Package ‘spacesRGB’

December 6, 2021

Type Package

Title Standard and User-Defined RGB Color Spaces, with Conversion Between RGB and CIE XYZ

Version 1.4-0

Encoding UTF-8

Date 2021-12-06

Author Glenn Davis [aut,cre]

Maintainer Glenn Davis <gdavis@gluonics.com>

Description Standard RGB spaces in-
cluded are sRGB, 'Adobe' RGB, 'ProPhoto' RGB, BT.709, and others. User-
defined RGB spaces are also possible. There is partial support for ACES Color workflows.

License GPL (>= 3)

LazyLoad yes

Depends R (>= 2.13.0)

Suggests spacesXYZ, rgl, microbenchmark, knitr, rmarkdown

Repository CRAN

NeedsCompilation no

VignetteBuilder knitr

BuildVignettes yes

Date/Publication 2021-12-06 05:30:02 UTC

R topics documented:

`Basic Parameterized TransferFunctions`

Basic Parameterized TransferFunctions

Description

Each of these functions *returns* a `TransferFunction` object, that depends on the argument values passed to it. The returned object has the parameter values "locked in". These `TransferFunction` objects are a mixture of EOTFs, OETFs, OOTFs, and general-purpose transfer functions.

Usage

```
power.OETF( gamma )
power.EOTF( gamma )
power.OOTF( gamma )
BT.1886.EOTF( gamma=2.4, Lb=0, Lw=1 )
XYZfromRGB.TF( primaries, white )
affine.TF( y0, y1 )
```

Arguments

`gamma`	the value of γ; it must be positive
`Lb`	the black level
`Lw`	the white level
`primaries`	a 3x2 or 4x2 matrix; see **Details**
`white`	a vector of length 1, 2, or 3; see **Details**
`y0`	the number to which 0 maps
`y1`	the number to which 1 maps

Details

There are 3 valid combinations of `primaries` and `white`, as given in this table:

`dim(primaries)`	`length(white)`	Description
4x2	1	`primaries` is a 4x2 matrix with CIE xy chromaticities of R,G,B,W in the rows
3x2	2	`primaries` is a 3x2 matrix with CIE xy chromaticities of R,G,B in the rows
3x2	3	`primaries` is a 3x2 matrix with CIE xy chromaticities of R,G,B in the rows

If `length(white)` is 1, then `white` is the whitepoint Y. If `length(white)` is 2, then `white` is the whitepoint xy (CIE chromaticity); the whitepoint Y is taken to be 1. If `length(white)` is 3, `white` is the whitepoint XYZ (CIE tristimulus).
`primaries` can also be a plain numeric vector of length 6 or 8, which is then converted to a 3x2 or 4x2 matrix, by row.

Value

`power.OETF()` returns a `TransferFunction` with the classical $1/\gamma$ power law. `power.EOTF()` returns a `TransferFunction` with the classical γ power law. `power.OOTF()` is the same as `power.EOTF()`, but having a different name may make the creation of new RGB spaces clearer. All three of these map [0,1] to [0,1].

`BT.1886.EOTF()` returns a `TransferFunction` that maps [0,1] to [`Lb`,`Lw`], with non-linearity given by `gamma`. The BT.1886 standard has details in Annex 1.

`XYZfromRGB.TF()` returns a 3D `TransferFunction` that is linear and maps RGB=(1,1,1) to the XYZ of white. The domain is set to the ACES cube $[-65504, 65504]^3$ and the range is set to the smallest enclosing box. For the inverse one can use `XYZfromRGB.TF()^-1`.

`affine.TF()` returns a 1D `TransferFunction` that maps $0 \to y_0$ and $1 \to y_1$ in an affine way. One must have $y_0 \neq y_1$, but is is OK to have $y_0 > y_1$. No quantities are associated with these values; the function is intended for arbitrary 1D scaling.

References

BT.1886. Reference electro-optical transfer function for flat panel displays used in HDTV studio production. March 2011.

See Also

`TransferFunction`

`Basic TransferFunctions`
Basic TransferFunctions

Description

`sRGB.EOTF`	the standardized sRGB transfer function
`BT.709.EOTF`	the standardized BT.709 transfer function
`BT.2020.EOTF`	the standardized BT.2020 transfer function
`ProPhotoRGB.EOTF`	the standardized ProPhotoRGB transfer function
`SMPTE.240M.EOTF`	the standardized SMPTE-240M transfer function

Details

All of these are built-in `TransferFunction` objects; they have no parameters and are ready-to-go. All are EOTFs and have domain and range the interval [0,1], and all are monotone increasing. All are defined in 2 pieces, with a linear segment near 0. All are easily inverted.

References

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

BT.709. Parameter values for the HDTV standards for production and international programme exchange. June 2015.

BT.2020. Parameter values for ultra-high definition television systems for production and international programme exchange. October 2015.

Wikipedia. **ProPhoto RGB**. https://en.wikipedia.org/wiki/ProPhoto_RGB_color_space.

ANSI/SMPTE 240M-1995. SMPTE STANDARD for Television Signal Parameters 1125-Line High-Definition Production Systems.

See Also

`TransferFunction`

Examples

```
# make plot comparing 5 EOTFs
colvec = c('black','red','blue','green','orange')
plot( sRGB.EOTF, color=colvec[1], main="The Basic 5 EOTFs" )
plot( BT.709.EOTF, color=colvec[2], add=TRUE )
plot( BT.2020.EOTF, color=colvec[3], add=TRUE )
plot( ProPhotoRGB.EOTF, color=colvec[4], add=TRUE )
plot( SMPTE.240M.EOTF, color=colvec[5], add=TRUE )
legend( 'topleft', legend=c('sRGB','BT.709','BT.2020','ProPhotoRGB','SMPTE.240M'),
            col=colvec, bty='n', lty=1, lwd=2 )
```

`composition` *The composition of TransferFunction objects*

Description

The function `composition(TF1,TF2)` returns a `TransferFunction` that is `TF1` followed by `TF2`. Four equivalent infix operators are also available.

Usage

```
## S3 method for class 'TransferFunction'
composition( TF1, TF2 )

## S3 method for class 'TransferFunction'
 TF1 * TF2
## S3 method for class 'TransferFunction'
 TF1 %;% TF2
## S3 method for class 'TransferFunction'
 TF1 %X% TF2
## S3 method for class 'TransferFunction'
 TF2 %O% TF1

identity.TF

## S3 method for class 'TransferFunction'
is.identity( TF )
```

Arguments

`TF1`	a `TransferFunction` object
`TF2`	a `TransferFunction` object
`TF`	a `TransferFunction` object

Details

In order to be composed, the dimensions of `TF1` and `TF2` must be equal, or the dimension of one of them must be 1. In the latter case, the function is applied to each coordinate in exactly the same way.

All the above represent the function `TF1` followed by `TF2`. In mathematics this operation is usually called *composition of functions* (and *composition of morphisms* in category theory), and in computer science and BT.2100 and BT.2390 it is called the *concatenation*. In BT.2390 it is also called the *cascade*.

