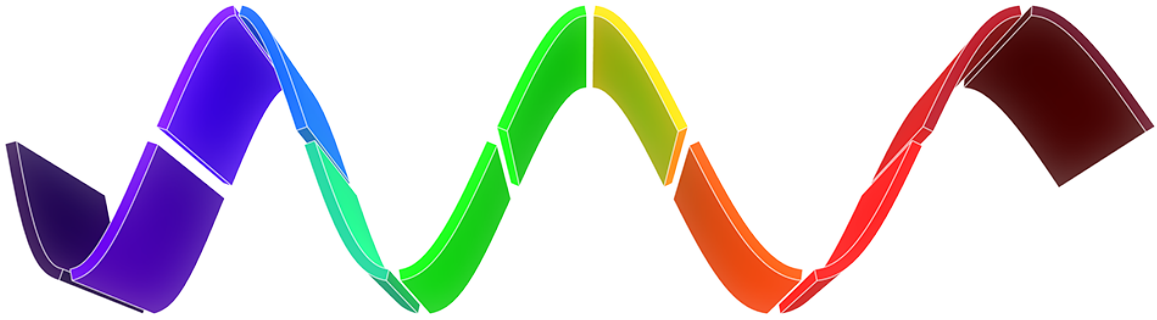




Colour Documentation

Release 0.3.13

Colour Developers

Apr 26, 2019

Contents

1	1.1	Draft Release Notes	3
2	1.2	Features	5
3	1.3	Online	7
4	1.4	Installation	9
5	1.5	Usage	11
5.1	Colour Manual		11
5.1.1	Tutorial		11
5.1.1.1	Overview		12
5.1.1.2	From Spectral Distribution		19
5.1.1.3	Convert to Tristimulus Values		29
5.1.1.4	From <i>CIE XYZ</i> Colourspace		29
5.1.1.5	Convert to Display Colours		30
5.1.1.6	Generate Colour Rendition Charts		31
5.1.1.7	Convert to Chromaticity Coordinates		33
5.1.1.8	And More...		34
5.1.2	Basics		35
5.1.2.1	N-Dimensional Arrays Support		35
5.1.2.2	Domain-Range Scales		36
5.1.3	Reference		39
5.1.3.1	Colour		39
5.1.3.2	Indices and tables		660
5.1.4	Bibliography		660
5.1.4.1	Indirect References		660
5.2	1.5.1 Examples		661
5.2.1	1.5.1.1 Chromatic Adaptation - <code>colour.adaptation</code>		661
5.2.2	1.5.1.2 Algebra - <code>colour.algebra</code>		662
5.2.2.1	1.5.1.2.1 Kernel Interpolation		662
5.2.2.2	1.5.1.2.2 Sprague (1880) Interpolation		662
5.2.3	1.5.1.3 Colour Appearance Models - <code>colour.appearance</code>		662
5.2.4	1.5.1.4 Colour Blindness - <code>colour.blindness</code>		662
5.2.5	1.5.1.5 Colour Correction - <code>colour.characterisation</code>		663
5.2.6	1.5.1.6 Colorimetry - <code>colour.colorimetry</code>		663
5.2.6.1	1.5.1.6.1 Spectral Computations		663

5.2.6.2	1.5.1.6.2	Multi-Spectral Computations	663
5.2.6.3	1.5.1.6.3	Blackbody Spectral Radiance Computation	664
5.2.6.4	1.5.1.6.4	Dominant, Complementary Wavelength & Colour Purity Computation	664
5.2.6.5	1.5.1.6.5	Lightness Computation	664
5.2.6.6	1.5.1.6.6	Luminance Computation	664
5.2.6.7	1.5.1.6.7	Whiteness Computation	665
5.2.6.8	1.5.1.6.8	Yellowness Computation	665
5.2.6.9	1.5.1.6.9	Luminous Flux, Efficiency & Efficacy Computation	665
5.2.7	1.5.1.7	Contrast Sensitivity Function - colour.contrast	665
5.2.8	1.5.1.8	Colour Difference - colour.difference	665
5.2.9	1.5.1.9	IO - colour.io	666
5.2.9.1	1.5.1.9.1	Images	666
5.2.9.2	1.5.1.9.2	Look Up Table (LUT) Data	666
5.2.10	1.5.1.10	Colour Models - colour.models	666
5.2.10.1	1.5.1.10.1	CIE xyY Colourspace	666
5.2.10.2	1.5.1.10.2	CIE L*a*b* Colourspace	666
5.2.10.3	1.5.1.10.3	CIE L*u*v* Colourspace	667
5.2.10.4	1.5.1.10.4	CIE 1960 UCS Colourspace	667
5.2.10.5	1.5.1.10.5	CIE 1964 U*V*W* Colourspace	667
5.2.10.6	1.5.1.10.6	Hunter L,a,b Colour Scale	667
5.2.10.7	1.5.1.10.7	Hunter Rd,a,b Colour Scale	667
5.2.10.8	1.5.1.10.8	CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)	667
5.2.10.9	1.5.1.10.9	CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)	668
5.2.10.10	1.5.1.10.10	IPT Colourspace	668
5.2.10.11	1.5.1.10.11	DIN99 Colourspace	668
5.2.10.12	1.5.1.10.12	hdr-CIELAB Colourspace	668
5.2.10.13	1.5.1.10.13	hdr-IPT Colourspace	668
5.2.10.14	1.5.1.10.14	OSA UCS Colourspace	668
5.2.10.15	1.5.1.10.15	JzAzBz Colourspace	668
5.2.10.16	1.5.1.10.16	YCbCr Colour Encoding	669
5.2.10.17	1.5.1.10.17	YCoCg Colour Encoding	669
5.2.10.18	1.5.1.10.18	ICTCP Colour Encoding	669
5.2.10.19	1.5.1.10.19	HSV Colourspace	669
5.2.10.20	1.5.1.10.20	Prismatic Colourspace	669
5.2.10.21	1.5.1.10.21	RGB Colourspace and Transformations	669
5.2.10.22	1.5.1.10.22	RGB Colourspace Derivation	669
5.2.10.23	1.5.1.10.23	RGB Colourspaces	670
5.2.10.24	1.5.1.10.24	OETFs	671
5.2.10.25	1.5.1.10.25	OETFs Reverse	671
5.2.10.26	1.5.1.10.26	EOTFs	671
5.2.10.27	1.5.1.10.27	EOTFs Reverse	672
5.2.10.28	1.5.1.10.28	OOTFs	672
5.2.10.29	1.5.1.10.29	OOTFs Reverse	672
5.2.10.30	1.5.1.10.30	Log Encoding / Decoding Curves	672
5.2.11	1.5.1.11	Colour Notation Systems - colour.notation	672
5.2.11.1	1.5.1.11.1	Munsell Value	672
5.2.11.2	1.5.1.11.2	Munsell Colour	673
5.2.12	1.5.1.12	Optical Phenomena - colour.phenomena	673
5.2.13	1.5.1.13	Light Quality - colour.quality	673
5.2.13.1	1.5.1.13.1	Colour Rendering Index	673
5.2.13.2	1.5.1.13.2	Colour Quality Scale	673

5.2.14	1.5.1.14	Spectral Up-sampling & Reflectance Recovery - colour.recovery	674
5.2.15	1.5.1.15	Correlated Colour Temperature Computation Methods - colour.temperature	674
5.2.16	1.5.1.16	Colour Volume - colour.volume	674
5.2.17	1.5.1.17	Plotting - colour.plotting	674
5.2.17.1	1.5.1.17.1	Visible Spectrum	675
5.2.17.2	1.5.1.17.2	Spectral Distribution	675
5.2.17.3	1.5.1.17.3	Blackbody	676
5.2.17.4	1.5.1.17.4	Colour Matching Functions	676
5.2.17.5	1.5.1.17.5	Luminous Efficiency	677
5.2.17.6	1.5.1.17.6	Colour Checker	678
5.2.17.7	1.5.1.17.7	Chromaticities Prediction	679
5.2.17.8	1.5.1.17.8	Colour Temperature	680
5.2.17.9	1.5.1.17.9	Chromaticities	681
5.2.17.10	1.5.1.17.10	Colour Rendering Index	682
6	1.6	Contributing	685
7	1.7	Changes	687
8	1.8	Bibliography	689
9	1.9	See Also	691
10	1.10	About	693
		Bibliography	695

Colour is a **Python** colour science package implementing a comprehensive number of colour theory transformations and algorithms.

It is open source and freely available under the [New BSD License](#) terms.

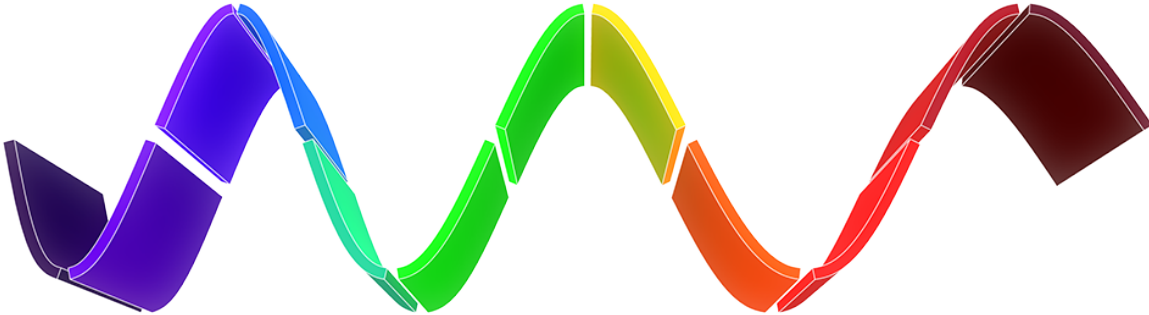


Table of Contents

- 1 *Colour Science for Python*
 - 1.1 *Draft Release Notes*
 - 1.2 *Features*
 - 1.3 *Online*
 - 1.4 *Installation*
 - 1.5 *Usage*
 - * 1.5.1 *Examples*
 - 1.6 *Contributing*
 - 1.7 *Changes*
 - 1.8 *Bibliography*
 - 1.9 *See Also*
 - 1.10 *About*

CHAPTER 1

1.1 Draft Release Notes

The draft release notes from the `develop` branch are available at this [url](#).

CHAPTER 2

1.2 Features

Colour features a rich dataset and collection of objects, please see the [features](#) page for more information.

CHAPTER 3

1.3 Online

Colour can be used online with [Google Colab](#).

CHAPTER 4

1.4 Installation

[Anaconda](#) from *Continuum Analytics* is the Python distribution we use to develop **Colour**: it ships all the scientific dependencies we require and is easily deployed cross-platform:

```
$ conda create -y -n python-colour
$ source activate python-colour
$ conda install -y -c conda-forge colour-science
```

Colour can be easily installed from the [Python Package Index](#) by issuing this command in a shell:

```
$ pip install colour-science
```

The detailed installation procedure is described in the [Installation Guide](#).

The two main references for **Colour** usage are the [Colour Manual](#) and the [Jupyter Notebooks](#) with detailed historical and theoretical context and images.

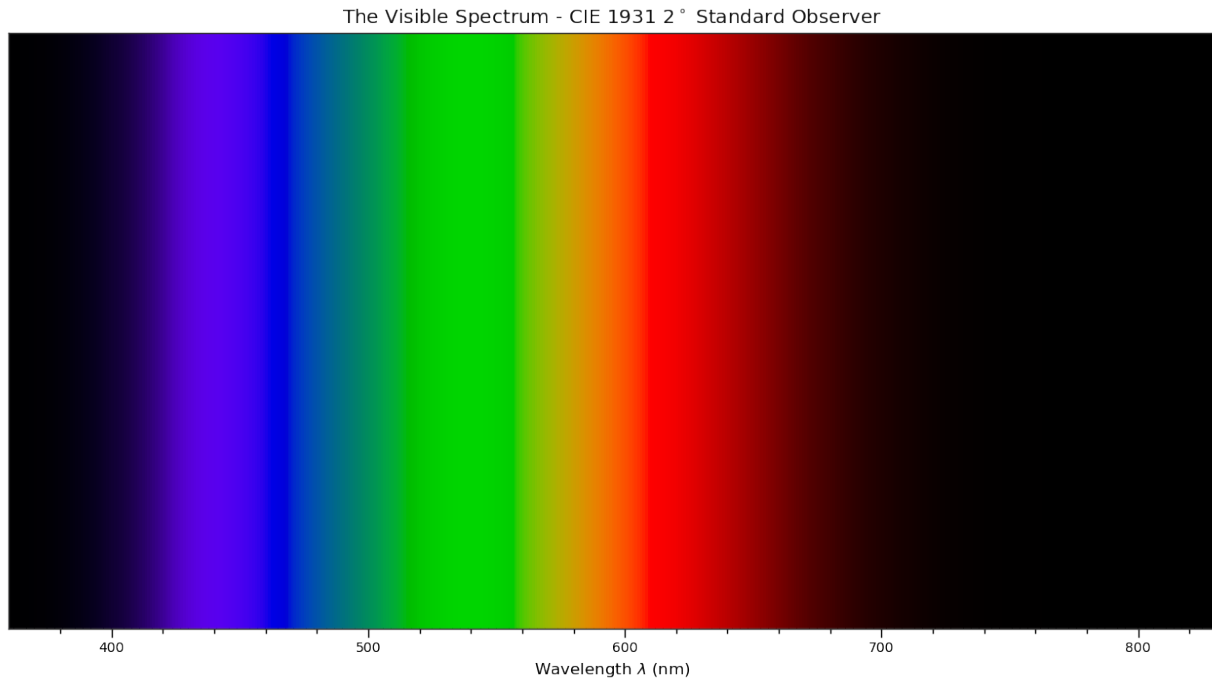
5.1 Colour Manual

5.1.1 Tutorial

[Colour](#) spreads over various domains of Colour Science, from colour models to optical phenomena, this tutorial does not give a complete overview of the API but is a good introduction to the main concepts.

Note: A directory with examples is available at this path in **Colour** installation: *colour/examples*. It can also be explored directly on [Github](#).

```
from colour.plotting import *  
  
colour_style()  
  
plot_visible_spectrum()
```



5.1.1.1 Overview

Colour is organised around various sub-packages:

- *adaptation*: Chromatic adaptation models and transformations.
- *algebra*: Algebra utilities.
- *appearance*: Colour appearance models.
- *biochemistry*: Biochemistry computations.
- *blindness*: Colour vision deficiency models.
- *continuous*: Base objects for continuous data representation.
- *contrast*: Objects for contrast sensitivity computation.
- *characterisation*: Colour fitting and camera characterisation.
- *colorimetry*: Core objects for colour computations.
- *constants*: CIE and CODATA constants.
- *corresponding*: Corresponding colour chromaticities computations.
- *difference*: Colour difference computations.
- *examples*: Examples for the sub-packages.
- *io*: Input / output objects for reading and writing data.
- *models*: Colour models.
- *notation*: Colour notation systems.
- *phenomena*: Computation of various optical phenomena.
- *plotting*: Diagrams, figures, etc. . .

- *quality*: Colour quality computation.
- *recovery*: Reflectance recovery.
- *temperature*: Colour temperature and correlated colour temperature computation.
- *utilities*: Various utilities and data structures.
- *volume*: Colourspace volumes computation and optimal colour stimuli.

Most of the public API is available from the root colour namespace:

```
import colour

print(colour.__all__[:5] + ['...'])
```

```
['domain_range_scale', 'get_domain_range_scale', 'set_domain_range_scale', 'CHROMATIC_ADAPTATION_METHODS',
↪, 'CHROMATIC_ADAPTATION_TRANSFORMS', '...']
```

The various sub-packages also expose their public API:

```
from pprint import pprint

import colour.plotting

for sub_package in ('adaptation', 'algebra', 'appearance', 'biochemistry',
                    'blindness', 'characterisation', 'colorimetry',
                    'constants', 'continuous', 'contrast', 'corresponding',
                    'difference', 'io', 'models', 'notation', 'phenomena',
                    'plotting', 'quality', 'recovery', 'temperature',
                    'utilities', 'volume'):
    print(sub_package.title())
    pprint(getattr(colour, sub_package).__all__[:5] + ['...'])
    print('\n')
```

```
Adaptation
['CHROMATIC_ADAPTATION_TRANSFORMS',
 'XYZ_SCALING_CAT',
 'VON_KRIES_CAT',
 'BRADFORD_CAT',
 'SHARP_CAT',
 '...']
```

```
Algebra
['cartesian_to_spherical',
 'spherical_to_cartesian',
 'cartesian_to_polar',
 'polar_to_cartesian',
 'cartesian_to_cylindrical',
 '...']
```

```
Appearance
['Hunt_InductionFactors',
 'HUNT_VIEWING_CONDITIONS',
 'Hunt_Specification',
 'XYZ_to_Hunt',
 'ATD95_Specification',
```

(continues on next page)

(continued from previous page)

```

'...']

Biochemistry
['reaction_rate_MichealisMenten',
 'substrate_concentration_MichealisMenten',
 '...']

Blindness
['CVD_MATRICES_MACHADO2010',
 'anomalous_trichromacy_cmfs_Machado2009',
 'anomalous_trichromacy_matrix_Machado2009',
 'cvd_matrix_Machado2009',
 '...']

Characterisation
['RGB_SpectralSensitivities',
 'RGB_DisplayPrimaries',
 'CAMERAS_RGB_SPECTRAL_SENSITIVITIES',
 'COLOURCHECKERS',
 'ColourChecker',
 '...']

Colorimetry
['SpectralShape',
 'DEFAULT_SPECTRAL_SHAPE',
 'SpectralDistribution',
 'MultiSpectralDistribution',
 'sd_blackbody',
 '...']

Constants
['K_M', 'KP_M', 'AVOGADRO_CONSTANT', 'BOLTZMANN_CONSTANT', 'LIGHT_SPEED', '...']

Continuous
['AbstractContinuousFunction', 'Signal', 'MultiSignal', '...']

Contrast
['optical_MTF_Barten1999',
 'pupil_diameter_Barten1999',
 'sigma_Barten1999',
 'retinal_illuminance_Barten1999',
 'maximum_angular_size_Barten1999',
 '...']

Corresponding
['BRENEMAN_EXPERIMENTS',
 'BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES',
 'corresponding_chromaticities_prediction_CIE1994',

```

(continues on next page)

(continued from previous page)

```
'corresponding_chromaticities_prediction_CMCCAT2000',
'corresponding_chromaticities_prediction_Fairchild1990',
'...']
```

Difference

```
['delta_E_CAM02LCD',
'delta_E_CAM02SCD',
'delta_E_CAM02UCS',
'delta_E_CAM16LCD',
'delta_E_CAM16SCD',
'...']
```

Io

```
['SpectralDistribution_IESTM2714',
'AbstractLUTSequenceOperator',
'LUT1D',
'LUT3x1D',
'LUT3D',
'...']
```

Models

```
['JmH_CIECAM02_to_CAM02LCD',
'CAM02LCD_to_JmH_CIECAM02',
'JmH_CIECAM02_to_CAM02SCD',
'CAM02SCD_to_JmH_CIECAM02',
'JmH_CIECAM02_to_CAM02UCS',
'...']
```

Notation

```
['MUNSELL_COLOURS_ALL',
'MUNSELL_COLOURS_1929',
'MUNSELL_COLOURS_REAL',
'MUNSELL_COLOURS',
'munsell_value',
'...']
```

Phenomena

```
['scattering_cross_section',
'rayleigh_optical_depth',
'rayleigh_scattering',
'sd_rayleigh_scattering',
'...']
```

Plotting

```
['ASTM_G_173_ETR',
'ASTM_G_173_GLOBAL_TILT',
'ASTM_G_173_DIRECT_CIRCUMSOLAR',
'COLOUR_STYLE_CONSTANTS',
'COLOUR_ARROW_STYLE',
'...']
```

(continues on next page)

(continued from previous page)

```

Quality
['TCS_SDS',
 'VS_SDS',
 'CRI_Specification',
 'colour_rendering_index',
 'CQS_Specification',
 '...']

```

```

Recovery
['SMITS_1999_SDS',
 'XYZ_to_sd_Meng2015',
 'RGB_to_sd_Smits1999',
 'XYZ_TO_SD_METHODS',
 'XYZ_to_sd',
 '...']

```

```

Temperature
['CCT_TO_UV_METHODS',
 'UV_TO_CCT_METHODS',
 'CCT_to_uv',
 'CCT_to_uv_Ohno2013',
 'CCT_to_uv_Robertson1968',
 '...']

```

```

Utilities
['Lookup',
 'Structure',
 'CaseInsensitiveMapping',
 'handle_numpy_errors',
 'ignore_numpy_errors',
 '...']

```

```

Volume
['ILLUMINANTS_OPTIMAL_COLOUR_STIMULI',
 'is_within_macadam_limits',
 'is_within_mesh_volume',
 'is_within_pointer_gamut',
 'generate_pulse_waves',
 '...']

```

The codebase is documented and most docstrings have usage examples:

```
print(colour.temperature.CCT_to_uv_Ohno2013.__doc__)
```

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature :math:T_{cp}, :math:\Delta_{uv} and colour matching functions using *Ohno (2013)* method.

Parameters

(continues on next page)

(continued from previous page)

```

CCT : numeric
    Correlated colour temperature :math:`T_{cp}`.
D_uv : numeric, optional
    :math:`\Delta_{uv}`.
cmfs : XYZ_ColourMatchingFunctions, optional
    Standard observer colour matching functions.

Returns
-----
ndarray
    *CIE UCS* colourspace *uv* chromaticity coordinates.

References
-----
:cite:`Ohno2014a`

Examples
-----
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> CCT = 6507.4342201047066
>>> D_uv = 0.003223690901513
>>> CCT_to_uv_Ohno2013(CCT, D_uv, cmfs) # doctest: +ELLIPSIS
array([ 0.1977999...,  0.3122004...])

```

At the core of **Colour** is the `colour.colorimetry` sub-package, it defines the objects needed for spectral computations and many others:

```

import colour.colorimetry as colorimetry

pprint(colorimetry.__all__)

```

```

['SpectralShape',
 'DEFAULT_SPECTRAL_SHAPE',
 'SpectralDistribution',
 'MultiSpectralDistribution',
 'sd_blackbody',
 'blackbody_spectral_radiance',
 'planck_law',
 'LMS_ConeFundamentals',
 'RGB_ColourMatchingFunctions',
 'XYZ_ColourMatchingFunctions',
 'CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_SDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_SDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS',
 'sd_constant',

```

(continues on next page)

(continued from previous page)

```

'sd_zeros',
'sd_ones',
'SD_GAUSSIAN_METHODS',
'sd_gaussian',
'sd_gaussian_normal',
'sd_gaussian_fwhm',
'SD_SINGLE_LED_METHODS',
'sd_single_led',
'sd_single_led_Ohno2005',
'SD_MULTI_LEDS_METHODS',
'sd_multi_leds',
'sd_multi_leds_Ohno2005',
'SD_TO_XYZ_METHODS',
'MULTI_SD_TO_XYZ_METHODS',
'sd_to_XYZ',
'multi_sds_to_XYZ',
'ASTME30815_PRACTISE_SHAPE',
'lagrange_coefficients_ASTME202211',
'trstimulus_weighting_factors_ASTME202211',
'adjust_trstimulus_weighting_factors_ASTME30815',
'sd_to_XYZ_integration',
'sd_to_XYZ_trstimulus_weighting_factors_ASTME30815',
'sd_to_XYZ_ASTME30815',
'multi_sds_to_XYZ_integration',
'wavelength_to_XYZ',
'BANDPASS_CORRECTION_METHODS',
'bandpass_correction',
'bandpass_correction_Stearns1988',
'sd_CIE_standard_illuminant_A',
'sd_CIE_illuminant_D_series',
'daylight_locus_function',
'sd_mesopic_luminous_efficiency_function',
'mesopic_weighting_function',
'LIGHTNESS_METHODS',
'lightness',
'lightness_Glasser1958',
'lightness_Wyszecki1963',
'lightness_CIE1976',
'lightness_Fairchild2010',
'lightness_Fairchild2011',
'intermediate_lightness_function_CIE1976',
'LUMINANCE_METHODS',
'luminance',
'luminance_Newhall1943',
'luminance_ASTMD153508',
'luminance_CIE1976',
'luminance_Fairchild2010',
'luminance_Fairchild2011',
'intermediate_luminance_function_CIE1976',
'dominant_wavelength',
'complementary_wavelength',
'excitation_purity',
'colorimetric_purity',
'luminous_flux',
'luminous_efficiency',
'luminous_efficacy',

```

(continues on next page)

(continued from previous page)

```
'RGB_10_degree_cmfs_to_LMS_10_degree_cmfs',
'RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs',
'RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs',
'LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs',
'LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs',
'WHITENESS_METHODS',
'whiteness',
'whiteness_Berger1959',
'whiteness-Taube1960',
'whiteness_Stensby1968',
'whiteness_ASTME313',
'whiteness_Ganz1979',
'whiteness_CIE2004',
'YELLOWNESS_METHODS',
'yellowness',
'yellowness_ASTMD1925',
'yellowness_ASTME313']
```

Colour computations leverage a comprehensive dataset available in most sub-packages, for example the `colour.colorimetry.dataset` defines the following components:

```
import colour.colorimetry.dataset as dataset

pprint(dataset.__all__)
```

```
['CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_SDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_SDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS']
```

5.1.1.2 From Spectral Distribution

Whether it be a sample spectral distribution, colour matching functions or illuminants, spectral data is manipulated using an object built with the `colour.SpectralDistribution` class or based on it:

```
# Defining a sample spectral distribution data.
sample_sd_data = {
    380: 0.048,
    385: 0.051,
    390: 0.055,
    395: 0.060,
    400: 0.065,
    405: 0.068,
    410: 0.068,
    415: 0.067,
```

(continues on next page)

(continued from previous page)

420: 0.064,
 425: 0.062,
 430: 0.059,
 435: 0.057,
 440: 0.055,
 445: 0.054,
 450: 0.053,
 455: 0.053,
 460: 0.052,
 465: 0.052,
 470: 0.052,
 475: 0.053,
 480: 0.054,
 485: 0.055,
 490: 0.057,
 495: 0.059,
 500: 0.061,
 505: 0.062,
 510: 0.065,
 515: 0.067,
 520: 0.070,
 525: 0.072,
 530: 0.074,
 535: 0.075,
 540: 0.076,
 545: 0.078,
 550: 0.079,
 555: 0.082,
 560: 0.087,
 565: 0.092,
 570: 0.100,
 575: 0.107,
 580: 0.115,
 585: 0.122,
 590: 0.129,
 595: 0.134,
 600: 0.138,
 605: 0.142,
 610: 0.146,
 615: 0.150,
 620: 0.154,
 625: 0.158,
 630: 0.163,
 635: 0.167,
 640: 0.173,
 645: 0.180,
 650: 0.188,
 655: 0.196,
 660: 0.204,
 665: 0.213,
 670: 0.222,
 675: 0.231,
 680: 0.242,
 685: 0.251,
 690: 0.261,
 695: 0.271,

(continues on next page)

(continued from previous page)

```

700: 0.282,
705: 0.294,
710: 0.305,
715: 0.318,
720: 0.334,
725: 0.354,
730: 0.372,
735: 0.392,
740: 0.409,
745: 0.420,
750: 0.436,
755: 0.450,
760: 0.462,
765: 0.465,
770: 0.448,
775: 0.432,
780: 0.421}

```

```

sd = colour.SpectralDistribution(sample_sd_data, name='Sample')
print(repr(sd))

```

```

SpectralDistribution([[ 3.80000000e+02,  4.80000000e-02],
 [ 3.85000000e+02,  5.10000000e-02],
 [ 3.90000000e+02,  5.50000000e-02],
 [ 3.95000000e+02,  6.00000000e-02],
 [ 4.00000000e+02,  6.50000000e-02],
 [ 4.05000000e+02,  6.80000000e-02],
 [ 4.10000000e+02,  6.80000000e-02],
 [ 4.15000000e+02,  6.70000000e-02],
 [ 4.20000000e+02,  6.40000000e-02],
 [ 4.25000000e+02,  6.20000000e-02],
 [ 4.30000000e+02,  5.90000000e-02],
 [ 4.35000000e+02,  5.70000000e-02],
 [ 4.40000000e+02,  5.50000000e-02],
 [ 4.45000000e+02,  5.40000000e-02],
 [ 4.50000000e+02,  5.30000000e-02],
 [ 4.55000000e+02,  5.30000000e-02],
 [ 4.60000000e+02,  5.20000000e-02],
 [ 4.65000000e+02,  5.20000000e-02],
 [ 4.70000000e+02,  5.20000000e-02],
 [ 4.75000000e+02,  5.30000000e-02],
 [ 4.80000000e+02,  5.40000000e-02],
 [ 4.85000000e+02,  5.50000000e-02],
 [ 4.90000000e+02,  5.70000000e-02],
 [ 4.95000000e+02,  5.90000000e-02],
 [ 5.00000000e+02,  6.10000000e-02],
 [ 5.05000000e+02,  6.20000000e-02],
 [ 5.10000000e+02,  6.50000000e-02],
 [ 5.15000000e+02,  6.70000000e-02],
 [ 5.20000000e+02,  7.00000000e-02],
 [ 5.25000000e+02,  7.20000000e-02],
 [ 5.30000000e+02,  7.40000000e-02],
 [ 5.35000000e+02,  7.50000000e-02],
 [ 5.40000000e+02,  7.60000000e-02],
 [ 5.45000000e+02,  7.80000000e-02],
 [ 5.50000000e+02,  7.90000000e-02],

```

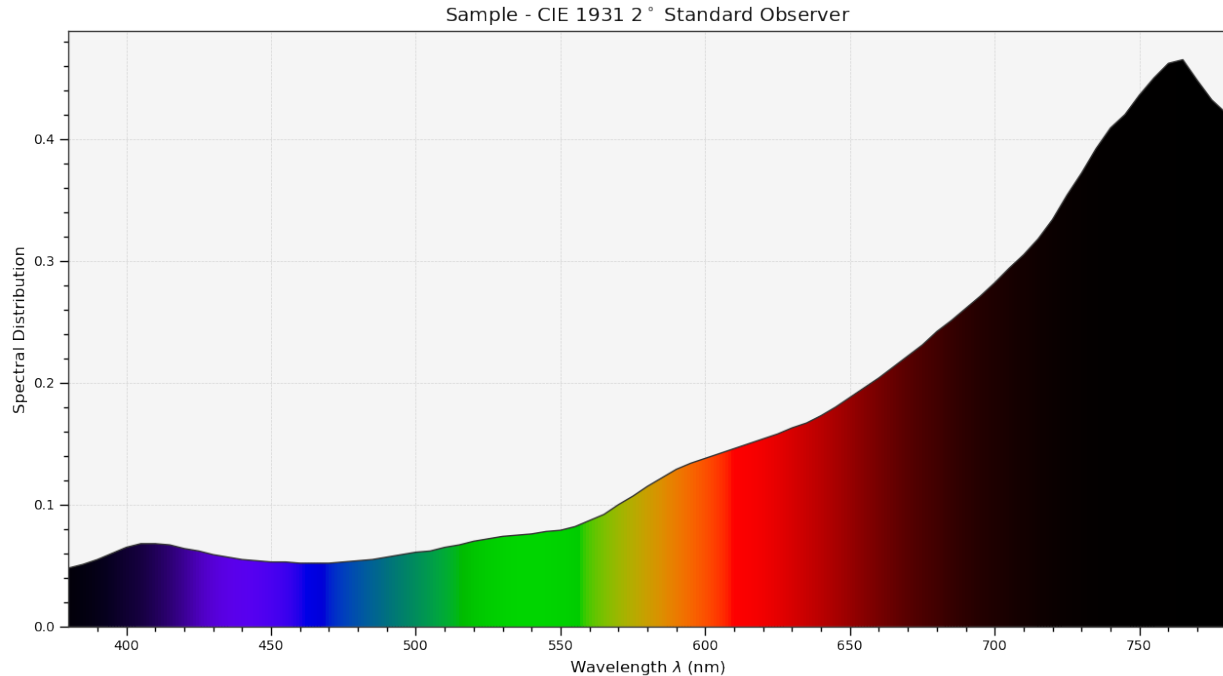
(continues on next page)

(continued from previous page)

```
[ 5.55000000e+02, 8.20000000e-02],
[ 5.60000000e+02, 8.70000000e-02],
[ 5.65000000e+02, 9.20000000e-02],
[ 5.70000000e+02, 1.00000000e-01],
[ 5.75000000e+02, 1.07000000e-01],
[ 5.80000000e+02, 1.15000000e-01],
[ 5.85000000e+02, 1.22000000e-01],
[ 5.90000000e+02, 1.29000000e-01],
[ 5.95000000e+02, 1.34000000e-01],
[ 6.00000000e+02, 1.38000000e-01],
[ 6.05000000e+02, 1.42000000e-01],
[ 6.10000000e+02, 1.46000000e-01],
[ 6.15000000e+02, 1.50000000e-01],
[ 6.20000000e+02, 1.54000000e-01],
[ 6.25000000e+02, 1.58000000e-01],
[ 6.30000000e+02, 1.63000000e-01],
[ 6.35000000e+02, 1.67000000e-01],
[ 6.40000000e+02, 1.73000000e-01],
[ 6.45000000e+02, 1.80000000e-01],
[ 6.50000000e+02, 1.88000000e-01],
[ 6.55000000e+02, 1.96000000e-01],
[ 6.60000000e+02, 2.04000000e-01],
[ 6.65000000e+02, 2.13000000e-01],
[ 6.70000000e+02, 2.22000000e-01],
[ 6.75000000e+02, 2.31000000e-01],
[ 6.80000000e+02, 2.42000000e-01],
[ 6.85000000e+02, 2.51000000e-01],
[ 6.90000000e+02, 2.61000000e-01],
[ 6.95000000e+02, 2.71000000e-01],
[ 7.00000000e+02, 2.82000000e-01],
[ 7.05000000e+02, 2.94000000e-01],
[ 7.10000000e+02, 3.05000000e-01],
[ 7.15000000e+02, 3.18000000e-01],
[ 7.20000000e+02, 3.34000000e-01],
[ 7.25000000e+02, 3.54000000e-01],
[ 7.30000000e+02, 3.72000000e-01],
[ 7.35000000e+02, 3.92000000e-01],
[ 7.40000000e+02, 4.09000000e-01],
[ 7.45000000e+02, 4.20000000e-01],
[ 7.50000000e+02, 4.36000000e-01],
[ 7.55000000e+02, 4.50000000e-01],
[ 7.60000000e+02, 4.62000000e-01],
[ 7.65000000e+02, 4.65000000e-01],
[ 7.70000000e+02, 4.48000000e-01],
[ 7.75000000e+02, 4.32000000e-01],
[ 7.80000000e+02, 4.21000000e-01]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={u'right': None, u'method': u'Constant', u'left': None})
```

The sample spectral distribution can be easily plotted against the visible spectrum:

```
# Plotting the sample spectral distribution.
plot_single_sd(sd)
```

With the sample spectral distribution defined, its shape is retrieved as follows:

```
# Displaying the sample spectral distribution shape.
print(sd.shape)
```

```
(380.0, 780.0, 5.0)
```

The returned shape is an instance of the `colour.SpectralShape` class:

```
repr(sd.shape)
```

```
'SpectralShape(380.0, 780.0, 5.0)'
```

The `colour.SpectralShape` class is used throughout **Colour** to define spectral dimensions and is instantiated as follows:

```
# Using *colour.SpectralShape* with iteration.
shape = colour.SpectralShape(start=0, end=10, interval=1)
for wavelength in shape:
    print(wavelength)

# *colour.SpectralShape.range* method is providing the complete range of values.
shape = colour.SpectralShape(0, 10, 0.5)
shape.range()
```

```
0.0
1.0
2.0
3.0
4.0
5.0
6.0
```

(continues on next page)

(continued from previous page)

```
7.0
8.0
9.0
10.0
```

```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,
        4.5,  5. ,  5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,
        9. ,  9.5, 10. ])
```

Colour defines three convenient objects to create constant spectral distributions:

- `colour.sd_constant`
- `colour.sd_zeros`
- `colour.sd_ones`

```
# Defining a constant spectral distribution.
sd_constant = colour.sd_constant(100)
print("Constant Spectral Distribution")
print(sd_constant.shape)
print(sd_constant[400])

# Defining a zeros filled spectral distribution.
print('\nZeros Filled Spectral Distribution')
sd_zeros = colour.sd_zeros()
print(sd_zeros.shape)
print(sd_zeros[400])

# Defining a ones filled spectral distribution.
print('\nOnes Filled Spectral Distribution')
sd_ones = colour.sd_ones()
print(sd_ones.shape)
print(sd_ones[400])
```

```
"Constant Spectral Distribution"
(360.0, 780.0, 1.0)
100.0

"Zeros Filled Spectral Distribution"
(360.0, 780.0, 1.0)
0.0

"Ones Filled Spectral Distribution"
(360.0, 780.0, 1.0)
1.0
```

By default the shape used by `colour.sd_constant`, `colour.sd_zeros` and `colour.sd_ones` is the one defined by the `colour.DEFAULT_SPECTRAL_SHAPE` attribute and based on *ASTM E308-15* practise shape.

```
print(repr(colour.DEFAULT_SPECTRAL_SHAPE))
```

```
SpectralShape(360, 780, 1)
```

A custom shape can be passed to construct a constant spectral distribution with user defined dimensions:

```
colour.sd_ones(colour.SpectralShape(400, 700, 5))[450]
```

```
1.0
```

The `colour.SpectralDistribution` class supports the following arithmetical operations:

- *addition*
- *subtraction*
- *multiplication*
- *division*
- *exponentiation*

```
sd1 = colour.sd_ones()
print('"Ones Filled Spectral Distribution"')
print(sd1[400])

print('\n"x2 Constant Multiplied"')
print((sd1 * 2)[400])

print('\n"+ Spectral Distribution"')
print((sd1 + colour.sd_ones())[400])
```

```
"Ones Filled Spectral Distribution"
1.0

"x2 Constant Multiplied"
2.0

"+ Spectral Distribution"
2.0
```

Often interpolation of the spectral distribution is required, this is achieved with the `colour.SpectralDistribution.interpolate` method. Depending on the wavelengths uniformity, the default interpolation method will differ. Following *CIE 167:2005* recommendation: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable [CIET13805a].

The uniformity of the sample spectral distribution is assessed as follows:

```
# Checking the sample spectral distribution uniformity.
print(sd.is_uniform())
```

```
True
```

In this case, since the sample spectral distribution is uniform the interpolation defaults to the `colour.SpragueInterpolator` interpolator.

Note: Interpolation happens in place and may alter the original data, use the `colour.SpectralDistribution.copy` method to generate a copy of the spectral distribution before interpolation.

```
# Copying the sample spectral distribution.
sd_copy = sd.copy()
```

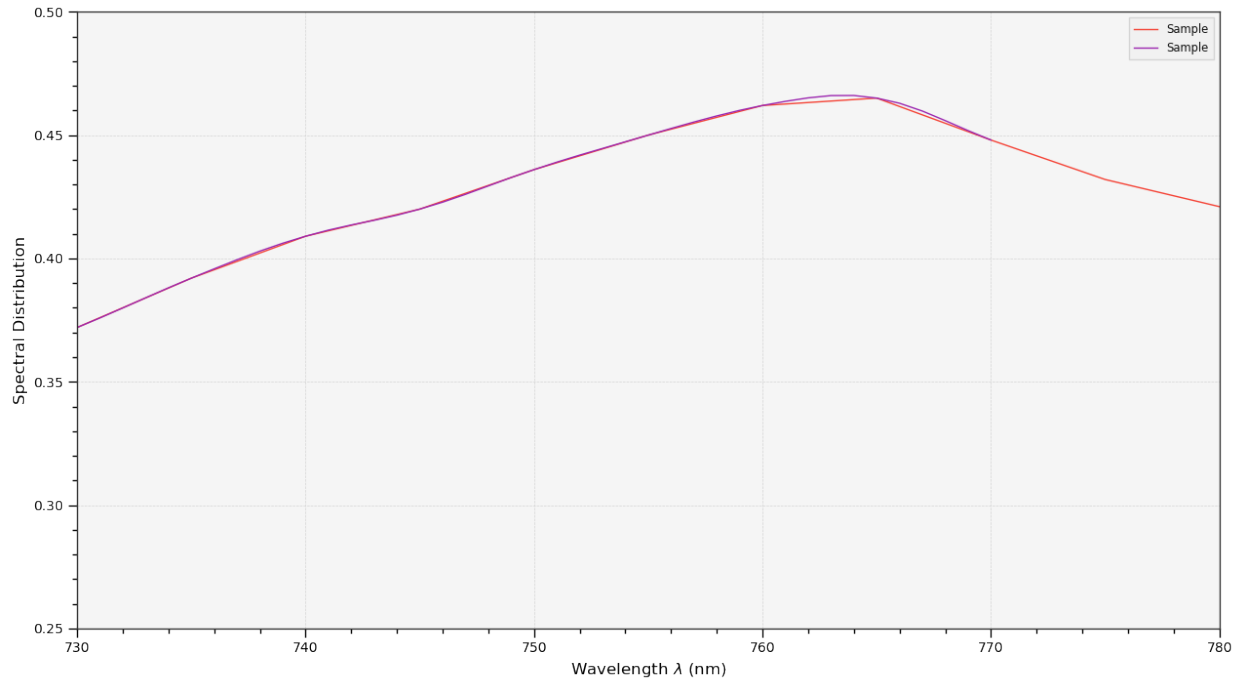
(continues on next page)

(continued from previous page)

```
# Interpolating the copied sample spectral distribution.
sd_copy.interpolate(colour.SpectralShape(400, 770, 1))
sd_copy[401]
```

```
0.065809599999999996
```

```
# Comparing the interpolated spectral distribution with the original one.
plot_multi_sds([sd, sd_copy], bounding_box=[730, 780, 0.25, 0.5])
```



Extrapolation although dangerous can be used to help aligning two spectral distributions together. *CIE publication CIE 15:2004 “Colorimetry”* recommends that unmeasured values may be set equal to the nearest measured value of the appropriate quantity in truncation [CIET14804d]:

```
# Extrapolating the copied sample spectral distribution.
sd_copy.extrapolate(colour.SpectralShape(340, 830))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.44800000000000018)
```

The underlying interpolator can be swapped for any of the **Colour** interpolators:

```
pprint([
    export for export in colour.algebra.interpolation.__all__
        if 'Interpolator' in export
])
```

```
[u'KernelInterpolator',
 u'LinearInterpolator',
 u'SpragueInterpolator',
```

(continues on next page)

(continued from previous page)

```
u'CubicSplineInterpolator',
u'PchipInterpolator',
u'NullInterpolator']
```

```
# Changing interpolator while trimming the copied spectral distribution.
sd_copy.interpolate(
    colour.SpectralShape(400, 700, 10), interpolator=colour.LinearInterpolator)
```

```
SpectralDistribution([[ 4.00000000e+02,  6.50000000e-02],
 [ 4.10000000e+02,  6.80000000e-02],
 [ 4.20000000e+02,  6.40000000e-02],
 [ 4.30000000e+02,  5.90000000e-02],
 [ 4.40000000e+02,  5.50000000e-02],
 [ 4.50000000e+02,  5.30000000e-02],
 [ 4.60000000e+02,  5.20000000e-02],
 [ 4.70000000e+02,  5.20000000e-02],
 [ 4.80000000e+02,  5.40000000e-02],
 [ 4.90000000e+02,  5.70000000e-02],
 [ 5.00000000e+02,  6.10000000e-02],
 [ 5.10000000e+02,  6.50000000e-02],
 [ 5.20000000e+02,  7.00000000e-02],
 [ 5.30000000e+02,  7.40000000e-02],
 [ 5.40000000e+02,  7.60000000e-02],
 [ 5.50000000e+02,  7.90000000e-02],
 [ 5.60000000e+02,  8.70000000e-02],
 [ 5.70000000e+02,  1.00000000e-01],
 [ 5.80000000e+02,  1.15000000e-01],
 [ 5.90000000e+02,  1.29000000e-01],
 [ 6.00000000e+02,  1.38000000e-01],
 [ 6.10000000e+02,  1.46000000e-01],
 [ 6.20000000e+02,  1.54000000e-01],
 [ 6.30000000e+02,  1.63000000e-01],
 [ 6.40000000e+02,  1.73000000e-01],
 [ 6.50000000e+02,  1.88000000e-01],
 [ 6.60000000e+02,  2.04000000e-01],
 [ 6.70000000e+02,  2.22000000e-01],
 [ 6.80000000e+02,  2.42000000e-01],
 [ 6.90000000e+02,  2.61000000e-01],
 [ 7.00000000e+02,  2.82000000e-01]],
    interpolator=SpragueInterpolator,
    interpolator_args={},
    extrapolator=Extrapolator,
    extrapolator_args={'u'right': None, u'method': u'Constant', u'left': None})
```

The extrapolation behaviour can be changed for Linear method instead of the Constant default method or even use arbitrary constant left and right values:

```
# Extrapolating the copied sample spectral distribution with *Linear* method.
sd_copy.extrapolate(
    colour.SpectralShape(340, 830),
    extrapolator_args={'method': 'Linear',
                       'right': 0})
sd_copy[340], sd_copy[830]
```

```
(0.0469999999999999348, 0.0)
```

Aligning a spectral distribution is a convenient way to first interpolate the current data within its original bounds, then, if required, extrapolate any missing values to match the requested shape:

```
# Aligning the cloned sample spectral distribution.
# The spectral distribution is first trimmed as above.
sd_copy.interpolate(colour.SpectralShape(400, 700))
sd_copy.align(colour.SpectralShape(340, 830, 5))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.28199999999999975)
```

The `colour.SpectralDistribution` class also supports various arithmetic operations like *addition*, *subtraction*, *multiplication*, *division* or *exponentiation* with *numeric* and *array_like* variables or other `colour.SpectralDistribution` class instances:

```
sd = colour.SpectralDistribution({
    410: 0.25,
    420: 0.50,
    430: 0.75,
    440: 1.0,
    450: 0.75,
    460: 0.50,
    480: 0.25
})

print((sd.copy() + 1).values)
print((sd.copy() * 2).values)
print((sd * [0.35, 1.55, 0.75, 2.55, 0.95, 0.65, 0.15]).values)
print((sd * colour.sd_constant(2, sd.shape) * colour.sd_constant(3, sd.shape)).values)
```

```
[ 1.25  1.5   1.75  2.    1.75  1.5   1.25]
[ 0.5   1.    1.5   2.    1.5   1.    0.5]
[ 0.0875 0.775 0.5625 2.55 0.7125 0.325 0.0375]
[ 1.5   3.    4.5   6.    4.5   3.    nan  1.5]
```

The spectral distribution can be normalised with an arbitrary factor:

```
print(sd.normalise().values)
print(sd.normalise(100).values)
```

```
[ 0.25  0.5   0.75  1.    0.75  0.5   0.25]
[ 25.   50.   75.  100.   75.   50.   25.]
```

At the heart of the `colour.SpectralDistribution` class is the `colour.continuous.Signal` class which implements the `colour.continuous.Signal.function` method.

Evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable.

It adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the `colour.continuous.Signal.domain` attribute corresponds with the function dependent and already known range stored in the `colour.continuous.Signal.range` attribute.

Describing the `colour.continuous.Signal` class is beyond the scope of this tutorial but the core capability can be described.

```
import numpy as np
```

```
range_ = np.linspace(10, 100, 10)
signal = colour.continuous.Signal(range_)
print(repr(signal))
```

```
Signal([[ 0.,  10.],
 [ 1.,  20.],
 [ 2.,  30.],
 [ 3.,  40.],
 [ 4.,  50.],
 [ 5.,  60.],
 [ 6.,  70.],
 [ 7.,  80.],
 [ 8.,  90.],
 [ 9., 100.]],
 interpolator=KernelInterpolator,
 interpolator_args={},
 extrapolator=Extrapolator,
 extrapolator_args={u'right': nan, u'method': u'Constant', u'left': nan})
```

```
# Returning the corresponding range *y* variable for any arbitrary independent domain *x* variable.
signal[np.random.uniform(0, 9, 10)]
```

```
array([ 55.91309735,  65.4172615 ,  65.54495059,  88.17819416,
        61.88860248,  10.53878826,  55.25130534,  46.14659783,
        86.41406136,  84.59897703])
```

5.1.1.3 Convert to Tristimulus Values

From a given spectral distribution, *CIE XYZ* tristimulus values can be calculated:

```
sd = colour.SpectralDistribution(sample_sd_data)
cmfs = colour.STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
illuminant = colour.ILLUMINANTS_SDS['D65']

# Calculating the sample spectral distribution *CIE XYZ* tristimulus values.
XYZ = colour.sd_to_XYZ(sd, cmfs, illuminant)
print(XYZ)
```

```
[ 10.97085572  9.70278591  6.05562778]
```

5.1.1.4 From *CIE XYZ* Colourspace

CIE XYZ is the central colourspace for Colour Science from which many computations are available, expanding to even more computations:

```
# Displaying objects interacting directly with the *CIE XYZ* colourspace.
pprint([name for name in colour.__all__ if name.startswith('XYZ_to')])
```

```
['XYZ_to_ATD95',
 'XYZ_to_CAM16',
```

(continues on next page)

(continued from previous page)

```
'XYZ_to_CIECAM02',
'XYZ_to_Hunt',
'XYZ_to_LLAB',
'XYZ_to_Nayatani95',
'XYZ_to_RLAB',
'XYZ_to_Hunter_Lab',
'XYZ_to_Hunter_Rdab',
'XYZ_to_IPT',
'XYZ_to_JzAzBz',
'XYZ_to_K_ab_HunterLab1966',
'XYZ_to_Lab',
'XYZ_to_Luv',
'XYZ_to_OSA_UCS',
'XYZ_to_RGB',
'XYZ_to_UCS',
'XYZ_to_UVW',
'XYZ_to_hdr_CIELab',
'XYZ_to_hdr_IPT',
'XYZ_to_sRGB',
'XYZ_to_xy',
'XYZ_to_xyY',
'XYZ_to_sd']
```

5.1.1.5 Convert to Display Colours

CIE XYZ tristimulus values can be converted into *sRGB* colourspace *RGB* values in order to display them on screen:

```
# The output domain of *colour.sd_to_XYZ* is [0, 100] and the input
# domain of *colour.XYZ_to_sRGB* is [0, 1]. It needs to be accounted for,
# thus the input *CIE XYZ* tristimulus values are scaled.
RGB = colour.XYZ_to_sRGB(XYZ / 100)
print(RGB)
```

```
[ 0.45675795  0.30986982  0.24861924]
```

```
# Plotting the *sRGB* colourspace colour of the *Sample* spectral distribution.
plot_single_colour_swatch(
    ColourSwatch('Sample', RGB),
    text_parameters={'size': 'x-large'})
```




5.1.1.6 Generate Colour Rendition Charts

Likewise, colour values from a colour rendition chart sample can be computed.

Note: This is useful for render time checks in the VFX industry, where a synthetic colour chart can be inserted into a render to ensure the colour management is acting as expected.

The `colour.characterisation` sub-package contains the dataset for various colour rendition charts:

```
# Colour rendition charts chromaticity coordinates.
print(sorted(colour.characterisation.COLOURCHECKERS.keys()))

# Colour rendition charts spectral distributions.
print(sorted(colour.characterisation.COLOURCHECKERS_SDS.keys()))
```

```
['BabelColor Average', 'ColorChecker 1976', 'ColorChecker 2005', 'ColorChecker24 - After November 2014',
 → 'ColorChecker24 - Before November 2014', 'babel_average', 'cc2005', 'cca2014', 'ccb2014']
['BabelColor Average', 'ColorChecker N Ohta', 'babel_average', 'cc_ohta']
```

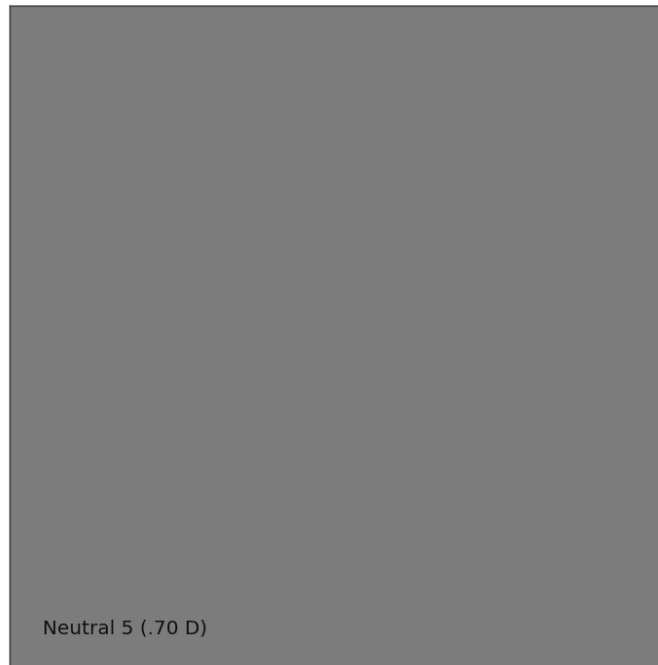
Note: The above `cc2005`, `babel_average` and `cc_ohta` keys are convenient aliases for respectively `ColorChecker 2005`, `BabelColor Average` and `ColorChecker N Ohta` keys.

```
# Plotting the *sRGB* colourspace colour of *neutral 5 (.70 D)* patch.
patch_name = 'neutral 5 (.70 D)'
patch_sd = colour.COLOURCHECKERS_SDS['ColorChecker N Ohta'][patch_name]
XYZ = colour.sd_to_XYZ(patch_sd, cmfs, illuminant)
RGB = colour.XYZ_to_sRGB(XYZ / 100)
```

(continues on next page)

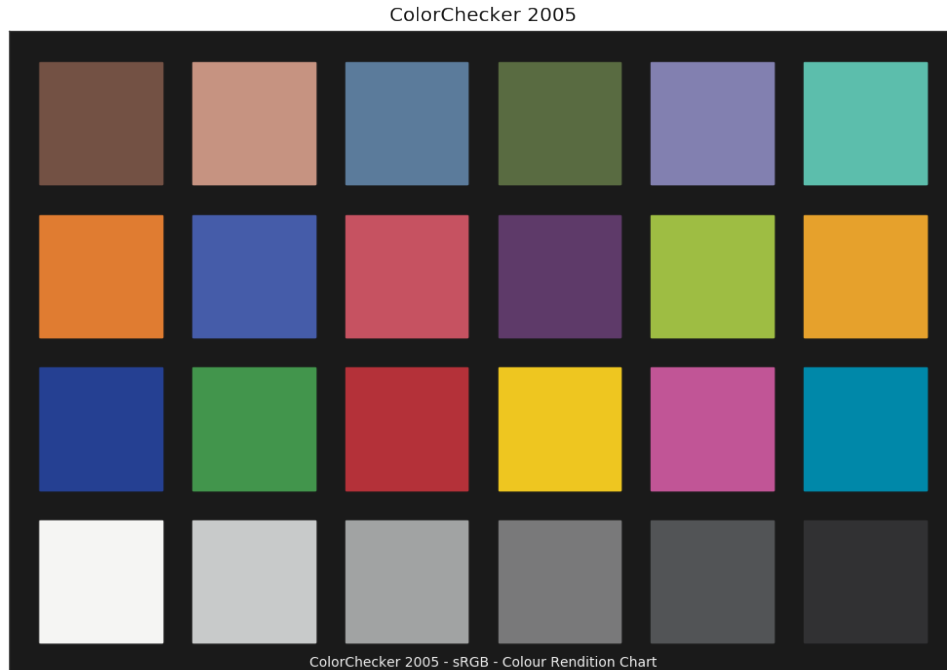
(continued from previous page)

```
plot_single_colour_swatch(  
    ColourSwatch(patch_name.title(), RGB),  
    text_parameters={'size': 'x-large'})
```



Colour defines a convenient plotting object to draw synthetic colour rendition charts figures:

```
plot_single_colour_checker(  
    colour_checker='ColorChecker 2005', text_parameters={'visible': False})
```



5.1.1.7 Convert to Chromaticity Coordinates

Given a spectral distribution, chromaticity coordinates xy can be computed using the `colour.XYZ_to_xy` definition:

```
# Computing *xy* chromaticity coordinates for the *neutral 5 (.70 D)* patch.
xy = colour.XYZ_to_xy(XYZ)
print(xy)
```

```
[ 0.31259787  0.32870029]
```

Chromaticity coordinates xy can be plotted into the *CIE 1931 Chromaticity Diagram*:

```
import matplotlib.pyplot as plt

# Plotting the *CIE 1931 Chromaticity Diagram*.
# The argument *standalone=False* is passed so that the plot doesn't get
# displayed and can be used as a basis for other plots.
plot_chromaticity_diagram_CIE1931(standalone=False)

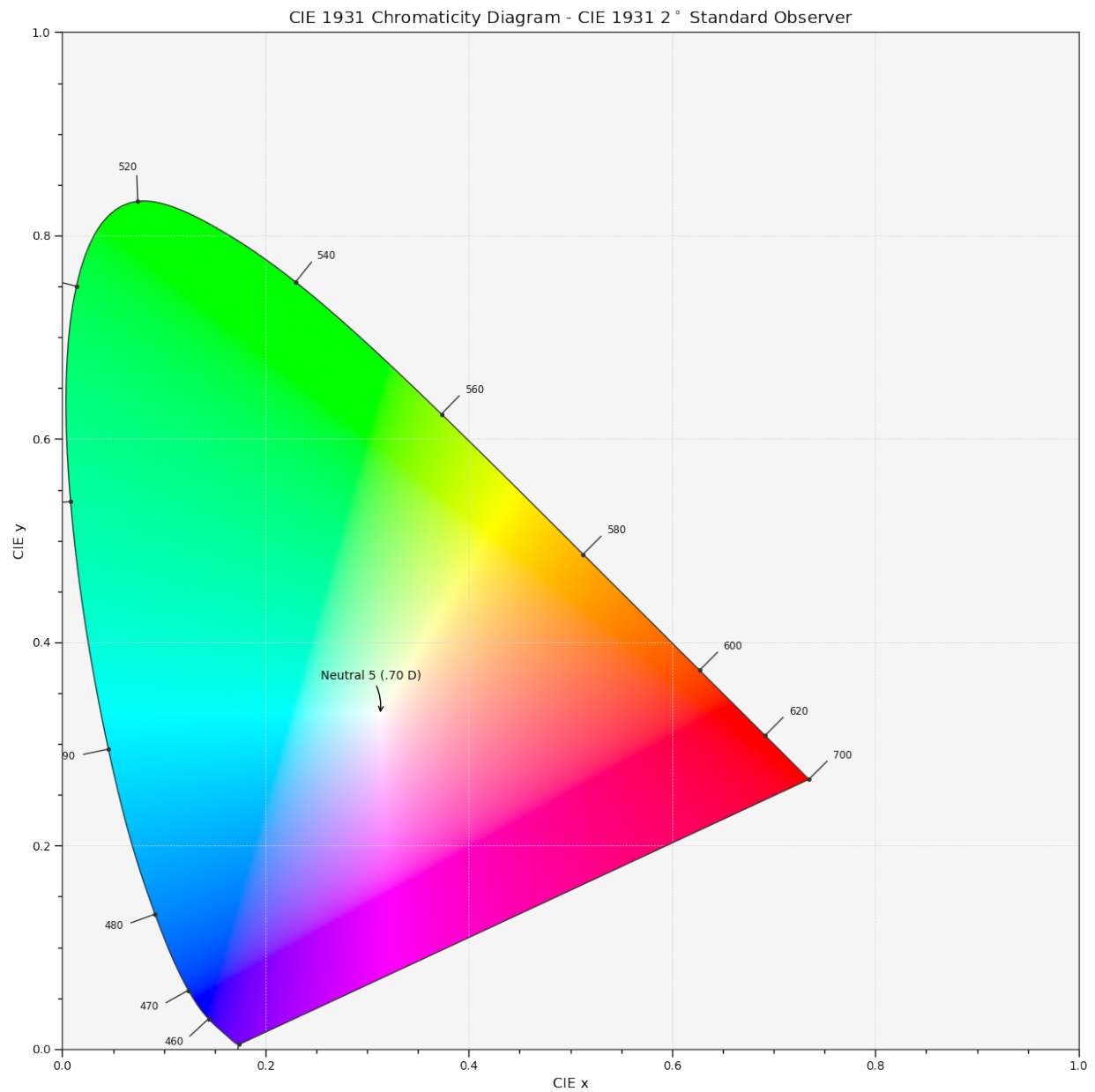
# Plotting the *xy* chromaticity coordinates.
x, y = xy
plt.plot(x, y, 'o-', color='white')

# Annotating the plot.
plt.annotate(patch_sd.name.title(),
             xy=xy,
             xytext=(-50, 30),
             textcoords='offset points',
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3, rad=-0.2'))
```

(continues on next page)

(continued from previous page)

```
# Displaying the plot.  
render(  
    standalone=True,  
    limits=(-0.1, 0.9, -0.1, 0.9),  
    x_tighten=True,  
    y_tighten=True)
```



5.1.1.8 And More...

With the hope that this small introduction was useful and gave envy to see more, a good place to explore the API further is the [Jupyter Notebooks](#) page.

5.1.2 Basics

5.1.2.1 N-Dimensional Arrays Support

Most of [Colour](#) definitions are fully vectorised and support n-dimensional arrays by leveraging [Numpy](#).

While it is recommended to use `ndarrays` as input for the API objects, it is possible to use tuples or lists:

```
import colour
```

```
xyY = (0.4316, 0.3777, 0.1008)
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475, 0.1008      , 0.05089373])
```

```
xyY = [0.4316, 0.3777, 0.1008]
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475, 0.1008      , 0.05089373])
```

```
xyY = [
    (0.4316, 0.3777, 0.1008),
    (0.4316, 0.3777, 0.1008),
    (0.4316, 0.3777, 0.1008),
]
colour.xyY_to_XYZ(xyY)
```

```
array([[ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373]])
```

As shown in the above example, there is widespread support for n-dimensional arrays:

```
import numpy as np
```

```
xyY = np.array([0.4316, 0.3777, 0.1008])
xyY = np.tile(xyY, (6, 1))
colour.xyY_to_XYZ(xyY)
```

```
array([[ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373],
       [ 0.11518475, 0.1008      , 0.05089373]])
```

```
colour.xyY_to_XYZ(xyY.reshape([2, 3, 3]))
```

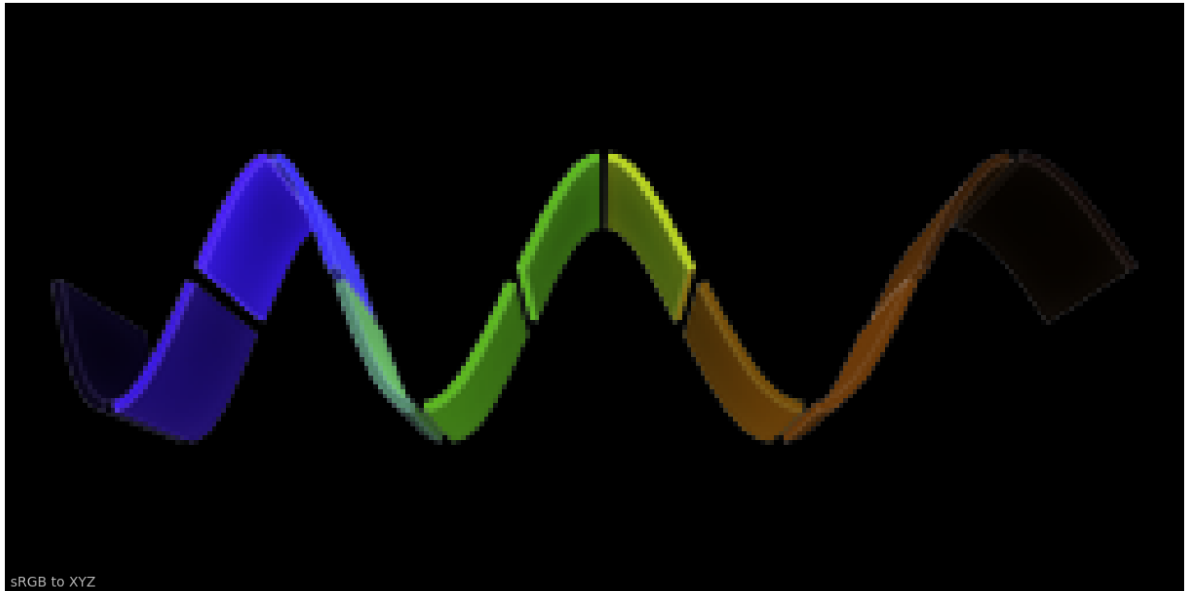
```
array([[[ 0.11518475, 0.1008      , 0.05089373],
        [ 0.11518475, 0.1008      , 0.05089373],
        [ 0.11518475, 0.1008      , 0.05089373]],

       [[ 0.11518475, 0.1008      , 0.05089373],
        [ 0.11518475, 0.1008      , 0.05089373],
        [ 0.11518475, 0.1008      , 0.05089373]])
```

Which enables image processing:

```
import colour.plotting

RGB = colour.read_image('_static/Logo_Small_001.png')
RGB = RGB[..., 0:3] # Discarding alpha channel.
XYZ = colour.sRGB_to_XYZ(RGB)
colour.plotting.plot_image(XYZ, text_parameters={'text': 'sRGB to XYZ'})
```



5.1.2.2 Domain-Range Scales

Note: This section has important information.

Colour adopts 4 main input domains and output ranges:

- *Scalars* usually in domain-range $[0, 1]$ (or $[0, 10]$ for *Munsell Value*).
- *Percentages* usually in domain-range $[0, 100]$.
- *Degrees* usually in domain-range $[0, 360]$.
- *Integers* usually in domain-range $[0, 2^{*n} - 1]$ where n is the bit depth.

It is error prone but it is also a direct consequence of the inconsistency of the colour science field itself. We have discussed at length about this and we were leaning toward normalisation of the whole API to domain-range $[0, 1]$, we never committed for reasons highlighted by the following points:

- Colour Scientist performing computations related to Munsell Renotation System would be very surprised if the output *Munsell Value* was in range $[0, 1]$ or $[0, 100]$.
- A Visual Effect Industry artist would be astonished to find out that conversion from *CIE XYZ* to *sRGB* was yielding values in range $[0, 100]$.

However benefits of having a consistent and predictable domain-range scale are numerous thus with [Colour 0.3.12](#) we have introduced a mechanism to allow users to work within one of the two available domain-range scales.

Scale - Reference

‘Reference’ is the default domain-range scale of **Colour**, objects adopt the implemented reference, i.e. paper, publication, etc., domain-range scale.

The **‘Reference’** domain-range scale is inconsistent, e.g. colour appearance models, spectral conversions are typically in domain-range $[0, 100]$ while RGB models will operate in domain-range $[0, 1]$. Some objects, e.g. `colour.colorimetry.lightness_Fairchild2011()` definition have mismatched domain-range: input domain $[0, 1]$ and output range $[0, 100]$.

Scale - 1

‘1’ is a domain-range scale converting all the relevant objects from **Colour** public API to domain-range $[0, 1]$:

- *Scalars* in domain-range $[0, 10]$, e.g. *Munsell Value* are scaled by 10.
- *Percentages* in domain-range $[0, 100]$ are scaled by 100.
- *Degrees* in domain-range $[0, 360]$ are scaled by 100.
- *Integers* in domain-range $[0, 2^{*n} - 1]$ where n is the bit depth are scaled by $2^{*n} - 1$.

Warning: The conversion to **‘1’** domain-range scale is a *soft* normalisation and similarly to the **‘Reference’** domain-range scale it is normal that you encounter values exceeding 1, e.g. High Dynamic Range Imagery (HDRI) or negative values, e.g. out-of-gamut RGB colourspace values.

Understanding the Domain-Range Scale of an Object

Using `colour.adaptation.chromatic_adaptation_CIE1994()` definition docstring as an example, the *Notes* section features two tables.

The first table is for the domain, and lists the input arguments affected by the two domain-range scales and which normalisation they should adopt depending the domain-range scale in use:

Domain	Scale - Reference	Scale - 1
XYZ_1	$[0, 100]$	$[0, 1]$
Y_o	$[0, 100]$	$[0, 1]$

The second table is for the range and lists the return value of the definition:

Range	Scale - Reference	Scale - 1
XYZ_2	$[0, 100]$	$[0, 1]$

Working with the Domain-Range Scales

The current domain-range scale is returned with the `colour.get_domain_range_scale()` definition:

```
import colour
```

```
colour.get_domain_range_scale()
```

```
u'reference'
```

Changing from the **'Reference'** default domain-range scale to **'1'** is done with the `colour.set_domain_range_scale()` definition:

```
XYZ_1 = [28.00, 21.26, 5.27]
xy_o1 = [0.4476, 0.4074]
xy_o2 = [0.3127, 0.3290]
Y_o = 20
E_o1 = 1000
E_o2 = 1000
colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
array([ 24.03379521,  21.15621214,  17.64301199])
```

```
colour.set_domain_range_scale('1')
```

```
XYZ_1 = [0.2800, 0.2126, 0.0527]
Y_o = 0.2
colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
array([ 0.24033795,  0.21156212,  0.17643012])
```

The output tristimulus values with the **'1'** domain-range scale are equal to those from **'Reference'** default domain-range scale divided by 100.

