Package 'colorSpec'

May 17, 2016

Type Package

Title Color Calculations with Emphasis on Spectral Data

Version 0.5-3

Encoding UTF-8

Date 2016-05-16

Author Glenn Davis [aut, cre]

Maintainer Glenn Davis <gdavis@gluonics.com>

Description Calculate with spectral properties of light sources, materials, cameras, eyes, and scanners. Build complex systems from simpler parts using a spectral product algebra. For light sources, compute CCT and CRI. For object colors, compute optimal colors and Logvinenko coordinates. Work with the standard CIE illuminants and color matching functions, and read spectra from text files, including CGATS files. Sample text files, and 4 vignettes are included.

License GPL (>= 3)

LazyLoad yes

LazyData yes

Depends R (>= 2.15)

Imports MASS

Suggests rgl, knitr

NeedsCompilation no

Repository CRAN

VignetteBuilder knitr

BuildVignettes yes

Date/Publication 2016-05-17 01:05:13

R topics documented:

`colorSpec-package` *Package colorSpec - Color Calculations with Emphasis on Spectral Data*

Description

Package **colorSpec** is for working with color spectral data in R.

Details

Features:

1. a clear classification of the commonly seen spectra into 4 types - depending on the vector space to which they belong
2. flexible organization for the spectra in memory, using an S3 class - **colorSpec**
3. a product algebra for the **colorSpec** objects
4. uniform handling of biological eyes, electronic cameras, and general action spectra
5. a few advanced calculations, such as computing optimal colors (aka Macadam Limits)
6. built-in essential tables, such as the CIE illuminants and color matching functions
7. a package logging system with log levels taken from the popular **Log4J**
8. support for reading a few spectrum file types, including CGATS
9. bonus files containing some other interesting spectra
10. minimal dependencies on other R packages

Non-features:

1. there is no support for many common 3D color spaces, such as CIELAB, HSL, etc.. For these spaces see packages **colorspace** and **colorscience**.
2. there are few non-linear operations
3. there is little support for scientific units; for these see package **colorscience**
4. photons are parameterized by wavelength in nm; other wavelength units and common alternative metrics such as wavenumber and electronvolt are not available

Regarding the non-linear operations in 2, the only such operations are conversion of linear RGB to display RGB, conversion of absorbance to transmittance, and the reparameterized wavelength in `computeADL`. The electronic camera model is purely linear with no dark current offset or other deviations.

Many ideas are taken from packages **hyperSpec**, **hsdar**, **pavo**, and **zoo**.

Author(s)

Glenn Davis <gdavis@gluonics.com>

See Also

`colorSpec` for the S3 class provided by this package.

colorSpec-guide.pdf in the doc directory. Go to the main index page and look for **User guides, package vignettes and other documentation.**

ABC *Standard Illuminants A, B, and C (1931)*

Description

`A.1nm`	standard Illuminant A, 360 to 780 nm at 1 nm intervals
`B.5nm`	standard Illuminant B, 320 to 780 nm at 5 nm intervals
`C.5nm`	standard Illuminant C, 320 to 780 nm at 5 nm intervals

Format

Each is a **colorSpec** object organized as a vector, with `quantity` equal to `"power"`.

Details

All of these have been divided by 100, to make the values at 560nm near 1 instead of 100.

Source

http://www.cvrl.org

References

Günther Wyszecki and W. S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982.

A	Table I(3.3.4) pp. 754-758.
B	Table II(3.3.4) pp. 759.
C	Table II(3.3.4) pp. 759.

See Also

`D50` `D65`

Examples

```
summary(xyz1931.1nm)
white.point = product( D65.1nm, xyz1931.1nm, wave='auto' )
```

---

applyspec *apply a function to each spectrum in a colorSpec object*

Description

apply a function to each spectrum in a colorSpec object

Usage

```
## S3 method for class 'colorSpec'
applyspec( x, FUN, ... )
```

Arguments

x	a **colorSpec** object with N wavelengths
FUN	a function that takes an N-vector as argument and returns an N-vector
...	additional arguments passed to FUN

Details

applyspec simply calls apply with the right MARGIN.

Value

a **colorSpec** object with the same dimensions, wavelength, quantity, and organization. If FUN does not return an N-vector, it is an ERROR and applyspec returns NULL.

See Also

quantity, wavelength, linearize, organization

Examples

```
#  convert absorbance to transmittance
path = system.file( "extdata/stains/Hematoxylin.txt", package='colorSpec' )
x = readSpectra( path )
x = applyspec( x, function(y) {10^(-y)} ) # this is what linearize(x) does
```

bind *Combine colorSpec Objects*

Description

Take a sequence of **colorSpec** objects and combine their spectra - if they all have the same wavelength and quantity

Usage

```
## S3 method for class 'colorSpec'
bind( ... )
```

Arguments

...	**colorSpec** objects with the same wavelength and quantity

Details

The organization of the returned object is the most complex of those in the input list, where the order of complexity is:

'matrix' < 'df.col' < 'df.row'

If the selected organization is 'df.row' the extradata is combined using merge.

The metadata of the returned object is copied from the first object in the input list.

Value

bind returns a **colorSpec** object, or NULL in case of ERROR. If the bind is successful, the number of spectra in the output object is the sum of the number of spectra in the input objects.

See Also

wavelength, quantity, organization, extradata, metadata

calibrate *make a linear modification to a colorSpec responder*

Description

make a linear modification to a **colorSpec** responder so a specific single spectrum (the stimulus) creates a given specific response.
The options are complicated, but in all cases the returned object is multiply(x,mat) where mat is an internally calculated MxM matrix. Stated another way, the spectra in the output are linear combinations of spectra in the input x.
In case of ERROR, a message is logged and the original x is returned.

Usage

```
## S3 method for class 'colorSpec'
calibrate( x, stimulus=NULL, response=NULL, method=NULL )
```

Arguments

x	a **colorSpec** responder with M spectra. The type must be 'responsivity.light' or 'responsivity.material'. The wavelength sequence of x must be regular.
stimulus	a **colorSpec** object with a single spectrum, with type 'light' or 'material' to match x. The wavelength sequence of stimulus must be equal to that of x. If stimulus is NULL, then an appropriate default is chosen, see **Details**.
response	an M-vector, or a scalar which is replicated to length M. All entries in response must be positive. If response is NULL, then an appropriate default *may be* chosen, see **Details**.
method	an MxM *adaption matrix*. method can also be 'scaling' and it is then set to the MxM identity matrix, which scales each responsivity spectrum in x independently. If M=3, method can also be 'Bradford', 'Von Kries', or 'MCAT02', and it is then set to the popular corresponding *chromatic adaption matrix*. For these special matrices, the spectra in x are **not** scaled independently; there is "cross-talk". If method is NULL, then an appropriate default is chosen, see **Details**.

Details

If stimulus is NULL, it is set to illuminantE or neutralMaterial to match x.

If response is NULL and the response of x is electrical or action, then response is set to an M-vector of all 1s. If response is NULL and the response of x is neural, then this is an ERROR and the user is prompted to supply a specific response.

If method is NULL and M=3 and the response of x is neural, then method is set to the 3x3 Bradford matrix.
Otherwise method is set to the MxM identity matrix, which scales each responsivity spectrum in x independently.

Value

a **colorSpec** object equal to multiply(x,mat) where mat is an internally calculated MxM matrix. The quantity and wavelength are preserved.
Note that mat is not the same as the the MxM *adaption matrix*. To inspect mat execute summary on the returned object. If method is 'scaling' then mat is diagonal and the diagonal entries are the M gain factors needed to achieve the calibration.

Note

Chromatic adaption transforms, such as 'Bradford', do not belong in the realm of spectra. For more about this subject see the explanation in *Digital Color Management*. This transform option is provided in calibrate because it is possible and convenient.

References

Edward J. Giorgianni and Thomas E. Madden. **Digital Color Management: Encoding Solutions.** 2nd Edition John Wiley. 2009. Chapter 15 - Myths and Misconceptions.

See Also

is.regular, quantity, wavelength, colorSpec, summary, illuminantE, neutralMaterial, product

Examples

```
# make an art gallery illuminated by illuminant A
gallery = product( A.1nm, 'artwork', xyz1931.1nm, wave='auto')
# chromatically adapt the output XYZs to D50 white point, using Bradford matrix
gallery.D50 = calibrate( gallery, response=officialXYZ('D50') )
gallery.D50

# make a flatbead scanner from illuminant F11 and a Flea2 camera
scanner = product( subset(Fs.5nm,'F11'), 'paper', Flea2.RGB, wave='auto')
# adjust RGB gain factors so the perfect reflecting diffuser yields RGB=(1,1,1)
scanner = calibrate( scanner )
scanner

# same flatbead scanner, but this time with some "white headroom"
scanner = product( subset(Fs.5nm,'F11'), 'paper', Flea2.RGB, wave='auto' )
scanner = calibrate( scanner, response=0.95 )
scanner
```

---

chop *chop spectra into low and high parts*

Description

chop all spectra in a **colorSpec** object into low and high parts at a blending interval

Usage

```
## S3 method for class 'colorSpec'
chop( x, interval, adj=0.5 )
```

Arguments

x	a **colorSpec** object
interval	a numeric vector with length 2 giving the endpoints of the interval, in nm
adj	a number in [0,1] defining weights of low and high parts over the interval

Details

For each spectrum, the low and high parts sum to the original spectrum. The low part vanishes on the right of the interval, and the high part vanishes on the left.

Value

`chop(x)` returns a **colorSpec** object with twice the number of spectra in `x` and with `organization` equal to `'matrix'`. The names of the new spectra are formed by appending ".lo" and ".hi" to the original spectrum names.

See Also

`organization`,

Examples

```
# chop blue butane flame into diatomic carbon and hydrocarbon parts
path = system.file( "extdata/sources/BlueFlame.txt", package="colorSpec" )
blueflame = readSpectra( path, seq(375,650,0.5) )
plot( chop( blueflame, interval=c(432,435), adj=0.8 ) )

# chop 'white' LED into blue and yellow parts
path = system.file( "extdata/sources/Gepe-G-2001-LED.sp", package="colorSpec" )
LED = readSpectra( path )
plot( chop( LED, c(470,495) ) )
```

---

colorSpec *constructing and testing colorSpec Objects*

Description

The function `colorSpec` is used to construct **colorSpec** objects.

`is.colorSpec` tests whether an object is a valid **colorSpec** object.

Usage

```
colorSpec( data, wavelength, quantity='auto', organization='auto' )

is.colorSpec(x)
```

Arguments

`data`	a vector or matrix of the spectrum values. In case `data` is a vector, there is a single spectrum and the number of points in that spectrum is the length of the vector. In case `data` is a matrix, the spectra are stored in the columns, so the number of points in each spectrum is the number of rows. It is OK for the matrix to have only 0 or 1 column. The column names (if any) are taken as the spectrum names. If no names are given, or if there are duplicate names, then

'S1', 'S2', ... are used. Names can also be assigned after construction too; see specnames. Row names are ignored. Compare this function with ts.

wavelength — a numeric vector of wavelengths for all the spectra. The length of this vector must be equal to NROW(data). The sequence must be increasing.

quantity — a character string giving the quantity of all spectra; see quantity for a list of possible values. In case of 'auto', a guess is made from the column names. This guess can be overridden later.

organization — a character string giving the desired organization of the returned **colorSpec** object. In case of 'auto', the organization is 'vector' or 'matrix' depending on data. Other possible organizations are 'df.col' or 'df.row'. The organization can be changed later, see organization for discussion of all 4 possible organizations.

x — an R object to test for validity.

Details

A **colorSpec** object is either a vector, matrix, or data.frame. It is of S3 class 'colorSpec' with these extra attributes:

wavelength a numeric vector of wavelengths for all the spectra. If the organization of the object is df.col, then this is absent.

quantity a character string that gives the physical quantity of all spectra, see quantity for a list of possible values.

metadata a list for user-defined data. The names 'path', 'header' and 'date' are already reserved; see metadata.

step.wl step between adjacent wavelengths in nm. This is assigned only when the wavelengths are regular; see is.regular.

specname only assigned when the organization is 'vector', in which case it is equal to the single character string name of the single spectrum. See specnames.

sequence only assigned when the object was returned from product. It is a list of the **colorSpec** terms in this product.

calibration only assigned when the object was returned from calibrate.

