Package ‘hsdar’

February 21, 2022

Type Package
Title Manage, Analyse and Simulate Hyperspectral Data
Version 1.0.4
Date 2022-02-20
Maintainer Lukas W. Lehnert <lukaslehnert@googlemail.com>
Depends R (>= 3.3.1), raster (>= 2.5-8), rgdal (>= 1.1-10), signal, methods, caret, Boruta
Suggests rgl (>= 0.98.1), RCurl, foreach, asdreader
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

R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsdar-package</td>
<td>3</td>
</tr>
<tr>
<td>addcp</td>
<td>5</td>
</tr>
<tr>
<td>apply.DistMat3D</td>
<td>6</td>
</tr>
<tr>
<td>apply.SpecLib</td>
<td>8</td>
</tr>
<tr>
<td>bandnames</td>
<td>9</td>
</tr>
<tr>
<td>bdri</td>
<td>10</td>
</tr>
<tr>
<td>Boruta::Boruta</td>
<td>11</td>
</tr>
<tr>
<td>cancer_spectra</td>
<td>13</td>
</tr>
<tr>
<td>caret::createDataPartition-methods</td>
<td>14</td>
</tr>
<tr>
<td>caret::createFolds-methods</td>
<td>14</td>
</tr>
<tr>
<td>caret::createResample-methods</td>
<td>14</td>
</tr>
<tr>
<td>caret::featurePlot-methods</td>
<td>15</td>
</tr>
<tr>
<td>caret::gafs</td>
<td>15</td>
</tr>
<tr>
<td>caret::preProcess-methods</td>
<td>16</td>
</tr>
</tbody>
</table>

1
R topics documented:

caret::rfe ................................................................. 17
caret::safs ............................................................. 18
caret::sbf ............................................................... 19
caret::setPredictor .................................................. 21
caret::setResponse ................................................... 22
caret::showCaretParameters ......................................... 23
caret::train-methods ................................................ 24
checkhull ................................................................. 24
clman ....................................................................... 25
Clman-class ............................................................ 27
cor.test ................................................................. 28
cubePlot ................................................................. 29
cut_specfeat ............................................................ 31
deletecp ................................................................. 32
derivative.speclib ..................................................... 33
dim.speclib .............................................................. 35
dist.speclib .............................................................. 36
distMat3D ................................................................. 37
DistMat3D-class ........................................................ 39
Extract Speclib by index ............................................. 40
feature_properties .................................................... 41
get.gaussian.response ................................................. 43
get.sensor.characteristics ........................................... 44
getcp ..................................................................... 46
getNRI ................................................................... 47
get_reflectance ......................................................... 48
glm.nri .................................................................. 49
hsdardocs ............................................................... 50
hsdar_parallel ........................................................ 51
HyperSpecRaster ....................................................... 52
HyperSpecRaster-class ................................................ 54
idSpeclib ................................................................. 54
import_USGS ........................................................... 55
makehull ................................................................. 56
mask ...................................................................... 58
meanfilter ............................................................... 59
merge ..................................................................... 60
noiseFiltering .......................................................... 61
nri ......................................................................... 63
Nri-class ................................................................. 64
Nri-methods ............................................................ 65
nri_best_performance ................................................ 66
plot.Nri ................................................................. 67
plot.Specfeat .......................................................... 69
plot.Speclib ........................................................... 71
postprocessASD ........................................................ 72
predictHyperspec ....................................................... 73
PROSAIL ................................................................. 75
**hsdar-package**

Manage, analyse and simulate hyperspectral data in R

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROSPECT</td>
<td>77</td>
</tr>
<tr>
<td>Raster-methods</td>
<td>80</td>
</tr>
<tr>
<td>rastermeta</td>
<td>82</td>
</tr>
<tr>
<td>read.ASD</td>
<td>83</td>
</tr>
<tr>
<td>read_header</td>
<td>83</td>
</tr>
<tr>
<td>rededge</td>
<td>84</td>
</tr>
<tr>
<td>SI</td>
<td>86</td>
</tr>
<tr>
<td>smgm</td>
<td>88</td>
</tr>
<tr>
<td>soilindex</td>
<td>90</td>
</tr>
<tr>
<td>specfeat</td>
<td>92</td>
</tr>
<tr>
<td>Specfeat-class</td>
<td>93</td>
</tr>
<tr>
<td>speclib</td>
<td>94</td>
</tr>
<tr>
<td>Speclib-class</td>
<td>97</td>
</tr>
<tr>
<td>speclib_raster-methods</td>
<td>99</td>
</tr>
<tr>
<td>spectra</td>
<td>101</td>
</tr>
<tr>
<td>spectralInterpolation</td>
<td>102</td>
</tr>
<tr>
<td>spectralResampling</td>
<td>103</td>
</tr>
<tr>
<td>spectral_data</td>
<td>106</td>
</tr>
<tr>
<td>sr</td>
<td>106</td>
</tr>
<tr>
<td>subset.nri</td>
<td>107</td>
</tr>
<tr>
<td>subset.speclib</td>
<td>109</td>
</tr>
<tr>
<td>t.test</td>
<td>110</td>
</tr>
<tr>
<td>transformSpeclib</td>
<td>111</td>
</tr>
<tr>
<td>unmix</td>
<td>113</td>
</tr>
<tr>
<td>updatecl</td>
<td>115</td>
</tr>
<tr>
<td>usagehistory</td>
<td>117</td>
</tr>
<tr>
<td>vegindex</td>
<td>118</td>
</tr>
<tr>
<td>wavelength</td>
<td>123</td>
</tr>
</tbody>
</table>

**Index**

125

**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 (noiseFiltering) for noise reduction, resampling of bands to various satellite sensors (spectralResampling), continuum removal (transformSpeclib), 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, smgm) indices as well as ndvi-like narrow band indices (nri). hsdar further enables to perform linear spectral unmixing of spectra (unmix) by use of endmember spectra. Note that some functions allow the parallel execution using the doMPI-, doMC- and foreach-packages. Execute 'hsdar_parallel()' to get supporting functions.

• Data simulation: hsdar has implemented the models PROSAIL 5B (PROSAIL, Jacquemoud et al. 2009) and PROSPECT 5 and D (PROSPECT, Jacquemoud and Baret 1990, Feret et al. 2017) 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.

![Diagram of classes used in hsdar](image)

Note that the asterisk marks all classes for which wrapper functions for the caret package exist.

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, Jörg Bendix

References


---

**addcp**  
*Manually add fix point to continuum line*

**Description**  
This function is used to add an additional fix point to a manually created hull of a single spectrum. This fix point is then used to re-construct a 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 point(s).

**Details**  
In some cases, it might be desirable to manually adapt automatically constructed segmented hulls (transformSpeclib). For example local maxima could be removed because they are very small and maybe afflicted with uncertainties which might legitimate it to manipulate the continuum line. Therefore, hsdar provides functions to remove and add “continuum points” from or to a continuum line. Manually adapted continuum lines can then be used to update band depth or ratio transformation. Handle these functions with care to avoid continuum lines too much build by subjective decisions. In the typical workflow, spectra are first transformed (transformSpeclib). Continuum points can then be retrieved (getcp) and manually adapted by adding addcp and deleting (deletecp) of points. Use checkhull to check for errors. If all uncertainties are removed, re-calculate the hull (makehull) and update the transformed spectrum (updatecl).

**Value**  
Object of class Clman containing the updated version of x.

**Author(s)**  
Lukas Lehnert and Hanna Meyer

**See Also**  
transformSpeclib, deletecp, getcp, checkhull, makehull, updatecl, idSpeclib
apply.DistMat3D

Apply function for class DistMat3D

Examples

```r
## 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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, xlim = c(2480, 2500), ylim=c(0.022,0.024))

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

Description

Apply function to values in a 3-D distance matrix. The 3-D distance matrix is an S4-class in hsdar to efficiently store distance values in hyperspectral datasets.

Usage

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

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 examples).
- **FUN**: Function to be applied. Matched with match.fun.
- **...**: Further arguments passed to FUN.
- **simplify**: Currently ignored.

Value

Depending on the length of the return value of the specified function, objects of classes numeric or matrix are returned.
apply.DistMat3D

Author(s)
Lukas Lehnert

See Also
apply, match.fun, DistMat3D

Examples

data(spectral_data)

## Part I: Create an object of class DistMat3D
## Calculate all possible NRI - combinations for WorldView-2-8
spec_WV <- spectralResampling(spectral_data, "WorldView2-8",
   response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Get all NRI-values as numeric vector
nri_values <- as.numeric(t(as.matrix(getNRI(nri_WV,
   getFiniteNri(nri_WV)))))

## Create object of class DistMat3D
dmat <- distMat3D(nri_values, 8, 45)

## Part II: Apply function mean to values in the new object
## Calculate mean value of all samples for all indices
meanIndexVals <- apply(dmat, MARGIN = 1, FUN = mean)
## Convert to DistMat3D
meanIndexVals <- distMat3D(meanIndexVals, 8, 1)

## Same but for array
nri_WV_dat <- as.array(dmat)
meanIndexVals_arr <- apply(nri_WV_dat, MARGIN = c(1, 2), FUN = mean)
## Convert to DistMat3D
meanIndexVals_arr <- distMat3D(meanIndexVals_arr)

## Test if equal
all(meanIndexVals_arr == meanIndexVals)

## Calculate mean value of all indices within each sample
meanSampleVals <- apply(dmat, MARGIN = 3, FUN = mean)
meanSampleVals_arr <- apply(nri_WV_dat, MARGIN = 3, FUN = mean,
   na.rm = TRUE)

## Test if equal
all(meanSampleVals == meanSampleVals_arr)

## User-defined function (in this case the median)
quant <- function(x)
   return(quantile(x, probs = 0.5))
## Apply user defined function to all samples for all indices
medianIndexVals <- apply(dmat, MARGIN = 1, FUN = quant)

---

**apply.Speclib**

**Apply function for class Speclib**

**Description**

Apply function over all spectra or a subset of spectra in a Speclib.

**Usage**

```r
## S4 method for signature 'Speclib'
apply(X, FUN, bySI = NULL, ..., simplify = TRUE)
```

**Arguments**

- **X**: Object of class Speclib
- **FUN**: Function to be applied. Matched with `match.fun`.
- **bySI**: Character string giving the name of the column in the SI to be used as subsets to apply function `FUN` on.
- **...**: Further arguments passed to `FUN`.
- **simplify**: Currently ignored.

**Value**

Object of class Speclib.

**Author(s)**

Lukas Lehnert

**See Also**

`apply`, `match.fun`, `Speclib`

**Examples**

```r
data(spectral_data)

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

## Same as above but separately for both seasons
mean_spectra <- apply(spectral_data, FUN = mean, bySI = "season")
plot(mean_spectra[1, ], ylim = c(0,50))
plot(mean_spectra[2, ], new = FALSE)
SI(mean_spectra)
```
Description

Returning and setting names of bands in Speclib

Usage

bandnames(x)
bandnames(x) <- value

Arguments

x
value

Object of class Speclib.
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.

Note

Bandnames are not mandatory in Speclibs. If not set, the default names are in the form V+index of bands.

Author(s)

Lukas Lehnert

See Also

Speclib

Examples

data(spectral_data)

## Return band names
bandnames(spectral_data)

## Change band names
bandnames(spectral_data) <- paste("Band", wavelength(spectral_data),
sep = ",")

## Return new band names
bandnames(spectral_data)
bdri

*Band depth ratio indices*

**Description**

Calculate band depth ratio indices for objects of class Specfeat.

**Usage**

```r
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

\[
bdri = \frac{BD}{Dc},
\]

with \( BD \) is the band depth calculated by `transformSpeclib` and \( Dc \) is the maximum band depth called band centre. Method "ndbi" calculates 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**

See Also
transformSpeclib, specfeat

Examples

data(spectral_data)

## Transform speclib
bd <- transformSpeclib(subset(spectral_data, season == "summer"),
method = "sh", out = "bd")

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

Boruta::Boruta

Methods for Function Boruta

Description

Methods for function Boruta in package Boruta. Please refer to help pages in the Boruta-package
for further information.

Usage

## S4 method for signature 'Speclib'
Boruta(x, y, ..., returnData = TRUE, includeTentative = FALSE, na.rm = FALSE)

## S4 method for signature 'Nri'
Boruta(x, y, ..., returnData = TRUE, includeTentative = FALSE, na.rm = FALSE)

## S4 method for signature 'Specfeat'
Boruta(x, y, ..., returnData = TRUE, includeTentative = FALSE, na.rm = FALSE)

get_Boruta(x)
Arguments

x  Object of class Speclib, Nri or Specfeat. For get_Boruta, x must be the output of Boruta as Speclib or Nri.
y  A numeric or factor vector containing the outcome for each sample. If missing, the response variable set by setResponse is used.
returnData  Logical. If TRUE, the updated object of x is returned, otherwise only the result of Boruta is returned.
includeTentative  Logical. If TRUE, the tentative variables are kept and returned in the Speclib-object.
na.rm  Logical. If TRUE, all variables are excluded which contain at least one non-finite value.
...  Further arguments passed to Boruta.

Value

If returnData == TRUE, an object of class Speclib or Nri, otherwise an object of class Boruta. Note that if x is an object of class Specfeat, the function returns an object of class Speclib containing the relevant transformed band values.

Author(s)

Lukas Lehnert

See Also

rfe, gafs

Examples

## Not run:
data(spectral_data)

## Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

## Set additional predictor variables from the SI
spectral_data <- setPredictor(spectral_data, "season")

## Run Boruta
## Note that this may take some time!
bor_res <- Boruta(spectral_data)

get_Boruta(bor_res)
plot(get_Boruta(bor_res))

## End(Not run)
Description

Hyperspectral samples from the human larynx

Usage

data(cancer_spectra)

Format

An object of class Spec11b

Details

This dataset contains hyperspectral data from the human larynx. The data were acquired in a project aiming to test the feasibility to use hyperspectral imaging for the non-invasive detection of cancer of the human larynx (head-and-neck squamous cell carcinoma). In hsdar, a subset of the total dataset is kindly provided by the project. This subset includes hyperspectral images from 25 patients including 10 cases with histopathological diagnosis of cancer. The images were acquired using an endoscope which was coupled with a monochromatic CCD camera. As light source, a special Polychrome V light machine was used. This allowed to change the wavelength of the impinging radiation so that hyperspectral cubes could be acquired by combining several images taken under different illuminations. The images were preprocessed using the methodology proposed by Regeling et al. (2015). The spectra were manually classified into cancerous and non-cancerous tissue by medical experts which is included in the SI of the data.

Author(s)

Bianca Regeling, Lukas Lehnert

References

Methods for Function `createDataPartition`

**Description**

Methods for function `createDataPartition` in package `caret`. Please refer to help pages in the `caret`-package for further information.

**Methods**

```
signature(y = ".CaretHyperspectral")  # Wrapper method for createDataPartition.
Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
```

Methods for Function `createFolds` and `createMultiFolds`

**Description**

Methods for functions `createFolds` and `createMultiFolds` in package `caret`

**Methods**

```
signature(y = ".CaretHyperspectral")  # Wrapper methods for createFolds and createMultiFolds.
Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
```

Methods for Function `createResample`

**Description**

Methods for function `createResample` in package `caret`

**Methods**

```
signature(y = ".CaretHyperspectral")  # Wrapper method for createResample.
Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
```
Methods for Function `featurePlot`

**Description**
Methods for function `featurePlot` in package `caret`

**Methods**

```r
signature(x = ".CaretHyperspectral")  # Wrapper method for `featurePlot`.
Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
```

Methods for Function `gafs`

**Description**
Methods for function `gafs` in package `caret`. Please refer to help pages in the `caret`-package for further information.

**Usage**

```r
## S4 method for signature 'Speclib'
gafs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

## S4 method for signature 'Nri'
gafs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

## S4 method for signature 'Specfeat'
gafs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

get_gafs(x)
```

**Arguments**

- **x**
  Object of class Speclib, Nri or Specfeat. For `get_gafs`, `x` must be the output of `gafs` as Speclib or Nri.

- **y**
  A numeric or factor vector containing the outcome for each sample. If missing, the response variable set by `setResponse` is used.

- **cutoff**
  The cutoff value of the correlation coefficients between response variables.

- **returnData**
  Logical. If TRUE, the updated object of `x` is returned, otherwise only the result of `gafs` is returned.

- **na.rm**
  Logical. If TRUE, all variables are excluded which contain at least one non-finite value.

- **...**
  Further arguments passed to `gafs`.
Value

If returnData == TRUE, an object of class Speclib or Nri, otherwise an object of class gafs. Note that if x is an object of class Specfeat, the function returns an object of class Speclib containing the relevant transformed band values.