Passing incorrectly scaled values to the `colour.adaptation.chromatic_adaptation_CIE1994()` definition would result in unexpected values and a warning in that case:

```
colour.set_domain_range_scale('Reference')
```

```
colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
File "<ipython-input-...>", line 4, in <module>
    E_o2)
File "/colour-science/colour/colour/adaptation/cie1994.py", line 134, in chromatic_adaptation_CIE1994
    warning(("Y_o" luminance factor must be in [18, 100] domain, '
/colour-science/colour/colour/utilities/verbose.py:207: ColourWarning: "Y_o" luminance factor must be_
in [18, 100] domain, unpredictable results may occur!
    warn(*args, **kwargs)
array([ 0.17171825,  0.13731098,  0.09972054])
```

Setting the **'1'** domain-range scale has the following effect on the `colour.adaptation.chromatic_adaptation_CIE1994()` definition:

As it expects values in domain $[0, 100]$, scaling occurs and the relevant input values, i.e. the values listed in the domain table, XYZ_1 and Y_o are converted from domain $[0, 1]$ to domain $[0, 100]$ by `colour.utilities.to_domain_100()` definition and conversely return value XYZ_2 is converted from range $[0, 100]$ to range $[0, 1]$ by `colour.utilities.from_range_100()` definition.

A convenient alternative to the `colour.set_domain_range_scale()` definition is the `colour.domain_range_scale` context manager and decorator. It temporarily overrides **Colour** domain-range scale with given scale value:

```
with colour.domain_range_scale('1'):
    colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
[ 0.24033795  0.21156212  0.17643012]
```

Multiprocessing on Windows with Domain-Range Scales

Windows does not have a `fork` system call, a consequence is that child processes do not necessarily inherit from changes made to global variables.

It has crucial consequences as **Colour** stores the current domain-range scale into a global variable.

The solution is to define an initialisation definition that defines the scale upon child processes spawning.

The `colour.utilities.multiprocessing_pool` context manager conveniently performs the required initialisation so that the domain-range scale is propagated appropriately to child processes.

5.1.3 Reference

5.1.3.1 Colour

Chromatic Adaptation

- *Chromatic Adaptation*
- *Fairchild (1990)*
- *CIE 1994*
- *CMCCAT2000*
- *Von Kries*

Chromatic Adaptation

colour

<code>chromatic_adaptation(XYZ, XYZ_w, XYZ_wr[, ...])</code>	Adapts given stimulus from test viewing conditions to reference viewing conditions.
<code>CHROMATIC_ADAPTATION_METHODS</code>	Supported chromatic adaptation methods.
<code>CMCCAT2000_VIEWING_CONDITIONS</code>	Reference <i>CMCCAT2000</i> chromatic adaptation model viewing conditions.

colour.chromatic_adaptation

`colour.chromatic_adaptation(XYZ, XYZ_w, XYZ_wr, method='Von Kries', **kwargs)`
 Adapts given stimulus from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition CIE XYZ tristimulus values of the white-point.
- **XYZ_wr** (array_like) – Reference viewing condition CIE XYZ tristimulus values of the whitepoint.
- **method** (unicode, optional) – {'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}, Computation method.

Other Parameters

- **E_o1** (numeric) – {`colour.adaptation.chromatic_adaptation_CIE1994()`}, Test illuminance E_{o1} in cd/m^2 .
- **E_o2** (numeric) – {`colour.adaptation.chromatic_adaptation_CIE1994()`}, Reference illuminance E_{o2} in cd/m^2 .
- **L_A1** (numeric or array_like) – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Luminance of test adapting field L_{A1} in cd/m^2 .
- **L_A2** (numeric or array_like) – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Luminance of reference adapting field L_{A2} in cd/m^2 .
- **Y_n** (numeric or array_like) – {`colour.adaptation.chromatic_adaptation_Fairchild1990()`}, Luminance Y_n of test adapting stimulus in cd/m^2 .
- **Y_o** (numeric) – {`colour.adaptation.chromatic_adaptation_CIE1994()`}, Luminance factor Y_o of achromatic background normalised to domain [0.18, 1] in 'Reference' domain-range scale.
- **direction** (unicode, optional) – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, {'Forward', 'Reverse'}, Chromatic adaptation direction.
- **discount_illuminant** (bool, optional) – {`colour.adaptation.chromatic_adaptation_Fairchild1990()`}, Truth value indicating if the illuminant should be discounted.
- **n** (numeric, optional) – {`colour.adaptation.chromatic_adaptation_CIE1994()`}, Noise component in fundamental primary system.
- **surround** (CMCCAT2000_InductionFactors, optional) – {`colour.adaptation.chromatic_adaptation_CMCCAT2000()`}, Surround viewing conditions induction factors.
- **transform** (unicode, optional) – {`colour.adaptation.chromatic_adaptation_VonKries()`}, {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns CIE XYZ_c tristimulus values of the stimulus corresponding colour.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
XYZ_w	[0, 1]	[0, 1]
XYZ_wr	[0, 1]	[0, 1]
Y_o	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 1]	[0, 1]

References

[CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

Examples

Von Kries chromatic adaptation:

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation(XYZ, XYZ_w, XYZ_wr)
... # doctest: +ELLIPSIS
array([ 0.2163881..., 0.1257    , 0.0384749...])
```

CIE 1994 chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.2800, 0.2126, 0.0527])
>>> XYZ_w = np.array([1.09867452, 1.00000000, 0.35591556])
>>> XYZ_wr = np.array([0.95045593, 1.00000000, 1.08905775])
>>> Y_o = 0.20
>>> E_o = 1000
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='CIE 1994', Y_o=Y_o, E_o1=E_o, E_o2=E_o)
... # doctest: +ELLIPSIS
array([ 0.2403379..., 0.2115621..., 0.1764301...])
```

CMCCAT2000 chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.2248, 0.2274, 0.0854])
>>> XYZ_w = np.array([1.1115, 1.0000, 0.3520])
>>> XYZ_wr = np.array([0.9481, 1.0000, 1.0730])
>>> L_A = 200
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='CMCCAT2000', L_A1=L_A, L_A2=L_A)
... # doctest: +ELLIPSIS
array([ 0.1952698..., 0.2306834..., 0.2497175...])
```

Fairchild (1990) chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.1953, 0.2307, 0.2497])
>>> Y_n = 200
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='Fairchild 1990', Y_n=Y_n)
... # doctest: +ELLIPSIS
array([ 0.2332526...,  0.2332455...,  0.7611593...])
```

colour.CHROMATIC_ADAPTATION_METHODS

`colour.CHROMATIC_ADAPTATION_METHODS = CaseInsensitiveMapping({'CIE 1994': ..., 'CMCCAT2000': ..., 'Fairchild 1990': ...})`
Supported chromatic adaptation methods.

References

[CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

CHROMATIC_ADAPTATION_METHODS [CaseInsensitiveMapping] {'CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries'}

colour.CMCCAT2000_VIEWING_CONDITIONS

`colour.CMCCAT2000_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`
Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12a]

CMCCAT2000_VIEWING_CONDITIONS [CaseInsensitiveMapping] ('Average', 'Dim', 'Dark')

Dataset

colour

CHROMATIC_ADAPTATION_TRANSFORMS

Supported chromatic adaptation transforms.

colour.CHROMATIC_ADAPTATION_TRANSFORMS

`colour.CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Kries': ..., 'Bradford': ..., 'Fairchild': ..., 'CMCCAT97': ..., 'CMCCAT2000': ..., 'CAT02_BRILL_CAT': ..., 'Bianco': ..., 'Bianco PC': ...})`
Supported chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMv07], [Lin09a], [WRC12b], [WRC12a], [Wik07a]

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}

Fairchild (1990)

colour.adaptation

<code>chromatic_adaptation_Fairchild1990(XYZ_1, ...)</code>	Adapts given stimulus <i>CIE XYZ_1</i> tristimulus values from test viewing conditions to reference viewing conditions using <i>Fairchild (1990)</i> chromatic adaptation model.
---	--

colour.adaptation.chromatic_adaptation_Fairchild1990

`colour.adaptation.chromatic_adaptation_Fairchild1990(XYZ_1, XYZ_n, XYZ_r, Y_n, discount_illuminant=False)`

Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *Fairchild (1990)* chromatic adaptation model.

Parameters

- **XYZ_1** (array_like) – *CIE XYZ_1* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – Test viewing condition *CIE XYZ_n* tristimulus values of white-point.
- **XYZ_r** (array_like) – Reference viewing condition *CIE XYZ_r* tristimulus values of whitepoint.
- **Y_n** (numeric or array_like) – Luminance Y_n of test adapting stimulus in cd/m^2 .
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns Adapted *CIE XYZ_2* tristimulus values of stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_1	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]
XYZ_r	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_2	[0, 100]	[0, 1]

References

[Fai91], [Fai13c]

Examples

```
>>> XYZ_1 = np.array([19.53, 23.07, 24.97])
>>> XYZ_n = np.array([111.15, 100.00, 35.20])
>>> XYZ_r = np.array([94.81, 100.00, 107.30])
>>> Y_n = 200
>>> chromatic_adaptation_Fairchild1990(XYZ_1, XYZ_n, XYZ_r, Y_n)
... # doctest: +ELLIPSIS
array([ 23.3252634...,  23.3245581...,  76.1159375...])
```

CIE 1994

colour.adaptation

<code>chromatic_adaptation_CIE1994(XYZ_1, xy_o1, ...)</code>	Adapts given stimulus <i>CIE XYZ_1</i> tristimulus values from test viewing conditions to reference viewing conditions using <i>CIE 1994</i> chromatic adaptation model.
--	--

colour.adaptation.chromatic_adaptation_CIE1994

`colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2, n=1)`
Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *CIE 1994* chromatic adaptation model.

Parameters

- **XYZ_1** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **xy_o1** (array_like) – Chromaticity coordinates x_{o1} and y_{o1} of test illuminant and background.
- **xy_o2** (array_like) – Chromaticity coordinates x_{o2} and y_{o2} of reference illuminant and background.
- **Y_o** (numeric) – Luminance factor Y_o of achromatic background as percentage normalised to domain [18, 100] in ‘Reference’ domain-range scale.
- **E_o1** (numeric) – Test illuminance E_{o1} in cd/m^2 .
- **E_o2** (numeric) – Reference illuminance E_{o2} in cd/m^2 .
- **n** (numeric, optional) – Noise component in fundamental primary system.

Returns Adapted *CIE XYZ_2* tristimulus values of test stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_1	[0, 100]	[0, 1]
Y_o	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_2	[0, 100]	[0, 1]

References

[CIET13294]

Examples

```
>>> XYZ_1 = np.array([28.00, 21.26, 5.27])
>>> xy_o1 = np.array([0.4476, 0.4074])
>>> xy_o2 = np.array([0.3127, 0.3290])
>>> Y_o = 20
>>> E_o1 = 1000
>>> E_o2 = 1000
>>> chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
... # doctest: +ELLIPSIS
array([ 24.0337952..., 21.1562121..., 17.6430119...])
```

CMCCAT2000

colour.adaptation

chromatic_adaptation_CMCCAT2000(XYZ, XYZ_w, ...)	Adapts given stimulus <i>CIE XYZ</i> tristimulus values using given viewing conditions.
CMCCAT2000_VIEWING_CONDITIONS	Reference <i>CMCCAT2000</i> chromatic adaptation model viewing conditions.

colour.adaptation.chromatic_adaptation_CMCCAT2000

colour.adaptation.chromatic_adaptation_CMCCAT2000(XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, surround=CMCCAT2000_InductionFactors(F=1), direction='Forward')

Adapts given stimulus *CIE XYZ* tristimulus values using given viewing conditions.

This definition is a convenient wrapper around `colour.adaptation.chromatic_adaptation_forward_CMCCAT2000()` and `colour.adaptation.chromatic_adaptation_reverse_CMCCAT2000()`.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Source viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Target viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2 .

- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2 .
- **surround** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.
- **direction** (unicode, optional) – {'Forward', 'Reverse'}, Chromatic adaptation direction.

Returns Adapted stimulus CIE XYZ tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_wr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ = np.array([22.48, 22.74, 8.54])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_CMCCAT2000(
...     XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, direction='Forward')
... # doctest: +ELLIPSIS
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

Using the CMCCAT2000 reverse model:

```
>>> XYZ = np.array([19.52698326, 23.06833960, 24.97175229])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_CMCCAT2000(
...     XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, direction='Reverse')
... # doctest: +ELLIPSIS
array([ 22.48,  22.74,   8.54])
```


colour.adaptation.CMCCAT2000_VIEWING_CONDITIONS

`colour.adaptation.CMCCAT2000_VIEWING_CONDITIONS` = `CaseInsensitiveMapping`({'Average': ..., 'Dim': ..., 'Dark': ...})
Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12a]

`CMCCAT2000_VIEWING_CONDITIONS` [`CaseInsensitiveMapping`] ('Average', 'Dim', 'Dark')

Ancillary Objects

`colour.adaptation`

<code>chromatic_adaptation_forward_CMCCAT2000(XYZ, ...)</code>	Adapts given stimulus <i>CIE XYZ</i> tristimulus values from test viewing conditions to reference viewing conditions using <i>CMCCAT2000</i> forward chromatic adaptation model.
<code>chromatic_adaptation_reverse_CMCCAT2000(...)</code>	Adapts given stimulus corresponding colour <i>CIE XYZ</i> tristimulus values from reference viewing conditions to test viewing conditions using <i>CMCCAT2000</i> reverse chromatic adaptation model.
<code>CMCCAT2000_InductionFactors</code>	<i>CMCCAT2000</i> chromatic adaptation model induction factors.

colour.adaptation.chromatic_adaptation_forward_CMCCAT2000

`colour.adaptation.chromatic_adaptation_forward_CMCCAT2000(XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, surround=CMCCAT2000_InductionFactors(F=1))`

Adapts given stimulus *CIE XYZ* tristimulus values from test viewing conditions to reference viewing conditions using *CMCCAT2000* forward chromatic adaptation model.

Parameters

- **XYZ** (`array_like`) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (`array_like`) – Test viewing condition *CIE XYZ* tristimulus values of the white-point.
- **XYZ_wr** (`array_like`) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (`numeric` or `array_like`) – Luminance of test adapting field L_{A1} in cd/m^2 .
- **L_A2** (`numeric` or `array_like`) – Luminance of reference adapting field L_{A2} in cd/m^2 .
- **surround** (`CMCCAT2000_InductionFactors`, optional) – Surround viewing conditions induction factors.

Returns *CIE XYZ_c* tristimulus values of the stimulus corresponding colour.

Return type `ndarray`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_wr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ = np.array([22.48, 22.74, 8.54])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_forward_CMCCAT2000(XYZ, XYZ_w, XYZ_wr, L_A1, L_A2)
... # doctest: +ELLIPSIS
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

colour.adaptation.chromatic_adaptation_reverse_CMCCAT2000

`colour.adaptation.chromatic_adaptation_reverse_CMCCAT2000(XYZ_c, XYZ_w, XYZ_wr, L_A1, L_A2, surround=CMCCAT2000_InductionFactors(F=1))`

Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* reverse chromatic adaptation model.

Parameters

- **XYZ_c** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of the white-point.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2 .
- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2 .
- **surround** (`CMCCAT2000_InductionFactors`, optional) – Surround viewing conditions induction factors.

Returns *CIE XYZ_c* tristimulus values of the adapted stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ _c	[0, 100]	[0, 1]
XYZ _w	[0, 100]	[0, 1]
XYZ _{wr}	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ_c = np.array([19.53, 23.07, 24.97])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_reverse_CMCCAT2000(XYZ_c, XYZ_w, XYZ_wr, L_A1,
...                                           L_A2)
... # doctest: +ELLIPSIS
array([ 22.4839876..., 22.7419485...,  8.5393392...])
```

colour.adaptation.CMCCAT2000_InductionFactors

class colour.adaptation.CMCCAT2000_InductionFactors

CMCCAT2000 chromatic adaptation model induction factors.

Parameters *F* (numeric or array_like) – *F* surround condition.

References

[LLRH02], [WRC12a]

Create new instance of CMCCAT2000_InductionFactors(*F*)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

F	Alias for field number 0
---	--------------------------

Von Kries

`colour.adaptation`

<code>chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)</code>	Adapts given stimulus from test viewing conditions to reference viewing conditions.
<code>CHROMATIC_ADAPTATION_TRANSFORMS</code>	Supported chromatic adaptation transforms.

`colour.adaptation.chromatic_adaptation_VonKries`

`colour.adaptation.chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr, transform='CAT02')`

Adapts given stimulus from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition CIE XYZ tristimulus values of whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition CIE XYZ tristimulus values of whitepoint.
- **transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns CIE XYZ_c tristimulus values of the stimulus corresponding colour.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
XYZ_n	[0, 1]	[0, 1]
XYZ_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 1]	[0, 1]

References

[Fai13b]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr) # doctest: +ELLIPSIS
array([ 0.2163881..., 0.1257..., 0.0384749...])
```

Using Bradford method:

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> transform = 'Bradford'
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr, transform)
... # doctest: +ELLIPSIS
array([ 0.2166600..., 0.1260477..., 0.0385506...])
```

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Kries': ...})
Supported chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMv07], [Lin09a], [WRC12b], [WRC12a], [Wik07a]

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}

Dataset

colour.adaptation

BRADFORD_CAT
BS_CAT
BS_PC_CAT
CAT02_BRILL_CAT
CAT02_CAT
CMCCAT2000_CAT
CMCCAT97_CAT
FAIRCHILD_CAT
SHARP_CAT
VON_KRIES_CAT
XYZ_SCALING_CAT

colour.adaptation.BRADFORD_CAT

colour.adaptation.BRADFORD_CAT = array([[0.8951, 0.2664, -0.1614], [-0.7502, 1.7135, 0.0367], [0.0389, -0.0...

`colour.adaptation.BS_CAT`

```
colour.adaptation.BS_CAT = array([[ 0.8752, 0.2787, -0.1539], [-0.8904, 1.8709, 0.0195], [-0.0061, 0.0162, 0.0113]])
```

`colour.adaptation.BS_PC_CAT`

```
colour.adaptation.BS_PC_CAT = array([[ 0.6489, 0.3915, -0.0404], [-0.3775, 1.3055, 0.072 ], [-0.0271, 0.0888, 0.0344]])
```

`colour.adaptation.CAT02_BRILL_CAT`

```
colour.adaptation.CAT02_BRILL_CAT = array([[ 0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [ 0. , 0. , 0. ]])
```

`colour.adaptation.CAT02_CAT`

```
colour.adaptation.CAT02_CAT = array([[ 0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [ 0.003 , 0.0136, 0.0044]])
```

`colour.adaptation.CMCCAT2000_CAT`

```
colour.adaptation.CMCCAT2000_CAT = array([[ 7.98200000e-01, 3.38900000e-01, -1.37100000e-01], [ -5.91800000e-02, 2.44900000e-01, 4.29600000e-02]])
```

`colour.adaptation.CMCCAT97_CAT`

```
colour.adaptation.CMCCAT97_CAT = array([[ 0.8951, -0.7502, 0.0389], [ 0.2664, 1.7135, 0.0685], [-0.1614, 0.0044, 0.0082]])
```

`colour.adaptation.FAIRCHILD_CAT`

```
colour.adaptation.FAIRCHILD_CAT = array([[ 0.8562, 0.3372, -0.1934], [-0.836 , 1.8327, 0.0033], [ 0.0357, -0.0044, 0.0082]])
```

`colour.adaptation.SHARP_CAT`

```
colour.adaptation.SHARP_CAT = array([[ 1.2694, -0.0988, -0.1706], [-0.8364, 1.8006, 0.0357], [ 0.0297, -0.0307, 0.0044]])
```

`colour.adaptation.VON_KRIES_CAT`

```
colour.adaptation.VON_KRIES_CAT = array([[ 0.40024, 0.7076 , -0.08081], [-0.2263 , 1.16532, 0.0457 ], [ 0. , 0. , 0. ]])
```

`colour.adaptation.XYZ_SCALING_CAT`

```
colour.adaptation.XYZ_SCALING_CAT = array([[ 1., 0., 0.], [ 0., 1., 0.], [ 0., 0., 1.]])
```

Ancillary Objects

`colour.adaptation`

<code>chromatic_adaptation_matrix_VonKries(XYZ_w, ...)</code>	Computes the <i>chromatic adaptation</i> matrix from test viewing conditions to reference viewing conditions.
---	---

colour.adaptation.chromatic_adaptation_matrix_VonKries

`colour.adaptation.chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr, transform='CAT02')`
 Computes the *chromatic adaptation* matrix from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of white-point.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of whitepoint.
- **transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns Chromatic adaptation matrix M_{cat} .

Return type ndarray

Raises `KeyError` – If chromatic adaptation method is not defined.

Notes

Domain	Scale - Reference	Scale - 1
XYZ_w	[0, 1]	[0, 1]
XYZ_wr	[0, 1]	[0, 1]

References

[Fai13b]

Examples

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr)
... # doctest: +ELLIPSIS
array([[ 1.0425738...,  0.0308910..., -0.0528125...],
       [ 0.0221934...,  1.0018566..., -0.0210737...],
       [-0.0011648..., -0.0034205...,  0.7617890...]])
```

Using Bradford method:

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> method = 'Bradford'
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr, method)
... # doctest: +ELLIPSIS
array([[ 1.0479297...,  0.0229468..., -0.0501922...],
       [ 0.0296278...,  0.9904344..., -0.0170738...],
       [-0.0092430...,  0.0150551...,  0.7518742...]])
```

Algebra

- *Extrapolation*
- *Interpolation*
- *Coordinates*
- *Geometry*
- *Matrix*
- *Random*
- *Regression*
- *Common*

Extrapolation

colour

<code>Extrapolator([interpolator, method, left, ...])</code>	Extrapolates the 1-D function of given interpolator.
--	--

colour.Extrapolator

```
class colour.Extrapolator(interpolator=None, method='Linear', left=None, right=None,
                           dtype=<class 'numpy.float64'>)
    Extrapolates the 1-D function of given interpolator.
```

The `colour.Extrapolator` class acts as a wrapper around a given *Colour* or *scipy* interpolator class instance with compatible signature. Two extrapolation methods are available:

- *Linear*: Linearly extrapolates given points using the slope defined by the interpolator boundaries ($xi[0]$, $xi[1]$) if $x < xi[0]$ and ($xi[-1]$, $xi[-2]$) if $x > xi[-1]$.
- *Constant*: Extrapolates given points by assigning the interpolator boundaries values $xi[0]$ if $x < xi[0]$ and $xi[-1]$ if $x > xi[-1]$.

Specifying the *left* and *right* arguments takes precedence on the chosen extrapolation method and will assign the respective *left* and *right* values to the given points.

Parameters

- **interpolator** (*object*) – Interpolator object.

- **method** (unicode, optional) – {'Linear', 'Constant'}, Extrapolation method.
- **left** (numeric, optional) – Value to return for $x < x_i[0]$.
- **right** (numeric, optional) – Value to return for $x > x_i[-1]$.
- **dtype** (type) – Data type used for internal conversions.

`__class__()`

Notes

- The interpolator must define x and y attributes.

References

[Sas], [WRC12d]

Examples

Extrapolating a single numeric variable:

```
>>> from colour.algebra import LinearInterpolator
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator)
>>> extrapolator(1)
-1.0
```

Extrapolating an *array_like* variable:

```
>>> extrapolator(np.array([6, 7, 8]))
array([ 4.,  5.,  6.])
```

Using the *Constant* extrapolation method:

```
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator, method='Constant')
>>> extrapolator(np.array([0.1, 0.2, 8, 9]))
array([ 1.,  1.,  3.,  3.])
```

Using defined *left* boundary and *Constant* extrapolation method:

```
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator, method='Constant', left=0)
>>> extrapolator(np.array([0.1, 0.2, 8, 9]))
array([ 0.,  0.,  3.,  3.])
```

`__init__(interpolator=None, method='Linear', left=None, right=None, dtype=<class 'numpy.float64'>)`

Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([interpolator, method, left, ...])</code>	Initialize self.
--	------------------

Attributes

<code>interpolator</code>	Getter and setter property for the <i>Colour</i> or <i>scipy</i> interpolator class instance.
<code>left</code>	Getter and setter property for left value to return for $x < xi[0]$.
<code>method</code>	Getter and setter property for the extrapolation method.
<code>right</code>	Getter and setter property for right value to return for $x > xi[-1]$.

Interpolation

colour

<code>KernelInterpolator(x, y[, window, kernel, ...])</code>	Kernel based interpolation of a 1-D function.
<code>NearestNeighbourInterpolator(*args, **kwargs)</code>	A nearest-neighbour interpolator.
<code>LinearInterpolator(x, y[, dtype])</code>	Linearly interpolates a 1-D function.
<code>NullInterpolator(x, y[, absolute_tolerance, ...])</code>	Performs 1-D function null interpolation, i.e.
<code>PchipInterpolator(x, y, *args, **kwargs)</code>	Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.
<code>SpragueInterpolator(x, y[, dtype])</code>	Constructs a fifth-order polynomial that passes through y dependent variable.
<code>lagrange_coefficients(r[, n])</code>	Computes the <i>Lagrange Coefficients</i> at given point r for degree n .
<code>TABLE_INTERPOLATION_METHODS</code>	Supported table interpolation methods.
<code>table_interpolation(V_xyz, table[, method])</code>	Performs interpolation of given V_{xyz} values using given interpolation table.

colour.KernelInterpolator

```
class colour.KernelInterpolator(x, y, window=3, kernel=<function kernel_lanczos>,
                                kernel_args=None, padding_args=None, dtype=<class
                                'numpy.float64'>)
```

Kernel based interpolation of a 1-D function.

The reconstruction of a continuous signal can be described as a linear convolution operation. Interpolation can be expressed as a convolution of the given discrete function $g(x)$ with some continuous interpolation kernel $k(w)$:

$$\hat{g}(w_0) = [k * g](w_0) = \sum_{x=-\infty}^{\infty} k(w_0 - x) \cdot g(x)$$

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.

- **window** (`int`, optional) – Width of the window in samples on each side.
- **kernel** (callable, optional) – Kernel to use for interpolation.
- **kernel_args** (`dict`, optional) – Arguments to use when calling the kernel.
- **padding_args** (`dict`, optional) – Arguments to use when padding *y* variable values with the `np.pad()` definition.
- **dtype** (`type`) – Data type used for internal conversions.

x
y
window
kernel
kernel_args
padding_args
__call__()

References

[BB09], [Wik05c]

Examples

Interpolating a single numeric variable:

```

>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = KernelInterpolator(x, y)
>>> f(0.5) # doctest: +ELLIPSIS
6.9411400...

```

Interpolating an *array_like* variable:

```

>>> f([0.25, 0.75]) # doctest: +ELLIPSIS
array([ 6.1806208...,  8.0823848...])

```

Using a different *lanczos* kernel:

```

>>> f = KernelInterpolator(x, y, kernel=kernel_sinc)
>>> f([0.25, 0.75]) # doctest: +ELLIPSIS
array([ 6.5147317...,  8.3965466...])

```

Using a different window size:

```

>>> f = KernelInterpolator(
...     x,
...     y,
...     window=16,
...     kernel=kernel_lanczos,
...     kernel_args={'a': 16})

```

(continues on next page)

(continued from previous page)

```
>>> f([0.25, 0.75]) # doctest: +ELLIPSIS
array([ 5.3961792...,  5.6521093...])
```

```
__init__(x, y, window=3, kernel=<function kernel_lanczos>, kernel_args=None,
padding_args=None, dtype=<class 'numpy.float64'>)
Initialize self. See help(type(self)) for accurate signature.
```

Methods

<code>__init__(x, y[, window, kernel, ...])</code>	Initialize self.
--	------------------

Attributes

<code>kernel</code>	Getter and setter property for the kernel callable.
<code>kernel_args</code>	Getter and setter property for the kernel call time arguments.
<code>padding_args</code>	Getter and setter property for the kernel call time arguments.
<code>window</code>	Getter and setter property for the window.
<code>x</code>	Getter and setter property for the independent x variable.
<code>y</code>	Getter and setter property for the dependent and already known y variable.

colour.NearestNeighbourInterpolator

class colour.NearestNeighbourInterpolator(*args, **kwargs)

A nearest-neighbour interpolator.

Other Parameters

- **x** (*array_like*) – Independent x variable values corresponding with y variable.
- **y** (*array_like*) – Dependent and already known y variable values to interpolate.
- **window** (*int, optional*) – Width of the window in samples on each side.
- **padding_args** (*dict, optional*) – Arguments to use when padding y variable values with the `np.pad()` definition.
- **dtype** (*type*) – Data type used for internal conversions.

```
__init__(*args, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes

kernel	Getter and setter property for the kernel callable.
kernel_args	Getter and setter property for the kernel call time arguments.
padding_args	Getter and setter property for the kernel call time arguments.
window	Getter and setter property for the window.
x	Getter and setter property for the independent x variable.
y	Getter and setter property for the dependent and already known y variable.

colour.LinearInterpolator

class colour.LinearInterpolator($x, y, dtype=<class\ 'numpy.float64'>$)
 Linearly interpolates a 1-D function.

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.
- **dtype** (type) – Data type used for internal conversions.

x
y
__call__()

Notes

- This class is a wrapper around `numpy.interp` definition.

Examples

Interpolating a single numeric variable:

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...              69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = LinearInterpolator(x, y)
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> f(0.5) # doctest: +ELLIPSIS
7.64...
```

Interpolating an `array_like` variable:

```
>>> f([0.25, 0.75])
array([ 6.7825,  8.5075])
```

__init__($x, y, dtype=<class\ 'numpy.float64'>$)
 Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__(x, y[, dtype])</code>	Initialize self.
--------------------------------------	------------------

Attributes

<code>x</code>	Getter and setter property for the independent x variable.
<code>y</code>	Getter and setter property for the dependent and already known y variable.

`colour.NullInterpolator`

class `colour.NullInterpolator`(*x*, *y*, *absolute_tolerance*=1e-06, *relative_tolerance*=1e-06, *default*=nan, *dtype*=<class 'numpy.float64'>)
Performs 1-D function null interpolation, i.e. a call within given tolerances will return existing y variable values and default if outside tolerances.

Parameters

- **x** (ndarray) – Independent x variable values corresponding with y variable.
- **y** (ndarray) – Dependent and already known y variable values to interpolate.
- **absolute_tolerance** (numeric, optional) – Absolute tolerance.
- **relative_tolerance** (numeric, optional) – Relative tolerance.
- **default** (numeric, optional) – Default value for interpolation outside tolerances.
- **dtype** (type) – Data type used for internal conversions.

x
y
relative_tolerance
absolute_tolerance
default
__call__()

Examples

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,  
...              69.5900, 27.8007, 86.0500])  
>>> x = np.arange(len(y))  
>>> f = NullInterpolator(x, y)  
>>> f(0.5)  
nan  
>>> f(1.0) # doctest: +ELLIPSIS  
9.3699999...  
>>> f = NullInterpolator(x, y, absolute_tolerance=0.01)  
>>> f(1.01) # doctest: +ELLIPSIS  
9.3699999...
```

```
__init__(x, y, absolute_tolerance=1e-06, relative_tolerance=1e-06, default=nan, dtype=<class
        'numpy.float64'>)
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

<code>__init__(x, y[, absolute_tolerance, ...])</code>	Initialize self.
--	------------------

Attributes

<code>absolute_tolerance</code>	Getter and setter property for the absolute tolerance.
<code>default</code>	Getter and setter property for the default value for call outside tolerances.
<code>relative_tolerance</code>	Getter and setter property for the relative tolerance.
<code>x</code>	Getter and setter property for the independent x variable.
<code>y</code>	Getter and setter property for the dependent and already known y variable.

colour.PchipInterpolator

class colour.**PchipInterpolator**(x, y, *args, **kwargs)
 Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.

y

Notes

- This class is a wrapper around `scipy.interpolate.PchipInterpolator` class.

```
__init__(x, y, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

<code>__init__(x, y, *args, **kwargs)</code>	Initialize self.
<code>antiderivative([nu])</code>	Construct a new piecewise polynomial representing the antiderivative.
<code>construct_fast(c, x[, extrapolate, axis])</code>	Construct the piecewise polynomial without making checks.
<code>derivative([nu])</code>	Construct a new piecewise polynomial representing the derivative.
<code>extend(c, x[, right])</code>	Add additional breakpoints and coefficients to the polynomial.

Continued on next page

Table 24 – continued from previous page

<code>from_derivatives(xi, yi[, orders, extrapolate])</code>	Construct a piecewise polynomial in the Bernstein basis, compatible with the specified values and derivatives at breakpoints.
<code>from_power_basis(pp[, extrapolate])</code>	Construct a piecewise polynomial in Bernstein basis from a power basis polynomial.
<code>integrate(a, b[, extrapolate])</code>	Compute a definite integral over a piecewise polynomial.
<code>roots()</code>	Return the roots of the interpolated function.

Attributes

<code>axis</code>	
<code>c</code>	
<code>extrapolate</code>	
<code>x</code>	
<code>y</code>	Getter and setter property for the dependent and already known y variable.

colour.SpragueInterpolator

class colour.SpragueInterpolator($x, y, dtype=<class\ 'numpy.float64'>$)

Constructs a fifth-order polynomial that passes through y dependent variable.

Sprague (1880) method is recommended by the *CIE* for interpolating functions having a uniformly spaced independent variable.

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.
- **dtype** (type) – Data type used for internal conversions.

x

y

`__call__()`

Notes

- The minimum number k of data points required along the interpolation axis is $k = 6$.

References

[CIET13805b], [WRC12e]

Examples

Interpolating a single numeric variable:


```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = SpragueInterpolator(x, y)
>>> f(0.5) # doctest: +ELLIPSIS
7.2185025...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75]) # doctest: +ELLIPSIS
array([ 6.7295161...,  7.8140625...])
```

__init__(*x*, *y*, *dtype*=<class 'numpy.float64'>)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (<i>x</i> , <i>y</i> [, <i>dtype</i>])	Initialize self.
--	------------------

Attributes

SPRAGUE_C_COEFFICIENTS	Defines the coefficients used to generate extra points for boundaries interpolation.
<i>x</i>	Getter and setter property for the independent <i>x</i> variable.
<i>y</i>	Getter and setter property for the dependent and already known <i>y</i> variable.

colour.lagrange_coefficients

colour.lagrange_coefficients(*r*, *n*=4)
Computes the *Lagrange Coefficients* at given point *r* for degree *n*.

Parameters

- **r** (numeric) – Point to get the *Lagrange Coefficients* at.
- **n** (int, optional) – Degree of the *Lagrange Coefficients* being calculated.

Returns

Return type ndarray

References

[Fai85], [Wik03b]

Examples

```
>>> lagrange_coefficients(0.1)
array([ 0.8265,  0.2755, -0.1305,  0.0285])
```

colour.TABLE_INTERPOLATION_METHODS

`colour.TABLE_INTERPOLATION_METHODS = CaseInsensitiveMapping({'Trilinear': ..., 'Tetrahedral': ...})`
Supported table interpolation methods.

References

[Boub], [Kir06]

TABLE_INTERPOLATION_METHODS [CaseInsensitiveMapping] {'Trilinear', 'Tetrahedral'}

colour.table_interpolation

`colour.table_interpolation(V_xyz, table, method='Trilinear')`
Performs interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_xyz** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional ($N \times N \times N \times 3$) interpolation table.
- **method** (unicode, optional) – {'Trilinear', 'Tetrahedral'}, Interpolation method.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Boub], [Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__), '..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'ColourCorrect.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz) # doctest: +ELLIPSIS
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation(V_xyz, table) # doctest: +ELLIPSIS
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
>>> table_interpolation(V_xyz, table, method='Tetrahedral')
... # doctest: +ELLIPSIS
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
```

(continues on next page)

(continued from previous page)

```
[ 0.5105603..., 0.6466722..., 0.1077296...],
 [ 1.1178206..., 0.1762039..., 0.2209534...]])
```

Interpolation Kernels

colour

<code>kernel_nearest_neighbour(x)</code>	Returns the <i>nearest-neighbour</i> kernel evaluated at given samples.
<code>kernel_linear(x)</code>	Returns the <i>linear</i> kernel evaluated at given samples.
<code>kernel_sinc(x[, a])</code>	Returns the <i>sinc</i> kernel evaluated at given samples.
<code>kernel_lanczos(x[, a])</code>	Returns the <i>lanczos</i> kernel evaluated at given samples.
<code>kernel_cardinal_spline(x[, a, b])</code>	Returns the <i>cardinal spline</i> kernel evaluated at given samples.

colour.kernel_nearest_neighbour

colour.**kernel_nearest_neighbour**(x)

Returns the *nearest-neighbour* kernel evaluated at given samples.

Parameters x (array_like) – Samples at which to evaluate the *nearest-neighbour* kernel.

Returns The *nearest-neighbour* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_nearest_neighbour(np.linspace(0, 1, 10))
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

colour.kernel_linear

colour.**kernel_linear**(x)

Returns the *linear* kernel evaluated at given samples.

Parameters x (array_like) – Samples at which to evaluate the *linear* kernel.

Returns The *linear* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_linear(np.linspace(0, 1, 10)) # doctest: +ELLIPSIS
array([ 1.          ,  0.8888888...,  0.7777777...,  0.6666666...,  0.5555555...,
        0.4444444...,  0.3333333...,  0.2222222...,  0.1111111...,  0.          ])
```

colour.kernel_sinc

colour.**kernel_sinc**(*x*, *a*=3)

Returns the *sinc* kernel evaluated at given samples.

Parameters

- **x** (array_like) – Samples at which to evaluate the *sinc* kernel.
- **a** (int, optional) – Size of the *sinc* kernel.

Returns The *sinc* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_sinc(np.linspace(0, 1, 10)) # doctest: +ELLIPSIS
array([ 1.0000000...e+00,  9.7981553...e-01,  9.2072542...e-01,
        8.2699334...e-01,  7.0531659...e-01,  5.6425327...e-01,
        4.1349667...e-01,  2.6306440...e-01,  1.2247694...e-01,
        3.8981718...e-17])
```

colour.kernel_lanczos

colour.**kernel_lanczos**(*x*, *a*=3)

Returns the *lanczos* kernel evaluated at given samples.

Parameters

- **x** (array_like) – Samples at which to evaluate the *lanczos* kernel.
- **a** (int, optional) – Size of the *lanczos* kernel.

Returns The *lanczos* kernel evaluated at given samples.

Return type ndarray

References

[Wik05c]

Examples

```
>>> kernel_lanczos(np.linspace(0, 1, 10)) # doctest: +ELLIPSIS
array([ 1.0000000...e+00,  9.7760615...e-01,  9.1243770...e-01,
        8.1030092...e-01,  6.8012706...e-01,  5.3295773...e-01,
        3.8071690...e-01,  2.3492839...e-01,  1.0554054...e-01,
        3.2237621...e-17])
```

colour.kernel_cardinal_spline

colour.**kernel_cardinal_spline**(*x*, *a*=0.5, *b*=0.0)

Returns the *cardinal spline* kernel evaluated at given samples.

Notable *cardinal spline* *a* and *b* parameterizations:

- *Catmull-Rom*: ($a = 0.5, b = 0$)
- *Cubic B-Spline*: ($a = 0, b = 1$)
- *Mitchell-Netravalli*: ($a = \frac{1}{3}, b = \frac{1}{3}$)

Parameters

- **x** (array_like) – Samples at which to evaluate the *cardinal spline* kernel.
- **a** (int, optional) – *a* control parameter.
- **b** (int, optional) – *b* control parameter.

Returns The *cardinal spline* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_cardinal_spline(np.linspace(0, 1, 10)) # doctest: +ELLIPSIS
array([ 1.          ,  0.9711934...,  0.8930041...,  0.7777777...,  0.6378600...,
        0.4855967...,  0.3333333...,  0.1934156...,  0.0781893...,  0.          ])
```

Ancillary Objects

colour.algebra

<code>table_interpolation_trilinear(V_xyz, table)</code>	Performs trilinear interpolation of given V_{xyz} values using given interpolation table.
<code>table_interpolation_tetrahedral(V_xyz, table)</code>	Performs tetrahedral interpolation of given V_{xyz} values using given interpolation table.

colour.algebra.table_interpolation_trilinear

colour.algebra.**table_interpolation_trilinear**(V_{xyz} , table)

Performs trilinear interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_{xyz}** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional ($N \times N \times N \times 3$) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Boub]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__), '..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'ColourCorrect.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz) # doctest: +ELLIPSIS
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_trilinear(V_xyz, table) # doctest: +ELLIPSIS
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
```

colour.algebra.table_interpolation_tetrahedral

colour.algebra.**table_interpolation_tetrahedral**(V_{xyz} , table)

Performs tetrahedral interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_{xyz}** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional ($N \times N \times N \times 3$) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__), '..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'ColourCorrect.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz) # doctest: +ELLIPSIS
[[ 0.9670298... 0.7148159... 0.9762744...]
 [ 0.5472322... 0.6977288... 0.0062302...]
 [ 0.9726843... 0.2160895... 0.2529823...]]
>>> table_interpolation_tetrahedral(V_xyz, table) # doctest: +ELLIPSIS
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Coordinates

colour.algebra

<code>cartesian_to_spherical(a)</code>	Transforms given Cartesian coordinates array xyz to Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth).
<code>spherical_to_cartesian(a)</code>	Transforms given Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to Cartesian coordinates array xyz .
<code>cartesian_to_polar(a)</code>	Transforms given Cartesian coordinates array xy to Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).
<code>polar_to_cartesian(a)</code>	Transforms given Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to Cartesian coordinates array xy .
<code>cartesian_to_cylindrical(a)</code>	Transforms given Cartesian coordinates array xyz to Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height).
<code>cylindrical_to_cartesian(a)</code>	Transforms given Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height) to Cartesian coordinates array xyz .

colour.algebra.cartesian_to_spherical

colour.algebra.`cartesian_to_spherical(a)`

Transforms given Cartesian coordinates array xyz to Spherical coordinates array $\rho\theta\phi$ (radial distance,

inclination or elevation and azimuth).

Parameters *a* (array_like) – Cartesian coordinates array xyz to transform.

Returns Spherical coordinates array $\rho\theta\phi$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_spherical(a) # doctest: +ELLIPSIS
array([ 6.7823299...,  1.0857465...,  0.3217505...])
```

colour.algebra.spherical_to_cartesian

colour.algebra.**spherical_to_cartesian**(*a*)

Transforms given Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to Cartesian coordinates array xyz .

Parameters *a* (array_like) – Spherical coordinates array $\rho\theta\phi$ to transform.

Returns Cartesian coordinates array xyz .

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([6.78232998, 1.08574654, 0.32175055])
>>> spherical_to_cartesian(a) # doctest: +ELLIPSIS
array([ 3.          ,  0.9999999...,  6.          ])
```

colour.algebra.cartesian_to_polar

colour.algebra.**cartesian_to_polar**(*a*)

Transforms given Cartesian coordinates array xy to Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).

Parameters *a* (array_like) – Cartesian coordinates array xy to transform.

Returns Polar coordinates array $\rho\phi$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1])
>>> cartesian_to_polar(a) # doctest: +ELLIPSIS
array([ 3.1622776...,  0.3217505...])
```

colour.algebra.polar_to_cartesian

colour.algebra.**polar_to_cartesian**(a)

Transforms given Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to Cartesian coordinates array xy .

Parameters a (array_like) – Polar coordinates array $\rho\phi$ to transform.

Returns Cartesian coordinates array xy .

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3.16227766, 0.32175055])
>>> polar_to_cartesian(a) # doctest: +ELLIPSIS
array([ 3.          ,  0.9999999...])
```

colour.algebra.cartesian_to_cylindrical

colour.algebra.**cartesian_to_cylindrical**(a)

Transforms given Cartesian coordinates array xyz to Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height).

Parameters a (array_like) – Cartesian coordinates array xyz to transform.

Returns Cylindrical coordinates array $\rho\phi z$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_cylindrical(a) # doctest: +ELLIPSIS
array([ 3.1622776..., 0.3217505..., 6.      ])
```

colour.algebra.cylindrical_to_cartesian

colour.algebra.cylindrical_to_cartesian(*a*)

Transforms given Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height) to Cartesian coordinates array xyz .

Parameters *a* (array_like) – Cylindrical coordinates array $\rho\phi z$ to transform.

Returns Cartesian coordinates array xyz .

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3.16227766, 0.32175055, 6.00000000])
>>> cylindrical_to_cartesian(a) # doctest: +ELLIPSIS
array([ 3.      , 0.9999999..., 6.      ])
```

Geometry

colour.algebra

<code>normalise_vector(a)</code>	Normalises given vector <i>a</i> .
<code>euclidean_distance(a, b)</code>	Returns the euclidean distance between point arrays <i>a</i> and <i>b</i> .
<code>extend_line_segment(a, b[, distance])</code>	Extends the line segment defined by point arrays <i>a</i> and <i>b</i> by given distance and return the new end point.
<code>intersect_line_segments(l_1, l_2)</code>	Computes <i>l</i> ₁ line segments intersections with <i>l</i> ₂ line segments.
<code>ellipse_coefficients_general_form(coefficients)</code>	Returns the general form ellipse coefficients from given canonical form ellipse coefficients.
<code>ellipse_coefficients_canonical_form(coefficients)</code>	Returns the canonical form ellipse coefficients from given general form ellipse coefficients.
<code>point_at_angle_on_ellipse(phi, coefficients)</code>	Returns the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.
<code>ELLIPSE_FITTING_METHODS</code>	Supported ellipse fitting methods.

Continued on next page

Table 31 – continued from previous page

<code>ellipse_fitting(a[, method])</code>	Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array <i>a</i> using given method.
---	---

`colour.algebra.normalise_vector`

`colour.algebra.normalise_vector(a)`

Normalises given vector *a*.

Parameters *a* (array_like) – Vector *a* to normalise.

Returns Normalised vector *a*.

Return type ndarray

Examples

```
>>> a = np.array([0.20654008, 0.12197225, 0.05136952])
>>> normalise_vector(a) # doctest: +ELLIPSIS
array([ 0.8419703..., 0.4972256..., 0.2094102...])
```

`colour.algebra.euclidean_distance`

`colour.algebra.euclidean_distance(a, b)`

Returns the euclidean distance between point arrays *a* and *b*.

Parameters

- *a* (array_like) – Point array *a*.
- *b* (array_like) – Point array *b*.

Returns Euclidean distance.

Return type numeric or ndarray

Examples

```
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> euclidean_distance(a, b) # doctest: +ELLIPSIS
451.7133019...
```

`colour.algebra.extend_line_segment`

`colour.algebra.extend_line_segment(a, b, distance=1)`

Extends the line segment defined by point arrays *a* and *b* by given distance and return the new end point.

Parameters

- *a* (array_like) – Point array *a*.

- **b** (array_like) – Point array b .
- **distance** (numeric, optional) – Distance to extend the line segment.

Returns New end point.

Return type ndarray

References

[Sae]

Notes

- Input line segment points coordinates are 2d coordinates.

Examples

```
>>> a = np.array([0.95694934, 0.13720932])
>>> b = np.array([0.28382835, 0.60608318])
>>> extend_line_segment(a, b) # doctest: +ELLIPSIS
array([-0.5367248..., 1.1776534...])
```

colour.algebra.intersect_line_segments

colour.algebra.**intersect_line_segments**(l_1 , l_2)

Computes l_1 line segments intersections with l_2 line segments.

Parameters

- **l_1** (array_like) – l_1 line segments array, each row is a line segment such as (x_1, y_1, x_2, y_2) where (x_1, y_1) and (x_2, y_2) are respectively the start and end points of l_1 line segments.
- **l_2** (array_like) – l_2 line segments array, each row is a line segment such as (x_3, y_3, x_4, y_4) where (x_3, y_3) and (x_4, y_4) are respectively the start and end points of l_2 line segments.

Returns Line segments intersections specification.

Return type *LineSegmentsIntersections_Specification*

References

[Boua], [Erda]

Notes

- Input line segments points coordinates are 2d coordinates.

Examples

```
>>> l_1 = np.array(
...     [[0.15416284, 0.7400497],
...      [0.26331502, 0.53373939]],
...     [[0.01457496, 0.91874701],
...      [0.90071485, 0.03342143]])
... )
>>> l_2 = np.array(
...     [[0.95694934, 0.13720932],
...      [0.28382835, 0.60608318]],
...     [[0.94422514, 0.85273554],
...      [0.00225923, 0.52122603]],
...     [[0.55203763, 0.48537741],
...      [0.76813415, 0.16071675]])
... )
>>> s = intersect_line_segments(l_1, l_2)
>>> s.xy # doctest: +ELLIPSIS
array([[ nan,      nan],
       [ 0.2279184..., 0.6006430...],
       [ nan,      nan]],
<BLANKLINE>
       [[ 0.4281451..., 0.5055568...],
        [ 0.3056055..., 0.6279838...],
        [ 0.7578749..., 0.1761301...]])
>>> s.intersect
array([[False,  True, False],
       [ True,  True,  True]], dtype=bool)
>>> s.parallel
array([[False, False, False],
       [False, False, False]], dtype=bool)
>>> s.coincident
array([[False, False, False],
       [False, False, False]], dtype=bool)
```

colour.algebra.ellipse_coefficients_general_form

colour.algebra.ellipse_coefficients_general_form(coefficients)

Returns the general form ellipse coefficients from given canonical form ellipse coefficients.

The canonical form ellipse coefficients are as follows: the center coordinates x_c and y_c , semi-major axis length a_a , semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a .

Parameters coefficients (array_like) – Canonical form ellipse coefficients.

Returns General form ellipse coefficients.

Return type ndarray

References

[Wik]

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> ellipse_coefficients_general_form(coefficients)
array([ 2.5, -3. ,  2.5, -1. , -1. , -3.5])
```

colour.algebra.ellipse_coefficients_canonical_form

colour.algebra.ellipse_coefficients_canonical_form(coefficients)

Returns the canonical form ellipse coefficients from given general form ellipse coefficients.

The general form ellipse coefficients are the coefficients of the implicit second-order polynomial/quadratic curve expressed as follows:

$$F(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters coefficients (array_like) – General form ellipse coefficients.

Returns Canonical form ellipse coefficients.

Return type ndarray

References

[Wik]

Examples

```
>>> coefficients = np.array([ 2.5, -3.0,  2.5, -1.0, -1.0, -3.5])
>>> ellipse_coefficients_canonical_form(coefficients)
array([ 0.5,  0.5,  2. ,  1. , 45. ])
```

colour.algebra.point_at_angle_on_ellipse

colour.algebra.point_at_angle_on_ellipse(phi, coefficients)

Returns the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.

Parameters

- **phi** (array_like) – Point at angle ϕ in degrees to retrieve the coordinates of.
- **coefficients** (array_like) – General form ellipse coefficients as follows: the center coordinates x_c and y_c , semi-major axis length a_a , semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a .

Returns Coordinates of the point at angle ϕ

Return type ndarray

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> point_at_angle_ellipse(45, coefficients) # doctest: +ELLIPSIS
array([ 1.,  2.]
```

colour.algebra.ELLIPSE_FITTING_METHODS

colour.algebra.ELLIPSE_FITTING_METHODS = CaseInsensitiveMapping({'Halir 1998': ...})
Supported ellipse fitting methods.

References

[HF98]

ELLIPSE_FITTING_METHODS [CaseInsensitiveMapping] {'Halir 1998'}

colour.algebra.ellipse_fitting

colour.algebra.ellipse_fitting(*a*, *method*='Halir 1998')

Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array *a* using given method.

The implicit second-order polynomial is expressed as follows:

$$F(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters

- **a** (array_like) – Point array *a* to be fitted.
- **method** (unicode, optional) – {'Halir 1998'}, Computation method.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array *a*.

Return type ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting(a) # doctest: +ELLIPSIS
array([ 0.2425356...,  0.          ,  0.9701425...,  0.          ,  0.          ,
        -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting(a))
array([-0., -0.,  2.,  1.,  0.]
```

Ancillary Objects

`colour.algebra`

<code>LineSegmentsIntersections_Specification</code>	Defines the specification for intersection of line segments l_1 and l_2 returned by <code>colour.algebra.intersect_line_segments()</code> definition.
<code>ellipse_fitting_Halir1998(a)</code>	Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using <i>Halir and Flusser (1998)</i> method.

`colour.algebra.LineSegmentsIntersections_Specification`

class `colour.algebra.LineSegmentsIntersections_Specification`

Defines the specification for intersection of line segments l_1 and l_2 returned by `colour.algebra.intersect_line_segments()` definition.

Parameters

- **xy** (array_like) – Array of l_1 and l_2 line segments intersections coordinates. Non existing segments intersections coordinates are set with *np.nan*.
- **intersect** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 intersect.
- **parallel** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 are parallel.
- **coincident** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 are coincident.

Create new instance of `LineSegmentsIntersections_Specification(xy, intersect, parallel, coincident)`

__init__()

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>coincident</code>	Alias for field number 3
<code>intersect</code>	Alias for field number 1
<code>parallel</code>	Alias for field number 2
<code>xy</code>	Alias for field number 0

`colour.algebra.ellipse_fitting_Halir1998`

`colour.algebra.ellipse_fitting_Halir1998(a)`

Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using *Halir and Flusser (1998)* method.

The implicit second-order polynomial is expressed as follows:

$$:math:\text{'F\left(x, y\right)' = ax^2 + bxy + cy^2 + dx + ey + f = 0'}$$

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters *a* (array_like) – Point array *a* to be fitted.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array *a*.

Return type ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting_Halir1998(a) # doctest: +ELLIPSIS
array([ 0.2425356..., 0.          , 0.9701425..., 0.          , 0.          ,
       -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting_Halir1998(a))
array([-0., -0., 2., 1., 0.]
```

Matrix

colour.algebra

<code>is_identity(a[, n])</code>	Returns if <i>a</i> array is an identity matrix.
----------------------------------	--

colour.algebra.is_identity

colour.algebra.**is_identity**(*a*, *n*=3)
Returns if *a* array is an identity matrix.

Parameters

- *a* (array_like, (N)) – Variable *a* to test.
- *n* (int, optional) – Matrix dimension.

Returns Is identity matrix.

Return type bool

Examples

```
>>> is_identity(np.array([1, 0, 0, 0, 1, 0, 0, 0, 1]).reshape(3, 3))
True
```

(continues on next page)

(continued from previous page)

```
>>> is_identity(np.array([1, 2, 0, 0, 1, 0, 0, 0, 1]).reshape(3, 3))
False
```

Random

colour.algebra

<code>random_triplet_generator(size[, limits, ...])</code>	Returns a generator yielding random triplets.
--	---

colour.algebra.random_triplet_generator

colour.algebra.**random_triplet_generator**(size, limits=array([[0, 1], [0, 1], [0, 1]]), random_state=<mtrand.RandomState object>)

Returns a generator yielding random triplets.

Parameters

- **size** (int) – Generator size.
- **limits** (array_like, (3, 2)) – Random values limits on each triplet axis.
- **random_state** (RandomState) – Mersenne Twister pseudo-random number generator.

Returns Random triplets generator.

Return type generator

Notes

- The test is assuming that np.random.RandomState() definition will return the same sequence no matter which OS or Python version is used. There is however no formal promise about the prng sequence reproducibility of either Python or Numpy implementations, see [Lau12].

Examples

```
>>> from pprint import pprint
>>> prng = np.random.RandomState(4)
>>> random_triplet_generator(10, random_state=prng)
... # doctest: +ELLIPSIS
array([[ 0.9670298...,  0.7793829...,  0.4361466...],
       [ 0.5472322...,  0.1976850...,  0.9489773...],
       [ 0.9726843...,  0.8629932...,  0.7863059...],
       [ 0.7148159...,  0.9834006...,  0.8662893...],
       [ 0.6977288...,  0.1638422...,  0.1731654...],
       [ 0.2160895...,  0.5973339...,  0.0749485...],
       [ 0.9762744...,  0.0089861...,  0.6007427...],
       [ 0.0062302...,  0.3865712...,  0.1679721...],
       [ 0.2529823...,  0.0441600...,  0.7333801...],
       [ 0.4347915...,  0.9566529...,  0.4084438...]])
```

Regression

colour.algebra

<code>least_square_mapping_MoorePenrose(y, x)</code>	Computes the <i>least-squares</i> mapping from dependent variable y to independent variable x using <i>Moore-Penrose</i> inverse.
--	---

colour.algebra.least_square_mapping_MoorePenrose

colour.algebra.**least_square_mapping_MoorePenrose**(y, x)
Computes the *least-squares* mapping from dependent variable y to independent variable x using *Moore-Penrose* inverse.

Parameters

- **y** (array_like) – Dependent and already known y variable.
- **x** (array_like, optional) – Independent x variable(s) values corresponding with y variable.

Returns *Least-squares* mapping.

Return type ndarray

References

[FMH15]

Examples

```
>>> prng = np.random.RandomState(2)
>>> y = prng.random_sample((24, 3))
>>> x = y + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> least_square_mapping_MoorePenrose(y, x) # doctest: +ELLIPSIS
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

Common

colour.algebra

<code>is_spow_enabled()</code>	Returns whether <i>Colour</i> safe / symmetrical power function is enabled.
<code>set_spow_enable(enable)</code>	Sets <i>Colour</i> safe / symmetrical power function enabled state.
<code>spow_enable(enable)</code>	A context manager and decorator temporarily setting <i>Colour</i> safe / symmetrical power function enabled state.

Continued on next page

Table 38 – continued from previous page

<code>spow(a, p)</code>	Raises given array a to the power p as follows: $\text{sign}(a) * a ^p$.
-------------------------	---

`colour.algebra.is_spow_enabled`

`colour.algebra.is_spow_enabled()`

Returns whether *Colour* safe / symmetrical power function is enabled.

Returns Whether *Colour* safe / symmetrical power function is enabled.

Return type `bool`

Examples

```
>>> with spow_enable(False):
...     is_spow_enabled()
False
>>> with spow_enable(True):
...     is_spow_enabled()
True
```

`colour.algebra.set_spow_enable`

`colour.algebra.set_spow_enable(enable)`

Sets *Colour* safe / symmetrical power function enabled state.

Parameters `enable` (`bool`) – Whether to enable *Colour* safe / symmetrical power function.

Examples

```
>>> with spow_enable(is_spow_enabled()):
...     print(is_spow_enabled())
...     set_spow_enable(False)
...     print(is_spow_enabled())
True
False
```

`colour.algebra.spow_enable`

class `colour.algebra.spow_enable(enable)`

A context manager and decorator temporarily setting *Colour* safe / symmetrical power function enabled state.

Parameters `enable` (`bool`) – Whether to enable or disable *Colour* safe / symmetrical power function.

`__init__`(`enable`)

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__(enable)</code>	Initialize self.
-------------------------------	------------------

`colour.algebra.spow`

`colour.algebra.spow(a, p)`

Raises given array a to the power p as follows: $\text{sign}(a) * |a|^p$.

This avoids NaNs generation when array a is negative and the power p is fractional.

Parameters

- **a** (numeric or array_like) – Array a .
- **p** (numeric or array_like) – Power p .

Returns Array a safely raised to the power p .

Return type numeric or ndarray

Examples

```
>>> np.power(-2, 0.15)
nan
>>> spow(-2, 0.15) # doctest: +ELLIPSIS
-1.1095694...
>>> spow(0, 0)
0.0
```

Colour Appearance Models

- *ATD (1995)*
- *CIECAM02*
- *CAM16*
- *Hunt*
- *LLAB(l : c)*
- *Nayatani (1995)*
- *RLAB*

ATD (1995)

`colour`

<code>XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2[, sigma])</code>	Computes the <i>ATD (1995)</i> colour vision model correlates.
<code>ATD95_Specification</code>	Defines the <i>ATD (1995)</i> colour vision model specification.

colour.XYZ_to_ATD95

`colour.XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2, sigma=300)`

Computes the *ATD (1995)* colour vision model correlates.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of test sample / stimulus.
- **XYZ_0** (array_like) – CIE XYZ tristimulus values of reference white.
- **Y_0** (numeric or array_like) – Absolute adapting field luminance in cd/m^2 .
- **k_1** (numeric or array_like) – Application specific weight k_1 .
- **k_2** (numeric or array_like) – Application specific weight k_2 .
- **sigma** (numeric or array_like, optional) – Constant σ varied to predict different types of data.

Returns *ATD (1995)* colour vision model specification.

Return type *ATD95_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
ATD95_Specification.h	[0, 360]	[0, 1]

- For unrelated colors, there is only self-adaptation and k_1 is set to 1.0 while k_2 is set to 0.0. For related colors such as typical colorimetric applications, k_1 is set to 0.0 and k_2 is set to a value between 15 and 50 (*Guth, 1995*).

References

[Fai13a], [Gut95]

Examples

```

>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_0 = np.array([95.05, 100.00, 108.88])
>>> Y_0 = 318.31
>>> k_1 = 0.0
>>> k_2 = 50.0
>>> XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2) # doctest: +ELLIPSIS
ATD95_Specification(h=1.9089869..., C=1.2064060..., Q=0.1814003..., A_1=0.1787931... T_1=0.
↪0286942..., D_1=0.0107584..., A_2=0.0192182..., T_2=0.0205377..., D_2=0.0107584...)

```

colour.ATD95_Specification

class colour.ATD95_Specification

Defines the *ATD (1995)* colour vision model specification.

This specification has field names consistent with the remaining colour appearance models in *colour*. appearance but diverge from *Fairchild (2013)* reference.

Parameters

- **h** (numeric or array_like) – Hue angle H in degrees.
- **C** (numeric or array_like) – Correlate of saturation C . *Guth (1995)* incorrectly uses the terms saturation and chroma interchangeably. However, C is here a measure of saturation rather than chroma since it is measured relative to the achromatic response for the stimulus rather than that of a similarly illuminated white.
- **Q** (numeric or array_like) – Correlate of brightness Br .
- **A_1** (numeric or array_like) – First stage A_1 response.
- **T_1** (numeric or array_like) – First stage T_1 response.
- **D_1** (numeric or array_like) – First stage D_1 response.
- **A_2** (numeric or array_like) – Second stage A_2 response.
- **T_2** (numeric or array_like) – Second stage A_2 response.
- **D_2** (numeric or array_like) – Second stage D_2 response.

Notes

- This specification is the one used in the current model implementation.

References

[Fai13a], [Gut95]

Create new instance of `ATD95_Specification(h, C, Q, A_1, T_1, D_1, A_2, T_2, D_2)`

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

A_1	Alias for field number 3
A_2	Alias for field number 6
C	Alias for field number 1
D_1	Alias for field number 5
D_2	Alias for field number 8
Q	Alias for field number 2
T_1	Alias for field number 4
T_2	Alias for field number 7
h	Alias for field number 0

CIECAM02

colour

<code>XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b[, ...])</code>	Computes the <i>CIECAM02</i> colour appearance model correlates from given <i>CIE XYZ</i> tristimulus values.
<code>CIECAM02_to_XYZ(CIECAM02_specification, ...)</code>	Converts <i>CIECAM02</i> specification to <i>CIE XYZ</i> tristimulus values.
<code>CIECAM02_Specification</code>	Defines the <i>CIECAM02</i> colour appearance model specification.
<code>CIECAM02_VIEWING_CONDITIONS</code>	Reference <i>CIECAM02</i> colour appearance model viewing conditions.

colour.XYZ_to_CIECAM02

`colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1), discount_illuminant=False)`
 Computes the *CIECAM02* colour appearance model correlates from given *CIE XYZ* tristimulus values.
 This is the *forward* implementation.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field luminance L_A in cd/m^2 , (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2 .
- **surround** (`CIECAM02_InductionFactors`, optional) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *CIECAM02* colour appearance model specification.

Return type *CIECAM02_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
CIECAM02_specification.h	[0, 360]	[0, 1]
CIECAM02_specification.H	[0, 360]	[0, 1]

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround) # doctest: +ELLIPSIS
CIECAM02_Specification(J=41.7310911..., C=0.1047077..., h=219.0484326..., s=2.3603053..., Q=195.
↪3713259..., M=0.1088421..., H=278.0607358..., HC=None)
```

colour.CIECAM02_to_XYZ

`colour.CIECAM02_to_XYZ(CIECAM02_specification, XYZ_w, L_A, Y_b, surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1), discount_illuminant=False)`

Converts *CIECAM02* specification to *CIE XYZ* tristimulus values.

This is the *reverse* implementation.

Parameters

- **CIECAM02_specification** (*CIECAM02_Specification*) – *CIECAM02* colour appearance model specification. Correlate of *Lightness J*, correlate of *chroma C* or correlate of *colourfulness M* and hue angle *h* in degrees must be specified, e.g. *JCh* or *JMh*.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field luminance L_A in cd/m^2 , (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2 .
- **surround** (*CIECAM02_InductionFactors*, optional) – Surround viewing conditions.
- **discount_illuminant** (bool, optional) – Discount the illuminant.

Returns XYZ – CIE XYZ tristimulus values.

Return type ndarray

Raises `ValueError` – If neither *C* or *M* correlates have been defined in the `CIECAM02_specification` argument.

Warning: The output range of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
<code>CIECAM02_specification.h</code>	[0, 360]	[0, 1]
<code>CIECAM02_specification.H</code>	[0, 360]	[0, 1]
<code>XYZ_w</code>	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- `CIECAM02_specification` can also be passed as a compatible argument to `colour.utilities.as_nametuple()` definition.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Examples

```
>>> specification = CIECAM02_Specification(J=41.731091132513917,
...                                         C=0.104707757171031,
...                                         h=219.048432658311780)
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CIECAM02_to_XYZ(specification, XYZ_w, L_A, Y_b) # doctest: +ELLIPSIS
array([ 19.01..., 20..., 21.78...])
```

colour.CIECAM02_Specification

class colour.CIECAM02_Specification

Defines the *CIECAM02* colour appearance model specification.

Parameters

- *J* (numeric or array_like) – Correlate of *Lightness J*.
- *C* (numeric or array_like) – Correlate of *chroma C*.
- *h* (numeric or array_like) – *Hue* angle *h* in degrees.
- *s* (numeric or array_like) – Correlate of *saturation s*.

- **Q** (numeric or array_like) – Correlate of *brightness* Q .
- **M** (numeric or array_like) – Correlate of *colourfulness* M .
- **H** (numeric or array_like) – *Hue* h quadrature H .
- **HC** (numeric or array_like) – *Hue* h composition H^C .

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Returns a new instance of the `colour.CIECAM02_Specification` class.

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>C</code>	Alias for field number 1
<code>H</code>	Alias for field number 6
<code>HC</code>	Alias for field number 7
<code>J</code>	Alias for field number 0
<code>M</code>	Alias for field number 5
<code>Q</code>	Alias for field number 4
<code>h</code>	Alias for field number 2
<code>s</code>	Alias for field number 3

`colour.CIECAM02_VIEWING_CONDITIONS`

`colour.CIECAM02_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`
Reference *CIECAM02* colour appearance model viewing conditions.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

`CIECAM02_VIEWING_CONDITIONS` [CaseInsensitiveMapping] {'Average', 'Dim', 'Dark'}

Ancillary Objects

`colour.appearance`

<code>CIECAM02_InductionFactors</code>	<i>CIECAM02</i> colour appearance model induction factors.
--	--

colour.appearance.CIECAM02_InductionFactors

class colour.appearance.CIECAM02_InductionFactors
CIECAM02 colour appearance model induction factors.

Parameters

- **F** (numeric or array_like) – Maximum degree of adaptation F .
- **c** (numeric or array_like) – Exponential non linearity c .
- **N_c** (numeric or array_like) – Chromatic induction factor N_c .

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Create new instance of CIECAM02_InductionFactors(F , c , N_c)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

F	Alias for field number 0
N_c	Alias for field number 2
c	Alias for field number 1

CAM16

colour

XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b[, ...])	Computes the CAM16 colour appearance model correlates from given CIE XYZ tristimulus values.
CAM16_to_XYZ(CAM16_specification, XYZ_w, ...)	Converts CAM16 specification to CIE XYZ tristimulus values.
CAM16_Specification	Defines the CAM16 colour appearance model specification.
CAM16_VIEWING_CONDITIONS	Reference CAM16 colour appearance model viewing conditions.

colour.XYZ_to_CAM16

colour.XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround=CIECAM02_InductionFactors($F=1$, $c=0.69$, $N_c=1$), discount_illuminant=False)
Computes the CAM16 colour appearance model correlates from given CIE XYZ tristimulus values.

This is the *forward* implementation.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – CIE XYZ tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field luminance L_A in cd/m^2 , (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2 .
- **surround** (CAM16_InductionFactors, optional) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns CAM16 colour appearance model specification.

Return type CAM16_Specification

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
CAM16_Specification.h	[0, 360]	[0, 1]
CAM16_Specification.H	[0, 360]	[0, 1]

References

[LLW+17]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CAM16_VIEWING_CONDITIONS['Average']
>>> XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround) # doctest: +ELLIPSIS
CAM16_Specification(J=41.7312079..., C=0.1033557..., h=217.0679597..., s=2.3450150..., Q=195.
↪3717089..., M=0.1074367..., H=275.5949861..., HC=None)
```

colour.CAM16_to_XYZ

```
colour.CAM16_to_XYZ(CAM16_specification, XYZ_w, L_A, Y_b, surround, discount_illuminant=False)
```

Converts CAM16 specification to CIE XYZ tristimulus values.

This is the *reverse* implementation.

Parameters

- **CAM16_specification** (`CAM16_Specification`) – CAM16 colour appearance model specification. Correlate of *Lightness* J , correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh .
- **XYZ_w** (array_like) – CIE XYZ tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2 , (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2 .
- **surround** (`CAM16_InductionFactors`, optional) – Surround viewing conditions.
- **discount_illuminant** (`bool`, optional) – Discount the illuminant.

Returns XYZ – CIE XYZ tristimulus values.

Return type ndarray

Raises `ValueError` – If neither C or M correlates have been defined in the CAM16_specification argument.

Notes

Domain	Scale - Reference	Scale - 1
CAM16_specification.h	[0, 360]	[0, 1]
CAM16_specification.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- CAM16_specification can also be passed as a compatible argument to `colour.utilities.as_namedtuple()` definition.