Value

colorSpec returns a **colorSpec** object, or NULL in case of ERROR.

is.colorSpec returns TRUE or FALSE. If FALSE it logs helpful reasons that x is invalid.

See Also

wavelength, quantity, metadata, step.wl, specnames, is.regular, coredata

Examples

```
#  make a synthetic Gaussian bandpass filter

center = 600
wave   = 400:700
trans  = exp( -(wave-center)^2 / 20^2 )

filter.bp   = colorSpec( trans, wave, 'transmittance' )

organization( filter.bp )  # returns: [1] "vector"

specnames( filter.bp ) = "myfilter"

# and now plot it
plot( filter.bp )
```

---

computeADL *compute ADL coordinates by ray tracing*

Description

Consider a **colorSpec** object x with type equal to responsivity.material. The set of all possible material reflectance functions (or transmittance functions) is convex, closed, and bounded (in any reasonable function space), and this implies that the set of all possible output responses from x is also convex, closed, and bounded. The latter set is called the *object-color solid* or *Rösch Farbkörper* for x. A color on the boundary of the *object-color solid* is called an *optimal color*. The special points **W** (the response to the perfect reflecting diffuser) and **0** are on the boundary of this set. The interior of the line segment of neutrals joining **0** to **W** is in the interior of the *object-color solid*. It is natural to parameterize this segment from 0 to 1 (from **0** to **W**). The solid is symmetrical about the neutral gray midpoint **G=W/2**.

Now suppose that x has 3 spectra (3 responses) and consider a color response **R** not equal to **G**. There is a ray based at **G** and passing through **R** that intersects the boundary of the *object-color solid* at an *optimal color* **B** on the boundary with Logvinenko coordinates (δ, ω). If these 2 coordinates are combined with α where **R** = $(1-\alpha)$**G** + α**B**, it yields the Logvinenko coordinates (α, δ, ω) of **R**. These coordinates are also denoted by ADL; see **References**. A response is in the *object-color solid* iff $\alpha <= 1$. A response is *optimal* iff $\alpha = 1$.

The coordinates of **0** are (α, δ, ω)=(1,0,0). The coordinates of **W** are (α, δ, ω)=(1,1,0). The coordinates of **G** are undefined.

Usage

```
## S3 method for class 'colorSpec'
computeADL( x, response )
```

Arguments

x	a **colorSpec** object with type equal to responsivity.material and 3 spectra
response	a vector of 3 numbers, or a matrix with 3 columns, that define 1 or more responses. A vector of 3 numbers is changed to a matrix with 1 row.

Details

For each response, a ray is computed and the ray tracing is done by probeOptimalColors.

Value

computeADL returns a data.frame with a row for each response. The columns in the data frame are:

response	the input response vector
ADL	the computed ADL coordinates of the response vector
omega	the reparameterized λ in the interval [0,1]; see **References**
lambda	lambda.1 and lambda.2 at the 2 transitions, in nm. lambda.1 < lambda.2 => bandpass, and lambda.1 > lambda.2 => bandstop.

If an individual ray could not be traced, the row contains NA in appropriate columns.
In case of global error, the function returns NULL.

Known Issues

The optimal color boundary is not differentiable at **0** and **W**. There may be numerical iteration failures if the response is near the neutral axis.

References

Logvinenko, A. D. An object-color space. Journal of Vision. 9(11):5, 1-23, (2009). http://journalofvision.org/9/11/5/. doi:10.1167/9.11.5.

Godau, Christoph and Brian Funt. XYZ to ADL: Calculating Logvinenko's Object Color Coordinates. Proceedings Eighteenth IS&T Color Imaging Conference. San Antonio. Nov 2009.

See Also

type, probeOptimalColors, vignette **optimals**

Examples

```
D50.eye = product( D50.5nm, 'varmat', xyz1931.1nm, wave='auto' )
computeADL( D50.eye, c(30,50,70) )
# response.x response.y response.z   ADL.alpha   ADL.delta  ADL.lambda    omega lambda.1 lambda.2
#       30       50       70   0.7364348   0.5384243 473.3909184 0.3008561 427.1431 555.5176
#since alpha < 1, this response is *inside* the object-color solid
```

computeCCT *Compute Correlated Color Temperature (CCT) of Light Spectra*

Description

Compute the CCT in Kelvin degrees of a **colorSpec** object with type equal to 'light'

Usage

```
## S3 method for class 'colorSpec'
computeCCT( x )

CCTfromXYZ( XYZ )
```

Arguments

x	an **colorSpec** R object with type equal to 'light', and M spectra
XYZ	a **colorSpec** 3-vector with XYZ tristimuls values

Details

In computeCCT, for each spectrum, XYZ is computed using xyz1931.1nm, and the result passed to CCTfromXYZ. If the quantity of x is 'photons' (actinometric) each spectrum is converted to 'power' (radiometric) on the fly.
In CCTfromXYZ, the CCT is computed using Robertson's Method, which can compute CCTs in the interval [1666.7K,Inf], see **References**.

Value

computeCCT returns a numeric vector of length M, where M is the number of spectra in x. The vector's names is set to specnames(x).
If the type of x is not 'light', then a warning is issued and all values are NA.
CCTfromXYZ returns a single number, but if the chromaticity of XYZ is too far from the daylight locus, then it returns NA.

Note

The table in the **References** contains an error at 325 mired, which was corrected in **Source**.

Source

http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_T.html

References

Günther Wyszecki and W. S. Stiles. Color Science: Concepts and Methods, Quantitative Data and Formulae, Second Edition. John Wiley & Sons, 1982. Table 1(3.11). pp. 227-228.

See Also

type, quantity, xyz1931, specnames

Examples

```
computeCCT( D65.1nm )       # returns 6502.068
computeCCT( A.1nm )         # returns 2855.656
CCTfromXYZ( c(1,1,1) )      # returns 5454.028
```

---

computeCRI *Compute Color Rendering Index (CRI) of Light Spectra*

Description

Compute the CIE 1974 color rendering index (CRI) of a light spectrum, called the *the test illuminant*.
From the given spectrum a *reference illuminant* is selected with the same CCT (Correlated Color Temperature). A selected set of 8 color samples is rendered in XYZ (1931) with both illuminants and 8 color differences are computed in a special **CIEUVW** color space. For each color difference a CRI is computed, where 100 is a perfect color match. The final CRI is the average of these 8 CRI values.

Usage

```
## S3 method for class 'colorSpec'
computeCRI( x, adapt=TRUE, attach=FALSE, tol=5.4e-3 )
```

Arguments

x	an **colorSpec** R object with type equal to 'light', and exactly 1 spectrum
adapt	if TRUE, then a special chromatic adaption is performed, see **Details**
attach	if TRUE, then a large list of intermediate calculations is attached to the returned number, as attribute data. This attached list includes data for all special 14 color samples, although the last 6 do not affect the returned CRI.
tol	for the CRI to be meaningful the chromaticities of the test and reference illuminants must be sufficiently close in the CIE 1960 chromaticity space. If the tolerance is exceeded, the function returns NA. The default tol=5.4e-3 is the one recommended by the CIE, but the argument allows the user to override it.

Details

If adapt is TRUE the 8 test uv points are chromatically adapted from the test illuminant to the reference illuminant using a special von Kries type transformation; see *Oleari* and *Wikipedia*. The test UVW values are computed relative to the reference illuminant.
If adapt is FALSE the 8 test uv points are *not* chromatically adapted, and the test UVW values are computed relative to the test illuminant.

Value

computeCCT returns a single number <= 100. In case of ERROR it returns NA. If attach is TRUE a large list is attached to the returned number.

Source

The test color reflectance spectra are taken from:
http://www.lrc.rpi.edu/programs/nlpip/lightinganswers/lightsources/scripts/NLPIP_LightSourceColor_Script.m

References

Oleari, Claudio, Gabriele Simone. Standard Colorimetry: Definitions, Algorithms and Software. John Wiley. 2016. pp. 465-470.

Günther Wyszecki and W. S. Stiles. Color Science: Concepts and Methods, Quantitative Data and Formulae, Second Edition. John Wiley & Sons, 1982. Table 1(3.11). p. 828.

Wikipedia. **Color rendering index**. http://en.wikipedia.org/wiki/Color_rendering_index

Hunt, R. W. G. and M. R. Pointer. Measuring Colour. 4th edition. John Wiley & Sons. 2011. Appendix 7.

See Also

type, xyz1931, computeCCT

Examples

```
computeCRI( subset(Fs.5nm,'F2') )       # returns 64.152
computeCRI( subset(Fs.5nm,'F4') )       # returns 51.352
```

---

convolvewith *Convolve each spectrum in a colorSpec object with a kernel*

Description

This function convolves each spectrum in a colorSpec object with a kernel of odd length. Its primary purpose is to correct raw spectrometer data (with positive instrumental bandwidth) to have bandwidth=0. Two popular correction kernels for this, with lengths 3 and 5, are built-in options, see **Details**.

Usage

```
## S3 method for class 'colorSpec'
convolvewith( x, coeff )
```

Arguments

x	a **colorSpec** object with N wavelengths
coeff	a convolution kernel of odd length. The center entry of this vector is taken as index 0 in the convolution. coeff can also be the string 'SS3' which means to apply the Stearns&Stearns bandwidth correction kernel coeff=c(-1,14,-1)/12, see **Details**. coeff can also be the string 'SS5' which means to apply the Stearns&Stearns bandwidth correction kernel coeff=c(1,-12,120,-12,1)/98, see **Details**.

Details

The built-in kernels, 'SS3' and 'SS5', were derived by *Stearns & Stearns* under specific hypotheses on the spectrometer profile, bandpass, and pitch; see **References**.
Missing values at both ends are filled by copying from the nearest valid value.

Value

a **colorSpec** object with the same dimensions, wavelength, quantity, and organization. If coeff is invalid it is an ERROR and convolvewith returns NULL.

References

Stearns, E.I., Stearns R.E. An example of a method for correction radiance data for bandpass error. *Color Research and Application*. 13-4. 257-259. 1988.

Schanda, Janos. CIE Colorimetry, in *Colorimetry: Understanding the CIE System*. Wiley Interscience. 2007. p. 124.

Oleari, Claudio, Gabriele Simone. *Standard Colorimetry: Definitions, Algorithms and Software*. John Wiley. 2016. p. 309.

See Also

quantity, wavelength, linearize, applyspec, organization

---

coredata *Extract the Core Data of a colorSpec Object*

Description

functions for extracting the core data contained in a **colorSpec** object.

Usage

```
## S3 method for class 'colorSpec'
coredata( x, forcemat=FALSE )

## S3 method for class 'colorSpec'
as.matrix( x, ... )
```

Arguments

x	a **colorSpec** object
forcemat	if x has only 1 spectrum, return a matrix with 1 column instead of a vector
...	extra arguments ignored

Value

coredata	If x has organization equal to 'vector' then it returns x, unless forcemat is TRUE when it returns x as a matrix with 1 column. If x has any other organization then it returns a matrix with spectra in the columns. All of the **colorSpec** attributes are stripped except the column names, and the row names are set to as.character(wavelength(x)).
as.matrix	a wrapper for coredata with forcemat=TRUE

See Also

organization

---

cs.options *Functions to set and retrieve colorSpec package options*

Description

Allow the user to set and examine a variety of options for **colorSpec**.

Usage

```
cs.options( ... )
```

Arguments

...	named arguments are set; unnamed arguments are retrieved, and must be single character strings. See example.

Value

returns a list with all options mentioned.
If no arguments are given, then it returns a list of all options.

Options used in logging

Name	Default Value	Description
loglevel	'WARN'	controls the amount of logging information produced.
.	.	.
logformat	'%t %l %n::%f(). %m'	the format of each log line. See logging page.
.	.	.
stoponerror	TRUE	stop when a logging event has level ERROR; a logical.