Author(s)

Lukas Lehnert

See Also

gafs

Examples

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

## Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

## Set additional predictor variables from the SI
spectral_data <- setPredictor(spectral_data, "season")

## Feature selection using genetic algorithms
## Note that this may take some time!
gafs_res <- gafs(spectral_data)

get_gafs(gafs_res)

## End(Not run)
```

Description

Methods for function preProcess in package caret. The function is mainly internally required, but can be also used to transform the reflectance values and the SI e.g., by centering, scaling etc.

Methods

signature(x = ".CaretHyperspectral") Wrapper method for preProcess. Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
Methods for Function \texttt{rfe}

Description

Methods for function \texttt{rfe} in package \texttt{caret}. Please refer to help pages in the \texttt{caret}-package for further information.

Usage

\begin{verbatim}
## S4 method for signature 'Speclib'
rfe(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

## S4 method for signature 'Nri'
\textbf{rfe}(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

## S4 method for signature 'Specfeat'
rfe(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

get_rfe(x)
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} Object of class Speclib, Nri or Specfeat. For \texttt{get_rfe}, \texttt{x} must be the output of \texttt{rfe} as Speclib or Nri.
\item \texttt{y} A numeric or factor vector containing the outcome for each sample. If missing, the response variable set by \texttt{setResponse} is used.
\item \texttt{cutoff} The cutoff value of the correlation coefficients between response variables.
\item \texttt{returnData} Logical. If TRUE, the updated object of \texttt{x} is returned, otherwise only the result of \texttt{rfe} is returned.
\item \texttt{na.rm} Logical. If TRUE, all variables are excluded which contain at least one non-finite value.
\item \texttt{...} Further arguments passed to \texttt{rfe}.
\end{itemize}

Value

If \texttt{returnData == TRUE}, an object of class Speclib or Nri, otherwise an object of class \texttt{rfe}. Note that if \texttt{x} is an object of class Specfeat, the function returns an object of class Speclib containing the relevant transformed band values.

Author(s)

Lukas Lehnert

See Also

\texttt{rfe}
Examples

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

# Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

# Set additional predictor variables from the SI
spectral_data <- setPredictor(spectral_data, "season")

# Recursive feature selection
# Note that this may take some time!
rfe_res <- rfe(spectral_data)

generic_rfe(rfe_res)

plot(generic_rfe(rfe_res))

# End(Not run)
```

---

**Description**

Methods for function `safs` in package *caret*. Please refer to help pages in the *caret*-package for further information.

**Usage**

```R
# S4 method for signature 'Speclib'
safs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

# S4 method for signature 'Nri'
safs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

# S4 method for signature 'Specfeat'
safs(x, y, cutoff = 0.95, returnData = TRUE, na.rm = FALSE, ...)

generic_safs(x)
```

**Arguments**

- `x`  
  Object of class `Speclib`, `Nri` or `Specfeat`. For `generic_safs`, `x` must be the output of `safs` as `Speclib` or `Nri`.

- `y`  
  A numeric or factor vector containing the outcome for each sample. If missing, the response variable set by `setResponse` is used.

- `cutoff`  
  The cutoff value of the correlation coefficients between response variables.
returnData  Logical. If TRUE, the updated object of x is returned, otherwise only the result of safs is returned.

na.rm   Logical. If TRUE, all variables are excluded which contain at least one non-finite value.

...  Further arguments passed to safs.

Value

If returnData == TRUE, an object of class Speclib or Nri, otherwise an object of class safs. Note that if x is an object of class Specfeat, the function returns an object of class Speclib containing the relevant transformed band values.

Author(s)

Lukas Lehnert

See Also

safs

Examples

## Not run:
data(spectral_data)

## Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

## Set additional predictor variables from the SI
spectral_data <- setPredictor(spectral_data, "season")

## Supervised feature selection using simulated annealing
## Note that this may take some time!
safs_res <- safs(spectral_data)

gs_safs(safs_res)

plot(get_safs(safs_res))

## End(Not run)

caret::sbf  Methods for Function sbf

Description

Methods for function sbf in package caret. Please refer to help pages in the caret-package for further information.
Usage

## S4 method for signature 'Speclib'
sbf(x, y, cutoff = 0.95, returnData = TRUE, ...)

## S4 method for signature 'Nri'
sbf(x, y, cutoff = 0.95, returnData = TRUE, ...)

## S4 method for signature 'Specfeat'
sbf(x, y, cutoff = 0.95, returnData = TRUE, ...)

get_sbf(sbf_obj)

Arguments

x Object of class Speclib, Nri or Specfeat.
y A numeric or factor vector containing the outcome for each sample. If missing, the response variable set by setResponse is used.
cutoff The cutoff value of the correlation coefficients between response variables.
returnData Logical. If TRUE, the updated object of x is returned, otherwise only the result of sbf is returned.
... Further arguments passed to sbf.
sbf_obj Object of class Speclib, Nri or Specfeat as output of sbf-function.

Value

If returnData == TRUE, an object of class Speclib or Nri, otherwise an object of class sbf. Note that if x is an object of class Specfeat, the function returns an object of class Speclib containing the relevant transformed band values.

Author(s)

Lukas Lehnert

See Also

sbf

Examples

## Not run:
data(spectral_data)

## Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

## Set additional predictor variables from the SI
spectral_data <- setPredictor(spectral_data, "season")
## Selection by filtering
## Note that this may take some time!
sbf_res <- sbf(spectral_data)
get_sbf(sbf_res)
plot(get_sbf(sbf_res))
## End(Not run)

caret::setPredictor  Set predictor variable(s)

### Description
Set predictor variable(s) to be used in model-fitting functions of package caret. This function can be used to define additional predictor variables stored in the SI of a Speclib- or Nri-object. If the passed object is of class Nri, By default, all Nri-indices (if the passed object is of class Nri) or all bands (if the passed object is of class Speclib) are used as predictors.

### Usage
```
## S4 method for signature '.CaretHyperspectral,character'
setPredictor(x, predictor)
```

### Arguments
- `x` Object of one of the following classes: Speclib, Nri, Specfeat.
- `predictor` Character vector. Name of additional predictor variable(s) (from the SI).

### Value
The updated object.

### Author(s)
Lukas Lehnert

### See Also
- showCaretParameters, setResponse
caret::setResponse

**Examples**

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

## Set "season" as additional predictor variable from the SI
spectral_data <- setPredictor(spectral_data, "season")

## Show caret related parameters stored in the Speclib
showCaretParameters(spectral_data)

## End(Not run)
```

---

caret::setResponse  Set response variable

---

**Description**

Set response variable to be used in model-fitting functions of package caret. The response variable must be set upon any model training using a hsdar-object in caret.

**Usage**

```r
## S4 method for signature '.CaretHyperspectral,character'
setResponse(x, response)
```

**Arguments**

- `x` Object of one of the following classes: Speclib, Nri, Specfeat.
- `response` Character. Name of response variable (from the SI).

**Value**

The updated object.

**Author(s)**

Lukas Lehnert

**See Also**

showCaretParameters, setPredictor
Examples

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

## Set response variable (Chlorophyll content)
spectral_data <- setResponse(spectral_data, "chlorophyll")

## Show caret related parameters stored in the Speclib
showCaretParameters(spectral_data)

## End(Not run)
```

### Description

Show caret related parameters in objects of classes Speclib, Nri, Specfeat. Several parameters such as predictor and response variables are internally stored and used for model training and validation in the caret-package.

### Usage

`showCaretParameters(x)`

### Arguments

- `x`  
  Object of one of the following classes: Speclib, Nri, Specfeat.

### Author(s)

Lukas Lehnert

### See Also

`sbf`
Methods for Function `train`

Description

Methods for functions `train` and `train.formula` in package `caret`.

Methods

- `signature(x = ".CaretHyperspectral")`: Wrapper method for `train`. Note that ".CaretHyperspectral" is a class union containing classes Speclib, Nri, Specfeat.
- `signature(form = "formula", data = "Speclib")`: Wrapper method for `train.formula` to be used with objects of class Speclib.

`checkhull`

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.

Details

In some cases, it might be desirable to manually adapt automatically constructed segmented hulls (`transformSpeclib`). For example local maxima could be removed because they are very small and maybe afflicted with uncertainties which might legitimate it to manipulate the continuum line. Therefore, hsdar provides functions to remove and add "continuum points" from or to a continuum line. Manually adapted continuum lines can then be used to update band depth or ratio transformation. Handle these functions with care to avoid continuum lines too much build by subjective decisions. In the typical workflow, spectra are first transformed (`transformSpeclib`). Continuum points can then be retrieved (`getcp`) and manually adapted by adding `addcp` and deleting `deletecp` of points. Use `checkhull` to check for errors. If all uncertainties are removed, recalculate the hull (`makehull`) and update the transformed spectrum (`updatecl`).

Value

Object of class list.
Author(s)

Lukas Lehnert and Hanna Meyer

See Also

transformSpeclib, addcp, deletecp, makehull, updatecl

Examples

```r
## 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1, 2))
plot(spec_clman, ispec = 1, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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

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

Description

Methods to create, manipulate and query objects of class 'Clman'. The class 'Clman' is used to store manually defined continuum lines and the associated spectra.
Usage

```r
## 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

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

Arguments

- `.Object,object` Matrix, numeric or array in cases of creation of 'Clman' objects otherwise object of class 'Clman'.
- `value` Object of class numeric, matrix or array which is used for replacement of the values in `x`.
- `...` Arguments passed to `speclib` or `plot.default`.
- `x` Object of class `clman`.
- `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.
- `hull.style` List of arguments passed to `lines` to construct the continuum line.
- `points.style` List of arguments passed to `points` to construct the continuum points. May be `NULL` to suppress plotting of fix points.

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 `transformSpeclib`.
**Clman-class**

**Author(s)**
Lukas Lehnert

**See Also**
- `dist.speclib`
- `Clman`
- `transformSpeclib`
- `plot`

**Examples**

```r
## 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Return first spectrum
spectra(spec_clman)[1,]

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

---

**Clman-class**

*Clman class*

**Description**

Class to store and handle manual continuum lines.

**Details**

The class is only required if a continuum line is manually adopted or entirely manually created. This is useful if the automatic approaches are not able to identify absorption features because, for instance, the spectrum has two pronounced maxima within the absorption feature of interest.

Clman is defined as `Speclib` extended by the following two slots:

- `cp`: Matrix containing the fix points (continuum points) of each spectrum.
- `hull`: Matrix containing the hull of each spectrum.

Normally, it is not necessary to manually change the values in any of the slots above. Use the functions `adcp` and `deletecp` to change the hulls manually. Functionality for conversion back to a `Speclib` with the final hull and the transformed spectra provides function `updatecl`.

**Note**

See figure in `hsdar-package` for an overview of classes in hsdar.
Author(s)
Lukas Lehnert and Hanna Meyer

See Also
transformSpecLib, plot, SpecLib, addcp, deletecp, updatecl

cor.test

Test for association/correlation between nri values and vector of samples

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 Nri or numerical vector
y Object of class Nri or numerical vector
... Further arguments passed to cor.test

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

```r
data(spectral_data)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectralResampling(spectral_data, "WorldView2-8",
 response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)
cortestnri <- cor.test(nri_WV, SI(spec_WV)$chlorophyll)
cortestnri
```

Description

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

Usage

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

Arguments

- **x**
  - Object of class `SpecLib`.
- **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.
- **z_interpolate**
  - Interpolate spectral dimension. This is useful if a cube is plotted which has a much larger spatial compared to spectral dimension. If `TRUE` the spectral dimension will be interpolated to the minimum of the spatial dimension. Alternatively, an integer value may be passed.
- **...**
  - Further arguments passed to `plotRGB` for the top of the cube. Currently, the following two arguments are supported:
• scale: Maximum (possible) value in the three channels. Defaults to the maximum value in the red, green and blue band selected by arguments r, g and b.

• stretch: Option to stretch the values to increase the contrast of the image: "lin" (default) or "hist"

Note

The function may take a lot of time if the images are large. Consider plotting a subset of the entire image instead of plotting the entire image. Please note that the example below demonstrates the functionality with a very small image.

For unknown reasons, it may be necessary to execute the function twice in order to get the right colors at the walls of the cube.

Author(s)

Lukas Lehnert

See Also

Speclib

Examples

## Not run:
## Create raster file using PROSPECT D
## Run PROSPECT for 1600 random chlorophyll content values
parameter <- data.frame(Cab = round(runif(1600, min = 10, max = 40), 0))
spectra <- PROSPECT(parameterList = parameter)
## Create SpatialPixelsDataFrame and fill data with spectra from PROSPECT
rows <- round(nspectra(spectra)/40, 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", drivername = "GTiff")

## Read the raster and plot 3D cube
wavelength <- wavelength(spectra)
ras <- speclib("example_in.tif", wavelength)
cubePlot(ras)

## End(Not run)
**cut_specfeat**  
*Cut absorption features*

**Description**  
Function cuts absorption features to a user-specified range. Since features may differ among spectra, it might be important to cut the features to specific wavelengths ranges.

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

**Arguments**  
- **x**  
  An object of class `Specfeat` containing isolated features determined by `specfeat`.
- **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.

**Details**  
The typical workflow to obtain feature properties is to first calculate the band depth `transformSpeclib`, then isolate the absorption features `specfeat`. Optionally, `cut_specfeat` allows to cut the features at specified wavelengths. Finally use `feature_properties` to retrieve characteristics of the features.

**Value**  
An object of class `Specfeat` containing the cut features.

**Author(s)**  
Hanna Meyer and Lukas Lehnert

**See Also**

`specfeat, Specfeat`

**Examples**

```r
data(spectral_data)

## Example to cut the features around 450nm and 700nm to a specific range
## Transform speclib
bd <- transformSpeclib(subset(spectral_data, season == "summer"),
                         method = "sh", out = "bd")
```
## Convert speclib to specfeat giving center wavelength of features
featureSelection <- specfeat(bd, 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)

deletecp

### Description
Delete fix point from continuum line.

### Usage
deletecp(x, ispec, cpdelete)

### Arguments
- **x**: Object of class `Clman`.
- **ispec**: ID or index of spectrum to be modified.
- **cpdelete**: Single value or vector of wavelength containing fix point(s) to be deleted.

### Details
In some cases, it might be desirable to manually adapt automatically constructed segmented hulls (`transformSpeclib`). For example local maxima could be removed because they are very small and maybe afflicted with uncertainties which might legitimate it to manipulate the continuum line. Therefore, hsdar provides functions to remove and add "continuum points" from or to a continuum line. Manually adapted continuum lines can then be used to update band depth or ratio transformation. Handle these functions with care to avoid continuum lines too much build by subjective decisions. In the typical workflow, spectra are first transformed (`transformSpeclib`). Continuum points can then be retrieved (`getcp`) and manually adapted by adding `addcp` and deleting (`deletecp`) of points. Use `checkhull` to check for errors. If all uncertainties are removed, re-calculate the hull (`makehull`) and update the transformed spectrum (`updatecl`).

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

### Author(s)
Lukas Lehnert and Hanna Meyer
See Also

transformSpeclib, 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1, 2))
plot(spec_clman, ispec = 1, xlim = c(1100, 1300), ylim=c(0.17,0.21))

## 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, xlim = c(1100, 1300), ylim=c(0.17,0.21))

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

---

derivative.speclib  Derivation

Description

Calculate derivations of spectra in Speclib.

Usage

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

Arguments

x  Object of class Speclib.

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 sgolayfilt.
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, default method)

Value

Object of class Speclib.

Author(s)

Lukas Lehnert

References


See Also

sgolayfilt, vegindex, soilindex

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)
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 for dim or single integer value for nspectra and nbands.

Author(s)

Lukas Lehnert

See Also

Speclib

Examples

data(spectral_data)
dim(spectral_data)
dist.speclib  

**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**

- **x**: Object of class Speclib. Note that spectra in x must be in range [0,1].
- **method**: The distance measure to be used. This must be one of "sam", "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
- **ref**: Object of class Speclib containing reference spectra.
- **...**: Further arguments, passed to other methods.

**Details**

Available distance measures are "spectral angle mapper" (sam, Kruse et al. 1993) 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


See Also
dist, Speclib

Examples

data(spectral_data)

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

## Calculate distance between all spectra from spring
## using spectral angle mapper
dist.speclib(subset(spectral_data, season == "spring"))

## Calculate spectral angle mapper between reference spectrum
## and spectral_data
## Use first spectrum from summer as reference
distance <- sam(subset(spectral_data, season == "spring"),
                subset(spectral_data, season == "summer")[1,])

distMat3D

Methods to create, manipulate and query objects of class 'DistMat3D'.

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'
x[i, j, n]

## S4 replacement method for signature 'DistMat3D'
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
DistMat3D-class

*DistMat3D class*

**Description**

Class to store effectively (large) distance matrices (up to 3D), which can be interpreted as a stack of traditional 2-D distance matrices. Therefore, the first two dimensions are of equal length and usually describe the wavelength in `hsdar`. This third dimension is normally the number of samples or pixels. In `hsdar`, objects of class DistMat3D are used e.g., to store nri-values. In this case, the first and second dimensions store the information which band #1 is substraced by which band #2, respectively. The third dimension is the sample. Since it usually does not matter if band #1 is substraced from band #2 or vice versa, the nri-matrix would contain the same absolute values on both triangles (as 2-D distance matrices would do). Therefore, `hsdar` defines and uses the class DistMat3D in which only one triangle is stored and memory demand is considerably reduced.

**See Also**

DistMat3D, apply, Nri

**Examples**

data(spectral_data)