The ACES literature uses infix notation with the symbol `'+'` which is unfortunate because in mathematics the plus symbol is only used for commutative operations, which composition certainly is not. The symbol `'*'` is offered here as an alternative, since `'*'` does not imply commutativity (e.g. as in MATLAB's matrix multiplication). In computer science the symbol `';'` is common, and so

%;% is offered as an alternative. In BT.2100 and BT.2390 the symbol $\otimes$ is used, and so %X% is offered as an alternative. And finally, in mathematics $\circ$ is used but in the opposite order, so that TF2 %O% TF1 is identical to composition(TF1,TF2).
Each TransferFunction object is actually a list of so-called *elementary* transfer functions. If TF1 has M_1 elementary functions and TF2 has M_2 elementary functions, then composition(TF1,TF2) has $\leq M_1 + M_2$ elementary functions. It can be strictly less if there is cancellation of elementary functions at the end of TF1 and the beginning of TF2.

Value

composition(TF1,TF2) returns a TransferFunction object, which applies TF1 followed by TF2. The individual objects TF1 and TF2 are stored inside the returned object. In case of ERROR it returns NULL. The 4 infix operators above all invoke composition().

identity.TF is a built-in global TransferFunction object which is a universal identity for composition. This means that for any TransferFunction TF, TF*identity.TF = identity.TF*TF = TF. Moreover, TF*TF^-1 = TF^1*TF = identity.TF. This is *not* the same as base::identity().

is.identity(TF) tests whether TF is the universal identity, and returns TRUE or FALSE.

References

Technical Bulletin. TB-2018-002. ACES Output Transform Details. June 2018 (draft).

ACES Retrospective and Enhancements March 2017.

BT.2100. Image parameter values for high dynamic range television for use in production and international programme exchange. June 2017.

BT.2390. High dynamic range television for production and international programme exchange. April 2018.

See Also

TransferFunction, transfer(), inverse()

Examples

```
comp = power.OOTF(2.2) * power.OOTF(1.4)
x = 0:100 / 100
max( abs( transfer(comp,x) - transfer(power.OOTF(2.2*1.4),x) ) )  #  1.110223e-16

comp * comp^-1
## This is a universal identity TransferFunction.

is.identity(comp * comp^-1)          # TRUE

identical( comp * identity.TF, comp )  # TRUE
```

`Digital Cinema Distribution Master`
Digital Cinema Distribution Master, the EOTF of DCDM

Description

`DCDM.EOTF`	the standardized DCDM transfer function

Details

This is a `TransferFunction` designed to be applied to XYZ, instead of the usual RGB. The electrical encoding of XYZ is denoted X'Y'Z'. The EOTF is:

$$X = (52.37/48) * (X')^{2.6}$$

and similarly for Y and Z.

References

SMPTE Standard RP 431-2. D-Cinema Quality - Reference Projector and Environment for the Display of DCDM in Review Rooms and Theaters. 2011.

See Also

`TransferFunction`

`Full Range to SMPTE Range`
Full Range to SMPTE Range

Description

`FullRangeToSMPTE.TF`	the standardized SMPTE range transfer function

Details

This is a `TransferFunction` object that maps from non-linear display signal RGB to itself. It maps from the full range [0,1] to the smaller range [64/1023, 940/1023] ≈ [0.06256109, 0.9188661].

The latter is the 10-bit "legal-SMPTE"" range. It does this in an affine way, and in fact simply uses `affine.TF()`.

See Also

`affine.TF()`, `TransferFunction`

`Hybrid Log-Gamma Transform` *Hybrid Log-Gamma Transform*

Description

The Hybrid Log-Gamma OETF is a transfer function that allows for the display of high dynamic range (HDR) video. The version here is that supported by the ACES (Academy Color Encoding System) and HEVC (High Efficiency Video Coding) standards.
For use with ACES, a specialized HLG-based OOTF is provided that references the Perceptual Quality (PQ) EOTF. It converts the ST.2084 (PQ) output to HLG using the method specified in Section 7 of BT.2390-0.

Usage

```
HLG.OETF()
HLG.OOTF( gamma=1.2, Lb=0, Lw=1000 )
```

Arguments

`gamma`	the applied exponent, from scene linear to display linear
`Lb`	the luminance of black, in cd/m^2, or nit.
`Lw`	the luminance of white, in cd/m^2, or nit.

Details

`HLG.OOTF()` is 3D and does not operate on each channel independently. It uses a scaling factor based on these RGB weights - (0.2627,0.6780,0.0593) - from Section 7 of BT.2390.

Value

`HLG.OETF()` returns a univariate `TransferFunction` that maps linear scene RGB to signal display RGB. The interval [0,1] maps to [0,1] (as in the HVEC standard).
`HLG.OOTF()` returns a multivariate `TransferFunction` of dimension 3 that maps linear scene RGB to linear display RGB. It maps the cube $[0,1]^3$ to the cube $[\texttt{Lb},\texttt{Lw}]^3$, but the image is only a proper subset of the cube.

References

ST-2084. SMPTE Standard - High Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. 2014.

BT.2390. High dynamic range television for production and international programme exchange. April 2018.

H.265 : High Efficiency Video Coding. https://www.itu.int/rec/T-REC-H.265-201802-I/en. 2018-02-13.

See Also

`TransferFunction`, `PQ.EOTF`

`inverse` *The inverse of a TransferFunction Object*

Description

The function `inverse()` returns a `TransferFunction` that is the inverse of the argument (if the argument is invertible).

`is.invertible()` tests whether a **TransferFunction** object has an inverse.

Usage

```
## S3 method for class 'TransferFunction'
inverse( TF )

## S3 method for class 'TransferFunction'
 TF ^ n

## S3 method for class 'TransferFunction'
is.invertible( TF )
```

Arguments

`TF`	a `TransferFunction` object
`n`	an integer exponent; valid values are -1, 0, and 1

Value

`inverse()` returns a `TransferFunction` object obtained by swapping `fun` and `funinv` and swapping `domain` and `range`. The names of the elements composing `TF` are changed appropriately. If `TF` is not invertible, it returns `NULL`.

If `n=-1`, `TF^n` returns `inverse(TF)`. If `n=1`, it returns `TF`. If `n=0`, it returns the universal `identity.TF`. For any other value of `n` it returns `NULL`.

`is.invertible()` returns `TRUE` or `FALSE`.