When setting loglevel an initial letter is sufficient.

See Also

logging

Examples

```
#   set 2 options and retrieve the 3rd
cs.options( loglevel="DEBUG", stoponerror=FALSE, "logformat" )
```

D50 *Standard Illuminant D50 (1964)*

Description

D50.5nm standard Illuminant D50, from 300 to 830 nm at 5 nm intervals.

Format

A **colorSpec** object organized as a vector, with 107 data points and specnames equal to 'D50'.

Details

This spectrum is not read from a table from a CIE publication. It is computed using the function daylightSpectra by following the special CIE recipe given in the **References**. The temperature is set to (14388/14380) * 5000 = 5002.781 Kelvin. The coefficients of the daylight components S_0, S_1, and S_2 are rounded to 3 decimal places. This linear combination is computed at 10nm intervals and then linearly interpolated to 5nm intervals. The result is normalized to value 1 at 560nm (instead of the usual 100), and finally rounded to 5 decimal places. See **Examples**.

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. Table I(3.3.4) pp. 754-758

CIE 15: Technical Report: Colorimetry, 3rd edition. CIE 15:2004. Table T.1, pp 30-32, and Note 5 on page 69.

Schanda, Janos. CIE Colorimetry, in *Colorimetry: Understanding the CIE System*. Wiley Interscience. 2007. p. 42.

See Also

ABC , D65 , daylightSpectra

Examples

```
#   the CIE recipe for computing D50.5nm
correction  = 14388 / 14380     # note 5, page 69 in CIE 15:2004
D50.10nm   = daylightSpectra( correction*5000, wavelength=seq(300,830,by=10), roundMs=TRUE )
D50.5nm     = resample( D50.10nm, seq(300,830,by=5), method='linear' )
D50.5nm     = round( D50.5nm, 5 )

summary( D50.5nm )
white.point = product( D50.5nm, xyz1931.1nm, wave='auto' )
```

D65 *Standard Illuminant D65 (1964)*

Description

D65.1nm	standard Illuminant D65, 300 to 830 nm at 1 nm intervals
D65.5nm	standard Illuminant D65, 380 to 780 nm at 5 nm intervals

Format

Each is a **colorSpec** object organized as a vector, with specnames equal to 'D65'.

Details

Both of these have been divided by 100, to make the values at 560nm equal to 1 instead of 100.

Source

http://www.cvrl.org

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. Table I(3.3.4) pp. 754-758

ASTM E 308-01. Standard Practice for Computing the Colors of Objects by Using the CIE System. Table 3. pages 3-4.

See Also

ABC, D50, daylightSpectra , daylight

Examples

```
summary( D65.1nm )
white.point = product( D65.1nm, xyz1931.1nm, wave='auto' )
```

---

daylight *Standard Daylight Components*

Description

daylight1964	spectral components S_0, S_1, S_2; from 300 to 830 nm at 5 nm intervals
daylight2013	smoothed spectral components S_0, S_1, S_2; from 300 to 830 nm at 1 nm intervals

Format

Each is a **colorSpec** object organized as a matrix with 3 columns

S0	component 0, the mean power spectrum
S1	component 1, the 1st characteristic spectrum
S2	component 2, the 2nd characteristic spectrum

Source

http://www.cie.co.at/publ/abst/datatables15_2004/CIE_sel_colorimetric_tables.xls

http://vision.vein.hu/~schanda/CIE%20TC1-74/

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. Table V(3.3.4) p. 762.

Smoothing spectral power distribution of daylights. Zsolt Kosztyan and Janos Schanda. Color Research & Application. Volume 38, Issue 5, pages 316-321, October 2013.

CIE 15: Technical Report: Colorimetry, 3rd edition. CIE 15:2004. Table T.2, pp 33-34

JUDD, D.B., MACADAM, D.L. and WYSZECKI, G., with the collaboration of BUDDE, H.W, CONDIT, H.R, HENDERSON, S.T, and SIMONDS, J.L. Spectral distribution of typical daylight as a function of correlated color temperature. J Opt. Soc. Am. 54, 1031-1040, 1964.

Zsolt Kosztyan and Janos Schanda. Smoothing spectral power distribution of daylights. Color Research & Application. Volume 38, Issue 5, pages 316-321, October 2013.

See Also

D65, D50, daylightSpectra

Examples

```
summary( daylight1964 )
day1964 = daylightSpectra( c(5000,6500), comp=daylight1964 )
day2013 = daylightSpectra( c(5000,6500), comp=daylight2013 )

plot( day1964, col='black' )
plot( day2013, col='black', add=TRUE )
```

DisplayRGB *Compute Display RGB from Linear RGB*

Description

All RGB displays have a non-linear "gamma function" of some sort. This function converts from linear RGB to an RGB appropriate for the gamma function of the display.

Usage

```
DisplayRGBfromLinearRGB( RGB, gamma='sRGB' )
```

Arguments

RGB	linear RGB values organized as a vector or matrix of any size; all 3 channels are treated the same way so size does not matter
gamma	either the string 'sRGB' or a positive number giving the gamma of the display.

Value

The function first clamps the input RGB to the interval [0,1]. If gamma='sRGB' (not case-sensitive) it then maps [0,1] to [0,1] using the special piecewise-defined sRGB function, see *Wikipedia*. In case gamma is a positive number, the function raises all values to the power 1/gamma. The dimensions and names of the input are copied to the output.
In case of error, the function returns the clamped input values.

Source

Wikipedia. **sRGB**. http://en.wikipedia.org/wiki/sRGB

See Also

RGBfromXYZ

Examples

```
DisplayRGBfromLinearRGB( c(0.2, 0.5) )
# [1] 0.4845292 0.7353570     #  this is display sRGB, in [0,1]

DisplayRGBfromLinearRGB( c(-0.1, 0.2, 0.5, 1), 2.2 )
# [1] 0.0000000 0.4811565 0.7297401 1.0000000    #  gamma=2.2

x = seq( 0, 1, len=101)
plot( x, DisplayRGBfromLinearRGB(x), type='l' )
```

`extradata` *extradata of a colorSpec object*

Description

Retrieve or set the extradata of a **colorSpec** object.

Usage

```
## S3 method for class 'colorSpec'
extradata(x)
extradata(x) <- value
```

Arguments

`x`	a **colorSpec** R object
`value`	a `data.frame` with `m` rows, where `m` is the number of spectra in `x`.

Details

If the `organization` of `x` is not `'df.row'`, then `extradata` cannot be stored in `x` and the assignment is ignored, with a warning. First change the `organization` to `'df.row'`, and then assign the `extradata`.

If the `organization` of `x` is `"df.row"`, but `value` does not have the right number of rows, the assignment is also ignored.

Value

`extradata` returns a `data.frame` with `m` rows, where `m` is the number of spectra in `x`. The `rownames` are set to the `specnames` of `x`. If there is no extra data then the number of columns in this `data.frame` is 0.

Note

If the assigment of `value` is successful, any existing `extradata` in `x` is discarded.

See Also

colorSpec, organization

F96T12 *Photon Irradiance of F96T12 Fluorescent Bulb*

Description

`F96T12`
Sylvania F96T12 CW/VHO 215-Watt fluorescent bulb photon irradiance, measured with a LI-COR LI-1800 spectroradiometer, from 300 to 900 nm at 1 nm intervals.

Format

A **colorSpec** object organized as a vector, with 601 data points and specnames equal to `'F96T12'`.

Details

The unit is (μmole of photons)$*sec^{-1} * m^{-2} * nm^{-1}$.

Source

Pedro J. Aphalo. http://www.mv.helsinki.fi/aphalo/photobio/lamps.html

See Also

ABC , D65 , daylightSpectra

Examples

```
sum( F96T12 )
# [1] 320.1132  photon irradiance, (micromoles of photons)*m^{-2}

sum( radiometric(F96T12) )
# [1] 68.91819  irradiance, watts*m^{-2}
```

Flea2.RGB *Flea2 Camera FL2-14S3C from Point Grey*

Description

`Flea2.RGB` an RGB responder to light, from 360 to 800 nm at 10 nm intervals

Format

A **colorSpec** object with `quantity` equal to `'power->electrical'` and 3 spectra: `Red`, `Green`, and `Blue`.

Details

This data is read from the file **Flea2-spectral.txt** which was digitized from the plot in **Flea2-spectral.png**.

Source

https://www.ptgrey.com/flea2-14-mp-color-firewire-1394b-sony-icx267-camera

See Also

`quantity`, vignette blueflame

Examples

```
#  Make a scanner from a tungsten source and a Flea2 camera
Flea2.scanner = product( A.1nm, "VARMATERIAL", Flea2.RGB, wavelength=420:680 )
Flea2.scanner = calibrate( Flea2.scanner )
```

`Fluorescents` *Standard series F Illuminants F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, and F12*

Description

`Fs.5nm` contains 12 CIE Fluorescent Illuminants, from 380 to 780 nm, at 5nm intervals.

Format

`Fs.5nm` is a **colorSpec** object with 12 spectra. It is organized as a data frame with `quantity` equal to `"power"`.

Details

The series F illuminants do not seem to be normalized in a consistent way.

Source

http://www.cis.rit.edu/research/mcsl2/online/CIE/Fluorescents.htm

See Also

ABC, D50, D65

Examples

```
#   plot only F4
plot( subset(Fs.5nm,"F4") )
```

`HigherPasserines` *Cone Fundamentals for the Higher Passerines*

Description

`HigherPasserines` Tetrachromatic Cone Fundamentals of Higher Passerine Birds

Format

A **colorSpec** object organized as a matrix with the 4 variables.

`UV`	the UV wavelength responsivity
`Short`	the short wavelength responsivity
`Medium`	the medium wavelength responsivity
`Long`	the long wavelength responsivity

The `wavelength` is from 300 to 700 nm, at 1nm intervals.

Source

http://onlinelibrary.wiley.com/doi/10.1111/j.1095-8312.2005.00540.x/suppinfo

References

Endler & Mielke. Comparing entire colour patterns as birds see them. Biological Journal of the Linnean Society. Volume 86, Issue 4, pages 405-431, December 2005. Original Name of File: BIJ_540_Endler_Mielke_OnlineAppendix.txt.

See Also

lms2000

Examples

```
summary(HigherPasserines)
```

Hoya *standard Hoya filters*

Description

Hoya 4 standard Hoya filters; from 300 to 750 nm at 10nm intervals.

Format

A **colorSpec** object with quantity equal to 'transmittance' and 4 spectra:

R-60	long-pass red filter with cutoff about 600nm
G-533	band-pass green filter with peak about 533nm
B-440	band-pass blue filter with peak about 440nm
LB-120	Light-balancing Blue filter with mired shift equal to -120

Source

http://www.hoyaoptics.com/

See Also

quantity

Examples

```
#   compute response of ACES scanner to the Hoya filters
product( Hoya, scanner.ACES, wave='auto' )
```

---

lightResponsivitySpectra *compute standard light responsivity spectra*

Description

Some action spectra standards are defined by simple equations; the erythemal spectrum for human sunburn is one of them.

Usage

```
erythemalSpectrum( wavelength=250:400 )
```

Arguments

wavelength	a vector of wavelengths, in nm

Details

This erythemal spectrum is defined in 4 pieces: $\lambda <= 298$, $298 <= \lambda <= 328$, $328 <= \lambda <= 400$, and $400 < \lambda$. All the units are nm. The spectrum is used in the definition of the international standard **UV Index**.

Value

For `erythemalSpectrum()`
A **colorSpec** object with `quantity` equal to `'power->action'`. The responsivity is 0 for $\lambda > 400$ nm, so this putting this spectrum in the category of human vision is a bit of a stretch.

Source

http://en.wikipedia.org/wiki/Ultraviolet_index

References

McKinlay, A.F., and B.L. Diffey. A reference action spectrum for ultraviolet induced erythema in human skin. CIE Res. Note, 6(1), 17-22. (1987)

See Also

daylight, `quantity`, materialSpectra, lightSpectra

`LightSpectra` *compute standard light spectra*

Description

Two families of standard illuminants that depend on temperature are the Planckian spectra (black-body spectra), and daylight spectra. For the daylight spectra, a smoothed version is available. Illuminant E, a third and trivial spectrum, is also available.