```r
## Mask channel crossing part (around 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, season == "summer"))

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

## Default print of DistMat3D-object
sam_dist

## Convert back to matrix
as.matrix(sam_dist)

## Get number of rows and samples
dim(sam_dist)

## Compare values in DistMat3D-object
small_dists <- sam_dist < 0.02

## Convert small_dists-object to DistMat3D
distMat3D(as.numeric(small_dists), 15, 1)
```
Details

S4-class 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

The data in the values slot is organized as follows: The first value is the distance at band #1 and band #2 for sample number #1, the second one is for band #1 and band #3 (sample #1) and so forth.

Methods to create objects of class DistMat3D for matrix and array objects exist. Additionally, methods to apply functions to the values exist.

Note

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

Author(s)

Lukas Lehnert

See Also

distMat3D, apply.DistMat3D

Extract Speclib by index

Indexing Speclib

Description

Access subsets of data in Speclibs both in spectrals and sample dimensions

Usage

## S4 method for signature 'Speclib'
x[i, j, ...]

Arguments

x Object of class Speclib to be indexed.
i Samples to be returned.
j Bands to be returned.
... Further arguments (currently ignored).
feature_properties

Details

The first index represents the sample dimension and the second one is the band dimension. If the sample dimension is indexed, care is taken that the SI and the id is indexed as well.

Value

Object of class Speclib containing the updated version of x.

Author(s)

Lukas Lehnert

See Also

Speclib, subset.speclib, SI, idSpeclib

Examples

data(spectral_data)

## Get the first five spectra
spec_1_5 <- spectral_data[1:5,]
spec_1_5

## Get the first ten bands
spec_1_10 <- spectral_data[,1:10]
spec_1_10

## Get the bands number 20 to 30 for the third and fifth spectra
spec_20_30 <- spectral_data[c(3,5),20:30]
spec_20_30
Details

The function calculates several parameters:

- **area**: The feature area is calculated by

  \[
  \text{area}_F = \sum_{k=\text{min}(\lambda)}^{\text{max}(\lambda)} BD\lambda,
  \]

  with \( \text{area}_F \) is the area of the feature \( i \), \( \text{min}(\lambda) \) is the minimum wavelength of the spectrum, \( \text{max}(\lambda) \) 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.

The typical workflow to obtain feature properties is to first calculate the band depth `transformSpeclib`, then isolate the absorption features `specfeat`. Optionally, `cut_specfeat` allows to cut the features at specified wavelengths. Finally use `feature_properties` to retrieve characteristics of the features.

Value

An object of class `Specfeat` containing the properties as (part of the) SI table.

Author(s)

Hanna Meyer & Lukas Lehnert

See Also

- `specfeat`

Examples

data(spectral_data)

### Example calculating the areas of the features around 450nm, 700nm, 1200nm and 1500nm.

bd <- transformSpeclib(subset(spectral_data, season == "summer"),
                        method = "sh", out = "bd")

### Convert speclib to specfeat giving center wavelength of features

featureSelection <- specfeat(bd, c(450,700,1200,1500))
## Calculate properties of features
```r
featureProp <- feature_properties(featureSelection)
```

## See resulting feature property variables
```r
head(SI(featureProp))
```

---

**get.gaussian.response**  
*Gaussian response function*

### Description
Simulate Gaussian response function for band(s) of a (satellite) sensor. Each band is either defined by center and full-width-half-maximum values or by passing its upper and lower border.

### Usage
```r
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 (5% sensitivity). 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 sensor passed to the function.

### Author(s)
Lukas Lehnert

### See Also
- `get.sensor.characteristics`
- `spectralResampling`
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$response, “stepsize”))
for (i in 1:nrow(data_RE$response$characteristics))
    lines(xwl_response, data_RE$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.sensor.characteristics

Sensor characteristics

Description

Get channel wavelength of implemented (multispectral) satellite sensors.

Usage

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

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.

Details

The following sensors are currently implemented: ALI, EnMAP, Hyperion, Landsat4, Landsat5, Landsat7, Landsat8, MODIS, Quickbird, RapidEye, Sentinel2a, Sentinel2b, WorldView2-4, WorldView2-8.

Spectral response functions are available for the following ones: Landsat4, Landsat5, Landsat7, Landsat8, Quickbird, RapidEye, Sentinel2a, Sentinel2b, WorldView2-4, WorldView2-8.

Author(s)

Lukas Lehnert

See Also

spectralResampling

Examples

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

## Sentinel 2A
data_s2a <- get.sensor.characteristics("Sentinel2a", TRUE)

## Plot response functions
plot(c(0,1)~c(attr(data_s2a$response, "minwl"),
    attr(data_s2a$response, "maxwl")),
type = "n", xlab = "Wavelength [nm]",
ylab = "Spectral response")
xwl_response <- seq.int(attr(data_s2a$response, "minwl"),
    attr(data_s2a$response, "maxwl"),
    attr(data_s2a$response, "stepsize"))
for (i in 1:nrow(data_s2a$characteristics))
    lines(xwl_response, data_s2a$response[,i], col = i)

## Sentinel 2B
data_s2b <- get.sensor.characteristics("Sentinel2b", TRUE)

## Add response functions
for (i in 1:nrow(data_s2b$characteristics))
    lines(xwl_response, data_s2b$response[,i], col = i, lty = "dashed")
legend("topright", legend = c("Sentinel2a", "Sentinel2b"),

...
getcp

Description
Get fix points of continuum line within spectral range.

Usage
getcp(x, ispec, subset = NULL)

Arguments
x
Object of class Clman.
ispec
ID or index of spectrum to be analysed.
subset
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
transformSpeclib, 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 <- transformSpeclib(spec, method = "sh", out = "raw")

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

Return normalized ratio index values at a given wavelength combination.

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 <- spectralResampling(spectral_data, "WorldView2-8",
    response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

## Build glm-models
glm_nri <- glm.nri(nri_WV ~ chlorophyll, preddata = spec_WV)

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

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

Get reflectance values

---

**Description**

Returns weighted or unweighted reflectance values at wavelength position.

**Usage**

```
get_reflectance(spectra, wavelength, position, weighted = FALSE, ...)
```

**Arguments**

- `spectra`: Object of class `Speclib` or `data.frame` with reflectance values.
- `wavelength`: Vector with wavelength values. May be missing if `spectra` is object of class `Speclib`.
- `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 `FALSE` the reflectance values of nearest neighbour to passed position are returned.
- `...`: Arguments to be passed to specific functions. Currently 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)

## Simulate multispectral sensor encompassing two bands
## to show effect of weighted and unweighted modes
spectral_data_res <- spectralResampling(spectral_data,
    sensor = data.frame(lb = c(400, 600), ub = c(500, 700)))

## Compare reflectance at 520 nm (in between both bands to
## show the difference between weighted and unweighted modes
weighted_reflectance <- get_reflectance(spectral_data_res, 520, weighted = TRUE)
unweighted_reflectance <- get_reflectance(spectral_data_res, 520, weighted = FALSE)

## Plot result
plot(weighted_reflectance, unweighted_reflectance,
     ylab = "Reflectance at 520 nm (unweighted)",
     xlab = "Reflectance at 520 nm (weighted)"
)

---

### glm.nri

(Generalised) Linear models from normalised ratio indices

#### Description

Build (generalised) linear models of normalised ratio indices as response and predictor variables usually stored in the SI.

#### Usage

```r
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 `lm`, `glm` and generic `print.default`

#### 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 <- spectralResampling(spectral_data, "WorldView2-8",
                            response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)

glmri <- glm.nri(nri_WV ~ chlorophyll, preddata = spec_WV)
glmri

plot(glmri)
Usage
hsdardocs(doc)

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

Author(s)
Lukas Lehnert

Examples
## Not run:
## Open introduction to hsdar (PDF-file)
hsdardocs("Hsdar-intro.pdf")

## 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)

Description
Get all functions which support parallel execution. Currently, the parallel backend functions in doMPI and doMC are supported.

Usage
hsdar_parallel()

Details
Parallel execution is performed via the foreach-package. Care is taken that a function will never run in parallel if the calling function is already using multicore processing.

Value
Vector containing supported function names
Author(s)
Lukas Lehnert

Examples

```r
## Not run:
supported_functions <- hsdar_parallel()
supported_functions
data(spectral_data)

## Example for Windows and other systems where doMPI is available
## Load library
library(doMPI)
## Register number of workers
c1 <- startMPIcluster(count = 3)
registerDoMPI(c1)

## Transform speclib using 3 cores
bd <- transformSpeclib(spectral_data)

## Close the cluster (important to get rid of processes)
closeCluster(c1)

## Example for Linux and other systems where doMC is available
## Load library
library(doMC)
## Register number of workers
registerDoMC(3)

## Transform speclib using 3 cores
bd <- transformSpeclib(spectral_data)

## End(Not run)
```

---

**HyperSpecRaster**

Handle hyperspectral cubes using raster package (deprecated)

Description

The HyperSpecRaster-Class is deprecated. Use Speclib instead.

Usage

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

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

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

## S4 method for signature 'Speclib'
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

- **SI**
  
  Optional data.frame containing SI data

- **nrow**
  
  Optional. Number of rows in HyperspecRaster. If omitted, function will try to get the information from the SI in Speclib (attr(x,"rastermeta"))

- **ncol**
  
  Optional. Number of columns in HyperspecRaster. See nrow above.

- **xmn**
  
  Optional. Minimum coordinate in x-dimension. See nrow above.

- **xmx**
  
  Optional. Maximum coordinate in x-dimension. See nrow above.

- **ymn**
  
  Optional. Minimum coordinate in y-dimension. See nrow above.

- **ymx**
  
  Optional. Maximum coordinate 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 brick

- **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

HyperSpecRaster-class  HyperSpecRaster* class (deprecated)

Description

This is a deprecated class. Use Speclib-class instead.

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.
- **SI (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
import_USGS

Arguments

- **x**: Object of class Speclib.
- **value**: Character vector of the same length as `nspectra(x)`, or NULL.

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)

---

import_USGS

**import USGS spectra**

Description

Import and download spectral data from USGS spectral library

Usage

```r
USGS_get_available_files(url = NULL)

USGS_retrieve_files(avl = USGS_get_available_files(),
                     pattern = NULL, retrieve = TRUE,
                     loadAsSpeclib = TRUE, tol = 0.1)
```

Arguments

- **url**: Character passing the url of the data. If NULL, the following URL is used:
- **avl**: List of available files. Typically the result of `USGS_get_available_files`.
- **pattern**: Search pattern to define a subset of all available spectra.
- **retrieve**: Logical. Should the data be downloaded?
- **loadAsSpeclib**: Logical. If TRUE, an object of class "Speclib" is returned
- **tol**: Discrepancy of the wavelength values between different spectra.
makehull

Re-calculate hull

Description

Re-calculates the hull after it was manually adapted

Usage

makehull(x, ispec)

Arguments

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

Details

In some cases, it might be desirable to manually adapt automatically constructed segmented hulls (transformSpeclib). For example local maxima could be removed because they are very small and maybe afflicted with uncertainties which might legitimate it to manipulate the continuum line. Therefore, hsdar provides functions to remove and add "continuum points" from or to a continuum line. Manually adapted continuum lines can then be used to update band depth or ratio transformation. Handle these functions with care to avoid continuum lines too much build by subjective decisions. In the typical workflow, spectra are first transformed (transformSpeclib). Continuum points can then be retrieved (getcp) and manually adapted by adding addcp and deleting (deletecp) of points. Use checkhull to check for errors. If all uncertainties are removed, re-calculate the hull (makehull) and update the transformed spectrum (updatecl).

Value

Object of class list.
Author(s)

Lukas Lehnert and Hanna Meyer

See Also

transformSpeclib, addcp, deletecp, makehull, updatecl

Clman

Examples

```r
## 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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

## Re-calculate hull
hull <- makehull(spec_clman, 1)

## Transform spectra using band depth
spec_bd <- transformSpeclib(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 spectra

### Description

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

### Usage

```r
## 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 `Speclib`.
- **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 (lb) and the second the upper boundary (ub) of the wavelength ranges to be masked. List must have two items consisting of vectors of length = 2. The first entry is used as lower and the second as upper boundary value. Vectors must contain corresponding lower and upper boundary values consecutively. The masked wavelength range(s) as defined by the lower and upper boundaries are excluded from the object of class `Speclib`.

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 spectra. Filter size is passed as number of bands averaged at both sides of the respective band value.

Usage

meanfilter(spectra, p = 5)

Arguments

spectra  Data frame, matrix or Speclib containing spectra
p        Filter size.
merge

Value
Filtered matrix or Speclib of same dimension as input matrix/Speclib

Author(s)
Lukas Lehnert

See Also
noiseFiltering

Examples

data(spectral_data)

spectra_filtered <- meanfilter(spectral_data, p = 10)
plot(spectra_filtered[1,])
plot(spectral_data[1,], new = FALSE)

merge

Merge speclibs

Description
Merge two Speclibs and their SI 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 (optional) objects of class Speclib.

Value
Object of class Speclib.

Author(s)
Lukas Lehnert
noiseFiltering

See Also

Speclib

Examples

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

## Merge two Speclibs
speclib_merged_1 <- merge(sp1, sp2)
nspectra(speclib_merged_1)

## Merge multiple Speclibs
sp3 <- spectral_data[c(21:30),]
speclib_merged_2 <- merge(sp1, sp2, sp3)
nspectra(speclib_merged_2)

Description

Smoothing of spectral data by Savitzky-Golay, lowess, spline, mean or user-defined filtering approaches.

Usage

noiseFiltering(x, method = "mean", ...)

Arguments

x Object of class Speclib.
method Character string giving the name of 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 the filter functions. The following arguments are important for the predefined methods:

- sgolay: n sets the filter length (must be odd).
- lowess: f defines the smoother span. This gives the proportion of bands in the spectrum which influence the smooth at each value. Larger values give more smoothness.
- spline: n defines at how many equally spaced points spanning the interval interpolation takes place.
- mean: p sets the filter size in number of bands. Note that larger values give more smoothness.

Refer to the links in the details section, and see examples.
Details

Smoothing of spectra by filtering approaches is an essential technique in pre-processing of hyper-spectral data with its contiguous spectra. By stepwise fitting of the spectral channels within a defined window size, it is used to minimize the variances caused by instrumental variations or the high noise levels resulting from the very fine wavelength resolution. Therefore, this function allows filtering using four different methods:

- Savitzky-Golay: Smoothing applying Savitzky-Golay-Filter. See `sgolayfilt` from signal-package for details.
- Lowess: Smoothing applying lowess-Filter. See `lowess` from stats-package for details.
- Spline: Smoothing applying spline-Filter. See `spline` from stats-package for details.
- Mean: Smoothing applying mean-Filter. See `meanfilter` for details.

Value

Object of class Speclib.

Author(s)

Lukas Lehnert, Wolfgang Obermeier

References


See Also

`sgolayfilt`, `lowess`, `spline`, `meanfilter`

Examples

data(spectral_data)

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

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

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

## Mean
meanflt <- noiseFiltering(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 <- noiseFiltering(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")

---

**nri**

*Normalised ratio index*

---

**Description**

Calculate normalised ratio index (nri) for a single given band combination or for all possible band combinations. Calculating nri is a frequently used method to standardize reflectance values and to find relationships between properties of the objects and their spectral data.

**Usage**

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

**Arguments**

- **x** List of class Speclib or of class Nri for print and as.matrix methods.
- **b1** Band 1 given as band number or wavelength.
- **b2** Band 2 given as band number or wavelength.
- **recursive** If TRUE indices for all possible band combinations are calculated. If FALSE, only a single nri for the given bands in b1 and b2 is calculated.
- **bywavelength** Flag to determine if b1 and b2 are band number (bywavelength = FALSE) or wavelength (bywavelength = TRUE) values.

**Details**

Function for nri 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


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 <- spectralResampling(spectral_data, "WorldView2-8",
response_function = FALSE)
nri_WV <- nri(spec_WV, recursive = TRUE)
nri_WV
Nri-methods

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. \( (g)lm.nri \).
• SI: Data.frame containing additional data
• usagehistory: Vector giving information on history of usage of the object.

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

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

## S4 method for signature 'Nri'
as.data.frame(x, na.rm = FALSE, ...)

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

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

getFiniteNri(x)
```
nri_best_performance

Arguments

- **x,object**: Object of class 'Nri'
- **na.rm**: Remove indices containing NA-values. Note that if TRUE, all indices are removed which have at least one NA value.
- **named_matrix**: Flag if column and row names are set to band indices used for the calculation of the nri-values.
- **...**: Further arguments passed to generic functions. Currently ignored.