References

[LLW+17]

Examples

```
>>> specification = CAM16_Specification(J=41.731207905126638,
...                                     C=0.103355738709070,
...                                     h=217.067959767393010)
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CAM16_to_XYZ(specification, XYZ_w, L_A, Y_b) # doctest: +ELLIPSIS
array([ 19.01..., 20..., 21.78...])
```

colour.CAM16_Specification

class colour.CAM16_Specification

Defines the *CAM16* colour appearance model specification.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness J*.
- **C** (numeric or array_like) – Correlate of *chroma C*.
- **h** (numeric or array_like) – *Hue* angle *h* in degrees.
- **s** (numeric or array_like) – Correlate of *saturation s*.
- **Q** (numeric or array_like) – Correlate of *brightness Q*.
- **M** (numeric or array_like) – Correlate of *colourfulness M*.
- **H** (numeric or array_like) – *Hue h* quadrature *H*.
- **HC** (numeric or array_like) – *Hue h* composition H^C .

References

[[LLW+17](#)]

Returns a new instance of the `colour.CAM16_Specification` class.

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>C</code>	Alias for field number 1
<code>H</code>	Alias for field number 6
<code>HC</code>	Alias for field number 7
<code>J</code>	Alias for field number 0
<code>M</code>	Alias for field number 5
<code>Q</code>	Alias for field number 4
<code>h</code>	Alias for field number 2
<code>s</code>	Alias for field number 3

colour.CAM16_VIEWING_CONDITIONS

`colour.CAM16_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`

Reference *CAM16* colour appearance model viewing conditions.

References

[LLW+17]

CAM16_VIEWING_CONDITIONS [CaseInsensitiveMapping] {'Average', 'Dim', 'Dark'}

Ancillary Objects

colour.appearance

CAM16_InductionFactors	CAM16 colour appearance model induction factors.
------------------------	--

colour.appearance.CAM16_InductionFactors

class colour.appearance.CAM16_InductionFactors
CAM16 colour appearance model induction factors.

Parameters

- **F** (numeric or array_like) – Maximum degree of adaptation F .
- **c** (numeric or array_like) – Exponential non linearity c .
- **N_c** (numeric or array_like) – Chromatic induction factor N_c .

References

[LLW+17]

Create new instance of CAM16_InductionFactors(F , c , N_c)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

F	Alias for field number 0
N_c	Alias for field number 2
c	Alias for field number 1

Hunt

colour

XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A[, ...])	Computes the <i>Hunt</i> colour appearance model correlates.
--	--

Continued on next page

Table 55 – continued from previous page

<code>Hunt_Specification</code>	Defines the <i>Hunt</i> colour appearance model specification.
<code>HUNT_VIEWING_CONDITIONS</code>	Reference <i>Hunt</i> colour appearance model viewing conditions.

colour.XYZ_to_Hunt

`colour.XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround=Hunt_InductionFactors(N_c=1, N_b=75, N_cb=None, N_bb=None), L_AS=None, CCT_w=None, XYZ_p=None, p=None, S=None, S_w=None, helson_judd_effect=False, discount_illuminant=True)`
 Computes the *Hunt* colour appearance model correlates.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – CIE XYZ tristimulus values of reference white.
- **XYZ_b** (array_like) – CIE XYZ tristimulus values of background.
- **L_A** (numeric or array_like) – Adapting field luminance L_A in cd/m^2 .
- **surround** (`Hunt_InductionFactors`, optional) – Surround viewing conditions induction factors.
- **L_AS** (numeric or array_like, optional) – Scotopic luminance L_{AS} of the illuminant, approximated if not specified.
- **CCT_w** (numeric or array_like, optional) – Correlated color temperature T_{cp} : of the illuminant, needed to approximate L_{AS} .
- **XYZ_p** (array_like, optional) – CIE XYZ tristimulus values of proximal field, assumed to be equal to background if not specified.
- **p** (numeric or array_like, optional) – Simultaneous contrast / assimilation factor p with value normalised to domain $[-1, 0]$ when simultaneous contrast occurs and normalised to domain $[0, 1]$ when assimilation occurs.
- **S** (numeric or array_like, optional) – Scotopic response S to the stimulus, approximated using tristimulus values Y of the stimulus if not specified.
- **S_w** (numeric or array_like, optional) – Scotopic response S_w for the reference white, approximated using the tristimulus values Y_w of the reference white if not specified.
- **helson_judd_effect** (`bool`, optional) – Truth value indicating whether the *Helson-Judd* effect should be accounted for.
- **discount_illuminant** (`bool`, optional) – Truth value indicating if the illuminant should be discounted.

Returns *Hunt* colour appearance model specification.

Return type *Hunt_Specification*

Raises `ValueError` – If an illegal arguments combination is specified.

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_b	[0, 100]	[0, 1]
XYZ_p	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Hunt_Specification.h	[0, 360]	[0, 1]

References

[Fai13f], [Hun04]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> XYZ_b = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> surround = HUNT_VIEWING_CONDITIONS['Normal Scenes']
>>> CCT_w = 6504.0
>>> XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround, CCT_w=CCT_w)
... # doctest: +ELLIPSIS
Hunt_Specification(J=30.0462678..., C=0.1210508..., h=269.2737594..., s=0.0199093..., Q=22.
↪2097654..., M=0.1238964..., H=None, HC=None)
```

colour.Hunt_Specification**class colour.Hunt_Specification**

Defines the *Hunt* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in *colour*. appearance but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J .
- **C** (numeric or array_like) – Correlate of *chroma* C_{94} .
- **h** (numeric or array_like) – *Hue* angle h_S in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s .
- **Q** (numeric or array_like) – Correlate of *brightness* Q .
- **M** (numeric or array_like) – Correlate of *colourfulness* M_{94} .
- **H** (numeric or array_like) – *Hue* h quadrature H .
- **HC** (numeric or array_like) – *Hue* h composition H_C .

Notes

- This specification is the one used in the current model implementation.

References

[Fai13f], [Hun04]

Create new instance of `Hunt_Specification(J, C, h, s, Q, M, H, HC)`

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>C</code>	Alias for field number 1
<code>H</code>	Alias for field number 6
<code>HC</code>	Alias for field number 7
<code>J</code>	Alias for field number 0
<code>M</code>	Alias for field number 5
<code>Q</code>	Alias for field number 4
<code>h</code>	Alias for field number 2
<code>s</code>	Alias for field number 3

`colour.HUNT_VIEWING_CONDITIONS`

`colour.HUNT_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Small Areas, Uniform Background & Surrounds': ...,
Reference Hunt colour appearance model viewing conditions.`

References

[Fai13f], [Hun04]

HUNT_VIEWING_CONDITIONS [`CaseInsensitiveMapping`] {'**Small Areas, Uniform Background & Surrounds**', '**Normal Scenes**', '**Television & CRT, Dim Surrounds**', '**Large Transparencies On Light Boxes**', '**Projected Transparencies, Dark Surrounds**'}

Aliases:

- '`small_uniform`': 'Small Areas, Uniform Background & Surrounds'
- '`normal`': 'Normal Scenes'
- '`tv_dim`': 'Television & CRT, Dim Surrounds'
- '`light_boxes`': 'Large Transparencies On Light Boxes'
- '`projected_dark`': 'Projected Transparencies, Dark Surrounds'

$LLAB(l : c)$

colour

<code>XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L[, surround, ...])</code>	Computes the $:math:LLAB(l:c)$ colour appearance model correlates.
<code>LLAB_Specification</code>	Defines the $:math:LLAB(l:c)$ colour appearance model specification.
<code>LLAB_VIEWING_CONDITIONS</code>	Reference $LLAB(l : c)$ colour appearance model viewing conditions.

colour.XYZ_to_LLAB

colour.XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L, surround=LLAB_InductionFactors(D=1, F_S=3, F_L=1, F_C=1))

Computes the $:math:LLAB(l:c)$ colour appearance model correlates.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of test sample / stimulus.
- **XYZ_0** (array_like) – CIE XYZ tristimulus values of reference white.
- **Y_b** (numeric or array_like) – Luminance factor of the background in cd/m^2 .
- **L** (numeric or array_like) – Absolute luminance L of reference white in cd/m^2 .
- **surround** (LLAB_InductionFactors, optional) – Surround viewing conditions induction factors.

Returns $:math:LLAB(l:c)$ colour appearance model specification.

Return type `LLAB_Specification`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
LLAB_Specification.h	[0, 360]	[0, 1]

References

[Fai13e], [LLK96], [LM96]

Examples

```

>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_0 = np.array([95.05, 100.00, 108.88])
>>> Y_b = 20.0
>>> L = 318.31
>>> surround = LLAB_VIEWING_CONDITIONS['ref_average_4_minus']
>>> XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L, surround) # doctest: +ELLIPSIS
LLAB_Specification(J=37.3668650..., C=0.0089496..., h=270..., s=0.0002395..., M=0.0190185...,
↪HC=None, a=..., b=-0.0190185...)

```

colour.LLAB_Specification

class colour.LLAB_Specification

Defines the :math:LLAB(l:c) colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in colour. appearance but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* L_L .
- **C** (numeric or array_like) – Correlate of *chroma* Ch_L .
- **h** (numeric or array_like) – *Hue* angle h_L in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s_L .
- **M** (numeric or array_like) – Correlate of *colourfulness* C_L .
- **HC** (numeric or array_like) – *Hue* h composition H^C .
- **a** (numeric or array_like) – Opponent signal A_L .
- **b** (numeric or array_like) – Opponent signal B_L .

Notes

- This specification is the one used in the current model implementation.

References

[Fai13e], [LLK96], [LM96]

Create new instance of LLAB_Specification(J, C, h, s, M, HC, a, b)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

C	Alias for field number 1
HC	Alias for field number 5
J	Alias for field number 0
M	Alias for field number 4
a	Alias for field number 6
b	Alias for field number 7
h	Alias for field number 2
s	Alias for field number 3

colour.LLAB_VIEWING_CONDITIONS

`colour.LLAB_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Reference Samples & Images, Average Surround, Subtending > 4', 'Reference Samples & Images, Average Surround, Subtending < 4', 'Television & VDU Displays, Dim Surround', 'Cut Sheet Transparency, Dim Surround', '35mm Projection Transparency, Dark Surround'})`
Reference *LLAB*(*l* : *c*) colour appearance model viewing conditions.

References

[Fai13e], [LLK96], [LM96]

LLAB_VIEWING_CONDITIONS [CaseInsensitiveMapping] {'Reference Samples & Images, Average Surround, Subtending > 4', 'Reference Samples & Images, Average Surround, Subtending < 4', 'Television & VDU Displays, Dim Surround', 'Cut Sheet Transparency, Dim Surround', '35mm Projection Transparency, Dark Surround'}

Aliases:

- 'ref_average_4_plus': 'Reference Samples & Images, Average Surround, Subtending > 4'
- 'ref_average_4_minus': 'Reference Samples & Images, Average Surround, Subtending < 4'
- 'tv_dim': 'Television & VDU Displays, Dim Surround'
- 'sheet_dim': 'Cut Sheet Transparency, Dim Surround'
- 'projected_dark': '35mm Projection Transparency, Dark Surround'

Ancillary Objects

`colour.appearance`

<code>LLAB_InductionFactors</code>	<code>:math:'LLAB(l:c)'</code> colour appearance model induction factors.
------------------------------------	---

colour.appearance.LLAB_InductionFactors

class `colour.appearance.LLAB_InductionFactors`
`:math:'LLAB(l:c)'` colour appearance model induction factors.

Parameters

- **D** (numeric or array_like) – *Discounting-the-Illuminant* factor D .
- **F_S** (numeric or array_like) – Surround induction factor F_S .
- **F_L** (numeric or array_like) – *Lightness* induction factor F_L .

- **F_C** (numeric or array_like) – *Chroma* induction factor F_C .

References

[Fai13e], [LLK96], [LM96]

Create new instance of `LLAB_InductionFactors(D, F_S, F_L, F_C)`

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>D</code>	Alias for field number 0
<code>F_C</code>	Alias for field number 3
<code>F_L</code>	Alias for field number 2
<code>F_S</code>	Alias for field number 1

Nayatani (1995)

`colour`

<code>XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or)</code>	Computes the <i>Nayatani (1995)</i> colour appearance model correlates.
<code>Nayatani95_Specification</code>	Defines the <i>Nayatani (1995)</i> colour appearance model specification.

`colour.XYZ_to_Nayatani95`

`colour.XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or, n=1)`

Computes the *Nayatani (1995)* colour appearance model correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_o** (numeric or array_like) – Luminance factor Y_o of achromatic background as percentage normalised to domain $[0.18, 1.0]$ in ‘**Reference**’ domain-range scale.
- **E_o** (numeric or array_like) – Illuminance E_o of the viewing field in lux.
- **E_or** (numeric or array_like) – Normalising illuminance E_{or} in lux usually normalised to domain $[1000, 3000]$.
- **n** (numeric or array_like, optional) – Noise term used in the non linear chromatic adaptation model.

Returns *Nayatani (1995)* colour appearance model specification.

Return type *Nayatani95_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Nayatani95_Specification.h	[0, 360]	[0, 1]

References

[Fai13g], [NSY95]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_n = np.array([95.05, 100.00, 108.88])
>>> Y_o = 20.0
>>> E_o = 5000.0
>>> E_or = 1000.0
>>> XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or) # doctest: +ELLIPSIS
Nayatani95_Specification(L_star_P=49.9998829..., C=0.0133550..., h=257.5232268..., s=0.0133550...,
↪ Q=62.6266734..., M=0.0167262..., H=None, HC=None, L_star_N=50.0039154...)
```

colour.Nayatani95_Specification

class colour.Nayatani95_Specification

Defines the *Nayatani (1995)* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour`, but diverge from *Fairchild (2013)* reference.

Parameters

- **L_star_P** (numeric or array_like) – Correlate of *achromatic Lightness* L_p^* .
- **C** (numeric or array_like) – Correlate of *chroma* C .
- **h** (numeric or array_like) – *Hue* angle θ in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* S .
- **Q** (numeric or array_like) – Correlate of *brightness* B_r .
- **M** (numeric or array_like) – Correlate of *colourfulness* M .
- **H** (numeric or array_like) – *Hue* h quadrature H .
- **HC** (numeric or array_like) – *Hue* h composition H_C .

- **L_star_N** (numeric or array_like) – Correlate of *normalised achromatic Lightness* L_n^* .

Notes

- This specification is the one used in the current model implementation.

References

[Fai13g], [NSY95]

Create new instance of `Nayatani95_Specification(L_star_P, C, h, s, Q, M, H, HC, L_star_N)`

__init__()

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>C</code>	Alias for field number 1
<code>H</code>	Alias for field number 6
<code>HC</code>	Alias for field number 7
<code>L_star_N</code>	Alias for field number 8
<code>L_star_P</code>	Alias for field number 0
<code>M</code>	Alias for field number 5
<code>Q</code>	Alias for field number 4
<code>h</code>	Alias for field number 2
<code>s</code>	Alias for field number 3

RLAB

colour

<code>XYZ_to_RLAB(XYZ, XYZ_n, Y_n[, sigma, D])</code>	Computes the <i>RLAB</i> model color appearance correlates.
<code>RLAB_D_FACTOR</code>	<i>RLAB</i> colour appearance model <i>Discounting-the-Illuminant</i> factor values.
<code>RLAB_Specification</code>	Defines the <i>RLAB</i> colour appearance model specification.
<code>RLAB_VIEWING_CONDITIONS</code>	Reference <i>RLAB</i> colour appearance model viewing conditions.

colour.XYZ_to_RLAB

`colour.XYZ_to_RLAB(XYZ, XYZ_n, Y_n, sigma=0.4347826086956522, D=1)`

Computes the *RLAB* model color appearance correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_n** (numeric or array_like) – Absolute adapting luminance in cd/m^2 .
- **sigma** (numeric or array_like, optional) – Relative luminance of the surround, see `colour.RLAB_VIEWING_CONDITIONS` for reference.
- **D** (numeric or array_like, optional) – *Discounting-the-Illuminant* factor normalised to domain [0, 1].

Returns *RLAB* colour appearance model specification.

Return type *RLAB_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
RLAB_Specification.h	[0, 360]	[0, 1]

References

[Fai96], [Fai13h]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_n = np.array([109.85, 100, 35.58])
>>> Y_n = 31.83
>>> sigma = RLAB_VIEWING_CONDITIONS['Average']
>>> D = RLAB_D_FACTOR['Hard Copy Images']
>>> XYZ_to_RLAB(XYZ, XYZ_n, Y_n, sigma, D) # doctest: +ELLIPSIS
RLAB_Specification(J=49.8347069..., C=54.8700585..., h=286.4860208..., s=1.1010410..., HC=None, u
↪ a=15.5711021..., b=-52.6142956...)
```

colour.RLAB_D_FACTOR

`colour.RLAB_D_FACTOR = CaseInsensitiveMapping({'Hard Copy Images': ..., 'Soft Copy Images': ..., 'Projected
RLAB colour appearance model Discounting-the-Illuminant factor values.`

References

[Fai96], [Fai13h]

RLAB_D_FACTOR [CaseInsensitiveMapping] {'Hard Copy Images', 'Soft Copy Images', 'Projected Transparencies, Dark Room'}

Aliases:

- 'hard_cp_img': 'Hard Copy Images'
- 'soft_cp_img': 'Soft Copy Images'
- 'projected_dark': 'Projected Transparencies, Dark Room'

colour.RLAB_Specification

class colour.RLAB_Specification

Defines the *RLAB* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour`, but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* L^R .
- **C** (numeric or array_like) – Correlate of *achromatic chroma* C^R .
- **h** (numeric or array_like) – *Hue* angle h^R in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s^R .
- **HC** (numeric or array_like) – *Hue h* composition H^C .
- **a** (numeric or array_like) – Red-green chromatic response a^R .
- **b** (numeric or array_like) – Yellow-blue chromatic response b^R .

Notes

- This specification is the one used in the current model implementation.

References

[Fai96], [Fai13h]

Create new instance of `RLAB_Specification(J, C, h, s, HC, a, b)`

__init__()

Initialize self. See `help(type(self))` for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

C	Alias for field number 1
HC	Alias for field number 4
J	Alias for field number 0
a	Alias for field number 5
b	Alias for field number 6
h	Alias for field number 2
s	Alias for field number 3

colour.RLAB_VIEWING_CONDITIONS

`colour.RLAB_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})`
Reference *RLAB* colour appearance model viewing conditions.

References

[Fai96], [Fai13h]

`RLAB_VIEWING_CONDITIONS` [CaseInsensitiveMapping] {'Average', 'Dim', 'Dark'}

Biochemistry

- *Michaelis–Menten Kinetics*

Michaelis–Menten Kinetics

`colour.biochemistry`

<code>reaction_rate_MichealisMenten(S, V_max, K_m)</code>	Describes the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S .
<code>substrate_concentration_MichealisMenten(v, ...)</code>	Describes the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v .

colour.biochemistry.reaction_rate_MichealisMenten

`colour.biochemistry.reaction_rate_MichealisMenten(S, V_max, K_m)`
Describes the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S .

Parameters

- **S** (array_like) – Concentration of a substrate S .
- **V_max** (array_like) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.

- **K_m** (array_like) – Substrate concentration V_{max} at which the reaction rate is half of V_{max} .

Returns Reaction rate v .

Return type array_like

References

[Wik03e]

Examples

```
>>> reaction_rate_MichealisMenten(0.5, 2.5, 0.8) # doctest: +ELLIPSIS
0.9615384...
```

colour.biochemistry.substrate_concentration_MichealisMenten

colour.biochemistry.**substrate_concentration_MichealisMenten**(v , V_{max} , K_m)

Describes the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v .

Parameters

- **v** (array_like) – Reaction rate v .
- **V_{max}** (array_like) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (array_like) – Substrate concentration V_{max} at which the reaction rate is half of V_{max} .

Returns Concentration of a substrate S .

Return type array_like

References

[Wik03e]

Examples

```
>>> substrate_concentration_MichealisMenten(0.961538461538461, 2.5, 0.8)
... # doctest: +ELLIPSIS
0.4999999...
```

Colour Vision Deficiency

- *Machado, Oliveira and Fernandes (2009)*

Machado, Oliveira and Fernandes (2009)

colour

<code>anomalous_trichromacy_cmfs_Machado2009(cmfs, ...)</code>	Shifts given <i>LMS</i> cone fundamentals colour matching functions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy using <i>Machado et al. (2009)</i> method..
<code>anomalous_trichromacy_matrix_Machado2009(...)</code>	Computes <i>Machado et al. (2009)</i> CVD matrix for given <i>LMS</i> cone fundamentals colour matching functions and display primaries tri-spectral distributions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy..
<code>cvd_matrix_Machado2009(deficiency, severity)</code>	Computes <i>Machado et al. (2009)</i> CVD matrix for given deficiency and severity using the pre-computed matrices dataset..

colour.anomalous_trichromacy_cmfs_Machado2009colour.**anomalous_trichromacy_cmfs_Machado2009**(*cmfs*, *d_LMS*)

Shifts given *LMS* cone fundamentals colour matching functions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy using *Machado et al. (2009)* method.

Parameters

- **cmfs** (*LMS_ConeFundamentals*) – *LMS* cone fundamentals colour matching functions.
- **d_LMS** (array_like) – Δ_{LMS} shift amount in nanometers.

Notes

- Input *LMS* cone fundamentals colour matching functions interval is expected to be 1 nanometer, incompatible input will be interpolated at 1 nanometer interval.
- Input Δ_{LMS} shift amount is in domain [0, 20].

Returns Anomalous trichromacy *LMS* cone fundamentals colour matching functions.

Return type *LMS_ConeFundamentals*

Warning: *Machado et al. (2009)* simulation of tritanomaly is based on the shift paradigm as an approximation to the actual phenomenon and restrain the model from trying to model tritanopia. The pre-generated matrices are using a shift value in domain [5, 59] contrary to the domain [0, 20] used for protanomaly and deuteranomaly simulation.

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> from colour import LMS_CMFS
>>> cmfs = LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> cmfs[450]
array([ 0.0498639,  0.0870524,  0.955393 ])
>>> anomalous_trichromacy_matrix_Machado2009(cmfs, np.array([15, 0, 0]))[450]
... # doctest: +ELLIPSIS
array([ 0.0891288...,  0.0870524 ,  0.955393  ])
```

colour.anomalous_trichromacy_matrix_Machado2009

`colour.anomalous_trichromacy_matrix_Machado2009(cmfs, primaries, d_LMS)`

Computes *Machado et al. (2009)* CVD matrix for given *LMS* cone fundamentals colour matching functions and display primaries tri-spectral distributions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy.

Parameters

- **cmfs** (*LMS_ConeFundamentals*) – *LMS* cone fundamentals colour matching functions.
- **primaries** (*RGB_DisplayPrimaries*) – *RGB* display primaries tri-spectral distributions.
- **d_LMS** (*array_like*) – Δ_{LMS} shift amount in nanometers.

Notes

- Input *LMS* cone fundamentals colour matching functions interval is expected to be 1 nanometer, incompatible input will be interpolated at 1 nanometer interval.
- Input Δ_{LMS} shift amount is in domain [0, 20].

Returns Anomalous trichromacy matrix.

Return type ndarray

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> from colour import DISPLAYS_RGB_PRIMARIES, LMS_CMFS
>>> cmfs = LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> d_LMS = np.array([15, 0, 0])
>>> primaries = DISPLAYS_RGB_PRIMARIES['Apple Studio Display']
>>> anomalous_trichromacy_matrix_Machado2009(cmfs, primaries, d_LMS)
... # doctest: +ELLIPSIS
array([[ -0.2777465...,  2.6515008..., -1.3737543...],
       [ 0.2718936...,  0.2004786...,  0.5276276...],
       [ 0.0064404...,  0.2592157...,  0.7343437...]])
```

colour.cvd_matrix_Machado2009

`colour.cvd_matrix_Machado2009(deficiency, severity)`

Computes *Machado et al. (2009)* CVD matrix for given deficiency and severity using the pre-computed matrices dataset.

Parameters

- **deficiency** (unicode) – {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'} Colour blindness / vision deficiency types : - *Protanomaly* : defective long-wavelength cones (L-cones). The complete absence of L-cones is known as *Protanopia* or *red-dichromacy*. - *Deuteranomaly* : defective medium-wavelength cones (M-cones) with peak of sensitivity moved towards the red sensitive cones. The complete absence of M-cones is known as *Deuteranopia*. - *Tritanomaly* : defective short-wavelength cones (S-cones), an alleviated form of blue-yellow color blindness. The complete absence of S-cones is known as *Tritanopia*.
- **severity** (numeric) – Severity of the colour vision deficiency in domain [0, 1].

Returns CVD matrix.

Return type ndarray

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> cvd_matrix_Machado2009('Protanomaly', 0.15) # doctest: +ELLIPSIS
array([[ 0.7869875...,  0.2694875..., -0.0564735...],
       [ 0.0431695...,  0.933774 ...,  0.023058 ...],
       [-0.004238 ..., -0.0024515...,  1.0066895...]])
```

Dataset

colour

CVD_MATRICES_MACHADO2010

Machado (2010) Simulation matrices Φ_{CVD} .

colour.CVD_MATRICES_MACHADO2010

`colour.CVD_MATRICES_MACHADO2010 = CaseInsensitiveMapping({'Protanomaly': ..., 'Deuteranomaly': ..., 'Tritanomaly': ...})`
Machado (2010) Simulation matrices Φ_{CVD} .

CVD_MATRICES_MACHADO2010 [CaseInsensitiveMapping] {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'}

Colour Characterisation

- [Colour Fitting](#)
- [Colour Rendition Charts](#)
- [Cameras](#)
- [Displays](#)

Colour Fitting

colour

POLYNOMIAL_EXPANSION_METHODS	Supported polynomial expansion methods.
polynomial_expansion(a[, method])	Performs polynomial expansion of given <i>a</i> array.
COLOUR_CORRECTION_MATRIX_METHODS	Supported colour correction matrix methods.
colour_correction_matrix(M_T, M_R[, method])	Computes a colour correction matrix from given M_T colour array to M_R colour array.
COLOUR_CORRECTION_METHODS	Supported colour correction methods.
colour_correction(RGB, M_T, M_R[, method])	Performs colour correction of given <i>RGB</i> colourspace array using the colour correction matrix from given M_T colour array to M_R colour array.

colour.POLYNOMIAL_EXPANSION_METHODS

`colour.POLYNOMIAL_EXPANSION_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ..., 'Vandermonde': ...})`
Supported polynomial expansion methods.

References

[CWCRO4], [FMH15], [WR04], [Wik03f]

POLYNOMIAL_EXPANSION_METHODS [CaseInsensitiveMapping] {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}

colour.polynomial_expansion

`colour.polynomial_expansion(a, method='Cheung 2004', **kwargs)`
Performs polynomial expansion of given *a* array.

Parameters

- **a** (array_like, (3, n)) – *a* array to expand.
- **method** (unicode, optional) – {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}, Computation method.

Other Parameters

- **degree** (int) – {`colour.characterisation.polynomial_expansion_Finlayson2015()`, `colour.characterisation.polynomial_expansion_Vandermonde()`}, Expanded

polynomial degree, must be one of [1, 2, 3, 4] for `colour.characterisation.polynomial_expansion_Finlayson2015()` definition.

- **terms** (*int*) – {`colour.characterisation.augmented_matrix_Cheung2004()`}, Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].
- **root_polynomial_expansion** (*bool*) – {`colour.characterisation.polynomial_expansion_Finlayson2015()`}, Whether to use the root-polynomials set for the expansion.

Returns Expanded a array.

Return type ndarray, (3, n)

References

[CWCRO4], [FMH15], [WR04], [Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion(RGB) # doctest: +ELLIPSIS
array([ 0.1722481..., 0.0917066..., 0.0641693...])
>>> polynomial_expansion(RGB, 'Cheung 2004', terms=5) # doctest: +ELLIPSIS
array([ 0.1722481..., 0.0917066..., 0.0641693..., 0.0010136..., 1...])
```

colour.COLOUR_CORRECTION_MATRIX_METHODS

`colour.COLOUR_CORRECTION_MATRIX_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ...})`
Supported colour correction matrix methods.

References

[CWCRO4], [FMH15], [WR04], [Wik03f]

POLYNOMIAL_EXPANSION_METHODS [CaseInsensitiveMapping] {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}

colour.colour_correction_matrix

`colour.colour_correction_matrix( $M_T$ ,  $M_R$ , method='Cheung 2004', **kwargs)`

Computes a colour correction matrix from given M_T colour array to M_R colour array.

The resulting colour correction matrix is computed using multiple linear or polynomial regression using given method. The purpose of that object is for example the matching of two *ColorChecker* colour rendition charts together.

Parameters

- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R .
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.

- **method** (unicode, optional) – {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}, Computation method.

Other Parameters

- **degree** (int) – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of [1, 2, 3, 4] for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **terms** (int) – {colour.characterisation.augmented_matrix_Cheung2004()}, Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].
- **root_polynomial_expansion** (bool) – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.

Returns Colour correction matrix.

Return type ndarray, (3, n)

References

[CWCRO4], [FMH15], [WR04], [Wik03f]

Examples

```
>>> M_T = np.array(
...     [[0.17224810, 0.09170660, 0.06416938],
...      [0.49189645, 0.27802050, 0.21923399],
...      [0.10999751, 0.18658946, 0.29938611],
...      [0.11666120, 0.14327905, 0.05713804],
...      [0.18988879, 0.18227649, 0.36056247],
...      [0.12501329, 0.42223442, 0.37027445],
...      [0.64785606, 0.22396782, 0.03365194],
...      [0.06761093, 0.11076896, 0.39779139],
...      [0.49101797, 0.09448929, 0.11623839],
...      [0.11622386, 0.04425753, 0.14469986],
...      [0.36867946, 0.44545230, 0.06028681],
...      [0.61632937, 0.32323906, 0.02437089],
...      [0.03016472, 0.06153243, 0.29014596],
...      [0.11103655, 0.30553067, 0.08149137],
...      [0.41162190, 0.05816656, 0.04845934],
...      [0.73339206, 0.53075188, 0.02475212],
...      [0.47347718, 0.08834792, 0.30310315],
...      [0.00000000, 0.25187016, 0.35062450],
...      [0.76809639, 0.78486240, 0.77808297],
...      [0.53822392, 0.54307997, 0.54710883],
...      [0.35458526, 0.35318419, 0.35524431],
...      [0.17976704, 0.18000531, 0.17991488],
...      [0.09351417, 0.09510603, 0.09675027],
...      [0.03405071, 0.03295077, 0.03702047]]
... )
>>> M_R = np.array(
...     [[0.15579559, 0.09715755, 0.07514556],
...      [0.39113140, 0.25943419, 0.21266708],
```

(continues on next page)

(continued from previous page)

```

...     [0.12824821, 0.18463570, 0.31508023],
...     [0.12028974, 0.13455659, 0.07408400],
...     [0.19368988, 0.21158946, 0.37955964],
...     [0.19957425, 0.36085439, 0.40678123],
...     [0.48896605, 0.20691688, 0.05816533],
...     [0.09775522, 0.16710693, 0.47147724],
...     [0.39358649, 0.12233400, 0.10526425],
...     [0.10780332, 0.07258529, 0.16151473],
...     [0.27502671, 0.34705454, 0.09728099],
...     [0.43980441, 0.26880559, 0.05430533],
...     [0.05887212, 0.11126272, 0.38552469],
...     [0.12705825, 0.25787860, 0.13566464],
...     [0.35612929, 0.07933258, 0.05118732],
...     [0.48131976, 0.42082843, 0.07120612],
...     [0.34665585, 0.15170714, 0.24969804],
...     [0.08261116, 0.24588716, 0.48707733],
...     [0.66054904, 0.65941137, 0.66376412],
...     [0.48051509, 0.47870296, 0.48230082],
...     [0.33045354, 0.32904184, 0.33228886],
...     [0.18001305, 0.17978567, 0.18004416],
...     [0.10283975, 0.10424680, 0.10384975],
...     [0.04742204, 0.04772203, 0.04914226]]
... )
>>> colour_correction_matrix(M_T, M_R) # doctest: +ELLIPSIS
array([[ 0.6982266...,  0.0307162...,  0.1621042...],
       [ 0.0689349...,  0.6757961...,  0.1643038...],
       [-0.0631495...,  0.0921247...,  0.9713415...]])

```

colour.COLOUR_CORRECTION_METHODS

`colour.COLOUR_CORRECTION_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ..., 'Vandermonde': ...})`
Supported colour correction methods.

References

[CWC04], [FMH15], [WR04], [Wik03f]

`COLOUR_CORRECTION_METHODS` [CaseInsensitiveMapping] {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}

colour.colour_correction

`colour.colour_correction(RGB, M_T, M_R, method='Cheung 2004', **kwargs)`

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given *M_T* colour array to *M_R* colour array.

Parameters

- **RGB** (array_like, (3, n)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (3, n)) – Test array *M_T* to fit onto array *M_R*.
- **M_R** (array_like, (3, n)) – Reference array the array *M_T* will be colour fitted against.

- **method** (unicode, optional) – {'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}, Computation method.

Other Parameters

- **degree** (int) – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of [1, 2, 3, 4] for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **terms** (int) – {colour.characterisation.augmented_matrix_Cheung2004()}, Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].
- **root_polynomial_expansion** (bool) – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[CWCRO4], [FMH15], [WR04], [Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> M_T = np.array(
...     [[0.17224810, 0.09170660, 0.06416938],
...      [0.49189645, 0.27802050, 0.21923399],
...      [0.10999751, 0.18658946, 0.29938611],
...      [0.11666120, 0.14327905, 0.05713804],
...      [0.18988879, 0.18227649, 0.36056247],
...      [0.12501329, 0.42223442, 0.37027445],
...      [0.64785606, 0.22396782, 0.03365194],
...      [0.06761093, 0.11076896, 0.39779139],
...      [0.49101797, 0.09448929, 0.11623839],
...      [0.11622386, 0.04425753, 0.14469986],
...      [0.36867946, 0.44545230, 0.06028681],
...      [0.61632937, 0.32323906, 0.02437089],
...      [0.03016472, 0.06153243, 0.29014596],
...      [0.11103655, 0.30553067, 0.08149137],
...      [0.41162190, 0.05816656, 0.04845934],
...      [0.73339206, 0.53075188, 0.02475212],
...      [0.47347718, 0.08834792, 0.30310315],
...      [0.00000000, 0.25187016, 0.35062450],
...      [0.76809639, 0.78486240, 0.77808297],
...      [0.53822392, 0.54307997, 0.54710883],
...      [0.35458526, 0.35318419, 0.35524431],
...      [0.17976704, 0.18000531, 0.17991488],
...      [0.09351417, 0.09510603, 0.09675027],
...      [0.03405071, 0.03295077, 0.03702047]]
... )
>>> M_R = np.array(
...     [[0.15579559, 0.09715755, 0.07514556],
```

(continues on next page)

(continued from previous page)

```

...     [0.39113140, 0.25943419, 0.21266708],
...     [0.12824821, 0.18463570, 0.31508023],
...     [0.12028974, 0.13455659, 0.07408400],
...     [0.19368988, 0.21158946, 0.37955964],
...     [0.19957425, 0.36085439, 0.40678123],
...     [0.48896605, 0.20691688, 0.05816533],
...     [0.09775522, 0.16710693, 0.47147724],
...     [0.39358649, 0.12233400, 0.10526425],
...     [0.10780332, 0.07258529, 0.16151473],
...     [0.27502671, 0.34705454, 0.09728099],
...     [0.43980441, 0.26880559, 0.05430533],
...     [0.05887212, 0.11126272, 0.38552469],
...     [0.12705825, 0.25787860, 0.13566464],
...     [0.35612929, 0.07933258, 0.05118732],
...     [0.48131976, 0.42082843, 0.07120612],
...     [0.34665585, 0.15170714, 0.24969804],
...     [0.08261116, 0.24588716, 0.48707733],
...     [0.66054904, 0.65941137, 0.66376412],
...     [0.48051509, 0.47870296, 0.48230082],
...     [0.33045354, 0.32904184, 0.33228886],
...     [0.18001305, 0.17978567, 0.18004416],
...     [0.10283975, 0.10424680, 0.10384975],
...     [0.04742204, 0.04772203, 0.04914226]]
... )
>>> colour_correction(RGB, M_T, M_R) # doctest: +ELLIPSIS
array([ 0.1334872..., 0.0843921..., 0.0599014...])

```

Ancillary Objects

colour.characterisation

<code>augmented_matrix_Cheung2004(RGB[, terms])</code>	Performs polynomial expansion of given <i>RGB</i> colourspace array using <i>Cheung et al.(2004)</i> method..
<code>polynomial_expansion_Finlayson2015(RGB[, ...])</code>	Performs polynomial expansion of given <i>RGB</i> colourspace array using <i>Finlayson et al.(2015)</i> method..
<code>polynomial_expansion_Vandermonde(a[, degree])</code>	Performs polynomial expansion of given <i>a</i> array using <i>Vandermonde</i> method.
<code>colour_correction_matrix_Cheung2004(M_T, M_R)</code>	Computes a colour correction from given M_T colour array to M_R colour array using <i>Cheung et al.(2004)</i> method..
<code>colour_correction_matrix_Finlayson2015(M_T, M_R)</code>	Computes a colour correction matrix from given M_T colour array to M_R colour array using <i>Finlayson et al.(2015)</i> method..
<code>colour_correction_matrix_Vandermonde(M_T, M_R)</code>	Computes a colour correction matrix from given M_T colour array to M_R colour array using <i>Vandermonde</i> method.
<code>colour_correction_Cheung2004(RGB, M_T, M_R)</code>	Performs colour correction of given <i>RGB</i> colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using <i>Cheung et al.(2004)</i> method..

Continued on next page

Table 74 – continued from previous page

<code>colour_correction_Finlayson2015</code> (<i>RGB</i> , <i>M_T</i> , <i>M_R</i>)	Performs colour correction of given <i>RGB</i> colourspace array using the colour correction matrix from given <i>M_T</i> colour array to <i>M_R</i> colour array using <i>Finlayson et al. (2015)</i> method..
<code>colour_correction_Vandermonde</code> (<i>RGB</i> , <i>M_T</i> , <i>M_R</i>)	Performs colour correction of given <i>RGB</i> colourspace array using the colour correction matrix from given <i>M_T</i> colour array to <i>M_R</i> colour array using <i>Vandermonde</i> method.

`colour.characterisation.augmented_matrix_Cheung2004`

`colour.characterisation.augmented_matrix_Cheung2004`(*RGB*, *terms*=3)

Performs polynomial expansion of given *RGB* colourspace array using *Cheung et al. (2004)* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to expand.
- **terms** (int, optional) – Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].

Returns Expanded *RGB* colourspace array.

Return type ndarray

Notes

- This definition combines the augmented matrices given in [CWCRO4] and [WR04].

References

[CWCRO4], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> augmented_matrix_Cheung2004(RGB, terms=5) # doctest: +ELLIPSIS
array([ 0.1722481...,  0.0917066...,  0.0641693...,  0.0010136...,  1...])
```

`colour.characterisation.polynomial_expansion_Finlayson2015`

`colour.characterisation.polynomial_expansion_Finlayson2015`(*RGB*, *degree*=1, *root_polynomial_expansion*=True)

Performs polynomial expansion of given *RGB* colourspace array using *Finlayson et al. (2015)* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to expand.
- **degree** (int, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Expanded *RGB* colourspace array.

Return type ndarray

References

[FMH15]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Finlayson2015(RGB, degree=2) # doctest: +ELLIPSIS
array([ 0.1722481...,  0.0917066...,  0.06416938...,  0.0078981...,  0.0029423...,
        0.0055265...])
```

colour.characterisation.polynomial_expansion_Vandermonde

colour.characterisation.**polynomial_expansion_Vandermonde**(*a*, *degree*=1)
 Performs polynomial expansion of given *a* array using *Vandermonde* method.

Parameters

- **a** (array_like) – *a* array to expand.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Expanded *a* array.

Return type ndarray

References

[Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Vandermonde(RGB) # doctest: +ELLIPSIS
array([ 0.1722481 ,  0.0917066 ,  0.06416938,  1.          ])
```

colour.characterisation.colour_correction_matrix_Cheung2004

colour.characterisation.**colour_correction_matrix_Cheung2004**(*M_T*, *M_R*, *terms*=3)
 Computes a colour correction from given *M_T* colour array to *M_R* colour array using *Cheung et al. (2004)* method.

Parameters

- **M_T** (array_like, (3, n)) – Test array *M_T* to fit onto array *M_R*.
- **M_R** (array_like, (3, n)) – Reference array the array *M_T* will be colour fitted against.

- **terms** (`int`, optional) – Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].

Returns Colour correction matrix.

Return type ndarray, (3, n)

References

[CWCR04], [WR04]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R) # doctest: +ELLIPSIS
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_matrix_Finlayson2015

colour.characterisation.colour_correction_matrix_Finlayson2015(M_T , M_R , *degree=1*,
root_polynomial_expansion=True)

Computes a colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.

Parameters

- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R .
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.
- **degree** (`int`, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (`bool`) – Whether to use the root-polynomials set for the expansion.

Returns Colour correction matrix.

Return type ndarray, (3, n)

References

[FMH15]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R) # doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_matrix_Vandermonde

colour.characterisation.colour_correction_matrix_Vandermonde(M_T , M_R , degree=1)

Computes a colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.

Parameters

- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R .
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Colour correction matrix.

Return type ndarray, (3, n)

References

[Wik03f]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R) # doctest: +ELLIPSIS
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_Cheung2004

colour.characterisation.colour_correction_Cheung2004(RGB , M_T , M_R , terms=3)

Performs colour correction of given RGB colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.

Parameters

- **RGB** (array_like, (3, n)) – RGB colourspace array to colour correct.
- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R .
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.
- **terms** (int, optional) – Number of terms of the expanded polynomial, must be one of [3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22].

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[CWCRO4], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Cheung2004(RGB, M_T, M_R) # doctest: +ELLIPSIS
array([ 0.1793456..., 0.1003392..., 0.0617218...])
```

colour.characterisation.colour_correction_Finlayson2015

`colour.characterisation.colour_correction_Finlayson2015(RGB, M_T, M_R, degree=1, root_polynomial_expansion=True)`

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.

Parameters

- **RGB** (array_like, (3, n)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R .
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[FMH15]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Finlayson2015(RGB, M_T, M_R) # doctest: +ELLIPSIS
array([ 0.1793456..., 0.1003392..., 0.0617218...])
```

colour.characterisation.colour_correction_Vandermonde

colour.characterisation.colour_correction_Vandermonde(*RGB*, *M_T*, *M_R*, *degree*=1)

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given *M_T* colour array to *M_R* colour array using *Vandermonde* method.

Parameters

- **RGB** (array_like, (3, n)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (3, n)) – Test array *M_T* to fit onto array *M_R*.
- **M_R** (array_like, (3, n)) – Reference array the array *M_T* will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Vandermonde(RGB, M_T, M_R) # doctest: +ELLIPSIS
array([ 0.2128689...,  0.1106242...,  0.036213 ...])
```

Colour Rendition Charts

Dataset

colour

COLOURCHECKERS	Aggregated <i>ColourCheckers</i> chromaticity coordinates.
COLOURCHECKERS_SDS	Aggregated <i>ColourCheckers</i> spectral distributions.

colour.COLOURCHECKERS

colour.COLOURCHECKERS = CaseInsensitiveMapping({'ColorChecker 1976': ..., 'ColorChecker 2005': ..., 'BabelColorChecker 2005': ...})
Aggregated *ColourCheckers* chromaticity coordinates.

References

[Bab12b], [Bab12a], [XR15]

COLOURCHECKERS [CaseInsensitiveMapping] {'ColorChecker 1976', 'ColorChecker 2005', 'BabelColor Average', 'ColorChecker24 - Before November 2014', 'ColorChecker24 - After November 2014'}

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc2005': 'ColorChecker 2005'
- 'ccb2014': 'ColorChecker24 - Before November 2014'
- 'cca2014': 'ColorChecker24 - After November 2014'

colour.COLOURCHECKERS_SDS

`colour.COLOURCHECKERS_SDS = CaseInsensitiveMapping({'BabelColor Average': ..., 'ColorChecker N Ohta': ..., Aggregated ColourCheckers spectral distributions.`

References

[Oht97], [Bab12b], [Bab12a], [MunsellCSsciencea]

COLOURCHECKERS [CaseInsensitiveMapping] {'BabelColor Average', 'ColorChecker N Ohta'}

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc_ohta': 'ColorChecker N Ohta'

Ancillary Objects

`colour.characterisation`

[ColourChecker](#)

ColourChecker data.

colour.characterisation.ColourChecker

class `colour.characterisation.ColourChecker`
ColourChecker data.

Parameters

- **name** (unicode) – *ColourChecker* name.
- **data** (OrderedDict) – chromaticity coordinates in *CIE xyY* colourspace.
- **illuminant** (array_like) – *ColourChecker* illuminant chromaticity coordinates.

Create new instance of *ColourChecker*(name, data, illuminant)

__init__()

Initialize self. See `help(type(self))` for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

data	Alias for field number 1
illuminant	Alias for field number 2
name	Alias for field number 0

Cameras

`colour.characterisation`

<code>RGB_SpectralSensitivities</code> ([data, domain, labels])	Implements support for a camera <i>RGB</i> spectral sensitivities.
---	--

`colour.characterisation.RGB_SpectralSensitivities`

class `colour.characterisation.RGB_SpectralSensitivities`(*data=None, domain=None, labels=None, **kwargs*)

Implements support for a camera *RGB* spectral sensitivities.

Parameters

- **data** (*Series* or *Dataframe* or *Signal* or *MultiSignal* or *MultiSpectralDistribution* or *array_like* or *dict_like*, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (*array_like*, optional) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both data and domain arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` attribute.
- **labels** (*array_like*, optional) – Names to use for the `colour.SpectralDistribution` class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distribution labels for figures, default to `colour.characterisation.RGB_SpectralSensitivities.labels` attribute value.

`__init__(data=None, domain=None, labels=None, **kwargs)`
 Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([data, domain, labels])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>normalise([factor])</code>	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas DataFrame</i> class instance.
<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>	
<code>domain</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain <i>x</i> variable.

Continued on next page

Table 81 – continued from previous page

dtype	Getter and setter property for the continuous signal dtype.
extrapolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
extrapolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
function	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
interpolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
interpolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
items	
labels	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
mapping	
name	Getter and setter property for the abstract continuous function name.
range	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
shape	Getter and setter property for the multi-spectral distribution shape.
signal_type	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
signals	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
strict_labels	Getter and setter property for the multi-spectral distribution strict labels.
strict_name	Getter and setter property for the multi-spectral distribution strict name.
title	
values	Getter and setter property for the multi-spectral distribution values.
wavelengths	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
x	
y	
z	

Dataset

colour

<code>CAMERAS_RGB_SPECTRAL_SENSITIVITIES</code>	Cameras <i>RGB</i> spectral sensitivities.
---	--

colour.CAMERAS_RGB_SPECTRAL_SENSITIVITIES

`colour.CAMERAS_RGB_SPECTRAL_SENSITIVITIES` = `CaseInsensitiveMapping`({'Nikon 5100 (NPL)': ..., 'Sigma SDMerill
Cameras *RGB* spectral sensitivities.

References

[DFGM15]

`CAMERAS_RGB_SPECTRAL_SENSITIVITIES` [`CaseInsensitiveMapping`] {**Nikon 5100 (NPL)**, **Sigma SDMerill (NPL)**}

Displays

`colour.characterisation`

<code>RGB_DisplayPrimaries([data, domain, labels])</code>	Implements support for a <i>RGB</i> display (such as a <i>CRT</i> or <i>LCD</i>) primaries multi-spectral distributions.
---	---

colour.characterisation.RGB_DisplayPrimaries

class `colour.characterisation.RGB_DisplayPrimaries`(*data=None*, *domain=None*, *labels=None*,
***kwargs*)

Implements support for a *RGB* display (such as a *CRT* or *LCD*) primaries multi-spectral distributions.

Parameters

- **data** (`Series` or `Dataframe` or `Signal` or `MultiSignal` or `MultiSpectralDistribution` or `array_like` or `dict_like`, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (`array_like`, optional) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both `data` and `domain` arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` attribute.
- **labels** (`array_like`, optional) – Names to use for the `colour.SpectralDistribution` class instances.

Other Parameters

- **name** (*unicode*, optional) – Multi-spectral distribution name.
- **interpolator** (*object*, optional) – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_args** (*dict_like*, optional) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object*, optional) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like*, optional) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.

- **strict_labels** (*array_like, optional*) – Multi-spectral distribution labels for figures, default to `colour.characterisation.RGB_DisplayPrimaries.labels` attribute value.

__init__(*data=None, domain=None, labels=None, **kwargs*)
Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__([data, domain, labels])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>normalise([factor])</code>	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas DataFrame</i> class instance.
<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>

Continued on next page

Table 85 – continued from previous page

domain	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain x variable.
dtype	Getter and setter property for the continuous signal dtype.
extrapolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
extrapolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
function	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
interpolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
interpolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
items	
labels	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
mapping	
name	Getter and setter property for the abstract continuous function name.
range	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
shape	Getter and setter property for the multi-spectral distribution shape.
signal_type	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
signals	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
strict_labels	Getter and setter property for the multi-spectral distribution strict labels.
strict_name	Getter and setter property for the multi-spectral distribution strict name.
title	
values	Getter and setter property for the multi-spectral distribution values.
wavelengths	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
x	
y	
z	

Dataset

colour

DISPLAYS_RGB_PRIMARIES

Displays *RGB* primaries multi-spectral distributions.

colour.DISPLAYS_RGB_PRIMARIES

`colour.DISPLAYS_RGB_PRIMARIES = CaseInsensitiveMapping({'Typical CRT Brainard 1997': ..., 'Apple Studio Display': ...})`
 Displays *RGB* primaries multi-spectral distributions.

References

[FW98], [Mac10]

DISPLAYS_RGB_PRIMARIES [CaseInsensitiveMapping] {**Apple Studio Display**, **Typical CRT Brainard 1997**}

Colorimetry

- *Spectral Data Structure*
- *Spectral Data Generation*
- *Conversion to Tristimulus Values*
 - *ASTM E308-15*
 - *Integration*
- *Spectral Bandpass Dependence Correction*
 - *Stearns and Stearns (1988)*
- *Colour Matching Functions*
- *Colour Matching Functions Transformations*
- *Illuminants and Light Sources*
- *Dominant Wavelength and Purity*
- *Luminous Efficiency Functions*
- *Lightness Computation*
 - *Glasser, Mckinney, Reilly and Schnelle (1958)*
 - *Wyszecki (1963)*
 - *CIE 1976*
 - *Fairchild and Wyble (2010)*
 - *Fairchild and Chen (2011)*
- *Luminance Computation*
 - *Newhall, Nickerson and Judd (1943)*
 - *CIE 1976*
 - *ASTM D1535-08e1*

- *Fairchild and Wyble (2010)*
- *Fairchild and Chen (2011)*
- *Whiteness Computation*
 - *Berger (1959)*
 - *Taube (1960)*
 - *Stensby (1968)*
 - *ASTM E313*
 - *Ganz and Griesser (1979)*
 - *CIE 2004*
- *Yellowness Computation*
 - *ASTM D1925*
 - *ASTM E313*

Spectral Data Structure

colour

<code>SpectralShape([start, end, interval])</code>	Defines the base object for spectral distribution shape.
<code>SpectralDistribution([data, domain])</code>	Defines the spectral distribution: the base object for spectral computations.
<code>MultiSpectralDistribution([data, domain, labels])</code>	Defines multi-spectral distribution: the base object for multi spectral computations.
<code>DEFAULT_SPECTRAL_SHAPE</code>	(360, 780, 1).
<code>ASTME30815_PRACTISE_SHAPE</code>	(360, 780, 1).

colour.SpectralShape

class colour.**SpectralShape**(*start=None, end=None, interval=None*)

Defines the base object for spectral distribution shape.

Parameters

- **start** (numeric, optional) – Wavelength λ_i range start in nm.
- **end** (numeric, optional) – Wavelength λ_i range end in nm.
- **interval** (numeric, optional) – Wavelength λ_i range interval.

start

end

interval

boundaries

__str__()

__repr__()

```
__iter__()  
__contains__()  
__len__()  
__eq__()  
__ne__()  
range()
```

Examples

```
>>> SpectralShape(360, 830, 1)  
SpectralShape(360, 830, 1)
```

```
__init__(start=None, end=None, interval=None)  
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

<code>__init__([start, end, interval])</code>	Initialize self.
<code>range([dtype])</code>	Returns an iterable range for the spectral shape.

Attributes

<code>boundaries</code>	Getter and setter property for the spectral shape boundaries.
<code>end</code>	Getter and setter property for the spectral shape end.
<code>interval</code>	Getter and setter property for the spectral shape interval.
<code>start</code>	Getter and setter property for the spectral shape start.

colour.SpectralDistribution

class colour.**SpectralDistribution**(*data=None, domain=None, **kwargs*)

Defines the spectral distribution: the base object for spectral computations.

The spectral distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (Series or `Signal`, `SpectralDistribution` or array_like or dict_like, optional) – Data to be stored in the spectral distribution.
- **domain** (array_like, optional) – Values to initialise the colour. `SpectralDistribution.wavelength` attribute with. If both data and domain

arguments are defined, the latter will be used to initialise the colour. `SpectralDistribution.wavelength` attribute.

Other Parameters

- **name** (*unicode, optional*) – Spectral distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function.
- **strict_name** (*unicode, optional*) – Spectral distribution name for figures, default to `colour.SpectralDistribution.name` attribute value.

strict_name

wavelengths

values

shape

__init__()

extrapolate()

interpolate()

align()

trim()

normalise()

References

[CIET13805a], [CIET13805c], [CIET14804h]

Examples

Instantiating a spectral distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data) # doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```
SpectralDistribution([[ 500.    ,    0.0651],
                    [ 520.    ,    0.0705],
                    [ 540.    ,    0.0772],
                    [ 560.    ,    0.087 ],
                    [ 580.    ,    0.1128],
                    [ 600.    ,    0.136 ]],
                    interpolator=SpragueInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data) # doctest: +ELLIPSIS
SpectralDistribution([[ 500.    ,    0.0651 ],
                    [ 510.    ,    0.31416],
                    [ 520.    ,    0.0705 ],
                    [ 540.    ,    0.0772 ],
                    [ 560.    ,    0.087  ],
                    [ 580.    ,    0.1128 ],
                    [ 600.    ,    0.136  ]],
                    interpolator=CubicSplineInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
```

__init__(*data=None, domain=None, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([data, domain])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.

Continued on next page

Table 90 – continued from previous page

<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain x variable is uniform.
<code>normalise([factor])</code>	Normalises the spectral distribution using given normalization factor.
<code>signal_unpack_data([data, domain, dtype])</code>	Unpack given data for continuous signal instantiation.
<code>to_series()</code>	Converts the continuous signal to a <i>Pandas</i> Series class instance.
<code>trim(shape)</code>	Trims the spectral distribution wavelengths to given spectral shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>	
<code>domain</code>	Getter and setter property for the continuous signal independent domain x variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the continuous signal extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the continuous signal extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the continuous signal callable.
<code>interpolator</code>	Getter and setter property for the continuous signal interpolator type.
<code>interpolator_args</code>	Getter and setter property for the continuous signal interpolator instantiation time arguments.
<code>items</code>	
<code>name</code>	Getter and setter property for the abstract continuous function name.
<code>range</code>	Getter and setter property for the continuous signal corresponding range y variable.
<code>shape</code>	Getter and setter property for the spectral distribution shape.
<code>strict_name</code>	Getter and setter property for the spectral distribution strict name.
<code>title</code>	
<code>values</code>	Getter and setter property for the spectral distribution values.
<code>wavelengths</code>	Getter and setter property for the spectral distribution wavelengths λ_n .

colour.MultiSpectralDistribution

class colour.MultiSpectralDistribution(*data=None, domain=None, labels=None, **kwargs*)

Defines multi-spectral distribution: the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc. . .

The multi-spectral distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (*Series or Dataframe or Signal or MultiSignal or MultiSpectralDistribution or array_like or dict_like, optional*) – Data to be stored in the multi-spectral distribution.
- **domain** (*array_like, optional*) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (*array_like, optional*) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distribution labels for figures, default to colour.MultiSpectralDistribution.labels attribute value.

strict_name
strict_labels
wavelengths
values
shape
extrapolate()
interpolate()
align()
trim()
normalise()
to_sds()

References

[CIET13805a], [CIET13805c], [CIET14804h]

Examples

Instantiating a multi-spectral distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> labels = ('x_bar', 'y_bar', 'z_bar')
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistribution(data, labels=labels)
... # doctest: +ELLIPSIS
MultiSpectral...([[ 500.      ,    0.0049 ,    0.323  ,    0.272  ],
... [ 510.      ,    0.0093 ,    0.503  ,    0.1582 ],
... [ 520.      ,    0.06327,    0.71   ,    0.07825],
... [ 530.      ,    0.1655 ,    0.862  ,    0.04216],
... [ 540.      ,    0.2904 ,    0.954  ,    0.0203  ],
... [ 550.      ,    0.43345,    0.99495,    0.00875],
... [ 560.      ,    0.5945 ,    0.995  ,    0.0039  ]],
... labels=[... 'x_bar', ... 'y_bar', ... 'z_bar'],
... interpolator=SpragueInterpolator,
... interpolator_args={},
... extrapolator=Extrapolator,
... extrapolator_args={...})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistribution(data, labels=labels)
... # doctest: +ELLIPSIS
MultiSpectral...([[ 500.      ,    0.0049 ,    0.323  ,    0.272  ],
... [ 510.      ,    0.0093 ,    0.503  ,    0.1582 ],
... [ 511.      ,    0.00314,    0.31416,    0.03142],
... [ 520.      ,    0.06327,    0.71   ,    0.07825],
... [ 530.      ,    0.1655 ,    0.862  ,    0.04216],
... [ 540.      ,    0.2904 ,    0.954  ,    0.0203  ],
... [ 550.      ,    0.43345,    0.99495,    0.00875],
... [ 560.      ,    0.5945 ,    0.995  ,    0.0039  ]],
... labels=[... 'x_bar', ... 'y_bar', ... 'z_bar'],
... interpolator=CubicSplineInterpolator,
... interpolator_args={},
... extrapolator=Extrapolator,
... extrapolator_args={...})
```

__init__(data=None, domain=None, labels=None, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([data, domain, labels])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>normalise([factor])</code>	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas</i> DataFrame class instance.
<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>	
<code>domain</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain <i>x</i> variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.

Continued on next page

Table 93 – continued from previous page

<code>extrapolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
<code>interpolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
<code>interpolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
<code>items</code>	
<code>labels</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
<code>mapping</code>	
<code>name</code>	Getter and setter property for the abstract continuous function name.
<code>range</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
<code>shape</code>	Getter and setter property for the multi-spectral distribution shape.
<code>signal_type</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
<code>signals</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
<code>strict_labels</code>	Getter and setter property for the multi-spectral distribution strict labels.
<code>strict_name</code>	Getter and setter property for the multi-spectral distribution strict name.
<code>title</code>	
<code>values</code>	Getter and setter property for the multi-spectral distribution values.
<code>wavelengths</code>	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
<code>x</code>	
<code>y</code>	
<code>z</code>	

colour.DEFAULT_SPECTRAL_SHAPE

```
colour.DEFAULT_SPECTRAL_SHAPE = SpectralShape(360, 780, 1)
(360, 780, 1).
```

References

[ASTMInternational15]

ASTME30815_PRACTISE_SHAPE : SpectralShape

Type Shape for *ASTM E308-15* practise

colour.ASTM E30815_PRACTISE_SHAPE

```
colour.ASTM E30815_PRACTISE_SHAPE = SpectralShape(360, 780, 1)
(360, 780, 1).
```

References

[ASTMInternational15]

ASTM E30815_PRACTISE_SHAPE : SpectralShape

Type Shape for *ASTM E308-15* practise

Spectral Data Generation

colour

<code>sd_CIE_standard_illuminant_A([shape])</code>	<i>CIE Standard Illuminant A</i> is intended to represent typical, domestic, tungsten-filament lighting.
<code>sd_CIE_illuminant_D_series(xy[, M1_M2_rounding])</code>	Returns the spectral distribution of given <i>CIE Illuminant D Series</i> using given <i>xy</i> chromaticity coordinates.
<code>sd_blackbody(temperature[, shape, c1, c2, n])</code>	Returns the spectral distribution of the planckian radiator for given temperature $T[K]$.
<code>sd_constant(k[, shape, dtype])</code>	Returns a spectral distribution of given spectral shape filled with constant k values.
<code>sd_ones([shape])</code>	Returns a spectral distribution of given spectral shape filled with ones.
<code>sd_zeros([shape])</code>	Returns a spectral distribution of given spectral shape filled with zeros.
<code>SD_GAUSSIAN_METHODS</code>	Supported gaussian spectral distribution computation methods.
<code>sd_gaussian(mu_peak_wavelength, sigma_fwhm)</code>	Returns a gaussian spectral distribution of given spectral shape using given method.
<code>SD_SINGLE_LED_METHODS</code>	Supported single <i>LED</i> spectral distribution computation methods.
<code>sd_single_led(peak_wavelength, fwhm[, ...])</code>	Returns a single <i>LED</i> spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to given method.
<code>SD_MULTI_LEDS_METHODS</code>	Supported multi <i>LED</i> spectral distribution computation methods.
<code>sd_multi_leds(peak_wavelengths, fwhm[, ...])</code>	Returns a multi <i>LED</i> spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to given method.

colour.sd_CIE_standard_illuminant_A

`colour.sd_CIE_standard_illuminant_A(shape=SpectralShape(360, 780, 1))`

CIE Standard Illuminant A is intended to represent typical, domestic, tungsten-filament lighting.

Its spectral distribution is that of a Planckian radiator at a temperature of approximately 2856 K. *CIE Standard Illuminant A* should be used in all applications of colorimetry involving the use of incandescent lighting, unless there are specific reasons for using a different illuminant.

Parameters `shape` (`SpectralShape`, optional) – Spectral shape used to create the spectral distribution of the *CIE Standard Illuminant A*.

Returns *CIE Standard Illuminant A*. spectral distribution.

Return type `SpectralDistribution`

References

[CIET14804a]

Examples

```
>>> from colour import SpectralShape
>>> sd_CIE_standard_illuminant_A(SpectralShape(400, 700, 10))
... # doctest: +ELLIPSIS
SpectralDistribution([[ 400.      ,  14.7080384...],
                    [ 410.      ,  17.6752521...],
                    [ 420.      ,  20.9949572...],
                    [ 430.      ,  24.6709226...],
                    [ 440.      ,  28.7027304...],
                    [ 450.      ,  33.0858929...],
                    [ 460.      ,  37.8120566...],
                    [ 470.      ,  42.8692762...],
                    [ 480.      ,  48.2423431...],
                    [ 490.      ,  53.9131532...],
                    [ 500.      ,  59.8610989...],
                    [ 510.      ,  66.0634727...],
                    [ 520.      ,  72.4958719...],
                    [ 530.      ,  79.1325945...],
                    [ 540.      ,  85.9470183...],
                    [ 550.      ,  92.9119589...],
                    [ 560.      ,  100.      ...],
                    [ 570.      ,  107.1837952...],
                    [ 580.      ,  114.4363383...],
                    [ 590.      ,  121.7312009...],
                    [ 600.      ,  129.0427389...],
                    [ 610.      ,  136.3462674...],
                    [ 620.      ,  143.6182057...],
                    [ 630.      ,  150.8361944...],
                    [ 640.      ,  157.9791857...],
                    [ 650.      ,  165.0275098...],
                    [ 660.      ,  171.9629200...],
                    [ 670.      ,  178.7686175...],
                    [ 680.      ,  185.4292591...],
                    [ 690.      ,  191.9309499...],
                    [ 700.      ,  198.2612232...],
```

(continues on next page)

(continued from previous page)

```

interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})

```

colour.sd_CIE_illuminant_D_series

colour.sd_CIE_illuminant_D_series(xy, M1_M2_rounding=True)

Returns the spectral distribution of given *CIE Illuminant D Series* using given *xy* chromaticity coordinates.

Parameters

- **xy** (array_like) – *xy* chromaticity coordinates.
- **M1_M2_rounding** (bool, optional) – Whether to round *M1* and *M2* variables to 3 decimal places in order to yield the internationally agreed values.

Returns *CIE Illuminant D Series* spectral distribution.

Return type *SpectralDistribution*

Notes

- The nominal *xy* chromaticity coordinates which have been computed with `colour.temperature.CCT_to_xy_CIE_D()` must be given according to *CIE 015:2004* recommendation and thus multiplied by 1.4388 / 1.4380.
- ***M1* and *M2* variables are rounded to 3 decimal places** according to *CIE 015:2004* recommendation.

References

[CIET14804g], [WS00d]

Examples

```

>>> from colour.utilities import numpy_print_options
>>> from colour.temperature import CCT_to_xy_CIE_D
>>> CCT_D65 = 6500 * 1.4388 / 1.4380
>>> xy = CCT_to_xy_CIE_D(CCT_D65)
>>> with numpy_print_options(suppress=True):
...     sd_CIE_illuminant_D_series(xy) # doctest: +ELLIPSIS
SpectralDistribution([[ 300.      ,  0.0341...],
                    [ 305.      ,  1.6643...],
                    [ 310.      ,  3.2945...],
                    [ 315.      , 11.7652...],
                    [ 320.      , 20.236 ...],
                    [ 325.      , 28.6447...],
                    [ 330.      , 37.0535...],
                    [ 335.      , 38.5011...],
                    [ 340.      , 39.9488...],
                    [ 345.      , 42.4302...],

```

(continues on next page)

(continued from previous page)

[350.	,	44.9117...],
[355.	,	45.775 ...],
[360.	,	46.6383...],
[365.	,	49.3637...],
[370.	,	52.0891...],
[375.	,	51.0323...],
[380.	,	49.9755...],
[385.	,	52.3118...],
[390.	,	54.6482...],
[395.	,	68.7015...],
[400.	,	82.7549...],
[405.	,	87.1204...],
[410.	,	91.486 ...],
[415.	,	92.4589...],
[420.	,	93.4318...],
[425.	,	90.0570...],
[430.	,	86.6823...],
[435.	,	95.7736...],
[440.	,	104.8649...],
[445.	,	110.9362...],
[450.	,	117.0076...],
[455.	,	117.4099...],
[460.	,	117.8122...],
[465.	,	116.3365...],
[470.	,	114.8609...],
[475.	,	115.3919...],
[480.	,	115.9229...],
[485.	,	112.3668...],
[490.	,	108.8107...],
[495.	,	109.0826...],
[500.	,	109.3545...],
[505.	,	108.5781...],
[510.	,	107.8017...],
[515.	,	106.2957...],
[520.	,	104.7898...],
[525.	,	106.2396...],
[530.	,	107.6895...],
[535.	,	106.0475...],
[540.	,	104.4055...],
[545.	,	104.2258...],
[550.	,	104.0462...],
[555.	,	102.0231...],
[560.	,	100. ...],
[565.	,	98.1671...],
[570.	,	96.3342...],
[575.	,	96.0611...],
[580.	,	95.788 ...],
[585.	,	92.2368...],
[590.	,	88.6856...],
[595.	,	89.3459...],
[600.	,	90.0062...],
[605.	,	89.8026...],
[610.	,	89.5991...],
[615.	,	88.6489...],
[620.	,	87.6987...],
[625.	,	85.4936...],

(continues on next page)

(continued from previous page)

```
[ 630.      , 83.2886...],
[ 635.      , 83.4939...],
[ 640.      , 83.6992...],
[ 645.      , 81.863 ...],
[ 650.      , 80.0268...],
[ 655.      , 80.1207...],
[ 660.      , 80.2146...],
[ 665.      , 81.2462...],
[ 670.      , 82.2778...],
[ 675.      , 80.281 ...],
[ 680.      , 78.2842...],
[ 685.      , 74.0027...],
[ 690.      , 69.7213...],
[ 695.      , 70.6652...],
[ 700.      , 71.6091...],
[ 705.      , 72.9790...],
[ 710.      , 74.349 ...],
[ 715.      , 67.9765...],
[ 720.      , 61.604 ...],
[ 725.      , 65.7448...],
[ 730.      , 69.8856...],
[ 735.      , 72.4863...],
[ 740.      , 75.087 ...],
[ 745.      , 69.3398...],
[ 750.      , 63.5927...],
[ 755.      , 55.0054...],
[ 760.      , 46.4182...],
[ 765.      , 56.6118...],
[ 770.      , 66.8054...],
[ 775.      , 65.0941...],
[ 780.      , 63.3828...],
[ 785.      , 63.8434...],
[ 790.      , 64.304 ...],
[ 795.      , 61.8779...],
[ 800.      , 59.4519...],
[ 805.      , 55.7054...],
[ 810.      , 51.959 ...],
[ 815.      , 54.6998...],
[ 820.      , 57.4406...],
[ 825.      , 58.8765...],
[ 830.      , 60.3125...]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})
```

colour.sd_blackbody

`colour.sd_blackbody(temperature, shape=SpectralShape(360, 780, 1), c1=3.741771e-16, c2=0.014388, n=1)`

Returns the spectral distribution of the planckian radiator for given temperature $T[K]$.

Parameters

- **temperature** (numeric) – Temperature $T[K]$ in kelvin degrees.

- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the spectral distribution of the planckian radiator.
- **c1** (numeric, optional) – The official value of c_1 is provided by the Committee on Data for Science and Technology (CODATA) and is $c_1 = 3,741771 \times 10^{16} \text{ W/m}_2$ (*Mohr and Taylor, 2000*).
- **c2** (numeric, optional) – Since T is measured on the International Temperature Scale, the value of c_2 used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) (*Preston-Thomas, 1990; Mielenz et al., 1991*), namely $c_2 = 1,4388 \times 10^{-2} \text{ m/K}$.
- **n** (numeric, optional) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Blackbody spectral distribution.

Return type [SpectralDistribution](#)

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> from colour.utilities import numpy_print_options
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> with numpy_print_options(suppress=True):
...     sd_blackbody(5000, cmfs.shape) # doctest: +ELLIPSIS
SpectralDistribution([[ 3.60000000e+02,  6.65427827e+12],
                    [ 3.61000000e+02,  6.70960528e+12],
                    [ 3.62000000e+02,  6.76482512e+12],
                    [ 3.63000000e+02,  6.81993308e+12],
                    [ 3.64000000e+02,  6.87492449e+12],
                    [ 3.65000000e+02,  6.92979475e+12],
                    [ 3.66000000e+02,  6.98453932e+12],
                    [ 3.67000000e+02,  7.03915372e+12],
                    [ 3.68000000e+02,  7.09363351e+12],
                    [ 3.69000000e+02,  7.14797433e+12],
                    [ 3.70000000e+02,  7.20217187e+12],
                    [ 3.71000000e+02,  7.25622190e+12],
                    [ 3.72000000e+02,  7.31012021e+12],
                    [ 3.73000000e+02,  7.36386268e+12],
                    [ 3.74000000e+02,  7.41744525e+12],
                    [ 3.75000000e+02,  7.47086391e+12],
                    [ 3.76000000e+02,  7.52411471e+12],
                    [ 3.77000000e+02,  7.57719377e+12],
                    [ 3.78000000e+02,  7.63009726e+12],
                    [ 3.79000000e+02,  7.68282141e+12],
                    [ 3.80000000e+02,  7.73536252e+12],
                    [ 3.81000000e+02,  7.78771695e+12],
                    [ 3.82000000e+02,  7.83988111e+12],
                    [ 3.83000000e+02,  7.89185148e+12],
                    [ 3.84000000e+02,  7.94362458e+12],
                    [ 3.85000000e+02,  7.99519703e+12],
                    [ 3.86000000e+02,  8.04656547e+12],
                    [ 3.87000000e+02,  8.09772662e+12],
                    [ 3.88000000e+02,  8.14867726e+12],
                    [ 3.89000000e+02,  8.19941421e+12],
                    [ 3.90000000e+02,  8.24993438e+12],
```