See Also

identity.TF

`Linear RGB and XYZ Calculation`

Convert Signal RGB coordinates to XYZ, or Linear RGB

Description

Convert signal RGB coordinates to XYZ, or to linear RGB

Usage

```
XYZfromRGB( RGB, space='sRGB', which='scene', TF=NULL, maxSignal=1 )

LinearRGBfromSignalRGB( RGB, space='sRGB', which='scene', TF=NULL, maxSignal=1 )
```

Arguments

`RGB`	a numeric Nx3 matrix with non-linear signal RGB coordinates in the rows, or a vector that can be converted to such a matrix, by row. These should be in the appropriate cube $[0,\mathtt{maxSignal}]^3$.
`space`	the name of an installed RGB space. The name matching is partial and case-insensitive.
`which`	the output XYZ or linear RGB - either `'scene'` or `'display'`. Usually the OOTF for the space is the identity and so these two are the same.
`TF`	if not `NULL`, `TF` is a `TransferFunction` that overrides the appropriate transfer function of `space`. `TF` can also be a positive number. If `TF=1`, then `TF` is set to `identity.TF`, so the returned RGB values are not clamped (see **Value**). If `TF!=1` it is used to create either `power.EOTF()` or `power.OETF()` as approriate. If `TF` is not `NULL` in `LinearRGBfromSignalRGB()`, then `space` is ignored.
`maxSignal`	maximum value of the input signal `RGB`. Other common values are 100, 255, 1023, 4095, and 65535. Even when 1, they are still taken to be non-linear signal values.

Value

`XYZfromRGB()` returns a `data.frame` with N rows and these columns:

`XYZ`	the calculated XYZ vectors. These are for viewing under the white point of the given RGB space.
`OutOfGamut`	a logical vector. TRUE means the input signal RGB is outside the cube $[0,\mathtt{maxSignal}]^3$. If `TF` is not `identity.TF`, then the input signal RGBs are clamped to this cube before further calculations.

`LinearRGBfromSignalRGB()` returns a `data.frame` with N rows and these columns:

`RGB`	the calculated linear RGB vectors, either scene linear or display linear.
`OutOfGamut`	a logical vector. TRUE means the input signal RGB is outside the cube `[0,maxSignal]`^{3}. If `TF` is not `identity.TF`, then the signal RGBs are clamped to this cube before linearizing.

In case of error, both functions return NULL.

References

Wikipedia. **RGB color space**. https://en.wikipedia.org/wiki/RGB_color_space

See Also

`RGBfromXYZ()`, `SignalRGBfromLinearRGB()`, `installRGB()`

Examples

```
XYZfromRGB( c(128,200,255, 0,0,0,  255,255,255), max=255 )$XYZ * 100
##              X         Y        Z
##  [1,] 47.60334  53.11601 102.3549
##  [2,]  0.00000   0.00000   0.0000
##  [3,] 95.04559 100.00000 108.9058


XYZfromRGB( c(128,200,255, 0,0,0,  255,255,255), space='Adobe', max=255 )$XYZ   * 100
##              X         Y        Z
##  [1,] 42.36398  50.82876 103.8704
##  [2,]  0.00000   0.00000   0.0000
##  [3,] 95.04559 100.00000 108.9058
```

`metadata` *metadata of a TransferFunction object*

Description

Retrieve or set the metadata of a **TransferFunction** object. The user is free to set this as he/she wishes.

Usage

```
## S3 method for class 'TransferFunction'
metadata( x, ...)

## S3 replacement method for class 'TransferFunction'
metadata( x, add=FALSE ) <- value
```

Arguments

x	a TransferFunction R object
...	optional names of metadata to return
value	a named list. If add is FALSE, value replaces any existing metadata. If add is TRUE, value is appended to the existing list of metadata. If a name already exists, its value is updated using modifyList(). Unnamed items in value are ignored.
add	if add=FALSE, any existing metadata is discarded. If add=TRUE then existing metadata is preserved, using modifyList().

Details

The metadata list is stored as attr(x,'metadata'). After construction this list is empty.

Value

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

See Also

modifyList

Examples

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

# get just the number 'gamma'
metadata( TF, 'gamma' )

# alternative method to get just the number 'gamma'
metadata( TF )$gamma

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

## End(Not run)
```

---

miscTF *Miscellaneous TransferFunction Methods*

Description

Miscellaneous TransferFunction methods

Usage

```
## S3 method for class 'TransferFunction'
dimension( TF )

## S3 method for class 'TransferFunction'
domain( TF )

## S3 method for class 'TransferFunction'
orientation( TF )
```

Arguments

TF	a TransferFunction object

Value

dimension() returns a positive integer - the dimension of the domain and range of TF. If TF is a universal identity, it returns NA.

domain() returns a 2xN matrix with the domain box of TF, where N is dimension(TF). If TF is a universal identity, it returns NA.

orientation() returns a real number. If the value is positive it means that TF preserves orientation. If the value is negative it means that TF reverses orientation. When dimension(TF)=1, this simply corresponds to the function being monotone increasing or decreasing, respectively. In case of ERROR, the function returns NA.

See Also

TransferFunction identity.TF

Examples

```
TF = affine.TF( 1, 108 )

dimension(TF) # 1

orientation(TF) # 107

orientation( affine.TF( 100, 1 ) ) # -99

domain(TF)
##      AU
##  min  0
##  max  1
```

Opto-Optical Transfer Function, parameterized
Opto-Optical Transfer Function, general

Description

This parameterized OOTF maps from ACES (linear scene) RGB to linear display RGB (both of these are optical in nature).
This transform bypasses non-linear signal display RGB (which is electrical in nature).

Usage