Usage

```
planckSpectra( temperature, normalize=TRUE, wavelength=300:830 )

daylightSpectra( temperature, components=colorSpec::daylight1964,
                              wavelength=NULL, roundMs=FALSE )

illuminantE( power=1, wavelength=380:780 )
```

Arguments

`temperature`	a vector of temperatures, in Kelvin
`components`	a **colorSpec** object with the daylight components S_0, S_1, and S_2. The default is `daylight1964` and a smoothed version `daylight2013` is also available.
`wavelength`	a vector of wavelengths. For `planckSpectra` and `illuminantE` this is required. For `daylightSpectra` this is optional. The default `wavelength=NULL` means to use the wavelengths in `components`, and otherwise these wavelengths are resampled using `resample`
`normalize`	a logical value. If `TRUE` the Planck spectra are normalized to have value 1 at 560nm. If `FALSE` then the quantity returned is radiant exitance with unit $W * m^{-2} * nm^{-1}$.
`roundMs`	a logical value. The original CIE method for the daylight spectra requires rounding intermediate coefficients M1 and M2 to 3 decimal places. This rounding is necessary to reproduce the tabulated values in Table T.1 of the CIE publication in **References**.
`power`	a vector of power levels

Details

For `daylightSpectra` the valid range of temperatures is 4000 to 25000 K. For a temperature outside this range the spectrum is set to all `NA`s.

The equations for `daylightSpectra` and `planckSpectra` are complex and can be found in the **References**.

`IlluminantE` is trivial - all constant `power`.

Value

For `planckSpectra` and `daylightSpectra` :
A **colorSpec** object with `quantity` equal to `'power'`, and `organization` equal to `'matrix'` or `'vector'`. The `specnames` are `PNNNN` or `DNNNN` for `planckSpectra` and `daylightSpectra` respectively.
The number of spectra in the object is the number of temperatures = `length(temperature)`.

For `illuminantE` :
A **colorSpec** object with `quantity` equal to `'power'`.
The number of spectra in the object is the number of power levels = `length(power)`.

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. page 146.

CIE 15: Technical Report: Colorimetry, 3rd edition. CIE 15:2004. Table T.1, pp 30-32, and Note 5 on page 69.

Schanda, Janos. CIE Colorimetry, in *Colorimetry: Understanding the CIE System*. Wiley Interscience. 2007. p. 42.

See Also

daylight, `resample`, `organization`, `quantity`, materialSpectra

`linearize` *linearize a colorSpec object - to make it ready for colorimetric calculations*

Description

linearize spectra and return modified object

Usage

```
## S3 method for class 'colorSpec'
linearize( x )
```

Arguments

`x`	a **colorSpec** object

Details

If the `quantity` of `x` is not `'absorbance'` then `x` is returned unchanged.

If the `quantity` is `'absorbance'` then absorbance is converted to transmittance using

$$transmittance = 10^{-absorbance}$$

Surprisingly, there does not seem to be a similar logarithmic version of reflectance. Plots with log(responsivity) is somewhat common, but does not seem to have a separate name. I have not seen log(radiometric power).

Value

`linearize` returns a **colorSpec** object with linear quantities.

See Also

`quantity`

lms1971 *Cone Fundamentals - 2-degree (1971)*

Description

`lms1971.5nm` the Vos & Walraven (1971) 2° cone fundamentals from 380 to 780 nm, at 5nm intervals

Format

A **colorSpec** object organized as a matrix with 3 columns:

`long`	the long wavelength responsivity
`medium`	the medium wavelength responsivity
`short`	the short wavelength responsivity

Source

http://www.cvrl.org/database/text/cones/vw.htm

References

Vos, J. J. & Walraven, P. L. On the derivation of the foveal receptor primaries. Vision Research. 11 (1971) pp. 799-818.

See Also

`lms2000`

Examples

```
summary(lms1971.5nm)
white.point = product( D65.1nm, lms1971.5nm, wave='auto' )
```

lms2000 *Cone Fundamentals - 2-degree (2000)*

Description

`lms2000.1nm` the Stockman and Sharpe (2000) 2° cone fundamentals from 390 to 830 nm, at 1nm intervals

Format

A **colorSpec** object organized as a matrix with 3 columns:

`long`	the long wavelength responsivity
`medium`	the medium wavelength responsivity
`short`	the short wavelength responsivity

Source

http://www.cvrl.org/cones.htm

References

Stockman, A., Sharpe, L. T., & Fach, C. C. (1999). The spectral sensitivity of the human short-wavelength cones. Vision Research, 39, 2901-2927.

Stockman, A., & Sharpe, L. T. (2000). Spectral sensitivities of the middle- and long-wavelength sensitive cones derived from measurements in observers of known genotype. Vision Research, 40, 1711-1737.

See Also

`lms1971`

Examples

```
summary(lms2000.1nm)
white.point = product( D65.1nm, lms2000.1nm, wave='auto' )
```

`logging` *Logging in colorSpec package*

Description

There is some flexibility in the **colorSpec** logging level and format. There is no flexibility in the output, it all goes to the console using `cat` and `print`, but see `sink`.

Details

`loglevel`
The levels are: "FATAL", "ERROR", "WARN", "INFO", "DEBUG", and "TRACE" - the usual ones from **Log4J**. The initial level is "WARN".
A log event with level "ERROR" stops execution if the option `stoponerror` is `TRUE`.
A "FATAL" log event (e.g. internal error), always stops execution.

`logformat`
The format is given by a string with standard **Log4J** *conversion specifications*:

%t	the date/time of the logging event.	%t can be followed by standard strftime specs in braces; see example.
%l	the level of the logging event	
%n	namespace where event occurred	
%f	function where event occurred	
%m	the message itself	

See Also

cs.options, sink

Examples

```
cs.options( logformat="%t{%H:%M:%OS3} %l %n::%f(). %m" )
```

---

materialSpectra *compute standard material spectra*

Description

Compute hypothetical neutral gray material reflectance/transmittance, and absorbance of the human lens, as a function of age.

Usage

```
neutralMaterial( gray=1, wavelength=380:780 )
lensAbsorbance( age=32, wavelength=400:700 )
```

Arguments

gray	a vector of gray levels, in the interval [0,1]. gray=1 represents the *perfect reflecting diffuser*.
age	a vector of ages in years; all ages must be >= 20.
wavelength	a vector of wavelengths for the returned object

Value

neutralMaterial returns a **colorSpec** object with quantity equal to 'reflectance'. The reflectance of each spectrum is constant. The number of spectra in the object is the number of gray levels = length(gray).

lensAbsorbance returns a **colorSpec** object with quantity equal to 'absorbance'. The absorbance model for the human lens is taken from *Pokorny*. The number of spectra in the object is the number of ages = length(age).

References

Pokorny, Joel, Vivianne C. Smith, and Margaret Lutze. Aging of the Human Lens. Applied Optics. Vol. 26, No. 8. 15 April 1987. Table I. Page 1439.

See Also

lightSpectra, quantity

Examples

```
#   make a perfect reflecting diffuser (PRD)
prd = neutralMaterial( 1 )

#   make a perfect transmitting filter (PTF)
ptf = prd
quantity(ptf) = 'transmittance'

#   compare transmittance at 3 ages: 20, 32, and 80 years
plot( linearize(lensAbsorbance( c(20,32,80) )), col='black', lty=1:3 )
```

---

mean *calculate mean of multiple spectra*

Description

compute mean of all spectra in a **colorSpec** object

Usage

```
## S3 method for class 'colorSpec'
mean( x, ... )
```

Arguments

x	a colorSpec object
...	further arguments ignored

Details

This function might be useful when capturing many spectra on a spectrometer and averaging them to reduce noise.

Value

a **colorSpec** object with single spectrum = average of all spectra in **colorSpec**.

`metadata` *metadata of a colorSpec object*

Description

Retrieve or set the metadata of a **colorSpec** object.

Usage

```
## S3 method for class 'colorSpec'
metadata(x, ...)
metadata(x) <- value
```

Arguments

`x`	a **colorSpec** R object
`...`	optional names of metadata to return
`value`	a named `list` that is appended to the existing list of metadata. Unnamed items are ignored. If a name already exists, its value is updated.

Details

The metadata list is stored as `attr(x,'metadata')`. Initially this list is empty.
The assignment operator looks like it replaces the internal metadata list of `x`, but it actually appends to it.

Value

`metadata()` with no argument returns the complete named list of metadata. If arguments are present, then only those `metadata` items are returned.

See Also

`colorSpec`; `extradata`

Examples

```
## Not run:
# get list of *all* metadata
metadata(x)

# get just the file 'path'
metadata( x, 'path' )

# set the 'date'
metadata( x ) = list( date="2016-04-01" )
```

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

`multiply` *multiply a colorSpec object by scalar, vector, or matrix*

Description

multiply spectra by coefficients and return modified object

Usage

```
## S3 method for class 'colorSpec'
multiply( x, s )

## S3 method for class 'colorSpec'
normalize( x, norm='L1' )
```

Arguments

`x`	a **colorSpec** object with M spectra
`s`	a scalar, an M-vector, or an MxP matrix
`norm`	one of `'L1'`, `'L2'`, or `'Linf'`, specifying one of the standard vector norms $L_1, L_2, or L_{inf}$. `norm` can also be a numeric wavelength (e.g. 560 nm), and then the spectrum is scaled to have value 1 at this wavelength. Of course, this is not a true vector norm.

Details

For `multiply`:
If `s` is an MxP matrix, say **S**, and one thinks of the spectra as organized in an NxM matrix **X**, then the new spectra are defined by the matrix **XS**, which is NxP. If the P column names of `s` are set, then they are copied to the spectrum names of the output. Otherwise, default spectrum names are assigned as in `colorSpec`.
If `s` is an M-vector, then **S**=`diag(s)` is computed and used in the previous sentence. This has the effect of multiplying spectrum i by `s`[i].
If `s` is a scalar then every spectrum is multiplied by `s`.
The multiplication may produce negative entries, but no check is made for this.
WARNING: An M-vector and an Mx1 matrix may yield quite different results.

For `normalize`:
`normalize` calls `multiply` with `s` = an M-vector. If the norm of a spectrum is 0, then it is left unchanged.

Value

`multiply` returns a **colorSpec** object with the matrix of spectra of `x` multiplied by `s`.

`normalize` returns a **colorSpec** object with each spectrum of `x` scaled to have given norm equal to 1.

In both functions, the `quantity` and `wavelength` are preserved.

Note

If `x` is organized as a matrix, and `s` is a scalar, the one can use the simpler and equivalent `s*x`.

See Also

`product`, `quantity`, `wavelength`, `colorSpec`

---

officialXYZ *Query the Official XYZ values for Standard Illuminants*

Description

In careful calcuations, it is sometime helpful to have the 'official' values of XYZ, i.e. with the right number of decimal places.

Usage

```
officialXYZ( name )
```

Arguments

`name`	a subvector of `c('A','B','C','D50','D50.ICC','D55','D65','D75', 'E','F2','F7','F11')`, which are the names of some standard illuminants

Details

All XYZ values are taken from the ASTM publication in **References**, except `B` which is taken from *Wyszecki & Stiles* and `D50.ICC` which is taken from ICC publications. The latter is different than that of ASTM.

Value

An Mx3 matrix where M is the length of `name`. Each row filled with the official XYZ, but if the illuminant name is not recognized the row is all `NA`s. The matrix `rownames` are set to `name`, and `colnames` to `c('X','Y','Z')`.

Note

The input names are case-sensitive. The output XYZ is normalized so that `Y=1`.

References

ASTM E 308 - 01. Standard Practice for Computing the Colors of Objects by Using the CIE System. (2001).

Günther Wyszecki and W. S. Stiles. Color Science: Concepts and Methods, Quantitative Data and Formulae, Second Edition. John Wiley & Sons, 1982. Table I(3.3.8) p. 769.