(continues on next page)

(continued from previous page)

```
[ 3.91000000e+02, 8.30023471e+12],
[ 3.92000000e+02, 8.35031222e+12],
[ 3.93000000e+02, 8.40016398e+12],
[ 3.94000000e+02, 8.44978711e+12],
[ 3.95000000e+02, 8.49917881e+12],
[ 3.96000000e+02, 8.54833632e+12],
[ 3.97000000e+02, 8.59725693e+12],
[ 3.98000000e+02, 8.64593802e+12],
[ 3.99000000e+02, 8.69437700e+12],
[ 4.00000000e+02, 8.74257133e+12],
[ 4.01000000e+02, 8.79051856e+12],
[ 4.02000000e+02, 8.83821626e+12],
[ 4.03000000e+02, 8.88566209e+12],
[ 4.04000000e+02, 8.93285373e+12],
[ 4.05000000e+02, 8.97978893e+12],
[ 4.06000000e+02, 9.02646551e+12],
[ 4.07000000e+02, 9.07288133e+12],
[ 4.08000000e+02, 9.11903431e+12],
[ 4.09000000e+02, 9.16492240e+12],
[ 4.10000000e+02, 9.21054364e+12],
[ 4.11000000e+02, 9.25589609e+12],
[ 4.12000000e+02, 9.30097789e+12],
[ 4.13000000e+02, 9.34578722e+12],
[ 4.14000000e+02, 9.39032230e+12],
[ 4.15000000e+02, 9.43458143e+12],
[ 4.16000000e+02, 9.47856292e+12],
[ 4.17000000e+02, 9.52226517e+12],
[ 4.18000000e+02, 9.56568661e+12],
[ 4.19000000e+02, 9.60882571e+12],
[ 4.20000000e+02, 9.65168102e+12],
[ 4.21000000e+02, 9.69425111e+12],
[ 4.22000000e+02, 9.73653461e+12],
[ 4.23000000e+02, 9.77853020e+12],
[ 4.24000000e+02, 9.82023659e+12],
[ 4.25000000e+02, 9.86165257e+12],
[ 4.26000000e+02, 9.90277693e+12],
[ 4.27000000e+02, 9.94360856e+12],
[ 4.28000000e+02, 9.98414634e+12],
[ 4.29000000e+02, 1.00243892e+13],
[ 4.30000000e+02, 1.00643363e+13],
[ 4.31000000e+02, 1.01039864e+13],
[ 4.32000000e+02, 1.01433388e+13],
[ 4.33000000e+02, 1.01823926e+13],
[ 4.34000000e+02, 1.02211468e+13],
[ 4.35000000e+02, 1.02596009e+13],
[ 4.36000000e+02, 1.02977539e+13],
[ 4.37000000e+02, 1.03356052e+13],
[ 4.38000000e+02, 1.03731541e+13],
[ 4.39000000e+02, 1.04104000e+13],
[ 4.40000000e+02, 1.04473423e+13],
[ 4.41000000e+02, 1.04839805e+13],
[ 4.42000000e+02, 1.05203140e+13],
[ 4.43000000e+02, 1.05563424e+13],
[ 4.44000000e+02, 1.05920652e+13],
[ 4.45000000e+02, 1.06274821e+13],
[ 4.46000000e+02, 1.06625927e+13],
```

(continues on next page)

(continued from previous page)

[4.47000000e+02,	1.06973967e+13],
[4.48000000e+02,	1.07318937e+13],
[4.49000000e+02,	1.07660835e+13],
[4.50000000e+02,	1.07999660e+13],
[4.51000000e+02,	1.08335408e+13],
[4.52000000e+02,	1.08668080e+13],
[4.53000000e+02,	1.08997673e+13],
[4.54000000e+02,	1.09324187e+13],
[4.55000000e+02,	1.09647621e+13],
[4.56000000e+02,	1.09967975e+13],
[4.57000000e+02,	1.10285249e+13],
[4.58000000e+02,	1.10599443e+13],
[4.59000000e+02,	1.10910559e+13],
[4.60000000e+02,	1.11218598e+13],
[4.61000000e+02,	1.11523560e+13],
[4.62000000e+02,	1.11825447e+13],
[4.63000000e+02,	1.12124262e+13],
[4.64000000e+02,	1.12420006e+13],
[4.65000000e+02,	1.12712681e+13],
[4.66000000e+02,	1.13002292e+13],
[4.67000000e+02,	1.13288840e+13],
[4.68000000e+02,	1.13572329e+13],
[4.69000000e+02,	1.13852762e+13],
[4.70000000e+02,	1.14130144e+13],
[4.71000000e+02,	1.14404478e+13],
[4.72000000e+02,	1.14675768e+13],
[4.73000000e+02,	1.14944020e+13],
[4.74000000e+02,	1.15209237e+13],
[4.75000000e+02,	1.15471425e+13],
[4.76000000e+02,	1.15730590e+13],
[4.77000000e+02,	1.15986736e+13],
[4.78000000e+02,	1.16239869e+13],
[4.79000000e+02,	1.16489996e+13],
[4.80000000e+02,	1.16737122e+13],
[4.81000000e+02,	1.16981253e+13],
[4.82000000e+02,	1.17222397e+13],
[4.83000000e+02,	1.17460559e+13],
[4.84000000e+02,	1.17695747e+13],
[4.85000000e+02,	1.17927969e+13],
[4.86000000e+02,	1.18157230e+13],
[4.87000000e+02,	1.18383540e+13],
[4.88000000e+02,	1.18606904e+13],
[4.89000000e+02,	1.18827333e+13],
[4.90000000e+02,	1.19044832e+13],
[4.91000000e+02,	1.19259412e+13],
[4.92000000e+02,	1.19471079e+13],
[4.93000000e+02,	1.19679843e+13],
[4.94000000e+02,	1.19885712e+13],
[4.95000000e+02,	1.20088695e+13],
[4.96000000e+02,	1.20288802e+13],
[4.97000000e+02,	1.20486041e+13],
[4.98000000e+02,	1.20680421e+13],
[4.99000000e+02,	1.20871953e+13],
[5.00000000e+02,	1.21060645e+13],
[5.01000000e+02,	1.21246508e+13],
[5.02000000e+02,	1.21429552e+13],

(continues on next page)

(continued from previous page)

```

[ 5.03000000e+02, 1.21609785e+13],
[ 5.04000000e+02, 1.21787220e+13],
[ 5.05000000e+02, 1.21961865e+13],
[ 5.06000000e+02, 1.22133731e+13],
[ 5.07000000e+02, 1.22302829e+13],
[ 5.08000000e+02, 1.22469170e+13],
[ 5.09000000e+02, 1.22632763e+13],
[ 5.10000000e+02, 1.22793620e+13],
[ 5.11000000e+02, 1.22951752e+13],
[ 5.12000000e+02, 1.23107171e+13],
[ 5.13000000e+02, 1.23259886e+13],
[ 5.14000000e+02, 1.23409909e+13],
[ 5.15000000e+02, 1.23557252e+13],
[ 5.16000000e+02, 1.23701926e+13],
[ 5.17000000e+02, 1.23843943e+13],
[ 5.18000000e+02, 1.23983314e+13],
[ 5.19000000e+02, 1.24120051e+13],
[ 5.20000000e+02, 1.24254166e+13],
[ 5.21000000e+02, 1.24385670e+13],
[ 5.22000000e+02, 1.24514576e+13],
[ 5.23000000e+02, 1.24640896e+13],
[ 5.24000000e+02, 1.24764641e+13],
[ 5.25000000e+02, 1.24885824e+13],
[ 5.26000000e+02, 1.25004457e+13],
[ 5.27000000e+02, 1.25120552e+13],
[ 5.28000000e+02, 1.25234122e+13],
[ 5.29000000e+02, 1.25345178e+13],
[ 5.30000000e+02, 1.25453735e+13],
[ 5.31000000e+02, 1.25559803e+13],
[ 5.32000000e+02, 1.25663396e+13],
[ 5.33000000e+02, 1.25764527e+13],
[ 5.34000000e+02, 1.25863207e+13],
[ 5.35000000e+02, 1.25959449e+13],
[ 5.36000000e+02, 1.26053268e+13],
[ 5.37000000e+02, 1.26144674e+13],
[ 5.38000000e+02, 1.26233681e+13],
[ 5.39000000e+02, 1.26320302e+13],
[ 5.40000000e+02, 1.26404551e+13],
[ 5.41000000e+02, 1.26486438e+13],
[ 5.42000000e+02, 1.26565979e+13],
[ 5.43000000e+02, 1.26643185e+13],
[ 5.44000000e+02, 1.26718071e+13],
[ 5.45000000e+02, 1.26790648e+13],
[ 5.46000000e+02, 1.26860930e+13],
[ 5.47000000e+02, 1.26928930e+13],
[ 5.48000000e+02, 1.26994662e+13],
[ 5.49000000e+02, 1.27058138e+13],
[ 5.50000000e+02, 1.27119372e+13],
[ 5.51000000e+02, 1.27178376e+13],
[ 5.52000000e+02, 1.27235164e+13],
[ 5.53000000e+02, 1.27289750e+13],
[ 5.54000000e+02, 1.27342146e+13],
[ 5.55000000e+02, 1.27392366e+13],
[ 5.56000000e+02, 1.27440423e+13],
[ 5.57000000e+02, 1.27486330e+13],
[ 5.58000000e+02, 1.27530100e+13],

```

(continues on next page)

(continued from previous page)

[5.59000000e+02,	1.27571748e+13],
[5.60000000e+02,	1.27611285e+13],
[5.61000000e+02,	1.27648725e+13],
[5.62000000e+02,	1.27684083e+13],
[5.63000000e+02,	1.27717370e+13],
[5.64000000e+02,	1.27748600e+13],
[5.65000000e+02,	1.27777787e+13],
[5.66000000e+02,	1.27804943e+13],
[5.67000000e+02,	1.27830082e+13],
[5.68000000e+02,	1.27853217e+13],
[5.69000000e+02,	1.27874362e+13],
[5.70000000e+02,	1.27893529e+13],
[5.71000000e+02,	1.27910732e+13],
[5.72000000e+02,	1.27925984e+13],
[5.73000000e+02,	1.27939299e+13],
[5.74000000e+02,	1.27950689e+13],
[5.75000000e+02,	1.27960167e+13],
[5.76000000e+02,	1.27967747e+13],
[5.77000000e+02,	1.27973442e+13],
[5.78000000e+02,	1.27977264e+13],
[5.79000000e+02,	1.27979228e+13],
[5.80000000e+02,	1.27979346e+13],
[5.81000000e+02,	1.27977630e+13],
[5.82000000e+02,	1.27974095e+13],
[5.83000000e+02,	1.27968753e+13],
[5.84000000e+02,	1.27961617e+13],
[5.85000000e+02,	1.27952700e+13],
[5.86000000e+02,	1.27942015e+13],
[5.87000000e+02,	1.27929575e+13],
[5.88000000e+02,	1.27915392e+13],
[5.89000000e+02,	1.27899480e+13],
[5.90000000e+02,	1.27881852e+13],
[5.91000000e+02,	1.27862519e+13],
[5.92000000e+02,	1.27841495e+13],
[5.93000000e+02,	1.27818793e+13],
[5.94000000e+02,	1.27794424e+13],
[5.95000000e+02,	1.27768403e+13],
[5.96000000e+02,	1.27740741e+13],
[5.97000000e+02,	1.27711451e+13],
[5.98000000e+02,	1.27680546e+13],
[5.99000000e+02,	1.27648037e+13],
[6.00000000e+02,	1.27613938e+13],
[6.01000000e+02,	1.27578261e+13],
[6.02000000e+02,	1.27541018e+13],
[6.03000000e+02,	1.27502222e+13],
[6.04000000e+02,	1.27461885e+13],
[6.05000000e+02,	1.27420020e+13],
[6.06000000e+02,	1.27376637e+13],
[6.07000000e+02,	1.27331750e+13],
[6.08000000e+02,	1.27285371e+13],
[6.09000000e+02,	1.27237512e+13],
[6.10000000e+02,	1.27188185e+13],
[6.11000000e+02,	1.27137402e+13],
[6.12000000e+02,	1.27085175e+13],
[6.13000000e+02,	1.27031516e+13],
[6.14000000e+02,	1.26976436e+13],

(continues on next page)

(continued from previous page)

```
[ 6.15000000e+02, 1.26919949e+13],
[ 6.16000000e+02, 1.26862064e+13],
[ 6.17000000e+02, 1.26802795e+13],
[ 6.18000000e+02, 1.26742153e+13],
[ 6.19000000e+02, 1.26680149e+13],
[ 6.20000000e+02, 1.26616795e+13],
[ 6.21000000e+02, 1.26552103e+13],
[ 6.22000000e+02, 1.26486085e+13],
[ 6.23000000e+02, 1.26418751e+13],
[ 6.24000000e+02, 1.26350113e+13],
[ 6.25000000e+02, 1.26280183e+13],
[ 6.26000000e+02, 1.26208972e+13],
[ 6.27000000e+02, 1.26136491e+13],
[ 6.28000000e+02, 1.26062751e+13],
[ 6.29000000e+02, 1.25987764e+13],
[ 6.30000000e+02, 1.25911540e+13],
[ 6.31000000e+02, 1.25834092e+13],
[ 6.32000000e+02, 1.25755429e+13],
[ 6.33000000e+02, 1.25675563e+13],
[ 6.34000000e+02, 1.25594505e+13],
[ 6.35000000e+02, 1.25512265e+13],
[ 6.36000000e+02, 1.25428855e+13],
[ 6.37000000e+02, 1.25344285e+13],
[ 6.38000000e+02, 1.25258566e+13],
[ 6.39000000e+02, 1.25171709e+13],
[ 6.40000000e+02, 1.25083724e+13],
[ 6.41000000e+02, 1.24994622e+13],
[ 6.42000000e+02, 1.24904413e+13],
[ 6.43000000e+02, 1.24813108e+13],
[ 6.44000000e+02, 1.24720718e+13],
[ 6.45000000e+02, 1.24627252e+13],
[ 6.46000000e+02, 1.24532721e+13],
[ 6.47000000e+02, 1.24437136e+13],
[ 6.48000000e+02, 1.24340506e+13],
[ 6.49000000e+02, 1.24242842e+13],
[ 6.50000000e+02, 1.24144153e+13],
[ 6.51000000e+02, 1.24044450e+13],
[ 6.52000000e+02, 1.23943743e+13],
[ 6.53000000e+02, 1.23842042e+13],
[ 6.54000000e+02, 1.23739356e+13],
[ 6.55000000e+02, 1.23635696e+13],
[ 6.56000000e+02, 1.23531072e+13],
[ 6.57000000e+02, 1.23425492e+13],
[ 6.58000000e+02, 1.23318967e+13],
[ 6.59000000e+02, 1.23211506e+13],
[ 6.60000000e+02, 1.23103120e+13],
[ 6.61000000e+02, 1.22993816e+13],
[ 6.62000000e+02, 1.22883606e+13],
[ 6.63000000e+02, 1.22772498e+13],
[ 6.64000000e+02, 1.22660502e+13],
[ 6.65000000e+02, 1.22547627e+13],
[ 6.66000000e+02, 1.22433883e+13],
[ 6.67000000e+02, 1.22319278e+13],
[ 6.68000000e+02, 1.22203821e+13],
[ 6.69000000e+02, 1.22087523e+13],
[ 6.70000000e+02, 1.21970391e+13],
```

(continues on next page)

(continued from previous page)

[6.71000000e+02,	1.21852435e+13],
[6.72000000e+02,	1.21733664e+13],
[6.73000000e+02,	1.21614087e+13],
[6.74000000e+02,	1.21493712e+13],
[6.75000000e+02,	1.21372548e+13],
[6.76000000e+02,	1.21250605e+13],
[6.77000000e+02,	1.21127890e+13],
[6.78000000e+02,	1.21004413e+13],
[6.79000000e+02,	1.20880182e+13],
[6.80000000e+02,	1.20755205e+13],
[6.81000000e+02,	1.20629491e+13],
[6.82000000e+02,	1.20503049e+13],
[6.83000000e+02,	1.20375887e+13],
[6.84000000e+02,	1.20248012e+13],
[6.85000000e+02,	1.20119434e+13],
[6.86000000e+02,	1.19990161e+13],
[6.87000000e+02,	1.19860200e+13],
[6.88000000e+02,	1.19729560e+13],
[6.89000000e+02,	1.19598249e+13],
[6.90000000e+02,	1.19466275e+13],
[6.91000000e+02,	1.19333646e+13],
[6.92000000e+02,	1.19200370e+13],
[6.93000000e+02,	1.19066454e+13],
[6.94000000e+02,	1.18931907e+13],
[6.95000000e+02,	1.18796736e+13],
[6.96000000e+02,	1.18660949e+13],
[6.97000000e+02,	1.18524554e+13],
[6.98000000e+02,	1.18387558e+13],
[6.99000000e+02,	1.18249969e+13],
[7.00000000e+02,	1.18111794e+13],
[7.01000000e+02,	1.17973040e+13],
[7.02000000e+02,	1.17833716e+13],
[7.03000000e+02,	1.17693829e+13],
[7.04000000e+02,	1.17553385e+13],
[7.05000000e+02,	1.17412392e+13],
[7.06000000e+02,	1.17270858e+13],
[7.07000000e+02,	1.17128789e+13],
[7.08000000e+02,	1.16986192e+13],
[7.09000000e+02,	1.16843075e+13],
[7.10000000e+02,	1.16699445e+13],
[7.11000000e+02,	1.16555309e+13],
[7.12000000e+02,	1.16410673e+13],
[7.13000000e+02,	1.16265544e+13],
[7.14000000e+02,	1.16119930e+13],
[7.15000000e+02,	1.15973836e+13],
[7.16000000e+02,	1.15827271e+13],
[7.17000000e+02,	1.15680240e+13],
[7.18000000e+02,	1.15532749e+13],
[7.19000000e+02,	1.15384807e+13],
[7.20000000e+02,	1.15236419e+13],
[7.21000000e+02,	1.15087591e+13],
[7.22000000e+02,	1.14938331e+13],
[7.23000000e+02,	1.14788644e+13],
[7.24000000e+02,	1.14638537e+13],
[7.25000000e+02,	1.14488017e+13],
[7.26000000e+02,	1.14337088e+13],

(continues on next page)

(continued from previous page)

```

[ 7.27000000e+02, 1.14185759e+13],
[ 7.28000000e+02, 1.14034034e+13],
[ 7.29000000e+02, 1.13881921e+13],
[ 7.30000000e+02, 1.13729424e+13],
[ 7.31000000e+02, 1.13576551e+13],
[ 7.32000000e+02, 1.13423307e+13],
[ 7.33000000e+02, 1.13269698e+13],
[ 7.34000000e+02, 1.13115730e+13],
[ 7.35000000e+02, 1.12961409e+13],
[ 7.36000000e+02, 1.12806741e+13],
[ 7.37000000e+02, 1.12651731e+13],
[ 7.38000000e+02, 1.12496385e+13],
[ 7.39000000e+02, 1.12340710e+13],
[ 7.40000000e+02, 1.12184710e+13],
[ 7.41000000e+02, 1.12028391e+13],
[ 7.42000000e+02, 1.11871759e+13],
[ 7.43000000e+02, 1.11714819e+13],
[ 7.44000000e+02, 1.11557577e+13],
[ 7.45000000e+02, 1.11400038e+13],
[ 7.46000000e+02, 1.11242208e+13],
[ 7.47000000e+02, 1.11084092e+13],
[ 7.48000000e+02, 1.10925695e+13],
[ 7.49000000e+02, 1.10767023e+13],
[ 7.50000000e+02, 1.10608080e+13],
[ 7.51000000e+02, 1.10448872e+13],
[ 7.52000000e+02, 1.10289405e+13],
[ 7.53000000e+02, 1.10129683e+13],
[ 7.54000000e+02, 1.09969711e+13],
[ 7.55000000e+02, 1.09809495e+13],
[ 7.56000000e+02, 1.09649039e+13],
[ 7.57000000e+02, 1.09488348e+13],
[ 7.58000000e+02, 1.09327427e+13],
[ 7.59000000e+02, 1.09166282e+13],
[ 7.60000000e+02, 1.09004917e+13],
[ 7.61000000e+02, 1.08843336e+13],
[ 7.62000000e+02, 1.08681545e+13],
[ 7.63000000e+02, 1.08519548e+13],
[ 7.64000000e+02, 1.08357350e+13],
[ 7.65000000e+02, 1.08194956e+13],
[ 7.66000000e+02, 1.08032370e+13],
[ 7.67000000e+02, 1.07869596e+13],
[ 7.68000000e+02, 1.07706640e+13],
[ 7.69000000e+02, 1.07543506e+13],
[ 7.70000000e+02, 1.07380198e+13],
[ 7.71000000e+02, 1.07216721e+13],
[ 7.72000000e+02, 1.07053078e+13],
[ 7.73000000e+02, 1.06889276e+13],
[ 7.74000000e+02, 1.06725317e+13],
[ 7.75000000e+02, 1.06561206e+13],
[ 7.76000000e+02, 1.06396947e+13],
[ 7.77000000e+02, 1.06232545e+13],
[ 7.78000000e+02, 1.06068004e+13],
[ 7.79000000e+02, 1.05903327e+13],
[ 7.80000000e+02, 1.05738520e+13],
[ 7.81000000e+02, 1.05573585e+13],
[ 7.82000000e+02, 1.05408527e+13],

```

(continues on next page)

(continued from previous page)

```

[ 7.83000000e+02, 1.05243351e+13],
[ 7.84000000e+02, 1.05078060e+13],
[ 7.85000000e+02, 1.04912657e+13],
[ 7.86000000e+02, 1.04747147e+13],
[ 7.87000000e+02, 1.04581535e+13],
[ 7.88000000e+02, 1.04415822e+13],
[ 7.89000000e+02, 1.04250014e+13],
[ 7.90000000e+02, 1.04084115e+13],
[ 7.91000000e+02, 1.03918127e+13],
[ 7.92000000e+02, 1.03752055e+13],
[ 7.93000000e+02, 1.03585902e+13],
[ 7.94000000e+02, 1.03419672e+13],
[ 7.95000000e+02, 1.03253369e+13],
[ 7.96000000e+02, 1.03086995e+13],
[ 7.97000000e+02, 1.02920556e+13],
[ 7.98000000e+02, 1.02754053e+13],
[ 7.99000000e+02, 1.02587492e+13],
[ 8.00000000e+02, 1.02420874e+13],
[ 8.01000000e+02, 1.02254204e+13],
[ 8.02000000e+02, 1.02087485e+13],
[ 8.03000000e+02, 1.01920720e+13],
[ 8.04000000e+02, 1.01753913e+13],
[ 8.05000000e+02, 1.01587067e+13],
[ 8.06000000e+02, 1.01420186e+13],
[ 8.07000000e+02, 1.01253271e+13],
[ 8.08000000e+02, 1.01086328e+13],
[ 8.09000000e+02, 1.00919358e+13],
[ 8.10000000e+02, 1.00752366e+13],
[ 8.11000000e+02, 1.00585353e+13],
[ 8.12000000e+02, 1.00418324e+13],
[ 8.13000000e+02, 1.00251282e+13],
[ 8.14000000e+02, 1.00084228e+13],
[ 8.15000000e+02, 9.99171677e+12],
[ 8.16000000e+02, 9.97501023e+12],
[ 8.17000000e+02, 9.95830354e+12],
[ 8.18000000e+02, 9.94159699e+12],
[ 8.19000000e+02, 9.92489086e+12],
[ 8.20000000e+02, 9.90818544e+12],
[ 8.21000000e+02, 9.89148102e+12],
[ 8.22000000e+02, 9.87477789e+12],
[ 8.23000000e+02, 9.85807631e+12],
[ 8.24000000e+02, 9.84137658e+12],
[ 8.25000000e+02, 9.82467896e+12],
[ 8.26000000e+02, 9.80798374e+12],
[ 8.27000000e+02, 9.79129116e+12],
[ 8.28000000e+02, 9.77460152e+12],
[ 8.29000000e+02, 9.75791506e+12],
[ 8.30000000e+02, 9.74123205e+12]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})

```

colour.sd_constant

`colour.sd_constant(k, shape=SpectralShape(360, 780, 1), dtype=<class 'numpy.float64'>)`
 Returns a spectral distribution of given spectral shape filled with constant k values.

Parameters

- **k** (numeric) – Constant k to fill the spectral distribution with.
- **shape** (`SpectralShape`, optional) – Spectral shape used to create the spectral distribution.
- **dtype** (`type`) – Data type used for the spectral distribution.

Returns Constant k to filled spectral distribution.

Return type `SpectralDistribution`

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_constant(100)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
100.0
```

colour.sd_ones

`colour.sd_ones(shape=SpectralShape(360, 780, 1))`
 Returns a spectral distribution of given spectral shape filled with ones.

Parameters **shape** (`SpectralShape`, optional) – Spectral shape used to create the spectral distribution.

Returns Ones filled spectral distribution.

Return type `SpectralDistribution`

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_ones()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
1.0
```

colour.sd_zeros

colour.sd_zeros(shape=SpectralShape(360, 780, 1))

Returns a spectral distribution of given spectral shape filled with zeros.

Parameters **shape** (*SpectralShape*, optional) – Spectral shape used to create the spectral distribution.

Returns Zeros filled spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_zeros()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
0.0
```

colour.SD_GAUSSIAN_METHODS

colour.SD_GAUSSIAN_METHODS = CaseInsensitiveMapping({'Normal': ..., 'FWHM': ...})

Supported gaussian spectral distribution computation methods.

SD_GAUSSIAN_METHODS [CaseInsensitiveMapping] {'Normal', 'FWHM'}

colour.sd_gaussian

colour.sd_gaussian(mu_peak_wavelength, sigma_fwhm, shape=SpectralShape(360, 780, 1), method='Normal')

Returns a gaussian spectral distribution of given spectral shape using given method.

Parameters

- **mu_peak_wavelength** (numeric) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma_fwhm** (numeric) – Standard deviation *sigma* of the gaussian spectral distribution or Full width at half maximum, i.e. width of the gaussian spectral distribution

measured between those points on the y axis which are half the maximum amplitude.

- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – {'Normal', 'FWHM'}, Computation method.

Returns Gaussian spectral distribution.

Return type [SpectralDistribution](#)

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_gaussian(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555] # doctest: +ELLIPSIS
1.0000000...
>>> sd[530] # doctest: +ELLIPSIS
0.6065306...
>>> sd = sd_gaussian(555, 25, method='FWHM')
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530] # doctest: +ELLIPSIS
0.3678794...
```

colour.SD_SINGLE_LED_METHODS

`colour.SD_SINGLE_LED_METHODS = CaseInsensitiveMapping({'Ohno 2005': ...})`
Supported single *LED* spectral distribution computation methods.

SD_SINGLE_LED_METHODS [CaseInsensitiveMapping] {'Ohno 2005'}

colour.sd_single_led

`colour.sd_single_led(peak_wavelength, fwhm, shape=SpectralShape(360, 780, 1), method='Ohno 2005')`

Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to given method.

Parameters

- **peak_wavelength** (numeric) – Wavelength the single *LED* spectral distribution will peak at.

- **fwhm** (numeric) – Full width at half maximum, i.e. width of the underlying gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.
- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – {'Ohno 2005'}, Computation method.

Returns Single *LED* spectral distribution.

Return type [SpectralDistribution](#)

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555] # doctest: +ELLIPSIS
1.0000000...
```

colour.SD_MULTI_LEDS_METHODS

`colour.SD_MULTI_LEDS_METHODS = CaseInsensitiveMapping({'Ohno 2005': ...})`
Supported multi *LED* spectral distribution computation methods.

SD_MULTI_LEDS_METHODS [[CaseInsensitiveMapping](#)] {'Ohno 2005'}

colour.sd_multi_leds

`colour.sd_multi_leds(peak_wavelengths, fwhm, peak_power_ratios=None, shape=SpectralShape(360, 780, 1), method='Ohno 2005')`

Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to given method.

Parameters

- **peak_wavelengths** (array_like) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **fwhm** (array_like) – Full widths at half maximum, i.e. widths of the underlying gaussian spectral distributions measured between those points on the y axis which are half the maximum amplitude.

- **peak_power_ratios** (array_like, optional) – Peak power ratios for each generated single *LED* spectral distributions.
- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – {'Ohno 2005'}, Computation method.

Returns Multi *LED* spectral distribution.

Return type [SpectralDistribution](#)

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_multi_leds(  
...     np.array([457, 530, 615]),  
...     np.array([20, 30, 20]),  
...     np.array([0.731, 1.000, 1.660]),  
... )  
>>> sd.shape  
SpectralShape(360.0, 780.0, 1.0)  
>>> sd[500] # doctest: +ELLIPSIS  
0.1295132...
```

`colour.colorimetry`

blackbody_spectral_radiance (wavelength, ...)	Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n .
daylight_locus_function (x_D)	Returns the daylight locus as xy chromaticity coordinates.
planck_law (wavelength, temperature[, c1, c2, n])	Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n .
sd_gaussian_normal (mu, sigma[, shape])	Returns a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation σ .
sd_gaussian_fwhm (peak_wavelength, fwhm[, shape])	Returns a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.
sd_single_led_Ohno2005 (peak_wavelength, fwhm)	Returns a single <i>LED</i> spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to <i>Ohno (2005)</i> method.

Continued on next page

Table 95 – continued from previous page

<code>sd_multi_leds_Ohno2005(peak_wavelengths, fwhm)</code>	Returns a multi <i>LED</i> spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to <i>Ohno (2005)</i> method.
---	--

colour.colorimetry.blackbody_spectral_radiance

`colour.colorimetry.blackbody_spectral_radiance(wavelength, temperature, c1=3.741771e-16, c2=0.014388, n=1)`

Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n .

Parameters

- **wavelength** (numeric or array_like) – Wavelength in meters.
- **temperature** (numeric or array_like) – Temperature $T[K]$ in kelvin degrees.
- **c1** (numeric or array_like, optional) – The official value of c_1 is provided by the Committee on Data for Science and Technology (CODATA) and is $c_1 = 3,741771 \times 10^{-16} \text{ W/m}^2$ (*Mohr and Taylor, 2000*).
- **c2** (numeric or array_like, optional) – Since T is measured on the International Temperature Scale, the value of c_2 used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) (*Preston-Thomas, 1990; Mielenz et al., 1991*), namely $c_2 = 1,4388 \times 10^{-2} \text{ m/K}$.
- **n** (numeric or array_like, optional) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

[CIET14804c]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500) # doctest: +ELLIPSIS
20472701909806.5...
```

colour.colorimetry.daylight_locus_function

colour.colorimetry.**daylight_locus_function**(*x_D*)

Returns the daylight locus as *xy* chromaticity coordinates.

Parameters *x_D* (numeric or array_like) – *x* chromaticity coordinates

Returns Daylight locus as *xy* chromaticity coordinates.

Return type numeric or array_like

References

[WS00a]

Examples

```
>>> daylight_locus_function(0.31270) # doctest: +ELLIPSIS
0.3291051...
```

colour.colorimetry.planck_law

colour.colorimetry.**planck_law**(*wavelength*, *temperature*, *c1*=3.741771e-16, *c2*=0.014388, *n*=1)

Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n .

Parameters

- **wavelength** (numeric or array_like) – Wavelength in meters.
- **temperature** (numeric or array_like) – Temperature $T[K]$ in kelvin degrees.
- **c1** (numeric or array_like, optional) – The official value of c_1 is provided by the Committee on Data for Science and Technology (CODATA) and is $c_1 = 3,741771 \times 10^{-16} \text{ W/m}^2$ (Mohr and Taylor, 2000).
- **c2** (numeric or array_like, optional) – Since T is measured on the International Temperature Scale, the value of c_2 used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) (Preston-Thomas, 1990; Mielenz et al., 1991), namely $c_2 = 1,4388 \times 10^{-2} \text{ m/K}$.
- **n** (numeric or array_like, optional) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although CIE 15:2004 recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

[CIET14804c]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500) # doctest: +ELLIPSIS
20472701909806.5...
```

colour.colorimetry.sd_gaussian_normal

colour.colorimetry.**sd_gaussian_normal**(*mu*, *sigma*, *shape*=*SpectralShape*(360, 780, 1))

Returns a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation *sigma*.

Parameters

- **mu** (numeric) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma** (numeric) – Standard deviation *sigma* of the gaussian spectral distribution.
- **shape** (*SpectralShape*, optional) – Spectral shape used to create the spectral distribution.

Returns Gaussian spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_gaussian_normal(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555] # doctest: +ELLIPSIS
1.0000000...
>>> sd[530] # doctest: +ELLIPSIS
0.6065306...
```

colour.colorimetry.sd_gaussian_fwhm

colour.colorimetry.**sd_gaussian_fwhm**(*peak_wavelength*, *fwhm*, *shape*=*SpectralShape*(360, 780, 1))

Returns a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.

Parameters

- **peak_wavelength** (numeric) – Wavelength the gaussian spectral distribution will peak at.
- **fwhm** (numeric) – Full width at half maximum, i.e. width of the gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.
- **shape** (`SpectralShape`, optional) – Spectral shape used to create the spectral distribution.

Returns Gaussian spectral distribution.

Return type `SpectralDistribution`

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

Examples

```
>>> sd = sd_gaussian_fwhm(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530] # doctest: +ELLIPSIS
0.3678794...
```

`colour.colorimetry.sd_single_led_Ohno2005`

`colour.colorimetry.sd_single_led_Ohno2005(peak_wavelength, fwhm, shape=SpectralShape(360, 780, 1))`

Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to *Ohno (2005)* method.

Parameters

- **peak_wavelength** (numeric) – Wavelength the single *LED* spectral distribution will peak at.
- **fwhm** (numeric) – Full width at half maximum, i.e. width of the underlying gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.
- **shape** (`SpectralShape`, optional) – Spectral shape used to create the spectral distribution.

Returns Single *LED* spectral distribution.

Return type `SpectralDistribution`

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led_Ohno2005(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555] # doctest: +ELLIPSIS
1.0000000...
```

colour.colorimetry.sd_multi_leds_Ohno2005

`colour.colorimetry.sd_multi_leds_Ohno2005(peak_wavelengths, fwhm, peak_power_ratios=None, shape=SpectralShape(360, 780, 1))`

Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to *Ohno (2005)* method.

The multi *LED* spectral distribution is generated using many single *LED* spectral distributions generated with `colour.sd_single_led_Ohno2005()` definition.

Parameters

- **peak_wavelengths** (array_like) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **fwhm** (array_like) – Full widths at half maximum, i.e. widths of the underlying gaussian spectral distributions measured between those points on the *y* axis which are half the maximum amplitude.
- **peak_power_ratios** (array_like, optional) – Peak power ratios for each generated single *LED* spectral distributions.
- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the spectral distribution.

Returns Multi *LED* spectral distribution.

Return type [SpectralDistribution](#)

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```

>>> sd = sd_multi_leds_Ohno2005(
...     np.array([457, 530, 615]),
...     np.array([20, 30, 20]),
...     np.array([0.731, 1.000, 1.660]),
... )
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[500] # doctest: +ELLIPSIS
0.1295132...

```

Conversion to Tristimulus Values

colour

<code>sd_to_XYZ(sd[, cmfs, illuminant, k, method])</code>	Converts given spectral distribution to <i>CIE XYZ</i> tristimulus values using given colour matching functions, illuminant and method.
<code>SD_TO_XYZ_METHODS</code>	Supported spectral distribution to <i>CIE XYZ</i> tristimulus values conversion methods.
<code>multi_sds_to_XYZ(msd[, shape, cmfs, ...])</code>	Converts given multi-spectral distribution array <i>msd</i> with given spectral shape to <i>CIE XYZ</i> tristimulus values using given colour matching functions and illuminant.
<code>MULTI_SD_TO_XYZ_METHODS</code>	Supported multi-spectral array to <i>CIE XYZ</i> tristimulus values conversion methods.
<code>wavelength_to_XYZ(wavelength[, cmfs])</code>	Converts given wavelength λ to <i>CIE XYZ</i> tristimulus values using given colour matching functions.

colour.sd_to_XYZ

`colour.sd_to_XYZ(sd, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralDistribution(name='1 Constant', ...), k=None, method='ASTM E308-15', **kwargs)`

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.

Parameters

- **sd** (*SpectralDistribution*) – Spectral distribution.
- **cmfs** (*XYZ_ColourMatchingFunctions*) – Standard observer colour matching functions.
- **illuminant** (*SpectralDistribution*, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k . For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal

to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

- **method** (unicode, optional) – {'ASTM E308-15', 'Integration'}, Computation method.

Other Parameters

- **mi_5nm_omission_method** (*bool*, *optional*) – {`colour.colorimetry.sd_to_XYZ_ASTME30815()`}, 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (*bool*, *optional*) – {`colour.colorimetry.sd_to_XYZ_ASTME30815()`}, 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **use_practice_range** (*bool*, *optional*) – {`colour.colorimetry.sd_to_XYZ_ASTME30815()`}, Practise ASTM E308-15 working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.

Returns CIE XYZ tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational11], [ASTMInternational15], [WS00f]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
```

(continues on next page)

(continued from previous page)

```

...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ(sd, cmfs, illuminant)
... # doctest: +ELLIPSIS
array([ 10.8399031...,  9.6840375...,  6.2164159...])
>>> sd_to_XYZ(sd, cmfs, illuminant, use_practice_range=False)
... # doctest: +ELLIPSIS
array([ 10.8399852...,  9.6840602...,  6.2164085...])
>>> sd_to_XYZ(sd, cmfs, illuminant, method='Integration')
... # doctest: +ELLIPSIS
array([ 10.8401846...,  9.6837311...,  6.2120912...])

```

colour.SD_TO_XYZ_METHODS

`colour.SD_TO_XYZ_METHODS` = `CaseInsensitiveMapping({'ASTM E308-15': ..., 'Integration': ..., 'astm2015': ...})`
Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.

References

[[ASTMInternational11](#)], [[ASTMInternational15](#)], [[WS00f](#)]

SD_TO_XYZ_METHODS [`CaseInsensitiveMapping`] {'ASTM E308-15', 'Integration'}

Aliases:

- 'astm2015': 'ASTM E308-15'

colour.multi_sds_to_XYZ

```

colour.multi_sds_to_XYZ(msd, shape=SpectralShape(360, 780, 1),
                        cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Stan-
                        dard Observer', ...), illuminant=SpectralDistribution(name='1 Constant',
                        ...), method='Integration', **kwargs)

```

Converts given multi-spectral distribution array *msd* with given spectral shape to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.

Parameters

- **msa** (`array_like`) – Multi-spectral distribution array *msd*, the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, *msd* shape should be (384, 512, 77).
- **shape** (`SpectralShape`, optional) – Spectral shape of the multi-spectral distribution array *msd*, *cmfs* and *illuminant* will be aligned with it.
- **cmfs** (`XYZ_ColourMatchingFunctions`) – Standard observer colour matching functions.
- **illuminant** (`SpectralDistribution`, optional) – Illuminant spectral distribution.
- **method** (`unicode`, optional) – {'Integration'}, Computation method.

Other Parameters `k` (*numeric, optional*) – `{colour.colorimetry.multi_sds_to_XYZ_integration()}`, Normalisation constant k . For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns CIE XYZ tristimulus values, for a 512x384 multi-spectral image with 77 bins, the output shape will be (384, 512, 3).

Return type `array_like`

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[WS00f]

Examples

```
>>> msd = np.array([
...     [
...         [0.0137, 0.0913, 0.0152, 0.0281, 0.1918, 0.0430],
...         [0.0159, 0.3145, 0.0842, 0.0907, 0.7103, 0.0437],
...         [0.0096, 0.2582, 0.4139, 0.2228, 0.0041, 0.3744],
...         [0.0111, 0.0709, 0.0220, 0.1249, 0.1817, 0.0020],
...         [0.0179, 0.2971, 0.5630, 0.2375, 0.0024, 0.5819],
...         [0.1057, 0.4620, 0.1918, 0.5625, 0.4209, 0.0027],
...     ],
...     [
...         [0.0433, 0.2683, 0.2373, 0.0518, 0.0118, 0.0823],
...         [0.0258, 0.0831, 0.0430, 0.3230, 0.2302, 0.0081],
...         [0.0248, 0.1203, 0.0054, 0.0065, 0.1860, 0.3625],
...         [0.0186, 0.1292, 0.0079, 0.4006, 0.9404, 0.3213],
...         [0.0310, 0.1682, 0.3719, 0.0861, 0.0041, 0.7849],
...         [0.0473, 0.3221, 0.2268, 0.3161, 0.1124, 0.0024],
...     ],
... ])
>>> multi_sds_to_XYZ(msd, SpectralShape(400, 700, 60))
... # doctest: +ELLIPSIS
array([[ 7.6862675...,  4.0925470...,  8.4950412...],
       [27.4119366..., 15.5014764..., 29.2825122...],
       [17.1283666..., 27.7798651..., 25.5232032...],
       [11.9824544...,  8.8127109...,  6.6518695...],
       [19.1030682..., 34.4597818..., 29.7653804...],
```

(continues on next page)

(continued from previous page)

```
[ 46.8243374..., 39.9551652..., 43.6541858...]],
<BLANKLINE>
[[ 8.0978189..., 12.7544378..., 25.8004512...],
 [ 23.4360673..., 19.6127966..., 7.9342408...],
 [ 7.0933208..., 2.7894394..., 11.1527704...],
 [ 45.6313772..., 29.0068105..., 11.9934522...],
 [ 8.9327884..., 19.4008147..., 17.1534186...],
 [ 24.6610235..., 26.1093760..., 30.7298791...]]])
```

colour.MULTI_SD_TO_XYZ_METHODS

`colour.MULTI_SD_TO_XYZ_METHODS = CaseInsensitiveMapping({'Integration': ...})`

Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.

References

[WS00f]

`MULTI_SD_TO_XYZ_METHODS` [CaseInsensitiveMapping] {'Integration'}

colour.wavelength_to_XYZ

`colour.wavelength_to_XYZ(wavelength, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...))`

Converts given wavelength λ to *CIE XYZ* tristimulus values using given colour matching functions.

If the wavelength λ is not available in the colour matching function, its value will be calculated according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable.

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in nm.
- **cmfs** (*XYZ_ColourMatchingFunctions*, optional) – Standard observer colour matching functions.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Raises *ValueError* – If wavelength λ is not contained in the colour matching functions domain.

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> from colour import CMFS
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> wavelength_to_XYZ(480, cmfs) # doctest: +ELLIPSIS
array([ 0.09564 , 0.13902 , 0.8129501...])
>>> wavelength_to_XYZ(480.5, cmfs) # doctest: +ELLIPSIS
array([ 0.0914287..., 0.1418350..., 0.7915726...])
```

ASTM E308-15

colour.colorimetry

<code>sd_to_XYZ_ASTME30815(sd[, cmfs, illuminant, ...])</code>	Converts given spectral distribution to <i>CIE XYZ</i> tristimulus values using given colour matching functions and illuminant according to practise <i>ASTM E308-15</i> method.
--	--

colour.colorimetry.sd_to_XYZ_ASTME30815

```
colour.colorimetry.sd_to_XYZ_ASTME30815(sd, cmfs=XYZ_ColourMatchingFunctions(name='CIE
1931 2 Degree Standard Observer', ...),
illuminant=SpectralDistribution(name='1
Constant', ...), use_practice_range=True,
mi_5nm_omission_method=True,
mi_20nm_interpolation_method=True, k=None)
```

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **sd** (*SpectralDistribution*) – Spectral distribution.
- **cmfs** (*XYZ_ColourMatchingFunctions*) – Standard observer colour matching functions.
- **illuminant** (*SpectralDistribution*, optional) – Illuminant spectral distribution.
- **use_practice_range** (*bool*, optional) – Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **mi_5nm_omission_method** (*bool*, optional) – 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (*bool*, optional) – 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **k** (*numeric*, optional) – Normalisation constant k . For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the *CIE*

1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns CIE XYZ tristimulus values.

Return type ndarray, (3,)

Warning:

- The tables of tristimulus weighting factors are cached in `colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE` attribute. Their identifier key is defined by the colour matching functions and illuminant names along the current shape such as: *CIE 1964 10 Degree Standard Observer; A, (360.0, 830.0, 10.0)* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational15]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
```

(continues on next page)

(continued from previous page)

```
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_ASTME30815(sd, cmfs, illuminant)
... # doctest: +ELLIPSIS
array([ 10.8399031...,  9.6840375...,  6.2164159...])
```

Ancillary Objects

colour.colorimetry

<code>sd_to_XYZ_tristimulus_weighting_factors_ASTME30815(sd, cmfs, illuminant)</code>	Converts given spectral distribution to <i>CIE XYZ</i> tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise <i>ASTM E308-15</i> method.
<code>adjust_tristimulus_weighting_factors_ASTME30815(weights, start, stop, step)</code>	Adjusts given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise <i>ASTM E308-15</i> method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.
<code>lagrange_coefficients_ASTME202211([...])</code>	Computes the <i>Lagrange Coefficients</i> for given interval size using practise <i>ASTM E2022-11</i> method.
<code>tristimulus_weighting_factors_ASTME202211(...)</code>	Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise <i>ASTM E2022-11</i> method.

colour.colorimetry.sd_to_XYZ_tristimulus_weighting_factors_ASTME30815

colour.colorimetry.sd_to_XYZ_tristimulus_weighting_factors_ASTME30815(*sd*, *cmfs*=XYZ_ColourMatchingFunctions(name='1931 2 Degree Standard Observer', ...), *illuminant*=SpectralDistribution(name='1 Constant', ...), *k*=None)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise *ASTM E308-15* method.

Parameters

- **sd** (*SpectralDistribution*) – Spectral distribution.
- **cmfs** (*XYZ_ColourMatchingFunctions*) – Standard observer colour matching functions.
- **illuminant** (*SpectralDistribution*, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant *k*. For reflecting or transmitting object colours, *k* is chosen so that $Y = 100$ for objects for which the spectral reflectance

factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns CIE XYZ tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational15]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_tristimulus_weighting_factors_ASTME30815(
...     sd, cmfs, illuminant) # doctest: +ELLIPSIS
array([ 10.8402899...,  9.6843539...,  6.2160858...])
```

colour.colorimetry.adjust_tristimulus_weighting_factors_ASTME30815

colour.colorimetry.**adjust_tristimulus_weighting_factors_ASTME30815**(*W*, *shape_r*, *shape_t*)

Adjusts given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise *ASTM E308-15* method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.

Parameters

- **W** (array_like) – Tristimulus weighting factors table.
- **shape_r** ([SpectralShape](#)) – Reference spectral shape.
- **shape_t** ([SpectralShape](#)) – Test spectral shape.

Returns Adjusted tristimulus weighting factors.

Return type ndarray

References

[[ASTMInternational15](#)]

Examples

```
>>> from colour import (CMFS, sd_CIE_standard_illuminant_A,
...     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> W = tristimulus_weighting_factors_ASTME202211(
...     cmfs, A, SpectralShape(360, 830, 20))
>>> with numpy_print_options(suppress=True):
...     adjust_tristimulus_weighting_factors_ASTME30815(
...         W, SpectralShape(360, 830, 20), SpectralShape(400, 700, 20))
... # doctest: +ELLIPSIS
array([[ 0.0509543...,  0.0040971...,  0.2144280...],
       [ 0.7734225...,  0.0779839...,  3.6965732...],
       [ 1.9000905...,  0.3037005...,  9.7554195...],
       [ 1.9707727...,  0.8552809..., 11.4867325...],
       [ 0.7183623...,  2.1457000...,  6.7845806...],
       [ 0.0426667...,  4.8985328...,  2.3208000...],
       [ 1.5223302...,  9.6471138...,  0.7430671...],
       [ 5.6770329..., 14.4609708...,  0.1958194...],
       [12.4451744..., 17.4742541...,  0.0051827...],
       [20.5535772..., 17.5838219..., -0.0026512...],
       [25.3315384..., 14.8957035...,  0.         ...],
       [21.5711570..., 10.0796619...,  0.         ...],
       [12.1785817...,  5.0680655...,  0.         ...],
       [ 4.6675746...,  1.8303239...,  0.         ...],
       [ 1.3236117...,  0.5129694...,  0.         ...],
       [ 0.4171109...,  0.1618194...,  0.         ...]])
```

colour.colorimetry.lagrange_coefficients_ASTME202211

`colour.colorimetry.lagrange_coefficients_ASTME202211(interval=10, interval_type='inner')`
Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.

Parameters

- **interval** (`int`) – Interval size in nm.
- **interval_type** (`unicode`, optional) – {'inner', 'boundary'}, If the interval is an *inner* interval *Lagrange Coefficients* are computed for degree 4. Degree 3 is used for a *boundary* interval.

Returns *Lagrange Coefficients*.

Return type `ndarray`

References

[ASTMInternational11]

Examples

```
>>> lagrange_coefficients_ASTME202211(10, 'inner')
... # doctest: +ELLIPSIS
array([[ -0.028...,  0.940...,  0.104..., -0.016...],
       [ -0.048...,  0.864...,  0.216..., -0.032...],
       [ -0.059...,  0.773...,  0.331..., -0.045...],
       [ -0.064...,  0.672...,  0.448..., -0.056...],
       [ -0.062...,  0.562...,  0.562..., -0.062...],
       [ -0.056...,  0.448...,  0.672..., -0.064...],
       [ -0.045...,  0.331...,  0.773..., -0.059...],
       [ -0.032...,  0.216...,  0.864..., -0.048...],
       [ -0.016...,  0.104...,  0.940..., -0.028...]])
>>> lagrange_coefficients_ASTME202211(10, 'boundary')
... # doctest: +ELLIPSIS
array([[ 0.85...,  0.19..., -0.04...],
       [ 0.72...,  0.36..., -0.08...],
       [ 0.59...,  0.51..., -0.10...],
       [ 0.48...,  0.64..., -0.12...],
       [ 0.37...,  0.75..., -0.12...],
       [ 0.28...,  0.84..., -0.12...],
       [ 0.19...,  0.91..., -0.10...],
       [ 0.12...,  0.96..., -0.08...],
       [ 0.05...,  0.99..., -0.04...]])
```

colour.colorimetry.tristimulus_weighting_factors_ASTME202211

`colour.colorimetry.tristimulus_weighting_factors_ASTME202211(cmfs, illuminant, shape, k=None)`

Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

The computed table of tristimulus weighting factors should be used with spectral data that has been corrected for spectral bandpass dependence.

Parameters

- **cmfs** ([XYZ_ColourMatchingFunctions](#)) – Standard observer colour matching functions.
- **illuminant** ([SpectralDistribution](#)) – Illuminant spectral distribution.
- **shape** ([SpectralShape](#)) – Shape used to build the table, only the interval is needed.
- **k** (numeric, optional) – Normalisation constant k . For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns Tristimulus weighting factors table.

Return type ndarray

Raises [ValueError](#) – If the colour matching functions or illuminant intervals are not equal to 1 nm.

Warning:

- The tables of tristimulus weighting factors are cached in `colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE` attribute. Their identifier key is defined by the colour matching functions and illuminant names along the current shape such as: *CIE 1964 10 Degree Standard Observer, A, (360.0, 830.0, 10.0)* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.

Notes

- Input colour matching functions and illuminant intervals are expected to be equal to 1 nm. If the illuminant data is not available at 1 nm interval, it needs to be interpolated using *CIE* recommendations: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable.

References

[[ASTMInternational11](#)]

Examples

```
>>> from colour import (CMFS, sd_CIE_standard_illuminant_A,
...                     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
```

(continues on next page)

(continued from previous page)

```

>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> with numpy_print_options(suppress=True):
...     tristimulus_weighting_factors_ASTME202211(
...         cmfs, A, SpectralShape(360, 830, 20))
... # doctest: +ELLIPSIS
array([[ -0.0002981..., -0.0000317..., -0.0013301...],
       [ -0.0087155..., -0.0008915..., -0.0407436...],
       [  0.0599679...,  0.0050203...,  0.2565018...],
       [  0.7734225...,  0.0779839...,  3.6965732...],
       [  1.9000905...,  0.3037005...,  9.7554195...],
       [  1.9707727...,  0.8552809..., 11.4867325...],
       [  0.7183623...,  2.1457000...,  6.7845806...],
       [  0.0426667...,  4.8985328...,  2.3208000...],
       [  1.5223302...,  9.6471138...,  0.7430671...],
       [  5.6770329..., 14.4609708...,  0.1958194...],
       [ 12.4451744..., 17.4742541...,  0.0051827...],
       [ 20.5535772..., 17.5838219..., -0.0026512...],
       [ 25.3315384..., 14.8957035...,  0.      ...],
       [ 21.5711570..., 10.0796619...,  0.      ...],
       [ 12.1785817...,  5.0680655...,  0.      ...],
       [  4.6675746...,  1.8303239...,  0.      ...],
       [  1.3236117...,  0.5129694...,  0.      ...],
       [  0.3175325...,  0.1230084...,  0.      ...],
       [  0.0746341...,  0.0290243...,  0.      ...],
       [  0.0182990...,  0.0071606...,  0.      ...],
       [  0.0047942...,  0.0018888...,  0.      ...],
       [  0.0013293...,  0.0005277...,  0.      ...],
       [  0.0004254...,  0.0001704...,  0.      ...],
       [  0.0000962...,  0.0000389...,  0.      ...]])

```

Integration

colour.colorimetry

<code>sd_to_XYZ_integration(sd[, cmfs, illuminant, k])</code>	Converts given spectral distribution to <i>CIE XYZ</i> tristimulus values using given colour matching functions and illuminant according to classical integration method.
<code>multi_sds_to_XYZ_integration(msd, shape[, ...])</code>	Converts given multi-spectral distribution array <i>msd</i> with given spectral shape to <i>CIE XYZ</i> tristimulus values using given colour matching functions and illuminant.

colour.colorimetry.sd_to_XYZ_integration

colour.colorimetry.**sd_to_XYZ_integration**(sd, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralDistribution(name='1 Constant', ...), k=None)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.

Parameters

- **sd** (`SpectralDistribution`) – Spectral distribution.
- **cmfs** (`XYZ_ColourMatchingFunctions`) – Standard observer colour matching functions.
- **illuminant** (`SpectralDistribution`, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k . For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k , must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns CIE XYZ tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[WS00f]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
```

(continues on next page)

(continued from previous page)

```
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_integration(sd, cmfs, illuminant)
... # doctest: +ELLIPSIS
array([ 10.8401846...,  9.6837311...,  6.2120912...])
```

colour.colorimetry.multi_sds_to_XYZ_integration

`colour.colorimetry.multi_sds_to_XYZ_integration(msd, shape, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralDistribution(name='1 Constant', ...), k=None)`

Converts given multi-spectral distribution array *msd* with given spectral shape to CIE XYZ tristimulus values using given colour matching functions and illuminant.

Parameters

- **msa** (array_like) – Multi-spectral distribution array *msd*, the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, *msd* shape should be (384, 512, 77).
- **shape** ([SpectralShape](#), optional) – Spectral shape of the multi-spectral distribution array *msd*, *cmfs* and *illuminant* will be aligned with it.
- **cmfs** ([XYZ_ColourMatchingFunctions](#)) – Standard observer colour matching functions.
- **illuminant** ([SpectralDistribution](#), optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant *k*. For reflecting or transmitting object colours, *k* is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants *k* is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the *Y* value is required to be numerically equal to the absolute value of a photometric quantity, the constant, *k*, must be put equal to the numerical value of K_m , the maximum spectral luminous efficacy (which is equal to $683 \text{ lm} \cdot \text{W}^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns CIE XYZ tristimulus values, for a 512x384 multi-spectral image with 77 bins, the output shape will be (384, 512, 3).

Return type array_like

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[WS00f]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> msd = np.array([
...     [
...         [0.0137, 0.0913, 0.0152, 0.0281, 0.1918, 0.0430],
...         [0.0159, 0.3145, 0.0842, 0.0907, 0.7103, 0.0437],
...         [0.0096, 0.2582, 0.4139, 0.2228, 0.0041, 0.3744],
...         [0.0111, 0.0709, 0.0220, 0.1249, 0.1817, 0.0020],
...         [0.0179, 0.2971, 0.5630, 0.2375, 0.0024, 0.5819],
...         [0.1057, 0.4620, 0.1918, 0.5625, 0.4209, 0.0027],
...     ],
...     [
...         [0.0433, 0.2683, 0.2373, 0.0518, 0.0118, 0.0823],
...         [0.0258, 0.0831, 0.0430, 0.3230, 0.2302, 0.0081],
...         [0.0248, 0.1203, 0.0054, 0.0065, 0.1860, 0.3625],
...         [0.0186, 0.1292, 0.0079, 0.4006, 0.9404, 0.3213],
...         [0.0310, 0.1682, 0.3719, 0.0861, 0.0041, 0.7849],
...         [0.0473, 0.3221, 0.2268, 0.3161, 0.1124, 0.0024],
...     ],
... ])
>>> D65 = ILLUMINANTS_SDS['D65']
>>> multi_sds_to_XYZ(
...     msd, SpectralShape(400, 700, 60), illuminant=D65)
... # doctest: +ELLIPSIS
array([[ 7.1958378...,  3.8605390..., 10.1016398...],
       [25.5738615..., 14.7200581..., 34.8440007...],
       [17.5854414..., 28.5668344..., 30.1806687...],
       [11.3271912...,  8.4598177...,  7.9015758...],
       [19.6581831..., 35.5918480..., 35.1430220...],
       [45.8212491..., 39.2600939..., 51.7907710...]],
      <BLANKLINE>
       [[ 8.8287837..., 13.3870357..., 30.5702050...],
       [22.3324362..., 18.9560919...,  9.3952305...],
       [ 6.6887212...,  2.5728891..., 13.2618778...],
       [41.8166227..., 27.1191979..., 14.2627944...],
       [ 9.2414098..., 20.2056200..., 20.1992502...],
       [24.7830551..., 26.2221584..., 36.4430633...]])
```

Spectral Bandpass Dependence Correction

colour

<code>bandpass_correction(sd[, method])</code>	Implements spectral bandpass dependence correction on given spectral distribution using given method.
<code>BANDPASS_CORRECTION_METHODS</code>	Supported spectral bandpass dependence correction methods.

colour.bandpass_correction

colour.**bandpass_correction**(sd, method='Stearns 1988')

Implements spectral bandpass dependence correction on given spectral distribution using given

method.

Parameters

- **sd** (*SpectralDistribution*) – Spectral distribution.
- **method** (unicode, optional) – ('Stearns 1988',) Correction method.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type *SpectralDistribution*

colour.BANDPASS_CORRECTION_METHODS

colour.BANDPASS_CORRECTION_METHODS = CaseInsensitiveMapping({'Stearns 1988': ...})

Supported spectral bandpass dependence correction methods.

BANDPASS_CORRECTION_METHODS [CaseInsensitiveMapping] {'Stearns 1988', }

Stearns and Stearns (1988)

colour.colorimetry

bandpass_correction_Stearns1988(sd)

Implements spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

colour.colorimetry.bandpass_correction_Stearns1988

colour.colorimetry.bandpass_correction_Stearns1988(sd)

Implements spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

Parameters **sd** (*SpectralDistribution*) – Spectral distribution.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type *SpectralDistribution*

References

[SS88], [WRC12c]

Examples

```
>>> from colour import SpectralDistribution
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
```

(continues on next page)

(continued from previous page)

```

...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction_Stearns1988(
...         SpectralDistribution(data))
... # doctest: +ELLIPSIS
SpectralDistribution([[ 500.      ,  0.0646518...],
                     [ 520.      ,  0.0704293...],
                     [ 540.      ,  0.0769485...],
                     [ 560.      ,  0.0856928...],
                     [ 580.      ,  0.1129644...],
                     [ 600.      ,  0.1379256...]],
                    interpolator=SpragueInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={...})

```

Colour Matching Functions

colour.colorimetry

<code>LMS_ConeFundamentals([data, domain, labels])</code>	Implements support for the Stockman and Sharpe <i>LMS</i> cone fundamentals colour matching functions.
<code>RGB_ColourMatchingFunctions([data, domain, ...])</code>	Implements support for the <i>CIE RGB</i> colour matching functions.
<code>XYZ_ColourMatchingFunctions([data, domain, ...])</code>	Implements support for the <i>CIE</i> Standard Observers <i>XYZ</i> colour matching functions.

colour.colorimetry.LMS_ConeFundamentals

class colour.colorimetry.LMS_ConeFundamentals(*data=None, domain=None, labels=None, **kwargs*)

Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (array_like, optional) – class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the Values to initialise the multiple colour.SpectralDistribution colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (unicode, optional) – Multi-spectral distribution name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.

- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distribution labels for figures, default to `colour.colorimetry.LMS_ConeFundamentals.labels` attribute value.

__init__(*data=None, domain=None, labels=None, **kwargs*)
Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__([data, domain, labels])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>normalise([factor])</code>	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas DataFrame</i> class instance.
<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.

Continued on next page

Table 103 – continued from previous page

trim_wavelengths(shape)	
zeros()	
Attributes	
data	
domain	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain x variable.
dtype	Getter and setter property for the continuous signal dtype.
extrapolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
extrapolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
function	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
interpolator	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
interpolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
items	
labels	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
mapping	
name	Getter and setter property for the abstract continuous function name.
range	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
shape	Getter and setter property for the multi-spectral distribution shape.
signal_type	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
signals	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
strict_labels	Getter and setter property for the multi-spectral distribution strict labels.
strict_name	Getter and setter property for the multi-spectral distribution strict name.
title	
values	Getter and setter property for the multi-spectral distribution values.
wavelengths	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
x	

Continued on next page

Table 104 – continued from previous page

y
z

colour.colorimetry.RGB_ColourMatchingFunctions

class colour.colorimetry.RGB_ColourMatchingFunctions(*data=None, domain=None, labels=None, **kwargs*)

Implements support for the CIE RGB colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (unicode, optional) – Multi-spectral distribution name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distribution labels for figures, default to colour.colorimetry.RGB_ColourMatchingFunctions.labels attribute value.

__init__(*data=None, domain=None, labels=None, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ ([<i>data, domain, labels</i>])	Initialize self.
align (<i>shape[, interpolator, ...]</i>)	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
arithmetical_operation (<i>a, operation[, in_place]</i>)	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
clone ()	

Continued on next page

Table 105 – continued from previous page

<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain x closest element.
<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain x variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>normalise([factor])</code>	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas</i> DataFrame class instance.
<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>	
<code>domain</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain x variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
<code>interpolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.

Continued on next page

Table 106 – continued from previous page

interpolator_args	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
items	
labels	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
mapping	
name	Getter and setter property for the abstract continuous function name.
range	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
shape	Getter and setter property for the multi-spectral distribution shape.
signal_type	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
signals	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
strict_labels	Getter and setter property for the multi-spectral distribution strict labels.
strict_name	Getter and setter property for the multi-spectral distribution strict name.
title	
values	Getter and setter property for the multi-spectral distribution values.
wavelengths	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
x	
y	
z	

`colour.colorimetry.XYZ_ColourMatchingFunctions`

class `colour.colorimetry.XYZ_ColourMatchingFunctions`(*data=None, domain=None, labels=None, **kwargs*)

Implements support for the CIE Standard Observers XYZ colour matching functions.

Parameters

- **data** (Series or Dataframe or `Signal` or `MultiSignal` or `MultiSpectralDistribution` or array_like or dict_like, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (array_like, optional) – Values to initialise the multiple `colour.SpectralDistribution` class instances `colour.continuous.Signal.wavelengths` attribute with. If both data and domain arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.wavelengths` attribute.
- **labels** (array_like, optional) – Names to use for the `colour.SpectralDistribution` class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distribution name.

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distribution labels for figures, default to `colour.colorimetry.XYZ_ColourMatchingFunctions.labels` attribute value.

`__init__`(*data=None, domain=None, labels=None, **kwargs*)
Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__</code> (<i>[data, domain, labels]</i>)	Initialize self.
<code>align</code> (<i>shape[, interpolator, ...]</i>)	Aligns the multi-spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation</code> (<i>a, operation[, in_place]</i>)	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone</code> ()	
<code>copy</code> ()	Returns a copy of the sub-class instance.
<code>domain_distance</code> (<i>a</i>)	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>extrapolate</code> (<i>shape[, extrapolator, ...]</i>)	Extrapolates the multi-spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan</code> (<i>[method, default]</i>)	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>get</code> ()	
<code>interpolate</code> (<i>shape[, interpolator, ...]</i>)	Interpolates the multi-spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform</code> ()	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data</code> (<i>[data, domain, ...]</i>)	Unpack given data for multi-continuous signal instantiation.
<code>normalise</code> (<i>[factor]</i>)	Normalises the multi-spectral distribution with given normalization factor.
<code>to_dataframe</code> ()	Converts the continuous signal to a <i>Pandas DataFrame</i> class instance.

Continued on next page

Table 107 – continued from previous page

<code>to_sds()</code>	Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.
<code>trim(shape)</code>	Trims the multi-spectral distribution wavelengths to given shape.
<code>trim_wavelengths(shape)</code>	
<code>zeros()</code>	

Attributes

<code>data</code>	
<code>domain</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain x variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
<code>interpolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
<code>interpolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
<code>items</code>	
<code>labels</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
<code>mapping</code>	
<code>name</code>	Getter and setter property for the abstract continuous function name.
<code>range</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
<code>shape</code>	Getter and setter property for the multi-spectral distribution shape.
<code>signal_type</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
<code>signals</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.
<code>strict_labels</code>	Getter and setter property for the multi-spectral distribution strict labels.
<code>strict_name</code>	Getter and setter property for the multi-spectral distribution strict name.
<code>title</code>	

Continued on next page

Table 108 – continued from previous page

values	Getter and setter property for the multi-spectral distribution values.
wavelengths	Getter and setter property for the multi-spectral distribution wavelengths λ_n .
x	
y	
z	

Dataset

colour

CMFS	Aggregated colour matching functions.
LMS_CMFS	<i>LMS</i> colour matching functions.
RGB_CMFS	<i>CIE RGB</i> colour matching functions.
STANDARD_OBSERVERS_CMFS	<i>CIE</i> Standard Observers <i>XYZ</i> colour matching functions.

colour.CMFS

```
colour.CMFS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 2 Degree Cone Fundamentals': ...},
    Aggregated colour matching functions.
```

References

[Bro09], [CVRd], [CVRe], [CVRf], [SS00], [CVRg], [Mac10]

CMFS [CaseInsensitiveMapping] {'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs', 'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}

colour.LMS_CMFS

```
colour.LMS_CMFS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 2 Degree Cone Fundamentals': ...},
    LMS colour matching functions.
```

References

[SS00], [Mac10]

LMS_CMFS [CaseInsensitiveMapping] {'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Smith & Pokorny 1975 Normal Trichromats'}

colour.RGB_CMFS

```
colour.RGB_CMFS = CaseInsensitiveMapping({'Wright & Guild 1931 2 Degree RGB CMFs': ..., 'Stiles & Burch 1955 2 Degree RGB CMFs': ..., 'Stiles & Burch 1959 10 Degree RGB CMFs': ...},
    CIE RGB colour matching functions.
```

References

[Bro09], [CVRf], [CVRg]

RGB_CMFS [CaseInsensitiveMapping] {'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs'}

colour.STANDARD_OBSERVERS_CMFS

`colour.STANDARD_OBSERVERS_CMFS` = `CaseInsensitiveMapping`({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1931 10 Degree Standard Observer': ...})
CIE Standard Observers XYZ colour matching functions.

References

[CVRd], [CVRe]

STANDARD_OBSERVERS_CMFS [CaseInsensitiveMapping] {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

Colour Matching Functions Transformations

Ancillary Objects

`colour.colorimetry`

<code>RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(...)</code>	Converts <i>Wright & Guild 1931 2 Degree RGB CMFs</i> colour matching functions into the <i>CIE 1931 2 Degree Standard Observer</i> colour matching functions.
<code>RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(...)</code>	Converts <i>Stiles & Burch 1959 10 Degree RGB CMFs</i> colour matching functions into the <i>CIE 1964 10 Degree Standard Observer</i> colour matching functions.
<code>RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(...)</code>	Converts <i>Stiles & Burch 1959 10 Degree RGB CMFs</i> colour matching functions into the <i>Stockman & Sharpe 10 Degree Cone Fundamentals</i> spectral sensitivity functions.
<code>LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(...)</code>	Converts <i>Stockman & Sharpe 2 Degree Cone Fundamentals</i> colour matching functions into the <i>CIE 2012 2 Degree Standard Observer</i> colour matching functions.
<code>LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(...)</code>	Converts <i>Stockman & Sharpe 10 Degree Cone Fundamentals</i> colour matching functions into the <i>CIE 2012 10 Degree Standard Observer</i> colour matching functions.

colour.colorimetry.RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs

`colour.colorimetry.RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(wavelength)`

Converts *Wright & Guild 1931 2 Degree RGB CMFs* colour matching functions into the *CIE 1931 2 Degree Standard Observer* colour matching functions.

Parameters `wavelength` (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 1931 2 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 1931 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00i]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(700) # doctest: +ELLIPSIS
array([ 0.0113577...,  0.004102 ,  0.          ])
```

colour.colorimetry.RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs

`colour.colorimetry.RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(wavelength)`

Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *CIE 1964 10 Degree Standard Observer* colour matching functions.