```
general.OOTF( display_pri, Ymin=0.00010, Ymid=7.2, Ymax=108,
                observerWP=NULL, limiting_pri=NULL,
                surround='dark', dynrange='SDR', glowmod='1.1', redmod='1.1' )
```

Arguments

display_pri	a 4x2 matrix containing the display primaries, or a numeric vector of length 8 that can be converted to such a matrix, by row. Some built-in matrices are REC709_PRI, etc. This argument cannot be NULL.
Ymin	the minimum display luminance, in cd/m^2, or nit.
Ymid	the middle display luminance, in cd/m^2, or nit.
Ymax	the maximum display luminance, in cd/m^2, or nit.
observerWP	the xy chromaticity of the assumed observer whitepoint. This is used to make a Chromatic Adaptation Transform (CAT) from the ACES whitepoint (approximately D60) to the assumed observer whitepoint. If observerWP is NULL, it is taken from display_pri. If ACES and observer whitepoints are the same, there is no CAT.
limiting_pri	a 4x2 matrix containing the limiting primaries, or a numeric vector of length 8 that can be converted to such a matrix, by row. If limiting_pri is not NULL, and not equal to display_pri, then the output RGB is clamped to the RGB cube that corresponds to limiting_pri.
surround	The level of the surround luminance. Valid values are 'dark' and 'dim'. If the level is 'dark' there is no special color compensation. Partial matching is enabled and matching is case-insensitive.
dynrange	the dynamic range of the display system. Valid values are 'SDR' (standard dynamic range) and 'HDR' (high dynamic range). If the value is 'HDR' then surround is ignored. Matching is partial and case-insensitive.
glowmod	the version of the Glow Modifier to use. The only version currently supported is "1.1". glowmod can also be NULL, NA, or FALSE, which means to use no Glow Modifier at all.

redmod	the version of the Red Modifier to use. The only version currently supported is "1.1". This string can also be "1.1+pinv" which means to use a precision inverse; the forward transfer is exactly the same. This precision inverse uses an iterative root-finder, and is slower than the approximate default inverse. redmod can also be NULL, NA, or FALSE, which means to use no Red Modifier at all.

Details

The transfer is complicated; here is a summary of the steps:

1. glow module (see argument glowmod)
2. red modifier (see argument redmod)
3. matrix conversion from AP0 RGB $\rightarrow$ AP1 RGB
4. clamp to non-negative RGB
5. global desaturation (as in RRT.TF)
6. single-stage tone-scale (SSTS) using Ymin, Ymid, and Ymax
7. absolute luminance to linear code-value, in cube $[0,1]^3$
8. matrix conversion from AP1 RGB to XYZ
9. dim surround compensation (optional, see arguments surround and dynrange)
10. clamp XYZ to limiting primaries (optional, see argument limiting_pri)
11. adapt XYZ from ACES whitepoint to observer whitepoint (optional, see argument observerWP)
12. matrix conversion from XYZ to linear display RGB (see argument display_pri)
13. scale and roll-white to avoid clipping (optional, only when observerWP is ACES whitepoint and display whitepoint is D65 or DCI whitepoint)
14. clamp to non-negative RGB

Value

general.OOTF() returns a TransferFunction of dimension 3 that maps ACES RGB to linear display RGB.
The domain of the returned TransferFunction depends on the values of Ymin, Ymid, and Ymax.
The range is $[0,1]^3$, for which clamping may be used.
The metadata contains the display primaries and whitepoint, which is useful in installRGB().

Source

This function was based on source code at: https://github.com/ampas/aces-dev; especially the file ACESlib.OutputTransforms.ctl. This transform is a sub-transform of the function outputTransform(); it omits the final EOTF^{-1} and optional Full-to-SMPTE range.

References

ST 2065-1:2012. SMPTE Standard - Academy Color Encoding Specification (ACES). 2013.

See Also

TransferFunction, installRGB(), metadata(), Standard Primaries

`Partial Output Device Transform, parameterized` *Partial Output Device Transform, general*

Description

A partial Output Device Transform (PODT) maps from OCES to linear display RGB (both of these are optical in nature). The adjective "partial" is used because this is an ODT that omits the final OETF (which maps from linear display RGB to signal display RGB).
This PODT is parameterized.

Usage

```
general.PODT( display_pri, Ymax=1, observerWP=NULL, surround='dark', limiting_pri=NULL )
```

Arguments

`display_pri`	a 4x2 matrix containing the display primaries, or a numeric vector of length 8 that can be converted to such a matrix, by row. Some built-in matrices are `REC709_PRI`, etc. `display_pri` can also be `NULL`, which means that the PODT maps to XYZ, instead of RGB. This is used in the case of DCDM (Digital Cinema Distribution Master). See the User Guide Appendix for examples of this. `display_pri` is stored in the metadata of the returned object and later used in `installRGB()` (if the PODT is passed in an argument).
`Ymax`	the maximum luminance of the output device, in cd/m^2 (or nits). This has no effect on the PODT itself. It is stored in the metadata and later used in `installRGB()` (if the PODT is passed in an argument) when computing the 3x3 matrix that transforms from display RGB to display XYZ.
`observerWP`	the xy chromaticity of the assumed observer whitepoint. This is used to make a Chromatic Adaptation Transform (CAT) from the ACES whitepoint (approximately D60) to the assumed observer whitepoint. If `observerWP` is `NULL`, it is taken from `display_pri`. If `display_pri` is `NULL`, then it is taken from `limiting_pri`. If `limiting_pri` is `NULL`, or if two whitepoints are the same, then there is no CAT.
`surround`	The level of the surround luminance. Valid values are `'dark'` and `'dim'`. If the level is `'dark'` there is no special color compensation. Partial matching is enabled and matching is case-insensitive.
`limiting_pri`	a 4x2 matrix containing the limiting primaries, or a numeric vector of length 8 that can be converted to such a matrix, by row. If `limiting_pri` is not `NULL`, and not equal to `display_pri`, then the output RGB is clamped to the RGB cube that corresponds to `limiting_pri`.

Details

The transfer is complicated; here is a summary of the steps:

1. matrix conversion from AP0 RGB $\rightarrow$ AP1 RGB
2. clamp to non-negative RGB
3. segmented spline, assuming `CINEMA_WHITE`=48 nit
4. absolute luminance to linear code-value, in cube $[0,1]^3$
5. scale and roll-white to avoid clipping (optional, only when `observerWP` is ACES whitepoint and display whitepoint is D65 or DCI whitepoint)
6. dim surround compensation with conversion to XYZ and back again (optional, see argument `surround`)
7. matrix conversion from AP1 RGB to XYZ
8. adapt XYZ from ACES whitepoint to observer whitepoint (optional, see argument `observerWP`)
9. clamp XYZ to limiting primaries (optional, see argument `limiting_pri`)
10. matrix conversion from XYZ to linear display RGB (but not for DCDM, see argument `display_pri`)
11. clamp linear display RGB (or XYZ for DCDM) to the cube $[0,1]^3$

Value

`general.PODT()` returns a `TransferFunction` of dimension 3 that maps OCES RGB to linear display RGB. The domain is $[0,10000]^3$ and the range is $[0,1]^3$.
The `metadata` contains the display primaries and whitepoint, which is useful in `installRGB()`.

Source

This function was based on source code at: `https://github.com/ampas/aces-dev`; especially the files under the folder `aces-dev-master/transforms/ctl/odt/`.

References

ST 2065-1:2012. SMPTE Standard - Academy Color Encoding Specification (ACES). 2013.

See Also

`TransferFunction`, `installRGB()`, `metadata()`, `RRT.TF`, Standard Primaries

`Perceptual Quality Transform` *Perceptual Quality Transform*

Description

The Perceptual Quantizer is a transfer function that allows for the display of high dynamic range (HDR) video.

Usage

```
PQ.EOTF( Lmax=10000 )
```

Arguments

`Lmax`	the maximum luminance, in cd/m^2, or nit.

Value

`PQ.EOTF()` returns a `TransferFunction` that maps signal-display RGB to linear-display RGB. The interval [0,1] maps to [0,`Lmax`].

References

ST-2084. SMPTE Standard - High Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. 2014.

See Also

`TransferFunction`

`plot` *plot a TransferFunction*

Description

plot a `TransferFunction` of dimension 1, 2, or 3.

Usage