Author(s)

Lukas Lehnert

See Also

glm.nri, glm, nri

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

```r
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), 
uppertriang = 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
- **uppertriang**: Flag to mark the upper triangle
- **...**: Further arguments passed to glm function. These must be the same as used for initial creation of glm.nri. For mark_nri_best_performance arguments are passed to polygon.
plot.Nri

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

Usage
```r
# 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,
     uppertriang = FALSE, zlog = 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 `hcl` for further explanation. If `colspace == "rgb"`, a vector of length >=2 giving the colours to be interpolated using `colorRamp`.
- **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.
- **uppertriang**: Flag if upper triangle is used for the plot. Note that if `TRUE` the current plot is used instead of starting a new plot.
- **zlog**: Flag indicating if color should be logarithmically scaled. Useful e.g. for p-values.
- **...**: 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

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

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

## Fit generalised linear models between NRI-values and chlorophyll
glmri <- glm.nri(nri_WV ~ chlorophyll, preddata = spec_WV)

## Plot p-values
plot(glmri, range = c(0, 0.05))

## Plot t-values
plot(glmri, coefficient = "t.value")

## Plot only t-values where p-values < 0.001
plot(glmri, coefficient = "t.value",
     constraint = "p.value < 0.001")

## Fit linear models between NRI-values and chlorophyll
lmri <- lm.nri(nri_WV ~ chlorophyll, preddata = spec_WV)

## Plot r.squared
plot(lmri)

## Example for EnMAP (Attention: Calculation time may be long!)
spec_EM <- spectralResampling(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 ~ chlorophyll, 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", uppertriang = TRUE, zlog = TRUE)
lines(c(400,1705),c(400,1705))

## End(Not run)
```

plot.Specfeat

## Description

Plot spectra in objects of class `Specfeat`. Specfeats contain spectral data after applying a transformation such as continuum removal (see function `transformSpeclib`).
Usage

## S4 method for signature 'Specfeat'
plot(x, fnumber = 1:n_features(x), 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 SI table to be used for colour.
changecol Flag indicating if line colours change according to values in column defined by
    stylebysubset
changetype Flag indicating if line types change according to values in column defined by
    stylebysubset
autolegend Flag if legend is plotted.
new Flag if a new plot should be 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, Specfeat

Examples

## Not run:
data(spectral_data)

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

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

## Plot features
plot(featureSelection)

## Advanced plotting example
plot(featureSelection, 1:2, stylebysubset = "season")

plot(featureSelection, 1:2, stylebysubset = "season", changecol = FALSE,
     changetype = TRUE)

## End(Not run)
Description

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

Usage

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

Arguments

- **x**: Object of class Speclib.
- **FUN**: Name of a function (character) or index or ID of single spectrum to plot (integer).
- **new**: If FALSE the plot is added to active existing plot.
- **...**: Further arguments passed to internal plot functions.

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 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")

postprocessASD(x, reference, removeCrossings = TRUE, correctReflectance = TRUE)

Arguments

x Object of class ’Speclib’ containing spectra to be processed.
reference Object of class ’Speclib’ containing single reference spectrum (sensitivity of the white reference standard).
removeCrossings Flag if channel crossings at 1000 nm and 1800 nm should be removed.
correctReflectance Flag if reflectance values should be corrected using the spectrum of the reference.

Value

Object of class Speclib.

Author(s)

Lukas Lehnert
**predictHyperspec**

**See Also**

`speclib`

---

**predictHyperspec**  
*Prediction based on train-object and Speclib*

**Description**

Perform predictions based on a train-object from the `caret`-package and a hyperspectral dataset from `hsdar`. See help file to function `predict.train` of the `caret`-package for general information on prediction with `caret`.

**Usage**

```r
## S4 method for signature 'train,.CaretHyperspectral,missing'
predictHyperspec(object, newdata, preProcess, ...)

## S4 method for signature 'train,.CaretHyperspectral,function'
predictHyperspec(object, newdata, preProcess, ...)
```

**Arguments**

- `object`  
  Object of class `train` from `caret`-package
- `newdata`  
  Object of class `Speclib` or `Nri` to predict on.
- `preProcess`  
  Optional function to be applied on newdata prior to the prediction.
- `...`  
  Further arguments passed to original train function and/or to the preProcess-function.

**Value**

Depending on the settings either a vector of predictions if type = "raw" or a data frame of class probabilities for type = "prob". In the latter case, there are columns for each class. For `predict.list`, a list results. Each element is produced by `predict.train`. See details in `predict.train` in the `caret`-package.

**Author(s)**

Lukas Lehnert

**See Also**

`predict.train`, `Speclib`
## Not run:
## The following example is taken from the journal paper
## "Hyperspectral Data Analysis in R: the hsdar Package"
## under review at the "Journal of Statistical Software"

```r
data(spectral_data)

spectral_data <- noiseFiltering(spectral_data, method = "sgolay", p = 15)

## Convert the chlorophyll measurements stored in the SI dataframe
## from SPAD-values into mg.
SI(spectral_data)$chlorophyll <-
  (117.1 * SI(spectral_data)$chlorophyll) / 
  (148.84 - SI(spectral_data)$chlorophyll)

## Mask spectra
spectral_data <- spectral_data[, wavelength(spectral_data) >= 310 & 
  wavelength(spectral_data) <= 1000]

## Transform reflectance values into band depth applying a segmented upper hull
## continuum removal.
spec_bd <- transformSpeclib(spectral_data, method = "sh", out = "bd")

## Select the chlorophyll absorption features at 460 nm and 670 nm for further
## processing
featureSpace <- specfeat(spec_bd, c(460, 670))

## Calculate all parameters from both selected features such as area, distance
## to Gauss curve etc.
featureSpace <- feature_properties(featureSpace)

## Set response and additional predictor variables for random forest model
featureSpace <- setResponse(featureSpace, "chlorophyll")
featureSpace <- setPredictor(featureSpace,
  names(SI(featureSpace))[-ncol(SI(featureSpace))])

## Define training and cross validation for random forest model tuning
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 5)

## Partition data set for training and validation
training_validation <- createDataPartition(featureSpace)

## Train random forest model based on training-subset
rfe_trained <- train(featureSpace[training_validation$Resample1,],
  trainControl = ctrl, method = "rf")

## Predict on the validation data set
pred <- predictHyperspec(rfe_trained, featureSpace[-training_validation$Resample1,])

## Plot result for visual interpretation
lim <- range(c(SI(featureSpace,i = -training_validation$Resample1)$chlorophyll,
  "value")
```

Examples
pred))
plot(SI(featureSpace,i = -training_validation$Resample1)$chlorophyll, pred,
ylab = "Predicted chlorophyll content",
xlab = "Estimated chlorophyll content",
xlim = lim, ylim = lim)
lines(par()$usr[c(1,2)],par()$usr[c(3,4)], lty = "dotted")
## End(Not run)

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, rsoil = 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.
rsoil An optional object of class 'Speclib' containing the background (soil) reflectance. Note that reflectance values must be in range [0...1].
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!

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description of parameter</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Leaf structure parameter</td>
<td>NA</td>
</tr>
<tr>
<td>Cab</td>
<td>Chlorophyll a+b concentration</td>
<td>$\mu g/cm^2$</td>
</tr>
<tr>
<td>Car</td>
<td>Carotenoid concentration</td>
<td>$\mu g/cm^2$</td>
</tr>
<tr>
<td>Caw</td>
<td>Equivalent water thickness</td>
<td>cm</td>
</tr>
<tr>
<td>Cbrown</td>
<td>Brown pigment</td>
<td>NA</td>
</tr>
<tr>
<td>Cm</td>
<td>Dry matter content</td>
<td>g/cm$^2$</td>
</tr>
<tr>
<td>LAI</td>
<td>Leaf Area Index</td>
<td>NA</td>
</tr>
<tr>
<td>psoil</td>
<td>Dry/Wet soil factor</td>
<td>NA</td>
</tr>
<tr>
<td>hspot</td>
<td>Hotspot parameter</td>
<td>NA</td>
</tr>
<tr>
<td>tts</td>
<td>Solar zenith angle</td>
<td>deg</td>
</tr>
<tr>
<td>tto</td>
<td>Observer zenith angle</td>
<td>deg</td>
</tr>
<tr>
<td>psi</td>
<td>Relative azimuth angle</td>
<td>deg</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>LIDF type</th>
<th>lidfa</th>
<th>lidfb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planophile</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Erectophile</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Plagiophile</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>Extremophile</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Spherical (default)</td>
<td>-0.35</td>
<td>-0.15</td>
</tr>
</tbody>
</table>

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 SI 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


See Also

PROSPECT, Speclib

Examples

```r
## 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 SI table
SI(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)
```

Description

Simulate plant spectrum using PROSPECT 5b or PROSPECT D. The inversion uses the concept after Feret et al. (2008) based on PROSPECT 5B.
Usage

PROSPECT(N = 1.5, Cab = 40, Car = 8, Anth = 1.0, Cbrown = 0.0,
   Cw = 0.01, Cm = 0.009, transmittance = FALSE,
   parameterList = NULL, version = "D")

## Inversion

PROSPECTinvert(x, P0 = NULL, lower = NULL, upper = NULL,
    transmittance_spectra = NULL, sam = FALSE,
    verbose = FALSE, ...)

Arguments

N  Structure parameter
Cab Chlorophyll content
Car Carotenoid content
Anth Anthocyanin 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.
version Sets the version of PROSPECT to be used (either "5B" or "D").
x, transmittance_spectra Speclib(s) containing the reflectance/transmittance values to be simulated during
    inversion of PROSPECT.
P0 Initial set of parameters (N, Cab etc.) as numeric vector.
lower, upper Lower and upper boundaries of parameters as numeric vectors.
sam Logical if spectral angle mapper is used as distance measurement. If FALSE,
    the root mean square error is used. Note that this flag has only an effect if no
    transmittance spectra are passed.
verbose If TRUE, the set of parameters during inversion is printed at each iteration.
... Parameters passed to optim

Details

This function uses the FORTRAN code of PROSPECT model (Version 5B an D). 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description of parameter</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Leaf structure parameter</td>
<td>NA</td>
</tr>
<tr>
<td>Cab</td>
<td>Chlorophyll a+b concentration</td>
<td>µg/cm²</td>
</tr>
<tr>
<td>Car</td>
<td>Carotenoid concentration</td>
<td>µg/cm²</td>
</tr>
<tr>
<td>Anth</td>
<td>Anthocyanin content</td>
<td>µg/cm²</td>
</tr>
<tr>
<td>Cw</td>
<td>Equivalent water thickness</td>
<td>cm</td>
</tr>
<tr>
<td>Cbrown</td>
<td>Brown pigment</td>
<td>NA</td>
</tr>
<tr>
<td>Cm</td>
<td>Dry matter content</td>
<td>g/cm²</td>
</tr>
</tbody>
</table>

The inversion uses the function `optim` and implements the Matlab-Code developed by Feret et al. (2008). Please note that the inversion currently only uses version 5B.

**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**


**See Also**

`PROSAIL`, `optim`, `Speclib`

**Examples**

```r
## 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

```r
data.frame(N = rep.int(seq(0.5, 1.5, 0.5), 2),
            Cab = rep.int(c(40, 3), rep.int(20, 3)))

spectra <- PROSPECT(parameterList = parameter)

## Print SI table
SI(spectra)

## Plot spectra for range from 400 to 800 nm
spectra <- spectra[, wavelength(spectra) >= 400 & wavelength(spectra) <= 800]

plot(subset(spectra, Cab == 20), col = "red", ylim = c(0, 0.5))
plot(subset(spectra, Cab == 40), col = "green", new = FALSE)

## Example for inversion
## Create spectrum using PROSAIL
spectrum <- PROSAIL(LAI = 4)

## Invert PROSPECT using Euclidean and SAM distances
param_rmse <- PROSPECTinvert(spectrum, NULL)  # transmittance_spectra = NULL
param_sam <- PROSPECTinvert(spectrum, transmittance_spectra = NULL, sam = TRUE)

## Model spectrum based on parameters from inversion
pro_rmse <- PROSPECT(N = param_rmse$par[1], Cab = param_rmse$par[2],
                     Car = param_rmse$par[3], Cbrown = param_rmse$par[4],
                     Cw = param_rmse$par[5], Cm = param_rmse$par[6],
                     version = "5B")

pro_sam <- PROSPECT(N = param_sam$par[1], Cab = param_sam$par[2],
                    Car = param_sam$par[3], Cbrown = param_sam$par[4],
                    Cw = param_sam$par[5], Cm = param_sam$par[6],
                    version = "5B")

## Plot result
plot(spectrum, ylim = c(0,0.5))
plot(pro_rmse, new = FALSE, col = "red")
plot(pro_sam, new = FALSE, col = "blue")
legend("topright", legend = c("original spectrum", "inverted with RMSE",
                             "inverted with SAM"), lty = "solid",
        col = c("black", "red", "blue"))
```

### Raster-methods

**Description**

Methods to manipulate, save, convert and plot spectra in Speclibs stored as RasterBrick
Usage

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

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

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

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

Arguments

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

Details

For `extract`, a Speclib is returned containing the data of `y` in the SI. Note that if `y` is a buffer, spatial lines or spatial polygon object, the respective data in `y` is copied for each spectrum so that the length of the SI equals the number of spectra.

For `writeRaster`, the Speclib is returned which is written to file. Please note that data in the SI and the wavelength information cannot be stored in a raster file at present. Therefore, it should be considered to store the entire Speclib as R-data file using the `save`-function in R.

Note for function `brick` that by default the values of the internal brick in the Speclib are copied to the new object. However, new brick objects with differing dimensions, bands etc. may be created if `values == FALSE` is passed as additional arguement.

Value

Speclib for `extract` and `writeRaster`. Object of class `Brick` for `brick`.

Author(s)

Lukas Lehnert
rastermeta

Create list containing rastermeta-information

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
**read.ASD**

*Read ASD binary file*

**Description**

Read spectra stored in ASD binary files using the package 'asdreader'.

**Usage**

```r
read.ASD(f, type = "reflectance", ...)
```

**Arguments**

- `f` Vector with files names to be read.
- `type` Character vector, which type of spectra to return. See ?get_spectra for options.
- `...` Additional arguments passed to get_spectra. Currently ignored.

**Value**

Object of class `Speclib`.