Parameters `wavelength` (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 1964 10 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 1964 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00m]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(700) # doctest: +ELLIPSIS
array([ 0.0096432...,  0.0037526..., -0.0000041...])
```

colour.colorimetry.RGB_10_degree_cmfs_to_LMS_10_degree_cmfs

colour.colorimetry.RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(*wavelength*)

Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral sensitivity functions.

Parameters *wavelength* (numeric or array_like) – Wavelength λ in nm.

Returns *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *Stockman & Sharpe 10 Degree Cone Fundamentals* already exists, this definition is intended for educational purpose.

References

[CIET13606]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(700) # doctest: +ELLIPSIS
array([ 0.0052860...,  0.0003252...,  0.          ])
```

colour.colorimetry.LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs

colour.colorimetry.LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(*wavelength*)

Converts *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 2 Degree Standard Observer* colour matching functions.

Parameters *wavelength* (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 2012 2 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRb]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(700) # doctest: +ELLIPSIS
array([ 0.0109677...,  0.0041959...,  0.          ])
```

colour.colorimetry.LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs

colour.colorimetry.LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(*wavelength*)

Converts *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 10 Degree Standard Observer* colour matching functions.

Parameters *wavelength* (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 2012 10 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRa]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(700) # doctest: +ELLIPSIS
array([ 0.0098162...,  0.0037761...,  0.          ])
```

Illuminants and Light Sources

Dataset

colour

ILLUMINANTS	Aggregated <i>CIE</i> illuminants chromaticity coordinates.
ILLUMINANTS_SDS	<i>CIE</i> illuminants spectral distributions.
HUNTERLAB_ILLUMINANTS	Aggregated <i>Hunter L,a,b</i> illuminant dataset.

Continued on next page

Table 111 – continued from previous page

<code>LIGHT_SOURCES</code>	Aggregated light sources chromaticity coordinates.
<code>LIGHT_SOURCES_SDS</code>	Aggregated light sources spectral distributions.

colour.ILLUMINANTS

`colour.ILLUMINANTS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ...})`
 Aggregated CIE illuminants chromaticity coordinates.

Notes

CIE Illuminant D Series D60 illuminant chromaticity coordinates were computed as follows:

```
CCT = 6000 * 1.4388 / 1.438
xy = colour.temperature.CCT_to_xy_CIE_D(CCT)

sd = colour.sd_CIE_illuminant_D_series(xy)
sd.interpolator = colour.LinearInterpolator
colour.XYZ_to_xy(
    colour.sd_to_XYZ(
        sd, colour.CMFS['CIE 1964 10 Degree Standard Observer']) / 100.0)
```

CIE Illuminant D Series D50 illuminant and *CIE Standard Illuminant D Series D65* chromaticity coordinates are rounded to 4 decimals as given in the typical RGB colourspace literature. Their chromaticity coordinates as given in [CIET14804d] are (0.34567, 0.35851) and (0.31272, 0.32903) respectively.

References

[CIET14804d], [DigitalCIInitiatives07], [TheAoMPAA Sciences ScienceaTCouncilAcademyCESACESPSubcommittee14c], [Wik06b]

ILLUMINANTS [CaseInsensitiveMapping] {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.ILLUMINANTS_SDS

`colour.ILLUMINANTS_SDS = CaseInsensitiveMapping({'A': ..., 'B': ..., 'C': ..., 'D50': ..., 'D55': ..., 'D60': ...})`
 CIE illuminants spectral distributions.

Notes

CIE Illuminant D Series D60 illuminant spectral distribution was computed as follows:

```
CCT = 6000 * 1.4388 / 1.438
xy = colour.temperature.CCT_to_xy_CIE_D(CCT)

sd = colour.sd_CIE_illuminant_D_series(xy)
```

References

[CIE04], [CIE]

ILLUMINANTS_SDS : CaseInsensitiveMapping

colour.HUNTERLAB_ILLUMINANTS

`colour.HUNTERLAB_ILLUMINANTS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ...})`
Aggregated *Hunter L,a,b* illuminant dataset.

References

[Hun08a], [Hun08b]

HUNTERLAB_ILLUMINANTS [CaseInsensitiveMapping] {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES

`colour.LIGHT_SOURCES = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ...})`
Aggregated light sources chromaticity coordinates.

LIGHT_SOURCES [CaseInsensitiveMapping] {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES_SDS

`colour.LIGHT_SOURCES_SDS = CaseInsensitiveMapping({'Natural': ..., 'Philips TL-84': ..., 'SA': ..., 'SC': ...})`
Aggregated light sources spectral distributions.

LIGHT_SOURCES_SDS : CaseInsensitiveMapping

Dominant Wavelength and Purity

colour

<code>dominant_wavelength(xy, xy_n[, cmfs, reverse])</code>	Returns the <i>dominant wavelength</i> λ_d for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.
---	--

Continued on next page

Table 112 – continued from previous page

<code>complementary_wavelength(xy, xy_n[, cmfs])</code>	Returns the <i>complementary wavelength</i> λ_c for given colour stimulus xy and the related xy_{wl} first and xy_{cw} second intersection coordinates with the spectral locus.
<code>excitation_purity(xy, xy_n[, cmfs])</code>	Returns the <i>excitation purity</i> P_e for given colour stimulus xy .
<code>colorimetric_purity(xy, xy_n[, cmfs])</code>	Returns the <i>colorimetric purity</i> P_c for given colour stimulus xy .

colour.dominant_wavelength

`colour.dominant_wavelength(xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), reverse=False)`

Returns the *dominant wavelength* λ_d for given colour stimulus xy and the related xy_{wl} first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_{wl} first intersection coordinates are on the line of purples, the *complementary wavelength* will be computed in lieu.

The *complementary wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_{wl} first intersection coordinates will be set to the *complementary dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (array_like) – Colour stimulus xy chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus xy chromaticity coordinates.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.
- **reverse** (bool, optional) – Reverse the computation direction to retrieve the *complementary wavelength*.

Returns *Dominant wavelength*, first intersection point xy chromaticity coordinates, second intersection point xy chromaticity coordinates.

Return type `tuple`

References

[CIET14804b], [Erdb]

Examples

Dominant wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(dominant_wavelength(xy, xy_n, cmfs)) # doctest: +ELLIPSIS
(array(616...),
```

(continues on next page)

(continued from previous page)

```
array([ 0.6835474...,  0.3162840...]),
array([ 0.6835474...,  0.3162840...]))
```

Complementary dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.37605506, 0.24452225])
>>> pprint(dominant_wavelength(xy, xy_n, cmfs)) # doctest: +ELLIPSIS
(array(-509.0),
 array([ 0.4572314...,  0.1362814...]),
 array([ 0.0104096...,  0.7320745...]))
```

colour.complementary_wavelength

`colour.complementary_wavelength(xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...))`

Returns the *complementary wavelength* λ_c for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_wl first intersection coordinates are on the line of purples, the *dominant wavelength* will be computed in lieu.

The *dominant wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_wl first intersection coordinates will be set to the *dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (array_like) – Colour stimulus xy chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus xy chromaticity coordinates.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.

Returns *Complementary wavelength*, first intersection point xy chromaticity coordinates, second intersection point xy chromaticity coordinates.

Return type `tuple`

References

[CIET14804b], [Erdbe]

Examples

Complementary wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.37605506, 0.24452225])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(complementary_wavelength(xy, xy_n, cmfs)) # doctest: +ELLIPSIS
(array(509.0),
 array([ 0.0104096...,  0.7320745...]),
 array([ 0.0104096...,  0.7320745...]))
```

Dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> pprint(complementary_wavelength(xy, xy_n, cmfs)) # doctest: +ELLIPSIS
(array(492.0),
 array([ 0.0364795 ,  0.3384712...]),
 array([ 0.0364795 ,  0.3384712...]))
```

colour.excitation_purity

`colour.excitation_purity(xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...))`

Returns the *excitation purity* P_e for given colour stimulus xy .

Parameters

- **xy** (array_like) – Colour stimulus xy chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus xy chromaticity coordinates.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.

Returns *Excitation purity* P_e .

Return type numeric or array_like

References

[CIET14804b], [Erdb]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> excitation_purity(xy, xy_n, cmfs) # doctest: +ELLIPSIS
0.6228856...
```

colour.colorimetric_purity

`colour.colorimetric_purity(xy, xy_n, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...))`

Returns the *colorimetric purity* P_c for given colour stimulus xy .

Parameters

- **xy** (array_like) – Colour stimulus xy chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus xy chromaticity coordinates.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.

Returns *Colorimetric purity* P_c .

Return type numeric or array_like

References

[CIET14804b], [Erdb]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> colorimetric_purity(xy, xy_n, cmfs) # doctest: +ELLIPSIS
0.6135828...
```

Luminous Efficiency Functions

colour

<code>luminous_efficacy(sd[, lef])</code>	Returns the <i>luminous efficacy</i> in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.
<code>luminous_efficiency(sd[, lef])</code>	Returns the <i>luminous efficiency</i> of given spectral distribution using given luminous efficiency function.
<code>luminous_flux(sd[, lef, K_m])</code>	Returns the <i>luminous flux</i> for given spectral distribution using given luminous efficiency function.
<code>sd_mesopic_luminous_efficiency_function(Lp)</code>	Returns the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p .

colour.luminous_efficacy

`colour.luminous_efficacy(sd, lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...))`

Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (*SpectralDistribution*) – test spectral distribution
- **lef** (*SpectralDistribution*, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficacy in $lm \cdot W^{-1}$.

Return type numeric

References

[Wik05d]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_efficiency(sd) # doctest: +ELLIPSIS
136.2170803...
```

colour.luminous_efficiency

`colour.luminous_efficiency(sd, lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...))`

Returns the *luminous efficiency* of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (`SpectralDistribution`) – test spectral distribution
- **lef** (`SpectralDistribution`, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficiency.

Return type numeric

References

[Wik03c]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_efficiency(sd) # doctest: +ELLIPSIS
0.1994393...
```

colour.luminous_flux

`colour.luminous_flux(sd, lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...), K_m=683.0)`

Returns the *luminous flux* for given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (`SpectralDistribution`) – test spectral distribution
- **lef** (`SpectralDistribution`, optional) – $V(\lambda)$ luminous efficiency function.
- **K_m** (numeric, optional) – $lm \cdot W^{-1}$ maximum photopic luminous efficiency

Returns Luminous flux.

Return type numeric

References

[Wik03c]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_flux(sd) # doctest: +ELLIPSIS
23807.6555273...
```

colour.sd_mesopic_luminous_efficiency_function

```
colour.sd_mesopic_luminous_efficiency_function(Lp, source='Blue Heavy', method='MOVE',
        photopic_lef=SpectralDistribution(name='CIE
        1924 Photopic Standard Observer', ...), sco-
        topic_lef=SpectralDistribution(name='CIE 1951
        Scotopic Standard Observer', ...))
```

Returns the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p .

Parameters

- **Lp** (numeric) – Photopic luminance L_p .
- **source** (unicode, optional) – {'Blue Heavy', 'Red Heavy'}, Light source colour temperature.
- **method** (unicode, optional) – {'MOVE', 'LRC'}, Method to calculate the weighting factor.
- **photopic_lef** (*SpectralDistribution*, optional) – $V(\lambda)$ photopic luminous efficiency function.
- **scotopic_lef** (*SpectralDistribution*, optional) – $V'(\lambda)$ scotopic luminous efficiency function.

Returns Mesopic luminous efficiency function $V_m(\lambda)$.

Return type *SpectralDistribution*

References

[Wik05e]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_mesopic_luminous_efficiency_function(0.2) # doctest: +ELLIPSIS
SpectralDistribution([[ 380.      ,  0.000424 ...],
                    [ 381.      ,  0.0004781...],
                    [ 382.      ,  0.0005399...],
                    [ 383.      ,  0.0006122...],
                    [ 384.      ,  0.0006961...],
                    [ 385.      ,  0.0007929...],
                    [ 386.      ,  0.000907 ...],
                    [ 387.      ,  0.0010389...],
                    [ 388.      ,  0.0011923...],
                    [ 389.      ,  0.0013703...],
```

(continues on next page)

(continued from previous page)

[390.	,	0.0015771...],
[391.	,	0.0018167...],
[392.	,	0.0020942...],
[393.	,	0.0024160...],
[394.	,	0.0027888...],
[395.	,	0.0032196...],
[396.	,	0.0037222...],
[397.	,	0.0042957...],
[398.	,	0.0049531...],
[399.	,	0.0057143...],
[400.	,	0.0065784...],
[401.	,	0.0075658...],
[402.	,	0.0086912...],
[403.	,	0.0099638...],
[404.	,	0.0114058...],
[405.	,	0.0130401...],
[406.	,	0.0148750...],
[407.	,	0.0169310...],
[408.	,	0.0192211...],
[409.	,	0.0217511...],
[410.	,	0.0245342...],
[411.	,	0.0275773...],
[412.	,	0.0309172...],
[413.	,	0.0345149...],
[414.	,	0.0383998...],
[415.	,	0.0425744...],
[416.	,	0.0471074...],
[417.	,	0.0519322...],
[418.	,	0.0570541...],
[419.	,	0.0625466...],
[420.	,	0.0683463...],
[421.	,	0.0745255...],
[422.	,	0.0809440...],
[423.	,	0.0877344...],
[424.	,	0.0948915...],
[425.	,	0.1022731...],
[426.	,	0.109877 ...],
[427.	,	0.1178421...],
[428.	,	0.1260316...],
[429.	,	0.1343772...],
[430.	,	0.143017 ...],
[431.	,	0.1518128...],
[432.	,	0.1608328...],
[433.	,	0.1700088...],
[434.	,	0.1792726...],
[435.	,	0.1886934...],
[436.	,	0.1982041...],
[437.	,	0.2078032...],
[438.	,	0.2174184...],
[439.	,	0.2271147...],
[440.	,	0.2368196...],
[441.	,	0.2464623...],
[442.	,	0.2561153...],
[443.	,	0.2657160...],
[444.	,	0.2753387...],
[445.	,	0.2848520...],

(continues on next page)

(continued from previous page)

[446.	,	0.2944648...],
[447.	,	0.3034902...],
[448.	,	0.3132347...],
[449.	,	0.3223257...],
[450.	,	0.3314513...],
[451.	,	0.3406129...],
[452.	,	0.3498117...],
[453.	,	0.3583617...],
[454.	,	0.3676377...],
[455.	,	0.3762670...],
[456.	,	0.3849392...],
[457.	,	0.3936540...],
[458.	,	0.4024077...],
[459.	,	0.4111965...],
[460.	,	0.4193298...],
[461.	,	0.4281803...],
[462.	,	0.4363804...],
[463.	,	0.4453117...],
[464.	,	0.4542949...],
[465.	,	0.4626509...],
[466.	,	0.4717570...],
[467.	,	0.4809300...],
[468.	,	0.4901776...],
[469.	,	0.4995075...],
[470.	,	0.5096145...],
[471.	,	0.5191293...],
[472.	,	0.5294259...],
[473.	,	0.5391316...],
[474.	,	0.5496217...],
[475.	,	0.5602103...],
[476.	,	0.5702197...],
[477.	,	0.5810207...],
[478.	,	0.5919093...],
[479.	,	0.6028683...],
[480.	,	0.6138806...],
[481.	,	0.6249373...],
[482.	,	0.6360619...],
[483.	,	0.6465989...],
[484.	,	0.6579538...],
[485.	,	0.6687841...],
[486.	,	0.6797939...],
[487.	,	0.6909887...],
[488.	,	0.7023827...],
[489.	,	0.7133032...],
[490.	,	0.7244513...],
[491.	,	0.7358470...],
[492.	,	0.7468118...],
[493.	,	0.7580294...],
[494.	,	0.7694964...],
[495.	,	0.7805225...],
[496.	,	0.7917805...],
[497.	,	0.8026123...],
[498.	,	0.8130793...],
[499.	,	0.8239297...],
[500.	,	0.8352251...],
[501.	,	0.8456342...],

(continues on next page)

(continued from previous page)

[502.	,	0.8564818...],
[503.	,	0.8676921...],
[504.	,	0.8785021...],
[505.	,	0.8881489...],
[506.	,	0.8986405...],
[507.	,	0.9079322...],
[508.	,	0.9174255...],
[509.	,	0.9257739...],
[510.	,	0.9350656...],
[511.	,	0.9432365...],
[512.	,	0.9509063...],
[513.	,	0.9586931...],
[514.	,	0.9658413...],
[515.	,	0.9722825...],
[516.	,	0.9779924...],
[517.	,	0.9836106...],
[518.	,	0.9883465...],
[519.	,	0.9920964...],
[520.	,	0.9954436...],
[521.	,	0.9976202...],
[522.	,	0.9993457...],
[523.	,	1. ...],
[524.	,	0.9996498...],
[525.	,	0.9990487...],
[526.	,	0.9975356...],
[527.	,	0.9957615...],
[528.	,	0.9930143...],
[529.	,	0.9899559...],
[530.	,	0.9858741...],
[531.	,	0.9814453...],
[532.	,	0.9766885...],
[533.	,	0.9709363...],
[534.	,	0.9648947...],
[535.	,	0.9585832...],
[536.	,	0.952012 ...],
[537.	,	0.9444916...],
[538.	,	0.9367089...],
[539.	,	0.9293506...],
[540.	,	0.9210429...],
[541.	,	0.9124772...],
[542.	,	0.9036604...],
[543.	,	0.8945958...],
[544.	,	0.8845999...],
[545.	,	0.8750500...],
[546.	,	0.8659457...],
[547.	,	0.8559224...],
[548.	,	0.8456846...],
[549.	,	0.8352499...],
[550.	,	0.8253229...],
[551.	,	0.8152079...],
[552.	,	0.8042205...],
[553.	,	0.7944209...],
[554.	,	0.7837466...],
[555.	,	0.7735680...],
[556.	,	0.7627808...],
[557.	,	0.7522710...],

(continues on next page)

(continued from previous page)

[558.	,	0.7417549...],
[559.	,	0.7312909...],
[560.	,	0.7207983...],
[561.	,	0.7101939...],
[562.	,	0.6996362...],
[563.	,	0.6890656...],
[564.	,	0.6785599...],
[565.	,	0.6680593...],
[566.	,	0.6575697...],
[567.	,	0.6471578...],
[568.	,	0.6368208...],
[569.	,	0.6264871...],
[570.	,	0.6161541...],
[571.	,	0.6058896...],
[572.	,	0.5957000...],
[573.	,	0.5855937...],
[574.	,	0.5754412...],
[575.	,	0.5653883...],
[576.	,	0.5553742...],
[577.	,	0.5454680...],
[578.	,	0.5355972...],
[579.	,	0.5258267...],
[580.	,	0.5160152...],
[581.	,	0.5062322...],
[582.	,	0.4965595...],
[583.	,	0.4868746...],
[584.	,	0.4773299...],
[585.	,	0.4678028...],
[586.	,	0.4583704...],
[587.	,	0.4489722...],
[588.	,	0.4397606...],
[589.	,	0.4306131...],
[590.	,	0.4215446...],
[591.	,	0.4125681...],
[592.	,	0.4037550...],
[593.	,	0.3950359...],
[594.	,	0.3864104...],
[595.	,	0.3778777...],
[596.	,	0.3694405...],
[597.	,	0.3611074...],
[598.	,	0.3528596...],
[599.	,	0.3447056...],
[600.	,	0.3366470...],
[601.	,	0.3286917...],
[602.	,	0.3208410...],
[603.	,	0.3130808...],
[604.	,	0.3054105...],
[605.	,	0.2978225...],
[606.	,	0.2903027...],
[607.	,	0.2828727...],
[608.	,	0.2755311...],
[609.	,	0.2682900...],
[610.	,	0.2611478...],
[611.	,	0.2541176...],
[612.	,	0.2471885...],
[613.	,	0.2403570...],

(continues on next page)

(continued from previous page)

[614.	,	0.2336057...],
[615.	,	0.2269379...],
[616.	,	0.2203527...],
[617.	,	0.2138465...],
[618.	,	0.2073946...],
[619.	,	0.2009789...],
[620.	,	0.1945818...],
[621.	,	0.1881943...],
[622.	,	0.1818226...],
[623.	,	0.1754987...],
[624.	,	0.1692476...],
[625.	,	0.1630876...],
[626.	,	0.1570257...],
[627.	,	0.151071 ...],
[628.	,	0.1452469...],
[629.	,	0.1395845...],
[630.	,	0.1341087...],
[631.	,	0.1288408...],
[632.	,	0.1237666...],
[633.	,	0.1188631...],
[634.	,	0.1141075...],
[635.	,	0.1094766...],
[636.	,	0.1049613...],
[637.	,	0.1005679...],
[638.	,	0.0962924...],
[639.	,	0.0921296...],
[640.	,	0.0880778...],
[641.	,	0.0841306...],
[642.	,	0.0802887...],
[643.	,	0.0765559...],
[644.	,	0.0729367...],
[645.	,	0.0694345...],
[646.	,	0.0660491...],
[647.	,	0.0627792...],
[648.	,	0.0596278...],
[649.	,	0.0565970...],
[650.	,	0.0536896...],
[651.	,	0.0509068...],
[652.	,	0.0482444...],
[653.	,	0.0456951...],
[654.	,	0.0432510...],
[655.	,	0.0409052...],
[656.	,	0.0386537...],
[657.	,	0.0364955...],
[658.	,	0.0344285...],
[659.	,	0.0324501...],
[660.	,	0.0305579...],
[661.	,	0.0287496...],
[662.	,	0.0270233...],
[663.	,	0.0253776...],
[664.	,	0.0238113...],
[665.	,	0.0223226...],
[666.	,	0.0209086...],
[667.	,	0.0195688...],
[668.	,	0.0183056...],
[669.	,	0.0171216...],

(continues on next page)

(continued from previous page)

[670.	,	0.0160192...],
[671.	,	0.0149986...],
[672.	,	0.0140537...],
[673.	,	0.0131784...],
[674.	,	0.0123662...],
[675.	,	0.0116107...],
[676.	,	0.0109098...],
[677.	,	0.0102587...],
[678.	,	0.0096476...],
[679.	,	0.0090665...],
[680.	,	0.0085053...],
[681.	,	0.0079567...],
[682.	,	0.0074229...],
[683.	,	0.0069094...],
[684.	,	0.0064213...],
[685.	,	0.0059637...],
[686.	,	0.0055377...],
[687.	,	0.0051402...],
[688.	,	0.00477 ...],
[689.	,	0.0044263...],
[690.	,	0.0041081...],
[691.	,	0.0038149...],
[692.	,	0.0035456...],
[693.	,	0.0032984...],
[694.	,	0.0030718...],
[695.	,	0.0028639...],
[696.	,	0.0026738...],
[697.	,	0.0025000...],
[698.	,	0.0023401...],
[699.	,	0.0021918...],
[700.	,	0.0020526...],
[701.	,	0.0019207...],
[702.	,	0.001796 ...],
[703.	,	0.0016784...],
[704.	,	0.0015683...],
[705.	,	0.0014657...],
[706.	,	0.0013702...],
[707.	,	0.001281 ...],
[708.	,	0.0011976...],
[709.	,	0.0011195...],
[710.	,	0.0010464...],
[711.	,	0.0009776...],
[712.	,	0.0009131...],
[713.	,	0.0008525...],
[714.	,	0.0007958...],
[715.	,	0.0007427...],
[716.	,	0.0006929...],
[717.	,	0.0006462...],
[718.	,	0.0006026...],
[719.	,	0.0005619...],
[720.	,	0.0005240...],
[721.	,	0.0004888...],
[722.	,	0.0004561...],
[723.	,	0.0004255...],
[724.	,	0.0003971...],
[725.	,	0.0003704...],

(continues on next page)

(continued from previous page)

```

[ 726.      , 0.0003455...],
[ 727.      , 0.0003221...],
[ 728.      , 0.0003001...],
[ 729.      , 0.0002796...],
[ 730.      , 0.0002604...],
[ 731.      , 0.0002423...],
[ 732.      , 0.0002254...],
[ 733.      , 0.0002095...],
[ 734.      , 0.0001947...],
[ 735.      , 0.0001809...],
[ 736.      , 0.0001680...],
[ 737.      , 0.0001560...],
[ 738.      , 0.0001449...],
[ 739.      , 0.0001345...],
[ 740.      , 0.0001249...],
[ 741.      , 0.0001159...],
[ 742.      , 0.0001076...],
[ 743.      , 0.0000999...],
[ 744.      , 0.0000927...],
[ 745.      , 0.0000862...],
[ 746.      , 0.0000801...],
[ 747.      , 0.0000745...],
[ 748.      , 0.0000693...],
[ 749.      , 0.0000646...],
[ 750.      , 0.0000602...],
[ 751.      , 0.0000561...],
[ 752.      , 0.0000523...],
[ 753.      , 0.0000488...],
[ 754.      , 0.0000456...],
[ 755.      , 0.0000425...],
[ 756.      , 0.0000397...],
[ 757.      , 0.0000370...],
[ 758.      , 0.0000346...],
[ 759.      , 0.0000322...],
[ 760.      , 0.0000301...],
[ 761.      , 0.0000281...],
[ 762.      , 0.0000262...],
[ 763.      , 0.0000244...],
[ 764.      , 0.0000228...],
[ 765.      , 0.0000213...],
[ 766.      , 0.0000198...],
[ 767.      , 0.0000185...],
[ 768.      , 0.0000173...],
[ 769.      , 0.0000161...],
[ 770.      , 0.0000150...],
[ 771.      , 0.0000140...],
[ 772.      , 0.0000131...],
[ 773.      , 0.0000122...],
[ 774.      , 0.0000114...],
[ 775.      , 0.0000106...],
[ 776.      , 0.0000099...],
[ 777.      , 0.0000092...],
[ 778.      , 0.0000086...],
[ 779.      , 0.0000080...],
[ 780.      , 0.0000075...]],
interpolator=SpragueInterpolator,

```

(continues on next page)

(continued from previous page)

```
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})
```

Dataset

colour

LEFS	Aggregated luminous efficiency functions.
PHOTOPIC_LEFS	Photopic luminous efficiency functions.
SCOTOPIC_LEFS	Scotopic luminous efficiency functions.

colour.LEFS

```
colour.LEFS = CaseInsensitiveMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modified CIE 1951 PH
Aggregated luminous efficiency functions.
```

References

[CVRc], [CVRe], [Wik05e]

LEFS [CaseInsensitiveMapping] {'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF', 'CIE 1951 Scotopic Standard Observer'}

colour.PHOTOPIC_LEFS

```
colour.PHOTOPIC_LEFS = CaseInsensitiveMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modified C
Photopic luminous efficiency functions.
```

References

[CVRc], [CVRe]

PHOTOPIC_LEFS [CaseInsensitiveMapping] {'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF'}

Aliases:

- 'cie_2_1924': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 Photopic 10 Degree Standard Observer'

colour.SCOTOPIC_LEFS

```
colour.SCOTOPIC_LEFS = CaseInsensitiveMapping({'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': ...})
Scotopic luminous efficiency functions.
```

References

[CVRe]

SCOTOPIC_LEFS [CaseInsensitiveMapping] {'CIE 1951 Scotopic Standard Observer', }

Aliases:

- 'cie_1951': 'CIE 1951 Scotopic Standard Observer'

Lightness Computation

colour

<code>lightness(Y[, method])</code>	Returns the <i>Lightness</i> L of given <i>luminance</i> Y using given method.
<code>LIGHTNESS_METHODS</code>	Supported <i>Lightness</i> computation methods.

colour.lightness

`colour.lightness(Y, method='CIE 1976', **kwargs)`

Returns the *Lightness* L of given *luminance* Y using given method.

Parameters

- **Y** (numeric or array_like) – *luminance* Y .
- **method** (unicode, optional) – {'CIE 1976', 'Glasser 1958', 'Wyszecki 1963', 'Fairchild 2010', 'Fairchild 2011'}, Computation method.

Other Parameters

- **Y_n** (numeric or array_like, optional) – {`colour.colorimetry.lightness_CIE1976()`}, White reference *luminance* Y_n .
- **epsilon** (numeric or array_like, optional) – {`colour.colorimetry.lightness_Fairchild2010()`, `colour.colorimetry.lightness_Fairchild2011()`}, ϵ exponent.

Returns *Lightness* L .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[CIET14804f], [FW10], [FC11], [GMRS58], [Wik07d], [Wys63], [WS00c]

Examples

```
>>> lightness(12.19722535) # doctest: +ELLIPSIS
41.5278758...
>>> lightness(12.19722535, Y_n=100) # doctest: +ELLIPSIS
41.5278758...
>>> lightness(12.19722535, Y_n=95) # doctest: +ELLIPSIS
42.5199307...
>>> lightness(12.19722535, method='Glasser 1958') # doctest: +ELLIPSIS
39.8351264...
>>> lightness(12.19722535, method='Wyszecki 1963') # doctest: +ELLIPSIS
40.5475745...
>>> lightness(12.19722535, epsilon=0.710, method='Fairchild 2011')
... # doctest: +ELLIPSIS
29.8295108...
```

colour.LIGHTNESS_METHODS

`colour.LIGHTNESS_METHODS` = `CaseInsensitiveMapping`({'Glasser 1958': ..., 'Wyszecki 1963': ..., 'CIE 1976': ...})
Supported *Lightness* computation methods.

References

[CIET14804f], [FW10], [FC11], [GMRS58], [Wys63], [WS00c]

LIGHTNESS_METHODS [`CaseInsensitiveMapping`] {'Glasser 1958', 'Wyszecki 1963', 'CIE 1976', 'Fairchild 2010', 'Fairchild 2011'}

Aliases:

- 'Lstar1976': 'CIE 1976'

Glasser, Mckinney, Reilly and Schnelle (1958)

`colour.colorimetry`

<code>lightness_Glasser1958(Y)</code>	Returns the <i>Lightness</i> L of given <i>luminance</i> Y using <i>Glasser et al. (1958)</i> method..
---------------------------------------	--

colour.colorimetry.lightness_Glasser1958

`colour.colorimetry.lightness_Glasser1958(Y)`
Returns the *Lightness* L of given *luminance* Y using *Glasser et al. (1958)* method.

Parameters Y (numeric or array_like) – *luminance* Y .

Returns *Lightness* L .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[GMRS58]

Examples

```
>>> lightness_Glasser1958(12.19722535) # doctest: +ELLIPSIS
39.8351264...
```

Wyszecki (1963)

colour.colorimetry

<code>lightness_Wyszecki1963(Y)</code>	Returns the <i>Lightness W</i> of given <i>luminance Y</i> using <i>Wyszecki (1963)</i> method.
--	---

colour.colorimetry.lightness_Wyszecki1963

colour.colorimetry.**lightness_Wyszecki1963**(Y)

Returns the *Lightness W* of given *luminance Y* using *Wyszecki (1963)* method.

Parameters Y (numeric or array_like) – *luminance Y*.

Returns *Lightness W*.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[Wys63]

Examples

```
>>> lightness_Wyszecki1963(12.19722535) # doctest: +ELLIPSIS
40.5475745...
```

CIE 1976

colour.colorimetry

<code>lightness_CIE1976(Y[, Y_n])</code>	Returns the <i>Lightness</i> L^* of given <i>luminance</i> Y using given reference white <i>luminance</i> Y_n as per CIE 1976 recommendation.
<code>intermediate_lightness_function_CIE1976(Y[, Y_n])</code>	Returns the intermediate value $f(Y/Y_n)$ in the <i>Lightness</i> L^* computation for given <i>luminance</i> Y using given reference white <i>luminance</i> Y_n as per CIE 1976 recommendation.

colour.colorimetry.lightness_CIE1976

colour.colorimetry.**lightness_CIE1976**(Y , $Y_n=100$)

Returns the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per CIE 1976 recommendation.

Parameters

- **Y** (numeric or array_like) – *luminance* Y .
- **Y_n** (numeric or array_like, optional) – White reference *luminance* Y_n .

Returns *Lightness* L^* .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> lightness_CIE1976(12.19722535) # doctest: +ELLIPSIS
41.5278758...
```

colour.colorimetry.intermediate_lightness_function_CIE1976

colour.colorimetry.**intermediate_lightness_function_CIE1976**(Y , $Y_n=100$)

Returns the intermediate value $f(Y/Y_n)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- Y (numeric or array_like) – *luminance* Y .
- Y_n (numeric or array_like, optional) – White reference *luminance* Y_n .

Returns Intermediate value $f(Y/Y_n)$.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

Range	Scale - Reference	Scale - 1
f_{Y/Y_n}	[0, 1]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> intermediate_lightness_function_CIE1976(12.19722535)
... # doctest: +ELLIPSIS
0.4959299...
>>> intermediate_lightness_function_CIE1976(12.19722535, 95)
... # doctest: +ELLIPSIS
0.5044821...
```

Fairchild and Wyble (2010)

colour.colorimetry

<code>lightness_Fairchild2010(Y[, epsilon])</code>	Computes <i>Lightness</i> L_{hdr} of given <i>luminance</i> Y using <i>Fairchild and Wyble (2010)</i> method according to <i>Michealis-Menten</i> kinetics.
--	---

colour.colorimetry.lightness_Fairchild2010

`colour.colorimetry.lightness_Fairchild2010(Y, epsilon=1.836)`

Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

Parameters

- **Y** (array_like) – *luminance* Y .
- **epsilon** (numeric or array_like, optional) – ϵ exponent.

Returns *Lightness* L_{hdr} .

Return type array_like

Warning: The input domain of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FW10]

Examples

```
>>> lightness_Fairchild2010(12.19722535 / 100) # doctest: +ELLIPSIS
31.9963902...
```

Fairchild and Chen (2011)

colour.colorimetry

<code>lightness_Fairchild2011(Y[, epsilon, method])</code>	Computes <i>Lightness</i> L_{hdr} of given <i>luminance</i> Y using <i>Fairchild and Chen (2011)</i> method according to <i>Michealis-Menten</i> kinetics.
--	--

colour.colorimetry.lightness_Fairchild2011

`colour.colorimetry.lightness_Fairchild2011(Y, epsilon=0.474, method='hdr-CIELAB')`

Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

Parameters

- **Y** (array_like) – *luminance* Y .
- **epsilon** (numeric or array_like, optional) – ϵ exponent.
- **method** (unicode, optional) – {'hdr-CIELAB', 'hdr-IPT'}, *Lightness* L_{hdr} computation method.

Returns *Lightness* L_{hdr} .

Return type array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FC11]

Examples

```
>>> lightness_Fairchild2011(12.19722535 / 100) # doctest: +ELLIPSIS
51.8529584...
>>> lightness_Fairchild2011(12.19722535 / 100, method='hdr-IPT')
... # doctest: +ELLIPSIS
51.6431084...
```

Luminance Computation

colour

<code>luminance(LV[, method])</code>	Returns the <i>luminance</i> Y of given <i>Lightness</i> L^* or given <i>Munsell</i> value V .
<code>LUMINANCE_METHODS</code>	Supported <i>luminance</i> computation methods.

colour.luminance

`colour.luminance(LV, method='CIE 1976', **kwargs)`

Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V .

Parameters

- **LV** (numeric or array_like) – *Lightness* L^* or *Munsell* value V .
- **method** (unicode, optional) – {'CIE 1976', 'Newhall 1943', 'ASTM D1535-08', 'Fairchild 2010', 'Fairchild 2011'}, Computation method.

Other Parameters

- **Y_n** (numeric or array_like, optional) – {`colour.colorimetry.luminance_CIE1976()`}, White reference *luminance* Y_n .
- **epsilon** (numeric or array_like, optional) – {`colour.colorimetry.lightness_Fairchild2010()`, `colour.colorimetry.lightness_Fairchild2011()`}, ϵ exponent.

Returns *luminance* Y .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
LV	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[ASTMInternational08], [CIET14804f], [FW10], [FC11], [NNJ43], [Wik01c], [WS00c]

Examples

```
>>> luminance(41.527875844653451) # doctest: +ELLIPSIS
12.1972253...
>>> luminance(41.527875844653451, Y_n=100) # doctest: +ELLIPSIS
12.1972253...
>>> luminance(42.51993072812094, Y_n=95) # doctest: +ELLIPSIS
12.1972253...
>>> luminance(4.08244375 * 10, method='Newhall 1943')
... # doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```

12.5500788...
>>> luminance(4.08244375 * 10, method='ASTM D1535-08')
... # doctest: +ELLIPSIS
12.2363426...
>>> luminance(29.829510892279330, epsilon=0.710, method='Fairchild 2011')
... # doctest: +ELLIPSIS
12.1972253...

```

colour.LUMINANCE_METHODS

`colour.LUMINANCE_METHODS` = `CaseInsensitiveMapping`({'Newhall 1943': ..., 'ASTM D1535-08': ..., 'CIE 1976': ...})
Supported *luminance* computation methods.

References

[ASTMInternational08], [CIET14804f], [FW10], [FC11], [NNJ43], [WS00c]

LUMINANCE_METHODS [`CaseInsensitiveMapping`] {'Newhall 1943', 'ASTM D1535-08', 'CIE 1976', 'Fairchild 2010'}

Aliases:

- 'astm2008': 'ASTM D1535-08'
- 'cie1976': 'CIE 1976'

Newhall, Nickerson and Judd (1943)

`colour.colorimetry`

<code>luminance_Newhall1943(V)</code>	Returns the <i>luminance</i> R_Y of given <i>Munsell</i> value V using <i>Newhall et al. (1943)</i> method..
---------------------------------------	--

colour.colorimetry.luminance_Newhall1943

`colour.colorimetry.luminance_Newhall1943(V)`

Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et al. (1943)* method.

Parameters V (numeric or array_like) – *Munsell* value V .

Returns *luminance* R_Y .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

Range	Scale - Reference	Scale - 1
R_Y	[0, 100]	[0, 1]

References

[NNJ43]

Examples

```
>>> luminance_Newhall1943(4.08244375) # doctest: +ELLIPSIS
12.5500788...
```

CIE 1976

colour.colorimetry

<code>luminance_CIE1976(L_star[, Y_n])</code>	Returns the <i>luminance</i> Y of given <i>Lightness</i> L^* with given reference white <i>luminance</i> Y_n .
<code>intermediate_luminance_function_CIE1976(f_Y_Y_n)</code>	Returns the <i>luminance</i> Y in the <i>luminance</i> Y computation for given intermediate value $f(Y/Y_n)$ using given reference white <i>luminance</i> Y_n as per CIE 1976 recommendation.

colour.colorimetry.luminance_CIE1976

colour.colorimetry.**luminance_CIE1976**(L_star , $Y_n=100$)

Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n .

Parameters

- **L_star** (numeric or array_like) – *Lightness* L^*
- **Y_n** (numeric or array_like) – White reference *luminance* Y_n .

Returns *luminance* Y .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> luminance_CIE1976(41.527875844653451) # doctest: +ELLIPSIS
12.1972253...
>>> luminance_CIE1976(41.527875844653451, 95) # doctest: +ELLIPSIS
11.5873640...
```

colour.colorimetry.intermediate_luminance_function_CIE1976

colour.colorimetry.**intermediate_luminance_function_CIE1976**(*f_Y_Y_n*, *Y_n*=100)

Returns the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Y_n)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **f_Y_Y_n** (numeric or array_like) – Intermediate value $f(Y/Y_n)$.
- **Y_n** (numeric or array_like) – White reference *luminance* Y_n .

Returns *luminance* Y .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
f_{Y/Y_n}	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

References

[CIET14804f], [WS00c]

Examples

```
>>> intermediate_luminance_function_CIE1976(0.495929964178047)
... # doctest: +ELLIPSIS
12.1972253...
>>> intermediate_luminance_function_CIE1976(0.504482161449319, 95)
... # doctest: +ELLIPSIS
12.1972253...
```

ASTM D1535-08e1

colour.colorimetry

<code>luminance_ASTMD153508(V)</code>	Returns the <i>luminance</i> Y of given <i>Munsell</i> value V using <i>ASTM D1535-08e1</i> method.
---------------------------------------	---

colour.colorimetry.luminance_ASTMD153508

colour.colorimetry.**luminance_ASTMD153508**(V)

Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

Parameters V (numeric or array_like) – *Munsell* value V .

Returns *luminance* Y .

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
V	$[0, 10]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
Y	$[0, 100]$	$[0, 1]$

References

[ASTMInternational08]

Examples

```
>>> luminance_ASTMD153508(4.08244375) # doctest: +ELLIPSIS
12.2363426...
```

Fairchild and Wyble (2010)

colour.colorimetry

<code>luminance_Fairchild2010(L_hdr[, epsilon])</code>	Computes <i>luminance</i> Y of given <i>Lightness</i> L_{hdr} using <i>Fairchild and Wyble (2010)</i> method according to <i>Michealis-Menten</i> kinetics.
--	---

colour.colorimetry.luminance_Fairchild2010

`colour.colorimetry.luminance_Fairchild2010(L_hdr, epsilon=1.836)`

Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

Parameters

- **L_hdr** (array_like) – *Lightness* L_{hdr} .
- **epsilon** (numeric or array_like, optional) – ϵ exponent.

Returns *luminance* Y .

Return type array_like

Warning: The output range of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
L_{hdr}	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FW10]

Examples

```
>>> luminance_Fairchild2010(31.996390226262736, 1.836)
... # doctest: +ELLIPSIS
0.1219722...
```

Fairchild and Chen (2011)

`colour.colorimetry`

<code>luminance_Fairchild2011(<i>L_hdr</i>[, <i>epsilon</i>, <i>method</i>])</code>	Computes <i>luminance</i> Y of given <i>Lightness</i> L_{hdr} using <i>Fairchild and Chen (2011)</i> method according to <i>Michealis-Menten</i> kinetics.
---	--

colour.colorimetry.luminance_Fairchild2011

`colour.colorimetry.luminance_Fairchild2011(L_hdr, epsilon=0.474, method='hdr-CIELAB')`

Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

Parameters

- **L_hdr** (array_like) – *Lightness* L_{hdr} .
- **epsilon** (numeric or array_like, optional) – ϵ exponent.
- **method** (unicode, optional) – {'hdr-CIELAB', 'hdr-IPT'}, *Lightness* L_{hdr} computation method.

Returns *luminance* Y .

Return type array_like

Notes

Domain	Scale - Reference	Scale - 1
L_{hdr}	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FC11]

Examples

```
>>> luminance_Fairchild2011(51.852958445912506) # doctest: +ELLIPSIS
0.1219722...
>>> luminance_Fairchild2011(51.643108411718522, method='hdr-IPT')
... # doctest: +ELLIPSIS
0.1219722...
```

Whiteness Computation

colour

<code>whiteness(XYZ, XYZ_0[, method])</code>	Returns the <i>whiteness</i> W using given method.
<code>WHITENESS_METHODS</code>	Supported <i>whiteness</i> computation methods.

colour.whiteness

`colour.whiteness(XYZ, XYZ_0, method='CIE 2004', **kwargs)`

Returns the *whiteness* W using given method.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of sample.
- **XYZ_0** (array_like) – CIE XYZ tristimulus values of reference white.
- **method** (unicode, optional) – {'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979'}, Computation method.

Other Parameters **observer** (unicode, optional) – {`colour.colorimetry.whiteness_CIE2004()`}, {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}, CIE Standard Observer used for computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.

Returns *whiteness* W .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[CIET14804i], [WS00k], [XRP12], [Wik04c]

Examples

```

>>> import numpy as np
>>> from colour.models import xyY_to_XYZ
>>> XYZ = xyY_to_XYZ(np.array([0.3167, 0.3334, 100]))
>>> XYZ_0 = xyY_to_XYZ(np.array([0.3139, 0.3311, 100]))
>>> whiteness(XYZ, XYZ_0) # doctest: +ELLIPSIS
array([ 93.85..., -1.305...])
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness(XYZ, XYZ_0, method='Taube 1960') # doctest: +ELLIPSIS
91.4071738...

```

colour.WHITENESS_METHODS

`colour.WHITENESS_METHODS` = `CaseInsensitiveMapping`({'Berger 1959': ..., 'Taube 1960': ..., 'Stensby 1968': ...})
Supported *whiteness* computation methods.

References

[CIET14804i], [XRP12]

WHITENESS_METHODS [`CaseInsensitiveMapping`] {'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979', 'CIE 2004'}

Aliases:

- 'cie2004': 'CIE 2004'

Berger (1959)

`colour.colorimetry`

<code>whiteness_Berger1959(XYZ, XYZ_0)</code>	Returns the <i>whiteness</i> index <i>WI</i> of given sample <i>CIE XYZ</i> tristimulus values using <i>Berger (1959)</i> method.
---	---

colour.colorimetry.whiteness_Berger1959

`colour.colorimetry.whiteness_Berger1959(XYZ, XYZ_0)`
Returns the *whiteness* index *WI* of given sample *CIE XYZ* tristimulus values using *Berger (1959)* method.

Parameters

- **XYZ** (`array_like`) – *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (`array_like`) – *CIE XYZ* tristimulus values of reference white.

Returns *Whiteness WI*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness WI* values larger than 33.33 indicate a bluish white and values smaller than 33.33 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness_Berger1959(XYZ, XYZ_0) # doctest: +ELLIPSIS
30.3638017...
```

Taube (1960)

colour.colorimetry

<code>whiteness-Taube1960(XYZ, XYZ_0)</code>	Returns the <i>whiteness</i> index <i>WI</i> of given sample <i>CIE XYZ</i> tristimulus values using <i>Taube (1960)</i> method.
--	--

colour.colorimetry.whiteness-Taube1960

colour.colorimetry.**whiteness-Taube1960**(XYZ, XYZ_0)

Returns the *whiteness* index *WI* of given sample *CIE XYZ* tristimulus values using *Taube (1960)* method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.

Returns *Whiteness WI*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness WI* values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness-Taube1960(XYZ, XYZ_0) # doctest: +ELLIPSIS
91.4071738...
```

Stensby (1968)

colour.colorimetry

<code>whiteness_Stensby1968(Lab)</code>	Returns the <i>whiteness</i> index <i>WI</i> of given sample CIE $L^*a^*b^*$ colourspace array using <i>Stensby (1968)</i> method.
---	--

colour.colorimetry.whiteness_Stensby1968

colour.colorimetry.**whiteness_Stensby1968**(*Lab*)

Returns the *whiteness* index *WI* of given sample CIE $L^*a^*b^*$ colourspace array using *Stensby (1968)* method.

Parameters *Lab* (array_like) – CIE $L^*a^*b^*$ colourspace array of sample.

Returns *Whiteness WI*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness WI* values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> Lab = np.array([100.00000000, -2.46875131, -16.72486654])
>>> whiteness_Stensby1968(Lab) # doctest: +ELLIPSIS
142.7683456...
```

ASTM E313

colour.colorimetry

<code>whiteness_ASTME313(XYZ)</code>	Returns the <i>whiteness</i> index <i>WI</i> of given sample <i>CIE XYZ</i> tristimulus values using <i>ASTM E313</i> method.
--------------------------------------	---

colour.colorimetry.whiteness_ASTME313

colour.colorimetry.**whiteness_ASTME313**(XYZ)

Returns the *whiteness* index *WI* of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

Parameters XYZ (array_like) – *CIE XYZ* tristimulus values of sample.

Returns *Whiteness WI*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> whiteness_ASTME313(XYZ) # doctest: +ELLIPSIS
55.7400000...
```

Ganz and Griesser (1979)

colour.colorimetry

<code>whiteness_Ganz1979(xy, Y)</code>	Returns the <i>whiteness</i> index W and <i>tint</i> T of given sample xy chromaticity coordinates using <i>Ganz and Griesser (1979)</i> method.
--	--

colour.colorimetry.whiteness_Ganz1979

colour.colorimetry.**whiteness_Ganz1979**(xy, Y)

Returns the *whiteness* index W and *tint* T of given sample xy chromaticity coordinates using *Ganz and Griesser (1979)* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates xy of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.

Returns *Whiteness* W and *tint* T .

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- The formula coefficients are valid for *CIE Standard Illuminant D Series D65* and *CIE 1964 10 Degree Standard Observer*.
- Positive output *tint* T values indicate a greener tint while negative values indicate a redder tint.
- Whiteness differences of less than 5 Ganz units appear to be indistinguishable to the human eye.
- Tint differences of less than 0.5 Ganz units appear to be indistinguishable to the human eye.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> whiteness_Ganz1979(xy, 100) # doctest: +ELLIPSIS
array([ 85.6003766...,  0.6789003...])
```

CIE 2004

colour.colorimetry

<code>whiteness_CIE2004(xy, Y, xy_n[, observer])</code>	Returns the <i>whiteness</i> W or W_{10} and <i>tint</i> T or T_{10} of given sample xy chromaticity coordinates using <i>CIE 2004</i> method.
---	--

colour.colorimetry.whiteness_CIE2004

colour.colorimetry.**whiteness_CIE2004**(xy, Y, xy_n, observer='CIE 1931 2 Degree Standard Observer')

Returns the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample xy chromaticity coordinates using *CIE 2004* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates xy of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.
- **xy_n** (array_like) – Chromaticity coordinates xy_n of perfect diffuser.
- **observer** (unicode, optional) – {'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}, *CIE Standard Observer* used for computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.

Returns *Whiteness* W or W_{10} and *tint* T or T_{10} of given sample.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- This method may be used only for samples whose values of W or W_{10} lie within the following limits: greater than 40 and less than $5Y - 280$, or $5Y_{10} - 280$.
- This method may be used only for samples whose values of T or T_{10} lie within the following limits: greater than -4 and less than +2.
- Output *whiteness* W or W_{10} values larger than 100 indicate a bluish white while values smaller than 100 indicate a yellowish white.
- Positive output *tint* T or T_{10} values indicate a greener tint while negative values indicate a redder tint.

References

[CIET14804i]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> xy_n = np.array([0.3139, 0.3311])
>>> whiteness_CIE2004(xy, 100, xy_n) # doctest: +ELLIPSIS
array([ 93.85..., -1.305...])
```

Yellowness Computation

colour

<code>yellowness(XYZ[, method])</code>	Returns the <i>yellowness</i> W using given method.
<code>YELLOWNESS_METHODS</code>	Supported <i>yellowness</i> computation methods.

colour.yellowness

`colour.yellowness(XYZ, method='ASTM E313')`
Returns the *yellowness* W using given method.

Parameters

- **XYZ** (array_like) – CIE XYZ tristimulus values of sample.
- **method** (unicode, optional) – {'ASTM E313', 'ASTM D1925'}, Computation method.

Returns *yellowness* Y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness(XYZ) # doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```

11.0650000...
>>> method = 'ASTM D1925'
>>> yellowness(XYZ, method=method) # doctest: +ELLIPSIS
10.2999999...

```

colour.YELLOWNESS_METHODS

`colour.YELLOWNESS_METHODS` = `CaseInsensitiveMapping`({'ASTM D1925': ..., 'ASTM E313': ...})
Supported *yellowness* computation methods.

References

[XRP12]

`YELLOWNESS_METHODS` [`CaseInsensitiveMapping`] {'ASTM E313', 'ASTM D1925'}

ASTM D1925

`colour.colorimetry`

<code>yellowness_ASTMD1925(XYZ)</code>	Returns the <i>yellowness</i> index <i>YI</i> of given sample <i>CIE XYZ</i> tristimulus values using <i>ASTM D1925</i> method.
--	---

colour.colorimetry.yellowness_ASTMD1925

`colour.colorimetry.yellowness_ASTMD1925(XYZ)`

Returns the *yellowness* index *YI* of given sample *CIE XYZ* tristimulus values using *ASTM D1925* method.

ASTM D1925 has been specifically developed for the definition of the Yellowness of homogeneous, non-fluorescent, almost neutral-transparent, white-scattering or opaque plastics as they will be reviewed under daylight condition. It can be other materials as well, as long as they fit into this description.

Parameters `XYZ` (`array_like`) – *CIE XYZ* tristimulus values of sample.

Returns *Whiteness YI*.

Return type numeric or `ndarray`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTMD1925(XYZ) # doctest: +ELLIPSIS
10.2999999...
```

ASTM E313

colour.colorimetry

<code>yellowness_ASTME313(XYZ)</code>	Returns the <i>yellowness</i> index <i>YI</i> of given sample <i>CIE XYZ</i> tristimulus values using <i>ASTM E313</i> method.
---------------------------------------	--

colour.colorimetry.yellowness_ASTME313

colour.colorimetry.**yellowness_ASTME313**(XYZ)

Returns the *yellowness* index *YI* of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

ASTM E313 has successfully been used for a variety of white or near white materials. This includes coatings, Plastics, Textiles.

Parameters XYZ (array_like) – *CIE XYZ* tristimulus values of sample.

Returns Whiteness *YI*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTME313(XYZ) # doctest: +ELLIPSIS
11.0650000...
```

Constants

- *CIE*
- *CODATA*
- *Common*

CIE

`colour.constants`

<code>K_M</code>	Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.
<code>KP_M</code>	Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

`colour.constants.K_M`

`colour.constants.K_M = 683.0`

Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.

`K_M` : numeric

Notes

- To be adequate for all practical applications the K_m value has been rounded from the original 683.002 value.

References

[WS00g]

`colour.constants.KP_M`

`colour.constants.KP_M = 1700.0`

Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

`KP_M` : numeric

Notes

- To be adequate for all practical applications the K'_m value has been rounded from the original 1700.06 value.

References

[WS00g]

CODATA

colour.constants

AVOGADRO_CONSTANT	Avogadro constant.
BOLTZMANN_CONSTANT	Boltzmann constant.
LIGHT_SPEED	Speed of light in vacuum.
PLANCK_CONSTANT	Planck constant.

colour.constants.AVOGADRO_CONSTANT

colour.constants.AVOGADRO_CONSTANT = 6.02214179e+23

Avogadro constant.

AVOGADRO_CONSTANT : numeric

colour.constants.BOLTZMANN_CONSTANT

colour.constants.BOLTZMANN_CONSTANT = 1.38065e-23

Boltzmann constant.

BOLTZMANN_CONSTANT : numeric

colour.constants.LIGHT_SPEED

colour.constants.LIGHT_SPEED = 299792458.0

Speed of light in vacuum.

LIGHT_SPEED : numeric

colour.constants.PLANCK_CONSTANT

colour.constants.PLANCK_CONSTANT = 6.62607e-34

Planck constant.

PLANCK_CONSTANT : numeric

Common

`colour.constants`

<code>DEFAULT_FLOAT_DTYPE</code>	alias of <code>numpy.float64</code>
<code>EPSILON</code>	
<code>FLOATING_POINT_NUMBER_PATTERN</code>	<code>str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str</code>
<code>INTEGER_THRESHOLD</code>	Integer threshold value.

`colour.constants.DEFAULT_FLOAT_DTYPE`

`colour.constants.DEFAULT_FLOAT_DTYPE`
alias of `numpy.float64`

`colour.constants.EPSILON`

`colour.constants.EPSILON = 2.2204460492503131e-16`

`colour.constants.FLOATING_POINT_NUMBER_PATTERN`

`colour.constants.FLOATING_POINT_NUMBER_PATTERN = '[0-9]*\.\?[0-9]+([eE][+-]?[0-9]+)?'`
`str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str`

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to 'strict'.

`colour.constants.INTEGER_THRESHOLD`

`colour.constants.INTEGER_THRESHOLD = 0.001`
Integer threshold value.
`INTEGER_THRESHOLD : numeric`

Contrast Sensitivity

- *Contrast Sensitivity*
- *Barten (1999) Contrast Sensitivity Function*

Contrast Sensitivity

`colour`

<code>contrast_sensitivity_function([method])</code>	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.
<code>CONTRAST_SENSITIVITY_METHODS</code>	Supported contrast sensitivity methods.

colour.contrast_sensitivity_function

`colour.contrast_sensitivity_function(method='Barten 1999', **kwargs)`

Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.

Parameters `method` (unicode, optional) – {'Barten 1999'}, Computation method.

Other Parameters

- **E** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Retinal illuminance E in Trolands.
- **N_max** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Maximum number of cycles N_{max} over which the eye can integrate the information.
- **T** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Integration time T in seconds of the eye.
- **X_0** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Angular size X_0 in degrees of the object in the x direction.
- **Y_0** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Angular size Y_0 in degrees of the object in the y direction.
- **X_max** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Maximum angular size Y_{max} in degrees of the integration area in the y direction.
- **k** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Signal-to-noise (SNR) ratio k .
- **n** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Quantum efficiency of the eye n .
- **p** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Photon conversion factor p in $\text{photons} \div \text{seconds} \div \text{degrees}^2 \div \text{Trolands}$ that depends on the light source.
- **phi_0** (numeric or array_like, optional) – {`colour.contrast.contrast_sensitivity_function_Barten1999()`}, Spectral density ϕ_0 in secondsdegrees^2 of the neural noise.

- **sigma** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **u** (*numeric*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spatial frequency u , the cycles per degree.
- **u_0** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spatial frequency u_0 in *cycles ÷ degrees* above which the lateral inhibition ceases.

Returns Contrast sensitivity S .

Return type ndarray

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function(u=4) # doctest: +ELLIPSIS
360.8691122...
>>> contrast_sensitivity_function('Barten 1999', u=4) # doctest: +ELLIPSIS
360.8691122...
```

colour.CONTRAST_SENSITIVITY_METHODS

colour.CONTRAST_SENSITIVITY_METHODS = CaseInsensitiveMapping({'Barten 1999': ...})
Supported contrast sensitivity methods.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

CONTRAST_SENSITIVITY_METHODS [CaseInsensitiveMapping] {'Barten 1999'}

Barten (1999) Contrast Sensitivity Function

colour.contrast

<code>contrast_sensitivity_function_Barten1999(u)</code>	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by <i>Barten (1999)</i> .
--	--

`colour.contrast.contrast_sensitivity_function_Barten1999`

```
colour.contrast.contrast_sensitivity_function_Barten1999(u, sigma=0.0087911571732306338,
                                                         k=3.0, T=0.1, X_0=60, Y_0=None,
                                                         X_max=12,          Y_max=None,
                                                         N_max=15, n=0.03, p=1227400.0,
                                                         E=66.082316060529919,
                                                         phi_0=3.0000000000000004e-08,
                                                         u_0=7)
```

Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

Contrast sensitivity is defined as the inverse of the modulation threshold of a sinusoidal luminance pattern. The modulation threshold of this pattern is generally defined by 50% probability of detection. The contrast sensitivity function or CSF gives the contrast sensitivity as a function of spatial frequency. In the CSF, the spatial frequency is expressed in angular units with respect to the eye. It reaches a maximum between 1 and 10 cycles per degree with a fall off at higher and lower spatial frequencies.

Parameters

- **u** (numeric) – Spatial frequency u , the cycles per degree.
- **sigma** (numeric or array_like, optional) – Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **k** (numeric or array_like, optional) – Signal-to-noise (SNR) ratio k .
- **T** (numeric or array_like, optional) – Integration time T in seconds of the eye.
- **X_0** (numeric or array_like, optional) – Angular size X_0 in degrees of the object in the x direction.
- **Y_0** (numeric or array_like, optional) – Angular size Y_0 in degrees of the object in the y direction.
- **X_max** (numeric or array_like, optional) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** (numeric or array_like, optional) – Maximum angular size Y_{max} in degrees of the integration area in the y direction.
- **N_max** (numeric or array_like, optional) – Maximum number of cycles N_{max} over which the eye can integrate the information.
- **n** (numeric or array_like, optional) – Quantum efficiency of the eye n .
- **p** (numeric or array_like, optional) – Photon conversion factor p in $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **E** (numeric or array_like, optional) – Retinal illuminance E in Trolands.
- **phi_0** (numeric or array_like, optional) – Spectral density ϕ_0 in $secondsdegrees^2$ of the neural noise.
- **u_0** (numeric or array_like, optional) – Spatial frequency u_0 in $cycles \div degrees$ above which the lateral inhibition ceases.

Returns Contrast sensitivity S .

Return type ndarray

Warning: This definition expects σ_0 and C_{ab} used in the computation of σ to be given in degrees and $\text{degrees} \div \text{mm}$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in arcmin and $\text{arcmin} \div \text{mm}$ respectively, thus they need to be divided by 60.

Notes

- The formula holds for bilateral viewing and for equal dimensions of the object in x and y direction. For monocular vision, the contrast sensitivity is a factor $\sqrt{2}$ smaller.
- *Barten (1999)* CSF default values for the k , σ_0 , C_{ab} , T , X_{max} , N_{max} , n , ϕ_0 and u_0 constants are valid for a standard observer with good vision and with an age between 20 and 30 years.
- The other constants have been filled using reference data from *Figure 31* in [InternationalTUnion15c] but must be adapted to the current use case.
- The product of u , the cycles per degree, and X_0 , the number of degrees, gives the number of cycles P_c in a pattern. Therefore, X_0 can be made a variable dependent on u such as $X_0 = P_c/u$.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function_Barten1999(4) # doctest: +ELLIPSIS
360.8691122...
```

Reproducing *Figure 31* in [InternationalTUnion15c] illustrating the minimum detectable contrast according to *Barten (1999)* model with the assumed conditions for UHDTV applications. The minimum detectable contrast MDC is then defined as follows:

$$:math:`MDC = 1 / CSF * 2 * (1 / 1.27)`$$

where 2 is used for the conversion from modulation to contrast and $1/1.27$ is used for the conversion from sinusoidal to rectangular waves.

```
>>> from scipy.optimize import fmin
>>> settings_BT2246 = {
...     'k': 3.0,
...     'T': 0.1,
...     'X_max': 12,
...     'N_max': 15,
...     'n': 0.03,
...     'p': 1.2274 * 10 ** 6,
...     'phi_0': 3 * 10 ** -8,
...     'u_0': 7,
... }
>>>
>>> def maximise_spatial_frequency(L):
...     maximised_spatial_frequency = []
...     for L_v in L:
...         X_0 = 60
...         d = pupil_diameter_Barten1999(L_v, X_0)
```

(continues on next page)

(continued from previous page)

```

...     sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
...     E = retinal_illuminance_Barten1999(L_v, d, True)
...     maximised_spatial_frequency.append(
...         fmin(lambda x: (
...             -contrast_sensitivity_function_Barten1999(
...                 u=x,
...                 sigma=sigma,
...                 X_0=X_0,
...                 E=E,
...                 **settings_BT2246)
...             ), 0, disp=False)[0])
...     return as_float(np.array(maximised_spatial_frequency))
>>>
>>> L = np.logspace(np.log10(0.01), np.log10(100), 10)
>>> X_0 = Y_0 = 60
>>> d = pupil_diameter_Barten1999(L, X_0, Y_0)
>>> sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
>>> E = retinal_illuminance_Barten1999(L, d)
>>> u = maximise_spatial_frequency(L)
>>> (1 / contrast_sensitivity_function_Barten1999(
...     u=u, sigma=sigma, E=E, X_0=X_0, Y_0=Y_0, **settings_BT2246)
...     * 2 * (1 / 1.27))
... # doctest: +ELLIPSIS
array([ 0.0207396...,  0.0134885...,  0.0096063...,  0.0077299...,  0.0068983...,
        0.0065057...,  0.0062712...,  0.0061198...,  0.0060365...,  0.0059984...])

```

Ancillary Objects

colour.contrast

<code>optical_MTF_Barten1999(u[, sigma])</code>	Returns the optical modulation transfer function (MTF) M_{opt} of the eye using <i>Barten (1999)</i> method.
<code>pupil_diameter_Barten1999(L[, X_0, Y_0])</code>	Returns the pupil diameter for given luminance and object or stimulus angular size using <i>Barten (1999)</i> method.
<code>sigma_Barten1999([sigma_0, C_ab, d])</code>	Returns the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using <i>Barten (1999)</i> method.
<code>retinal_illuminance_Barten1999(L[, d, ...])</code>	Returns the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using <i>Barten (1999)</i> method.
<code>maximum_angular_size_Barten1999(u[, X_0, ...])</code>	Returns the maximum angular size X of the object considered using <i>Barten (1999)</i> method.

colour.contrast.optical_MTF_Barten1999

colour.contrast.optical_MTF_Barten1999(*u*, *sigma*=0.01)

Returns the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.

Parameters

- **u** (numeric or array_like) – Spatial frequency u , the cycles per degree.
- **sigma** (numeric or array_like, optional) – Standard deviation σ of the line-spread

function resulting from the convolution of the different elements of the convolution process.

Returns Optical modulation transfer function (MTF) M_{opt} of the eye.

Return type numeric or array_like

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> optical_MTF_Barten1999(4, 0.01) # doctest: +ELLIPSIS
0.9689107...
```

colour.contrast.pupil_diameter_Barten1999

colour.contrast.pupil_diameter_Barten1999(L , $X_0=60$, $Y_0=None$)

Returns the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.

Parameters

- L (numeric or array_like) – Average luminance L in cd/m^2 .
- X_0 (numeric or array_like, optional) – Angular size of the object X_0 in degrees in the x direction.
- Y_0 (numeric or array_like, optional) – Angular size of the object X_0 in degrees in the y direction.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> pupil_diameter_Barten1999(100, 60, 60) # doctest: +ELLIPSIS
2.0777571...
```

colour.contrast.sigma_Barten1999

colour.contrast.sigma_Barten1999($\sigma_0=0.008333333333333333$,
 $C_{ab}=0.0013333333333333333$, $d=2.1$)

Returns the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using *Barten (1999)* method.

The σ quantity depends on the pupil diameter d of the eye lens. For very small pupil diameters, σ increases inversely proportionally with pupil size because of diffraction, and for large pupil diameters, σ increases about linearly with pupil size because of chromatic aberration and others aberrations.

Parameters

- **sigma_0** (numeric or array_like, optional) – Constant σ_0 in degrees.
- **C_ab** (numeric or array_like, optional) – Spherical aberration of the eye C_{ab} in *degrees ÷ mm*.
- **d** (numeric or array_like, optional) – Pupil diameter d in millimeters.

Returns Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.

Return type ndarray

Warning: This definition expects σ_0 and C_{ab} to be given in degrees and *degrees ÷ mm* respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in *arcmin* and *arcmin ÷ mm* respectively, thus they need to be divided by 60.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> sigma_Barten1999(0.5 / 60, 0.08 / 60, 2.1) # doctest: +ELLIPSIS
0.0087911...
```

colour.contrast.retinal_illuminance_Barten1999

colour.contrast.**retinal_illuminance_Barten1999**(L , $d=2.1$, *apply_stiles_crawford_effect_correction=True*)

Returns the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.

Parameters

- **L** (numeric or array_like) – Average luminance L in cd/m^2 .
- **d** (numeric or array_like, optional) – Pupil diameter d in millimeters.
- **apply_stiles_crawford_effect_correction** (bool, optional) – Whether to apply the correction for *Stiles-Crawford* effect.

Returns Retinal illuminance E in Trolands.

Return type ndarray

Notes

- This definition is for use with photopic viewing conditions and thus corrects for the Stiles-Crawford effect by default, i.e. directional sensitivity of the cone cells with lower response of cone cells receiving light from the edge of the pupil.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> retinal_illuminance_Barten1999(100, 2.1) # doctest: +ELLIPSIS
330.4115803...
>>> retinal_illuminance_Barten1999(100, 2.1, False) # doctest: +ELLIPSIS
346.3605900...
```

colour.contrast.maximum_angular_size_Barten1999

colour.contrast.maximum_angular_size_Barten1999(*u*, *X_0*=60, *X_max*=12, *N_max*=15)

Returns the maximum angular size X of the object considered using *Barten (1999)* method.

Parameters

- **u** (numeric) – Spatial frequency u , the cycles per degree.
- **X_0** (numeric or array_like, optional) – Angular size X_0 in degrees of the object in the x direction.
- **X_max** (numeric or array_like, optional) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **N_max** (numeric or array_like, optional) – Maximum number of cycles N_{max} over which the eye can integrate the information.

Returns Maximum angular size X of the object considered.