```
## S3 method for class 'TransferFunction'
plot( x, color='red', main=TRUE, add=FALSE, ... )
```

Arguments

`x`	a `TransferFunction` object with dimension N = 1, 2, or 3.
`color`	Any value acceptable as the `col` argument to `graphics::lines()`. If N=3 this argument is currently ignored.
`main`	If `main=TRUE` then a main title is generated from the object `x`. If `main=FALSE` then no main title is displayed. And if `main` is a character string then that string is used as the main title. If N=3 this argument is currently ignored.
`add`	If `add=TRUE` then the lines are added to an existing plot. If N=3 this argument is currently ignored.
`...`	other graphical parameters, see **Details**

Details

If N=1 a conventional plot is drawn using `graphics::lines()`. Commonly used graphical parameters applicable when N=1 are:

`log` passed on to `plot.default()`. Care must be taken because many transfer functions have 0 in their domains.

If N=2 a grid is generated in the domain box, and the image of that grid is plotted using using `graphics::lines()`.
If N=3 a grid is generated in the domain box, and the image of that grid is plotted in 3D using `rgl::lines3d()`.

Value

`TRUE` or `FALSE`

See Also

`graphics::lines()`, `rgl::lines3d()`

`plotPatchesRGB` *Plot Patches defined by 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, space='sRGB', which='signal', maxColorValue=1,
                background='gray50', shape='full', add=FALSE, labels=FALSE, ... )
```

Arguments

`obj`	an Nx3 matrix of RGBs for N patches, preferably with assigned `rownames`. `obj` can also be a `data.frame` containing a unique matrix column whose name starts with the string `'RGB'`. If `obj` has columns `LEFT,TOP,WIDTH,HEIGHT` then these are used to place the patches, with the Y coordinate increasing going *down* the page. If `obj` has columns `LEFT,BOTTOM,WIDTH,HEIGHT` then these are used to place the patches, with the Y coordinate increasing going *up* the page. If there are no columns defining the location and size of the patches, then defaults are supplied, see **Details**.
`space`	the name of an installed RGB space. When the input RGB is linear, a transfer function of this RGB space is used to convert linear RGB to signal RGB, see **Details**.
`which`	the meaning of the RGB values in `obj`. Valid values are `'signal'`, `'scene'`, and `'display'`. See the Figure on page 2. Partial matching is used. For the RGB processing, see **Details**.
`maxColorValue`	a positive number used for input RGB scaling, see **Details**
`background`	the color for the background behind all the patches. If it is a character string, it is passed directly to `par()` as parameter `bg`. If it is a numeric vector of length 3, it is processed just like the input RGB in `obj`, see **Details**. If it is a number, it is interpreted as graylevel, replicated to length 3, and treated as in the previous sentence.
`shape`	If `shape='full'` (the default) then the full rectangle is drawn. If `shape='half'` then the rectangle is shrunk to 1/2 size, and with the same center. If shape is one of `'left'`, `'right'`, `'bottom'`, or `'top'` then only a half-rectangle is drawn, and keeping the specified side. If shape is one of `'topleft'`, `'topright'`, `'bottomleft'`, or `'bottomright'`, then only a triangular half of the rectangle is drawn, and keeping the specified vertex. If `shape='hhex'` then a hexagon is drawn inscribed in the rectangle with 2 horizontal opposite sides (in contact with the rectangle sides). And if the aspect ratio of the rectangle is $2:\sqrt{3}$ the hexagon is regular. If `shape='vhex'` then the inscribed rectangle has 2 vertical opposite sides.
`add`	if `TRUE` then the patches are added to an existing plot
`labels`	controls how the patches are labeled, using `rownames(obj)`, or 1:N if `rownames(obj)` is `NULL`. The function used is `graphics::text()`. If `labels=FALSE` then no labels are plotted. If `labels=TRUE` then labels are plotted in the center of the patch when there are columns defining the location and size of the patches, and to the right of the patch otherwise. `labels` can also be a character string defining the location where the labels are drawn. It can be the side of the patches, i.e. `left`, `right`, `top`, or `bottom`, or the corner of the patches, i.e. `bottomleft`, `bottomright`, `topleft`, or `topright`.
`...`	additional arguments passed to `graphics::text()`. For example: `adj`, `cex`, etc.

Details

If `which='signal'` then the input RGBs are converted to hex codes using `rgb()` using the `maxColorValue` argument, and the `space` argument is ignored.
If `which='scene'` or `which='display'` then the input linear RGBs are normalized by division by `maxColorValue`, and then converted to signal RGB using `SignalRGBfromLinearRGB()` with the `space` argument. The signal RGB is then converted to hex codes using `rgb()`.

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.

Value

`TRUE` if successful, and `FALSE` otherwise

See Also

`SignalRGBfromLinearRGB()`, `installRGB()`, `rgb()`

Examples

```
set.seed(0)
RGB = round( 255 * matrix( runif(6*3), 6, 3 ) )
plotPatchesRGB( RGB, max=255 )
```

---

print *Print Basic Facts about a TransferFunction*

Description

Each `TransferFunction` object is actually a list of so-called *elementary* transfer functions; for details on this see `composition()`. This `print()` calls an internal `print()` function for each elementary function individually. The internal `print()` also calls an internal `validate()` (with default arguments) which runs some basic tests and formats the results nicely for printing, see `validate()`.

Usage

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

Arguments

`x`	a `TransferFunction` object consisting of M *elementary* transfer functions
`...`	further arguments ignored, but required by the generic `print()`

Value

The function returns `TRUE` or `FALSE`.

See Also

TransferFunction, validate(), composition()

Examples

```
tf = sRGB.EOTF^-1 * power.EOTF(2.5)
tf
## #--------------------   [sRGB.EOTF]^-1    ---------------------#
## [sRGB.EOTF]^-1 is a univariate TransferFunction.
## domain:      [0,1]  (linear display)
## range:       [0,1]  (non-linear signal)
## invertible:  Yes
## orientation: preserving
## range-test points = 1300, max(distance)=0.
## validation:  Passed
## #--------------------    power.EOTF(2.5)    ---------------------#
## power.EOTF(2.5) is a univariate TransferFunction.
## domain:      [0,1]  (non-linear signal)
## range:       [0,1]  (linear display)
## invertible:  Yes
## orientation: preserving
## range-test points = 1300, max(distance)=0.
## validation:  Passed
```

Reference Rendering Transform

Reference Rendering Transform

Description

The fixed RRT.TF corresponds to the RRT in aces-dev 1.1.
A parameterized version general.RRT() is also provided - for experimentation. This one returns a TransferFunction with the argument values "locked-in".

Usage

```
RRT.TF
general.RRT( glowmod="1.1", redmod="1.1" )
```

Arguments

glowmod	the version of the Glow Modifier to use. The only version currently supported is "1.1". glowmod can also be NULL, NA, or FALSE, which means to use no Glow Modifier at all.

`redmod`	the version of the Red Modifier to use. The only version currently supported is `"1.1"`. This string can also be `"1.1+pinv"` which means to use a precision inverse; the forward transfer is exactly the same. This precision inverse uses an iterative root-finder, and is slower than the approximate default inverse. `redmod` can also be `NULL`, `NA`, or `FALSE`, which means to use no Red Modifier at all.

