{500:550}))/$ $(\max(D_{650:750}) + \max(D_{500:550}))$	Penuelas et al. (1994)
EGFR	$\max(D_{650:750})/\max(D_{500:550})$	Penuelas et al. (1994)
EVI	$2.5 \cdot ((R_{800} - R_{670})/$ $(R_{800} - (6 \cdot R_{670}) - (7.5 \cdot R_{475}) + 1))$	Huete et al. (1997)
GDVI	$(R_{800}^n - R_{680}^n)/(R_{800}^n + R_{680}^n)**$	Wu (2014)
GI	$R_{554}/R_{677}$	Smith et al. (1995)
Gitelson	$1/R_{700}$	Gitelson et al. (1999)
Gitelson2	$(R_{750} - R_{800}/R_{695} - R_{740}) - 1$	Gitelson et al. (2003)
GMI1	$R_{750}/R_{550}$	Gitelson et al. (2003)
GMI2	$R_{750}/R_{700}$	Gitelson et al. (2003)



Green NDVI	$(R_{800} - R_{550}) / (R_{800} + R_{550})$	Gitelson et al. (1996)
LWVI_1	$(R_{1094} - R_{983}) / (R_{1094} + R_{983})$	Galvao et al. (2005)
LWVI_2	$(R_{1094} - R_{1205}) / (R_{1094} + R_{1205})$	Galvao et al. (2005)
Maccioni	$(R_{780} - R_{710}) / (R_{780} - R_{680})$	Maccioni et al. (2001)
MCARI	$((R_{700} - R_{670}) - 0.2 \cdot (R_{700} - R_{550})) \cdot (R_{700} / R_{670})$	Daughtry et al. (2000)
MCARI/OSAVI		Daughtry et al. (2000)
MCARI2	$((R_{750} - R_{705}) - 0.2 \cdot (R_{750} - R_{550})) \cdot (R_{750} / R_{705})$	Wu et al. (2008)
MCARI2/OSAVI2		Wu et al. (2008)
mND705	$(R_{750} - R_{705}) / (R_{750} + R_{705} - 2 \cdot R_{445})$	Sims and Gamon (2002)
mNDVI	$(R_{800} - R_{680}) / (R_{800} + R_{680} - 2 \cdot R_{445})$	Sims and Gamon (2002)
MPRI	$(R_{515} - R_{530}) / (R_{515} + R_{530})$	Hernandez-Clemente et al. (2011)
mREIP	Red-edge inflection point using Gaussain fit	Miller et al. (1990)
MSAVI	$0.5 \cdot (2 \cdot R_{800} + 1 - ((2 \cdot R_{800} + 1)^2 - 8 \cdot (R_{800} - R_{670}))^{0.5})$	Qi et al. (1994)
MSI	$R_{1600} / R_{817}$	Hunt and Rock (1989)
mSR	$(R_{800} - R_{445}) / (R_{680} - R_{445})$	Sims and Gamon (2002)
mSR2	$(R_{750} / R_{705}) - 1 / (R_{750} / R_{705} + 1)^{0.5}$	Chen (1996)
mSR705	$(R_{750} - R_{445}) / (R_{705} - R_{445})$	Sims and Gamon (2002)
MTCI	$(R_{754} - R_{709}) / (R_{709} - R_{681})$	Dash and Curran (2004)
MTVI	$1.2 \cdot (1.2 \cdot (R_{800} - R_{550}) - 2.5 \cdot (R_{670} - R_{550}))$	Haboudane et al. (2004)
NDLI	$(\log(1/R_{1754}) - \log(1/R_{1680})) / (\log(1/R_{1754}) + \log(1/R_{1680}))$	Serrano et al. (2002)
NDNI	$(\log(1/R_{1510}) - \log(1/R_{1680})) / (\log(1/R_{1510}) + \log(1/R_{1680}))$	Serrano et al. (2002)
NDVI	$(R_{800} - R_{680}) / (R_{800} + R_{680})$	Tucker (1979)
NDVI2	$(R_{750} - R_{705}) / (R_{750} + R_{705})$	Gitelson and Merzlyak (1994)
NDVI3	$(R_{682} - R_{553}) / (R_{682} + R_{553})$	Gandia et al. (2004)
NDWI	$(R_{860} - R_{1240}) / (R_{860} + R_{1240})$	Gao (1996)
NPCI	$(R_{680} - R_{430}) / (R_{680} + R_{430})$	Penuelas et al. (1994)
OSAVI	$(1 + 0.16) \cdot (R_{800} - R_{670}) / (R_{800} + R_{670} + 0.16)$	Rondeaux et al. (1996)
OSAVI2	$(1 + 0.16) \cdot (R_{750} - R_{705}) / (R_{750} + R_{705} + 0.16)$	Wu et al. (2008)
PARS	$R_{746} / R_{513}$	Chappelle et al. (1992)
PRI	$(R_{531} - R_{570}) / (R_{531} + R_{570})$	Gamon et al. (1992)
PRI_norm	$PRI \cdot (-1) / (RDVI \cdot R_{700} / R_{670})$	Zarco-Tejada et al. (2013)
PRI*CI2	$PRI \cdot CI2$	Garrity et al. (2011)
PSRI	$(R_{678} - R_{500}) / R_{750}$	Merzlyak et al. (1999)
PSSR	$R_{800} / R_{635}$	Blackburn (1998)
PSND	$(R_{800} - R_{470}) / (R_{800} - R_{470})$	Blackburn (1998)
PWI	$R_{970} / R_{900}$	Penuelas et al. (1997)
RDVI	$(R_{800} - R_{670}) / \sqrt{R_{800} + R_{670}}$	Roujean and Breon (1995)