See Also

ABC, D50, D65, Fluorescents, illuminantE

Examples

```
officialXYZ( c('A','D50','D50.ICC','D65') )
#                 X Y         Z
# A       1.0985000 1 0.3558500
# D50     0.9642200 1 0.8252100
# D50.ICC 0.9642029 1 0.8249054
# D65     0.9504700 1 1.0888300
```

---

organization *organization of a colorSpec object*

Description

Retrieve or set the organization of a **colorSpec** object.

Usage

```
## S3 method for class 'colorSpec'
organization(x)
organization(x) <- value
```

Arguments

x	a **colorSpec** R object
value	a valid organization: 'vector', 'matrix', 'df.col', or 'df.row'.

Details

If the organization of x is "vector", then x is a vector representing a single spectrum. Compare this with ts.

If the organization of x is "matrix", then x is a matrix and the spectra are stored in the columns.

If the organization of x is "df.col", then x is a data.frame with n+1 columns, where n is the number of spectra. The wavelengths are stored in column 1, and the spectra in columns 2:(n+1). This organization is good for printing to the console, and writing to files.

If the organization of x is "df.row", then x is a data.frame with n rows, where n is the number of spectra. The spectra are stored in the last column, which is a "model.matrix" with the name "spectra". The other columns preceding spectra (if present) contain additional data associated with the spectra; see extradata.

Value

organization returns a valid organization: 'vector', 'matrix', 'df.col', or 'df.row'.

Note

If x has more than 1 spectrum, then organization of "vector" is invalid and ignored.

If x has organization equal to df.row and also has extradata, then changing the organization will discard the extradata.

See Also

colorSpec; extradata

Examples

```
organization(Hoya)              # returns 'df.row'
organization(Hoya) = 'matrix'   # extradata in Hoya is silently discarded
```

---

photometric *convert illuminant spectra to photometric units*

Description

Convert radiometric units of power to photometric units, using the standard photometric weighting curve. Actinometric units (number of photons) are converted to radiometric units (power of photons) on-the-fly.

Usage

```
## S3 method for class 'colorSpec'
photometric( x )
```

Arguments

x	a **colorSpec** object with type equal to 'light'

Details

The function computes the dot product of x with the photopic responsivity spectrum y from xyz1931.1nm. This product is an Mx1 matrix, where M is the number of spectra in x. It is then multiplied by the CIE-standard coefficient of 683 lumen/watt and returned.

The 5 radiometric inputs and photometric outputs are:

radiant flux power $[watt]$	—>	**luminous flux** $[lumen]$
irradiance $[watt * m^{-2}]$	—>	**illuminance** $[lumen * m^{-2} = lux]$
radiant exitance $[watt * m^{-2}]$	—>	**luminous exitance** $[lumen * m^{-2} = lux]$
radiant intensity $[watt * sr^{-1}]$	—>	**luminous intensity** $[lumen * sr^{-1} = candela]$
radiance $[watt * sr^{-}1 * m^{-2}]$	—>	**luminance** $[candela * m^{-2} = nit]$

Value

`photometric` returns an Mx1 matrix, where M is the number of spectra in `x`. The `rownames` are `specnames(x)`, and the `colnames` are `'Y'`.
In case of ERROR it returns `NULL`.

Note

To get the right output quantity and units, the user must know the input quantity and units. For example, pre-conversion from microwatts to watts might be necessary.
The *scoptic* weighting curve (for very low light levels) is not used.

References

Poynton, Charles. Digital Video and HD - Algorithms and Interfaces. Morgan Kaufmann. Second Edition. 2012. Appendix B, pp. 573-580.

See Also

`quantity`, `type`, `radiometric`

Examples

```
photometric( solar.irradiance )  # units are watt*m^{-2}

#                     Y   # units are lux
# AirMass.0   133100.41
# GlobalTilt  109494.88
# AirMass.1.5  97142.25
```

---

plot *plot spectra*

Description

plot the spectra in a **colorSpec** object as lines

Usage

```
## S3 method for class 'colorSpec'
plot( x, color=NULL, subset=NULL, main=TRUE, legend=TRUE, CCT=FALSE, add=FALSE, ... )
```

Arguments

x	a colorSpec object
color	If color=NULL then colors are computed from the spectra themselves. If the type of x is 'material' the color is computed using illuminant D65.1nm and responder BT.709.RGB with no further normalization. Otherwise the spectrum color is faked by changing its quantity to 'power' and taking the product with BT.709.RGB. The resulting RGBs are normalized to have a maximum of 1. This RGB normalization is done *before* processing the subset argument. If color='auto' then a suitable set of colors is generated using colorRampPalette. Otherwise color is passed along to lines as the col argument, e.g. col='black'.
subset	specifies a subset of x to plot; see subset for acceptable arguments.
main	If main=TRUE then a main title is generated from the file 'path' in the metadata list, or from deparse(substitute(x)). If main=FALSE then no main title is displayed. And if main is a string then that string is used as the main title.
legend	If legend=TRUE then a pretty legend using specnames is placed in the 'topright' corner of the plot. If legend is a string it is interpreted as naming a corner of the plot and passed as such to the function legend. If legend=FALSE then no legend is drawn.
CCT	If CCT=TRUE and the type of x is 'light' then the CCT of each spectrum is added to the legend; see computeCCT.
add	If add=TRUE then lines are added to an existing plot, and these arguments are ignored: main, ylab, xlim, ylim, and log; see **Details**.
...	other graphical parameters, see **Details**

Details

Commonly used graphical parameters are:

lty, lwd passed on to lines, with no checks

pch If pch is an integer, it is passed on to points which adds points on top of the lines

ylab If ylab is a string then it is passed on to plot.default, otherwise suitable default string is generated.

xlim, ylim If xlim and ylim are 2-vectors, they are passed to plot.default. If one of the components is NA then a suitable default is supplied.

log passed on to plot.default. Care must be taken for y because many spectra are 0 at some wavelengths, and even negative. Use ylim in such cases.

Value

TRUE or FALSE

See Also

print, summary in base.

Examples

```
print( xyz1931.1nm )

xyz1931.1nm     # same thing, just calls print()
```

`plotOptimals` *Plot Optimal Colors*

Description

Consider a **colorSpec** object `x` with `type` equal to `responsivity.material` and 3 responsivity spectra. The function `plotOptimals3D` makes a wireframe plot of the *object-color solid* for `x`. The 3D drawing package **rgl** is required.
The set of all possible material reflectance functions (or transmittance functions) is convex, closed, and bounded (in any reasonable function space), and this implies that the set of all possible output responses from `x` is also convex, closed, and bounded. The latter set is called the *object-color solid*, or *Rösch Farbkörper*, for `x`. A color on the boundary of the *object-color solid* is called an *optimal color*. The special points **W** (the response to the perfect reflecting diffuser) and **0** are on the boundary of this set. The interior of the line segment of neutrals joining **0** to **W** is in the interior of the set. For more see discussion `probeOptimalColors`.

Usage

```
## S3 method for class 'colorSpec'
plotOptimals3D( x, size=c(33,33) )
```

Arguments

`x`	a **colorSpec** object with `type` equal to `responsivity.material` and 3 spectra
`size`	an integer 2-vector with the number of meridians and parallels to plot

Details

The boundary of the *object-color solid* for `x` has a natural parameterization by ω and δ, which are analogous to longitude and latitude for the sphere. See *Logvinenko* for more details. These 2 parameters define reflectance spectra with 2 or fewer transitions between 0 and 1. By default, the function draws a wireframe with 33 meridians and 33 parallels.
In addition it draws the box with opposite vertices at the "poles" **0** and **W** and the diagonal segment of neutral grays that connects **0** and **W**. It draws a small ball at the midpoint; the *Rösch Farbkörper* is symmetrical about this midpoint.

Value

`TRUE` or `FALSE`

Note 1

As both **References** point out, the plotted set of 2-transition colors is only optimal under certain conditions on x, which are fortunately true for the human responsivity functions xyz1931.1nm. But they are certainly not true for *all* x, so the plotted surface seen here might really be sub-optimal.
For general x it may require reflectance functions with 3,4,5,... transitions to define the optimals.

Note 2

If all responsivity functions of x are non-negative, the *object-color solid* of x is inside the box. If the responsivity functions of x have negative lobes, the *object-color solid* of x extends outside the box. Indeed, the box may actually be *inside* the optimals.
The responsivity functions cannot all simultaneously vanish at any wavelength. In that case the mapping from the ω and δ sphere to the output response space is not injective.

Future Work

For an object x with 2 spectra, it would not take much work to write a new function plotOptimals2D that plots the 1-transition colors for x. These are the short-pass and long-pass colors, also known as *edge colors* or *Kantenfarben*. With a lot more work it would be possible to plot the true optimals for x, with any number of transitions between 0 and 1.

References

Logvinenko, A. D. An object-color space. Journal of Vision. 9(11):5, 1-23, (2009). http://journalofvision.org/9/11/5/. doi:10.1167/9.11.5.

West, G. and M. H. Brill. Conditions under which Schrödinger object colors are optimal. Journal of the Optical Society of America. 73. pp. 1223-1225. 1983.

See Also

type, probeOptimalColors vignette **optimals**

Examples

```
# requires package rgl
library( rgl )
human = product( D50.5nm, 'slot', xyz1931.5nm, wave=400:770 )
plotOptimals3D( human )
scanner = product( D50.5nm, 'slot', BT.709.RGB, wave=400:770 )
plotOptimals3D( scanner )
```

plotPatchesRGB *Plot Patches defined by Linear RGB*

Description

RGB patches are a very common way of comparing color renderings. This function draws rectangular patches, and can also draw triangles formed by omitting one vertex from the rectangle.

Usage

```
plotPatchesRGB( obj, normalize=FALSE, gamma='sRGB', background='gray50',
                labels=TRUE, shape='full', add=FALSE )
```

Arguments

obj	an Mx3 matrix of linear RGBs for M patches, with assigned rownames. Or a data frame containing a matrix column with the exact name 'RGB', also with assigned rownames. If the data frame has optional columns LEFT,TOP,WIDTH,HEIGHT then these are used to draw the patches. The Y coordinates increases going down the page.
normalize	If TRUE then all RGBs are scaled so their maximum is 1
gamma	either the string 'sRGB' or a positive number giving the gamma function of the display; see DisplayRGBfromLinearRGB for details.
background	the color for the background behind all the patches
labels	if TRUE then the rownames are drawn using text.
shape	If shape='full' (the default) then the full rectangle is drawn. If 'half' then the rectangle is shrunk to 1/2 size, and with the same center. If shape is one of 'left', 'right', 'bottom', 'top' then only a half-rectangle is drawn, but keeping the specified side. If shape is one of 'topleft', 'topright', 'bottomleft', 'bottomright', then only a triangular half of the rectangle is drawn, but keeping the specified vertex.
add	if TRUE then the patches are added to an existing plot.

Details

If obj is a matrix, or a data frame without columns LEFT,TOP,WIDTH,HEIGHT, then the patches are drawn vertically stacked and abutting from top to bottom. See the **blueflame** vignette for examples. And see the **gallery** vignette for examples of a data frame *with* those columns.

Value

TRUE if successful, and FALSE otherwise

Note

The gamma function (aka Electro-Optical Conversion Function) of the display maps the interval [0,1] to itself in a non-linear fashion.

See Also

DisplayRGBfromLinearRGB

---

print *Convert colorSpec object to readable text*

Description

display a **colorSpec** object as readable text

Usage

```
## S3 method for class 'colorSpec'
print( x, ...)

## S3 method for class 'colorSpec'
summary( object, long=TRUE, ... )
```

Arguments

x	a colorSpec object
object	a colorSpec object
long	logical indicating whether to print metadata, calibration, and product terms
...	further arguments ignored

Details

print shows a summary of the wavelength vector, and names of all spectra. For each spectrum it prints the range of values, LambdaMax, and extradata if any.
summary prints the same as print, and if long is TRUE it also prints metadata and data about product terms (if any). The function print simply calls summary with long=TRUE.

Value

Both functions return (invisibly) the character vector that was just printed.

See Also

extradata, print, summary in base.