**Author(s)**

Lukas Lehnert

**See Also**

- `speclib`

---

**read_header**

*Get reflectance values*

**Description**

Read ENVI header file

**Usage**

```r
read_header(file, ...)```

**Arguments**

- `file` Path of file to be read.
- `...` Arguments to be passed to specific functions. Currently ignored.
Value

A named list containing the information in the header file

Author(s)

Lukas Lehnert

---

| rededge | Red edge parameter |

**Description**

Derive red edge parameters from hyperspectral data. Red edge is the sharp increase of reflectance values in the near infrared.

**Usage**

rededge(x)

**Arguments**

x List of class SpecLib

**Details**

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

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

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. The following figure shows the location of the red edge parameters in an example second derivation and reflectance spectrum.
Value

A data frame containing parameters for each spectrum.

Author(s)

Hanna Meyer

References


See Also

vegindex, derivative.speclib, noiseFiltering

Examples

# compare R0 for spectra taken in different seasons
data(spectral_data)
rd <- rededge(spectral_data)
boxplot(rd$R0 ~ SI(spectral_data)$season, ylab = "R0")

# visualize red edge parameter of one spectrum
plot(spectral_data[1,], xlim=c(500,900), ylim=c(0,50))
plot(spectral_data[1,], xlim=c(500,900), ylim=c(0,50))
x <- c(rd$l0[1], rd$lp[1], rd$ls[1])
y <- c(rd$R0[1], rd$Rp[1], rd$Rs[1])
points(x, y)
text(x, y, c("l0", "lp", "ls"), pos = 3, offset = 1)
Handling supplementary information (SI) of spectra

Description

Supplementary information (SI) can be any additional data available for each spectrum in a Speclib- or Nri-object. These functions are used to set or return SI-data of a Speclib or Nri-object. Note that SI-data is automatically subsetted if indexing and extracting single spectra from a Speclib- or Nri-object. SI-data may encompass (several) raster files which must have the same extent, resolution and x- and y-dimensions as the raster file used as spectral information.

Usage

```r
## S4 method for signature 'Speclib'
SI(object, i, j)

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

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

## S4 method for signature 'Nri'
SI(object)

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

## S4 replacement method for signature 'Nri,matrix'
SI(object) <- value
```

Arguments

- **object**: Object of class Speclib or Nri.
- **i**: Index of rows to keep. Note that in combination with raster files in the SI, it is MUCH faster to pass row index instead of cutting the resulting data frame. Thus, SI(object,i) should be used instead of SI(object)[i,].
- **j**: Index of columns to keep. See comment above for usage with raster files in the SI.
- **value**: Data frame with nrow(value) == nspectra(object), NULL or vector with length nspectra(object). Alternatively, objects of class RasterLayer are accepted. Please note that the function does not check for integrity in the latter case (e.g., no error will occur if number of spectra does not match number of pixel in the RasterLayer-object).
Details

Names of items in SI are used within the function `subset` to select/deselect spectra via logical expression. Values can be accessed via the "\$"-sign (see examples). Note that the function does not check the integrity of the data stored in the SI (e.g., if there are values for each spectrum).

Value

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

Author(s)

Lukas Lehnert

See Also

`Speclib`, `Nri`

Examples

data(spectral_data)

## Returning SI
si_spec <- SI(spectral_data)
head(si_spec)

## Adding new SI item
SI(spectral_data)$MeasurementID <- c(1:nspectra(spectral_data))
head(SI(spectral_data))

## Replacing SI
SI(spectral_data) <- SI(spectral_data)[,c(1:3)]
head(SI(spectral_data))

## Adding SI to a Speclib without SI
spec_new <- speclib(spectra(spectral_data), wavelength(spectral_data))
## This throws an error
#SI(spec_new)$NewColumn <- 1:nspectra(spec_new)
## This works:
SI(spec_new) <- data.frame(NewColumn = 1:nspectra(spec_new))
## Now, you can add a column as explained above:
SI(spec_new)$SecondCol <- c(1:nspectra(spec_new))*100
## Print SI
SI(spec_new)

## Not run:
## Example for raster file in SI

## 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(3, 10), rep.int(4, 10), rep.int(5, 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", drivername = "GTiff")

infile <- "example_in.tif"
wavelength <- wavelength(spectra)
ra <- speclib(infile, wavelength)
tr <- blockSize(ra)

## Write LAI to separate raster file
LAI <- SI(spectra)$LAI
SI_file <- "example_SI.tif"
SI_raster <- setValues(raster(infile), LAI)
SI_raster <- writeRaster(SI_raster, SI_file)

## Read LAI file and calculate NDVI for each pixel where LAI >= 1
outfile <- "example_result_ndvi.tif"
SI(ra) <- raster(SI_file)
names(SI(ra)) <- "LAI"
res <- writeStart(ra, outfile, overwrite = TRUE, nl = 1)
for (i in 1:tr$n)
{
  v <- getValuesBlock(ra, row=tr$row[i], nrows=tr$nrows[i])
  mask(v) <- c(1350, 1450)
  LAI <- SI(v)$LAI
  v <- as.matrix(vegindex(v, index="NDVI"))
  v[LAi < 1] <- NA
  res <- writeValues(res, v, tr$row[i])
}
res <- writeStop(res)

## End(Not run)

---

**SMGM**

**Description**

Calculate Gaussian model on soil spectra
Usage

\texttt{smgm(x, percentage = TRUE, gridsize = 50)}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} Object of class Speclib.
\item \texttt{percentage} \hspace{1cm} Flag if spectra in \texttt{x} are in range [0, 100]. If \texttt{FALSE}, the spectra are scaled to [0,100].
\item \texttt{gridsize} \hspace{1cm} Size of the grid used to perform least squares approximation.
\end{itemize}

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 \texttt{transformSpeclib}). 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 \texttt{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 SI 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


See Also

\texttt{soilindex, Speclib}
Examples

```r
## Use PROSAIL to simulate spectra with different soil moisture content
Spektr.lib <- noiseFiltering(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)

SI(smgm_val)
```

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 `get_reflectance` 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>Reference*</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI_TM</td>
<td>(((TM_1^2 + TM_2^2 + TM_3^2)/3)^{0.5})**</td>
<td>Mathieu et al. (1998)</td>
</tr>
</tbody>
</table>
### soilindex

<table>
<thead>
<tr>
<th>Index</th>
<th>Formula</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI_TM</td>
<td>((TM_3 - TM_2)/(TM_3 + TM_2))**</td>
<td>Escadafal and Huete (1991)</td>
</tr>
<tr>
<td>HI_TM</td>
<td>((2 \cdot TM_3 - TM_2 - TM_1)/(TM_2 - TM_1))**</td>
<td>Escadafal et al. (1994)</td>
</tr>
<tr>
<td>NDI</td>
<td>((R_{840} - R_{1650})/(R_{840} + R_{1650}))</td>
<td>McNairn, H. and Protz, R. (1993)</td>
</tr>
<tr>
<td>NSMI</td>
<td>((R_{1800} - R_{2119})/(R_{1800} + R_{2119}))</td>
<td>Haubrock et al. (2008)</td>
</tr>
<tr>
<td>RI</td>
<td>(R_{4093}^2/(R_{447} \cdot R_{556}))</td>
<td>Ben-Dor et al. (2006)</td>
</tr>
<tr>
<td>RI_TM</td>
<td>(TM_3^2/(TM_1 \cdot TM_2^3))**</td>
<td>Madeira et al. (1997), Mathieu et al. (1998)</td>
</tr>
<tr>
<td>SI_TM</td>
<td>((TM_3 - TM_1)/(TM_3 + TM_1))**</td>
<td>Escadafal et al. (1994)</td>
</tr>
<tr>
<td>SWIR SI</td>
<td>(-41.59 \cdot (R_{2210} - R_{2090}) + 1.24 \cdot (R_{2280} - R_{2090}) + 0.64)</td>
<td>Lobell et al. (2001)</td>
</tr>
</tbody>
</table>

* 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 `spectralResampling` 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 absorption features

Description

Function isolates absorption features from band depth or ratio transformed reflectance spectra.

Usage

specfeat(x, FWL, tol = 1e-7)

Arguments

x
Object of class Speclib containing the band depth or ratio transformed reflectance spectra.

FWL
A vector containing one wavelength included in each feature to be isolated, e.g. the major absorption features. Features which include these specified wavelengths will be isolated.

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.

Details

A feature is defined as the part of the spectrum between two fix points in the transformed spectra (band depth values of 0). This function separates features at wavelengths of interest according to this rule. Hence it allows a subsequent characterization of the features of interest, e.g. via feature_properties or visual inspection via plot.Specfeat. The typical workflow to obtain feature properties is to first calculate the band depth transformSpeclib, then isolate the absorption features specfeat. Optionally, cut_specfeat allows to cut the features at specified wavelengths. Finally use feature_properties to retrieve characteristics of the features.

Value

An object of class Specfeat containing the isolated features.

Author(s)

Hanna Meyer and Lukas Lehnert
Specfeat-class

*Specfeat class*

Description

Class to handle spectral feature data. Spectral features are absorption (transmission or reflection) bands defined e.g. by continuum removal (see `transformSpeclib`).

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 `specfeat`.

Note

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

Author(s)

Lukas Lehnert

See Also

`Speclib`, `specfeat`
Methods to create objects of class Speclib

Description

Methods to create objects of class Speclib from various data sources such as matrixes and raster files (e.g. GeoTiff).

Usage

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

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

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

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

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

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

show(object)

is.speclib(x)

### Arguments

- **spectra**: Data frame, matrix or raster object of class 'RasterBrick' or 'SpatialGridDataFrame' with spectral data. Alternatively, spectra may be the path to a raster file containing hyperspectral data.

- **x, object**: Object to be converted to or from Speclib. For conversion to Speclib it can be 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.

- **...**: Further arguments passed to specific (generic) functions. They encompass particularly the following additional parameters:
  - **fwhm**: Vector containing full-width-half-max values for each band. Default: NULL
  - **SI**: Data frame with supplementary information to each spectrum. Default: NULL
  - **transformation**: Kind of transformation applied to spectral data (character). See `transformSpeclib` for available ones. If transformation = NULL, no transformation is assumed (default).
  - **usagehistory**: Character string or vector used for history of usage. Default: NULL
  - **continuousdata**: Flag indicating if spectra are quasi continuous or discrete sensor spectra (deprecated). Default: "auto"
  - **wlunit**: Unit of wavelength in spectra. Default: "nm". See details how other units are treated.
  - **xlabel**: Label of wavelength data to be used for plots etc. Default: "Wavelength"
  - **ylabel**: Label of spectral signal to be used for plots etc. Default: "Reflectance"
  - **rastermeta**: List of meta information for SpatialGridDataFrame. If missing, meta data in speclib is used. Use function `rastermeta` to create valid objects. Default: NULL

### Details

**Spectral data**: The spectral data (usually reflectance values) are stored in an object of class '.Spectra'. 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. 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/package=raster 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 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 spectralResampling may only work correctly with continuous data!

The internal and recommended wavelength unit is nm. If Speclibs are created with wavelength values in other units than nm as passed by wlunit-argument, wavelength values are automatically converted to nm. In this case, functions requiring to pass wavelength information (e.g., mask etc) expect the unit to match the one initially set. The only exception is the Nri-class which always uses and expects nm as unit of passed wavelength values. The following units are automatically detected: μm, µm, nm, mm, cm, dm, m.

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

- wavelength: Vector with wavelength information. Always stored in nm.
- 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.
- SI: Data frame containing additional data to each spectrum. May be used for linear regression etc. Handle with function SI.
- usagehistory: Vector giving information on history of usage of speclib. Handle with function usagehistory.

Value

An object of class Speclib containing the following slots is returned:
• wavelength: Vector with wavelength information. Always stored in nm.
• 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`.
• SI: Data frame containing additional data to each spectrum. May be used for linear regression etc. Handle with function `SI`.
• 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,
SI

Examples

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

spectra <- speclib(spectra, wavelength)
Details

**Spectral data:** The spectral data (usually reflectance values) are stored in an object of class `.Spectra`. 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`. 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/package=raster](https://CRAN.R-project.org/package=raster) 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 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 `spectralResampling` may only work correctly with continuous data!

The internal and recommended wavelength unit is nm. If Speclibs are created with wavelength values in other units than nm as passed by `wlunit`-argument, wavelength values are automatically converted to nm. In this case, functions requiring to pass wavelength information (e.g., `mask` etc) expect the unit to match the one initially set. The only exception is the Nri-class which always uses and expects nm as unit of passed wavelength values. The following units are automatically detected: `mu`, `µm`, `nm`, `mm`, `cm`, `dm`, `m`.

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

- wavelength: Vector with wavelength information. Always stored in nm.
- 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`.
- SI: Data frame containing additional data to each spectrum. May be used for linear regression etc. Handle with function SI.
- usagehistory: Vector giving information on history of usage of speclib. Handle with function `usagehistory`. 

98

**Speclib-class**
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, SI

Description

Functions for processing large hyperspectral raster files using the low-level functions provided by the raster-package. For a detailed overview see the vignette "Writing functions for large raster files" shipped along with the raster-package.

Usage

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

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

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

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

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

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

## S4 method for signature 'Speclib'
writeStop(x)
Arguments

- **x**: Object of class `Speclib`.
- **filename**: Name of the new file to create.
- **v**: Object to write the data to file. May be one of the following classes: "Speclib", "matrix" or "numeric".
- **start**: Integer. Row number (counting starts at 1) from where to start writing `v`.
- **...** Further arguments passed to respective functions in the `raster`-packages.

Author(s)

Lukas Lehnert

Examples

```r
## 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", drivername = "GTiff")

## Examples for Speclib using file example_in.tif
## Example 1:
## Noise reduction in spectra
infile <- "example_in.tif"
outfile <- "example_result_1.tif"
wavelength <- spectra$wavelength
ra <- speclib(infile, wavelength)
tr <- blockSize(ra)
res <- writeStart(ra, outfile, overwrite = TRUE)
for (i in 1:tr$n)
{
  v <- getValuesBlock(ra, row=tr$row[i], nrows=tr$nrows[i])
  v <- noiseFiltering(v, method="sgolay", n=25)
  res <- writeValues(res, v, tr$row[i])
}
```
## Example 2:
## masking spectra and calculating vegetation indices
outfile <- "example_result_2.tif"
n_veg <- as.numeric(length(vegindex()))
res <- writeStart(ra, outfile, overwrite = TRUE, nl = n_veg)
for (i in 1:tr$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)

---

**spectra**

*Handling spectra*

### Description

Returning and setting spectra in Speclib

### Usage

```r
## S4 method for signature 'Speclib'
spectra(object, i, j, ...)  
## 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.
- **i**: Index of spectra to return. If missing all spectra are returned.
- **j**: Index of bands to return. If missing all bands are returned.
... Passed to internal function. Currently only one parameter is accepted: return_names: Logical indicating, if names of columns and rows should be set to bandnames and idSpeclib.

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")

_________________________
spectralInterpolation  Interpolate spectra_________________________

Description
Interpolate spectra to user defined bands. Currently, only a linear interpolation is supported

Usage
spectralInterpolation(x, sensor)

Arguments
x Object of class Speclib.
sensor data.frame containing definition of sensor characteristics. See details section for further information.
The characteristics must 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.

Value

Object of class Speclib containing the updated version of x.

Author(s)

Lukas Lehnert

See Also

spectralResampling

Examples

```r
## Load example data
data(spectral_data)
## Create sensor featuring 10 times higher spectral resolution
bounds <- seq(min(wavelength(spectral_data)),
               max(wavelength(spectral_data)),
               length.out = nbands(spectral_data)*10)
sensor <- data.frame(lb = bounds[-1*100 + 1], ub = bounds[-1])
## Interpolate first spectrum
inter <- spectralInterpolation(spectral_data[,1], sensor = sensor)
```

spectralResampling  Spectral resampling

Description

Resample spectra to (satellite) sensors

Usage

```r
spectralResampling(x, sensor, rm.NA = TRUE, continuousdata = "auto",
             response_function = TRUE)
```
spectralResampling

Arguments

- **x**: Object of class Speclib. Data to be spectrally resampled.
- **sensor**: Character or data.frame 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 Speclib 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 spectra[min(ch):max(ch)] is calculated. If response_function is an object of class Speclib the function assumes that the spectra in the object are spectral response values. In this case the wavelength dimension determines the spectral response values for the respective wavelength and the sample dimension separates between the different bands. Note that if response_function is an object of class Speclib, sensor may be missing. In this case the function calculates the central wavelength and the fwhm-values from the spectral response functions.

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.

The third option is to use a Speclib containing the spectral response values instead of reflectances. In this case, the sensor-argument may be missing and the function automatically determines the sensor’s central wavelength and the fwhm-values based on the spectral response values. See examples.

If sensor characteristics are defined manually and no Speclib with spectral response values is passed, a Gaussian response is assumed.