REP_LE	Red-edge position through linear extrapolation.	Cho and Skidmore (2006)
REP_Li	$700 + 40 \cdot ((R_{670} + R_{780}/2)/(R_{740} - R_{700}))$	Guyot and Baret (1988)
SAVI	$(1 + L) \cdot (R_{800} - R_{670})/(R_{800} + R_{670} + L)$	Huete (1988)
SIPI	$(R_{800} - R_{445})/(R_{800} - R_{680})$	Penuelas et al. (1995), Penuelas et al. (1995a)
SPVI	$0.4 \cdot 3.7 \cdot (R_{800} - R_{670}) - 1.2 \cdot ((R_{530} - R_{670})^2)^{0.5}$	Vincini et al. (2006)
SR	$R_{800}/R_{680}$	Jordan (1969)
SR1	$R_{750}/R_{700}$	Gitelson and Merzlyak (1997)
SR2	$R_{752}/R_{690}$	Gitelson and Merzlyak (1997)
SR3	$R_{750}/R_{550}$	Gitelson and Merzlyak (1997)
SR4	$R_{700}/R_{670}$	McMurtey et al. (1994)
SR5	$R_{675}/R_{700}$	Chappelle et al. (1992)
SR6	$R_{750}/R_{710}$	Zarco-Tejada and Miller (1999)
SR7	$R_{440}/R_{690}$	Lichtenthaler et al. (1996)
SR8	$R_{515}/R_{550}$	Hernandez-Clemente et al. (2012)
SRPI	$R_{430}/R_{680}$	Penuelas et al. (1995)
SRWI	$R_{850}/R_{1240}$	Zarco-Tejada et al. (2003)
Sum_Dr1	$\sum_{i=626}^{795} D1_i$	Elvidge and Chen (1995)
Sum_Dr2	$\sum_{i=680}^{780} D1_i$	Filella and Penuelas (1994)
SWIR FI	$R_{2133}^2/(R_{2225} \cdot R_{2209}^3)$	Levin et al. (2007)
SWIR LI	$3.87 \cdot (R_{2210} - R_{2090}) - 27.51 \cdot (R_{2280} - R_{2090}) - 0.2$	Lobell et al. (2001)
SWIR SI	$-41.59 \cdot (R_{2210} - R_{2090}) + 1.24 \cdot (R_{2280} - R_{2090}) + 0.64$	Lobell et al. (2001)
SWIR VI	$37.72 \cdot (R_{2210} - R_{2090}) + 26.27 \cdot (R_{2280} - R_{2090}) + 0.57$	Lobell et al. (2001)
TCARI	$3 \cdot ((R_{700} - R_{670}) - 0.2 \cdot (R_{700} - R_{550}) \cdot (R_{700}/R_{670}))$	Haboudane et al. (2002)
TCARI/OSAVI	TCARI/OSAVI	Haboudane et al. (2002)
TCARI2	$3 \cdot ((R_{750} - R_{705}) - 0.2 \cdot (R_{750} - R_{550}) \cdot (R_{750}/R_{705}))$	Wu et al. (2008)
TCARI2/OSAVI2	TCARI2/OSAVI2	Wu et al. (2008)
TGI	$-0.5(190(R_{670} - R_{550}) - 120(R_{670} - R_{480}))$	Hunt et al. (2013)
TVI	$0.5 \cdot (120 \cdot (R_{750} - R_{550}) - 200 \cdot (R_{670} - R_{550}))$	Broge and Leblanc (2000)
Vogelmann	$R_{740}/R_{720}$	Vogelmann et al. (1993)
Vogelmann2	$(R_{734} - R_{747})/(R_{715} + R_{726})$	Vogelmann et al. (1993)
Vogelmann3	$D_{715}/D_{705}$	Vogelmann et al. (1993)
Vogelmann4	$(R_{734} - R_{747})/(R_{715} + R_{720})$	Vogelmann et al. (1993)