Details

`RRT.TF` is a `Transferfunction` that maps ACES RGB to OCES RGB. Both spaces are relative to the AP1 primaries. `RRT.TF` is constructed by calling `general.RRT()` with its default arguments. The transfer is complicated; here is a summary of the steps starting with ACES RGB as input:

1. glow module (see argument `glowmod`)
2. red modifier (see argument `redmod`)
3. matrix conversion from AP0 RGB $\rightarrow$ AP1 RGB
4. clamp to non-negative RGB
5. global desaturation
6. segmented spline, applied to each channel separately
7. matrix conversion from AP1 $\rightarrow$ AP0 (now OCES RGB)

Value

`general.RRT()` returns a `Transferfunction` that maps ACES RGB to OCES RGB. The domain is $[0,47000]^3$ and the range is $[0,10000]^3$.

References

ST 2065-1:2012. SMPTE Standard - Academy Color Encoding Specification (ACES). 2013.

See Also

`TransferFunction`

`RGB Space Management` *Manage RGB Spaces*

Description

Install/uninstall RGB spaces in a dictionary. The dictionary comes with 8 RGB spaces pre-installed. To query the dictionary, use `getRGB()` and `summaryRGB()`.

Usage

```
installRGB( space, scene, display=NULL, OETF=NULL, EOTF=NULL, OOTF=NULL, overwrite=FALSE )
uninstallRGB( space )
```

Arguments

space
: name of the RGB space to install or uninstall or query. After the RGB space is installed, the string space can be used in the conversion functions, e.g. RGBfromXYZ() and XYZfromRGB().

scene
: the specification of the scene primaries and whitepoint. There are many options here. The 1st option is a 4x2 matrix with the CIE xy chromaticities of R,G,B,W in the rows, in that order. The 2nd option is a list with 2 items: the primaries data and the whitepoint data. These are described in the section **Primaries and Whitepoint Details** below. If scene is NULL, it will duplicate the data from argument display.

display
: the specification of the display primaries and whitepoint. The options are the same as for argument scene. If this is NULL (the default), the function will first look at the metadata of the transfer functions. These built-in transfer functions - general.OOTF() and general.PODT() - already have this metadata assigned. If the metadata is not found, it will duplicate the data from argument scene.

OETF
: a TransferFunction of dimension 1 or 3.
OETF can also be a positive number γ, which is then passed to power.OETF() to create the TransferFunction. This is the classical $1/\gamma$ power law.
OETF can also be NULL; see section **Transfer Function Details** for valid combinations.

EOTF
: a TransferFunction of dimension 1 or 3.
EOTF can also be a positive number γ, which is then passed to power.EOTF() to create the TransferFunction. This is the classical γ power law.
EOTF can also be one of these strings: 'sRGB', 'ProPhotoRGB', 'BT.709', 'BT.2020', or '240M', which then installs the appropriate special EOTF function.
EOTF can also be NULL; see section **Transfer Function Details** for valid combinations.

OOTF
: a TransferFunction of dimension 1 or 3.
OOTF can also be a positive number γ, which is then passed to power.OOTF() to create the TransferFunction. This is the classical γ power law.
EOTF can also be NULL; see section **Transfer Function Details** for valid combinations.

overwrite
: in installRGB(), space is compared with previously installed RGB space names in case-insensitive way. If there is a match, and overwrite is FALSE, then the installation fails. If overwrite is TRUE, then the existing space is overwritten.

Details

Both installRGB() and uninstallRGB() check for matches with existing names. The matching is full (not partial) and case-insensitive. So it is impossible to have 2 spaces that differ only in case.

Value

installRGB() and uninstallRGB() return TRUE or FALSE.

Primaries and Whitepoint Details

The arguments `scene` and `display` can be a list with 2 items: `primaries` and `white` in that order. There are 3 options for this list, as given in this table:

`dim(primaries)`	`length(white)`	Description
4x2	1	`primaries` is a 4x2 matrix with CIE xy chromaticities of R,G,B,W in the rows
3x2	2	`primaries` is a 3x2 matrix with CIE xy chromaticities of R,G,B in the rows
3x2	3	`primaries` is a 3x2 matrix with CIE xy chromaticities of R,G,B in the rows

If `length(white)` is 1, then `white` is the whitepoint Y. If `length(white)` is 2, then `white` is the whitepoint xy (CIE chromaticity); the whitepoint Y is taken to be 1. If `length(white)` is 3, `white` is the whitepoint XYZ (CIE tristimulus).
The whitepoint is linearly transformed to RGB=(1,1,1). For better numeric compatibility with standards, xy is recommended. For better numeric compatibility with *Lindbloom*, XYZ is recommended. See the **Examples** below.

Transfer Function Details

The 3 transfer functions - `OETF`, `EOTF`, `OOTF` - can be `NULL` (the default) or given. This yields 8 combinations, but only 6 are valid, as given in this table:

`OETF`	`EOTF`	`OOTF`	Description
given	given	given	INVALID
given	given	`OETF*EOTF`	`OOTF` is the composition `OETF` followed by `EOTF`
given	`OETF^-1*OOTF`	given	`EOTF` is the composition `OETF^-1` followed by `OOTF`
`OOTF*EOTF^-1`	given	given	`OETF` is the composition `OOTF` followed by `EOTF^-1`
given	`OETF^-1`	`identity.TF`	`EOTF` is set to `OETF^-1`, and `OOTF` is set to the identity
`EOTF^-1`	given	`identity.TF`	`OETF` is set to `EOTF^-1`, and `OOTF` is set to the identity
`NULL`	`NULL`	given	INVALID
`NULL`	`NULL`	`NULL`	all 3 transfer functions are set to `identity.TF`.

Think of these 3 functions as forming a triangle. If all 3 are given, the transfers may be ambiguous, i.e. the triangle may not commute. If 2 functions are given, the 3rd is computed from those 2. If only 1 function is given, and it is EOTF or OETF, then it makes sense to make the other one the inverse of the given one, so that the OOTF is the identity. If only the OOTF is given, there is no well-defined way to define the other 2. If none are given, as in the last row, this might be useful for testing conversion between RGB and XYZ.

Warning

All the RGB spaces are stored in a dictionary. If `installRGB()` is successful, the installed space is only in the dictionary until the end of the R session. To make it persist, please put the function call in an R script that is executed after the package is loaded.
The dictionary comes with 8 RGB spaces pre-installed.

References

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

See Also

getRGB(), summaryRGB() RGBfromXYZ(), XYZfromRGB(), TransferFunction, power.OETF(), power.EOTF(), power.OOTF()

Examples

```
#  install native RGB space for NEC PA242W display
prim = matrix( c(0.675,0.316, 0.199,0.715, 0.157,0.026), 3, 2, byrow=TRUE )
installRGB( 'PA242W', scene=NULL, display=list(primaries=prim,white=c(0.95047,1,1.08883)), OETF=2 )

#  install a linear version of sRGB  (OETF=1)
prim    = matrix( c(0.64,0.33,  0.30,0.60, 0.15,0.06), 3, 2, byrow=TRUE )
installRGB( 'linear-sRGB', scene=NULL, display=list(prim,c(0.3127,0.3290)), OETF=1 )

# make plot comparing three EOTFs
plot( getRGB('sRGB')$EOTF, col='black' )
plot( getRGB('linear')$EOTF, col='red', add=TRUE )
plot( getRGB('PA242W')$EOTF, col='blue', add=TRUE )

# Install an RGB space named 'HD+2.4', with encoding from BT.709 and display from BT.1886.
# the OOTF for this space is non-trivial
prim    = matrix( c(0.64,0.33,  0.30,0.60,  0.15,0.06 ), 3, 2, byrow=TRUE )
white   = c( 0.3127, 0.3290 )
installRGB( "HD+2.4", scene=NULL, display=list(prim,white),
                          OETF=(BT.709.EOTF)^-1, EOTF=BT.1886.EOTF(), over=TRUE )

# make plot comparing two OOTFs
plot( getRGB('HD+2.4')$OOTF, col='red')
plot( getRGB('sRGB')$OOTF, col='black', add=TRUE  )
```

RGB Space Query *Query RGB Spaces*

Description

Query and summarize the installed RGB spaces. The RGB spaces are stored in a dictionary, which comes with 8 RGB spaces pre-installed.