Return type numeric or ndarray

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> maximum_angular_size_Barten1999(4) # doctest: +ELLIPSIS
3.5729480...
```

Continuous Signal

- *Continuous Signal*

Continuous Signal

`colour.continuous`

<code>AbstractContinuousFunction([name])</code>	Defines the base class for abstract continuous function.
<code>Signal([data, domain])</code>	Defines the base class for continuous signal.
<code>MultiSignal([data, domain, labels])</code>	Defines the base class for multi-continuous signal, a container for multiple <code>colour.continuous.Signal</code> sub-class instances.

`colour.continuous.AbstractContinuousFunction`

class `colour.continuous.AbstractContinuousFunction(name=None)`

Defines the base class for abstract continuous function.

This is an ABCMeta abstract class that must be inherited by sub-classes.

The sub-classes are expected to implement the `colour.continuous.AbstractContinuousFunction.function()` method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. A conventional implementation adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the `colour.continuous.AbstractContinuousFunction.domain` attribute corresponds with the function dependent and already known range stored in the `colour.continuous.AbstractContinuousFunction.range` attribute.

Parameters `name` (unicode, optional) – Continuous function name.

`name`

`domain`

`range`

`interpolator`

`interpolator_args`

`extrapolator`

`extrapolator_args`

`function`

`__str__()`

`__repr__()`

`__hash__()`

`__getitem__()`

`__setitem__()`

`__contains__()`

`__len__()`

`__eq__()`

`__ne__()`

```

__iadd__()
__add__()
__isub__()
__sub__()
__imul__()
__mul__()
__idiv__()
__div__()
__ipow__()
__pow__()
arithmetical_operation()
fill_nan()
domain_distance()
is_uniform()
copy()
__init__(name=None)
    Initialize self. See help(type(self)) for accurate signature.

```

Methods

<code>__init__([name])</code>	Initialize self.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method, must be reimplemented by sub-classes.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.

Attributes

<code>domain</code>	Getter and setter property for the abstract continuous function independent domain <i>x</i> variable, must be reimplemented by sub-classes.
<code>extrapolator</code>	Getter and setter property for the abstract continuous function extrapolator type, must be reimplemented by sub-classes.

Continued on next page

Table 145 – continued from previous page

<code>extrapolator_args</code>	Getter and setter property for the abstract continuous function extrapolator instantiation time arguments, must be reimplemented by sub-classes.
<code>function</code>	Getter and setter property for the abstract continuous function callable, must be reimplemented by sub-classes.
<code>interpolator</code>	Getter and setter property for the abstract continuous function interpolator type, must be reimplemented by sub-classes.
<code>interpolator_args</code>	Getter and setter property for the abstract continuous function interpolator instantiation time arguments, must be reimplemented by sub-classes.
<code>name</code>	Getter and setter property for the abstract continuous function name.
<code>range</code>	Getter and setter property for the abstract continuous function corresponding range y variable, must be reimplemented by sub-classes.

colour.continuous.Signal

class colour.continuous.Signal(*data=None, domain=None, **kwargs*)

Defines the base class for continuous signal.

The class implements the `Signal.function()` method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. It adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the `colour.continuous.Signal.domain` attribute corresponds with the function dependent and already known range stored in the `colour.continuous.Signal.range` attribute.

Parameters

- **data** (Series or `Signal` or array_like or dict_like, optional) – Data to be stored in the continuous signal.
- **domain** (array_like, optional) – Values to initialise the `colour.continuous.Signal.domain` attribute with. If both data and domain arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.domain` attribute.

Other Parameters

- **name** (unicode, optional) – Continuous signal name.
- **dtype** (type, optional) – {`np.float16`, `np.float32`, `np.float64`, `np.float128`}, Floating point data type.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

`dtype`
`domain`
`range`
`interpolator`
`interpolator_args`
`extrapolator`
`extrapolator_args`
`function`
`__str__()`
`__repr__()`
`__hash__()`
`__getitem__()`
`__setitem__()`
`__contains__()`
`__eq__()`
`__ne__()`
`arithmetical_operation()`
`signal_unpack_data()`
`fill_nan()`
`to_series()`

Examples

Instantiation with implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(Signal(range_))
[[ 0.  10.]
 [ 1.  20.]
 [ 2.  30.]
 [ 3.  40.]
 [ 4.  50.]
 [ 5.  60.]
 [ 6.  70.]
 [ 7.  80.]
 [ 8.  90.]
 [ 9. 100.]]
```

Instantiation with explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(Signal(range_, domain))
[[ 100.  10.]
 [ 200.  20.]
 [ 300.  30.]
```

(continues on next page)

(continued from previous page)

```
[ 400.  40.]
[ 500.  50.]
[ 600.  60.]
[ 700.  70.]
[ 800.  80.]
[ 900.  90.]
[1000. 100.]
```

Instantiation with a *dict*:

```
>>> print(Signal(dict(zip(domain, range_))))
[[ 100.  10.]
 [ 200.  20.]
 [ 300.  30.]
 [ 400.  40.]
 [ 500.  50.]
 [ 600.  60.]
 [ 700.  70.]
 [ 800.  80.]
 [ 900.  90.]
 [1000. 100.]
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     print(Signal( # doctest: +SKIP
...         Series(dict(zip(domain, range_)))))
[[ 100.  10.]
 [ 200.  20.]
 [ 300.  30.]
 [ 400.  40.]
 [ 500.  50.]
 [ 600.  60.]
 [ 700.  70.]
 [ 800.  80.]
 [ 900.  90.]
 [1000. 100.]
```

Retrieving domain y variable for arbitrary range x variable:

```
>>> x = 150
>>> range_ = np.sin(np.linspace(0, 1, 10))
>>> Signal(range_, domain)[x] # doctest: +ELLIPSIS
0.0359701...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(range_, domain)[x] # doctest: +ELLIPSIS
array([ ...,  4.7669395...e-01,  8.4147098...e-01])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> Signal(
...     range_,
...     domain,
```

(continues on next page)

(continued from previous page)

```
...     interpolator=CubicSplineInterpolator)[x] # doctest: +ELLIPSIS
0.0555274...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x] # doctest: +ELLIPSIS
array([ 0.          ,  0.4794253...,  0.8414709...])
```

__init__(*data=None, domain=None, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([data, domain])</code>	Initialize self.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>signal_unpack_data([data, domain, dtype])</code>	Unpack given data for continuous signal instantiation.
<code>to_series()</code>	Converts the continuous signal to a <i>Pandas</i> Series class instance.

Attributes

<code>domain</code>	Getter and setter property for the continuous signal independent domain <i>x</i> variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the continuous signal extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the continuous signal extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the continuous signal callable.
<code>interpolator</code>	Getter and setter property for the continuous signal interpolator type.
<code>interpolator_args</code>	Getter and setter property for the continuous signal interpolator instantiation time arguments.

Continued on next page

Table 147 – continued from previous page

name	Getter and setter property for the abstract continuous function name.
range	Getter and setter property for the continuous signal corresponding range y variable.

colour.continuous.MultiSignal

class colour.continuous.**MultiSignal**(*data=None, domain=None, labels=None, **kwargs*)

Defines the base class for multi-continuous signal, a container for multiple `colour.continuous.Signal` sub-class instances.

Parameters

- **data** (Series or Dataframe or `Signal` or `MultiSignal` or array_like or dict_like, optional) – Data to be stored in the multi-continuous signal.
- **domain** (array_like, optional) – Values to initialise the multiple `colour.continuous.Signal` sub-class instances `colour.continuous.Signal.domain` attribute with. If both data and domain arguments are defined, the latter will be used to initialise the `colour.continuous.Signal.domain` attribute.
- **labels** (array_like, optional) – Names to use for the `colour.continuous.Signal` sub-class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-continuous signal name.
- **dtype** (*type, optional*) – {`np.float16`, `np.float32`, `np.float64`, `np.float128`}, Floating point data type.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the `colour.continuous.Signal` sub-class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.continuous.Signal` sub-class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.continuous.Signal` sub-class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.continuous.Signal` sub-class instances.
- **signal_type** (*type, optional*) – The `colour.continuous.Signal` sub-class type used for instances.

dtype**domain****range****interpolator****interpolator_args****extrapolator****extrapolator_args****function****signals**

`labels`
`signal_type`
`__str__()`
`__repr__()`
`__hash__()`
`__getitem__()`
`__setitem__()`
`__contains__()`
`__eq__()`
`__ne__()`
`arithmetical_operation()`
`multi_signal_unpack_data()`
`fill_nan()`
`to_dataframe()`

Examples

Instantiation with implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(MultiSignal(range_))
[[ 0.  10.]
 [ 1.  20.]
 [ 2.  30.]
 [ 3.  40.]
 [ 4.  50.]
 [ 5.  60.]
 [ 6.  70.]
 [ 7.  80.]
 [ 8.  90.]
 [ 9. 100.]
```

Instantiation with explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(MultiSignal(range_, domain))
[[ 100.  10.]
 [ 200.  20.]
 [ 300.  30.]
 [ 400.  40.]
 [ 500.  50.]
 [ 600.  60.]
 [ 700.  70.]
 [ 800.  80.]
 [ 900.  90.]
 [1000. 100.]
```

Instantiation with multiple signals:

```

>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignal(range_, domain))
[[ 100.  10.  20.  30.]
 [ 200.  20.  30.  40.]
 [ 300.  30.  40.  50.]
 [ 400.  40.  50.  60.]
 [ 500.  50.  60.  70.]
 [ 600.  60.  70.  80.]
 [ 700.  70.  80.  90.]
 [ 800.  80.  90. 100.]
 [ 900.  90. 100. 110.]
 [1000. 100. 110. 120.]]

```

Instantiation with a *dict*:

```

>>> print(MultiSignal(dict(zip(domain, range_))))
[[ 100.  10.  20.  30.]
 [ 200.  20.  30.  40.]
 [ 300.  30.  40.  50.]
 [ 400.  40.  50.  60.]
 [ 500.  50.  60.  70.]
 [ 600.  60.  70.  80.]
 [ 700.  70.  80.  90.]
 [ 800.  80.  90. 100.]
 [ 900.  90. 100. 110.]
 [1000. 100. 110. 120.]]

```

Instantiation using a *Signal* sub-class:

```

>>> class NotSignal(Signal):
...     pass

```

```

>>> multi_signal = MultiSignal(range_, domain, signal_type=NotSignal)
>>> print(multi_signal)
[[ 100.  10.  20.  30.]
 [ 200.  20.  30.  40.]
 [ 300.  30.  40.  50.]
 [ 400.  40.  50.  60.]
 [ 500.  50.  60.  70.]
 [ 600.  60.  70.  80.]
 [ 700.  70.  80.  90.]
 [ 800.  80.  90. 100.]
 [ 900.  90. 100. 110.]
 [1000. 100. 110. 120.]]
>>> type(multi_signal.signals[0]) # doctest: +SKIP
<class 'multi_signal.NotSignal'>

```

Instantiation with a *Pandas Series*:

```

>>> if is_pandas_installed():
...     from pandas import Series
...     print(MultiSignal( # doctest: +SKIP
...         Series(dict(zip(domain, np.linspace(10, 100, 10)))))
[[ 100.  10.]
 [ 200.  20.]
 [ 300.  30.]

```

(continues on next page)

(continued from previous page)

```
[ 400.  40.]
[ 500.  50.]
[ 600.  60.]
[ 700.  70.]
[ 800.  80.]
[ 900.  90.]
[1000. 100.]
```

Instantiation with a *Pandas Dataframe*:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     data = dict(zip(['a', 'b', 'c'], tsplit(range_)))
...     print(MultiSignal( # doctest: +SKIP
...         DataFrame(data, domain)))
[[ 100.  10.  20.  30.]
 [ 200.  20.  30.  40.]
 [ 300.  30.  40.  50.]
 [ 400.  40.  50.  60.]
 [ 500.  50.  60.  70.]
 [ 600.  60.  70.  80.]
 [ 700.  70.  80.  90.]
 [ 800.  80.  90. 100.]
 [ 900.  90. 100. 110.]
[1000. 100. 110. 120.]
```

Retrieving domain y variable for arbitrary range x variable:

```
>>> x = 150
>>> range_ = tstack([np.sin(np.linspace(0, 1, 10))] * 3)
>>> range_ += np.array([0.0, 0.25, 0.5])
>>> MultiSignal(range_, domain)[x] # doctest: +ELLIPSIS
array([ 0.0359701..., 0.2845447..., 0.5331193...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignal(range_, domain)[x] # doctest: +ELLIPSIS
array([[ 4.4085384...e-20,  2.5000000...e-01,  5.0000000...e-01],
       [ 4.7669395...e-01,  7.2526859...e-01,  9.7384323...e-01],
       [ 8.4147098...e-01,  1.0914709...e+00,  1.3414709...e+00]])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> MultiSignal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x] # doctest: +ELLIPSIS
array([ 0.0555274..., 0.3055274..., 0.5555274...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x] # doctest: +ELLIPSIS
array([[ 0.         ..., 0.25      ..., 0.5        ...],
       [ 0.4794253..., 0.7294253..., 0.9794253...],
       [ 0.8414709..., 1.0914709..., 1.3414709...]])
```


`__init__(data=None, domain=None, labels=None, **kwargs)`
 Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([data, domain, labels])</code>	Initialize self.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain <i>x</i> variable and corresponding range <i>y</i> variable using given method.
<code>is_uniform()</code>	Returns if independent domain <i>x</i> variable is uniform.
<code>multi_signal_unpack_data([data, domain, ...])</code>	Unpack given data for multi-continuous signal instantiation.
<code>to_dataframe()</code>	Converts the continuous signal to a <i>Pandas</i> DataFrame class instance.

Attributes

<code>domain</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances independent domain <i>x</i> variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances callable.
<code>interpolator</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator type.
<code>interpolator_args</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances interpolator instantiation time arguments.
<code>labels</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances name.
<code>name</code>	Getter and setter property for the abstract continuous function name.

Continued on next page

Table 149 – continued from previous page

<code>range</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances corresponding range y variable.
<code>signal_type</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances type.
<code>signals</code>	Getter and setter property for the <code>colour.continuous.Signal</code> sub-class instances.

Corresponding Chromaticities

- *Prediction*
 - *Fairchild (1990)*
 - *CIE 1994*
 - *CMCCAT2000*
 - *Von Kries*

Prediction

`colour`

<code>corresponding_chromaticities_prediction([...])</code>	Returns the corresponding chromaticities prediction for given chromatic adaptation model.
<code>CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS</code>	Aggregated corresponding chromaticities prediction models.

`colour.corresponding_chromaticities_prediction`

`colour.corresponding_chromaticities_prediction(experiment=1, model='Von Kries', **kwargs)`
Returns the corresponding chromaticities prediction for given chromatic adaptation model.

Parameters

- **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.
- **model** (unicode, optional) – {'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}, Chromatic adaptation model.

Other Parameters **transform** (unicode, optional) – {`colour.corresponding_chromaticities_prediction_VonKries()`, {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}}, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type `tuple`

References

[Bre87], [CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction(2, 'CMCCAT2000')
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr) # doctest: +SKIP
[((0.207, 0.486), (0.2083210..., 0.4727168...)),
 ((0.449, 0.511), (0.4459270..., 0.5077735...)),
 ((0.263, 0.505), (0.2640262..., 0.4955361...)),
 ((0.322, 0.545), (0.3316884..., 0.5431580...)),
 ((0.316, 0.537), (0.3222624..., 0.5357624...)),
 ((0.265, 0.553), (0.2710705..., 0.5501997...)),
 ((0.221, 0.538), (0.2261826..., 0.5294740...)),
 ((0.135, 0.532), (0.1439693..., 0.5190984...)),
 ((0.145, 0.472), (0.1494835..., 0.4556760...)),
 ((0.163, 0.331), (0.1563172..., 0.3164151...)),
 ((0.176, 0.431), (0.1763199..., 0.4127589...)),
 ((0.244, 0.349), (0.2287638..., 0.3499324...))]
```

colour.CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS

`colour.CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS = CaseInsensitiveMapping({'CIE 1994': ..., 'CMCCAT2000': ...})`
 Aggregated corresponding chromaticities prediction models.

References

[Bre87], [CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS [CaseInsensitiveMapping] {'CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries'}

Aliases:

- 'vonkries': 'Von Kries'

Dataset

colour

BRENEMAN_EXPERIMENTS	<i>Breneman (1987) experiments.</i>
BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES	<i>Breneman (1987) experiments primaries chromaticities.</i>

colour.BRENEMAN_EXPERIMENTS

`colour.BRENEMAN_EXPERIMENTS = {1: (BrenemanExperimentResult(name='Illuminant', uvp_t=array([ 0.259, 0.526])), ...)}`

References

[Bre87]

BRENEMAN_EXPERIMENTS : dict

colour.BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES

colour.BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES = {1: PrimariesChromaticityCoordinates(experiment=1, il
Breneman (1987) experiments primaries chromaticities.

References

[Bre87]

BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES : dict

Fairchild (1990)

colour.corresponding

corresponding_chromaticities_prediction_Fairchild1990 Returns (the) corresponding chromaticities predic-
tion for *Fairchild (1990)* chromatic adaptation
model.

colour.corresponding.corresponding_chromaticities_prediction_Fairchild1990

colour.corresponding.corresponding_chromaticities_prediction_Fairchild1990(*experiment=1*)
Returns the corresponding chromaticities prediction for *Fairchild (1990)* chromatic adaptation model.

Parameters *experiment* (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)*
experiment number.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [Fai91], [Fai13c]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_Fairchild1990(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr) # doctest: +SKIP
[[((0.207, 0.486), (0.2089528..., 0.4724034...)),
  ((0.449, 0.511), (0.4375652..., 0.5121030...)),
  ((0.263, 0.505), (0.2621362..., 0.4972538...))]
```

(continues on next page)

(continued from previous page)

```
((0.322, 0.545), (0.3235312..., 0.5475665...)),
((0.316, 0.537), (0.3151390..., 0.5398333...)),
((0.265, 0.553), (0.2634745..., 0.5544335...)),
((0.221, 0.538), (0.2211595..., 0.5324470...)),
((0.135, 0.532), (0.1396949..., 0.5207234...)),
((0.145, 0.472), (0.1512288..., 0.4533041...)),
((0.163, 0.331), (0.1715691..., 0.3026264...)),
((0.176, 0.431), (0.1825792..., 0.4077892...)),
((0.244, 0.349), (0.2418904..., 0.3413401...))]
```

CIE 1994

colour.corresponding

`corresponding_chromaticities_prediction_CIE1994` Returns the corresponding chromaticities prediction for CIE 1994 chromatic adaptation model.

`colour.corresponding.corresponding_chromaticities_prediction_CIE1994`

`colour.corresponding.corresponding_chromaticities_prediction_CIE1994(experiment=1)`
Returns the corresponding chromaticities prediction for CIE 1994 chromatic adaptation model.

Parameters `experiment` (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)*
experiment number.

Returns Corresponding chromaticities prediction.

Return type `tuple`

References

[Bre87], [CIET13294]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CIE1994(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr) # doctest: +SKIP
[[((0.207, 0.486), (0.2133909..., 0.4939794...)),
 ((0.449, 0.511), (0.4450345..., 0.5120939...)),
 ((0.263, 0.505), (0.2693262..., 0.5083212...)),
 ((0.322, 0.545), (0.3308593..., 0.5443940...)),
 ((0.316, 0.537), (0.3225195..., 0.5377826...)),
 ((0.265, 0.553), (0.2709737..., 0.5513666...)),
 ((0.221, 0.538), (0.2280786..., 0.5351592...)),
 ((0.135, 0.532), (0.1439436..., 0.5303576...)),
 ((0.145, 0.472), (0.1500743..., 0.4842895...)),
 ((0.163, 0.331), (0.1559955..., 0.3772379...)),
 ((0.176, 0.431), (0.1806318..., 0.4518475...)),
 ((0.244, 0.349), (0.2454445..., 0.4018004...))]
```

CMCCAT2000

`colour.corresponding`

`corresponding_chromaticities_prediction_CMCCAT2000`(`fn`). Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

`colour.corresponding.corresponding_chromaticities_prediction_CMCCAT2000`

`colour.corresponding.corresponding_chromaticities_prediction_CMCCAT2000`(`experiment=1`)
Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

Parameters `experiment` (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)*
experiment number.

Returns Corresponding chromaticities prediction.

Return type `tuple`

References

[Bre87], [LLRH02], [WRC12a]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CMCCAT2000(2)
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr) # doctest: +SKIP
[[ (0.207, 0.486), (0.2083210..., 0.4727168...),
  (0.449, 0.511), (0.4459270..., 0.5077735...),
  (0.263, 0.505), (0.2640262..., 0.4955361...),
  (0.322, 0.545), (0.3316884..., 0.5431580...),
  (0.316, 0.537), (0.3222624..., 0.5357624...),
  (0.265, 0.553), (0.2710705..., 0.5501997...),
  (0.221, 0.538), (0.2261826..., 0.5294740...),
  (0.135, 0.532), (0.1439693..., 0.5190984...),
  (0.145, 0.472), (0.1494835..., 0.4556760...),
  (0.163, 0.331), (0.1563172..., 0.3164151...),
  (0.176, 0.431), (0.1763199..., 0.4127589...),
  (0.244, 0.349), (0.2287638..., 0.3499324...)]
```

Von Kries

`colour.corresponding`

`corresponding_chromaticities_prediction_VonKries`(`fn`). Returns the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

colour.corresponding.corresponding_chromaticities_prediction_VonKries

`colour.corresponding.corresponding_chromaticities_prediction_VonKries(experiment=1, transform='CAT02')`

Returns the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

Parameters

- **experiment** (integer, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number.
- **transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type `tuple`

References

[Bre87], [Fail3b]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_VonKries(2, 'Bradford')
>>> pr = [(p.uvp_m, p.uvp_p) for p in pr]
>>> pprint(pr) # doctest: +SKIP
[((0.207, 0.486), (0.2082014..., 0.4722922...)),
 ((0.449, 0.511), (0.4489102..., 0.5071602...)),
 ((0.263, 0.505), (0.2643545..., 0.4959631...)),
 ((0.322, 0.545), (0.3348730..., 0.5471220...)),
 ((0.316, 0.537), (0.3248758..., 0.5390589...)),
 ((0.265, 0.553), (0.2733105..., 0.5555028...)),
 ((0.221, 0.538), (0.2271480..., 0.5331317...)),
 ((0.135, 0.532), (0.1442730..., 0.5226804...)),
 ((0.145, 0.472), (0.1498745..., 0.4550785...)),
 ((0.163, 0.331), (0.1564975..., 0.3148795...)),
 ((0.176, 0.431), (0.1760593..., 0.4103772...)),
 ((0.244, 0.349), (0.2259805..., 0.3465291...))]
```

Colour Difference

- *Delta E*
- *CIE 1976*
- *CIE 1994*
- *CIE 2000*
- *CMC*

- *Luo, Cui and Li (2006)*
- *Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)*
- *DIN99*

Delta E

colour

<code>delta_E(a, b[, method])</code>	Returns the difference ΔE_{ab} between two given CIE $L^*a^*b^*$ or $J'a'b'$ colour space arrays using given method.
<code>DELTA_E_METHODS</code>	Supported ΔE_{ab} computation methods.

colour.delta_E

`colour.delta_E(a, b, method='CIE 2000', **kwargs)`

Returns the difference ΔE_{ab} between two given CIE $L^*a^*b^*$ or $J'a'b'$ colour space arrays using given method.

Parameters

- **a** (array_like) – CIE $L^*a^*b^*$ or $J'a'b'$ colour space array a .
- **b** (array_like) – CIE $L^*a^*b^*$ or $J'a'b'$ colour space array b .
- **method** (unicode, optional) – {'CIE 2000', 'CIE 1976', 'CIE 1994', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'} Computation method.

Other Parameters

- **textiles** (bool, optional) – {`colour.difference.delta_E_CIE1994()`, `colour.difference.delta_E_CIE2000()`, `colour.difference.delta_E_DIN99()`}, Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$, $k_1 = 0.048$, $k_2 = 0.014$, $k_E = 2$, $k_C H = 0.5$ weights are used instead of $k_L = k_C = k_H = 1$, $k_1 = 0.045$, $k_2 = 0.015$, $k_E = k_C H = 1.0$.
- **l** (numeric, optional) – {`colour.difference.delta_E_CIE2000()`}, Lightness weighting factor.
- **c** (numeric, optional) – {`colour.difference.delta_E_CIE2000()`}, Chroma weighting factor.

Returns Colour difference ΔE_{ab} .

Return type numeric or ndarray

References

[ASTMInternational07], [LLW+17], [Lin03a], [Lin11], [Lin09b], [Lin09c], [LCL06], [Mel13], [Wik08c]

Examples

```
>>> import numpy as np
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E(a, b) # doctest: +ELLIPSIS
94.0356490...
>>> delta_E(a, b, method='CIE 2000') # doctest: +ELLIPSIS
94.0356490...
>>> delta_E(a, b, method='CIE 1976') # doctest: +ELLIPSIS
451.7133019...
>>> delta_E(a, b, method='CIE 1994') # doctest: +ELLIPSIS
83.7792255...
>>> delta_E(a, b, method='CIE 1994', textiles=False)
... # doctest: +ELLIPSIS
83.7792255...
>>> delta_E(a, b, method='DIN99') # doctest: +ELLIPSIS
66.1119282...
>>> a = np.array([54.90433134, -0.08450395, -0.06854831])
>>> b = np.array([54.90433134, -0.08442362, -0.06848314])
>>> delta_E(a, b, method='CAM02-UCS') # doctest: +ELLIPSIS
0.0001034...
>>> delta_E(a, b, method='CAM16-LCD') # doctest: +ELLIPSIS
0.0001034...
```

colour.DELTA_E_METHODS

`colour.DELTA_E_METHODS` = `CaseInsensitiveMapping`({'CIE 1976': ..., 'CIE 1994': ..., 'CIE 2000': ..., 'CMC': ...})
Supported ΔE_{ab} computation methods.

References

[ASTMInternational07], [LLW+17], [Lin03a], [Lin11], [Lin09b], [Lin09c], [LCL06], [Mel13], [Wik08c]

DELTA_E_METHODS [`CaseInsensitiveMapping`] {'CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'}

Aliases:

- 'cie1976': 'CIE 1976'
- 'cie1994': 'CIE 1994'
- 'cie2000': 'CIE 2000'

CIE 1976

`colour.difference`

`delta_E_CIE1976(Lab_1, Lab_2)`Returns the difference ΔE_{76} between two given *CIE* $L^*a^*b^*$ colourspace arrays using *CIE 1976* recommendation.

`colour.difference.delta_E_CIE1976`

`colour.difference.delta_E_CIE1976(Lab_1, Lab_2)`Returns the difference ΔE_{76} between two given *CIE* $L^*a^*b^*$ colourspace arrays using *CIE 1976* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE* $L^*a^*b^*$ colourspace array 1.
- **Lab_2** (array_like) – *CIE* $L^*a^*b^*$ colourspace array 2.

Returns Colour difference ΔE_{76} .**Return type** numeric or ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[\[Lin03a\]](#)

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1976(Lab_1, Lab_2) # doctest: +ELLIPSIS
451.7133019...
```

CIE 1994

`colour.difference`

<code>delta_E_CIE1994(Lab_1, Lab_2[, textiles])</code>	Returns the difference ΔE_{94} between two given <i>CIE</i> $L^*a^*b^*$ colourpace arrays using <i>CIE 1994</i> recommendation.
--	---

`colour.difference.delta_E_CIE1994`

`colour.difference.delta_E_CIE1994(Lab_1, Lab_2, textiles=False)`

Returns the difference ΔE_{94} between two given *CIE* $L^*a^*b^*$ colourpace arrays using *CIE 1994* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE* $L^*a^*b^*$ colourpace array 1.
- **Lab_2** (array_like) – *CIE* $L^*a^*b^*$ colourpace array 2.
- **textiles** (bool, optional) – Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$, $k_1 = 0.048$, $k_2 = 0.014$ weights are used instead of $k_L = k_C = k_H = 1$, $k_1 = 0.045$, $k_2 = 0.015$.

Returns Colour difference ΔE_{94} .

Return type numeric or ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- *CIE 1994* colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.

References

[[Lin11](#)]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1994(Lab_1, Lab_2) # doctest: +ELLIPSIS
83.7792255...
>>> delta_E_CIE1994(Lab_1, Lab_2, textiles=True) # doctest: +ELLIPSIS
88.3355530...
```

CIE 2000

colour.difference

<code>delta_E_CIE2000(Lab_1, Lab_2[, textiles])</code>	Returns the difference ΔE_{00} between two given CIE $L^*a^*b^*$ colour space arrays using CIE 2000 recommendation.
--	---

colour.difference.delta_E_CIE2000

colour.difference.delta_E_CIE2000(Lab_1, Lab_2, textiles=False)

Returns the difference ΔE_{00} between two given CIE $L^*a^*b^*$ colour space arrays using CIE 2000 recommendation.

Parameters

- **Lab_1** (array_like) – CIE $L^*a^*b^*$ colour space array 1.
- **Lab_2** (array_like) – CIE $L^*a^*b^*$ colour space array 2.
- **textiles** (bool, optional) – Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$ weights are used instead of $k_L = k_C = k_H = 1$.

Returns Colour difference ΔE_{00} .

Return type numeric or ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- CIE 2000 colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.
- Parametric factors $k_L = k_C = k_H = 1$ weights under *reference conditions*:
 - Illumination: D65 source
 - Illuminance: 1000 lx
 - Observer: Normal colour vision
 - Background field: Uniform, neutral gray with $L^* = 50$
 - Viewing mode: Object
 - Sample size: Greater than 4 degrees
 - Sample separation: Direct edge contact

- Sample colour-difference magnitude: Lower than 5.0 ΔE_{00}
- Sample structure: Homogeneous (without texture)

References

[Lin09b], [Mel13]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2) # doctest: +ELLIPSIS
94.0356490...
>>> Lab_2 = np.array([50.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2) # doctest: +ELLIPSIS
100.8779470...
>>> delta_E_CIE2000(Lab_1, Lab_2, textiles=True) # doctest: +ELLIPSIS
95.7920535...
```

CMC

colour.difference

<code>delta_E_CMC(Lab_1, Lab_2[, l, c])</code>	Returns the difference ΔE_{CMC} between two given CIE $L^*a^*b^*$ colourspace arrays using <i>Colour Measurement Committee</i> recommendation.
--	--

colour.difference.delta_E_CMC

colour.difference.**delta_E_CMC**(Lab_1, Lab_2, l=2, c=1)

Returns the difference ΔE_{CMC} between two given CIE $L^*a^*b^*$ colourspace arrays using *Colour Measurement Committee* recommendation.

The quasimetric has two parameters: *Lightness* (l) and *chroma* (c), allowing the users to weight the difference based on the ratio of l:c. Commonly used values are 2:1 for acceptability and 1:1 for the threshold of imperceptibility.

Parameters

- **Lab_1** (array_like) – CIE $L^*a^*b^*$ colourspace array 1.
- **Lab_2** (array_like) – CIE $L^*a^*b^*$ colourspace array 2.
- **l** (numeric, optional) – Lightness weighting factor.
- **c** (numeric, optional) – Chroma weighting factor.

Returns Colour difference ΔE_{CMC} .

Return type numeric or ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[Lin09c]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CMC(Lab_1, Lab_2) # doctest: +ELLIPSIS
172.7047712...
```

Luo, Cui and Li (2006)

`colour.difference`

<code>delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al. (2017) CAM16-LCD</i> colourspaces $J'a'b'$ arrays..
<code>delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al. (2017) CAM16-SCD</i> colourspaces $J'a'b'$ arrays..
<code>delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al. (2017) CAM16-UCS</i> colourspaces $J'a'b'$ arrays..

`colour.difference.delta_E_CAM02LCD`

`colour.difference.delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)`

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02LCD()` definition for an usage example.

colour.difference.delta_E_CAM02SCD`colour.difference.delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)`Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.**Parameters**

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.**Return type** numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

`colour.difference.delta_E_CAM02UCS`

`colour.difference.delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)`

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02UCS()` definition for an usage example.

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

`colour.difference`

<code>delta_E_CAM16LCD(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al.(2017) CAM16-LCD</i> colourspaces $J'a'b'$ arrays..
<code>delta_E_CAM16SCD(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al.(2017) CAM16-SCD</i> colourspaces $J'a'b'$ arrays..

Continued on next page

Table 162 – continued from previous page

<code>delta_E_CAM16UCS(Jpapbp_1, Jpapbp_2)</code>	Returns the difference $\Delta E'$ between two given <i>Li et al. (2017) CAM16-UCS</i> colourspaces $J'a'b'$ arrays..
---	---

colour.difference.delta_E_CAM16LCD

`colour.difference.delta_E_CAM16LCD(Jpapbp_1, Jpapbp_2)`

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02LCD()` definition for an usage example.

colour.difference.delta_E_CAM16SCD

`colour.difference.delta_E_CAM16SCD(Jpapbp_1, Jpapbp_2)`

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

`colour.difference.delta_E_CAM16UCS`

`colour.difference.delta_E_CAM16UCS(Jpapbp_1, Jpapbp_2)`

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02UCS()` definition for an usage example.

DIN99

`colour.difference`

<code>delta_E_DIN99(Lab_1, Lab_2[, textiles])</code>	Returns the difference ΔE_{DIN99} between two given <i>CIE L*a*b*</i> colourspace arrays using <i>DIN99</i> formula.
--	--

`colour.difference.delta_E_DIN99`

`colour.difference.delta_E_DIN99(Lab_1, Lab_2, textiles=False)`

Returns the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* formula.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace array 2.

Returns Colour difference ΔE_{DIN99} .

Return type numeric or ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_1 = np.array([60.2574, -34.0099, 36.2677])
>>> Lab_2 = np.array([60.4626, -34.1751, 39.4387])
>>> delta_E_DIN99(Lab_1, Lab_2) # doctest: +ELLIPSIS
1.1772166...
```

Input and Output

- *Image Data*
- *Look Up Table (LUT) Data*
- *CSV Tabular Data*
- *IES TM-27-14 Data*
- *X-Rite Data*

Image Data

colour

<code>read_image(path[, bit_depth, attributes])</code>	Reads given image using <i>OpenImageIO</i> .
<code>write_image(image, path[, bit_depth, attributes])</code>	Writes given image using <i>OpenImageIO</i> .

colour.read_image

`colour.read_image(path, bit_depth='float32', attributes=False)`
 Reads given image using *OpenImageIO*.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – {'float32', 'uint8', 'uint16', 'float16'}, Image bit_depth.
- **attributes** (bool, optional) – Whether to return the image attributes.

Returns Image as a ndarray.

Return type ndarray

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.exr')
>>> image = read_image(path) # doctest: +SKIP
```

colour.write_image

`colour.write_image(image, path, bit_depth='float32', attributes=None)`

Writes given image using *OpenImageIO*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – {'float32', 'uint8', 'uint16', 'float16'}, Image bit_depth.
- **attributes** (array_like, optional) – An array of `colour.io.ImageAttribute_Specification` class instances used to set attributes of the image.

Returns Definition success.

Return type `bool`

Examples

Basic image writing:

```
>>> import os
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.exr')
>>> image = read_image(path) # doctest: +SKIP
>>> path = os.path.join('tests', 'resources', 'CMSTestPattern.tif')
>>> write_image(image, path) # doctest: +SKIP
True
```

Advanced image writing while setting attributes:

```
>>> compression = ImageAttribute_Specification('Compression', 'none')
>>> write_image(image, path, 'uint8', [compression]) # doctest: +SKIP
True
```

Ancillary Objects

`colour.io`

`ImageAttribute_Specification`

Defines the an image specification attribute.

colour.io.ImageAttribute_Specification

class `colour.io.ImageAttribute_Specification`

Defines the an image specification attribute.

Parameters

- **name** (unicode) – Attribute name.
- **value** (object) – Attribute value.
- **type** (TypeDesc, optional) – Attribute type as an *OpenImageIO* TypeDesc class instance.

Returns a new instance of the `colour.ImageAttribute_Specification` class.

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>count</code>	Return number of occurrences of value.
<code>index</code>	Return first index of value.

Attributes

<code>name</code>	Alias for field number 0
<code>type_</code>	Alias for field number 2
<code>value</code>	Alias for field number 1

Look Up Table (LUT) Data

`colour`

<code>LUT1D([table, name, domain, size, comments])</code>	Defines the base class for a 1D <i>LUT</i> .
<code>LUT3x1D([table, name, domain, size, comments])</code>	Defines the base class for a 3x1D <i>LUT</i> .
<code>LUT3D([table, name, domain, size, comments])</code>	Defines the base class for a 3D <i>LUT</i> .
<code>LUTSequence(*args)</code>	Defines the base class for a <i>LUT</i> sequence, i.e.
<code>read_LUT(path[, method])</code>	Reads given <i>LUT</i> file using given method.
<code>write_LUT(LUT, path[, decimals, method])</code>	Writes given <i>LUT</i> to given file using given method.

`colour.LUT1D`

class `colour.LUT1D(table=None, name=None, domain=None, size=10, comments=None)`
Defines the base class for a 1D *LUT*.

Parameters

- **table** (array_like, optional) – Underlying *LUT* table.
- **name** (unicode, optional) – *LUT* name.
- **domain** (unicode, optional) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (int, optional) – Size of the instantiation time default table.
- **comments** (array_like, optional) – Comments to add to the *LUT*.

`is_domain_explicit()`

`linear_table()``apply()``as_LUT()`

Examples

Instantiating a unity LUT with a table with 16 elements:

```
>>> print(LUT1D(size=16))
LUT1D - Unity 16
-----
<BLANKLINE>
Dimensions : 1
Domain      : [ 0.  1.]
Size        : (16,)
```

Instantiating a LUT using a custom table with 16 elements:

```
>>> print(LUT1D(LUT1D.linear_table(16) ** (1 / 2.2))) # doctest: +ELLIPSIS
LUT1D - ...
-----
<BLANKLINE>
Dimensions : 1
Domain      : [ 0.  1.]
Size        : (16,)
```

Instantiating a LUT using a custom table with 16 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> print(LUT1D(
...     spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT1D - My LUT
-----
<BLANKLINE>
Dimensions : 1
Domain      : [-0.1  1.5]
Size        : (16,)
Comment 01  : A first comment.
Comment 02  : A second comment.
```

`__init__(table=None, name=None, domain=None, size=10, comments=None)`
 Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__([table, name, domain, size, comments])` Initialize self.

Continued on next page

Table 169 – continued from previous page

<code>apply(</code> <code>RGB[, interpolator, interpolator_args]</code> <code>)</code>	Applies the <i>LUT</i> to given <i>RGB</i> colourspace array using given method.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.
<code>as_LUT(cls[, force_conversion])</code>	Converts the <i>LUT</i> to given <i>cls</i> class instance.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>is_domain_explicit()</code>	Returns whether the <i>LUT</i> domain is explicit (or implicit).
<code>linear_table([size, domain])</code>	Returns a linear table, the number of output samples <i>n</i> is equal to size.

Attributes

<code>comments</code>	Getter and setter property for the <i>LUT</i> comments.
<code>dimensions</code>	Getter and setter property for the <i>LUT</i> dimensions.
<code>domain</code>	Getter and setter property for the <i>LUT</i> domain.
<code>name</code>	Getter and setter property for the <i>LUT</i> name.
<code>size</code>	Getter property for the <i>LUT</i> size.
<code>table</code>	Getter and setter property for the underlying <i>LUT</i> table.

colour.LUT3x1D

class `colour.LUT3x1D(table=None, name=None, domain=None, size=10, comments=None)`
 Defines the base class for a 3x1D *LUT*.

Parameters

- **table** (array_like, optional) – Underlying *LUT* table.
- **name** (unicode, optional) – *LUT* name.
- **domain** (unicode, optional) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (int, optional) – Size of the instantiation time default table.
- **comments** (array_like, optional) – Comments to add to the *LUT*.

`is_domain_explicit()`

`linear_table()`

`apply()`

`as_LUT()`

Examples

Instantiating a unity *LUT* with a table with 16x3 elements:


```
>>> print(LUT3x1D(size=16))
LUT3x1D - Unity 16
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements:

```
>>> print(LUT3x1D(LUT3x1D.linear_table(16) ** (1 / 2.2)))
... # doctest: +ELLIPSIS
LUT3x1D - ...
-----...
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[ -0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(LUT3x1D(
...     spow(LUT3x1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT3x1D - My LUT
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ -0.1 -0.2 -0.4]
                [ 1.5  3.   6.  ]]
Size        : (16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

__init__(*table=None, name=None, domain=None, size=10, comments=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (<i>[table, name, domain, size, comments]</i>)	Initialize self.
apply (<i>RGB[, interpolator, interpolator_args]</i>)	Applies the <i>LUT</i> to given <i>RGB</i> colourspace array using given method.
arithmetical_operation (<i>a, operation[, in_place]</i>)	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.

Continued on next page

Table 171 – continued from previous page

<code>as_LUT(cls[, force_conversion])</code>	Converts the <i>LUT</i> to given <i>cls</i> class instance.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>is_domain_explicit()</code>	Returns whether the <i>LUT</i> domain is explicit (or implicit).
<code>linear_table([size, domain])</code>	Returns a linear table, the number of output samples <i>n</i> is equal to <code>size * 3</code> or <code>size[0] + size[1] + size[2]</code> .

Attributes

<code>comments</code>	Getter and setter property for the <i>LUT</i> comments.
<code>dimensions</code>	Getter and setter property for the <i>LUT</i> dimensions.
<code>domain</code>	Getter and setter property for the <i>LUT</i> domain.
<code>name</code>	Getter and setter property for the <i>LUT</i> name.
<code>size</code>	Getter property for the <i>LUT</i> size.
<code>table</code>	Getter and setter property for the underlying <i>LUT</i> table.

colour.LUT3D

class `colour.LUT3D(table=None, name=None, domain=None, size=33, comments=None)`

Defines the base class for a 3D *LUT*.

Parameters

- **table** (array_like, optional) – Underlying *LUT* table.
- **name** (unicode, optional) – *LUT* name.
- **domain** (unicode, optional) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (int, optional) – Size of the instantiation time default table.
- **comments** (array_like, optional) – Comments to add to the *LUT*.

`is_domain_explicit()`

`linear_table()`

`apply()`

`as_LUT()`

Examples

Instantiating a unity *LUT* with a table with 16x16x16x3 elements:

```
>>> print(LUT3D(size=16))
LUT3D - Unity 16
-----
<BLANKLINE>
Dimensions : 3
```

(continues on next page)

(continued from previous page)

```
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements:

```
>>> print(LUT3D(LUT3D.linear_table(16) ** (1 / 2.2))) # doctest: +ELLIPSIS
LUT3D - ...
-----...
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[ -0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(LUT3D(
...     spow(LUT3D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT3D - My LUT
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ -0.1 -0.2 -0.4]
                [ 1.5  3.   6.  ]]
Size        : (16, 16, 16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

```
__init__(table=None, name=None, domain=None, size=33, comments=None)
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

<code>__init__([table, name, domain, size, comments])</code>	Initialize self.
<code>apply(RGB[, interpolator, interpolator_args])</code>	Applies the <i>LUT</i> to given <i>RGB</i> colourspace array using given method.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.
<code>as_LUT(cls[, force_conversion])</code>	Converts the <i>LUT</i> to given <i>cls</i> class instance.
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>is_domain_explicit()</code>	Returns whether the <i>LUT</i> domain is explicit (or implicit).

Continued on next page

Table 173 – continued from previous page

<code>linear_table([size, domain])</code>	Returns a linear table, the number of output samples n is equal to <code>size**3 * 3</code> or <code>size[0] * size[1] * size[2] * 3</code> .
---	---

Attributes

<code>comments</code>	Getter and setter property for the <i>LUT</i> comments.
<code>dimensions</code>	Getter and setter property for the <i>LUT</i> dimensions.
<code>domain</code>	Getter and setter property for the <i>LUT</i> domain.
<code>name</code>	Getter and setter property for the <i>LUT</i> name.
<code>size</code>	Getter property for the <i>LUT</i> size.
<code>table</code>	Getter and setter property for the underlying <i>LUT</i> table.

colour.LUTSequence

class `colour.LUTSequence(*args)`

Defines the base class for a *LUT* sequence, i.e. a series of *LUT*s.

The `colour.LUTSequence` class can be used to model series of *LUT*s such as when a shaper *LUT* is combined with a 3D *LUT*.

Other Parameters `*args` (*list, optional*) – Sequence of `colour.LUT1D`, `colour.LUT3x1D`, `colour.LUT3D` or `colour.io.lut.l.AbstractLUTSequenceOperator` class instances.

sequence

`__getitem__()`

`__setitem__()`

`__delitem__()`

`__len__()`

`__str__()`

`__repr__()`

`__eq__()`

`__ne__()`

`insert()`

`apply()`

`copy()`

Examples

```
>>> LUT_1 = LUT1D()
>>> LUT_2 = LUT3D(size=3)
>>> LUT_3 = LUT3x1D()
>>> print(LUTSequence(LUT_1, LUT_2, LUT_3))
```

(continues on next page)

(continued from previous page)

```

LUT Sequence
-----
<BLANKLINE>
Overview
<BLANKLINE>
    LUT1D ---> LUT3D ---> LUT3x1D
<BLANKLINE>
Operations
<BLANKLINE>
    LUT1D - Unity 10
    -----
<BLANKLINE>
    Dimensions : 1
    Domain      : [ 0.  1.]
    Size        : (10,)
<BLANKLINE>
    LUT3D - Unity 3
    -----
<BLANKLINE>
    Dimensions : 3
    Domain      : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size        : (3, 3, 3)
<BLANKLINE>
    LUT3x1D - Unity 10
    -----
<BLANKLINE>
    Dimensions : 2
    Domain      : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size        : (10, 3)

```

__init__(*args)

Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__</code> (*args)	Initialize self.
<code>append</code> (value)	S.append(value) – append value to the end of the sequence
<code>apply</code> (RGB[, interpolator_1D, ...])	Applies the <i>LUT</i> sequence sequentially to given <i>RGB</i> colourspace array.
<code>clear</code> ()	
<code>copy</code> ()	Returns a copy of the <i>LUT</i> sequence.
<code>count</code> (value)	
<code>extend</code> (values)	S.extend(iterable) – extend sequence by appending elements from the iterable
<code>index</code> (value, [start, [stop]])	Raises <code>ValueError</code> if the value is not present.
<code>insert</code> (index, LUT)	Inserts given <i>LUT</i> at given index into the <i>LUT</i> sequence.
<code>pop</code> ([index])	Raise <code>IndexError</code> if list is empty or index is out of range.

Continued on next page

Table 175 – continued from previous page

<code>remove(value)</code>	<code>S.remove(value)</code> – remove first occurrence of value.
<code>reverse()</code>	<code>S.reverse()</code> – reverse <i>IN PLACE</i>

Attributes

<code>sequence</code>	Getter and setter property for the underlying <i>LUT</i> sequence.
-----------------------	--

colour.read_LUT

`colour.read_LUT(path, method=None, **kwargs)`

Reads given *LUT* file using given method.

Parameters

- **path** (unicode) – *LUT* path.
- **method** (unicode, optional) – {`None`, `'Cinespace'`, `'Iridas Cube'`, `'Resolve Cube'`, `'Sony SPI1D'`, `'Sony SPI3D'`}, Reading method, if `None`, the method will be auto-detected according to extension.

Returns `LUT1D`, `LUT3x1D` or `LUT3D` class instance.

Return type `LUT1D` or `LUT3x1D` or `LUT3D`

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Reading a 3x1D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'ACES_Proxy_10_to_ACES.cube')
>>> print(read_LUT(path))
LUT3x1D - ACES Proxy 10 to ACES
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
               [ 1.  1.  1.]]
Size        : (32, 3)
```

Reading a 1D *Sony .sp1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_sp1d',
...     'oetf_reverse_sRGB_1D.sp1d')
>>> print(read_LUT(path))
LUT1D - oetf reverse sRGB 1D
```

(continues on next page)

(continued from previous page)

```

-----
<BLANKLINE>
Dimensions : 1
Domain      : [-0.1  1.5]
Size        : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.oetf_reverse_sRGB".

```

Reading a 3D Sony *.spi3d* LUT:

```

>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'ColourCorrect.spi3d')
>>> print(read_LUT(path))
LUT3D - ColourCorrect
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
               [ 1.  1.  1.]]
Size        : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.

```

colour.write_LUT

`colour.write_LUT(LUT, path, decimals=7, method=None, **kwargs)`

Writes given *LUT* to given file using given method.

Parameters

- **LUT** (`LUT1D` or `LUT3x1D` or `LUT3D`) – `LUT1D`, `LUT3x1D` or `LUT3D` class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (`int`, optional) – Formatting decimals.
- **method** (unicode, optional) – {`None`, `‘Cinespace’`, `‘Iridas Cube’`, `‘Resolve Cube’`, `‘Sony SPI1D’`, `‘Sony SPI3D’`}, Writing method, if `None`, the method will be auto-detected according to extension.

Returns Definition success.

Return type `bool`

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Writing a 3x1D *Iridas .cube* LUT:

```
>>> import numpy as np
>>> from colour.algebra import spow
>>> domain = np.array([[ -0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.cube') # doctest: +SKIP
```

Writing a 1D Sony *.spi1d* LUT:

```
>>> domain = np.array([ -0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.spi1d') # doctest: +SKIP
```

Writing a 3D Sony *.spi3d* LUT:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     'My LUT',
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.cube') # doctest: +SKIP
```

Ancillary Objects

colour.io

AbstractLUTSequenceOperator	Defines the base class for <i>LUT</i> sequence operators.
LUT_to_LUT(LUT, cls[, force_conversion])	Converts given <i>LUT</i> to given <i>cls</i> class instance.
read_LUT_Cinespace(path)	Reads given <i>Cinespace</i> <i>.csp</i> <i>LUT</i> file.
write_LUT_Cinespace(LUT, path[, decimals])	Writes given <i>LUT</i> to given <i>Cinespace</i> <i>.csp</i> <i>LUT</i> file.
read_LUT_IridasCube(path)	Reads given <i>Iridas</i> <i>.cube</i> <i>LUT</i> file.
write_LUT_IridasCube(LUT, path[, decimals])	Writes given <i>LUT</i> to given <i>Iridas</i> <i>.cube</i> <i>LUT</i> file.
read_LUT_SonySPI1D(path)	Reads given Sony <i>.spi1d</i> <i>LUT</i> file.
write_LUT_SonySPI1D(LUT, path[, decimals])	Writes given <i>LUT</i> to given Sony <i>.spi1d</i> <i>LUT</i> file.
read_LUT_SonySPI3D(path)	Reads given Sony <i>.spi3d</i> <i>LUT</i> file.
write_LUT_SonySPI3D(LUT, path[, decimals])	Writes given <i>LUT</i> to given Sony <i>.spi3d</i> <i>LUT</i> file.

colour.io.AbstractLUTSequenceOperator

class colour.io.**AbstractLUTSequenceOperator**

Defines the base class for *LUT* sequence operators.

This is an ABCMeta abstract class that must be inherited by sub-classes.

apply()

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

<code>apply(RGB, *args)</code>	Applies the <i>LUT</i> sequence operator to given <i>RGB</i> colourspace array.
--------------------------------	---

colour.io.LUT_to_LUT

`colour.io.LUT_to_LUT(LUT, cls, force_conversion=False, **kwargs)`
 Converts given *LUT* to given *cls* class instance.

Parameters

- **cls** (*LUT1D* or *LUT3x1D* or *LUT3D*) – *LUT* class instance.
- **force_conversion** (*bool*, optional) – Whether to force the conversion as it might be destructive.

Other Parameters

- **interpolator** (*object*, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like*, optional) – Arguments to use when instantiating the interpolating function.
- **size** (*int*, optional) – Expected table size in case of an upcast to or a downcast from a *LUT3D* class instance.
- **channel_weights** (*array_like*, optional) – Channel weights in case of a downcast from a *LUT3x1D* or *LUT3D* class instance.

Returns Converted *LUT* class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

Warning: Some conversions are destructive and raise a *ValueError* exception by default.

Raises *ValueError* – If the conversion is destructive.

Examples

```
>>> print(LUT_to_LUT(LUT1D(), LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 1D to 3D
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (33, 33, 33, 3)
>>> print(LUT_to_LUT(LUT3x1D(), LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
-----
<BLANKLINE>
Dimensions : 1
Domain      : [ 0.  1.]
```

(continues on next page)

(continued from previous page)

```
Size      : (10,)
>>> print(LUT_to_LUT(LUT3D(), LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-----
<BLANKLINE>
Dimensions : 1
Domain      : [ 0.  1.]
Size        : (10,)
```

colour.io.read_LUT_Cinespace

colour.io.read_LUT_Cinespace(path)

Reads given *Cinespace* .csp LUT file.

Parameters path (unicode) – LUT path.

Returns LUT3x1D or LUT3D or LUTSequence class instance.

Return type *LUT3x1D* or *LUT3D* or *LUTSequence*

References

[[RisingSResearch](#)]

Examples

Reading a 3x1D *Cinespace* .csp LUT:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'cinespace',
...     'ACES_Proxy_10_to_ACES.csp')
>>> print(read_LUT_Cinespace(path))
LUT3x1D - ACES Proxy 10 to ACES
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
               [ 1.  1.  1.]]
Size        : (32, 3)
```

Reading a 3D *Cinespace* .csp LUT:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'cinespace',
...     'ColourCorrect.csp')
>>> print(read_LUT_Cinespace(path))
LUT3D - Generated by Foundry::LUT
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
               [ 1.  1.  1.]]
Size        : (4, 4, 4, 3)
```

colour.io.write_LUT_Cinespace

colour.io.write_LUT_Cinespace(*LUT*, *path*, *decimals*=7)

Writes given *LUT* to given *Cinespace* .csp *LUT* file.

Parameters

- **LUT** ([LUT1D](#) or [LUT3x1D](#) or [LUT3D](#) or [LUTSequence](#)) – LUT1D, LUT3x1D or LUT3D or LUTSequence class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type bool

References

[[RisingSResearch](#)]

Examples

Writing a 3x1D *Cinespace* .csp *LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_Cinespace(LUT, 'My_LUT.cube') # doctest: +SKIP
```

Writing a 3D *Cinespace* .csp *LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_Cinespace(LUT, 'My_LUT.cube') # doctest: +SKIP
```

colour.io.read_LUT_IridasCube

colour.io.read_LUT_IridasCube(*path*)

Reads given *Iridas* .cube *LUT* file.

Parameters *path* (unicode) – *LUT* path.

Returns LUT3x1D or LUT3D class instance.

Return type [LUT3x1D](#) or LUT3d

References

[AdobeSystems13c]

Examples

Reading a 3x1D *Iridas* .cube LUT:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'ACES_Proxy_10_to_ACES.cube')
>>> print(read_LUT_IridasCube(path))
LUT3x1D - ACES Proxy 10 to ACES
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (32, 3)
```

Reading a 3D *Iridas* .cube LUT:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'ColourCorrect.cube')
>>> print(read_LUT_IridasCube(path))
LUT3D - Generated by Foundry::LUT
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (4, 4, 4, 3)
```

Reading a 3D *Iridas* .cube LUT with comments:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'Demo.cube')
>>> print(read_LUT_IridasCube(path))
LUT3x1D - Demo
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[ 0.  0.  0.]
                [ 1.  2.  3.]]
Size        : (3, 3)
Comment 01 : Comments can go anywhere
```

colour.io.write_LUT_IridasCube

colour.io.write_LUT_IridasCube(LUT, path, decimals=7)

Writes given LUT to given *Iridas* .cube LUT file.

Parameters

- **LUT** (`LUT3x1D` or `LUT3d` or `LUTSequence`) – `LUT3x1D`, `LUT3D` or `LUTSequence` class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type `bool`

Warning:

- If a `LUTSequence` class instance is passed as *LUT*, the first *LUT* in the *LUT* sequence will be used.

References

[AdobeSystems13c]

Examples

Writing a 3x1D *Iridas* .cube *LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_IridasCube(LUT, 'My_LUT.cube') # doctest: +SKIP
```

Writing a 3D *Iridas* .cube *LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_IridasCube(LUT, 'My_LUT.cube') # doctest: +SKIP
```

colour.io.read_LUT_SonySPI1D

`colour.io.read_LUT_SonySPI1D(path)`

Reads given *Sony* .spi1d *LUT* file.

Parameters **path** (unicode) – *LUT* path.

Returns `LUT1D` or `LUT3x1D` class instance.

Return type `LUT1D` or `LUT3x1D`

Examples

Reading a 1D *Sony .spi1d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'oetf_reverse_sRGB_1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT1D - oetf reverse sRGB 1D
-----
<BLANKLINE>
Dimensions : 1
Domain      : [-0.1  1.5]
Size        : (16,)
Comment 01  : Generated by "Colour 0.3.11".
Comment 02  : "colour.models.oetf_reverse_sRGB".
```

Reading a 3x1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'oetf_reverse_sRGB_3x1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT3x1D - oetf reverse sRGB 3x1D
-----
<BLANKLINE>
Dimensions : 2
Domain      : [[-0.1 -0.1 -0.1]
               [ 1.5  1.5  1.5]]
Size        : (16, 3)
Comment 01  : Generated by "Colour 0.3.11".
Comment 02  : "colour.models.oetf_reverse_sRGB".
```

colour.io.write_LUT_SonySPI1D

`colour.io.write_LUT_SonySPI1D(LUT, path, decimals=7)`

Writes given *LUT* to given *Sony .spi1d LUT* file.

Parameters

- **LUT** (`LUT1D` or `LUT2d`) – *LUT1D*, *LUT3x1D* or *LUTSequence* class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type `bool`

Warning:

- If a *LUTSequence* class instance is passed as *LUT*, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 1D *Sony .spi1d LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI1D(LUT, 'My_LUT.cube') # doctest: +SKIP
```

Writing a 3x1D *Sony .spi1d LUT*:

```
>>> domain = np.array([[-0.1, -0.1, -0.1], [1.5, 1.5, 1.5]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI1D(LUT, 'My_LUT.cube') # doctest: +SKIP
```

colour.io.read_LUT_SonySPI3D

`colour.io.read_LUT_SonySPI3D(path)`

Reads given *Sony .spi3d LUT* file.

Parameters `path` (unicode) – *LUT* path.

Returns `LUT3D` or `LUT3x1D` class instance.

Return type `LUT3D` or `LUT3x1D`

Examples

Reading a 3D *Sony .spi3d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'ColourCorrect.spi3d')
>>> print(read_LUT_SonySPI3D(path))
LUT3D - ColourCorrect
-----
<BLANKLINE>
Dimensions : 3
Domain      : [[ 0.  0.  0.]
                [ 1.  1.  1.]]
Size        : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
```

colour.io.write_LUT_SonySPI3D

`colour.io.write_LUT_SonySPI3D(LUT, path, decimals=7)`

Writes given *LUT* to given Sony *.spi3d* *LUT* file.

Parameters

- **LUT** (`LUT3D`) – *LUT3D* or *LUTSequence* class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type `bool`

Warning:

- If a *LUTSequence* class instance is passed as *LUT*, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 3D Sony *.spi3d* *LUT*:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     'My LUT',
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI3D(LUT, 'My_LUT.cube') # doctest: +SKIP
```

CSV Tabular Data

colour

<code>read_sds_from_csv_file(path[, delimiter, ...])</code>	Reads the spectral data from given CSV file and return its content as an <i>OrderedDict</i> of <code>colour.SpectralDistribution</code> classes.
<code>read_spectral_data_from_csv_file(path[, ...])</code>	Reads the spectral data from given CSV file in the following form:
<code>write_sds_to_csv_file(sds, path[, ...])</code>	Writes the given spectral distributions to given CSV file.

colour.read_sds_from_csv_file

`colour.read_sds_from_csv_file(path, delimiter=',', fields=None, default=0)`

Reads the spectral data from given CSV file and return its content as an *OrderedDict* of `colour.SpectralDistribution` classes.

Parameters

- **path** (unicode) – Absolute CSV file path.
- **delimiter** (unicode, optional) – CSV file content delimiter.
- **fields** (array_like, optional) – CSV file spectral data fields names. If no value is provided the first line of the file will be used for as spectral data fields names.
- **default** (numeric) – Default value for fields row with missing value.

Returns `colour.SpectralDistribution` classes of given CSV file.

Return type `OrderedDict`

Examples

```
>>> from colour.utilities import numpy_print_options
>>> import os
>>> csv_file = os.path.join(os.path.dirname(__file__), 'tests',
...                         'resources', 'colorchecker_n_ohita.csv')
>>> sds = read_sds_from_csv_file(csv_file)
>>> print(tuple(sds.keys()))
('1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18',
 '19', '20', '21', '22', '23', '24')
>>> with numpy_print_options(suppress=True):
...     sds['1'] # doctest: +ELLIPSIS
SpectralDistribution([[ 380.    ,    0.048],
                    [ 385.    ,    0.051],
                    [ 390.    ,    0.055],
                    [ 395.    ,    0.06 ],
                    [ 400.    ,    0.065],
                    [ 405.    ,    0.068],
                    [ 410.    ,    0.068],
                    [ 415.    ,    0.067],
                    [ 420.    ,    0.064],
                    [ 425.    ,    0.062],
                    [ 430.    ,    0.059],
                    [ 435.    ,    0.057],
                    [ 440.    ,    0.055],
                    [ 445.    ,    0.054],
                    [ 450.    ,    0.053],
                    [ 455.    ,    0.053],
                    [ 460.    ,    0.052],
                    [ 465.    ,    0.052],
                    [ 470.    ,    0.052],
                    [ 475.    ,    0.053],
                    [ 480.    ,    0.054],
                    [ 485.    ,    0.055],
                    [ 490.    ,    0.057],
                    [ 495.    ,    0.059],
                    [ 500.    ,    0.061],
                    [ 505.    ,    0.062],
                    [ 510.    ,    0.065],
                    [ 515.    ,    0.067],
                    [ 520.    ,    0.07 ],
                    [ 525.    ,    0.072],
                    [ 530.    ,    0.074],
                    [ 535.    ,    0.075],
                    [ 540.    ,    0.076],
```

(continues on next page)

(continued from previous page)

```
[ 545. , 0.078],
[ 550. , 0.079],
[ 555. , 0.082],
[ 560. , 0.087],
[ 565. , 0.092],
[ 570. , 0.1 ],
[ 575. , 0.107],
[ 580. , 0.115],
[ 585. , 0.122],
[ 590. , 0.129],
[ 595. , 0.134],
[ 600. , 0.138],
[ 605. , 0.142],
[ 610. , 0.146],
[ 615. , 0.15 ],
[ 620. , 0.154],
[ 625. , 0.158],
[ 630. , 0.163],
[ 635. , 0.167],
[ 640. , 0.173],
[ 645. , 0.18 ],
[ 650. , 0.188],
[ 655. , 0.196],
[ 660. , 0.204],
[ 665. , 0.213],
[ 670. , 0.222],
[ 675. , 0.231],
[ 680. , 0.242],
[ 685. , 0.251],
[ 690. , 0.261],
[ 695. , 0.271],
[ 700. , 0.282],
[ 705. , 0.294],
[ 710. , 0.305],
[ 715. , 0.318],
[ 720. , 0.334],
[ 725. , 0.354],
[ 730. , 0.372],
[ 735. , 0.392],
[ 740. , 0.409],
[ 745. , 0.42 ],
[ 750. , 0.436],
[ 755. , 0.45 ],
[ 760. , 0.462],
[ 765. , 0.465],
[ 770. , 0.448],
[ 775. , 0.432],
[ 780. , 0.421]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})
```

colour.read_spectral_data_from_csv_file

`colour.read_spectral_data_from_csv_file(path, delimiter=',', fields=None, default=0)`

Reads the spectral data from given CSV file in the following form:

```
390, 4.15003E-04, 3.68349E-04, 9.54729E-03 395, 1.05192E-03, 9.58658E-04, 2.38250E-02 400,
2.40836E-03, 2.26991E-03, 5.66498E-02 ... 830, 9.74306E-07, 9.53411E-08, 0.00000
```

and returns it as an *OrderedDict* of *dict* as follows:

```
OrderedDict([(‘field’, {‘wavelength’: ‘value’, ..., ‘wavelength’: ‘value’}), ..., (‘field’, {‘wavelength’:
‘value’, ..., ‘wavelength’: ‘value’})])
```

Parameters

- **path** (unicode) – Absolute CSV file path.
- **delimiter** (unicode, optional) – CSV file content delimiter.
- **fields** (array_like, optional) – CSV file spectral data fields names. If no value is provided the first line of the file will be used as spectral data fields names.
- **default** (numeric, optional) – Default value for fields row with missing value.

Returns CSV file content.

Return type *OrderedDict*

Raises *RuntimeError* – If the CSV spectral data file doesn’t define the appropriate fields.

Notes

- A CSV spectral data file should define at least define two fields: one for the wavelengths and one for the associated values of one spectral distribution.
- If no value is provided for the fields names, the first line of the file will be used as spectral data fields names.

Examples

```
>>> import os
>>> from pprint import pprint
>>> csv_file = os.path.join(os.path.dirname(__file__), 'tests',
...                         'resources', 'colorchecker_n_ohda.csv')
>>> sds_data = read_spectral_data_from_csv_file(csv_file)
>>> pprint(list(sds_data.keys()))
['1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9',
 '10',
 '11',
 '12',
 '13',
```

(continues on next page)

(continued from previous page)

```
'14',  
'15',  
'16',  
'17',  
'18',  
'19',  
'20',  
'21',  
'22',  
'23',  
'24']
```

`colour.write_sds_to_csv_file`

`colour.write_sds_to_csv_file(sds, path, delimiter=',', fields=None)`

Writes the given spectral distributions to given CSV file.

Parameters

- **sds** (`dict`) – Spectral distributions to write.
- **path** (`unicode`) – Absolute CSV file path.
- **delimiter** (`unicode`, optional) – CSV file content delimiter.
- **fields** (`array_like`, optional) – CSV file spectral data fields names. If no value is provided the order of fields will be the one defined by the sorted spectral distributions *dict*.

Returns Definition success.

Return type `bool`

Raises `RuntimeError` – If the given spectral distributions have different shapes.

IES TM-27-14 Data

`colour`

<code>SpectralDistribution_IESTM2714([path, ...])</code>	Defines a <i>IES TM-27-14</i> spectral distribution.
--	--

`colour.SpectralDistribution_IESTM2714`

```
class colour.SpectralDistribution_IESTM2714(path=None, header=None, spectral_quantity=None,  
                                           reflection_geometry=None, transmission_geometry=None,  
                                           bandwidth_FWHM=None, bandwidth_corrected=None)
```

Defines a *IES TM-27-14* spectral distribution.

This class can read and write *IES TM-27-14* spectral data XML files.

Parameters

- **path** (`unicode`, optional) – Spectral data XML file path.

- **header** (IES_TM2714_Header, optional) – *IES TM-27-14* spectral distribution header.
- **spectral_quantity** (unicode, optional) – {‘flux’, ‘absorptance’, ‘transmittance’, ‘reflectance’, ‘intensity’, ‘irradiance’, ‘radiance’, ‘exitance’, ‘R-Factor’, ‘T-Factor’, ‘relative’, ‘other’}, Quantity of measurement for each element of the spectral data.
- **reflection_geometry** (unicode, optional) – {‘di:8’, ‘de:8’, ‘8:di’, ‘8:de’, ‘d:d’, ‘d:0’, ‘45a:0’, ‘45c:0’, ‘0:45a’, ‘45x:0’, ‘0:45x’, ‘other’}, Spectral reflectance factors geometric conditions.
- **transmission_geometry** (unicode, optional) – {‘0:0’, ‘di:0’, ‘de:0’, ‘0:di’, ‘0:de’, ‘d:d’, ‘other’}, Spectral transmittance factors geometric conditions.
- **bandwidth_FWHM** (numeric, optional) – Spectroradiometer full-width half-maximum bandwidth in nanometers.
- **bandwidth_corrected** (bool, optional) – Specifies if bandwidth correction has been applied to the measured data.

Notes

Reflection Geometry

- di:8: Diffuse / eight-degree, specular component included.
- de:8: Diffuse / eight-degree, specular component excluded.
- 8:di: Eight-degree / diffuse, specular component included.
- 8:de: Eight-degree / diffuse, specular component excluded.
- d:d: Diffuse / diffuse.
- d:0: Alternative diffuse.
- 45a:0: Forty-five degree annular / normal.
- 45c:0: Forty-five degree circumferential / normal.
- 0:45a: Normal / forty-five degree annular.
- 45x:0: Forty-five degree directional / normal.
- 0:45x: Normal / forty-five degree directional.
- other: User-specified in comments.

Transmission Geometry

- 0:0: Normal / normal.
- di:0: Diffuse / normal, regular component included.
- de:0: Diffuse / normal, regular component excluded.
- 0:di: Normal / diffuse, regular component included.
- 0:de: Normal / diffuse, regular component excluded.
- d:d: Diffuse / diffuse.
- other: User-specified in comments.

mapping

path

header
spectral_quantity
reflection_geometry
transmission_geometry
bandwidth_FWHM
bandwidth_corrected
read()
write()

References

[IESCCommitteeTM2714WGroup14]

Examples

```

>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx'))
>>> sd.read()
True
>>> sd.header.manufacturer
'Unknown'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> sd[501.7] # doctest: +ELLIPSIS
0.0950000...

```

__init__(*path=None, header=None, spectral_quantity=None, reflection_geometry=None, transmission_geometry=None, bandwidth_FWHM=None, bandwidth_corrected=None*)
 Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__([path, header, spectral_quantity, ...])</code>	Initialize self.
<code>align(shape[, interpolator, ...])</code>	Aligns the spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.
<code>arithmetical_operation(a, operation[, in_place])</code>	Performs given arithmetical operation with <i>a</i> operand, the operation can be either performed on a copy or in-place.
<code>clone()</code>	
<code>copy()</code>	Returns a copy of the sub-class instance.
<code>domain_distance(a)</code>	Returns the euclidean distance between given array and independent domain <i>x</i> closest element.