\* For references please type: `hsdardocs("References.pdf")`.  
 \*\* For GDVI n must be defined appending an underscore and the intended exponent to the index name. E.g., for  $n = 2$ , the correct index name would be "GDVI\_2". Note that GDVI-indices with  $n = 2, 3, 4$  will be derived if all available indices are calculated.

### Self-defining indices:

Self-defined indices may be passed using the following syntax:

- Rxxx: Reflectance at wavelength 'xxx'. Note that R must be upper case.
- Dxxx: First derivation of reflectance values at wavelength 'xxx'. Note that D must be upper case.

Using this syntax, complex indices can be easily defined. Note that the entire definition of the index must be passed as one character string. Consequently, the NDVI would be written as `"(R800-R680)/(R800+R680)"`.

### Value

A vector containing indices values. If index is a vector with length  $> 1$ , a data frame with `ncol = length(index)` and `nrow = number of spectra in x` is returned.

If function is called without any arguments, return value will be a vector containing all available indices in alphabetical order.

### Author(s)

Hanna Meyer and Lukas Lehnert

### References

See `hsdardocs("References.pdf")`

### See Also

[soilindex](#), [derivative.speclib](#), [rededge](#), [get\\_reflectance](#)

### Examples

```
## Example calculating NDVI
data(spectral_data)
ndvi <- vegindex(spectral_data, "NDVI")

## Example calculating all available indices
## Get available indices
avl <- vegindex()
vi <- vegindex(spectral_data, avl)

## Self-defined indices
## NDVI
vi <- vegindex(spectral_data, "(R800-R680)/(R800+R680)")
## NDNI
vi <- vegindex(spectral_data,
```

```

      "(log(1/R1510) - log(1/R1680))/(log(1/R1510) + log(1/R1680))")
## D1
vi <- vegindex(spectral_data, "D730/D706")

```

---

wavelength	<i>Handling wavelength</i>
------------	----------------------------

---

## Description

Returning and setting wavelength in Speclib and HyperSpecRaster

## Usage

```

## S4 method for signature 'Speclib'
wavelength(object)

## S4 replacement method for signature 'Speclib,data.frame'
wavelength(object) <- value

## S4 replacement method for signature 'Speclib,numeric'
wavelength(object) <- value

## S4 method for signature 'HyperSpecRaster'
wavelength(object)

## S4 replacement method for signature 'HyperSpecRaster,numeric'
wavelength(object) <- value

```

## Arguments

object	Object of class Speclib or HyperSpecRaster.
value	Numeric vector or data.frame containing wavelength values.

## Value

For wavelength<-, the updated object. Otherwise a numeric vector of the spectra in x is returned.