Usage

```
summaryRGB( verbosity=1 )

getRGB( space )
getWhiteXYZ( space, which='scene' )
```

Arguments

`space`	name of the RGB space to query. The name matching is partial and case-insensitive.
`verbosity`	an integer that controls the return value of `summaryRGB()`, see **Value**.
`which`	the source of the whitepoint, either `'scene'` or `'display'`. Matching is partial and case-insensitive.

Details

The function `getWhiteXYZ()` is provided because some applications only need the whitepoint for chromatic adaptation purposes, and this function is faster than `getRGB()`.

Value

`summaryRGB()`, with the default `verbosity=1`, returns a `data.frame` with a row for each RGB space. The row contains primary, whitepoint, and transfer function information for each space. The primary/whitepoint data is for both scene and display; all the data is numerical and the columns are labeled. There are 22 columns so the display is very wide.
The transfer function data is a very short string. If the OETF is classical (pure 1/γ power law), the string is 1/γ. If the OETF is not classical, the string is 1/~γ, where γ is the best-fit (or approximate or effective) γ to the OETF in the L^1-norm.
Similarly, if the EOTF is classical (pure γ power law) the string is γ, and if the EOTF is not classical the string is ~γ.
The OOTF is the quotient (to 2 decimal places) of the gammas of EOTF and OETF (either true gamma or best-fit gamma). If either gamma is best-fit then the string is preceede by a `'~'`, which means *effective*.
If the `TransferFunction` has dimension 1, but the domain and range are not the interval [0,1], the string is `'1D'`. If the `TransferFunction` has dimension 3, the string is `'3D'`.
If `verbosity=0`, `summaryRGB()` returns the names of all the RGB spaces.

`getRGB()` returns a list with these items:

`space`	the full and original name of the RGB space
`scene`	a list with items `primaries`, `whiteXYZ`, `RGB2XYZ`, and `XYZ2RGB`
`display`	a list with items `primaries`, `whiteXYZ`, `RGB2XYZ`, and `XYZ2RGB`
`EOTF`	Electro-Optical Transfer Function
`OETF`	Opto-Electronic Transfer Function
`OOTF`	Opto-Optical Transfer Function, and numerically equal to `OETF*EOTF`

The items in the lists `scene` and `display` are

`primaries`	4x2 matrix with the xy chromaticities of the RGB primaries and white
`whiteXYZ`	XYZ of the display white point, which maps to RGB=(1,1,1)
`RGB2XYZ`	3x3 matrix taking RGB to XYZ
`XYZ2RGB`	3x3 matrix taking XYZ to RGB

All transfer functions are actual TransferFunctions objects, and not the numerical exponent or character string name. They are suitable for plotting with plot.TransferFunction(); see the **Examples**. In case of error, getRGB() returns NULL.

getWhiteXYZ() returns a numeric 3-vector with the XYZ of the whitepoint of the scene or the display. In case of error it returns NULL.

References

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

See Also

installRGB(), plot.TransferFunction()

Examples

```
# make plot comparing three EOTFs
plot( getRGB('sRGB')$EOTF, col='black' )
plot( getRGB('BT.709')$EOTF, col='blue', add=TRUE )
plot( getRGB('ProPhotoRGB')$EOTF, col='red', add=TRUE )
```

Signal RGB Calculation

Convert XYZ or Linear RGB to Signal RGB

Description

Convert XYZ or Linear RGB to Signal RGB, multiple RGB spaces are available

Usage

```
RGBfromXYZ( XYZ, space='sRGB', which='scene', TF=NULL, maxSignal=1 )

SignalRGBfromLinearRGB( RGB, space='sRGB', which='scene', TF=NULL, maxSignal=1 )
```

Arguments

XYZ	a numeric Nx3 matrix with CIE XYZ coordinates in the rows, or a vector that can be converted to such a matrix, by row. The XYZ are assumed to be viewed under the white-point of the given RGB space.
RGB	a numeric Nx3 matrix with linear RGB coordinates in the rows, or a vector that can be converted to such a matrix, by row. The RGB may be outside the corresponding domain box (either scene or display), see **Details**.
space	the name of an installed RGB space. The name matching is partial and case-insensitive.
which	the input linear RGB - either 'scene' or 'display'.

TF	if not NULL, TF is a TransferFunction that overrides the appropriate transfer function of space. TF can also be a positive number. If TF=1, then TF is set to identity.TF, so the returned RGB values are actually linear, and they are not clamped to the appropriate domain box (see **Value**). If TF!=1 it is used to create either power.EOTF() or power.OETF() as approriate. If TF is not NULL in SignalRGBfromLinearRGB(), then space is ignored.
maxSignal	maximum value of non-linear RGB. Other common values are 100, 255, 1023, 4095, and 65535. Even when 1, they are still taken to be non-linear Signal values.

Value

a data.frame with N rows and these columns

RGB	signal RGB. If TF is not the identity, all input linear RGB values are clamped to the appropriate domain box, which implies that the signal RGBs are inside the cube [0,maxSignal]^3. Values are not rounded.
OutOfGamut	logical vector, TRUE means the input linear RGB was outside the domain box before clamping it.

In case of error, the functions return NULL.

References

Wikipedia. **RGB color space**. https://en.wikipedia.org/wiki/RGB_color_space

See Also

XYZfromRGB(), LinearRGBfromSignalRGB(), installRGB(), identity.TF

Examples

```
RGBfromXYZ( c(80.310897,90.306510,84.613450, 100,100,100)/100, max=255 )
##       RGB.R    RGB.G    RGB.B OutOfGamut
##  1 230.1676 249.4122 225.2472      FALSE
##  2 255.0000 249.1125 244.4704       TRUE
```

Standard Primaries *Standard Primaries*

Description

xy Chromaticities for some standard primary sets. These include Red, Green, Blue, and White.