Value

Object of class Speclib

Note

The spectral response functions are kindly provided by the operators of the 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
**spectralResampling**

**Author(s)**

Lukas Lehnert

**See Also**

`get.sensor.characteristics`, `get.gaussian.response`

**Examples**

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

## Resample to RapidEye
data_RE <- spectralResampling(spectral_data, "RapidEye",
                               response_function = TRUE)

## Plot resampled spectra
plot(data_RE)

## Compare different methods of spectral resampling
par(mfrow=c(1,3))
ga <- spectralResampling(spectral_data, "RapidEye",
                          response_function = FALSE)
plot(ga)
re <- spectralResampling(spectral_data, "RapidEye",
                          response_function = TRUE)
plot(re)
o <- spectralResampling(spectral_data, "RapidEye",
                        response_function = NA)
plot(o)

## Usage of Speclib with spectral response values
## Define 3 bands (RGB)
center <- c(460, 530, 600)
fwhm <- 70
wl <- c(310:750)

## Create spectral response with gaussian density function
response <- speclib(t(sapply(center, function(center, wl, fwhm)
                      {a <- dnorm(wl, mean = center, sd = fwhm/2)
                       a <- (a-min(a))/(max(a) - min(a))
                       return(a)
                      }, wl, fwhm)), wl)

## Plot response functions
for (i in 1:3)
  plot(response[i,], new = i == 1, col = c("blue", "green", "red")[i])

## Perform resampling
rgb_data <- spectralResampling(spectral_data, response_function = response)
```
spectral_data

Hyperspectral samples

Description
Hyperspectral samples from a FACE experiment in Germany

Usage
data(spectral_data)

Format
An object of class Speclib

Details
Data has been sampled during vegetation period 2014 in spring and summer. 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)
Wolfgang A. Obermeier, Lukas Lehnert, Hanna Meyer

sr
Simple ratio index

Description
Calculate simple ratio index (sr) for a single given band combination or for all possible band combinations. Calculating sr is a frequently used method to standardize reflectance values and to find relationships between properties of the objects and their spectral data.

Usage
sr(x, b1, b2, recursive = FALSE, bywavelength = TRUE)

Arguments
x List of class Speclib or of class Nri for print and as.matrix methods.
b1 Band 1 given as band number or wavelength.
b2 Band 2 given as band number or wavelength.
recursive If TRUE indices for all possible band combinations are calculated. If FALSE, only a single sr for the given bands in b1 and b2 is calculated.
bywavelength Flag to determine if b1 and b2 are band number (bywavelength = FALSE) or wavelength (bywavelength = TRUE) values.
Details

Function performs the following calculation:

\[ nri_{B_1, B_2} = \frac{R_{B_1}}{R_{B_2}}; \]

with \( R \) being reflectance values at wavelength \( B_1 \) and \( B_2 \), 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 simple ratio indices.

Author(s)

Lukas Lehnert

See Also

nri, glm.nri, glm, Speclib, Nri

Examples

data(spectral_data)

## Calculate SR of Jordan (1969) (R_{800}/R_{680})
sr_600_680 <- sr(spectral_data, b1=800, b2=680)

## Calculate all possible combinations for WorldView-2-8
spec_WV <- spectralResampling(spectral_data, "WorldView2-8",
    response_function = FALSE)
sr_WV <- sr(spec_WV, recursive = TRUE)
sr_WV

subset.nri  
Subsetting Nri-objects

Description

Return subsets of Nri-objects which meet conditions.

Usage

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

---
Arguments

- **x**: Object of class ‘Nri’.
- **subset**: Logical expression indicating spectra to keep: missing values are taken as false. See details section.
- ... Further arguments passed to `agrep`.

Details

Matchable objects are SI data. Use column names to identify the respective SI. See `SI` to access SI of a `Nri`. IDs of samples may be accessed using “id.nri” as variable name.

Value

Object of class `Nri`.

Author(s)

Lukas Lehnert

See Also

`Nri`, `SI`

Examples

```r
data(spectral_data)

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

## Return names of SI data
names(SI(nri_WV))

## Divide into both seasons
sp_summer <- subset(nri_WV, season == "summer")
sp_spring <- subset(nri_WV, season == "spring")

## Print both Nri-objects
sp_summer
sp_spring

## Divide into both seasons and years
sp_summer_14 <- subset(nri_WV, season == "summer" & year == 2014)
sp_spring_14 <- subset(nri_WV, season == "spring" & year == 2014)

## Print both Nri-objects
sp_summer_14
sp_spring_14
```
Description

Function to return subsets of Speclibs by defined conditions.

Usage

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

Arguments

- **x**: Object of class 'Speclib'.
- **subset**: Logical expression indicating spectra to keep: missing values are taken as false. Multiple expressions can be applied using logical operators AND and OR. See details section.
- **...**: Further arguments passed to `agrep`.

Details

Matchable objects are SI data. Use column names to identify the respective SI. See `SI` to access SI of a Speclib. IDs of spectra may be accessed using "id.speclib" as variable name. To subset certain wavelength ranges of a Speclib refer to `mask`.

Value

Object of class Speclib.

Author(s)

Lukas Lehnert, Wolfgang Obermeier

See Also

`Speclib`, `SI`, `mask`
Examples

data(spectral_data)

## Return names of SI data
names(SI(spectral_data))

## Divide into both seasons
sp_summer <- subset(spectral_data, season == "summer")
sp_spring <- subset(spectral_data, season == "spring")

## Divide into both seasons and years
sp_summer_14 <- subset(spectral_data, season == "summer" & year == 2014)
sp_spring_14 <- subset(spectral_data, season == "spring" & year == 2014)

## Plot all speclibs
plot(sp_spring_14, col="green", ylim = c(0,80))
plot(sp_summer_14, col="red", new = FALSE)

t.test

Description

Performs Student’s t-tests for normalized ratio index 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, Nri

Examples

data(spectral_data)

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

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

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

transformSpeclib Transform spectra

Description

Transform spectra by using convex hull or segmented upper hull

Usage

transformSpeclib(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", "difference" 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 \( \lambda \).
- "difference": the spectra are transformed by
  \[ diff_\lambda = CV_\lambda - R_\lambda \]
- "ratio": the spectra are transformed by
  \[ \text{ratio}_\lambda = \frac{R_\lambda}{CV_\lambda} \]

In some cases it might be useful to apply noiseFiltering 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 out != "raw" an object of class Speclib containing transformed spectra is returned. Otherwise the return object will be of class Clman.

**Note**

For large Speclibs, it may be feasible to run the function on multiple cores. See hsdar_parallel() for further information.

**Author(s)**

Hanna Meyer and Lukas Lehnert

**References**


See Also

clman, addcp, deletecp, checkhull

Examples

```r
## Example spectrum for wavelength values
## between 400 and 1000 nm
example_spectrum <- PROSPECT()[,c(1:600)]

## Default (convex hull and band depth)
ch_bd <- transformSpeclib(example_spectrum)

## Construct convex hull but calculate ratios
ch_ratio <- transformSpeclib(example_spectrum, out = "ratio")

## Return continuum line of convex hull
ch_raw <- transformSpeclib(example_spectrum, out = "raw")

## Plot results
par(mfrow=c(2,2))
plot(example_spectrum)
plot(ch_raw, ispec = 1, main = "Continuum line",
     ylim = c(0,0.5))
plot(ch_bd, main = "Band depth")
plot(ch_ratio, main = "Ratio")

## Same example but with segmented hull

## Segmented hull and band depth
sh_bd <- transformSpeclib(example_spectrum, method = "sh",
                           out = "bd")

## Segmented hull and ratios
sh_ratio <- transformSpeclib(example_spectrum, method = "sh",
                             out = "ratio")

## Return continuum line of segmented hull
sh_raw <- transformSpeclib(example_spectrum, method = "sh",
                          out = "raw")

## Plot results
par(mfrow=c(2,2))
plot(example_spectrum)
plot(sh_raw, ispec = 1, main = "Continuum line",
     ylim = c(0,0.5))
plot(sh_bd, main = "Band depth")
plot(sh_ratio, main = "Ratio")
```

unmix

Unmix spectra
Description

Perform linear spectral unmixing on hyperspectral data 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 HyperSpecRaster (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 \( f_1, f_2, ..., f_n \geq 0 \) 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
updatecl  

References


Examples

```r
## Not run:
## Use PROSAIL to generate some vegetation spectra with different LAI
parameter <- data.frame(LAI = seq(0, 1, 0.01))
spectral_data <- PROSAIL(parameterList = parameter)

## Get endmember spectra
## Retrieve all available spectra
avl <- USGS_get_available_files()

## Download all spectra matching "grass-fescue"
grass_spectra <- USGS_retrieve_files(avl = avl, pattern = "grass-fescue")
limestone <- USGS_retrieve_files(avl = avl, pattern = "limestone")

## Integrate all spectra to Quickbird
grass_spectra_qb <- spectralResampling(grass_spectra[1,], "Quickbird")
limestone_qb <- spectralResampling(limestone, "Quickbird")
spectral_data_qb <- spectralResampling(spectral_data, "Quickbird")

em <- speclib(spectra = rbind(spectra(grass_spectra_qb),
                                spectra(limestone_qb))/100,
               wavelength = wavelength(limestone_qb))

## Unmix
unmix_res <- unmix(spectral_data_qb, em)

unmix_res

plot(unmix_res$fractions[1,] ~ SI(spectral_data_qb)$LAI, type = "l",
xlab = "LAI", ylab = "Unmixed fraction of vegetation")
## End(Not run)
```

updatecl  

Check transformed Speclib

Description

Update a transformed Speclib with a re-calculated hull

Usage

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

- **x**: Object of class `Speclib` transformed by `transformSpeclib`.
- **hull**: Hull to be applied to `x`. Output of function `makehull`.

Details

In some cases, it might be desirable to manually adapt automatically constructed segmended hulls (`transformSpeclib`). For example local maxima could be removed because they are very small and maybe afflicted with uncertainties which might legitimate it to manipulate the continuum line. Therefore, hsdar provides functions to remove and add "continuum points" from or to a continuum line. Manually adapted continuum lines can then be used to update band depth or ratio transformation. Handle these functions with care to avoid continuum lines too much build by subjective decisions. In the typical workflow, spectra are first transformed (`transformSpeclib`). Continuum points can then be retrieved (`getcp`) and manually adapted by adding `addcp` and deleting (`deletecp`) of points. Use `checkhull` to check for errors. If all uncertainties are removed, re-calculate the hull (`makehull`) and update the transformed spectrum (`updatecl`).

Value

Object of class `Speclib`.

Author(s)

Lukas Lehnert and Hanna Meyer

See Also

`transformSpeclib`, `makehull`, `Speclib`

Examples

```r
## 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 <- transformSpeclib(spec, method = "sh", out = "raw")

## Plot original line
par(mfrow = c(1,2))
plot(spec_clman, ispec = 1, xlim = c(2480, 2500), ylim=c(0.022,0.024))

## 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, xlim = c(2480, 2500), ylim=c(0.022,0.024))

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

## Re-calculate hull
hull <- makehull(spec_clman, 1)

## Transform spectra using band depth
spec_bd <- transformSpeclib(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

**Usage history**

Function to read and write history of usage for Speclibs. Similar to a log file, the history of usage records processing steps applied to a Speclib.

#### Description

Function to read and write history of usage for Speclibs. Similar to a log file, the history of usage records processing steps applied to a Speclib.