## Author(s)

Lukas Lehnert

## See Also

[Speclib](#), [HyperSpecRaster](#)

*wavelength*

93

**Examples**

```
data(spectral_data)
```

```
wavelength(spectral_data)
```

# Index

- \*Topic **aplot**
  - Clman, 12
  - plot.Nri, 52
  - plot.Specfeat, 54
  - plot.Speclib, 55
  - specfeat, 70
- \*Topic **classes**
  - Clman, 12
  - Clman-class, 15
  - distMat3D, 25
  - DistMat3D-class, 27
  - HyperSpecRaster-class, 41
  - Nri-class, 49
  - specfeat, 70
  - Specfeat-class, 71
  - speclib, 72
  - Speclib-class, 74
- \*Topic **datasets**
  - endmember\_spectra, 27
  - response\_functions, 64
  - spectral\_data, 78
- \*Topic **documentation**
  - hsdardocs, 38
- \*Topic **methods**
  - HyperSpecRaster, 39
  - Raster-methods, 61
- \*Topic **multivariate**
  - feature\_properties, 28
  - getNRI, 35
  - glm.nri, 37
  - nri, 48
  - nri\_best\_performance, 51
  - rededge, 63
  - soilindex, 68
  - spectral\_resampling, 77
  - transform\_speclib, 81
  - unmix, 83
  - vegindex, 87
- \*Topic **package**
  - hsdar-package, 3
- \*Topic **smooth**
  - meanfilter, 46
  - smoothSpeclib, 66
- \*Topic **spatial**
  - HyperSpecRaster, 39
  - HyperSpecRaster-class, 41
  - Raster-methods, 61
- \*Topic **utilities**
  - addcp, 4
  - apply.DistMat3D, 5
  - apply.Speclib, 7
  - attribute, 8
  - bandnames, 9
  - bdri, 10
  - checkhull, 11
  - cubePlot, 16
  - deletecp, 20
  - derivative.speclib, 21
  - dim.speclib, 22
  - get.gaussian.response, 30
  - get.response, 31
  - get.sensor.characteristics, 32
  - get.sensor.name, 33
  - getcp, 34
  - hsdardocs, 38
  - idSpeclib, 42
  - makehull, 43
  - mask, 44
  - merge, 47
  - spectra, 76
  - subset.speclib, 79
  - updatecl, 84
  - usagehistory, 86
  - wavelength, 92
  - (g)lm.nri, 49
  - <, ANY, DistMat3D-method (distMat3D), 25
  - <, DistMat3D, ANY-method (distMat3D), 25
  - <, DistMat3D, DistMat3D-method

- (distMat3D), 25
- <=, ANY, DistMat3D-method (distMat3D), 25
- <=, DistMat3D, ANY-method (distMat3D), 25
- <=, DistMat3D, DistMat3D-method (distMat3D), 25
- ==, ANY, DistMat3D-method (distMat3D), 25
- ==, DistMat3D, ANY-method (distMat3D), 25
- ==, DistMat3D, DistMat3D-method (distMat3D), 25
- >, ANY, DistMat3D-method (distMat3D), 25
- >, DistMat3D, ANY-method (distMat3D), 25
- >, DistMat3D, DistMat3D-method (distMat3D), 25
- >=, ANY, DistMat3D-method (distMat3D), 25
- >=, DistMat3D, ANY-method (distMat3D), 25
- >=, DistMat3D, DistMat3D-method (distMat3D), 25
- [, DistMat3D, ANY, ANY-method (distMat3D), 25
- [, Speclib, ANY, ANY-method (speclib), 72
- [<-, DistMat3D, ANY, ANY-method (distMat3D), 25
- \$, Nri-method (Nri-methods), 50
- \$, Speclib-method (speclib), 72
  