Examples

```
print( xyz1931.1nm )

xyz1931.1nm     # same thing, just calls print()
```

`probeOptimalColors` *compute optimal colors by ray tracing*

Description

Consider a **colorSpec** object `x` with `type` equal to `responsivity.material`. The set of all possible material reflectance functions (or transmittance functions) is convex, closed, and bounded (in any reasonable function space), and this implies that the set of all possible output responses from `x` is also convex, closed, and bounded. The latter set is called the *object-color solid* or *Rösch Farbkörper* for `x`. A color on the boundary of the *object-color solid* is called an *optimal color*. The special points **W** (the response to the perfect reflecting diffuser) and **0** are on the boundary of this set. The interior of the line segment of neutrals joining **0** to **W** is in the interior of the *object-color solid*. It is natural to parameterize this segment from 0 to 1 (from **0** to **W**).

A ray r that is based at a point on the interior of the neutral line segment must intersect the boundary of the *object-color solid* in a unique optimal color. The purpose of the function `probeOptimalColors()` is to compute that intersection point.

The function only works as stated if:

1. the number of spectra in `x` is 3 (e.g. RGB or XYZ)
2. the chromaticity diagram of `x` is convex and well-ordered (no reversals)

The 1st condition makes the situation simple enough to deal with. The 2nd condition implies that a reflectance function is optimal iff it takes the values 0 or 1, and has 0, 1, or 2 transitions; see *Logvinenko* or *West* for the proof of this. The proof in *Schrödinger* is flawed. This 2-transition condition also simplifies the situation. As an example, the CIE chromaticity diagrams (both 1931 and 1964) are convex. For counter-examples see the **References**. If a color defined by a reflectance function with 0, 1, or 2 transitions is called a *Schrödinger color* then it would be accurate to say that `probeOptimalColors()` computes *Schrödinger object colors*.

Usage

```
## S3 method for class 'colorSpec'
probeOptimalColors( x, gray, direction, tol=1.e-6, aux=FALSE )
```

Arguments

`x`	a **colorSpec** object with `type` equal to `responsivity.material` and 3 spectra
`gray`	vector of numbers in the open interval (0,1) that define neutral grays on the line segment from black to white; this neutral gray point is the basepoint of a probe ray
`direction`	a vector of 3 numbers, or a matrix with 3 columns, that define 1 or more directions for the probe rays. A vector of 3 numbers is changed to a matrix with 1 row.
`tol`	error tolerance for the intersection of probe and object-color boundary
`aux`	a logical that specifies whether to return extra performance data; see **Details**

Details

Each gray level and each direction defines a ray. So the total number of rays traced is `length(gray)` * `ncol(direction)`. The intersection problem is reduced to a 2-dimension root finding problem which is solved using Newton's Method. The initial estimate is found by precomputing a fine quadrilateral mesh over the optimal colors.
The responsivity functions may be negative, but may not all simultaneously vanish at any wavelength. In that case the mapping from the ω and δ sphere to the output response space is not injective.

Value

`probeOptimalColors` returns a `data.frame` with a row for each traced ray. There are `length(gray)` * `ncol(direction)` rays. The columns in the output are:

`gray`	the graylevel defining the $basepoint$ of the ray. $basepoint = gray * W$
`direction`	the $direction$ of the ray
`s`	computed scalar so that $basepoint + s * direction$ is optimal
`optimal`	the optimal color on the boundary; $optimal = basepoint + s * direction$
`lambda`	lambda.1 and lambda.2 at the 2 transitions, in nm. lambda.1 < lambda.2 => bandpass, and lambda.1 > lambda.2 => bandstop.
`dol`	`delta` and `omega` - the Logvinenko parameters (δ, ω) for optimal colors, plus `lambda` in nm. ω is the reparameterization of λ ; see **References**

And if aux is TRUE, these auxiliary columns related to performance:

`time_grid`	time to find initial estimate point on boundary, in seconds
`iters`	number of interations of Newton's Method to find the ray intersection
`btracks`	total # of backtracks in "damped" Newton's method
`time_newt`	time spent in Newton iterations, in seconds
`error`	root-finding error, in coordinates of the optimal color. Always less than argument `tol`

If an individual ray could not be traced (see **Known Issues**), the row contains `NA` in appropriate columns.
In case of global error, the function returns `NULL`.

Known Issues

The optimal color boundary is not differentiable at **0** and **W**. There may be numerical iteration failures near these 2 points.

References

Logvinenko, A. D. An object-color space. Journal of Vision. 9(11):5, 1-23, (2009). http://journalofvision.org/9/11/5/. doi:10.1167/9.11.5.

Schrödinger, E. (1920). Theorie der Pigmente von grösster Leuchtkraft. Annalen der Physik, 62, 603-622.

West, G. and M. H. Brill. Conditions under which Schrödinger object colors are optimal. Journal of the Optical Society of America. 73. pp. 1223-1225. 1983.

See Also

type, vignette **optimals**

`product` *Compute the product of colorSpec objects*

Description

Take a sequence of **colorSpec** objects and compute their product. The product is associative. Only certain types of sequences are allowed, see **Details**.

Usage

```
## S3 method for class 'colorSpec'
product( ... )
```

Arguments

`...`	Unnamed arguments are **colorSpec** objects, and possibly a single character string, see **Details**. Named arguments (if any) are passed to `resample`. The most important named argument is `wavelength`. The default `wavelength='identical'` means that all the **colorSpec** objects must have the same wavelength sequence; if they do not it is an ERROR. `wavelength` can be a new wavelength sequence, and all the objects are then `resample`d at these new wavelengths. `wavelength` can also be `'auto'` or `NULL` which means to compute a suitable wavelength sequence from those of the objects, see **Details**. It is OK to abbreviate the string `wavelength` (e.g. to `wave`); see **Examples**. Other possible named arguments are `method` and `span`.

Details

To explain the allowable product sequences it is helpful to introduce some simple notation for the objects:

notation	**colorSpec** `type`	**description of the object**
L	`light`	a light source
M	`material`	a material
R_L	`responsivity.light`	a light responder (aka detector)
R_M	`responsivity.material`	a material responder

It is also helpful to define a sequence of positive integers to be *conformable* iff it has at most one value greater than 1. For example, a sequence of all 1s is conformable. A sequence of all q's is conformable. The sequences `c(1,3)` and `c(1,1,4,1,1,4,1)` are conformable, but `c(1,1,4,1,3,4,1)` is not.

There are 6 types of sequences for which the product is defined:

1. $M_1 * M_2 * ... * M_m \mapsto M'$
The product of m materials is another material. Think of a stack of m transmitting filters effectively forming a new filter. If we think of each object as a matrix (with the spectra in the columns), then the product is element-by-element using R's `*` - the Hadamard product. The numbers of spectra in the terms must be conformable. If some objects have 1 spectrum and all the others have q, then the column-vector spectrums are repeated q times to form a matrix with q columns. If the numbers of spectra are not conformable, it is an ERROR and the function returns `NULL`.
As an example, suppose M_1 has 1 spectrum and M_2 has q spectra, and $m = 2$. Then the product is a material with q spectra. Think of an IR-blocking filter followed by the RGB filters in a 3-CCD camera.

2. $L * M_1 * M_2 * ... * M_m \mapsto L'$
The product of a light source followed by m materials is a light source. Think of a light source followed by a stack of m transmitting filters, effectively forming a new light source. The numbers of spectra in the terms must be conformable as in sequence 1, and the matrices are multiplied element by element.
As an example, suppose L has 1 spectrum and M_1 has q spectra, and $m = 1$. Then the product is a light source with q spectra. Think of a light source followed by a filter wheel with q filters.

3. $M_1 * M_2 * ... * M_m * R_L \mapsto R'_L$
The product of m materials followed by a light responder, is a light responder. Think of a stack of m transmitting filters in front of a camera, effectively forming a new camera. The numbers of spectra in the terms must be conformable as in sequence 1, and the matrices are multiplied element by element.
As an example, suppose R_L has 1 spectrum and M_1 has q spectra, and $m = 1$. Then the product is a light responder with q spectra. Think of a 3-CCD camera in which all 3 CCDs have exactly the same responsivity and so can be modeled with a single object R_L.

4. $L * M_1 * M_2 * ... * M_m * R_L \mapsto matrix$
The product of a light source, followed by m materials, followed by a light responder, is a matrix! The numbers of spectra in the terms must splittable into a conformable left part (L' from sequence 2.) and a conformable right part (R'_L from sequence 3.). There is a row for each spectrum in L', and a column for each spectrum in R'_L. Suppose the element-by-element product of the left part is $n \times p$ and the element-by-element product of the right part is and $n \times q$, where n is the number of wavelengths. Then the output matrix is the usual matrix product `%*%` of the transpose of the left part times and right part, which is $p \times q$.
As an example, think of a light source followed by a reflective color target with 24 patches followed by an RGB camera. The sequence of spectra is `c(1,24,3)` which is splittable into `c(1,24)` and

`c(3)`. The product matrix is 24×3. See the **gallery** vignette for a worked-out example.
Note that is OK for there to be no materials in this product; it is OK if $m = 0$. See the **blueflame** vignette for a worked-out example.

5. $L * M_1 * ... * \bullet * ... * M_m * R_L \mapsto R'_M$
This is the strangest product. The bullet symbol • means that a variable material is inserted at that slot in the sequence (or light path). For each material spectrum inserted there is a response from R_L. Therefore the product of this sequence is a material responder R_M. Think of a light source L going through a transparent object • on a flatbed scanner and into a camera R_L. For more about the mathematics of this product, see the **colorSpec-guide.pdf** in the doc directory. These material responder spectra are the same as the *effective spectral responsivities* in *Digital Color Management.*
The numbers of spectra in the terms must be conformable as in sequence 1.
In the function `product()` the location of the • is marked by any character string whatsoever - it's up to the user who might choose something that describes the typical material. For example one might choose:
`scanner = product( A.1nm, 'photo', Flea2.RGB, wave='auto')`
to model a scanner that is most commonly used to scan photographs. Other possible strings could be `'artwork'`, `'varmaterial'`, or even `'slot'`. See the **gallery** vignette for a worked-out example.

6. $M_1 * M_2 * ... * M_m * R_M \mapsto matrix$
The product of m materials followed by a material responder, is a matrix ! The sequence of numbers of spectra must be splittable into left and right parts as in sequence 4, and the product matrix is formed the same way. One reason for computing this matrix in 2 steps is that one can `calibrate` the material responder separately in a customizable way. See the **gallery** vignette for a worked-out example with a flatbed scanner.

Note that sequences 4. and 6. are the only ones that use the usual matrix product `%*%`.

The argument `wavelength` can also be `'auto'` or `NULL`. In this case the intersection of all the wavelength ranges of the objects is computed. If the intersection is empty, it is an ERROR and the function returns `NULL`. The wavelength step `step.wl` is taken to be the smallest over all the object wavelength sequences. If the minimum `step.wl` is less than 1 nanometer, it is rounded off to the nearest power of 2 (e.g 1, 0.5, 0.25, ...).
Although the function signature shows the **colorSpec** objects, followed by `wavelength`, followed by more arguments, actually they can come in any order. The unnamed arguments are taken to be **colorSpec** objects and the named arguments are taken to be arguments for `resample`.

Value

`product()` returns a **colorSpec** object or a matrix, see **Details**. In case of a **colorSpec** object, the `organization` is `'matrix'` or `'vector'`; any `extradata` is lost. However, all terms in the product are saved in `attr(*,'sequence')`. One can use `str` to inspect this attribute.
All terms are converted to `radiometric` on-the-fly and the returned **colorSpec** object is also radiometric.
In case of ERROR it returns `NULL`.

References

Edward J. Giorgianni and Thomas E. Madden. **Digital Color Management: Encoding Solutions.** 2nd Edition John Wiley. 2009. Figure 10.11a. page 141.