#### Usage

```r
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 containing 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) <- NULL
spectral_data

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

---

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 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.
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 get_reflectance 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>Reference*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boochs</td>
<td>$D_{703}$</td>
<td>Boochs et al. (1990)</td>
</tr>
<tr>
<td>Boochs2</td>
<td>$D_{720}$</td>
<td>Boochs et al. (1990)</td>
</tr>
<tr>
<td>CAI</td>
<td>$0.5 \cdot (R_{2000} + R_{2200}) - R_{2100}$</td>
<td>Nagler et al. (2003)</td>
</tr>
<tr>
<td>CARI</td>
<td>$a = (R_{700} - R_{550})/150 \quad b = R_{550} - (a \cdot 550)$</td>
<td>Kim et al. (1994)</td>
</tr>
<tr>
<td></td>
<td>$R_{700} \cdot \text{abs}(a \cdot 670 + R_{670} + b)/R_{670}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(a^2 + 1)^{0.5}$</td>
<td></td>
</tr>
<tr>
<td>Carter</td>
<td>$R_{695}/R_{740}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>Carter2</td>
<td>$R_{695}/R_{760}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>Carter3</td>
<td>$R_{605}/R_{760}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>Carter4</td>
<td>$R_{710}/R_{760}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>Carter5</td>
<td>$R_{695}/R_{670}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>Carter6</td>
<td>$R_{550}$</td>
<td>Carter (1994)</td>
</tr>
<tr>
<td>CI</td>
<td>$R_{675} \cdot R_{690}/R_{683}^{2}$</td>
<td>Zarco-Tejada et al. (2003)</td>
</tr>
<tr>
<td>CI2</td>
<td>$R_{760}/R_{700} - 1$</td>
<td>Gitelson et al. (2003)</td>
</tr>
<tr>
<td>CIInt</td>
<td>$\int_{600nm}^{735nm} R$</td>
<td>Oppelt and Mauser (2004)</td>
</tr>
<tr>
<td>CRI1</td>
<td>$1/R_{515} - 1/R_{550}$</td>
<td>Gitelson et al. (2003)</td>
</tr>
<tr>
<td>CRI2</td>
<td>$1/R_{515} - 1/R_{770}$</td>
<td>Gitelson et al. (2003)</td>
</tr>
<tr>
<td>CRI3</td>
<td>$1/R_{515} - 1/R_{550} \cdot R_{770}$</td>
<td>Gitelson et al. (2003)</td>
</tr>
<tr>
<td>CRI4</td>
<td>$1/R_{515} - 1/R_{700} \cdot R_{770}$</td>
<td>Gitelson et al. (2003)</td>
</tr>
<tr>
<td>D1</td>
<td>$D_{730}/D_{706}$</td>
<td>Zarco-Tejada et al. (2003)</td>
</tr>
<tr>
<td>D2</td>
<td>$D_{705}/D_{722}$</td>
<td>Zarco-Tejada et al. (2003)</td>
</tr>
<tr>
<td>Datt</td>
<td>$(R_{850} - R_{710})/(R_{850} - R_{680})$</td>
<td>Datt (1999b)</td>
</tr>
<tr>
<td>Datt2</td>
<td>$R_{850}/R_{710}$</td>
<td>Datt (1999b)</td>
</tr>
<tr>
<td>Datt3</td>
<td>$D_{754}/D_{704}$</td>
<td>Datt (1999b)</td>
</tr>
<tr>
<td>Datt4</td>
<td>$R_{672}/(R_{550} \cdot R_{708})$</td>
<td>Datt (1998)</td>
</tr>
<tr>
<td>Datt5</td>
<td>$R_{672}/R_{550}$</td>
<td>Datt (1998)</td>
</tr>
<tr>
<td>Datt6</td>
<td>$(R_{860})/(R_{550} \cdot R_{708})$</td>
<td>Datt (1998)</td>
</tr>
<tr>
<td>Datt7</td>
<td>$(R_{860} - R_{2218})/(R_{860} - R_{1928})$</td>
<td>Datt (1999a)</td>
</tr>
<tr>
<td>Datt8</td>
<td>$(R_{860} - R_{1788})/(R_{860} - R_{1928})$</td>
<td>Datt (1999a)</td>
</tr>
<tr>
<td>DD</td>
<td>$(R_{749} - R_{720}) - (R_{701} - R_{672})$</td>
<td>le Maire et al. (2004)</td>
</tr>
<tr>
<td>DDn</td>
<td>$2 \cdot (R_{710} - R_{660} - R_{760})$</td>
<td>le Maire et al. (2008)</td>
</tr>
<tr>
<td>DPI</td>
<td>$(D_{688} \cdot D_{710})/D_{697}$</td>
<td>Zarco-Tejada et al. (2003)</td>
</tr>
<tr>
<td>DWSI1</td>
<td>$R_{800}/R_{1660}$</td>
<td>Apan et al. (2004)</td>
</tr>
</tbody>
</table>
DWSI2: \( R_{1660}/R_{550} \)
DWSI3: \( R_{1660}/R_{680} \)
DWSI4: \( R_{550}/R_{680} \)
DWSI5: \( (R_{800} + R_{550})/(R_{1660} + R_{680}) \)
EGFN: \( (\text{max}(D_{550:750}) - \text{max}(D_{500:550}))/\text{max}(D_{550:750}) \)
EGFR: \( \text{max}(D_{550:750})/\text{max}(D_{500:550}) \)
EVI: \( 2.5 \cdot ((R_{800} - R_{670})/(R_{800} - R_{670} - (7.5 \cdot R_{475}) + 1)) \)
GDVI: \( (R_{800} - R_{680})/(R_{800} + R_{680}) \)
GI: \( R_{554}/R_{677} \)
Gitelson1: \( 1/R_{700} \)
Gitelson2: \( (R_{750} - R_{800}/R_{695} - R_{740}) - 1 \)
GMI1: \( R_{750}/R_{550} \)
GMI2: \( R_{750}/R_{700} \)
Green NDVI: \( (R_{800} - R_{550})/(R_{800} + R_{550}) \)
LWVI1: \( (R_{1094} - R_{983})/(R_{1094} + R_{983}) \)
LWVI2: \( (R_{1094} - R_{1205})/(R_{1094} + R_{1205}) \)
Macciioni: \( (R_{740} - R_{710})/(R_{740} - R_{680}) \)
MCARI: \( (R_{700} - R_{670}) - 0.2 \cdot (R_{700} - R_{550}) \)
MCARI2: \( (R_{750} - R_{705}) - 0.2 \cdot (R_{750} - R_{550}) \)
MCARI3: \( R_{682}/(R_{682} + R_{553}) \)
NDVI: \( R_{860}/(R_{860} + R_{1240}) \)

MREIP: Red-edge inflection point using Gaussian fit
MSAI1: \( 0.5 \cdot (2 \cdot R_{800} + 1 - (2 \cdot R_{800} + 1)^2 - 8 \cdot (R_{800} - R_{670})^{0.5}) \)
MSI: \( R_{1600}/R_{817} \)
MRR: \( (R_{800} - R_{445})/(R_{680} - R_{445}) \)
MRR2: \( (R_{750} - R_{705}) - 1/(R_{750}/R_{705})^{0.5} \)
MRR705: \( (R_{750} - R_{445})/(R_{705} - R_{445}) \)
MTCI: \( (R_{754} - R_{709})/(R_{709} - R_{681}) \)
MTVI: \( 1.2 \cdot (1.2 \cdot (R_{800} - R_{550}) - 2.5 \cdot (R_{670} - R_{550})) \)
NDVI: \( (R_{800} - R_{680})/(R_{800} + R_{680}) \)
NDVI2: \( (R_{750} - R_{705})/(R_{750} + R_{705}) \)

\( mND705 \): \( (R_{750} - R_{705})/(R_{750} + R_{705} - 2 \cdot R_{445}) \)
\( mNDVI \): \( (R_{800} - R_{680})/(R_{800} + R_{680} - 2 \cdot R_{445}) \)
\( MPRI \): \( (R_{515} - R_{530})/(R_{515} + R_{530}) \)

\( mREIP \): Red-edge inflection point using Gaussian fit
\( MSAVI \): Miller et al. (1990)
\( MSAI1 \): Qi et al. (1994)
\( MSI \): Hunt and Rock (1989)
\( MSR \): Sims and Gamon (2002)
\( MSR2 \): Chen (1996)
\( MSR705 \): Sims and Gamon (2002)
\( MTCI \): Dash and Curran (2004)
\( MTVI \): Haboudane et al. (2004)
\( NDLI \): Serrano et al. (2002)
\( NDLNI \): Serrano et al. (2002)
\( NDVI \): Tucker (1979)
\( NDVI2 \): Gitelson and Merzlyak (1994)
\( NDVI3 \): Gandia et al. (2004)
\( NDWI \): Gao (1996)
NPCI \[ \frac{(R_{680} - R_{430})}{(R_{680} + R_{430})} \]

OSAVI \[ (1 + 0.16) \cdot \frac{(R_{800} - R_{670})}{(R_{800} + R_{670} + 0.16)} \]

OSAVI2 \[ (1 + 0.16) \cdot \frac{(R_{750} - R_{705})}{(R_{750} + R_{705} + 0.16)} \]

PARS \[ R_{746}/R_{513} \]

PRI \[ \frac{(R_{531} - R_{570})}{(R_{531} + R_{570})} \]

PRI\_norm \[ PRI \cdot (-1)/(RDVI \cdot R_{700}/R_{670}) \]

PRI*CI2 \[ PRI \cdot CI2 \]

PSRI \[ (R_{678} - R_{500}/R_{750}) \]

PSSR \[ R_{600}/R_{635} \]

PSND \[ (R_{800} - R_{470})/(R_{800} - R_{470}) \]

PW1 \[ R_{600}/R_{970} \]

RDVI \[ \frac{(R_{800} - R_{670})}{\sqrt{R_{800} + R_{670}}} \]

REP\_LE \[ \text{Red-edge position through linear extrapolation.} \]

REP\_Li \[ R_{re} = \frac{(R_{670} + R_{750})}{2} \]

700 + 40 \cdot ((R_{re} - R_{700})/(R_{740} - R_{700}))

SAVI \[ (1 + L) \cdot \frac{(R_{800} - R_{670})}{(R_{800} + R_{670} + L)} \]

SIP1 \[ (R_{800} - R_{445})/(R_{800} - R_{680}) \]

SPVI \[ 0.4 \cdot 3.7 \cdot \frac{(R_{800} - R_{670})}{1.2 \cdot ((R_{530} - R_{670})^2)^{0.5}} \]

SR \[ R_{800}/R_{680} \]

SR1 \[ R_{750}/R_{700} \]

SR2 \[ R_{752}/R_{690} \]

SR3 \[ R_{750}/R_{550} \]

SR4 \[ R_{700}/R_{670} \]

SR5 \[ R_{675}/R_{700} \]

SR6 \[ R_{750}/R_{710} \]

SR7 \[ R_{440}/R_{690} \]

SR8 \[ R_{515}/R_{550} \]

SRPI \[ R_{430}/R_{680} \]

SRWI \[ R_{850}/R_{1240} \]

\( \sum_{k=626}^{795} D1 \_i \)

\( \sum_{i=680}^{860} D1 \_i \)

SWIR FI \[ \frac{R_{2133}^2}{(R_{2225} \cdot R_{2329})} \]

SWIR LI \[ 3.87 \cdot (R_{2210} - R_{2090}) - 27.51 \cdot (R_{2280} - R_{2090}) - 0.2 \]

SWIR SI \[ -41.59 \cdot (R_{2210} - R_{2090}) + 1.24 \cdot (R_{2280} - R_{2090}) + 0.64 \]

Penuelas et al. (1994)

Rondeaux et al. (1996)

Wu et al. (2008)

Chappelle et al. (1992)

Gamon et al. (1992)

Zarco-Tejada et al. (2013)

Garrity et al. (2011)

Merzlyak et al. (1999)

Blackburn (1998)

Blackburn (1998)

Penuelas et al. (1997)

Roujean and Breon (1995)

Cho and Skidmore (2006)

Guyot and Baret (1988)

Huete (1988)

Penuelas et al. (1995),

Penuelas et al. (1995a)

Vincini et al. (2006)

Jordan (1969)

Gitelson and Merzlyak (1997)

Gitelson and Merzlyak (1997)

Gitelson and Merzlyak (1997)

McMurtrey et al. (1994)

Chappelle et al. (1992)

Zarco-Tejada and Miller (1999)

Lichtenhaler et al. (1996)

Hernandez-Clemente et al. (2012)

Penuelas et al. (1995)

Zarco-Tejada et al. (2003)

Elvidge and Penuelas (1994)

Levin et al. (2007)

Lobell et al. (2001)

Lobell et al. (2001)
SWIR VI \[ 37.72 \cdot (R_{2210} - R_{2090}) + \\
26.27 \cdot (R_{2280} - R_{2090}) + 0.57 \]

TCARI \[ 3 \cdot ((R_{700} - R_{670}) - 0.2 \cdot (R_{700} - R_{550}) \\
(R_{700}/R_{670})) \]

TCARI/OSAVI \[ TCARI/OSAVI \]

TCARI2 \[ 3 \cdot ((R_{750} - R_{705}) - 0.2 \cdot (R_{750} - R_{550}) \\
(R_{750}/R_{705})) \]

TCARI2/OSAVI2 \[ TCARI2/OSAVI2 \]

TGI \[ -0.5(190(R_{670} - R_{550}) - 120(R_{670} - R_{480})) \]

TVI \[ 0.5 \cdot (120 \cdot (R_{750} - R_{550}) - \\
200 \cdot (R_{670} - R_{550})) \]

Vogelmann \[ R_{740}/R_{720} \]

Vogelmann2 \[ (R_{734} - R_{747})/(R_{715} + R_{720}) \]

Vogelmann3 \[ D_{715}/D_{705} \]

Vogelmann4 \[ (R_{734} - R_{747})/(R_{715} + R_{720}) \]

Lobell et al. (2001)

Haboudane et al. (2002)

Haboudane et al. (2002)

Wu et al. (2008)

Wu et al. (2008)

Hunt et al. (2013)

Broge and Leblanc (2000)

Vogelmann et al. (1993)

Vogelmann et al. (1993)

Vogelmann et al. (1993)

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.
- Sxxx: Second derivation of reflectance values at wavelength 'xxx'. Note that S 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")
## Example calculating NDVI
```
data(spectral_data)
dvi <- vegindex(spectral_data, "NDVI")
```

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

## Self-defined indices
## NDVI
```
v1 <- vegindex(spectral_data, "(R800-R680)/(R800+R680)"")
## NDNI
v1 <- vegindex(spectral_data, "((log(1/R1510) - log(1/R1680))/(log(1/R1510) + log(1/R1680)))")
## D1
v1 <- vegindex(spectral_data, "D730/D706")
## Example using second derivative spectra
v1 <- vegindex(spectral_data, "(S930-S710)/(S930+S710)"")
```

### Description

Methods to access and set wavelength (band center) and full-width-half-max (fwhm) values for class `Speclib`.

### 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 'Speclib'
```
fwhm(object)
```

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

object  Object of class Speclib.
value   Numeric vector or data.frame containing wavelength values. Must always be in nm!

Details

Wavelength (band center) and full-width-half-max (fwhm) values are given for each spectral band. The wavelength is mandatory for creation of Speclib and is used within the whole functionality of the package (e.g., noiseFiltering, spectralResampling, vegindex, nri, plot.Speclib, mask).

Value

For wavelength<- and fwhm<-, the updated object. Otherwise a numeric vector of the wavelength and fwhm-values in nm is returned.

Author(s)

Lukas Lehnert

See Also

Speclib

Examples

data(spectral_data)

wavelength(spectral_data)
Index

*Topic `aplot`
  - clman, 25
  - plot.Nri, 67
  - plot.Specfeat, 69
  - plot.Speclib, 71
  - specfeat, 92

*Topic `classes`
  - Clman-class, 27
  - DistMat3D, 37
  - DistMat3D-class, 39
  - HyperSpecRaster-class, 54
  - Nri-class, 64
  - specfeat, 92
  - Specfeat-class, 93
  - speclib, 94
  - Speclib-class, 97

*Topic `datasets`
  - cancer_spectra, 13
  - spectral_data, 106

*Topic `documentation`
  - hsdardocs, 50

*Topic `methods`
  - Boruta::Boruta, 11
  - caret::createDataPartition-methods, 14
  - caret::createFolds-methods, 14
  - caret::createResample-methods, 14
  - caret::featurePlot-methods, 15
  - caret::gafs, 15
  - caret::preProcess-methods, 16
  - caret::rfe, 17
  - caret::safs, 18
  - caret::sbf, 19
  - caret::setPredictor, 21
  - caret::setResponse, 22
  - caret::train-methods, 24
  - HyperSpecRaster, 52
  - Raster-methods, 80

*Topic `multivariate`
  - feature_properties, 41
  - getNRI, 47
  - glm.nri, 49
  - import_USGS, 55
  - nri, 63
  - nri_best_performance, 66
  - rededge, 84
  - soilindex, 90
  - spectralResampling, 103
  - sr, 106
  - transformSpeclib, 111
  - unmix, 113
  - vegindex, 118

*Topic `package`
  - hsdar-package, 3

*Topic `smooth`
  - meanfilter, 59
  - noiseFiltering, 61

*Topic `spatial`
  - HyperSpecRaster, 52
  - HyperSpecRaster-class, 54
  - Raster-methods, 80

*Topic `utilities`
  - addcp, 5
  - apply.DistMat3D, 6
  - apply.Speclib, 8
  - bandnames, 9
  - bdri, 10
  - checkhull, 24
  - cubePlot, 29
  - deletecp, 32
  - derivative.speclib, 33
  - dim.speclib, 35
  - Extract Speclib by index, 40
  - get.gaussian.response, 43
  - get.sensor.characteristics, 44
  - gettcp, 46
  - hsdar_parallel, 51
  - hsdardocs, 50
idSpeclib, 54
makehull, 56
mask, 58
merge, 60
postprocessASD, 72
predictHyperspec, 73
read.ASD, 83
SI, 86
spectra, 101
spectralInterpolation, 102
subset.nri, 107
subset.speclib, 109
updatecl, 115
usagehistory, 117
wavelength, 123
(g)lm.nri, 65
<,ANY,DistMat3D-method (distMat3D), 37
<,DistMat3D,ANY-method (distMat3D), 37
<=,ANY,DistMat3D-method (distMat3D), 37
<=,DistMat3D,ANY-method (distMat3D), 37
<=,DistMat3D,DistMat3D-method (distMat3D), 37
==,ANY,DistMat3D-method (distMat3D), 37
==,DistMat3D,ANY-method (distMat3D), 37
==,DistMat3D,DistMat3D-method (distMat3D), 37
>,ANY,DistMat3D-method (distMat3D), 37
>,DistMat3D,ANY-method (distMat3D), 37
>,DistMat3D,DistMat3D-method (distMat3D), 37
>=,ANY,DistMat3D-method (distMat3D), 37
>=,DistMat3D,ANY-method (distMat3D), 37
>=,DistMat3D,DistMat3D-method (distMat3D), 37
[,Spectra,ANY,ANY,ANY-method (spectra), 101
[,DistMat3D,ANY,ANY,ANY-method (distMat3D), 37
[,DistMat3D,ANY,ANY-method (distMat3D), 37
[,DistMat3D-method (distMat3D), 37
[,Nri,ANY,ANY,ANY-method (Nri-methods), 65
[,Nri,ANY,ANY-method (Nri-methods), 65
[,SpecFeat,ANY,ANY,ANY-method (specfeat), 92
[,Speclib,ANY,ANY,ANY-method (Extract Speclib by index), 40
[,Speclib,ANY,ANY-method (Extract Speclib by index), 40
[,Speclib-method (Extract Speclib by index), 40
<-,DistMat3D,ANY,ANY,ANY-method (distMat3D), 37
<-,DistMat3D,ANY,ANY-method (distMat3D), 37
<-,DistMat3D-method (distMat3D), 37
$,Nri-method (Nri-methods), 65
$,Speclib-method (speclib), 94
addcp, 5, 5, 24, 25, 28, 32, 33, 46, 56, 57, 112, 113, 116
agrep, 108, 109
apply, 3, 7, 8, 39
apply,DistMat3D-method (apply.DistMat3D), 6
apply,Speclib-method (apply.Speclib), 8
apply.DistMat3D, 6, 40
apply.Speclib, 8
as.array,DistMat3D-method (distMat3D), 37
as.data.frame,Nri-method (Nri-methods), 65
as.matrix,DistMat3D-method (distMat3D), 37
as.matrix,Nri-method (Nri-methods), 65
bandnames, 9, 102
bandnames<- (bandnames), 9
bdri, 10
blockSize,Speclib-method (speclib_raster-methods), 99
Boruta, 12
Boruta,Nri-method (Boruta::Boruta), 11
Boruta,Specfeat-method (Boruta::Boruta), 11
Boruta,Speclib-method (Boruta::Boruta), 11
Boruta-methods (Boruta::Boruta), 11
Boruta::Boruta, 11
brick, 53, 54, 96, 98
brick,Speclib,ANY-method (HyperSpecRaster), 52
INDEX

brick, Speclib-method (Raster-methods), 80
cancer_spectra, 13
caret::createDataPartition-methods, 14