- addcp, 4, 12, 20, 34, 43, 82
- agrep, 79
- apply, 3, 6, 7, 26
- apply, DistMat3D-method (apply.DistMat3D), 5
- apply, Speclib-method (apply.Speclib), 7
- apply.DistMat3D, 5
- apply.Speclib, 7
- as.array, DistMat3D-method (distMat3D), 25
- as.data.frame, Nri-method (Nri-methods), 50
- as.matrix, DistMat3D-method (distMat3D), 25
- as.matrix, Nri-method (Nri-methods), 50
- attribute, 8, 74, 75, 79, 80
- attribute, Speclib-method (attribute), 8
- attribute.speclib (attribute), 8
- attribute<- (attribute), 8
- attribute<-, Speclib, data.frame-method (attribute), 8
- attribute<-, Speclib, matrix-method (attribute), 8
  
- bandnames, 9, 76
- bandnames<- (bandnames), 9
- bdri, 10
- brick, 40, 42, 74
- brick, Speclib, ANY-method (HyperSpecRaster), 39
  
- checkhull, 5, 11, 20, 82
- Clman, 12, 13, 14, 20, 34, 43, 82
- clman, 14
- Clman-class, 15
- colorRamp, 53
- cor.test, 15, 16, 53
- cor.test, Nri-method (cor.test), 15
- cor.test.nri (cor.test), 15
- createspeclib, 14
- createspeclib (speclib), 72
- cubePlot, 16
- cut\_specfeat, 3, 17, 70
  
- define.features, 11, 18, 18, 29, 70, 71
- deletecp, 5, 12, 20, 34, 43, 82
- derivative.speclib, 3, 21, 63, 64, 91
- dim, 74, 75
- dim, DistMat3D-method (distMat3D), 25
- dim, Speclib-method (dim.speclib), 22
- dim.speclib, 22
- dist, 23, 24
- dist.speclib, 14, 23
- DistMat3D, 6, 26, 49
- distMat3D, 25, 27
- distMat3D, array-method (distMat3D), 25
- distMat3D, matrix-method (distMat3D), 25
- distMat3D, numeric-method (distMat3D), 25
- DistMat3D-class, 27
  
- endmember\_spectra, 27
- extract, Speclib, ANY-method (Raster-methods), 61
  
- feature\_properties, 10, 28
  
- get.gaussian.response, 30, 30, 32, 78
- get.response, 31, 64
- get.sensor.characteristics, 30, 32, 32, 33, 77, 78
- get.sensor.name, 31, 32, 33
- get\_landsat4\_response (response\_functions), 64