Wikipedia. **Hadamard product (matrices)**. http://en.wikipedia.org/wiki/Hadamard_product_%28matrices%29

See Also

wavelength, type, resample, calibrate, radiometric, step.wl

Examples

```
#  sequence 1.
path = system.file( "extdata/filters/Midwest-SP700-2014.txt", package='colorSpec' )
blocker.IR = readSpectra( path )
product( blocker.IR, Hoya, wave='auto' )


#  sequence 2.
product( subset(solar.irradiance,1), atmosphere2003, blocker.IR, Hoya, wave='auto' )


#  sequence 3.
plumbicon = readSpectra( system.file( "extdata/cameras/plumbicon30mm.txt", package='colorSpec' ) )
product( blocker.IR, subset(Hoya,1:3), plumbicon, wave='auto' )


#  sequence 4.
product( D65.1nm, Flea2.RGB )  # a 1x3 matrix, no materials
product( D65.1nm, neutralMaterial(0.01), Flea2.RGB, wave='auto' ) # a 1x3 matrix, 1 material
path = system.file( "extdata/sources/Lumencor-SpectraX.txt", package='colorSpec' )
lumencor = readSpectra( path, wave=340:660 )
product( lumencor, Flea2.RGB, wave='auto' )   # a 7x3 matrix, no materials


#   sequence 5.
#   make an RGB scanner
bluebalancer = subset(Hoya,'LB')
# combine tungsten light source A.1nm with blue light-balance filter
# use the string 'artwork' to mark the variable material slot
scanner = product( A.1nm, bluebalancer, 'artwork', Flea2.RGB, wave='auto' )


#  sequence 6.
scanner = calibrate( scanner )
target = readSpectra( system.file( "extdata/targets/N130501.txt", package='colorSpec') )
product( target, scanner, wave='auto' )  #  a 288x3 matrix
```

quantity *quantity of a colorSpec object*

Description

Retrieve or set the quantity of a **colorSpec** object.

Usage

```
## S3 method for class 'colorSpec'
quantity(x)
quantity(x) <- value

## S3 method for class 'colorSpec'
type(x)
```

Arguments

x	a **colorSpec** R object
value	a valid quantity string; see Details.

Details

There are 4 valid types, which are further divided into 14 valid quantities. All of these are strings:

For type='light'
quantity = 'power' (radiometric), or 'photons/sec' (actinometric)

For type='responsivity.light'
quantity = 'power->electrical', 'power->neural', 'power->action',
'photons->electrical', 'photons->neural', or 'photons->action'

For type='material'
quantity = 'reflectance', 'transmittance', or 'absorbance'

For type='responsivity.material'
quantity = 'material->electrical', 'material->neural', or 'material->action'

For more explanation and examples of all these see colorSpec-guide.pdf in the doc directory. Go to the main index page and look for **User guides, package vignettes and other documentation**.

Value

quantity returns the quantity of x

type returns the type of x, which depends only on the quantity.

Note

Although quantity is intended to represent a physical quantity, the units are left arbitrary in most cases. Exceptions are reflectance, transmittance, and absorbance which are dimensionless.

Changing the quantity does not change the underlying numbers. For example, changing photons/sec to power does not do numerical conversion. For this conversion, see radiometric.

Similarly, see linearize for conversion from absorbance to transmittance.

See Also

colorSpec; radiometric; linearize

---

radiometric *force a colorSpec object to be radiometric*

Description

convert a **colorSpec** object to have quantity that is radiometric (power of photons) - to make it ready for colorimetric calculations

Usage

```
## S3 method for class 'colorSpec'
radiometric( x )
```

Arguments

x	a **colorSpec** object

Details

If the quantity starts with 'power' then x is returned unchanged. And if the type of x is 'material' or 'responsivity.material' then x is returned unchanged.

Otherwise, the quantity starts with 'photons' so the quantity of x is actinometric (proportional to number of photons/sec).

If the type is 'light' then the most common actinometric unit of photon flux is (μmole of photons)/sec. The conversion equation is:

$$p = x * (10^{-6} * N_A * h * c/\lambda)$$

where p is proportional to the power of photons, x is the photon flux, N_A is Avogadro's constant, h is Planck's constant, c is the speed of light, and λ is the wavelength in meters. The output power unit is watts.
If the unit of x is not (μmole of photons)/sec, then the output should be scaled appropriately. For example, if the photon flux is exaphotons/sec, then divide the output by 0.6022.

If the type is 'responsivity.light', then the most common actinometric unit of responsivity to light is quantum efficiency (QE). The conversion equation is:

$$r = x * \lambda * e/(h * c)$$

where r is the responsivity, x is the quantum efficiency, and e is the charge of an electron (in C). The output responsivity unit is amps/watt (A/W) or coulombs/joule (C/J).
If the unit of x is not quantum efficiency, then the output should be scaled appropriately.

Value

radiometric returns a **colorSpec** object with quantity that is radiometric (power of photons) and not actinometric (number of photons).

Source

Wikipedia. **Photon counting**. http://en.wikipedia.org/wiki/Photon_counting

See Also

quantity, type, F96T12

Examples

```
sum( F96T12 )
# [1] 320.1132  photon irradiance, (micromoles of photons)*m^{-2}

sum( radiometric(F96T12) )
# [1] 68.91819  irradiance, watts*m^{-2}
```

---

readSpectra *read colorSpec objects from files*

Description

These functions read colorSpec objects from files. In case of ERROR, they return NULL. There are 5 different file formats supported; see Details.

Usage

```
readSpectra( pathvec, ... )

readSpectraXYY( path )
readSpectraSpreadsheet( path )
readSpectrumScope( path )

readSpectraCGATS( path )
readSpectraControl( path )
```

Arguments

pathvec	a character vector to (possibly) multiple files. The file extension and a few lines from each file are read and a guess is made regarding the file format.
...	optional arguments passed on to resample. The most important is wavelength. If these are missing then resample is not called.
path	a path to a single file with the corresponding format: XYY, Spreadsheet, Scope, CGATS, or Control. See Details. If the function cannot recognize the format, it returns NULL.

Details

The file formats are:

XYY
There is a column header line matching ^(wave|wl) (not case sensitive) followed by the the names of the spectra. All lines above this one are taken to be metadata. The separarator on this header line can be space, tab, or comma; the line is examined and the separator found there is used in the lines with data below. The organization of the returned object is 'df.col'. This is probably the most common file format; see the sample file ciexyz31_1.csv.

Spreadsheet
There is a line matching "^(ID|SAMPLE|Time)". This line and lines below must be tab-separated. Fields matching '^[A-Z]+([0-9.]+)nm$' are taken to be spectral data and other fields are taken to be extradata. All lines above this one are taken to be metadata. The organization of the returned object is 'df.row'. This is a good format for automated acquisition, using a spectrometer, of many spectra. See the sample file N130501.txt from **Wolf Faust**.

Scope
This is a file format used by **Ocean Optics** spectrometer software. There is a line >>>>>Begin Processed Spectral Data<<<<< followed by wavelength and power separated by a tab. There is only 1 spectrum per file. The organization of the returned object is 'vector'. See the sample file pos1-20x.scope.

CGATS
This is a complex format that is best understood by looking at some samples, such as extdata/filters/Rosco.txt; see also *CGATS.17*. The fields with spectral data match the pattern "^(nm|SPEC_|SPECTRAL_)[0-9.]+" and other fields are considered extradata. The organization of the returned object is 'df.row'.

Control
This is a personal format used for digitizing images of plots from manufacturer datasheets and academic papers. It is structured like a .INI file. There is a [Control] section establishing a simple linear map from pixels to the wavelength and spectrum quantities. Only 3 points are really necessary. It is OK for there to be a little rotation of the plot axes relative to the image. This is followed by a section for each spectrum, in XY pixel units only. Conversion to wavelength and spectral quantities happens on-the-fly. Extrapolation can be a problem, especially when the value is near 0. To force constant extrapolation, repeat the control point (knot) at the endpoint. See the sample file

Lumencor-SpectraX.txt. The organization of the returned objects is 'vector'.

Value

readSpectra returns a single **colorSpec** object or NULL in case of ERROR. If there are multiple files in pathvec and they cannot be combined using bind because their wavelengths are different, it is an ERROR. To avoid this ERROR, the wavelength argument can be used for resampling to a common wavelength. If there are multiple files, the organization of the returned object is "df.row" and the first column is the path from which the spectrum was read.

The functions readSpectraXYY, readSpectraSpreadsheet, and readSpectraScope, return a single **colorSpec** object, or NULL in case of ERROR.

The functions readSpectraCGATS and readSpectraControl are more complicated. These 2 file formats can contain multiple spectra with different wavelength vectors so both functions return a *list* of **colorSpec** objects. When readSpectra is called with a wavelength argument, these multiple spectra are resampled using resample and combined using bind into a single **colorSpec** object, which makes it much more convenient to read such files.

Note

During import, each read function tries to guess the quantity from spectrum names or other cues. For example the first line in **N130501.txt** is IT8.7/1, which indicates that the quantity is 'transmittance' (a reflective target is denoted by IT8.7/2). If a confident guess cannot be made, it makes a wild guess and issues a warning. If the quantity is incorrect, one can assign the correct value after import. Alternatively one can add a line to the header part of the file with the keyword 'quantity' followed by the correct value. It is OK to put the value in quotes. See example files under **extdata**.

References

CGATS.17 Text File Format. http://www.colorwiki.com/wiki/CGATS.17_Text_File_Format.

See Also

wavelength, quantity, metadata, resample, bind

Examples

```
#   read file with header declaring the quantity to be photons->neural
bird = readSpectra( system.file( "extdata/eyes/BirdEyes.txt", package='colorSpec' ) )
quantity(bird)   # [1] "photons->neural"
```

`resample` *resample a colorSpec Object to new wavelengths*

Description

interpolate or smooth to new wavelengths. Simple extrapolation is also performed.

Usage

```
## S3 method for class 'colorSpec'
resample( x, wavelength, method='auto', span=0.02 )
```

Arguments

`x`	a **colorSpec** object
`wavelength`	vector of new wavelengths, in nanometers
`method`	interpolation methods available are `'sprague'`, `'spline'`, and `'linear'`. Also available is `'auto'` which means to use `'sprague'` if `x` is regular, and `'spline'` otherwise. This is the CIE recommendation. Smoothing methods available are `'loess'`. See **Details**.
`span`	smoothing argument passed to `loess` before interpolation, and not used by other methods. The default value `span=0.02` is suitable for `.scope` spectra but may be too small in many other cases.

Details

For method `'sprague'` the quintic polynomial in *De Kerf* is used. Six weights are applied to nearby data values: 3 in front and 3 behind. The `'sprague'` method is only supported when `x` is regular.

For method `'spline'` the function `spline` is called with `method='natural'`. Four weights are applied to nearby data values: 2 in front and 2 behind. The `'spline'` method is supported even when `x` is irregular.

For method `'linear'` the function `approx` is called.

For method `'loess'` the function `loess` is called with the given `span` parameter. Smoothing is most useful for noisy data, e.g. raw data from a spectrometer. I have found that `span=0.02` works well for `.scope` files, but this may be too small in other cases, when it triggers an error in `loess()`.

For the non-linear methods, undershoot can sometimes create negative values, and some effort is made to clip these to 0, but only when appropriate.

Value

`resample(x)` returns a **colorSpec** object with the new `wavelength`. Other properties, e.g. `organization`, `quantity`, ..., are preserved.
In case of ERROR, the function returns `NULL`.

References

De Kerf, Joseph L. F. **The interpolation method of Sprague-Karup**. *Journal of Computational and Applied Mathematics*. volume I, no 2, 1975. equation (S).

See Also

organization, quantity, wavelength, is.regular, spline, approx, loess

Examples

```
path = system.file( "extdata/sources/pos1-20x.scope", package='colorSpec' )
y = readSpectra( path )
# plot noisy data in gray
plot( y, col='gray' )
# plot smoothed plot in black on top of the noisy one to check quality
plot( resample( y, 200:880, meth='loess', span=0.02 ), col='black', add=TRUE )
```

---

scanner *standard RGB scanners*

Description

scanner.ACES is an RGB responder to material; an ACES/SMPTE standard for scanning RGB film. The 3 spectra are defined from 368 to 728 nm, at 2nm intervals.

Format

A **colorSpec** object with quantity equal to 'material->electrical' and 3 spectra: r, g, and b.