caret::createFolds-methods, 14
caret::createResample-methods, 14
caret::featurePlot-methods, 15
caret::gafs, 15
caret::preProcess-methods, 16
caret::rfe, 17
caret::safs, 18
caret::sbf, 19
caret::setPredictor, 21
caret::setResponse, 22
caret::showCaretParameters, 23
caret::train-methods, 24
cellFromCol, Speclib-method (spectra), 101
cellFromLine, Speclib-method (spectra), 101
cellFromPolygon, Speclib-method (spectra), 101
cellFromRow, Speclib-method (spectra), 101
cellFromRowCol, Speclib-method (spectra), 101
cellFromRowColCombine, Speclib-method (spectra), 101
cellFromXY, Speclib-method (spectra), 101
checkhull, 5
class::Clman-class, 27
class::clman, 25
class::Clnam-class, 27
colFromX, Speclib-method (spectra), 101
colorRamp, 68
cor.test, 28, 28, 68
cor.test,Nri-method (cor.test), 28
cor.test.Nri (cor.test), 28
createDataPartition, 14
createDataPartition,.CaretHyperspectral-method
  (caret::createDataPartition-methods), 14
createDataPartition,.CaretHyperspectral-method
  (caret::createFolds-methods), 14
createFolds, 14
caret::createFolds-methods, 14
caret::createFolds,ANY-method
  (caret::createFolds-methods), 14
caret::createFolds-methods
  (caret::createFolds-methods), 14
caret::createMultiFolds, 14
caret::createMultiFolds,.CaretHyperspectral-method
  (caret::createFolds-methods), 14
caret::createMultiFolds-methods
  (caret::createFolds-methods), 14
createMultiFolds,ANY-method
  (caret::createMultiFolds-methods), 14
caret::createMultiFolds-methods
  (caret::createFolds-methods), 14
createResample, 14
caret::createResample-methods, 14
caret::createResample,.CaretHyperspectral-method
  (caret::createResample-methods), 14
caret::createResample-methods
  (caret::createResample-methods), 14
createResample,ANY-method
  (caret::createResample-methods), 14
caret::createResample-methods
  (caret::createResample-methods), 14
cubePlot, 29
cut_specfeat, 4, 31, 31, 42, 92, 93
deletecp, 5, 24, 25, 27, 28, 32, 32, 46, 56, 57,
  112, 113, 116
derivative.speclib, 4, 33, 85, 123
dim, 97, 99
dim, DistMat3D-method (distMat3D), 37
dim,Nri-method (Nri-methods), 65
dim,Speclib-method (dim.speclib), 35
dim.speclib, 35
dist, 36, 37
dist.speclib, 27, 36
DistMat3D, 6, 7, 39, 65
distMat3D, ANY-method (distMat3D), 37
distMat3D, array-method (distMat3D), 37
distMat3D, matrix-method (distMat3D), 37
distMat3D, numeric-method (distMat3D), 37
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistMat3D-class</td>
<td>39</td>
</tr>
<tr>
<td>Extract Speclib by index</td>
<td>40</td>
</tr>
<tr>
<td>extract,Speclib-method</td>
<td>80</td>
</tr>
<tr>
<td>feature_properties, 10, 31, 41, 42, 92, 93</td>
<td></td>
</tr>
<tr>
<td>featurePlot, 15</td>
<td></td>
</tr>
<tr>
<td>featurePlot,.CareyHyperspectral-method (caret::featurePlot-methods)</td>
<td>15</td>
</tr>
<tr>
<td>featurePlot,ANY-method (caret::featurePlot-methods)</td>
<td>15</td>
</tr>
<tr>
<td>featurePlot-methods (caret::featurePlot-methods)</td>
<td>15</td>
</tr>
<tr>
<td>fourCellsFromXY,Speclib-method (spectra)</td>
<td>101</td>
</tr>
<tr>
<td>fwhm (wavelength)</td>
<td>123</td>
</tr>
<tr>
<td>fwhm,Speclib-method (wavelength)</td>
<td>123</td>
</tr>
<tr>
<td>fwhm&lt;-,Speclib,numeric-method (wavelength)</td>
<td>123</td>
</tr>
<tr>
<td>gafs, 12, 15, 16</td>
<td></td>
</tr>
<tr>
<td>gafs,Nri-method (caret::gafs)</td>
<td>15</td>
</tr>
<tr>
<td>gafs,Specfeat-method (caret::gafs)</td>
<td>15</td>
</tr>
<tr>
<td>gafs-methods (caret::gafs)</td>
<td>15</td>
</tr>
<tr>
<td>get.gaussian.response, 43, 105</td>
<td></td>
</tr>
<tr>
<td>get.sensor.characteristics, 43, 44, 104, 105</td>
<td></td>
</tr>
<tr>
<td>get_Boruta (Boruta::Boruta), 11</td>
<td></td>
</tr>
<tr>
<td>get_gafs (caret::gafs), 15</td>
<td></td>
</tr>
<tr>
<td>get_landsat4_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_landsat5_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_landsat7_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_landsat8_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_quickbird_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_reflectance, 48, 90, 91, 118, 123</td>
<td></td>
</tr>
<tr>
<td>get_rfe (caret::rfe), 17</td>
<td></td>
</tr>
<tr>
<td>get_safs (caret::safs), 18</td>
<td></td>
</tr>
<tr>
<td>get_sbf (caret::sbf), 19</td>
<td></td>
</tr>
<tr>
<td>get_sentinel2_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_wv2_4_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>get_wv2_8_response (spectralResampling), 103</td>
<td></td>
</tr>
<tr>
<td>getcp, 5, 24, 32, 33, 46, 56, 116</td>
<td></td>
</tr>
<tr>
<td>getFiniteNri (Nri-methods), 65</td>
<td></td>
</tr>
<tr>
<td>getNRI, 28, 47, 49, 50</td>
<td></td>
</tr>
<tr>
<td>getValuesBlock,HyperSpecRaster (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>getValuesBlock,HyperSpecRaster-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>getValuesBlock,Speclib-method (speclib_raster-methods), 99</td>
<td></td>
</tr>
<tr>
<td>glm, 49, 50, 64, 66–68, 70, 107</td>
<td></td>
</tr>
<tr>
<td>glm.nri, 28, 49, 64, 66–68, 70, 107</td>
<td></td>
</tr>
<tr>
<td>hcl, 68</td>
<td></td>
</tr>
<tr>
<td>hsdar (hsdar-package), 3</td>
<td></td>
</tr>
<tr>
<td>hsdar-package, 3</td>
<td></td>
</tr>
<tr>
<td>hsdar_parallel, 4, 51, 112</td>
<td></td>
</tr>
<tr>
<td>hsdardocs, 50, 104</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster, 52, 82, 114</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,character,numeric-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,HyperSpecRaster-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,RasterBrick,numeric-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,RasterLayer,numeric-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,Speclib,ANY-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster,Speclib-method (HyperSpecRaster), 52</td>
<td></td>
</tr>
<tr>
<td>HyperSpecRaster-class, 54</td>
<td></td>
</tr>
<tr>
<td>idSpeclib, 5, 41, 54, 97, 99, 102</td>
<td></td>
</tr>
<tr>
<td>idSpeclib&lt;- (idSpeclib), 54</td>
<td></td>
</tr>
<tr>
<td>import_USGS, 55</td>
<td></td>
</tr>
<tr>
<td>initialize,.SI-method (SI), 86</td>
<td></td>
</tr>
<tr>
<td>initialize,Clman-method (clman), 25</td>
<td></td>
</tr>
<tr>
<td>initialize,Speclib-method (speclib), 94</td>
<td></td>
</tr>
<tr>
<td>interpolate.mask (mask), 58</td>
<td></td>
</tr>
<tr>
<td>is.speclib (speclib), 94</td>
<td></td>
</tr>
<tr>
<td>Landsat_4_response (spectralResampling), 103</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Landsat_5_response (spectralResampling), 103
Landsat_7_response (spectralResampling), 103
Landsat_8_response (spectralResampling), 103
legendSpecLib (plot.SpecLib), 71
lines, 26
list.available.sensors (get.sensor.characteristics), 44
lm, 49, 50
lm.nri, 28
lm.nri (glm.nri), 49
lowess, 62
makehull, 5, 24, 25, 32, 33, 56, 56, 57, 116
mark.nri_best_performance (nri_best_performance), 66
mask, 58, 97, 99, 109, 124
mask, SpecLib-method (mask), 58
mask<- (mask), 58
mask<-, SpecLib.data.frame-method (mask), 58
mask<-, SpecLib.list-method (mask), 58
mask<-, SpecLib.matrix-method (mask), 58
mask<-, SpecLib.numeric-method (mask), 58
maskSpecLib (mask), 58
match.fun, 6–8
meanfilter, 59, 62
merge, 60
merge, SpecLib.SpecLib-method (merge), 60
n_features (specfeat), 92
n_features, SpecFeat-method (specfeat), 92
names, SI-method (SI), 86
nbands (dim.speclib), 35
ncol, SI-method (SI), 86
ncol, Spectra-method (speclib), 94
ncol, DistMat3D-method (DistMat3D), 37
noiseFiltering, 4, 60, 61, 85, 112, 124
Nri, 28, 39, 47, 64, 87, 107, 111
nri, 4, 39, 47, 63, 66, 68, 70, 107, 124
Nri-class, 64
Nri-methods, 65
nri_best_performance, 47, 49, 66
nrow, SI-method (SI), 86
nrow, Spectra-method (speclib), 94
nrow, DistMat3D-method (DistMat3D), 37
nspectra (dim.speclib), 35
optim, 78, 79
plot, 3, 27, 28, 49, 50, 97, 99
plot, Clman ANY-method (clman), 25
plot, Clman-method (clman), 25
plot, Nri ANY-method (plot.Nri), 67
plot, Nri-method (plot.Nri), 67
plot, SpecFeat ANY-method (plot.SpecFeat), 69
plot, SpecFeat-method (plot.SpecFeat), 69
plot, SpecLib-method (plot.SpecLib), 71
plot, SpecLib ANY-method (plot.SpecLib), 71
plot, SpecLib-method (plot.SpecLib), 71
plot.Nri, 67
plot.SpecFeat, 69, 92, 93
plot.SpecLib, 71, 124
plotRGB, 29
plotRGB, SpecLib-method (Raster-methods), 80
points, 26
polygon, 66
postprocessASD, 72
predict.train, 73
predictHyperspec, 73
predictHyperspec, train:. CaretHyperspectral function-method
(predictHyperspec), 73
predictHyperspec, train:. CaretHyperspectral missing-method
(predictHyperspec), 73
preProcess, 16
preProcess,. CaretHyperspectral-method
(caret::preProcess-methods), 16
preProcess, ANY-method
(caret::preProcess-methods), 16
preProcess-class
(caret::preProcess-methods), 16
preProcess-methods
(caret::preProcess-methods), 16
print,. Spectra-method (spectra), 101
print, Nri-method (Nri-methods), 65
print, SpecLib-method (speclib), 94
print.default, 49
print.getNRI (getNRI), 47
PROSAIL, 4, 75, 79
PROSPECT, 4, 77, 77
PROSPECTInvert (PROSPECT), 77
Quickbird_response
  (spectralResampling), 103

RapidEye_response(spectralResampling), 103
Raster-methods, 80
rastermeta, 82, 95, 97
read.ASD, 83
read_header, 83
readAll,Speclib-method (spectra), 101
readGDAL, 97, 99
redege, 4, 84, 123
response_functions
  (spectralResampling), 103
rfe, 12, 17
rfe,Nri-method (caret::rfe), 17
rfe,Specfeat-method (caret::rfe), 17
rfe,Speclib-method (caret::rfe), 17
rfe-methods (caret::rfe), 17
rowFromY,Speclib-method (spectra), 101
safs, 19
safs,Nri-method (caret::safs), 18
safs,Specfeat-method (caret::safs), 18
safs,Speclib-method (caret::safs), 18
safs-methods (caret::safs), 18
sam (dist.speclib), 36
sam_distance (dist.speclib), 36
save, 81
sbf, 20, 23
sbf,Nri-method (caret::sbf), 19
sbf,Specfeat-method (caret::sbf), 19
sbf,Speclib-method (caret::sbf), 19
sbf-methods (caret::sbf), 19
Sentinel2A_response
  (spectralResampling), 103
Sentinel2B_response
  (spectralResampling), 103
setPredictor, 22
setPredictor (caret::setPredictor), 21
setPredictor, .CaretHyperspectral,character-method
  (caret::setPredictor), 21
setPredictor-methods
  (caret::setPredictor), 21
setResponse, 12, 15, 17, 18, 20, 21
setResponse (caret::setResponse), 22
setResponse, .CaretHyperspectral,character-method
  (caret::setResponse), 22
setResponse-methods
  (caret::setResponse), 22
golayfilt, 33, 34, 62
show,.preProcessHyperspectral-method
  (caret::preProcess-methods), 16
show,.Spectra-method (spectra), 101
show,DistMat3D-method (distMat3D), 37
show,HyperSpecRaster-method
  (HyperSpecRaster-class), 54
show,Nri-method (Nri-methods), 65
show,Speclib-method (speclib), 94
showCaretParameters, 21, 22
showCaretParameters
  (caret::showCaretParameters), 23
showCaretParameters,.CaretHyperspectral-method
  (caret::showCaretParameters), 23
SI,Nri,ANY,ANY-method (SI), 86
SI,Nri,ANY,missing-method (SI), 86
SI,Nri,missing,ANY-method (SI), 86
SI,Nri,missing,missing-method (SI), 86
SI,Nri-method (SI), 86
SI,Speclib,ANY,ANY-method (SI), 86
SI,Speclib,ANY,missing-method (SI), 86
SI,Speclib,missing,ANY-method (SI), 86
SI,Speclib,missing,missing-method (SI), 86
SI,Speclib-method (SI), 86
SI.speclib (SI), 86
SI<- (SI), 86
SI<-,Nri,ANY-method (SI), 86
SI<-,Nri,data.frame-method (SI), 86
SI<-,Nri,matrix-method (SI), 86
SI<-,Speclib,ANY-method (SI), 86
SI<-,Speclib,data.frame-method (SI), 86
SI<-,Speclib,matrix-method (SI), 86
smoothSpeclib (noiseFiltering), 61
specfeat, 31, 69, 70, 93
specfeat, 11, 31, 42, 92, 92, 93
Specfeat-class, 93
Speclib, 3, 8, 9, 27, 28, 30, 34, 35, 37, 41, 52, 54, 55, 59, 61, 64, 65, 71, 73, 77, 79,
82, 87, 89, 93, 96–98, 102, 104, 107, 109, 112, 116, 118, 124
INDEX

speclib, 26, 73, 83, 94
speclib, character, numeric-method
(speclib), 94
speclib, hyperSpec, ANY-method (speclib), 94
speclib, HyperSpecRaster, ANY-method
(speclib), 94
speclib, matrix, data.frame-method
(speclib), 94
speclib, matrix, matrix-method (speclib), 94
speclib, matrix, numeric-method
(speclib), 94
speclib, numeric, data.frame-method
(speclib), 94
speclib, numeric, matrix-method
(speclib), 94
speclib, numeric, numeric-method
(speclib), 94
speclib, RasterBrick, data.frame-method
(speclib), 94
speclib, RasterBrick, matrix-method
(speclib), 94
speclib, RasterBrick, numeric-method
(speclib), 94
speclib, SpatialGridDataFrame, data.frame-method
(speclib), 94
speclib, SpatialGridDataFrame, matrix-method
(speclib), 94
speclib, SpatialGridDataFrame, numeric-method
(speclib), 94
speclib, Speclib, numeric-method
(speclib), 94
Speclib-class, 97
speclib_raster-methods, 99
spectra, 48, 96–99, 101
spectra, Clman-method (clman), 25
spectra, Speclib-method (spectra), 101
spectra, Speclib (spectra), 101
spectra<-(spectra), 101
spectra<-, Clman, data.frame-method
(clman), 25
spectra<-, Clman, matrix-method (clman), 25
spectra<-, Clman, numeric-method (clman), 25
spectra<-, Speclib, data.frame-method
(spectra), 101
spectra<-, Speclib, matrix-method
(spectra), 101
spectra<-, Speclib, numeric-method
(spectra), 101
spectra<-, Speclib, RasterBrick-method
(spectra), 101
spectral.resampling
(spectralResampling), 103
spectral_data, 106
spectralInterpolation, 102
spectralResampling, 4, 43, 45, 91, 96, 98,
103, 103, 124
spline, 62
sr, 106
subset, 71, 87
subset, Nri-method (subset.nri), 107
subset, Speclib-method (subset.speclib),
109
subset.nri, 107
subset.speclib, 41, 109
t.test, 68, 110, 110, 111
t.test, Nri-method (t.test), 110
t.test.nri (t.test), 110
train, 24
train, CaretHyperspectral-method
(caret::train-methods), 24
train, ANY-method
(caret::train-methods), 24
train-methods (caret::train-methods), 24
train.formula, 24
transformSpeclib, 4, 5, 10, 11, 24–28,
31–33, 41, 42, 46, 56, 57, 69, 89, 92,
93, 95, 111, 116
unmix, 4, 113
updatecl, 5, 24, 25, 27, 28, 32, 33, 56, 57,
115, 116
usagehistory, 96–98, 117
usagehistory<-(usagehistory), 117
USGS_get_available_files (import_USGS),
55
USGS_retrieve_files (import_USGS), 55
vegindex, 4, 34, 85, 91, 118, 124
wavelength, 123
wavelength, HyperSpecRaster-method
(wavelength), 123
wavelength,Nri-method (Nri-methods), 65
wavelength,Speclib-method (wavelength), 123
wavelength<-(wavelength), 123
wavelength<-,HyperSpecRaster,numeric-method (wavelength), 123
wavelength<-,Speclib,data.frame-method (wavelength), 123
wavelength<-,Speclib,numeric-method (wavelength), 123
writeRaster,Speclib,character-method (Raster-methods), 80
writeStart,HyperSpecRaster,character-method (HyperSpecRaster), 52
writeStart,HyperSpecRaster,Speclib-method (HyperSpecRaster), 52
writeStart,Speclib,character-method (speclib_raster-methods), 99
writeStop,Speclib-method (speclib_raster-methods), 99
writeValues,HyperSpecRaster,Speclib-method (HyperSpecRaster), 52
writeValues,RasterBrick,Speclib-method (HyperSpecRaster), 52
writeValues,RasterLayer,Speclib-method (HyperSpecRaster), 52
writeValues,Speclib,matrix-method (speclib_raster-methods), 99
writeValues,Speclib,numeric-method (speclib_raster-methods), 99
writeValues,Speclib,Speclib-method (speclib_raster-methods), 99
WV_2_8_response (spectralResampling), 103