- get\_landsat5\_response  
(response\_functions), 64
- get\_landsat7\_response  
(response\_functions), 64
- get\_landsat8\_response  
(response\_functions), 64
- get\_quickbird\_response  
(response\_functions), 64
- get\_RapidEye\_response (get\_response), 31
- get\_reflectance, 36, 68, 69, 87, 91
- get\_reflectance (response\_functions), 64
- get\_reflectance, Speclib-method  
(get\_reflectance), 36
- get\_reflectance.speclib  
(get\_reflectance), 36
- get\_response\_function (get\_response), 31
- get\_wv2\_4\_response  
(response\_functions), 64
- get\_wv2\_8\_response  
(response\_functions), 64
- getcp, 5, 20, 34
- getNRI, 16, 35, 37, 38
- getValuesBlock, HyperSpecRaster  
(HyperSpecRaster), 39
- getValuesBlock, HyperSpecRaster-method  
(HyperSpecRaster), 39
- glm, 37, 38, 49–51, 53, 55
- glm.nri, 16, 37, 48–51, 53, 55
- hcl, 53
- hsdar (hsdar-package), 3
- hsdar-package, 3
- hsdardocs, 38, 64
- HyperSpecRaster, 17, 39, 62, 83, 92
- HyperSpecRaster, character, numeric-method  
(HyperSpecRaster), 39
- HyperSpecRaster, HyperSpecRaster-method  
(HyperSpecRaster), 39
- HyperSpecRaster, RasterBrick, numeric-method  
(HyperSpecRaster), 39
- HyperSpecRaster, RasterLayer, numeric-method  
(HyperSpecRaster), 39
- HyperSpecRaster, Speclib, ANY-method  
(HyperSpecRaster), 39
- HyperSpecRaster-class, 41
- idSpeclib, 5, 42, 74–76
- idSpeclib<- (idSpeclib), 42
- initialize, C1man-method (c1man), 14
- initialize, Speclib-method (speclib), 72
- interpolate.mask (mask), 44
- is.speclib (speclib), 72
- Landsat\_4\_response  
(response\_functions), 64
- Landsat\_5\_response  
(response\_functions), 64
- Landsat\_7\_response  
(response\_functions), 64
- Landsat\_8\_response  
(response\_functions), 64
- legendSpeclib (plot.Speclib), 55
- list.available.sensors  
(get\_sensor.characteristics),  
32
- lm, 37, 38
- lm.nri, 16
- lm.nri (glm.nri), 37
- lowess, 66, 67
- makehull, 5, 12, 20, 43, 43, 85
- mark\_nri\_best\_performance  
(nri\_best\_performance), 51
- mask, 13, 44, 74, 75
- mask, Speclib-method (mask), 44
- mask<- (mask), 44
- mask<-, Speclib, data.frame-method  
(mask), 44
- mask<-, Speclib, list-method (mask), 44
- mask<-, Speclib, matrix-method (mask), 44
- mask<-, Speclib, numeric-method (mask), 44
- maskSpeclib (mask), 44
- match.fun, 6, 7
- meanfilter, 46, 66, 67
- merge, 47
- merge, Speclib, Speclib-method (merge), 47
- nbands (dim.speclib), 22
- ncol, .Spectra-method (speclib), 72
- ncol, DistMat3D-method (distMat3D), 25
- Nri, 16, 26, 35, 49
- nri, 3, 35, 48, 53, 55
- Nri-class, 49
- Nri-methods, 50
- nri\_best\_performance, 35, 37, 51
- nrow, .Spectra-method (speclib), 72
- nrow, DistMat3D-method (distMat3D), 25
- nspectra (dim.speclib), 22



- plot, [3](#), [16](#), [37](#), [38](#), [55](#), [74](#), [75](#)
- plot, Clman, ANY-method (plot.Speclib), [55](#)
- plot, Clman-method (Clman), [12](#)
- plot, Nri, ANY-method (plot.Nri), [52](#)
- plot, Nri-method (plot.Nri), [52](#)
- plot, Specfeat, ANY-method (plot.Specfeat), [54](#)
- plot, Specfeat-method (specfeat), [70](#)
- plot, Speclib, ANY-method (plot.Speclib), [55](#)
- plot.Nri, [52](#)
- plot.Specfeat, [54](#)
- plot.Speclib, [55](#)
- polygon, [51](#)
- print, Speclib-method (speclib), [72](#)
- print.default, [37](#)
- print.getNRI (getNRI), [35](#)
- PROSAIL, [3](#), [56](#), [60](#)
- PROSPECT, [3](#), [59](#), [59](#)
- Quickbird\_response (response\_functions), [64](#)
- RapidEye\_response (response\_functions), [64](#)
- Raster-methods, [61](#)
- rastermeta, [62](#), [73](#), [74](#)
- readGDAL, [74](#), [75](#)
- rededge, [3](#), [63](#), [91](#)
- response\_functions, [64](#)
- sam (dist.speclib), [23](#)
- sam\_distance (dist.speclib), [23](#)
- sgolayfilt, [21](#), [22](#), [66](#), [67](#)
- show, DistMat3D-method (distMat3D), [25](#)
- show, HyperSpecRaster-method (HyperSpecRaster-class), [41](#)
- show, Nri-method (Nri-methods), [50](#)
- show, Speclib-method (speclib), [72](#)
- simgm, [65](#)
- smoothSpeclib, [3](#), [46](#), [63](#), [64](#), [66](#), [82](#)
- soilindex, [3](#), [66](#), [68](#), [91](#)
- Specfeat, [18](#), [19](#), [70](#)
- specfeat, [11](#), [18](#), [19](#), [29](#), [70](#)
- Specfeat-class, [71](#)
- Speclib, [3](#), [7-9](#), [15](#), [19](#), [21](#), [23](#), [24](#), [28](#), [31](#), [32](#), [42](#), [45](#), [47](#), [49](#), [50](#), [56](#), [59](#), [60](#), [62](#), [66](#), [71](#), [73-75](#), [77](#), [80](#), [82](#), [85](#), [86](#), [92](#)
- speclib, [72](#)
- speclib, HyperSpecRaster, ANY-method (speclib), [72](#)
- speclib, matrix, data.frame-method (speclib), [72](#)
- speclib, matrix, matrix-method (speclib), [72](#)
- speclib, matrix, numeric-method (speclib), [72](#)
- speclib, numeric, data.frame-method (speclib), [72](#)
- speclib, numeric, matrix-method (speclib), [72](#)
- speclib, numeric, numeric-method (speclib), [72](#)
- speclib, RasterBrick, data.frame-method (speclib), [72](#)
- speclib, RasterBrick, matrix-method (speclib), [72](#)
- speclib, RasterBrick, numeric-method (speclib), [72](#)
- speclib, SpatialGridDataFrame, data.frame-method (speclib), [72](#)
- speclib, SpatialGridDataFrame, matrix-method (speclib), [72](#)
- speclib, SpatialGridDataFrame, numeric-method (speclib), [72](#)
- Speclib-class, [74](#)
- spectra, [32](#), [36](#), [74](#), [75](#), [76](#)
- spectra, Clman-method (clman), [14](#)
- spectra, Speclib-method (spectra), [76](#)
- spectra.Speclib (spectra), [76](#)
- spectra<- (spectra), [76](#)
- spectra<-, Clman, data.frame-method (clman), [14](#)
- spectra<-, Clman, matrix-method (clman), [14](#)
- spectra<-, Clman, numeric-method (clman), [14](#)
- spectra<-, Speclib, data.frame-method (spectra), [76](#)
- spectra<-, Speclib, matrix-method (spectra), [76](#)
- spectra<-, Speclib, numeric-method (spectra), [76](#)
- spectra<-, Speclib, RasterBrick-method (spectra), [76](#)
- spectral.resampling, [3](#), [32](#), [69](#), [75](#), [77](#)
- spectral\_data, [78](#)