Details

The responsivities have been scaled (by calibrate) so the response to the *perfect transmitting filter* (PTF) is RGB=(1,1,1).

References

Technical Bulletin TB-2014-005. Informative Notes on SMPTE ST 2065-2 - Academy Printing Density (APD). Spectral Responsivities, Reference Measurement Device and Spectral Calculation.

SMPTE ST 2065-3 Academy Density Exchange Encoding (ADX). Encoding Academy Printing Density (APD) Values.

The Academy of Motion Picture Arts and Sciences. Science and Technology Council. Academy Color Encoding System (ACES) Project Committee. Version 1.0 December 19, 2014. Annex A Spectral Responsivities.

See Also

quantity, calibrate

Examples

```
#   compute response of ACES scanner to the Hoya filters
product( Hoya, scanner.ACES, wave='auto' )
```

`solar.irradiance` *Standard Solar Irradiance - Extraterrestrial and Terrestrial*

Description

`solar.irradiance`
Three power spectra; from 280 to 1000 nm at 1 nm intervals. Units are $W * m^{-2} * nm^{-1}$.

`atmosphere2003`
a transmittance spectrum = a ratio of 2 spectra from `solar.irradiance`

Format

`solar.irradiance` is a **colorSpec** object with `quantity` equal to `'power'` and with 3 spectra:

`AirMass.0` Extraterrestrial Radiation (solar spectrum at top of atmosphere) at mean Earth-Sun distance

`GlobalTilt` spectral radiation from solar disk plus sky diffuse and diffuse reflected from ground on south facing surface tilted 37 deg from horizontal

`AirMass.1.5` the sum of Direct and Circumsolar irradiance, see **Details**

`atmosphere2003` is a **colorSpec** object with `quantity` equal to `'transmittance'` and with 1 spectrum:

`AirMass.1.5` the ratio of `AirMass.1.5`/`AirMass.0` from `solar.irradiance`

Details

Direct is Direct Normal Irradiance Nearly parallel (0.5 deg divergent cone) radiation on surface with surface normal tracking (pointing to) the sun, excluding scattered sky and reflected ground radiation.

Circumsolar is Spectral irradiance within +/- 2.5 degree (5 degree diameter) field of view centered on the 0.5 deg diameter solar disk, but excluding the radiation from the disk.

Source

http://rredc.nrel.gov/solar/spectra/am1.5/astmg173/astmg173.html

References

ASTM G173-03 Reference Spectra Derived from SMARTS v. 2.9.2.
Standard Tables for Reference Solar Spectral Irradiances: Direct Normal and Hemispherical on 37-deg Tilted Surface (2003)

See Also

D65, D50, daylightSpectra, vignette blueflame

---

specnames *specnames of a colorSpec object*

Description

Retrieve or set the specnames of a **colorSpec** object. Retrieve the number of spectra.

Usage

```
## S3 method for class 'colorSpec'
specnames(x)
specnames(x) <- value

## S3 method for class 'colorSpec'
numSpectra(x)
```

Arguments

x	a **colorSpec** R object
value	a character vector with length equal to the number of spectra in x.

Details

If the organization of x is "vector" then x is a vector and value is a single string, which is stored as attr(x,'specname').

If the organization of x is "matrix", then x is a matrix and value is stored as colnames(x).

If the organization of x is "df.col", then x is a data.frame with n+1 columns, where n is the number of spectra. value is stored as colnames(x)[2:(n+1)].

If the organization of x is "df.row", then x is a data.frame and value is stored as row.names(x).

Value

specnames returns a character vector with the names of the spectra.

numSpectra(x) is the same as length(specnames(x)) but much more efficient.

See Also

rownames, colnames

`standardRGB` *Convert from XYZ to some standard RGB spaces*

Description

To display an XYZ value, it typically must be converted to a standard RGB space. This is the function to do it.

Usage

```
RGBfromXYZ( XYZ, space )
```

Arguments

`XYZ`	a 3-vector, or a matrix with 3 columns with XYZs in the rows
`space`	the name of the RGB space - either `'sRGB'` or `'Adobe RGB'`. The match is case-insensitive, and spaces in the string are ignored.

Details

The input `XYZ` is multiplied by the appropriate 3x3 conversion matrix (for **sRGB** or **Adobe RGB**). These matrices are taken from *Lindbloom* and not from the corresponding *Wikipedia* articles; for the reason see **Note**.

Value

An Mx3 matrix where M is the number of rows in `XYZ`, or M=1 if XYZ is a 3-vector. Each row of the returned matrix is filled with linear RGB in the appropriate RGB space. Values outside the unit cube are not clipped. To compute non-linear display RGB, see `DisplayRGBfromLinearRGB`.
In case of error the function returns `NULL`.

Note

An RGB space is normally defined by the xy chromaticities of the 3 primaries and the white point. We follow *Lindbloom* in using the xy chromaticities of the 3 primaries and the 'official' XYZ of the white point. Using this XYZ of the white point makes the color space a little more consistent with other areas of color.
For example, for D65 we have xyY=(0.3127,0.3290,1) -> XYZ=(0.9504559,1,1.0890578). But from ASTM E308, D65 XYZ=(0.95047,1,1.08883), which is a little different.

Source

Lindbloom, Bruce. RGB/XYZ Matrices.
http://brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html

Wikipedia. **sRGB**. http://en.wikipedia.org/wiki/SRGB

Wikipedia. **Adobe RGB**. http://en.wikipedia.org/wiki/Adobe_RGB_color_space

See Also

D65, officialXYZ, DisplayRGBfromLinearRGB

Examples

```
RGBfromXYZ( officialXYZ('D65'), 'sRGB' )
#      R G B
# [1,] 1 1 1    # not really 1s, but difference < 1.e-7

RGBfromXYZ( c(.3127,0.3290,0.3583)/0.3290, 'sRGB' )
#              R        G       B
# [1,] 0.9998409 1.000023 1.00024    difference > 1.e-5
```

---

subset *extract a subset of a colorSpec Object*

Description

extract a subset of the spectra in a **colorSpec** object.
The subset can be specified by indexes, by a logical vector, or by a regular expression matching the specnames

Usage

```
## S3 method for class 'colorSpec'
subset( x, subset, ... )
```

Arguments

x	a **colorSpec** object
subset	an integer vector, a logical vector, or a regular expression
...	additional arguments ignored

Details

If subset is an integer vector, each integer must be between 1 and M, where M the number of spectra in x. No duplicates are allowed. The number of spectra returned is equal to length(subset). It is OK for the length to be 0, in which case the function returns the empty subset.

If subset is a logical vector, its length must be equal to M. The number of spectra returned is equal to the number of TRUEs in subset.

If subset is a regular expression, the number of spectra returned is equal to the number of specnames(x) matched by the expression.

Value

`subset(x)` returns a **colorSpec** object with the same `organization` as `x`. Exception: if the organization of `x` is `'vector'` and the subset is empty, then the returned object is a matrix with 0 columns.

Note

`subset` can also be used for re-ordering the spectra; just set argument `subset` to the desired permutation vector.

See Also

`organization`

Examples

```
tritanope = subset( lms2000.1nm, 1:2 )  # keep long and medium cone fundamentals, but drop the short

sml2000.1nm = subset( lms2000.1nm, 3:1 ) #   reorder from short to long
```

---

theoreticalRGB *Theoretical RGB Cameras - BT.709.RGB and Adobe.RGB*

Description

`BT.709.RGB` a theoretical RGB responder to light. The 3 responsivity spectra are constructed so that the RGBs from this theoretical camera, when displayed on an sRGB display after correct EOCF adjustment, would emit light with the same XYZs as the captured scene (up to a constant multiple).

`Adobe.RGB` a theoretical RGB responder to light. The 3 responsivity spectra are constructed so that the RGBs from this theoretical camera, when displayed on an Adobe RGB display after correct EOCF adjustment, would emit light with the same XYZs as the captured scene (up to a constant multiple).

Format

Both are **colorSpec** objects with `quantity` equal to `'power->electrical'` and 3 spectra: `r`, `g`, and `b`. The wavelengths are 360 to 830 nm at 1 nm intervals.

Details

All responsitivity spectra are linear combinations of the spectra in `xyz1931.1nm`. These 2 theoretical cameras satisfy the *Maxwell-Ives criterion* by construction. The responsivities are scaled so the response to `D65.1nm` is RGB=(1,1,1). All spectra have negative lobes.
The BT.709 primaries and white point are the same as those of sRGB (though the EOCF functions are different).

Adobe RGB and sRGB share the same Red, Blue, and White chromaticities, and only differ in the Green. This implies that for both cameras the Green output is 0 at Red and Blue, and 1 at White. This in turn implies that the Green output is identical for both cameras for all input spectra, and so the Green responsivity spectra are identical for both cameras.

References

Poynton, Charles. Digital Video and HD - Algorithms and Interfaces. Morgan Kaufmann. Second Edition. 2012. Figure 26.5 on page 302.

See Also

quantity, D65.1nm, xyz1931.1nm, vignette blueflame

---

wavelength *wavelength vector of a colorSpec object*

Description

Retrieve or set the wavelengths of a **colorSpec** object. Retrieve the number of wavelengths, and whether the wavelength sequence is regular.

Usage

```
## S3 method for class 'colorSpec'
wavelength(x)
wavelength(x) <- value

## S3 method for class 'colorSpec'
numWavelengths(x)

## S3 method for class 'colorSpec'
is.regular(x)
step.wl(x)
```

Arguments

x	a **colorSpec** R object
value	a numeric vector with length equal to the number of wavelengths in x. The wavelengths must be increasing. The unit must be nanometers.

Details

If the organization of x is 'df.col', then x is a data.frame and the wavelength vector is stored in the first column of x.
Otherwise, the wavelength vector is stored as attr(x,'wavelength').

Value

`wavelength` returns a numeric vector with the wavelength of the spectra, in nanometers.

`numWavelengths(x)` is the same as `length(wavelength(x))` but much more efficient.

`is.regular` returns TRUE or FALSE, depending on whether the step between consecutive wavelengths is a constant. A truncation error of 1.e-6 nm is tolerated here. For example, the X-Rite ColorMunki spectrometer in hi-res mode has a step of 3.33333nm, and it is considered regular.

`step.wl` returns the *mean* step in nm, whether the wavelengths are regular or not.

See Also

`colorSpec`

---

xyz1931 *CIE Color Matching Functions - 2-degree (1931)*

Description

`xyz1931.1nm`	the 1931 2° functions from 360 to 830 nm, at 1nm intervals
`xyz1931.5nm`	the 1931 2° functions from 380 to 780 nm, at 5nm intervals

Format

Each is a **colorSpec** object organized as a matrix with 3 variables.

`x`	the x-bar responsivity function
`y`	the y-bar responsivity function
`z`	the z-bar responsivity function

Source

http://www.cvrl.org

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. Table I(3.3.1). pp. 723-735.

ASTM E 308 - 01. Standard Practice for Computing the Colors of Objects by Using the CIE System. Table 1

See Also

xyz1964

Examples

```
summary(xyz1931.1nm)
white.point = product( D65.1nm, xyz1931.1nm, wave='auto' )
```

---

xyz1964 *CIE Color Matching Functions - 10-degree (1964)*

Description

xyz1964.1nm	the 10° 1964 functions from 360 to 830 nm, at 1nm intervals
xyz1964.5nm	the 10° 1964 functions from 380 to 780 nm, at 5nm intervals

Format

Each is a **colorSpec** object organized as a matrix with 3 columns.

x	the x-bar responsivity function
y	the y-bar responsivity function
z	the z-bar responsivity function

Source

http://www.cvrl.org

References

Günther Wyszecki and W.S. Stiles. **Color Science : Concepts and Methods, Quantitative Data and Formulae**. Second Edition. Wiley-Interscience. 1982. Table I(3.3.1). pp. 723-735.

ASTM E 308 - 01. Standard Practice for Computing the Colors of Objects by Using the CIE System. Table 1

See Also

xyz1931

Examples

```
summary(xyz1964.1nm)
white.point = product( D65.1nm, xyz1964.1nm, wave='auto' )
```

Index