AP0_PRI	ACES Scene-Referred Primaries, from SMPTE ST2065-1
AP1_PRI	working space and rendering primaries for ACES 1.0
REC709_PRI	Rec.709 (aka BT.709) primaries
REC2020_PRI	Rec.2020 (aka BT.2020) primaries

`P3D65_PRI`	RGB primaries from DCI-P3, with D65 for the whitepoint
`P3D60_PRI`	RGB primaries from DCI-P3, with ACES whitepoint (approximately D60)
`P3DCI_PRI`	RGB primaries from DCI-P3, with DCI whitepoint

Details

All of these are built-in 4x2 matrices, with xy coordinates in the rows, and in RGBW order.

References

ST 2065-1:2012. SMPTE Standard - Academy Color Encoding Specification (ACES). 2013.

SMPTE Standard RP 431-2. D-Cinema Quality - Reference Projector and Environment for the Display of DCDM in Review Rooms and Theaters. 2011.

Wikipedia. **DCI-P3**. https://en.wikipedia.org/wiki/DCI-P3.

BT.709. Parameter values for the HDTV standards for production and international programme exchange. June 2015.

BT.2020. Parameter values for ultra-high definition television systems for production and international programme exchange. October 2015.

Examples

```
AP0_PRI
##         x        y
##  R 0.73470  0.26530
##  G 0.00000  1.00000
##  B 0.00010 -0.07700
##  W 0.32168  0.33767
```

`transfer` *Apply TransferFunction to a Vector or an Array*

Description

The function `transfer()` applies the given `TransferFunction` to the given vector or array `x` and returns a numeric object of the same dimensions.

Usage

```
## S3 method for class 'TransferFunction'
transfer( TF, x, domaincheck=TRUE )
```

Arguments

`TF`	a `TransferFunction` object, with dimension N
`x`	a numeric vector or array. If N≥2 then `x` must be an MxN matrix, or a vector that can be converted to such a matrix, by row.
`domaincheck`	check whether numbers or rows of `x` are in the domain box of `TF` before application

Value

Let N := `dimension(TF)`.
If N=1 then `x` can have any length or dimension; the function is applied to each number in `x` in a vectorized way, and the returned object is then assigned the same dimensions as `x`. If `x` is a matrix then the returned object is assigned the same `rownames`. If a number is `NA` then the returned number is also `NA`.
If N≥2 and `x` is an MxN matrix, then the function is applied to each row of `x` individually and the returned object is a matrix with the same dimensions and `rownames` as `x`. If any number in a row is `NA` then the returned row is all `NA`s.
If `TF` is a universal identity (e.g. `identity.TF`), the function returns `x` with no checking.
In case of a global error (e.g. dimension mismatch) the function returns `NULL`.

See Also

`TransferFunction`, `identity.TF`

`TransferFunction` *Constructing and Testing TransferFunction Objects*

Description

The function `TransferFunction()` is the constructor for **TransferFunction** objects.

`is.TransferFunction()` tests whether an object is a valid **TransferFunction** object.
`as.TransferFunction()` converts other variables to a **TransferFunction** object, and is designed to be overridden by other packages.

Usage

```
TransferFunction( fun, funinv, domain, range, id=NULL )

is.TransferFunction(x)

## Default S3 method:
as.TransferFunction( ... )
```

Arguments

`fun`	a function that accepts a numeric argument, and returns one of the same length. The dimension of `fun` is determined by the arguments `domain` and `range`. The function must be *injective* and this is checked if the function is univariate. The requirements for univariate and multivariate functions are very different, see **Details**.
`funinv`	a function that the inverse for `fun`. If `fun` is univariate and `funinv=NULL`, then an approximation for the inverse is computed using `stats::splinefun()`. If `fun` is multivariate and `funinv=NULL`, then it is an ERROR.

domain	a 2xN matrix, or a numeric vector that can be converted to such a matrix, by column. In each column, the entry in row 1 must be strictly less than the entry in row 2. The columns of domain define N intervals whose product is a box in R^N that is the domain of fun. The box must be finite. If N=1 then the box is just an interval, and fun is univariate. Otherwise, fun is multivariate with dimension N.
range	a 2xN matrix, or a numeric vector that can be converted to such a matrix, by column. The N here must be equal to the N for domain. The matrix defines a box that encloses the image of domain under fun. The box must be finite.
id	a character string that is helpful when printing the object, and in logging messages. If id=NULL then an appropriate string is created from the function call.
x	an R object to test for being a valid **TransferFunction** object.
...	arguments for use in other packages.

Details

If fun is univariate, then it must be able to accept a numeric vector of any length, and apply the function to each number in the vector; i.e. fun must be vectorized. If a number in the vector is NA, then the function must silently return NA for that number; usually this is not a problem. The function is *not* required to test whether the number is in the domain interval; this is handled by the TransferFunction code.
If fun is multivariate with dimension N, then it must be able to accept a vector of length N and return a vector of length N. It is *not* required to accept an MxN matrix. It is *not* required to test whether the vector is in the domain box.
The function funinv has the same requirements as fun.

Value

TransferFunction() returns a **TransferFunction** object, or NULL in case of ERROR.

is.TransferFunction() returns TRUE or FALSE. It only checks the class, using base::inherits().

as.TransferFunction.default() issues an ERROR message and returns NULL..

See Also

dimension(), composition(), is.invertible(), metadata(), inverse(), transfer(), orientation(), validate(), print.TransferFunction(), plot.TransferFunction()

Examples

```
#  make a test TransferFunction

myfun = function(x) {x*x}

test  = TransferFunction( myfun, sqrt, domain=c(0,3), range=c(0,9), id='test.TF' )

#  print it
test
#--------------------   test.TF    ---------------------#
```

```
## test.TF is a univariate TransferFunction.
## domain:      [0,3]  (x)
## range:       [0,9]  (y)
## invertible:  Yes
## orientation: preserving
## range-test points = 1300, max(distance)=0.
## validation:  Passed

# and now plot it
plot( test )
```

---

validate *Validate a TransferFunction by applying some simple Tests*

Description

Each `TransferFunction` object is actually a list of so-called *elementary* transfer functions; for details on this see `composition()`. This `validate()` applies an internal `validate()` function to each elementary function individually. The internal `validate()` function generates some points in the domain of the function and checks that all points are transfered into the range of the function. If the function is also invertible, it checks that the inverse transfers back to the original point.

Usage

```
## S3 method for class 'TransferFunction'
validate( TF, points=1300, tol=5.e-7, domain=NULL )
```

Arguments

`TF`	a `TransferFunction` object with dimension N, and consisting of M elementary transfer functions
`points`	the number of points to test, in each elementary function
`tol`	the numerical tolerance for the inversion test - this is relative to the length of the corresponding side of the domain box
`domain`	a 2xN matrix to use as an alternate domain, for the first elementary function in the list only. `domain` can also be a vector of length 2, which is then replicated to a 2xN matrix.

Value

The function returns a logical vector of length M. The value of the i'th element is the validation status of the i'th elementary function. The returned vector has the attribute `'message'` which is a list of length M with explanatory text. For nicely formatted text see `print()`.

See Also

`TransferFunction`, `identity.TF`, `composition()`, `print.TransferFunction()`

Index