spline, [66](#), [67](#)  
subset, [56](#)  
subset, Speclib-method (subset.speclib),  
[79](#)  
subset.speclib, [79](#)

t.test, [53](#), [80](#), [80](#), [81](#)  
t.test, Nri-method (t.test), [80](#)  
t.test.nri (t.test), [80](#)  
transform\_speclib, [3](#), [5](#), [10–14](#), [19](#), [20](#), [34](#),  
[43](#), [65](#), [81](#), [85](#)

unmix, [3](#), [83](#)  
updatecl, [5](#), [12](#), [20](#), [43](#), [84](#)  
usagehistory, [74](#), [75](#), [86](#)  
usagehistory<- (usagehistory), [86](#)

vegindex, [3](#), [22](#), [64](#), [69](#), [75](#), [87](#)

wavelength, [92](#)  
wavelength, HyperSpecRaster-method  
(wavelength), [92](#)  
wavelength, Nri-method (Nri-methods), [50](#)  
wavelength, Speclib-method (wavelength),  
[92](#)  
wavelength<- (wavelength), [92](#)  
wavelength<- , HyperSpecRaster, numeric-method  
(wavelength), [92](#)  
wavelength<- , Speclib, data.frame-method  
(wavelength), [92](#)  
wavelength<- , Speclib, numeric-method  
(wavelength), [92](#)  
writeRaster, Speclib, character-method  
(Raster-methods), [61](#)  
writeStart, HyperSpecRaster, character-method  
(HyperSpecRaster), [39](#)  
writeStart, HyperSpecRaster, Speclib-method  
(HyperSpecRaster), [39](#)  
writeValues, HyperSpecRaster, Speclib-method  
(HyperSpecRaster), [39](#)  
writeValues, RasterBrick, Speclib-method  
(HyperSpecRaster), [39](#)  
writeValues, RasterLayer, Speclib-method  
(HyperSpecRaster), [39](#)  
WV\_2\_8\_response (response\_functions), [64](#)