Continued on next page

Table 181 – continued from previous page

<code>extrapolate(shape[, extrapolator, ...])</code>	Extrapolates the spectral distribution in-place according to <i>CIE 15:2004</i> and <i>CIE 167:2005</i> recommendations or given extrapolation arguments.
<code>fill_nan([method, default])</code>	Fill NaNs in independent domain x variable and corresponding range y variable using given method.
<code>get()</code>	
<code>interpolate(shape[, interpolator, ...])</code>	Interpolates the spectral distribution in-place according to <i>CIE 167:2005</i> recommendation or given interpolation arguments.
<code>is_uniform()</code>	Returns if independent domain x variable is uniform.
<code>normalise([factor])</code>	Normalises the spectral distribution using given normalization factor.
<code>read()</code>	Reads and parses the spectral data XML file path.
<code>signal_unpack_data([data, domain, dtype])</code>	Unpack given data for continuous signal instantiation.
<code>to_series()</code>	Converts the continuous signal to a <i>Pandas</i> Series class instance.
<code>trim(shape)</code>	Trims the spectral distribution wavelengths to given spectral shape.
<code>trim_wavelengths(shape)</code>	
<code>write()</code>	Write the spectral distribution spectral data to XML file path.
<code>zeros()</code>	

Attributes

<code>bandwidth_FWHM</code>	Getter and setter property for the full-width half-maximum bandwidth.
<code>bandwidth_corrected</code>	Getter and setter property for whether bandwidth correction has been applied to the measured data.
<code>data</code>	
<code>domain</code>	Getter and setter property for the continuous signal independent domain x variable.
<code>dtype</code>	Getter and setter property for the continuous signal dtype.
<code>extrapolator</code>	Getter and setter property for the continuous signal extrapolator type.
<code>extrapolator_args</code>	Getter and setter property for the continuous signal extrapolator instantiation time arguments.
<code>function</code>	Getter and setter property for the continuous signal callable.
<code>header</code>	Getter and setter property for the header.
<code>interpolator</code>	Getter and setter property for the continuous signal interpolator type.
<code>interpolator_args</code>	Getter and setter property for the continuous signal interpolator instantiation time arguments.

Continued on next page

Table 182 – continued from previous page

items	
mapping	Getter and setter property for the mapping structure.
name	Getter and setter property for the abstract continuous function name.
path	Getter and setter property for the path.
range	Getter and setter property for the continuous signal corresponding range y variable.
reflection_geometry	Getter and setter property for the reflection geometry.
shape	Getter and setter property for the spectral distribution shape.
spectral_quantity	Getter and setter property for the spectral quantity.
strict_name	Getter and setter property for the spectral distribution strict name.
title	
transmission_geometry	Getter and setter property for the transmission geometry.
values	Getter and setter property for the spectral distribution values.
wavelengths	Getter and setter property for the spectral distribution wavelengths λ_n .

X-Rite Data

colour

<code>read_sds_from_xrite_file(path)</code>	Reads the spectral data from given <i>X-Rite</i> file and returns it as an <i>OrderedDict</i> of <code>colour.SpectralDistribution</code> classes.
---	--

colour.read_sds_from_xrite_file

`colour.read_sds_from_xrite_file(path)`

Reads the spectral data from given *X-Rite* file and returns it as an *OrderedDict* of `colour.SpectralDistribution` classes.

Parameters `path` (unicode) – Absolute *X-Rite* file path.

Returns `colour.SpectralDistribution` classes of given *X-Rite* file.

Return type `OrderedDict`

Notes

- This parser is minimalistic and absolutely not bullet proof.

Examples

```
>>> import os
>>> from pprint import pprint
>>> xrite_file = os.path.join(os.path.dirname(__file__), 'tests',
...                           'resources',
...                           'xrite_digital_colour_checker.txt')
>>> sds_data = read_sds_from_xrite_file(xrite_file)
>>> pprint(list(sds_data.keys())) # doctest: +SKIP
['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']
```

Colour Models

- *Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates*
- *CIE $L^*a^*b^*$ Colourspace*
- *CIE $L^*u^*v^*$ Colourspace*
- *CIE 1960 UCS Colourspace*
- *CIE 1964 $U^*V^*W^*$ Colourspace*
- *Hunter L,a,b Colour Scale*
- *Hunter Rd,a,b Colour Scale*
- *DIN99 Colourspace*
- *CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)*
- *CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)*
- *IPT Colourspace*
- *hdr-CIELAB Colourspace*
- *hdr-IPT Colourspace*
- *OSA UCS Colourspace*
- *$JzAzBz$ Colourspace*
- *RGB Colourspace and Transformations*
 - *RGB Colourspace Derivation*
 - *RGB Colourspaces*
 - *Colour Component Transfer Functions*
 - *Opto-Electronic Transfer Functions*
 - *Electro-Optical Transfer Functions*
 - *Opto-Optical Transfer Functions*
 - *Log Encoding and Decoding Curves*
 - *ACES Spectral Conversion*
 - *Colour Encodings*

- * *Y'CbCr Colour Encoding*
- * *YCoCg Colour Encoding*
- * *IC_TC_P Colour Encoding*
- *RGB Representations*
 - * *Prismatic Colourspace*
 - * *HSV Colourspace*
 - * *HSL Colourspace*
 - * *CMY Colourspace*
- *Pointer's Gamut*

Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates

colour

<code>XYZ_to_xyY(XYZ[, illuminant])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>CIE xyY</i> colourspace and reference <i>illuminant</i> .
<code>xyY_to_XYZ(xyY)</code>	Converts from <i>CIE xyY</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>XYZ_to_xy(XYZ[, illuminant])</code>	Returns the <i>xy</i> chromaticity coordinates from given <i>CIE XYZ</i> tristimulus values.
<code>xy_to_XYZ(xy)</code>	Returns the <i>CIE XYZ</i> tristimulus values from given <i>xy</i> chromaticity coordinates.
<code>xyY_to_xy(xyY)</code>	Converts from <i>CIE xyY</i> colourspace to <i>xy</i> chromaticity coordinates.
<code>xy_to_xyY(xy[, Y])</code>	Converts from <i>xy</i> chromaticity coordinates to <i>CIE xyY</i> colourspace by extending the array last dimension with given <i>Y</i> <i>luminance</i> .

colour.XYZ_to_xyY

colour.XYZ_to_xyY(XYZ, illuminant=array([0.3127, 0.329]))

Converts from *CIE XYZ* tristimulus values to *CIE xyY* colourspace and reference *illuminant*.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant* chromaticity coordinates.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Lin03b], [Wik05a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xyY(XYZ) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794..., 0.1219722...])
```

colour.xyY_to_XYZ

colour.xyY_to_XYZ(xyY)

Converts from *CIE xyY* colour space to *CIE XYZ* tristimulus values.

Parameters xyY (array_like) – *CIE xyY* colour space array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Lin09d], [Wik05a]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_XYZ(xyY) # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
```

colour.XYZ_to_xy

colour.XYZ_to_xy(XYZ, illuminant=array([0.3127, 0.329]))

Returns the *xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xy(XYZ) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794...])
```

colour.xy_to_XYZ

colour.xy_to_XYZ(xy)

Returns the *CIE XYZ* tristimulus values from given *xy* chromaticity coordinates.

Parameters **xy** (array_like) – *xy* chromaticity coordinates.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_XYZ(xy) # doctest: +ELLIPSIS
array([ 1.6933366..., 1.        , 0.4211574...])
```

colour.xyY_to_xy

colour.**xyY_to_xy**(xyY)

Converts from *CIE xyY* colourspace to *xy* chromaticity coordinates.

xyY argument with last dimension being equal to 2 will be assumed to be a *xy* chromaticity coordinates argument and will be returned directly by the definition.

Parameters *xyY* (array_like) – *CIE xyY* colourspace array or *xy* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>xyY</i>	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_xy(xyY) # doctest: +ELLIPSIS
array([ 0.54369557..., 0.32107944...])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xyY_to_xy(xy) # doctest: +ELLIPSIS
array([ 0.54369557..., 0.32107944...])
```

colour.xy_to_xyY

colour.**xy_to_xyY**(xy, Y=1)

Converts from *xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given *Y* luminance.

xy argument with last dimension being equal to 3 will be assumed to be a *CIE xyY* colourspace array argument and will be returned directly by the definition.

Parameters

- **xy** (array_like) – *xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (numeric, optional) – Optional *Y luminance* value used to construct the *CIE xyY* colourspace array, the default *Y luminance* value is 1.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

- This definition is a convenient object provided to implement support of illuminant argument *luminance* value in various `colour.models` package objects such as `colour.Lab_to_XYZ()` or `colour.Luv_to_XYZ()`.

References

[Wik05a]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794..., 1.          ])
>>> xy = np.array([0.54369557, 0.32107944, 1.00000000])
>>> xy_to_xyY(xy) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794..., 1.          ])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy, 100) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794..., 100.         ])
```

CIE L*a*b* Colourspace

colour

<code>XYZ_to_Lab(XYZ[, illuminant])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>CIE L*a*b*</i> colourspace.
<code>Lab_to_XYZ(Lab[, illuminant])</code>	Converts from <i>CIE L*a*b*</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>Lab_to_LCHab(Lab)</code>	Converts from <i>CIE L*a*b*</i> colourspace to <i>CIE L*C*Hab</i> colourspace.

Continued on next page

Table 185 – continued from previous page

<code>LCHab_to_Lab(LCHab)</code>	Converts from <i>CIE L*C*Hab</i> colourspace to <i>CIE L*a*b*</i> colourspace.
----------------------------------	--

`colour.XYZ_to_Lab`

`colour.XYZ_to_Lab(XYZ, illuminant=array([ 0.3127, 0.329 ]))`

Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804f]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Lab(XYZ) # doctest: +ELLIPSIS
array([ 41.5278752..., 52.6385830..., 26.9231792...])
```

`colour.Lab_to_XYZ`

`colour.Lab_to_XYZ(Lab, illuminant=array([ 0.3127, 0.329 ]))`

Converts from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (array_like) – *CIE L*a*b** colourspace array.

- **illuminant** (array_like, optional) – Reference *illuminant* *xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f]

Examples

```
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_XYZ(Lab) # doctest: +ELLIPSIS
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Lab_to_LCHab

colour.Lab_to_LCHab(Lab)

Converts from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.

Parameters Lab (array_like) – *CIE L*a*b** colourspace array.

Returns *CIE L*C*Hab* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] ab : [0, 360]	L : [0, 1] C : [0, 1] ab : [0, 1]

References

[CIET14804f]

Examples

```
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_LCHab(Lab) # doctest: +ELLIPSIS
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.LCHab_to_Lab

`colour.LCHab_to_Lab(LCHab)`

Converts from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace.

Parameters **LCHab** (array_like) – *CIE L*C*Hab* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] ab : [0, 360]	L : [0, 1] C : [0, 1] ab : [0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804f]

Examples

```
>>> LCHab = np.array([41.52787529, 59.12425901, 27.08848784])
>>> LCHab_to_Lab(LCHab) # doctest: +ELLIPSIS
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE $L^*u^*v^*$ Colourspace

colour

<code>XYZ_to_Luv(XYZ[, illuminant])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>CIE $L^*u^*v^*$</i> colourspace.
<code>Luv_to_XYZ(Luv[, illuminant])</code>	Converts from <i>CIE $L^*u^*v^*$</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>Luv_to_LCHuv(Luv)</code>	Converts from <i>CIE $L^*u^*v^*$</i> colourspace to <i>CIE L^*C^*Huv</i> colourspace.
<code>LCHuv_to_Luv(LCHuv)</code>	Converts from <i>CIE L^*C^*Huv</i> colourspace to <i>CIE $L^*u^*v^*$</i> colourspace.
<code>Luv_to_uv(Luv[, illuminant])</code>	Returns the uv^p chromaticity coordinates from given <i>CIE $L^*u^*v^*$</i> colourspace array.
<code>uv_to_Luv(uv[, illuminant, Y])</code>	Returns the <i>CIE $L^*u^*v^*$</i> colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given <i>L Lightness</i> .
<code>Luv_uv_to_xy(uv)</code>	Returns the <i>xy</i> chromaticity coordinates from given <i>CIE $L^*u^*v^*$</i> colourspace uv^p chromaticity coordinates.
<code>xy_to_Luv_uv(xy)</code>	Returns the <i>CIE $L^*u^*v^*$</i> colourspace uv^p chromaticity coordinates from given <i>xy</i> chromaticity coordinates.

colour.XYZ_to_Luv

colour.XYZ_to_Luv(XYZ, illuminant=array([0.3127, 0.329]))

Converts from *CIE XYZ* tristimulus values to *CIE $L^*u^*v^*$* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE $L^*u^*v^*$* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804f], [Wik07c]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Luv(XYZ) # doctest: +ELLIPSIS
array([ 41.5278752...,  96.8362605...,  17.7521014...])
```

colour.Luv_to_XYZ

`colour.Luv_to_XYZ(Luv, illuminant=array([ 0.3127, 0.329 ]))`

Converts from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Luv** (array_like) – *CIE L*u*v** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f], [Wik07c]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_XYZ(Luv) # doctest: +ELLIPSIS
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Luv_to_LCHuv

`colour.Luv_to_LCHuv(Luv)`

Converts from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.

Parameters `Luv` (array_like) – *CIE L*u*v** colourspace array.

Returns *CIE L*C*Huv* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] uv : [0, 360]	L : [0, 1] C : [0, 1] uv : [0, 1]

References

[CIET14804f]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_LCHuv(Luv) # doctest: +ELLIPSIS
array([ 41.5278752..., 98.4499795..., 10.3881634...])
```

colour.LCHuv_to_Luv

`colour.LCHuv_to_Luv(LCHuv)`

Converts from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.

Parameters `LCHuv` (array_like) – *CIE L*C*Huv* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] uv : [0, 360]	L : [0, 1] C : [0, 1] uv : [0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804f]

Examples

```
>>> LCHuv = np.array([41.52787529, 98.44997950, 10.38816348])
>>> LCHuv_to_Luv(LCHuv) # doctest: +ELLIPSIS
array([ 41.5278752..., 96.8362605..., 17.7521014...])
```

colour.Luv_to_uv

`colour.Luv_to_uv(Luv, illuminant=array([ 0.3127, 0.329 ]))`

Returns the uv^p chromaticity coordinates from given CIE $L^*u^*v^*$ colourspace array.

Parameters

- **Luv** (array_like) – CIE $L^*u^*v^*$ colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant* xy chromaticity coordinates or CIE xyY colourspace array.

Returns uv^p chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804e]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_uv(Luv) # doctest: +ELLIPSIS
array([ 0.3772021..., 0.5012026...])
```

colour.uv_to_Luv

colour.uv_to_Luv(uv, illuminant=array([0.3127, 0.329]), Y=1)

Returns the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given *L Lightness*.

Parameters

- **uv** (array_like) – uv^p chromaticity coordinates.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (numeric, optional) – Optional *Y luminance* value used to construct the intermediate *CIE XYZ* colourspace array, the default *Y luminance* value is 1.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804e]

Examples

```
>>> uv = np.array([0.37720213, 0.50120264])
>>> uv_to_Luv(uv) # doctest: +ELLIPSIS
array([ 100.          , 233.1837603..., 42.7474385...])
```

colour.Luv_uv_to_xy

colour.Luv_uv_to_xy(uv)

Returns the *xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.

Parameters **uv** (array_like) – *CIE L*u*v** $u''v''$ chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

References

[Wik07e]

Examples

```
>>> uv = np.array([0.37720213, 0.50120264])
>>> Luv_uv_to_xy(uv) # doctest: +ELLIPSIS
array([ 0.5436955..., 0.3210794...])
```

colour.xy_to_Luv_uv

colour.xy_to_Luv_uv(xy)

Returns the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given xy chromaticity coordinates.

Parameters xy (array_like) – xy chromaticity coordinates.

Returns *CIE L*u*v* u”v”* chromaticity coordinates.

Return type ndarray

References

[Wik07c]

Examples

```
>>> xy = np.array([0.54369558, 0.32107944])
>>> xy_to_Luv_uv(xy) # doctest: +ELLIPSIS
array([ 0.3772021..., 0.5012026...])
```

CIE 1960 UCS Colourspace

colour

XYZ_to_UCS(XYZ)	Converts from <i>CIE XYZ</i> tristimulus values to <i>CIE 1960 UCS</i> colourspace.
UCS_to_XYZ(UVW)	Converts from <i>CIE 1960 UCS</i> colourspace to <i>CIE XYZ</i> tristimulus values.
UCS_to_uv(UVW)	Returns the uv chromaticity coordinates from given <i>CIE 1960 UCS</i> colourspace array.
uv_to_UCS(uv[, V])	Returns the <i>CIE 1960 UCS</i> colourspace array from given uv chromaticity coordinates.
UCS_uv_to_xy(uv)	Returns the xy chromaticity coordinates from given <i>CIE 1960 UCS</i> colourspace uv chromaticity coordinates.
xy_to_UCS_uv(xy)	Returns the <i>CIE 1960 UCS</i> colourspace uv chromaticity coordinates from given xy chromaticity coordinates.

colour.XYZ_to_UCS

colour.XYZ_to_UCS(XYZ)

Converts from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.

Parameters XYZ (array_like) – *CIE XYZ* tristimulus values.

Returns *CIE 1960 UCS* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wik08d], [Wik08a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_UCS(XYZ) # doctest: +ELLIPSIS
array([ 0.1376933..., 0.1219722..., 0.1053731...])
```

colour.UCS_to_XYZ

colour.UCS_to_XYZ(UVW)

Converts from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.

Parameters UVW (array_like) – *CIE 1960 UCS* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik08d], [Wik08a]

Examples

```
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_XYZ(UVW) # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
```

colour.UCS_to_uv

colour.UCS_to_uv(UVW)

Returns the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.

Parameters UVW (array_like) – *CIE 1960 UCS* colourspace array.

Returns *uv* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wik08d]

Examples

```
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_uv(UVW) # doctest: +ELLIPSIS
array([ 0.3772021..., 0.3341350...])
```

colour.uv_to_UCS

colour.uv_to_UCS(uv, V=1)

Returns the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.

Parameters

- **uv** (array_like) – *uv* chromaticity coordinates.
- **V** (numeric, optional) – Optional *V luminance* value used to construct the *CIE 1960 UCS* colourspace array, the default *V luminance* is set to 1.

Returns *CIE 1960 UCS* colourspace array.

Return type ndarray

References

[Wik08d]

Examples

```
>>> uv = np.array([0.37720213, 0.33413508])
>>> uv_to_UCS(uv) # doctest: +ELLIPSIS
array([ 1.1288911...,  1.          ,  0.8639104...])
```

colour.UCS_uv_to_xy

`colour.UCS_uv_to_xy(uv)`

Returns the *xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.

Parameters *uv* (array_like) – *CIE UCS uv* chromaticity coordinates.

Returns *xy* chromaticity coordinates.

Return type ndarray

References

[Wik08d]

Examples

```
>>> uv = np.array([0.37720213, 0.33413508])
>>> UCS_uv_to_xy(uv) # doctest: +ELLIPSIS
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_UCS_uv

`colour.xy_to_UCS_uv(xy)`

Returns the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *xy* chromaticity coordinates.

Parameters *xy* (array_like) – *xy* chromaticity coordinates.

Returns *CIE UCS uv* chromaticity coordinates.

Return type ndarray

References

[Wik08d]

Examples

```
>>> xy = np.array([0.54369555, 0.32107941])
>>> xy_to_UCS_uv(xy) # doctest: +ELLIPSIS
array([ 0.3772021..., 0.3341350...])
```

CIE 1964 U*V*W* Colourspace

colour

<code>XYZ_to_UVW(XYZ[, illuminant])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>CIE 1964 U*V*W*</i> colourspace.
<code>UVW_to_XYZ(UVW[, illuminant])</code>	Converts <i>CIE 1964 U*V*W*</i> colourspace to <i>CIE XYZ</i> tristimulus values.

colour.XYZ_to_UVW

`colour.XYZ_to_UVW(XYZ, illuminant=array([ 0.3127, 0.329 ]))`

Converts from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE 1964 U*V*W** colourspace array.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]

References

[Wik08b]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_UVW(XYZ) # doctest: +ELLIPSIS
array([ 94.5503572..., 11.5553652..., 40.5475740...])
```

colour.UVW_to_XYZ

colour.UVW_to_XYZ(UVW, illuminant=array([0.3127, 0.329]))

Converts CIE 1964 $U^*V^*W^*$ colourspace to CIE XYZ tristimulus values.

Parameters

- **UVW** (array_like) – CIE 1964 $U^*V^*W^*$ colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant* xy chromaticity coordinates or CIE xyY colourspace array.

Returns CIE XYZ tristimulus values.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

Domain	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik08b]

Examples

```
>>> import numpy as np
>>> UVW = np.array([94.55035725, 11.55536523, 40.54757405])
>>> UVW_to_XYZ(UVW)
array([ 20.654008, 12.197225, 5.136952])
```

Hunter L,a,b Colour Scale

colour

<code>XYZ_to_Hunter_Lab(XYZ[, XYZ_n, K_ab])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>Hunter L,a,b</i> colour scale.
<code>Hunter_Lab_to_XYZ(Lab[, XYZ_n, K_ab])</code>	Converts from <i>Hunter L,a,b</i> colour scale to <i>CIE XYZ</i> tristimulus values.
<code>XYZ_to_K_ab_HunterLab1966(XYZ)</code>	Converts from <i>whitepoint CIE XYZ</i> tristimulus values to <i>Hunter L,a,b K_a</i> and <i>K_b</i> chromaticity coefficients.

colour.XYZ_to_Hunter_Lab

`colour.XYZ_to_Hunter_Lab(XYZ, XYZ_n=array([ 95.02, 100., 108.82]), K_ab=array([ 172.3, 67.2]))`
 Converts from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if *K_ab* is set to *None* it will be computed using `colour.XYZ_to_K_ab_HunterLab1966()`.

Returns *Hunter L,a,b* colour scale array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[Hun08a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Lab(XYZ, D65.XYZ_n, D65.K_ab) # doctest: +ELLIPSIS
array([ 34.9245257...,  47.0618985...,  14.3861510...])
```

colour.Hunter_Lab_to_XYZ

`colour.Hunter_Lab_to_XYZ(Lab, XYZ_n=array([ 95.02, 100., 108.82]), K_ab=array([ 172.3, 67.2]))`
 Converts from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (array_like) – *Hunter L,a,b* colour scale array.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if *K_ab* is set to *None* it will be computed using `colour.XYZ_to_K_ab_HunterLab1966()`.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[Hun08a]

Examples

```
>>> Lab = np.array([34.92452577, 47.06189858, 14.38615107])
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> Hunter_Lab_to_XYZ(Lab, D65.XYZ_n, D65.K_ab)
array([ 20.654008,  12.197225,   5.136952])
```

colour.XYZ_to_K_ab_HunterLab1966

colour.XYZ_to_K_ab_HunterLab1966(XYZ)

Converts from *whitepoint* CIE XYZ tristimulus values to Hunter L,a,b K_a and K_b chromaticity coefficients.

Parameters XYZ (array_like) – *Whitepoint* CIE XYZ tristimulus values.

Returns Hunter L,a,b K_a and K_b chromaticity coefficients.

Return type ndarray

References

[Hun08b]

Examples

```
>>> XYZ = np.array([109.850, 100.000, 35.585])
>>> XYZ_to_K_ab_HunterLab1966(XYZ) # doctest: +ELLIPSIS
array([ 185.2378721...,  38.4219142...])
```

Hunter R_d,a,b Colour Scale

colour

XYZ_to_Hunter_Rdab(XYZ[, XYZ_n, K_ab])	Converts from CIE XYZ tristimulus values to Hunter R_d,a,b colour scale.
Hunter_Rdab_to_XYZ(R_d_ab[, XYZ_n, K_ab])	Converts from Hunter R_d,a,b colour scale to CIE XYZ tristimulus values.

colour.XYZ_to_Hunter_Rdab

colour.XYZ_to_Hunter_Rdab(XYZ, XYZ_n=array([95.02, 100., 108.82]), K_ab=array([172.3, 67.2]))

Converts from CIE XYZ tristimulus values to Hunter R_d,a,b colour scale.

Parameters

- XYZ (array_like) – CIE XYZ tristimulus values.
- XYZ_n (array_like, optional) – Reference *illuminant* tristimulus values.
- K_ab (array_like, optional) – Reference *illuminant* chromaticity coefficients, if K_ab is set to *None* it will be computed using colour.XYZ_to_K_ab_HunterLab1966().

Returns Hunter R_d,a,b colour scale array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]

References

[Hun12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Rdab(XYZ, D65.XYZ_n, D65.K_ab)
... # doctest: +ELLIPSIS
array([ 12.197225 ...,  57.1253787...,  17.4624134...])
```

colour.Hunter_Rdab_to_XYZ

`colour.Hunter_Rdab_to_XYZ(R_d_ab, XYZ_n=array([ 95.02, 100., 108.82]), K_ab=array([ 172.3, 67.2]))`

Converts from *Hunter Rd,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **R_d_ab** (array_like) – *Hunter Rd,a,b* colour scale array.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if *K_ab* is set to *None* it will be computed using `colour.XYZ_to_K_ab_HunterLab1966()`.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[Hun12]

Examples

```
>>> import numpy as np
>>> R_d_ab = np.array([12.19722500, 57.12537874, 17.46241341])
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> Hunter_Rdab_to_XYZ(R_d_ab, D65.XYZ_n, D65.K_ab)
array([ 20.654008, 12.197225,  5.136952])
```

DIN99 Colourspace

colour

<code>Lab_to_DIN99(Lab[, k_E, k_CH])</code>	Converts from <i>CIE L*a*b*</i> colourspace to <i>DIN99</i> colourspace.
<code>DIN99_to_Lab(Lab_99[, k_E, k_CH])</code>	Converts from <i>DIN99</i> colourspace to <i>CIE L*a*b*</i> colourspace.

colour.Lab_to_DIN99

`colour.Lab_to_DIN99(Lab, k_E=1, k_CH=1)`

Converts from *CIE L*a*b** colourspace to *DIN99* colourspace.

Parameters

- **Lab** (array_like) – *CIE L*a*b** colourspace array.
- **k_E** (numeric, optional) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (numeric, optional) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.

Returns *DIN99* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_DIN99(Lab) # doctest: +ELLIPSIS
array([ 53.2282198..., 28.4163465...,  3.8983955...])
```

colour.DIN99_to_Lab

`colour.DIN99_to_Lab(Lab_99, k_E=1, k_CH=1)`

Converts from *DIN99* colourspace to *CIE L*a*b** colourspace.

Parameters

- **Lab_99** (array_like) – *DIN99* colourspace array.
- **k_E** (numeric, optional) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (numeric, optional) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_99 = np.array([53.22821988, 28.41634656, 3.89839552])
>>> DIN99_to_Lab(Lab_99) # doctest: +ELLIPSIS
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

colour

JMh_CIECAM02_to_CAM02LCD(JMh)	Converts from <i>CIECAM02 JMh</i> correlates array to <i>Luo et al.(2006) CAM02-LCD</i> colourspace <i>J'a'b'</i> array..
CAM02LCD_to_JMh_CIECAM02(Jpapbp)	Converts from <i>Luo et al.(2006) CAM02-LCD</i> colourspace <i>J'a'b'</i> array to <i>CIECAM02 JMh</i> correlates array..
JMh_CIECAM02_to_CAM02SCD(JMh)	Converts from <i>CIECAM02 JMh</i> correlates array to <i>Luo et al.(2006) CAM02-SCD</i> colourspace <i>J'a'b'</i> array..
CAM02SCD_to_JMh_CIECAM02(Jpapbp)	Converts from <i>Luo et al.(2006) CAM02-SCD</i> colourspace <i>J'a'b'</i> array to <i>CIECAM02 JMh</i> correlates array..
JMh_CIECAM02_to_CAM02UCS(JMh)	Converts from <i>CIECAM02 JMh</i> correlates array to <i>Luo et al.(2006) CAM02-UCS</i> colourspace <i>J'a'b'</i> array..
CAM02UCS_to_JMh_CIECAM02(Jpapbp)	Converts from <i>Luo et al.(2006) CAM02-UCS</i> colourspace <i>J'a'b'</i> array to <i>CIECAM02 JMh</i> correlates array..

colour.JMh_CIECAM02_to_CAM02LCD

colour.JMh_CIECAM02_to_CAM02LCD(JMh)

Converts from *CIECAM02 JMh* correlates array to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Parameters JMh (array_like) – *CIECAM02* correlates array *JMh*.

Returns *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02LCD(JMh) # doctest: +ELLIPSIS
array([ 54.9043313..., -0.0845039..., -0.0685483...])
```

colour.CAM02LCD_to_JMh_CIECAM02

colour.CAM02LCD_to_JMh_CIECAM02(Jpapbp)

Converts from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02 JMh* correlates array.

Parameters Jpapbp (array_like) – *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array *JMh*.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08450395, -0.06854831])
>>> CAM02LCD_to_JMh_CIECAM02(Jpapbp) # doctest: +ELLIPSIS
array([ 4.1731091...e+01,  1.0884217...e-01,  2.1904843...e+02])
```

colour.JMh_CIECAM02_to_CAM02SCD

`colour.JMh_CIECAM02_to_CAM02SCD(JMh)`

Converts from *CIECAM02 JMh* correlates array to Luo et al. (2006) CAM02-SCD colourspace *J'a'b'* array.

Parameters *JMh* (array_like) – *CIECAM02* correlates array *JMh*.

Returns Luo et al. (2006) CAM02-SCD colourspace *J'a'b'* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02SCD(JMh) # doctest: +ELLIPSIS
array([ 54.9043313..., -0.0843617..., -0.0684329...])
```

colour.CAM02SCD_to_JMh_CIECAM02

colour.CAM02SCD_to_JMh_CIECAM02(*Jpapbp*)

Converts from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02 JMh* correlates array.

Parameters *Jpapbp* (array_like) – *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array *JMh*.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpabpp = np.array([54.90433134, -0.08436178, -0.06843298])
>>> CAM02SCD_to_JMh_CIECAM02(Jpabpp) # doctest: +ELLIPSIS
array([ 4.1731091...e+01,  1.0884217...e-01,  2.1904843...e+02])
```

colour.JMh_CIECAM02_to_CAM02UCS

colour.JMh_CIECAM02_to_CAM02UCS(JMh)

Converts from *CIECAM02 JMh* correlates array to *Luo et al. (2006) CAM02-UCS* colourspace *J'a'b'* array.

Parameters JMh (array_like) – *CIECAM02* correlates array *JMh*.

Returns *Luo et al. (2006) CAM02-UCS* colourspace *J'a'b'* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpabpp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
```

(continues on next page)

(continued from previous page)

```
... XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02UCS(JMh) # doctest: +ELLIPSIS
array([ 54.9043313..., -0.0844236..., -0.0684831...])
```

colour.CAM02UCS_to_JMh_CIECAM02

colour.CAM02UCS_to_JMh_CIECAM02(*Jpapbp*)

Converts from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02 JMh* correlates array.

Parameters *Jpapbp* (array_like) – *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array *JMh*.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08442362, -0.06848314])
>>> CAM02UCS_to_JMh_CIECAM02(Jpapbp) # doctest: +ELLIPSIS
array([ 4.1731091...e+01,  1.0884217...e-01,  2.1904843...e+02])
```

CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

colour

JMh_CAM16_to_CAM16LCD(JMh, *[, ...])

Converts from *CAM16 JMh* correlates array to *Li et al.(2017) CAM16-LCD* colourspace $J'a'b'$ array..

Continued on next page

Table 193 – continued from previous page

<code>CAM16LCD_to_JMh_CAM16(Jpapbp, *[, ...])</code>	Converts from <i>Li et al.(2017)</i> CAM16-LCD colourspace $J'a'b'$ array to CAM16 JMh correlates array..
<code>JMh_CAM16_to_CAM16SCD(JMh, *[, ...])</code>	Converts from CAM16 JMh correlates array to <i>Li et al.(2017)</i> CAM16-SCD colourspace $J'a'b'$ array..
<code>CAM16SCD_to_JMh_CAM16(Jpapbp, *[, ...])</code>	Converts from <i>Li et al.(2017)</i> CAM16-SCD colourspace $J'a'b'$ array to CAM16 JMh correlates array..
<code>JMh_CAM16_to_CAM16UCS(JMh, *[, ...])</code>	Converts from CAM16 JMh correlates array to <i>Li et al.(2017)</i> CAM16-UCS colourspace $J'a'b'$ array..
<code>CAM16UCS_to_JMh_CAM16(Jpapbp, *[, ...])</code>	Converts from <i>Li et al.(2017)</i> CAM16-UCS colourspace $J'a'b'$ array to CAM16 JMh correlates array..

colour.JMh_CAM16_to_CAM16LCD

`colour.JMh_CAM16_to_CAM16LCD(JMh, *, coefficients=Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053))`
 Converts from CAM16 JMh correlates array to *Li et al. (2017)* CAM16-LCD colourspace $J'a'b'$ array.

Parameters JMh (array_like) – CAM16 correlates array JMh.

Returns *Li et al. (2017)* CAM16-LCD colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02LCD()` definition for an usage example.

colour.CAM16LCD_to_JMh_CAM16

```
colour.CAM16LCD_to_JMh_CAM16(Jpapbp, *, coefficients=Coefficients_UCS_Luo2006(K_L=0.77,  
c_1=0.007, c_2=0.0053))
```

Converts from *Li et al. (2017)* CAM16-LCD colourspace $J'a'b'$ array to CAM16 JMh correlates array.

Parameters `Jpapbp` (array_like) – *Li et al. (2017)* CAM16-LCD colourspace $J'a'b'$ array.

Returns CAM16 correlates array JMh .

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02LCD_to_JMh_CIECAM02()` definition for an usage example.

colour.JMh_CAM16_to_CAM16SCD

```
colour.JMh_CAM16_to_CAM16SCD(JMh, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.24,  
c_1=0.007, c_2=0.0363))
```

Converts from CAM16 JMh correlates array to *Li et al. (2017)* CAM16-SCD colourspace $J'a'b'$ array.

Parameters `JMh` (array_like) – CAM16 correlates array JMh .

Returns *Li et al. (2017)* CAM16-SCD colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02SCD()` definition for an usage example.

colour.CAM16SCD_to_JMh_CAM16

`colour.CAM16SCD_to_JMh_CAM16(Jpapbp, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363))`

Converts from *Li et al. (2017)* CAM16-SCD colourspace $J'a'b'$ array to CAM16 JMh correlates array.

Parameters `Jpapbp` (array_like) – *Li et al. (2017)* CAM16-SCD colourspace $J'a'b'$ array.

Returns CAM16 correlates array *JMh*.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02SCD_to_JMh_CIECAM02()` definition for an usage example.

`colour.JMh_CAM16_to_CAM16UCS`

```
colour.JMh_CAM16_to_CAM16UCS(JMh, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007,
c_2=0.0228))
```

Converts from *CAM16 JMh* correlates array to *Li et al. (2017) CAM16-UCS* colourspace *J'a'b'* array.

Parameters *JMh* (array_like) – *CAM16* correlates array *JMh*.

Returns *Li et al. (2017) CAM16-UCS* colourspace *J'a'b'* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02UCS()` definition for an usage example.

`colour.CAM16UCS_to_JMh_CAM16`

```
colour.CAM16UCS_to_JMh_CAM16(Jpapbp, *, coefficients=Coefficients_UCS_Luo2006(K_L=1.0,
c_1=0.007, c_2=0.0228))
```

Converts from *Li et al. (2017) CAM16-UCS* colourspace *J'a'b'* array to *CAM16 JMh* correlates array.

Parameters *Jpapbp* (array_like) – *Li et al. (2017) CAM16-UCS* colourspace *J'a'b'* array.

Returns *CAM16* correlates array *JMh*.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02UCS_to_JMh_CIECAM02()` definition for an usage example.

IPT Colourspace

colour

<code>XYZ_to_IPT(XYZ)</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>IPT</i> colourspace.
<code>IPT_to_XYZ(IPT)</code>	Converts from <i>IPT</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>IPT_hue_angle(IPT)</code>	Computes the hue angle in degrees from <i>IPT</i> colourspace.

colour.XYZ_to_IPT

`colour.XYZ_to_IPT(XYZ)`

Converts from *CIE XYZ* tristimulus values to *IPT* colourspace.

Parameters *XYZ* (array_like) – *CIE XYZ* tristimulus values.

Returns *IPT* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT	I : [0, 1]	I : [0, 1]
	P : [-1, 1]	P : [-1, 1]
	T : [-1, 1]	T : [-1, 1]

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

[Fai13d]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IPT(XYZ) # doctest: +ELLIPSIS
array([ 0.3842619..., 0.3848730..., 0.1888683...])
```

colour.IPT_to_XYZ

colour.IPT_to_XYZ(*IPT*)

Converts from *IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters *IPT* (array_like) – *IPT* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1]	I : [0, 1]
	P : [-1, 1]	P : [-1, 1]
	T : [-1, 1]	T : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Fai13d]

Examples

```
>>> IPT = np.array([0.38426191, 0.38487306, 0.18886838])
>>> IPT_to_XYZ(IPT) # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
```

colour.IPT_hue_angle

colour.**IPThue_angle**(*IPT*)

Computes the hue angle in degrees from *IPT* colourspace.

Parameters *IPT* (array_like) – *IPT* colourspace array.

Returns Hue angle in degrees.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
hue	[0, 360]	[0, 1]

References

[Fai13d]

Examples

```
>>> IPT = np.array([0.96907232, 1, 1.12179215])
>>> IPT_hue_angle(IPT) # doctest: +ELLIPSIS
48.2852074...
```

hdr-CIELAB Colourspace

colour

<code>XYZ_to_hdr_CIELab(XYZ[, illuminant, Y_s, ...])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>hdr-CIELAB</i> colourspace.
<code>hdr_CIELab_to_XYZ(Lab_hdr[, illuminant, ...])</code>	Converts from <i>hdr-CIELAB</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>HDR_CIELAB_METHODS</code>	Supported <i>hdr-CIELAB</i> colourspace computation methods.

colour.XYZ_to_hdr_CIELab

`colour.XYZ_to_hdr_CIELab(XYZ, illuminant=array([ 0.3127, 0.329 ]), Y_s=0.2, Y_abs=100, method='Fairchild 2011')`

Converts from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2 .
- **method** (unicode, optional) – {'Fairchild 2011', 'Fairchild 2010'}, Computation method.

Returns *hdr-CIELAB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]

- Conversion to polar coordinates to compute the *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with `colour.Lab_to_LCHab()` definition.
- Conversion to cartesian coordinates from the *Lightness* L_{hdr} , *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with `colour.LCHab_to_Lab()` definition.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_CIELab(XYZ) # doctest: +ELLIPSIS
array([ 51.8700206..., 60.4763385..., 32.1455191...])
>>> XYZ_to_hdr_CIELab(XYZ, method='Fairchild 2010') # doctest: +ELLIPSIS
array([ 31.9962111..., 128.0076303..., 48.7695230...])
```


colour.hdr_CIELab_to_XYZ

`colour.hdr_CIELab_to_XYZ(Lab_hdr, illuminant=array([ 0.3127, 0.329 ]), Y_s=0.2, Y_abs=100, method='Fairchild 2011')`

Converts from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab_hdr** (array_like) – *hdr-CIELAB* colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2 .
- **method** (unicode, optional) – {'Fairchild 2011', 'Fairchild 2010'}, Computation method.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> Lab_hdr = np.array([51.87002062, 60.4763385, 32.14551912])
>>> hdr_CIELab_to_XYZ(Lab_hdr) # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
>>> Lab_hdr = np.array([31.99621114, 128.00763036, 48.76952309])
>>> hdr_CIELab_to_XYZ(Lab_hdr, method='Fairchild 2010')
... # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
```

colour.HDR_CIELAB_METHODS

`colour.HDR_CIELAB_METHODS` = ('Fairchild 2010', 'Fairchild 2011')
Supported *hdr-CIELAB* colourspace computation methods.

References

[FW10], [FC11]

`HDR_CIELAB_METHODS` [tuple] {'Fairchild 2011', 'Fairchild 2010'}

hdr-IPT Colourspace

`colour`

<code>XYZ_to_hdr_IPT(XYZ[, Y_s, Y_abs, method])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>hdr-IPT</i> colourspace.
<code>hdr_IPT_to_XYZ(IPT_hdr[, Y_s, Y_abs, method])</code>	Converts from <i>hdr-IPT</i> colourspace to <i>CIE XYZ</i> tristimulus values.
<code>HDR_IPT_METHODS</code>	Supported <i>hdr-IPT</i> colourspace computation methods.

colour.XYZ_to_hdr_IPT

`colour.XYZ_to_hdr_IPT(XYZ, Y_s=0.2, Y_abs=100, method='Fairchild 2011')`
Converts from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2 .
- **method** (unicode, optional) – {'Fairchild 2011', 'Fairchild 2010'}, Computation method.

Returns *hdr-IPT* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_IPT(XYZ) # doctest: +ELLIPSIS
array([ 48.3937634..., 42.4499020..., 22.0195403...])
>>> XYZ_to_hdr_IPT(XYZ, method='Fairchild 2010') # doctest: +ELLIPSIS
array([ 30.0287314..., 83.9384506..., 34.9028738...])
```

colour.hdr_IPT_to_XYZ

`colour.hdr_IPT_to_XYZ(IPT_hdr, Y_s=0.2, Y_abs=100, method='Fairchild 2011')`

Converts from *hdr-IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **IPT_hdr** (array_like) – *hdr-IPT* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2 .
- **method** (unicode, optional) – {'Fairchild 2011', 'Fairchild 2010'}, Computation method.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Do-main	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> IPT_hdr = np.array([48.39376346, 42.44990202, 22.01954033])
>>> hdr_IPT_to_XYZ(IPT_hdr) # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
>>> IPT_hdr = np.array([30.02873147, 83.93845061, 34.90287382])
>>> hdr_IPT_to_XYZ(IPT_hdr, method='Fairchild 2010')
... # doctest: +ELLIPSIS
array([ 0.2065400..., 0.1219722..., 0.0513695...])
```

colour.HDR_IPT_METHODS

`colour.HDR_IPT_METHODS` = ('Fairchild 2010', 'Fairchild 2011')
Supported *hdr-IPT* colourspace computation methods.

References

[FW10], [FC11]

`HDR_IPT_METHODS` [tuple] {'Fairchild 2011', 'Fairchild 2010'}

OSA UCS Colourspace

`colour`

<code>XYZ_to_OSA_UCS(XYZ)</code>	Converts from <i>CIE XYZ</i> tristimulus values under the <i>CIE 1964 10 Degree Standard Observer</i> to <i>OSA UCS</i> colourspace.
<code>OSA_UCS_to_XYZ(Ljg[, optimisation_parameters])</code>	Converts from <i>OSA UCS</i> colourspace to <i>CIE XYZ</i> tristimulus values under the <i>CIE 1964 10 Degree Standard Observer</i> .

colour.XYZ_to_OSA_UCS

`colour.XYZ_to_OSA_UCS(XYZ)`

Converts from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.

The lightness axis, *L* is usually in range [-9, 5] and centered around middle gray (Munsell N/6). The yellow-blue axis, *j* is usually in range [-15, 15]. The red-green axis, *g* is usually in range [-20, 15].

Parameters `XYZ` (array_like) – *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Returns *OSA UCS Ljg* lightness, jaune (yellowness), and greenness.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

- *OSA UCS* uses the *CIE 1964 10 Degree Standard Observer*.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_OSA_UCS(XYZ) # doctest: +ELLIPSIS
array([-3.0049979..., 2.9971369..., -9.6678423...])
```

colour.OSA_UCS_to_XYZ

`colour.OSA_UCS_to_XYZ(Ljg, optimisation_parameters=None)`

Converts from *OSA UCS* colour space to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Parameters

- **Ljg** (array_like) – *OSA UCS* *Ljg* lightness, jaune (yellowness), and greenness.
- **optimisation_parameters** (dict_like, optional) – Parameters for `scipy.optimize.fmin()` definition.

Returns *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Return type ndarray

Warning: There is no analytical reverse transformation from *OSA UCS* to *Ljg* lightness, jaune (yellowness), and greenness to *CIE XYZ* tristimulus values, the current implementation relies on optimization using `scipy.optimize.fmin()` definition and thus has reduced precision and poor performance.

Notes

Domain	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- OSA UCS uses the CIE 1964 10 Degree Standard Observer.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> Ljg = np.array([-3.00499790, 2.99713697, -9.66784231])
>>> OSA_UCS_to_XYZ(Ljg) # doctest: +ELLIPSIS
array([ 20.6540240..., 12.1972369...,  5.1369372...])
```

$J_zA_zB_z$ Colourspace

colour

<code>XYZ_to_JzAzBz(XYZ_D65[, constants])</code>	Converts from CIE XYZ tristimulus values to $J_zA_zB_z$ colourspace.
<code>JzAzBz_to_XYZ(JzAzBz[, constants])</code>	Converts from $J_zA_zB_z$ colourspace to CIE XYZ tristimulus values.

colour.XYZ_to_JzAzBz

`colour.XYZ_to_JzAzBz(XYZ_D65, constants={'b': 1.15, 'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'd': -0.56, 'd_0': 1.6295499532821565e-11, 'g': 0.66, 'm_1': 0.1593017578125, 'm_2': 134.03437499999998})`

Converts from CIE XYZ tristimulus values to $J_zA_zB_z$ colourspace.

Parameters

- **XYZ_D65** (array_like) – CIE XYZ tristimulus values under CIE Standard Illuminant D Series D65.
- **constants** (Structure, optional) – $J_zA_zB_z$ colourspace constants.

Returns $J_zA_zB_z$ colourspace array where J_z is Lightness, A_z is redness-greenness and B_z is yellowness-blueness.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
JzAzBz	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]

References

[SCKL17]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_JzAzBz(XYZ) # doctest: +ELLIPSIS
array([ 0.0053504..., 0.0092430..., 0.0052600...])
```

colour.JzAzBz_to_XYZ

colour.JzAzBz_to_XYZ(JzAzBz, constants={'b': 1.15, 'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'd': -0.56, 'd_0': 1.6295499532821565e-11, 'g': 0.66, 'm_1': 0.1593017578125, 'm_2': 134.03437499999998})

Converts from $J_z A_z B_z$ colourspace to CIE XYZ tristimulus values.

Parameters

- **JzAzBz** (array_like) – $J_z A_z B_z$ colourspace array where J_z is Lightness, A_z is redness-greenness and B_z is yellowness-blueness.
- **constants** (Structure, optional) – $J_z A_z B_z$ colourspace constants.

Returns CIE XYZ tristimulus values under CIE Standard Illuminant D Series D65.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JzAzBz	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[SCKL17]

Examples

```
>>> JzAzBz = np.array([0.00535048, 0.00924302, 0.00526007])
>>> JzAzBz_to_XYZ(JzAzBz) # doctest: +ELLIPSIS
array([ 0.2065402...,  0.1219723...,  0.0513696...])
```

RGB Colourspace and Transformations

colour

<code>XYZ_to_RGB(XYZ, illuminant_XYZ, ...[, ...])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>RGB</i> colourspace array.
<code>RGB_to_XYZ(RGB, illuminant_RGB, ...[, ...])</code>	Converts given <i>RGB</i> colourspace array to <i>CIE XYZ</i> tristimulus values.
<code>RGB_to_RGB(RGB, input_colourspace, ...[, ...])</code>	Converts given <i>RGB</i> colourspace array from given input <i>RGB</i> colourspace to output <i>RGB</i> colourspace using given <i>chromatic adaptation</i> method.
<code>RGB_to_RGB_matrix(input_colourspace, ...[, ...])</code>	Computes the matrix <i>M</i> converting from given input <i>RGB</i> colourspace to output <i>RGB</i> colourspace using given <i>chromatic adaptation</i> method.

colour.XYZ_to_RGB

`colour.XYZ_to_RGB(XYZ, illuminant_XYZ, illuminant_RGB, XYZ_to_RGB_matrix, chromatic_adaptation_transform='CAT02', encoding_cctf=None)`

Converts from *CIE XYZ* tristimulus values to *RGB* colourspace array.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant_XYZ** (array_like) – *CIE XYZ* tristimulus values *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **illuminant_RGB** (array_like) – *RGB* colourspace *illuminant xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **XYZ_to_RGB_matrix** (array_like) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **encoding_cctf** (object, optional) – Encoding colour component transfer function (Encoding CCTF) or opto-electronic transfer function (OETF / OECF).

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> XYZ = np.array([0.21638819, 0.12570000, 0.03847493])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = np.array(
...     [[3.24062548, -1.53720797, -0.49862860],
...      [-0.96893071, 1.87575606, 0.04151752],
...      [0.05571012, -0.20402105, 1.05699594]]
... )
>>> XYZ_to_RGB(XYZ, illuminant_XYZ, illuminant_RGB, XYZ_to_RGB_matrix,
...             chromatic_adaptation_transform) # doctest: +ELLIPSIS
array([ 0.4559557..., 0.0303970..., 0.0408724...])
```

colour.RGB_to_XYZ

`colour.RGB_to_XYZ(`*RGB*`,` *illuminant_RGB*`,` *illuminant_XYZ*`,` *RGB_to_XYZ_matrix*`,` *chromatic_adaptation_transform*`=`*'CAT02'*`,` *decoding_cctf*`=`*None*`)`

Converts given *RGB* colourspace array to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (*array_like*) – *RGB* colourspace array.
- **illuminant_RGB** (*array_like*) – *RGB* colourspace *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.
- **illuminant_XYZ** (*array_like*) – *CIE XYZ* tristimulus values *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.
- **RGB_to_XYZ_matrix** (*array_like*) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (*unicode*, *optional*) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', *None*}, *Chromatic adaptation transform*, if *None* no chromatic adaptation is performed.
- **decoding_cctf** (*object*, *optional*) – *Decoding colour component transfer function* (*Decoding CCTF*) or *electro-optical transfer function* (*EOTF* / *EOCF*).

Returns *CIE XYZ* tristimulus values.

Return type *ndarray*

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> RGB_to_XYZ_matrix = np.array(
...     [[0.41240000, 0.35760000, 0.18050000],
...      [0.21260000, 0.71520000, 0.07220000],
...      [0.01930000, 0.11920000, 0.95050000]]
... )
>>> RGB_to_XYZ(RGB, illuminant_RGB, illuminant_XYZ, RGB_to_XYZ_matrix,
...             chromatic_adaptation_transform) # doctest: +ELLIPSIS
array([ 0.2163881..., 0.1257 , 0.0384749...])
```

colour.RGB_to_RGB

`colour.RGB_to_RGB(RGB, input_colourspace, output_colourspace, chromatic_adaptation_transform='CAT02', apply_decoding_cctf=False, apply_encoding_cctf=False)`

Converts given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **RGB** (*array_like*) – *RGB* colourspace array.
- **input_colourspace** (*RGB_Colourspace*) – *RGB* input colourspace.
- **output_colourspace** (*RGB_Colourspace*) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (*unicode*, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_decoding_cctf** (*bool*, optional) – Apply input colourspace decoding colour component transfer function / electro-optical transfer function.
- **apply_encoding_cctf** (*bool*, optional) – Apply output colourspace encoding colour component transfer function / opto-electronic transfer function.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> RGB_to_RGB(RGB, sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
... # doctest: +ELLIPSIS
array([ 0.2568891...,  0.0721446...,  0.0465553...])
```

colour.RGB_to_RGB_matrix

`colour.RGB_to_RGB_matrix(input_colourspace, output_colourspace, chromatic_adaptation_transform='CAT02')`

Computes the matrix M converting from given input RGB colourspace to output RGB colourspace using given *chromatic adaptation* method.

Parameters

- **input_colourspace** (*RGB_Colourspace*) – RGB input colourspace.
- **output_colourspace** (*RGB_Colourspace*) – RGB output colourspace.
- **chromatic_adaptation_transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.

Returns Conversion matrix M .

Return type ndarray

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB_to_RGB_matrix(sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
... # doctest: +ELLIPSIS
array([[ 0.5288241...,  0.3340609...,  0.1373616...],
       [ 0.0975294...,  0.8790074...,  0.0233981...],
       [ 0.0163599...,  0.1066124...,  0.8772485...]])
```

Ancillary Objects

colour

<code>XYZ_to_sRGB(XYZ[, illuminant, ...])</code>	Converts from <i>CIE XYZ</i> tristimulus values to <i>sRGB</i> colourspace.
<code>sRGB_to_XYZ(RGB[, illuminant, ...])</code>	Converts from <i>sRGB</i> colourspace to <i>CIE XYZ</i> tristimulus values.

`colour.XYZ_to_sRGB`

`colour.XYZ_to_sRGB(XYZ, illuminant=array([ 0.3127, 0.329 ]), chromatic_adaptation_transform='CAT02', apply_encoding_cctf=True)`
Converts from *CIE XYZ* tristimulus values to *sRGB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, *Chromatic adaptation* transform.
- **apply_encoding_cctf** (bool, optional) – Apply *sRGB* encoding colour component transfer function / opto-electronic transfer function.

Returns *sRGB* colour array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_sRGB(XYZ) # doctest: +ELLIPSIS
array([ 0.7057393..., 0.1924826..., 0.2235416...])
```

`colour.sRGB_to_XYZ`

`colour.sRGB_to_XYZ(RGB, illuminant=array([ 0.3127, 0.329 ]), chromatic_adaptation_method='CAT02', apply_decoding_cctf=True)`
Converts from *sRGB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *sRGB* colourspace array.

- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_method** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, *Chromatic adaptation method*.
- **apply_decoding_cctf** (bool, optional) – Apply *sRGB* decoding colour component transfer function / electro-optical transfer function.

Returns CIE XYZ tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> RGB = np.array([0.70573936, 0.19248266, 0.22354169])
>>> sRGB_to_XYZ(RGB) # doctest: +ELLIPSIS
array([ 0.2065429..., 0.1219794..., 0.0513714...])
```

RGB Colourspace Derivation

colour

<code>normalised_primary_matrix(primaries, whitepoint)</code>	Returns the <i>normalised primary matrix</i> using given <i>primaries</i> and <i>whitepoint xy</i> chromaticity coordinates.
<code>chromatically_adapted_primaries(primaries, ...)</code>	Chromatically adapts given <i>primaries xy</i> chromaticity coordinates from test <i>whitepoint_t</i> to reference <i>whitepoint_r</i> .
<code>primaries_whitepoint(npm)</code>	Returns the <i>primaries</i> and <i>whitepoint xy</i> chromaticity coordinates using given <i>normalised primary matrix</i> .
<code>RGB_luminance(RGB, primaries, whitepoint)</code>	Returns the <i>luminance Y</i> of given <i>RGB</i> components from given <i>primaries</i> and <i>whitepoint</i> .
<code>RGB_luminance_equation(primaries, whitepoint)</code>	Returns the <i>luminance equation</i> from given <i>primaries</i> and <i>whitepoint</i> .

colour.normalised_primary_matrix

colour.normalised_primary_matrix(primaries, whitepoint)

Returns the *normalised primary matrix* using given *primaries* and *whitepoint xy* chromaticity coordinates.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries *xy* chromaticity coordinates.
- **whitepoint** (array_like) – Illuminant / whitepoint *xy* chromaticity coordinates.

Returns *Normalised primary matrix*.

Return type ndarray, (3, 3)

References

[Society of MPEngineers93]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> w = np.array([0.32168, 0.33767])
>>> normalised_primary_matrix(p, w) # doctest: +ELLIPSIS
array([[ 9.5255239...e-01,  0.0000000...e+00,  9.3678631...e-05],
       [ 3.4396645...e-01,  7.2816609...e-01, -7.2132546...e-02],
       [ 0.0000000...e+00,  0.0000000...e+00,  1.0088251...e+00]])
```

colour.chromatically_adapted_primaries

colour.chromatically_adapted_primaries(primaries, whitepoint_t, whitepoint_r, chromatic_adaptation_transform='CAT02')

Chromatically adapts given *primaries xy* chromaticity coordinates from test *whitepoint_t* to reference *whitepoint_r*.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries *xy* chromaticity coordinates.
- **whitepoint_t** (array_like) – Test illuminant / whitepoint *xy* chromaticity coordinates.
- **whitepoint_r** (array_like) – Reference illuminant / whitepoint *xy* chromaticity coordinates.
- **chromatic_adaptation_transform** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, *Chromatic adaptation transform*.

Returns Chromatically adapted primaries *xy* chromaticity coordinates.

Return type ndarray

Examples

```
>>> p = np.array([0.64, 0.33, 0.30, 0.60, 0.15, 0.06])
>>> w_t = np.array([0.31270, 0.32900])
>>> w_r = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> chromatically_adapted_primaries(p, w_t, w_r,
...                               chromatic_adaptation_transform)
... # doctest: +ELLIPSIS
array([[ 0.6484414...,  0.3308533...],
       [ 0.3211951...,  0.5978443...],
       [ 0.1558932...,  0.0660492...]])
```

colour.primitives_whitepoint

colour.primitives_whitepoint(*npm*)

Returns the *primaries* and *whitepoint xy* chromaticity coordinates using given *normalised primary matrix*.

Parameters *npm* (array_like, (3, 3)) – *Normalised primary matrix*.

Returns *Primaries* and *whitepoint xy* chromaticity coordinates.

Return type tuple

References

[Tri15]

Examples

```
>>> npm = np.array([[9.52552396e-01, 0.00000000e+00, 9.36786317e-05],
...                 [3.43966450e-01, 7.28166097e-01, -7.21325464e-02],
...                 [0.00000000e+00, 0.00000000e+00, 1.00882518e+00]])
>>> p, w = primitives_whitepoint(npm)
>>> p # doctest: +ELLIPSIS
array([[ 7.3470000...e-01,  2.6530000...e-01],
       [ 0.0000000...e+00,  1.0000000...e+00],
       [ 1.0000000...e-04, -7.7000000...e-02]])
>>> w # doctest: +ELLIPSIS
array([ 0.32168,  0.33767])
```

colour.RGB_luminance

colour.RGB_luminance(*RGB*, *primaries*, *whitepoint*)

Returns the *luminance Y* of given *RGB* components from given *primaries* and *whitepoint*.

Parameters

- **RGB** (array_like) – *RGB* chromaticity coordinate matrix.
- **primaries** (array_like, (3, 2)) – *Primaries* chromaticity coordinate matrix.
- **whitepoint** (array_like) – *Illuminant / whitepoint* chromaticity coordinates.

Returns *Luminance Y*.

Return type numeric or ndarray

Examples

```
>>> RGB = np.array([0.21959402, 0.06986677, 0.04703877])
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_luminance(RGB, p, whitepoint) # doctest: +ELLIPSIS
0.1230145...
```

colour.RGB_luminance_equation

colour.**RGB_luminance_equation**(primaries, whitepoint)

Returns the *luminance equation* from given *primaries* and *whitepoint*.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries chromaticity coordinates.
- **whitepoint** (array_like) – Illuminant / whitepoint chromaticity coordinates.

Returns *Luminance equation*.

Return type unicode

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> # Doctests skip for Python 2.x compatibility.
>>> RGB_luminance_equation(p, whitepoint) # doctest: +SKIP
'Y = 0.3439664...(R) + 0.7281660...(G) + -0.0721325...(B)'
```

RGB Colourspaces

colour

RGB_Colourspace(name, primaries, whitepoint)	Implements support for the <i>RGB</i> colourspaces dataset from colour.models.dataset.aces_rgb, etc....
RGB_COLOURSPACES	Aggregated <i>RGB</i> colourspaces.

colour.RGB_Colourspace

```
class colour.RGB_Colourspace(name, primaries, whitepoint, white-
                             point_name=None, RGB_to_XYZ_matrix=None,
                             XYZ_to_RGB_matrix=None, encoding_cctf=None, decod-
                             ing_cctf=None, use_derived_RGB_to_XYZ_matrix=False,
                             use_derived_XYZ_to_RGB_matrix=False)
```

Implements support for the *RGB* colourspaces dataset from `colour.models.dataset.aces_rgb`, etc. ...

Colour science literature related to *RGB* colourspaces and encodings defines their dataset using different degree of precision or rounding. While instances where a whitepoint is being defined with a value different than its canonical agreed one are rare, it is however very common to have normalised primary matrices rounded at different decimals. This can yield large discrepancies in computations.

Such an occurrence is the *V-Gamut* colourspace white paper, that defines the *V-Gamut* to *ITU-R BT.709* conversion matrix as follows:

```
[[ 1.806576 -0.695697 -0.110879]
 [-0.170090 1.305955 -0.135865]
 [-0.025206 -0.154468 1.179674]]
```

Computing this matrix using *ITU-R BT.709* colourspace derived normalised primary matrix yields:

```
[[ 1.8065736 -0.6956981 -0.1108786]
 [-0.1700890 1.3059548 -0.1358648]
 [-0.0252057 -0.1544678 1.1796737]]
```

The latter matrix is almost equals with the former, however performing the same computation using *IEC 61966-2-1:1999 sRGB* colourspace normalised primary matrix introduces severe disparities:

```
[[ 1.8063853 -0.6956147 -0.1109453]
 [-0.1699311 1.3058387 -0.1358616]
 [-0.0251630 -0.1544899 1.1797117]]
```

In order to provide support for both literature defined dataset and accurate computations enabling transformations without loss of precision, the `colour.RGB_Colourspace` class provides two sets of transformation matrices:

- Instantiation transformation matrices
- Derived transformation matrices

Upon instantiation, the `colour.RGB_Colourspace` class stores the given `RGB_to_XYZ_matrix` and `XYZ_to_RGB_matrix` arguments and also computes their derived counterpart using the `primaries` and `whitepoint` arguments.

Whether the initialisation or derived matrices are used in subsequent computations is dependent on the `colour.RGB_Colourspace.use_derived_RGB_to_XYZ_matrix` and `colour.RGB_Colourspace.use_derived_XYZ_to_RGB_matrix` attributes values.

Parameters

- **name** (unicode) – *RGB* colourspace name.
- **primaries** (array_like) – *RGB* colourspace primaries.
- **whitepoint** (array_like) – *RGB* colourspace whitepoint.
- **whitepoint_name** (unicode, optional) – *RGB* colourspace whitepoint name.

- **RGB_to_XYZ_matrix** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **XYZ_to_RGB_matrix** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **encoding_cctf** (object, optional) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF) that maps estimated tristimulus values in a scene to $R'G'B'$ video component signal value.
- **decoding_cctf** (object, optional) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF) that maps an $R'G'B'$ video component signal value to tristimulus values at the display.
- **use_derived_RGB_to_XYZ_matrix** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_XYZ_to_RGB_matrix** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

name

primaries

whitepoint

whitepoint_name

RGB_to_XYZ_matrix

XYZ_to_RGB_matrix

encoding_cctf

decoding_cctf

use_derived_RGB_to_XYZ_matrix

use_derived_XYZ_to_RGB_matrix

__str__()

__repr__()

use_derived_transformation_matrices()

chromatically_adapt()

copy()

Notes

- The normalised primary matrix defined by `colour.RGB_Colourspace.RGB_to_XYZ_matrix` attribute is treated as the prime matrix from which the inverse will be calculated as required by the internal derivation mechanism. This behaviour has been chosen in accordance with literature where commonly a *RGB* colourspace is defined by its normalised primary matrix as it is directly computed from the chosen primaries and whitepoint.

References

[[InternationalECommission99](#)], [[Pan14](#)]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_to_XYZ_matrix = np.identity(3)
>>> XYZ_to_RGB_matrix = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                               RGB_to_XYZ_matrix, XYZ_to_RGB_matrix)
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
>>> colourspace.RGB_to_XYZ_matrix # doctest: +ELLIPSIS
array([[ 9.5255239...e-01,  0.0000000...e+00,  9.3678631...e-05],
       [ 3.4396645...e-01,  7.2816609...e-01, -7.2132546...e-02],
       [ 0.0000000...e+00,  0.0000000...e+00,  1.0088251...e+00]])
>>> colourspace.XYZ_to_RGB_matrix # doctest: +ELLIPSIS
array([[ 1.0498110...e+00,  0.0000000...e+00, -9.7484540...e-05],
       [-4.9590302...e-01,  1.3733130...e+00,  9.8240036...e-02],
       [ 0.0000000...e+00,  0.0000000...e+00,  9.9125201...e-01]])
>>> colourspace.use_derived_RGB_to_XYZ_matrix = False
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_XYZ_to_RGB_matrix = False
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

`__init__(name, primaries, whitepoint, whitepoint_name=None, RGB_to_XYZ_matrix=None, XYZ_to_RGB_matrix=None, encoding_cctf=None, decoding_cctf=None, use_derived_RGB_to_XYZ_matrix=False, use_derived_XYZ_to_RGB_matrix=False)`
Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__(name, primaries, whitepoint[, ...])</code>	Initialize self.
<code>chromatically_adapt(whitepoint[, ...])</code>	Chromatically adapts the <i>RGB</i> colourspace <i>primaries</i> <i>xy</i> chromaticity coordinates from <i>RGB</i> colourspace whitepoint to reference whitepoint.
<code>copy()</code>	Returns a copy of the <i>RGB</i> colourspace.
<code>use_derived_transformation_matrices([usage])</code>	Enables or disables usage of both derived transformations matrices, the normalised primary matrix and its inverse in subsequent computations.

Attributes

<code>RGB_to_XYZ_matrix</code>	Getter and setter property for the transformation matrix from colourspace to <i>CIE XYZ</i> tristimulus values.
<code>XYZ_to_RGB_matrix</code>	Getter and setter property for the transformation matrix from <i>CIE XYZ</i> tristimulus values to colourspace.
<code>decoding_cctf</code>	Getter and setter property for the decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).
<code>encoding_cctf</code>	Getter and setter property for the encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).
<code>illuminant</code>	
<code>name</code>	Getter and setter property for the name.
<code>primaries</code>	Getter and setter property for the primaries.
<code>use_derived_RGB_to_XYZ_matrix</code>	Getter and setter property for whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
<code>use_derived_XYZ_to_RGB_matrix</code>	Getter and setter property for Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.
<code>whitepoint</code>	Getter and setter property for the whitepoint.
<code>whitepoint_name</code>	Getter and setter property for the whitepoint_name.

colour.RGB_COLOURSPACES

`colour.RGB_COLOURSPACES = CaseInsensitiveMapping({'ACES2065-1': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESp' ...})`
Aggregated *RGB* colourspaces.

`RGB_COLOURSPACES : CaseInsensitiveMapping`

Aliases:

- 'aces': `ACES_2065_1_COLOURSPACE.name`
- 'adobe1998': `ADOBE_RGB_1998_COLOURSPACE.name`
- 'prophoto': `PROPHOTO_RGB_COLOURSPACE.name`

`colour.models`

<code>ACES_2065_1_COLOURSPACE</code>	<i>ACES2065-1</i> colourspace, base encoding, used for exchange of full fidelity images and archiving.
<code>ACES_CC_COLOURSPACE</code>	<i>ACEScc</i> colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

Continued on next page

Table 205 – continued from previous page

ACES_CCT_COLOURSPACE	<i>ACEScct</i> colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.
ACES_PROXY_COLOURSPACE	<i>ACESproxy</i> colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management.
ACES_CG_COLOURSPACE	<i>ACEScg</i> colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.
ADOBE_RGB_1998_COLOURSPACE	<i>Adobe RGB (1998)</i> colourspace.
ADOBE_WIDE_GAMUT_RGB_COLOURSPACE	<i>Adobe Wide Gamut RGB</i> colourspace.
ALEXA_WIDE_GAMUT_COLOURSPACE	<i>ALEXA Wide Gamut</i> colourspace.
APPLE_RGB_COLOURSPACE	<i>Apple RGB</i> colourspace.
BEST_RGB_COLOURSPACE	<i>Best RGB</i> colourspace.
BETA_RGB_COLOURSPACE	<i>Beta RGB</i> colourspace.
BT470_525_COLOURSPACE	<i>ITU-R BT.470 - 525</i> colourspace.
BT470_625_COLOURSPACE	<i>ITU-R BT.470 - 625</i> colourspace.
BT709_COLOURSPACE	<i>ITU-R BT.709</i> colourspace.
BT2020_COLOURSPACE	<i>ITU-R BT.2020</i> colourspace.
CIE_RGB_COLOURSPACE	<i>CIE RGB</i> colourspace.
CINEMA_GAMUT_COLOURSPACE	<i>Cinema Gamut</i> colourspace.
COLOR_MATCH_RGB_COLOURSPACE	<i>ColorMatch RGB</i> colourspace.
DCDM_XYZ_COLOURSPACE	<i>DCDM XYZ</i> colourspace.
DCI_P3_COLOURSPACE	<i>DCI-P3</i> colourspace.
DCI_P3_P_COLOURSPACE	<i>DCI-P3+</i> colourspace.
DON_RGB_4_COLOURSPACE	<i>Don RGB 4</i> colourspace.
ECI_RGB_V2_COLOURSPACE	<i>ECI RGB v2</i> colourspace.
EKTA_SPACE_PS_5_COLOURSPACE	<i>Ekta Space PS 5</i> colourspace.
PROTUNE_NATIVE_COLOURSPACE	<i>Protune Native</i> colourspace.
MAX_RGB_COLOURSPACE	<i>Max RGB</i> colourspace.
NTSC_COLOURSPACE	<i>NTSC</i> colourspace.
P3_D65_COLOURSPACE	<i>P3-D65</i> colourspace.
PAL_SECAM_COLOURSPACE	<i>Pal/Secam</i> colourspace.
RED_COLOR_COLOURSPACE	<i>REDcolor</i> colourspace.
RED_COLOR_2_COLOURSPACE	<i>REDcolor2</i> colourspace.
RED_COLOR_3_COLOURSPACE	<i>REDcolor3</i> colourspace.
RED_COLOR_4_COLOURSPACE	<i>REDcolor4</i> colourspace.
RED_WIDE_GAMUT_RGB_COLOURSPACE	<i>REDWideGamutRGB</i> colourspace.
DRAGON_COLOR_COLOURSPACE	<i>DRAGONcolor</i> colourspace.
DRAGON_COLOR_2_COLOURSPACE	<i>DRAGONcolor2</i> colourspace.
ROMM_RGB_COLOURSPACE	<i>ROMM RGB</i> colourspace.
RIMM_RGB_COLOURSPACE	<i>RIMM RGB</i> colourspace.
ERIMM_RGB_COLOURSPACE	<i>ERIMM RGB</i> colourspace.
PROPHOTO_RGB_COLOURSPACE	<i>ProPhoto RGB</i> colourspace, an alias colourspace for <i>ROMM RGB</i> .
RUSSELL_RGB_COLOURSPACE	<i>Russell RGB</i> colourspace.
SMPTE_240M_COLOURSPACE	<i>SMPTE 240M</i> colourspace.
S_GAMUT_COLOURSPACE	<i>S-Gamut</i> colourspace.
S_GAMUT3_COLOURSPACE	<i>S-Gamut3</i> colourspace.

Continued on next page

Table 205 – continued from previous page

<code>S_GAMUT3_CINE_COLOURSPACE</code>	<i>S-Gamut3.Cine</i> colourspace.
<code>sRGB_COLOURSPACE</code>	<i>sRGB</i> colourspace.
<code>V_GAMUT_COLOURSPACE</code>	<i>V-Gamut</i> colourspace.
<code>XTREME_RGB_COLOURSPACE</code>	<i>Xtreme RGB</i> colourspace.

`colour.models.ACES_2065_1_COLOURSPACE`

`colour.models.ACES_2065_1_COLOURSPACE = RGB_Colourspace(ACES2065-1, [[ 7.34700000e-01, 2.65300000e-01], [ 0.00000000e+00, 0.00000000e+00], [ 0.00000000e+00, 0.00000000e+00], [ 0.00000000e+00, 0.00000000e+00], [ 0.00000000e+00, 0.00000000e+00])`
ACES2065-1 colourspace, base encoding, used for exchange of full fidelity images and archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

`ACES_2065_1_COLOURSPACE` : `RGB_Colourspace`

`colour.models.ACES_CC_COLOURSPACE`

`colour.models.ACES_CC_COLOURSPACE = RGB_Colourspace(ACEScc, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.045 ], [ 0.018, 0.018 ], [ 0.018, 0.018 ], [ 0.018, 0.018 ]])`
ACEScc colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

`ACES_CC_COLOURSPACE` : `RGB_Colourspace`

`colour.models.ACES_CCT_COLOURSPACE`

`colour.models.ACES_CCT_COLOURSPACE = RGB_Colourspace(ACEScct, [[ 0.713, 0.293], [ 0.165, 0.83 ], [ 0.128, 0.045 ], [ 0.018, 0.018 ], [ 0.018, 0.018 ], [ 0.018, 0.018 ]])`
ACEScct colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16]

`ACES_CCT_COLOURSPACE` : `RGB_Colourspace`

References

[Wik04d]

ADOBE_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ALEXA_WIDE_GAMUT_COLOURSPACE

colour.models.ALEXA_WIDE_GAMUT_COLOURSPACE = RGB_Colourspace(ALEXA Wide Gamut, [[0.684 , 0.313], [0.221 , 0.719], [0.023 , 0.075]],
ALEXA Wide Gamut colourspace.

References

[ARR12]

ALEXA_WIDE_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.APPLE_RGB_COLOURSPACE

colour.models.APPLE_RGB_COLOURSPACE = RGB_Colourspace(Apple RGB, [[0.625, 0.34], [0.28 , 0.595], [0.155, 0.845]],
Apple RGB colourspace.

References

[SBS99]

APPLE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BEST_RGB_COLOURSPACE

colour.models.BEST_RGB_COLOURSPACE = RGB_Colourspace(Best RGB, [[0.73519164, 0.26480836], [0.21533613, 0.78466387], [0.02966717, 0.97033283]],
Best RGB colourspace.

References

[Huta]

BEST_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BETA_RGB_COLOURSPACE

colour.models.BETA_RGB_COLOURSPACE = RGB_Colourspace(Beta RGB, [[0.6888, 0.3112], [0.1986, 0.7551], [0.1216, 0.8784]],
Beta RGB colourspace.

References

[Lin14]

BETA_RGB_COLOURSPACE : RGB_Colourspace

`colour.models.BT470_525_COLOURSPACE`

`colour.models.BT470_525_COLOURSPACE` = `RGB_Colourspace`(ITU-R BT.470 - 525, $\begin{bmatrix} 0.67 & 0.33 \\ 0.21 & 0.71 \\ 0.08 & 0.92 \end{bmatrix}$, $\begin{bmatrix} 0.2989 & 0.5876 & 0.0939 \end{bmatrix}$, $\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$)
ITU-R BT.470 - 525 colourspace.

References

[[InternationalTUnion98](#)]

BT470_525_COLOURSPACE : `RGB_Colourspace`

`colour.models.BT470_625_COLOURSPACE`

`colour.models.BT470_625_COLOURSPACE` = `RGB_Colourspace`(ITU-R BT.470 - 625, $\begin{bmatrix} 0.64 & 0.33 \\ 0.29 & 0.6 \\ 0.08 & 0.92 \end{bmatrix}$, $\begin{bmatrix} 0.2989 & 0.5876 & 0.0939 \end{bmatrix}$, $\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$)
ITU-R BT.470 - 625 colourspace.

References

[[InternationalTUnion98](#)]

BT470_625_COLOURSPACE : `RGB_Colourspace`

`colour.models.BT709_COLOURSPACE`

`colour.models.BT709_COLOURSPACE` = `RGB_Colourspace`(ITU-R BT.709, $\begin{bmatrix} 0.64 & 0.33 \\ 0.3 & 0.6 \\ 0.15 & 0.06 \end{bmatrix}$, $\begin{bmatrix} 0.2989 & 0.5876 & 0.0939 \end{bmatrix}$, $\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$)
ITU-R BT.709 colourspace.

References

[[InternationalTUnion15b](#)]

BT709_COLOURSPACE : `RGB_Colourspace`

`colour.models.BT2020_COLOURSPACE`

`colour.models.BT2020_COLOURSPACE` = `RGB_Colourspace`(ITU-R BT.2020, $\begin{bmatrix} 0.708 & 0.292 \\ 0.17 & 0.797 \\ 0.131 & 0.869 \end{bmatrix}$, $\begin{bmatrix} 0.2989 & 0.5876 & 0.0939 \end{bmatrix}$, $\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$)
ITU-R BT.2020 colourspace.

References

[[InternationalTUnion15a](#)]

BT2020_COLOURSPACE : `RGB_Colourspace`

`colour.models.CIE_RGB_COLOURSPACE`

`colour.models.CIE_RGB_COLOURSPACE` = `RGB_Colourspace`(CIE RGB, $\begin{bmatrix} 0.73474284 & 0.26525716 \\ 0.27377903 & 0.71722097 \\ 0.00446154 & 0.99553846 \end{bmatrix}$, $\begin{bmatrix} 0.2989 & 0.5876 & 0.0939 \end{bmatrix}$, $\begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$)
CIE RGB colourspace.

References

[FBH97]

CIE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.CINEMA_GAMUT_COLOURSPACE

`colour.models.CINEMA_GAMUT_COLOURSPACE = RGB_Colourspace(Cinema Gamut, [[ 0.74, 0.27], [ 0.17, 1.14], [ 0.08, 0.24]], [ 0.33, 0.72, 0.03], [ 0.4168, 0.7152, 0.0680], [ 0.224, 0.717, 0.059], [ 0.081, 0.357, 0.662])`
Cinema Gamut colourspace.

References

[Can14]

CINEMA_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.COLOR_MATCH_RGB_COLOURSPACE

`colour.models.COLOR_MATCH_RGB_COLOURSPACE = RGB_Colourspace(ColorMatch RGB, [[ 0.63 , 0.34 ], [ 0.295, 0.605]], [ 0.33, 0.72, 0.03], [ 0.4168, 0.7152, 0.0680], [ 0.224, 0.717, 0.059], [ 0.081, 0.357, 0.662])`
ColorMatch RGB colourspace.

References

[Lin14]

COLOR_MATCH_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DCDM_XYZ_COLOURSPACE

`colour.models.DCDM_XYZ_COLOURSPACE = RGB_Colourspace(DCDM XYZ, [[ 1., 0.], [ 0., 1.], [ 0., 0.]], [ 0.333333, 0.333333, 0.333333], [ 0.4061, 0.7142, 0.0897], [ 0.2829, 0.7172, 0.0000], [ 0.0000, 0.3441, 0.6559])`
DCDM XYZ colourspace.

References

[DigitalInitiatives07]

DCDM_XYZ_COLOURSPACE : RGB_Colourspace

colour.models.DCI_P3_COLOURSPACE

`colour.models.DCI_P3_COLOURSPACE = RGB_Colourspace(DCI-P3, [[ 0.68 , 0.32 ], [ 0.265, 0.69 ], [ 0.15 , 0.06 ]], [ 0.33, 0.72, 0.03], [ 0.4168, 0.7152, 0.0680], [ 0.224, 0.717, 0.059], [ 0.081, 0.357, 0.662])`
DCI-P3 colourspace.

References

[DigitalInitiatives07], [HewlettPDCompany09]

DCI_P3_COLOURSPACE : RGB_Colourspace

`colour.models.DCI_P3_P_COLOURSPACE`

`colour.models.DCI_P3_P_COLOURSPACE = RGB_Colourspace(DCI-P3+, [[0.74, 0.27], [0.22, 0.78], [0.09, -0.09]],
DCI-P3+ colourspace.`

References

[[Can14](#)]

DCI_P3_P_COLOURSPACE : RGB_Colourspace

`colour.models.DON_RGB_4_COLOURSPACE`

`colour.models.DON_RGB_4_COLOURSPACE = RGB_Colourspace(Don RGB 4, [[0.69612069, 0.29956897], [0.21468298, 0.78532702], [0.08070181, 0.01360221]],
Don RGB 4 colourspace.`

References

[[Hutb](#)]

DON_RGB_4_COLOURSPACE : RGB_Colourspace

`colour.models.ECI_RGB_V2_COLOURSPACE`

`colour.models.ECI_RGB_V2_COLOURSPACE = RGB_Colourspace(ECI RGB v2, [[0.67010309, 0.32989691], [0.20990566, 0.79009434], [0.08129014, 0.31870986]],
ECI RGB v2 colourspace.`

References

[[EuropeanCInitiative02](#)]

ECI_RGB_V2_COLOURSPACE : RGB_Colourspace

`colour.models.EKTA_SPACE_PS_5_COLOURSPACE`

`colour.models.EKTA_SPACE_PS_5_COLOURSPACE = RGB_Colourspace(Ekta Space PS 5, [[0.69473684, 0.30526316], [0.30526316, 0.69473684], [0.00, 0.00]],
Ekta Space PS 5 colourspace.`

References

[[Hol](#)]

EKTA_SPACE_PS_5_COLOURSPACE : RGB_Colourspace

`colour.models.PROTUNE_NATIVE_COLOURSPACE`

`colour.models.PROTUNE_NATIVE_COLOURSPACE = RGB_Colourspace(Protune Native, [[0.69848046, 0.19302645], [0.30151954, 0.80697355], [0.00, 0.00]],
Protune Native colourspace.`

References

[GDM16], [Man15]

PROTUNE_NATIVE_COLOURSPACE : RGB_Colourspace

colour.models.MAX_RGB_COLOURSPACE

colour.models.MAX_RGB_COLOURSPACE = RGB_Colourspace(Max_RGB, [[0.73413379, 0.26586621], [0.10039113, 0.89960887], [0.08094726, 0.71905274]],
Max RGB colourspace.

References

[Hut0]

MAX_RGB_COLOURSPACE : RGB_Colourspace

colour.models.NTSC_COLOURSPACE

colour.models.NTSC_COLOURSPACE = RGB_Colourspace(NTSC, [[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]], [0.3127, 0.6873],
NTSC colourspace.

References

[InternationalTUnion98]

NTSC_COLOURSPACE : RGB_Colourspace

colour.models.P3_D65_COLOURSPACE

colour.models.P3_D65_COLOURSPACE = RGB_Colourspace(P3-D65, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]], [0.4363, 0.5637],
P3-D65 colourspace.

P3_D65_COLOURSPACE : RGB_Colourspace

colour.models.PAL_SECAM_COLOURSPACE

colour.models.PAL_SECAM_COLOURSPACE = RGB_Colourspace(Pal/Secam, [[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]], [0.4363, 0.5637],
Pal/Secam colourspace.

References

[InternationalTUnion98]

PAL_SECAM_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_COLOURSPACE

colour.models.RED_COLOR_COLOURSPACE = RGB_Colourspace(REDcolor, [[0.70105856, 0.33018098], [0.29881132, 0.70118868], [0.00013989, 0.99986011]],
REDcolor colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_2_COLOURSPACE

colour.models.RED_COLOR_2_COLOURSPACE = RGB_Colourspace(REDcolor2, [[0.89740722, 0.33077623], [0.29602209, REDcolor2] colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_3_COLOURSPACE

colour.models.RED_COLOR_3_COLOURSPACE = RGB_Colourspace(REDcolor3, [[0.70259866, 0.33018559], [0.29578224, REDcolor3] colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_3_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_4_COLOURSPACE

colour.models.RED_COLOR_4_COLOURSPACE = RGB_Colourspace(REDcolor4, [[0.70259815, 0.3301851], [0.29578233, REDcolor4] colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_4_COLOURSPACE : RGB_Colourspace

colour.models.RED_WIDE_GAMUT_RGB_COLOURSPACE

colour.models.RED_WIDE_GAMUT_RGB_COLOURSPACE = RGB_Colourspace(REDWideGamutRGB, [[0.780308, 0.304253], [0.1 REDWideGamutRGB] colourspace.

References

[Man15], [Nat16], [SonyImageworks12]

RED_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_COLOURSPACE

colour.models.DRAGON_COLOR_COLOURSPACE = RGB_Colourspace(DRAGONcolor, [[0.75865589, 0.33035535], [0.294923
DRAGONcolor colourspace.

References

[Man15], [SonyImageworks12]

DRAGON_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_2_COLOURSPACE

colour.models.DRAGON_COLOR_2_COLOURSPACE = RGB_Colourspace(DRAGONcolor2, [[0.75865621, 0.33035584], [0.294923
DRAGONcolor2 colourspace.

References

[Man15], [SonyImageworks12]

DRAGON_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.ROMM_RGB_COLOURSPACE

colour.models.ROMM_RGB_COLOURSPACE = RGB_Colourspace(ROMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.596000
ROMM RGB colourspace.

References

[ANS03], [SWG00]

ROMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.RIMM_RGB_COLOURSPACE

colour.models.RIMM_RGB_COLOURSPACE = RGB_Colourspace(RIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.596000
RIMM RGB colourspace. In cases in which it is necessary to identify a specific precision level, the notation *RIMM8 RGB*, *RIMM12 RGB* and *RIMM16 RGB* is used.

References

[SWG00]

RIMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ERIMM_RGB_COLOURSPACE

colour.models.ERIMM_RGB_COLOURSPACE = RGB_Colourspace(ERIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.596000
ERIMM RGB colourspace.

[SWG00]

Our models: **PROPHOTO**, **RGB**, **COLOURSPACE**

```
colour.models.PROPHOTO_RGB_COLOURSPACE = RGB Colourspace(ProPhoto RGB, [[ 7.34700000e-01, 2.65300000e-01], [
```

References

colour.models.RUSSELL_RGB_COLOURSPACE

References

colour.models.SMPTE_240M_COLOURSPACE

References

colour.models.S_GAMUT_COLOURSPACE

References

373

colour.models.S_GAMUT3_COLOURSPACE

`colour.models.S_GAMUT3_COLOURSPACE = RGB_Colourspace(S-Gamut3, [[ 0.73 , 0.28 ], [ 0.14 , 0.855], [ 0.1 , -0.0555 ], [ 0.0156, 0.0187, 0.498 ]], S-Gamut3 colourspace.`

References

[[SonyCorporationc](#)]

S_GAMUT3_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT3_CINE_COLOURSPACE

`colour.models.S_GAMUT3_CINE_COLOURSPACE = RGB_Colourspace(S-Gamut3.Cine, [[ 0.766, 0.275], [ 0.225, 0.8 ], [ 0.0156, 0.0187, 0.498 ]], S-Gamut3.Cine colourspace.`

References

[[SonyCorporationa](#)]

S_GAMUT3_CINE_COLOURSPACE : RGB_Colourspace

colour.models.sRGB_COLOURSPACE

`colour.models.sRGB_COLOURSPACE = RGB_Colourspace(sRGB, [[ 0.64, 0.33], [ 0.3 , 0.6 ], [ 0.15, 0.06]], [ 0.3127, 0.6873 ], sRGB colourspace.`

References

[[InternationalECommission99](#)], [[InternationalTUnion15b](#)]

sRGB_COLOURSPACE : RGB_Colourspace

colour.models.V_GAMUT_COLOURSPACE

`colour.models.V_GAMUT_COLOURSPACE = RGB_Colourspace(V-Gamut, [[ 0.73 , 0.28 ], [ 0.165, 0.84 ], [ 0.1 , -0.0555 ], [ 0.0156, 0.0187, 0.498 ]], V-Gamut colourspace.`

References

[[Pan14](#)]

V_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.XTREME_RGB_COLOURSPACE

`colour.models.XTREME_RGB_COLOURSPACE = RGB_Colourspace(Xtreme RGB, [[ 1., 0.], [ 0., 1.], [ 0., 0.]], [ 0.3426, 0.3426, 0.3426 ], Xtreme RGB colourspace.`

References

[Hutd]

XTREME_RGB_COLOURSPACE : RGB_Colourspace

Colour Component Transfer Functions

colour

<code>encoding_cctf(value[, function])</code>	Encodes linear RGB values to non linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).
<code>ENCODING_CCTFS</code>	Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by <code>colour.LOG_ENCODING_CURVES</code> , <code>colour.OETFS</code> , <code>colour.EOTFS_REVERSE</code> attributes and 3 gamma encoding functions ($1 / 2.2$, $1 / 2.4$, $1 / 2.6$).
<code>decoding_cctf(value[, function])</code>	Decodes non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).
<code>DECODING_CCTFS</code>	Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by <code>colour.LOG_DECODING_CURVES</code> , <code>colour.EOTFS</code> , <code>colour.OETFS_REVERSE</code> attributes and 3 gamma decoding functions (2.2 , 2.4 , 2.6).

colour.encoding_cctf

`colour.encoding_cctf(value, function='sRGB', **kwargs)`

Encodes linear RGB values to non linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).

Parameters

- **value** (numeric or array_like) – Linear RGB values.
- **function** (unicode, optional) – {`colour.ENCODING_CCTFS`}, Computation function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for the relevant encoding CCTF of the `colour.ENCODING_CCTFS` attribute collection.

Warning: For *ITU-R BT.2100*, only the reverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, please refer to the `colour.oetf()` definition for the opto-electronic transfer functions (OETF / OECF).

Returns Non linear $R'G'B'$ values.

Return type numeric or ndarray

Examples

```
>>> encoding_cctf(0.18, function='PLog', log_reference=400)
... # doctest: +ELLIPSIS
0.3910068...
>>> encoding_cctf(0.18, function='ST 2084', L_p=1000)
... # doctest: +ELLIPSIS
0.1820115...
>>> encoding_cctf( # doctest: +ELLIPSIS
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.ENCODING_CCTFS

`colour.ENCODING_CCTFS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log`

Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by `colour.LOG_ENCODING_CURVES`, `colour.OETFs`, `colour.EOTFS_REVERSE` attributes and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).

Warning: For *ITU-R BT.2100*, only the reverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, please refer to the `colour.OETFs` attribute for the opto-electronic transfer functions (OETF / OECF).

ENCODING_CCTFS [CaseInsensitiveMapping] {`colour.LOG_ENCODING_CURVES`, `colour.OETFs`, `colour.EOTFS_REVERSE`}

colour.decoding_cctf

`colour.decoding_cctf(value, function='Cineon', **kwargs)`

Decodes non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).

Parameters

- **value** (numeric or array_like) – Non-linear $R'G'B'$ values.
- **function** (unicode, optional) – {`colour.DECODING_CCTFS`}, Computation function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for the relevant decoding CCTF of the `colour.DECODING_CCTFS` attribute collection.

Warning: For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, please refer to the `colour.oetf_reverse()` definition for the reverse opto-electronic transfer functions (OETF / OECF).

Returns Linear RGB values.

Return type numeric or ndarray

Examples

```
>>> decoding_cctf(0.391006842619746, function='PLog', log_reference=400)
... # doctest: +ELLIPSIS
0.1...
>>> decoding_cctf(0.182011532850008, function='ST 2084', L_p=1000)
... # doctest: +ELLIPSIS
0.1...
>>> decoding_cctf( # doctest: +ELLIPSIS
...     0.461356129500442, function='ITU-R BT.1886')
0.1...
```

colour.DECODING_CCTFS

`colour.DECODING_CCTFS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log`

Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by `colour.LOG_DECODING_CURVES`, `colour.EOTFS`, `colour.OETFS_REVERSE` attributes and 3 gamma decoding functions (2.2, 2.4, 2.6).

Warning: For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, please refer to the `colour.OETFS_REVERSE` attribute for the reverse opto-electronic transfer functions (OETF / OECF).

Notes

- The order by which this attribute is defined and updated is critically important to ensure that *ITU-R BT.2100* definitions are reciprocal.

DECODING_CCTFS [CaseInsensitiveMapping] {`colour.LOG_DECODING_CURVES`, `colour.EOTFS`,
 `colour.OETFS_REVERSE`}

Opto-Electronic Transfer Functions

`colour`

<code>oetf(value[, function])</code>	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).
<code>OETFS</code>	Supported opto-electrical transfer functions (OETFs / OECFs).
<code>oetf_reverse(value[, function])</code>	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given reverse opto-electronic transfer function (OETF / OECF).
<code>OETFS_REVERSE</code>	Supported reverse opto-electrical transfer functions (OETFs / OECFs).

colour.oetf

`colour.oetf(value, function='sRGB', **kwargs)`

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'sRGB', 'ARIB STD-B67', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}, Opto-electronic transfer function (OETF / OECF).

Other Parameters

- **E_clip** (numeric, optional) – {`colour.models.oetf_RIMMRGB()`}, Maximum exposure level.
- **I_max** (numeric, optional) – {`colour.models.oetf_ROMMRGB()`, `colour.models.oetf_RIMMRGB()`}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **L_p** (numeric, optional) – {`colour.models.oetf_ST2084()`}, Display peak luminance cd/m^2 .
- **is_12_bits_system** (bool) – {`colour.models.oetf_BT2020()`}, ITU-R BT.2020 α and β constants are used if system is not 12-bit.
- **r** (numeric, optional) – {`colour.models.oetf_ARIBSTDB67()`}, Video level corresponding to reference white level.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> oetf(0.18) # doctest: +ELLIPSIS
0.4613561...
>>> oetf(0.18, function='ITU-R BT.2020') # doctest: +ELLIPSIS
0.4090077...
>>> oetf(0.18, function='ST 2084', L_p=1000)
... # doctest: +ELLIPSIS
0.1820115...
```

colour.OETFS

`colour.OETFS = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'DICOM GSDF': ..., 'ITU-R BT.2020': ..., 'ITU-R`
Supported opto-electrical transfer functions (OETFs / OECFs).

`OETFS [CaseInsensitiveMapping] {'sRGB', 'ARIB STD-B67', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R`
`BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'ProPhoto RGB', 'RIMM`
`RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}`

colour.oetf_reverse

`colour.oetf_reverse(value, function='sRGB', **kwargs)`

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given reverse opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'sRGB', 'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}, Reverse opto-electronic transfer function (OETF / OECF).

Other Parameters **r** (numeric, optional) – {`colour.models.oetf_ARIBSTDB67()`}, Video level corresponding to reference white level.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> oetf_reverse(0.461356129500442) # doctest: +ELLIPSIS
0.1...
>>> oetf_reverse( # doctest: +ELLIPSIS
...     0.409007728864150, function='ITU-R BT.601')
0.1...
```

colour.OETFS_REVERSE

`colour.OETFS_REVERSE = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'ITU-R BT.2100 HLD': ..., 'ITU-R BT.2100 PQ': ...})`
Supported reverse opto-electrical transfer functions (OETFs / OECFs).

OETFS_REVERSE [CaseInsensitiveMapping] {'sRGB', 'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}

`colour.models`

<code>oetf_ARIBSTDB67(E[, r, constants])</code>	Defines <i>ARIB STD-B67 (Hybrid Log-Gamma)</i> opto-electrical transfer function (OETF / OECF).
<code>oetf_reverse_ARIBSTDB67(E_p[, r, constants])</code>	Defines <i>ARIB STD-B67 (Hybrid Log-Gamma)</i> reverse opto-electrical transfer function (OETF / OECF).
<code>oetf_DICOMGSDF(L[, out_int])</code>	Defines the <i>DICOM - Grayscale Standard Display Function</i> opto-electronic transfer function (OETF / OECF).
<code>oetf_BT2020(E[, is_12_bits_system, constants])</code>	Defines <i>Recommendation ITU-R BT.2020</i> opto-electrical transfer function (OETF / OECF).
<code>oetf_BT2100_HLG(E)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> opto-electrical transfer function (OETF / OECF).
<code>oetf_reverse_BT2100_HLG(E)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> reverse opto-electrical transfer function (OETF / OECF).

Continued on next page

Table 208 – continued from previous page

<code>oetf_BT2100_PQ(E)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> opto-electrical transfer function (OETF / OECF).
<code>oetf_reverse_BT2100_PQ(E_p)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> reverse opto-electrical transfer function (OETF / OECF).
<code>oetf_BT601(L)</code>	Defines <i>Recommendation ITU-R BT.601-7</i> opto-electronic transfer function (OETF / OECF).
<code>oetf_reverse_BT601(E)</code>	Defines <i>Recommendation ITU-R BT.601-7</i> reverse opto-electronic transfer function (OETF / OECF).
<code>oetf_BT709(L)</code>	Defines <i>Recommendation ITU-R BT.709-6</i> opto-electronic transfer function (OETF / OECF).
<code>oetf_reverse_BT709(V)</code>	Defines <i>Recommendation ITU-R BT.709-6</i> reverse opto-electronic transfer function (OETF / OECF).
<code>oetf_ProPhotoRGB(X[, bit_depth, out_int])</code>	Defines the <i>ROMM RGB</i> encoding opto-electronic transfer function (OETF / OECF).
<code>oetf_RIMMRGB(X[, bit_depth, out_int, E_clip])</code>	Defines the <i>RIMM RGB</i> encoding opto-electronic transfer function (OETF / OECF).
<code>oetf_ROMMRGB(X[, bit_depth, out_int])</code>	Defines the <i>ROMM RGB</i> encoding opto-electronic transfer function (OETF / OECF).
<code>oetf_SMPTE240M(L_c)</code>	Defines <i>SMPTE 240M</i> opto-electrical transfer function (OETF / OECF).
<code>oetf_ST2084(C[, L_p, constants])</code>	Defines <i>SMPTE ST 2084:2014</i> optimised perceptual opto-electronic transfer function (OETF / OECF).
<code>oetf_sRGB(L)</code>	Defines the <i>sRGB</i> colourspace opto-electronic transfer function (OETF / OECF).
<code>oetf_reverse_sRGB(V)</code>	Defines the <i>sRGB</i> colourspace reverse opto-electronic transfer function (OETF / OECF).

colour.models.oetf_ARIBSTDB67

`colour.models.oetf_ARIBSTDB67(E, r=0.5, constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991073})`

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).

Parameters

- **E** (numeric or array_like) – Voltage normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.
- **r** (numeric, optional) – Video level corresponding to reference white level.
- **constants** (`Structure`, optional) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Resulting non-linear signal E' .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_ARIBSTDB67(0.18) # doctest: +ELLIPSIS
0.2121320...
```

colour.models.oetf_reverse_ARIBSTDB67

`colour.models.oetf_reverse_ARIBSTDB67( $E_p$ ,  $r=0.5$ , constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991073})`

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* reverse opto-electrical transfer function (OETF / OECF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E' .
- **r** (numeric, optional) – Video level corresponding to reference white level.
- **constants** (`Structure`, optional) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Voltage E normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_reverse_ARIBSTDB67(0.212132034355964) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.oetf_DICOMGSDF

colour.models.oetf_DICOMGSDF(*L*, out_int=False)

Defines the *DICOM - Grayscale Standard Display Function* opto-electronic transfer function (OETF / OECF).

Parameters

- *L* (numeric or array_like) – Luminance *L*.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Just-Noticeable Difference (JND) Index, *j*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

References

[[NationalEMAssociation04](#)]

Examples

```
>>> oetf_DICOMGSDF(130.0662) # doctest: +ELLIPSIS
0.5004862...
>>> oetf_DICOMGSDF(130.0662, out_int=True)
512
```

colour.models.oetf_BT2020

colour.models.oetf_BT2020(*E*, is_12_bits_system=False, constants={'alpha': <function <lambda>>, 'beta': <function <lambda>>})

Defines *Recommendation ITU-R BT.2020* opto-electrical transfer function (OETF / OECF).

Parameters

- *E* (numeric or array_like) – Voltage *E* normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera colour channel R, G, B.
- **is_12_bits_system** (bool) – *BT.709* *alpha* and *beta* constants are used if system is not 12-bit.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting non-linear signal E' .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[[InternationalTUnion15a](#)]

Examples

```
>>> oetf_BT2020(0.18) # doctest: +ELLIPSIS
0.4090077...
```

colour.models.oetf_BT2100_HLG

colour.models.oetf_BT2100_HLG(E)

Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear *HLG* signal value.

Parameters E (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Returns E' is the resulting non-linear signal R', G', B' .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[[Bor17](#)], [[InternationalTUnion16](#)]

Examples

```
>>> oetf_BT2100_HLG(0.18 / 12) # doctest: +ELLIPSIS
0.2121320...
```

colour.models.oetf_reverse_BT2100_HLG

colour.models.oetf_reverse_BT2100_HLG(E)

Defines *Recommendation ITU-R BT.2100 Reference HLG* reverse opto-electrical transfer function (OETF / OECF).

Parameters E_p (numeric or array_like) – E' is the resulting non-linear signal R', G', B' .

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> oetf_reverse_BT2100_HLG(0.212132034355964) # doctest: +ELLIPSIS
0.0149999...
```

colour.models.oetf_BT2100_PQ

colour.models.oetf_BT2100_PQ(E)

Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear PQ signal value.

Parameters E (numeric or array_like) – $E = R_S, G_S, B_S; Y_S$; or I_S is the signal determined by scene light and scaled by camera exposure.

Returns E' is the resulting non-linear signal (R', G', B').

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> oetf_BT2100_PQ(0.1) # doctest: +ELLIPSIS
0.7247698...
```

colour.models.oetf_reverse_BT2100_PQ

colour.models.oetf_reverse_BT2100_PQ(*E_p*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-electrical transfer function (OETF / OECF).

Parameters *E_p* (numeric or array_like) – E' is the resulting non-linear signal (R' , G' , B').

Returns $E = R_S, G_S, B_S; Y_S; \text{or } I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> oetf_reverse_BT2100_PQ(0.724769816665726) # doctest: +ELLIPSIS
0.0999999...
```

colour.models.oetf_BT601

colour.models.oetf_BT601(*L*)

Defines *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF / OECF).

Parameters *L* (numeric or array_like) – Luminance *L* of the image.

Returns Corresponding electrical signal *E*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[[InternationalTUnion11b](#)]

Examples

```
>>> oetf_BT601(0.18) # doctest: +ELLIPSIS
0.4090077...
```

colour.models.oetf_reverse_BT601

colour.models.oetf_reverse_BT601(*E*)

Defines *Recommendation ITU-R BT.601-7* reverse opto-electronic transfer function (OETF / OECF).

Parameters *E* (numeric or array_like) – Electrical signal *E*.

Returns Corresponding *luminance L* of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[[InternationalTUnion11b](#)]

Examples

```
>>> oetf_reverse_BT601(0.409007728864150) # doctest: +ELLIPSIS
0.1...
```

colour.models.oetf_BT709

colour.models.oetf_BT709(*L*)

Defines *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF / OECF).

Parameters *L* (numeric or array_like) – *Luminance L* of the image.

Returns Corresponding electrical signal *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[[InternationalTUnion15b](#)]

Examples

```
>>> oetf_BT709(0.18) # doctest: +ELLIPSIS
0.4090077...
```

colour.models.oetf_reverse_BT709

colour.models.oetf_reverse_BT709(*V*)

Defines *Recommendation ITU-R BT.709-6* reverse opto-electronic transfer function (OETF / OECF).

Parameters *V* (numeric or array_like) – Electrical signal *V*.

Returns Corresponding *luminance L* of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>V</i>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>L</i>	[0, 1]	[0, 1]

References

[[InternationalTUnion15b](#)]

Examples

```
>>> oetf_reverse_BT709(0.409007728864150) # doctest: +ELLIPSIS
0.1...
```

colour.models.oetf_ProPhotoRGB

colour.models.oetf_ProPhotoRGB(*X*, *bit_depth*=8, *out_int*=False)

Defines the *ROMM RGB* encoding opto-electronic transfer function (OETF / OECF).

Parameters

- *X* (numeric or array_like) – Linear data X_{ROMM} .
- *bit_depth* (int, optional) – Bit depth used for conversion.
- *out_int* (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear data X'_{ROMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
<i>X</i>	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> oetf_ROMMRGB(0.18) # doctest: +ELLIPSIS
0.3857114...
>>> oetf_ROMMRGB(0.18, out_int=True)
98
```

colour.models.oetf_RIMMRGB

`colour.models.oetf_RIMMRGB(X, bit_depth=8, out_int=False, E_clip=2.0)`

Defines the *RIMM* RGB encoding opto-electronic transfer function (OETF / OECF).

RIMM RGB encoding non-linearity is based on that specified by *Recommendation ITU-R BT.709-6*.

Parameters

- X*** (numeric or array_like) – Linear data X_{RIMM} .
- bit_depth*** (int, optional) – Bit depth used for conversion.
- out_int*** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- E_clip*** (numeric, optional) – Maximum exposure level.

Returns Non-linear data X'_{RIMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> oetf_RIMMRGB(0.18) # doctest: +ELLIPSIS
0.2916737...
>>> oetf_RIMMRGB(0.18, out_int=True)
74
```

colour.models.oetf_ROMMRGB

colour.models.oetf_ROMMRGB(*X*, *bit_depth*=8, *out_int*=False)

Defines the *ROMM* RGB encoding opto-electronic transfer function (OETF / OECF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ROMM} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear data X'_{ROMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> oetf_ROMMRGB(0.18) # doctest: +ELLIPSIS
0.3857114...
>>> oetf_ROMMRGB(0.18, out_int=True)
98
```


colour.models.oetf_SMPTE240M

colour.models.oetf_SMPTE240M(*L_c*)

Defines *SMPTE 240M* opto-electrical transfer function (OETF / OECF).

Parameters *L_c* (numeric or array_like) – Light input L_c to the reference camera normalised to the system reference white.

Returns Video signal output V_c of the reference camera normalised to the system reference white.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_c	[0, 1]	[0, 1]

References

[Society of Motion Picture and Television Engineers 99]

Examples

```
>>> oetf_SMPTE240M(0.18) # doctest: +ELLIPSIS
0.4022857...
```

colour.models.oetf_ST2084

colour.models.oetf_ST2084(*C*, *L_p*=10000, constants={'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'm_1': 0.1593017578125, 'm_2': 78.84375})

Defines *SMPTE ST 2084:2014* optimised perceptual opto-electronic transfer function (OETF / OECF).

Parameters

- **C** (numeric or array_like) – Target optical output C in cd/m^2 of the ideal reference display.
- **L_p** (numeric, optional) – System peak luminance cd/m^2 , this parameter should stay at its default $10000\text{cd}/\text{m}^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (*Structure*, optional) – *SMPTE ST 2084:2014* constants.

Returns Color value abbreviated as N , that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.

Return type numeric or ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

Domain	Scale - Reference	Scale - 1
C	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
N	[0, 1]	[0, 1]

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

References

[Mil14], [SocietyofPhotographicEngineers14]

Examples

```
>>> oetf_ST2084(100) # doctest: +ELLIPSIS
0.5080784...
```

colour.models.oetf_sRGB

colour.models.oetf_sRGB(*L*)

Defines the *sRGB* colourspace opto-electronic transfer function (OETF / OECF).

Parameters *L* (numeric or array_like) – *Luminance L* of the image.

Returns Corresponding electrical signal *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalCommission99], [InternationalTUnion15b]

Examples

```
>>> oetf_sRGB(0.18) # doctest: +ELLIPSIS
0.4613561...
```

colour.models.oetf_reverse_sRGB

colour.models.oetf_reverse_sRGB(*V*)

Defines the *sRGB* colourspace reverse opto-electronic transfer function (OETF / OECF).

Parameters *V* (numeric or array_like) – Electrical signal *V*.

Returns Corresponding *luminance* *L* of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>V</i>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>L</i>	[0, 1]	[0, 1]

References

[[InternationalECommission99](#)], [[InternationalTUnion15b](#)]

Examples

```
>>> oetf_reverse_sRGB(0.461356129500442) # doctest: +ELLIPSIS
0.1...
```

Ancillary Objects

colour

<code>gamma_function(a[, exponent, ...])</code>	Defines a typical gamma encoding / decoding function.
<code>linear_function(a)</code>	Defines a typical linear encoding / decoding function, essentially a pass-through function.

colour.gamma_function

colour.gamma_function(*a*, *exponent*=1, *negative_number_handling*='Indeterminate')

Defines a typical gamma encoding / decoding function.

Parameters

- **a** (numeric or array_like) – Array to encode / decode.
- **exponent** (numeric or array_like, optional) – Encoding / decoding exponent.
- **negative_number_handling** (unicode, optional) – {'Indeterminate', 'Mirror', 'Preserve', 'Clamp'}, Defines the behaviour for a negative numbers and / or the definition return value:
 - *Indeterminate*: The behaviour will be indeterminate and definition return value might contain *nans*.
 - *Mirror*: The definition return value will be mirrored around abscissa and ordinate axis, i.e. Blackmagic Design: Davinci Resolve behaviour.
 - *Preserve*: The definition will preserve any negative number in a, i.e. The Foundry Nuke behaviour.
 - *Clamp*: The definition will clamp any negative number in a to 0.

Returns Encoded / decoded array.

Return type numeric or ndarray

Raises `ValueError` – If the negative number handling method is not defined.

Examples

```
>>> gamma_function(0.18, 2.2) # doctest: +ELLIPSIS
0.0229932...
>>> gamma_function(-0.18, 2.0) # doctest: +ELLIPSIS
0.0323999...
>>> gamma_function(-0.18, 2.2)
nan
>>> gamma_function(-0.18, 2.2, 'Mirror') # doctest: +ELLIPSIS
-0.0229932...
>>> gamma_function(-0.18, 2.2, 'Preserve') # doctest: +ELLIPSIS
-0.1...
>>> gamma_function(-0.18, 2.2, 'Clamp') # doctest: +ELLIPSIS
0.0
```

colour.linear_function

`colour.linear_function(a)`

Defines a typical linear encoding / decoding function, essentially a pass-through function.

Parameters **a** (numeric or array_like) – Array to encode / decode.

Returns Encoded / decoded array.

Return type numeric or ndarray

Examples

```
>>> linear_function(0.18)
0.18
```

Electro-Optical Transfer Functions

colour

<code>eotf(value[, function])</code>	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).
<code>EOTFS</code>	Supported electro-optical transfer functions (EOTFs / EOCFs).
<code>eotf_reverse(value[, function])</code>	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given reverse electro-optical transfer function (EOTF / EOCF).
<code>EOTFS_REVERSE</code>	Supported reverse electro-optical transfer functions (EOTFs / EOCFs).

colour.eotf

`colour.eotf(value, function='ITU-R BT.1886', **kwargs)`

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'ITU-R BT.1886', 'DCDM', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}, Electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **E_clip** (numeric, optional) – {`colour.models.eotf_RIMMRGB()`}, Maximum exposure level.
- **I_max** (numeric, optional) – {`colour.models.eotf_ROMMRGB()`, `colour.models.eotf_RIMMRGB()`}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **L_B** (numeric, optional) – {`colour.models.eotf_BT1886()`, `colour.models.eotf_BT2100_HLG()`}, Screen luminance for black.
- **L_W** (numeric, optional) – {`colour.models.eotf_BT1886()`, `colour.models.eotf_BT2100_HLG()`}, Screen luminance for white.
- **L_p** (numeric, optional) – {`colour.models.eotf_ST2084()`}, Display peak luminance cd/m^2 .
- **gamma** (numeric, optional) – {`colour.models.eotf_BT2100_HLG()`}, System gamma value, 1.2 at the nominal display peak luminance of $1000\text{cd}/\text{m}^2$.
- **is_12_bits_system** (bool) – {`colour.models.eotf_BT2020()`}, ITU-R BT.2020 α and β constants are used if system is not 12-bit.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> eotf(0.461356129500442) # doctest: +ELLIPSIS
0.1...
>>> eotf(0.409007728864150, function='ITU-R BT.2020')
... # doctest: +ELLIPSIS
0.1...
>>> eotf(0.182011532850008, function='ST 2084', L_p=1000)
... # doctest: +ELLIPSIS
0.1...
```

colour.EOTFS

`colour.EOTFS = CaseInsensitiveMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2020': ...})`
Supported electro-optical transfer functions (EOTFs / EOCFs).

EOTFS [CaseInsensitiveMapping] {'DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ProPhoto RGB', 'RIMM RGB', 'ROMM RGB', 'SMPTE 240M', 'ST 2084'}

colour.eotf_reverse

`colour.eotf_reverse(value, function='ITU-R BT.1886', **kwargs)`

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'ITU-R BT.1886', 'DCDM', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}, Reverse electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **L_B** (numeric, optional) – {`colour.models.eotf_reverse_BT1886()`, `colour.models.eotf_reverse_BT2100_HLG()`}, Screen luminance for black.
- **L_W** (numeric, optional) – {`colour.models.eotf_reverse_BT1886()`, `colour.models.eotf_reverse_BT2100_HLG()`}, Screen luminance for white.
- **gamma** (numeric, optional) – {`colour.models.eotf_BT2100_HLG()`}, System gamma value, 1.2 at the nominal display peak luminance of 1000cd/m^2 .
- **out_int** (bool, optional) – {`colour.models.eotf_reverse_DCDM()`}, Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> eotf_reverse(0.11699185725296059) # doctest: +ELLIPSIS
0.4090077...
>>> eotf_reverse( # doctest: +ELLIPSIS
...               0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.EOTFS_REVERSE

`colour.EOTFS_REVERSE = CaseInsensitiveMapping({'DCDM': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})`
Supported reverse electro-optical transfer functions (EOTFs / EOCFs).

EOTFS_REVERSE [CaseInsensitiveMapping] {'DCDM', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}

`colour.models`

<code>eotf_DCDM(XYZ_p[, in_int])</code>	Defines the <i>DCDM</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_reverse_DCDM(XYZ[, out_int])</code>	Defines the <i>DCDM</i> reverse electro-optical transfer function (EOTF / EOCF).
<code>eotf_DICOMGSDF(J[, in_int])</code>	Defines the <i>DICOM - Grayscale Standard Display Function</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_BT1886(V[, L_B, L_W])</code>	Defines <i>Recommendation ITU-R BT.1886</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_reverse_BT1886(L[, L_B, L_W])</code>	Defines <i>Recommendation ITU-R BT.1886</i> reverse electro-optical transfer function (EOTF / EOCF).
<code>eotf_BT2020(E_p[, is_12_bits_system, constants])</code>	Defines <i>Recommendation ITU-R BT.2020</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_BT2100_HLG(E_p[, L_B, L_W, gamma])</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_reverse_BT2100_HLG(F_D[, L_B, L_W, gamma])</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> reverse electro-optical transfer function (EOTF / EOCF).
<code>eotf_BT2100_PQ(E_p)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_reverse_BT2100_PQ(F_D)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> reverse electro-optical transfer function (EOTF / EOCF).
<code>eotf_ProPhotoRGB(X_p[, bit_depth, in_int])</code>	Defines the <i>ROMM RGB</i> encoding electro-optical transfer function (EOTF / EOCF).
<code>eotf_RIMMRGB(X_p[, bit_depth, in_int, E_clip])</code>	Defines the <i>RIMM RGB</i> encoding electro-optical transfer function (EOTF / EOCF).
<code>eotf_ROMMRGB(X_p[, bit_depth, in_int])</code>	Defines the <i>ROMM RGB</i> encoding electro-optical transfer function (EOTF / EOCF).
<code>eotf_SMPTE240M(V_r)</code>	Defines <i>SMPTE 240M</i> electro-optical transfer function (EOTF / EOCF).
<code>eotf_ST2084(N[, L_p, constants])</code>	Defines <i>SMPTE ST 2084:2014</i> optimised perceptual electro-optical transfer function (EOTF / EOCF).

colour.models.eotf_DCDM

colour.models.eotf_DCDM(*XYZ_p*, *in_int=False*)

Defines the *DCDM* electro-optical transfer function (EOTF / EOCF).

Parameters

- **XYZ_p** (numeric or array_like) – Non-linear *CIE XYZ'* tristimulus values.
- **in_int** (`bool`, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns *CIE XYZ* tristimulus values.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
XYZ_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_DCDM(0.11281860951766724) # doctest: +ELLIPSIS
0.18...
>>> eotf_DCDM(462, in_int=True) # doctest: +ELLIPSIS
0.18...
```

colour.models.eotf_reverse_DCDM

colour.models.eotf_reverse_DCDM(*XYZ*, *out_int=False*)

Defines the *DCDM* reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **XYZ** (numeric or array_like) – *CIE XYZ* tristimulus values.
- **out_int** (`bool`, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear *CIE XYZ'* tristimulus values.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
XYZ_p	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_reverse_DCDM(0.18) # doctest: +ELLIPSIS
0.1128186...
>>> eotf_reverse_DCDM(0.18, out_int=True)
462
```

colour.models.eotf_DICOMGSDF

colour.models.eotf_DICOMGSDF(*J*, *in_int=False*)

Defines the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF / EOCF).

Parameters

- J** (numeric or array_like) – Just-Noticeable Difference (JND) Index, *j*.
- in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Corresponding *luminance L*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[NationalEMAssociation04]

Examples

```
>>> eotf_DICOMGSDF(0.500486263438448) # doctest: +ELLIPSIS
130.0628647...
>>> eotf_DICOMGSDF(512, in_int=True) # doctest: +ELLIPSIS
130.0652840...
```

colour.models.eotf_BT1886

colour.models.eotf_BT1886(*V*, *L_B*=0, *L_W*=1)

Defines *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF / EOCF).

Parameters

- **V** (numeric or array_like) – Input video signal level (normalised, black at $V = 0$, to white at $V = 1$. For content mastered per *Recommendation ITU-R BT.709*, 10-bit digital code values D map into values of V per the following equation: $V = (D - 64)/876$
- **L_B** (numeric, optional) – Screen luminance for black.
- **L_W** (numeric, optional) – Screen luminance for white.

Returns Screen luminance in cd/m^2 .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion11a]

Examples

```
>>> eotf_BT1886(0.409007728864150) # doctest: +ELLIPSIS
0.1169918...
```

colour.models.eotf_reverse_BT1886

`colour.models.eotf_reverse_BT1886(L, L_B=0, L_W=1)`

Defines *Recommendation ITU-R BT.1886* reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **L** (numeric or array_like) – Screen luminance in cd/m^2 .
- **L_B** (numeric, optional) – Screen luminance for black.
- **L_W** (numeric, optional) – Screen luminance for white.

Returns Input video signal level (normalised, black at $V = 0$, to white at $V = 1$).

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[[InternationalTUnion11a](#)]

Examples

```
>>> eotf_reverse_BT1886(0.11699185725296059) # doctest: +ELLIPSIS
0.4090077...
```

colour.models.eotf_BT2020

`colour.models.eotf_BT2020(E_p, is_12_bits_system=False, constants={'alpha': <function <lambda>>, 'beta': <function <lambda>>})`

Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E' .
- **is_12_bits_system** (bool) – *BT.709* α and β constants are used if system is not 12-bit.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting voltage E .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[[InternationalTUnion15a](#)]

Examples

```
>>> eotf_BT2020(0.705515089922121) # doctest: +ELLIPSIS
0.4999999...
```

colour.models.eotf_BT2100_HLG

colour.models.eotf_BT2100_HLG(*E_p*, *L_B*=0, *L_W*=1000, *gamma*=None)

Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *HLG* signal into display light.

Parameters

- **E_p** (numeric or array_like) – E' denotes a non-linear colour value R' , G' , B' or L' , M' , S' in *HLG* space.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2 .
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000\text{cd}/\text{m}^2$.

Returns Luminance F_D of a displayed linear component R_D , G_D , B_D or Y_D or I_D , in cd/m^2 .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> eotf_BT2100_HLG(0.212132034355964) # doctest: +ELLIPSIS
6.4760398...
```

colour.models.eotf_reverse_BT2100_HLG

colour.models.eotf_reverse_BT2100_HLG(F_D , $L_B=0$, $L_W=1000$, $\gamma=None$)

Defines Recommendation ITU-R BT.2100 Reference HLG reverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **F_D** (numeric or array_like) – Luminance F_D of a displayed linear component R_D, G_D, B_D or Y_D or I_D , in cd/m^2 .
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2 .
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **γ** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in HLG space.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> eotf_reverse_BT2100_HLG(6.476039825649814) # doctest: +ELLIPSIS
0.2121320...
```

colour.models.eotf_BT2100_PQ

colour.models.eotf_BT2100_PQ(E_p)

Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *PQ* signal into display light.

Parameters E_p (numeric or array_like) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space $[0, 1]$.

Returns F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D , in cd/m^2 .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	$[0, 1]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
F_D	$[0, 1]$	$[0, 1]$

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> eotf_BT2100_PQ(0.724769816665726) # doctest: +ELLIPSIS
779.9883608...
```

colour.models.eotf_reverse_BT2100_PQ

colour.models.eotf_reverse_BT2100_PQ(F_D)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse electro-optical transfer function (EOTF / EOCF).

Parameters F_D (numeric or array_like) – F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D , in cd/m^2 .

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space $[0, 1]$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	$[0, 1]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> eotf_reverse_BT2100_PQ(779.988360834085370) # doctest: +ELLIPSIS
0.7247698...
```

colour.models.eotf_ProPhotoRGB

`colour.models.eotf_ProPhotoRGB(X_p, bit_depth=8, in_int=False)`

Defines the *ROMM* RGB encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM} .
- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **in_int** (`bool`, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Linear data X_{ROMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> eotf_ROMMRGB(0.385711424751138) # doctest: +ELLIPSIS
0.1...
>>> eotf_ROMMRGB(98, in_int=True) # doctest: +ELLIPSIS
0.1...
```

colour.models.eotf_RIMMRGB

colour.models.eotf_RIMMRGB(*X_p*, *bit_depth*=8, *in_int*=False, *E_clip*=2.0)

Defines the *RIMM* RGB encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{RIMM} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **E_clip** (numeric, optional) – Maximum exposure level.

Returns Linear data X_{RIMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> eotf_RIMMRGB(0.291673732475746) # doctest: +ELLIPSIS
0.1...
>>> eotf_RIMMRGB(74, in_int=True) # doctest: +ELLIPSIS
0.1...
```


colour.models.eotf_ROMMRGB

colour.models.eotf_ROMMRGB(X_p , bit_depth=8, in_int=False)

Defines the *ROMM RGB* encoding electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Linear data X_{ROMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> eotf_ROMMRGB(0.385711424751138) # doctest: +ELLIPSIS
0.1...
>>> eotf_ROMMRGB(98, in_int=True) # doctest: +ELLIPSIS
0.1...
```

colour.models.eotf_SMPTE240M

colour.models.eotf_SMPTE240M(V_r)

Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).

Parameters V_r (numeric or array_like) – Video signal level V_r driving the reference reproducer normalised to the system reference white.

Returns Light output L_r from the reference reproducer normalised to the system reference white.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

References

[SocietyoMPaTEngineers99]

Examples

```
>>> eotf_SMPTE240M(0.402285796753870) # doctest: +ELLIPSIS
0.1...
```

colour.models.eotf_ST2084

`colour.models.eotf_ST2084(N, L_p=10000, constants={'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'm_1': 0.1593017578125, 'm_2': 78.84375})`

Defines *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF / EOCF).

This perceptual quantizer (PQ) has been modeled by Dolby Laboratories using *Barten (1999)* contrast sensitivity function.

Parameters

- **N** (numeric or array_like) – Color value abbreviated as *N*, that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.
- **L_p** (numeric, optional) – System peak luminance cd/m^2 , this parameter should stay at its default $10000cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (*Structure*, optional) – *SMPTE ST 2084:2014* constants.

Returns Target optical output *C* in cd/m^2 of the ideal reference display.

Return type numeric or ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

Domain	Scale - Reference	Scale - 1
N	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
C	[0, 1]	[0, 1]

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

References

[Mil14], [SocietyoMPaTEngineers14]

Examples

```
>>> eotf_ST2084(0.508078421517399) # doctest: +ELLIPSIS
100.0000000...
```

Opto-Optical Transfer Functions

colour

<code>eotf(value[, function])</code>	Maps relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).
<code>OOTFS</code>	Supported opto-optical transfer functions (OOTFs / OOCFs).
<code>eotf_reverse(value[, function])</code>	Maps relative display linear light to scene linear light using given reverse opto-optical transfer function (OOTF / OOCF).
<code>OOTFS_REVERSE</code>	Supported reverse opto-optical transfer functions (OOTFs / OOCFs).

colour.eotf

`colour.eotf(value, function='ITU-R BT.2100 PQ', **kwargs)`

Maps relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}
Opto-optical transfer function (OOTF / OOCF).

Returns Luminance of a displayed linear component.

Return type numeric or ndarray

Examples

```
>>> ootf(0.1) # doctest: +ELLIPSIS
779.9883608...
>>> ootf(0.1, function='ITU-R BT.2100 HLG') # doctest: +ELLIPSIS
63.0957344...
```

colour.OOTFS

`colour.OOTFS = CaseInsensitiveMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})`
Supported opto-optical transfer functions (OOTFs / OOCFs).

OOTFS [CaseInsensitiveMapping] {'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}

colour.ootf_reverse

`colour.ootf_reverse(value, function='ITU-R BT.2100 PQ', **kwargs)`
Maps relative display linear light to scene linear light using given reverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – {'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'} Reverse opto-optical transfer function (OOTF / OOCF).

Other Parameters

- **L_B** (numeric, optional) – {`colour.models.ootf_reverse_BT2100_HLG()`}, L_B is the display luminance for black in cd/m^2 .
- **L_W** (numeric, optional) – {`colour.models.ootf_reverse_BT2100_HLG()`}, L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – {`colour.models.ootf_reverse_BT2100_HLG()`}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns Luminance of scene linear light.

Return type numeric or ndarray

Examples

```
>>> ootf_reverse(779.988360834115840) # doctest: +ELLIPSIS
0.1000000...
>>> ootf_reverse( # doctest: +ELLIPSIS
...     63.095734448019336, function='ITU-R BT.2100 HLG')
0.1000000...
```

colour.OOTFS_REVERSE

`colour.OOTFS_REVERSE = CaseInsensitiveMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})`
Supported reverse opto-optical transfer functions (OOTFs / OOCFs).

OOTFS_REVERSE [CaseInsensitiveMapping] {'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}

`colour.models`

<code>ootf_BT2100_HLG(E[, L_B, L_W, gamma])</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> opto-optical transfer function (OOTF / OOCF).
<code>ootf_reverse_BT2100_HLG(F_D[, L_B, L_W, gamma])</code>	Defines <i>Recommendation ITU-R BT.2100 Reference HLG</i> reverse opto-optical transfer function (OOTF / OOCF).
<code>ootf_BT2100_PQ(E)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> opto-optical transfer function (OOTF / OOCF).
<code>ootf_reverse_BT2100_PQ(F_D)</code>	Defines <i>Recommendation ITU-R BT.2100 Reference PQ</i> reverse opto-optical transfer function (OOTF / OOCF).

`colour.models.ootf_BT2100_HLG`

`colour.models.ootf_BT2100_HLG(E, L_B=0, L_W=1000, gamma=None)`

Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters

- **E** (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2 .
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of 1000cd/m^2 .

Returns F_D is the luminance of a displayed linear component $R_D, G_D, \text{or } B_D$, in cd/m^2 .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> ootf_BT2100_HLG(0.1) # doctest: +ELLIPSIS
63.0957344...
```

colour.models.ootf_reverse_BT2100_HLG

colour.models.ootf_reverse_BT2100_HLG(F_D , $L_B=0$, $L_W=1000$, $\gamma=None$)

Defines Recommendation ITU-R BT.2100 Reference HLG reverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component R_D, G_D , or B_D , in cd/m^2 .
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2 .
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> ootf_reverse_BT2100_HLG(63.095734448019336) # doctest: +ELLIPSIS
0.1000000...
```

colour.models.ootf_BT2100_PQ

colour.models.ootf_BT2100_PQ(E)

Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters E (numeric or array_like) – $E = R_S, G_S, B_S; Y_S; \text{or } I_S$ is the signal determined by scene light and scaled by camera exposure.

Returns F_D is the luminance of a displayed linear component ($R_D, G_D, B_D; Y_D; \text{or } I_D$).

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> ootf_BT2100_PQ(0.1) # doctest: +ELLIPSIS
779.9883608...
```

colour.models.ootf_reverse_BT2100_PQ

colour.models.ootf_reverse_BT2100_PQ(F_D)

Defines *Recommendation ITU-R BT.2100 Reference PQ* reverse opto-optical transfer function (OOTF / OOCF).

Parameters F_D (numeric or array_like) – F_D is the luminance of a displayed linear component ($R_D, G_D, B_D; Y_D; \text{or } I_D$).

Returns $E = R_S, G_S, B_S; Y_S; \text{or } I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion16]

Examples

```
>>> ootf_reverse_BT2100_PQ(779.988360834115840) # doctest: +ELLIPSIS
0.1000000...
```

Log Encoding and Decoding Curves

colour

<code>log_encoding_curve(value[, curve])</code>	Encodes linear-light values to $R'G'B'$ video component signal value using given <i>log</i> curve.
<code>LOG_ENCODING_CURVES</code>	Supported <i>log</i> encoding curves.
<code>log_decoding_curve(value[, curve])</code>	Decodes $R'G'B'$ video component signal value to linear-light values using given <i>log</i> curve.
<code>LOG_DECODING_CURVES</code>	Supported <i>log</i> decoding curves.

colour.log_encoding_curve

`colour.log_encoding_curve(value, curve='Cineon', **kwargs)`

Encodes linear-light values to $R'G'B'$ video component signal value using given *log* curve.

Parameters

- **value** (numeric or array_like) – Value.
- **curve** (unicode, optional) – {'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}, Computation curve.

Other Parameters

- **EI** (int, optional) – {colour.models.log_encoding_ALEXALogC()}, Ei.
- **E_clip** (numeric, optional) – {colour.models.log_encoding_ERIMMRGB()}, Maximum exposure limit.
- **E_min** (numeric, optional) – {colour.models.log_encoding_ERIMMRGB()}, Minimum exposure limit.
- **I_max** (numeric, optional) – {colour.models.log_encoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.

- **bit_depth** (*unicode, optional*) – {colour.models.log_encoding_ACESproxy(), colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, {8, 10, 12}, Bit depth used for conversion, *ACESproxy* uses {10, 12}.
- **black_offset** (*numeric or array_like*) – {colour.models.log_encoding_Cineon(), colour.models.log_encoding_Panalog(), colour.models.log_encoding_REDLog(), colour.models.log_encoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Density per code value.
- **firmware** (*unicode, optional*) – {colour.models.log_encoding_ALEXALogC()}, {'SUP 3.x', 'SUP 2.x'}, Alexa firmware version.
- **in_reflection** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, Whether the light level x to a camera is reflection.
- **linear_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Linear reference.
- **log_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Log reference.
- **out_legal** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2(), colour.models.log_encoding_SLog3()}, Whether the non-linear Sony S-Log, Sony S-Log2 or Sony S-Log3 data y is encoded in legal range.
- **negative_gamma** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Negative gamma.
- **method** (*unicode, optional*) – {colour.models.log_encoding_ALEXALogC()}, {'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_encoding_curve(0.18) # doctest: +ELLIPSIS
0.4573196...
>>> log_encoding_curve(0.18, curve='ACESc') # doctest: +ELLIPSIS
0.4135884...
>>> log_encoding_curve(0.18, curve='PLog', log_reference=400)
... # doctest: +ELLIPSIS
0.3910068...
>>> log_encoding_curve(0.18, curve='S-Log') # doctest: +ELLIPSIS
0.3849708...
```

colour.LOG_ENCODING_CURVES

colour.LOG_ENCODING_CURVES = CaseInsensitiveMapping({'ACESc': ..., 'ACESct': ..., 'ACESproxy': ..., 'ALEXALog C': ...})
Supported *log* encoding curves.

```
LOG_ENCODING_CURVES [CaseInsensitiveMapping] {'ACESc', 'ACESct', 'ACESproxy', 'ALEXALog C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}
```

colour.log_decoding_curve

`colour.log_decoding_curve(value, curve='Cineon', **kwargs)`

Decodes $R'G'B'$ video component signal value to linear-light values using given log curve.

Parameters

- **value** (numeric or array_like) – Value.
- **curve** (unicode, optional) – {'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}, Computation curve.

Other Parameters

- **EI** (int, optional) – {colour.models.log_decoding_ALEXALogC()}, Ei.
- **E_clip** (numeric, optional) – {colour.models.log_decoding_ERIMMRGB()}, Maximum exposure limit.
- **E_min** (numeric, optional) – {colour.models.log_decoding_ERIMMRGB()}, Minimum exposure limit.
- **I_max** (numeric, optional) – {colour.models.log_decoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **bit_depth** (int, optional) – {colour.models.log_decoding_ACESproxy(), colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, {8, 10, 12}, Bit depth used for conversion, *ACESproxy* uses {10, 12}.
- **black_offset** (numeric or array_like) – {colour.models.log_decoding_Cineon(), colour.models.log_decoding_Panalog(), colour.models.log_decoding_REDLog(), colour.models.log_decoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (numeric or array_like) – {colour.models.log_decoding_PivotedLog()}, Density per code value.
- **firmware** (unicode, optional) – {colour.models.log_decoding_ALEXALogC()}, {'SUP 3.x', 'SUP 2.x'}, Alexa firmware version.
- **in_legal** (bool, optional) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2(), colour.models.log_decoding_SLog3()}, Whether the non-linear *Sony S-Log*, *Sony S-Log2* or *Sony S-Log3* data y is encoded in legal range.
- **linear_reference** (numeric or array_like) – {colour.models.log_decoding_PivotedLog()}, Linear reference.
- **log_reference** (numeric or array_like) – {colour.models.log_decoding_PivotedLog()}, Log reference.
- **negative_gamma** (numeric or array_like) – {colour.models.log_decoding_PivotedLog()}, Negative gamma.
- **out_reflection** (bool, optional) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, Whether the light level x to a camera is reflection.
- **method** (unicode, optional) – {colour.models.log_decoding_ALEXALogC()}, {'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_decoding_curve(0.457319613085418) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_curve(0.413588402492442, curve='ACESc')
... # doctest: +ELLIPSIS
0.1...
>>> log_decoding_curve(0.391006842619746, curve='PLog', log_reference=400)
... # doctest: +ELLIPSIS
0.1...
>>> log_decoding_curve(0.376512722254600, curve='S-Log')
... # doctest: +ELLIPSIS
0.1...
```

colour.LOG_DECODING_CURVES

`colour.LOG_DECODING_CURVES` = `CaseInsensitiveMapping`({'ACESc': ..., 'ACESct': ..., 'ACESproxy': ..., 'ALEX

Supported log decoding curves.

`LOG_DECODING_CURVES` [`CaseInsensitiveMapping`] {'ACESc', 'ACESct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}

`colour.models`

<code>log_encoding_ACESc(lin_AP1)</code>	Defines the <i>ACESc</i> colourspace log encoding / opto-electronic transfer function.
<code>log_decoding_ACESc(ACESc)</code>	Defines the <i>ACESc</i> colourspace log decoding / electro-optical transfer function.
<code>log_encoding_ACESct(lin_AP1[, constants])</code>	Defines the <i>ACESct</i> colourspace log encoding / opto-electronic transfer function.
<code>log_decoding_ACESct(ACESct[, constants])</code>	Defines the <i>ACESct</i> colourspace log decoding / electro-optical transfer function.
<code>log_encoding_ACESproxy(lin_AP1[, bit_depth, ...])</code>	Defines the <i>ACESproxy</i> colourspace log encoding curve / opto-electronic transfer function.
<code>log_decoding_ACESproxy(ACESproxy[, ...])</code>	Defines the <i>ACESproxy</i> colourspace log decoding curve / electro-optical transfer function.
<code>log_encoding_ALEXALogC(x[, firmware, method, EI])</code>	Defines the <i>ALEXA Log C</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_ALEXALogC(t[, firmware, method, EI])</code>	Defines the <i>ALEXA Log C</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_CanonLog2(x[, bit_depth, ...])</code>	Defines the <i>Canon Log 2</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_CanonLog2(clog2[, bit_depth, ...])</code>	Defines the <i>Canon Log 2</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_CanonLog3(x[, bit_depth, ...])</code>	Defines the <i>Canon Log 3</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_CanonLog3(clog3[, bit_depth, ...])</code>	Defines the <i>Canon Log 3</i> log decoding curve / electro-optical transfer function.

Continued on next page

Table 215 – continued from previous page

<code>log_encoding_CanonLog(x[, bit_depth, ...])</code>	Defines the <i>Canon Log</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_CanonLog(clog[, bit_depth, ...])</code>	Defines the <i>Canon Log</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_Cineon(x[, black_offset])</code>	Defines the <i>Cineon</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_Cineon(y[, black_offset])</code>	Defines the <i>Cineon</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_ERIMMRGB(X[, bit_depth, ...])</code>	Defines the <i>ERIMM RGB</i> log encoding curve / opto-electronic transfer function (OETF / OECF).
<code>log_decoding_ERIMMRGB(X_p[, bit_depth, ...])</code>	Defines the <i>ERIMM RGB</i> log decoding curve / electro-optical transfer function (EOTF / EOCF).
<code>log_encoding_Log3G10(x[, legacy_curve])</code>	Defines the <i>Log3G10</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_Log3G10(y[, legacy_curve])</code>	Defines the <i>Log3G10</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_Log3G12(x)</code>	Defines the <i>Log3G12</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_Log3G12(y)</code>	Defines the <i>Log3G12</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_Panalog(x[, black_offset])</code>	Defines the <i>Panalog</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_Panalog(y[, black_offset])</code>	Defines the <i>Panalog</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_PivotedLog(x[, log_reference, ...])</code>	Defines the <i>Josh Pines</i> style <i>Pivoted Log</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_PivotedLog(y[, log_reference, ...])</code>	Defines the <i>Josh Pines</i> style <i>Pivoted Log</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_Protune(x)</code>	Defines the <i>Protune</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_Protune(y)</code>	Defines the <i>Protune</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_REDLog(x[, black_offset])</code>	Defines the <i>REDLog</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_REDLog(y[, black_offset])</code>	Defines the <i>REDLog</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_REDLogFilm(x[, black_offset])</code>	Defines the <i>REDLogFilm</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_REDLogFilm(y[, black_offset])</code>	Defines the <i>REDLogFilm</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_SLog(x[, bit_depth, out_legal, ...])</code>	Defines the <i>Sony S-Log</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_SLog(y[, bit_depth, in_legal, ...])</code>	Defines the <i>Sony S-Log</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_SLog2(x[, bit_depth, ...])</code>	Defines the <i>Sony S-Log2</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_SLog2(y[, bit_depth, in_legal, ...])</code>	Defines the <i>Sony S-Log2</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_SLog3(x[, bit_depth, ...])</code>	Defines the <i>Sony S-Log3</i> log encoding curve / opto-electronic transfer function.

Continued on next page

Table 215 – continued from previous page

<code>log_decoding_SLog3(y[, bit_depth, in_legal, ...])</code>	Defines the <i>Sony S-Log3</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_VLog(L_in[, bit_depth, ...])</code>	Defines the <i>Panasonic V-Log</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_VLog(V_out[, bit_depth, ...])</code>	Defines the <i>Panasonic V-Log</i> log decoding curve / electro-optical transfer function.
<code>log_encoding_ViperLog(x)</code>	Defines the <i>Viper Log</i> log encoding curve / opto-electronic transfer function.
<code>log_decoding_ViperLog(y)</code>	Defines the <i>Viper Log</i> log decoding curve / electro-optical transfer function.

`colour.models.log_encoding_ACEScc`

`colour.models.log_encoding_ACEScc(lin_AP1)`

Defines the *ACEScc* colourspace log encoding / opto-electronic transfer function.

Parameters `lin_AP1` (numeric or array_like) – *lin_AP1* value.

Returns *ACEScc* non-linear value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<code>lin_AP1</code>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>ACEScc</i>	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_encoding_ACEScc(0.18) # doctest: +ELLIPSIS
0.4135884...
```

`colour.models.log_decoding_ACEScc`

`colour.models.log_decoding_ACEScc(ACEScc)`

Defines the *ACEScc* colourspace log decoding / electro-optical transfer function.

Parameters *ACEScc* (numeric or array_like) – *ACEScc* non-linear value.

Returns *lin_AP1* value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
ACEScc	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_decoding_ACEScc(0.413588402492442) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_ACEScct

```
colour.models.log_encoding_ACEScct(lin_AP1, constants={'A': 10.5402377416545, 'B':
0.0729055341958355, 'X_BRK': 0.0078125, 'Y_BRK':
0.155251141552511})
```

Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.

Parameters

- **lin_AP1** (numeric or array_like) – *lin_AP1* value.
- **constants** (*Structure*, optional) – *ACEScct* constants.

Returns *ACEScct* non-linear value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyC

Examples

```
>>> log_encoding_ACEScct(0.18) # doctest: +ELLIPSIS
0.4135884...
```

colour.models.log_decoding_ACEScct

```
colour.models.log_decoding_ACEScct(ACEScct, constants={'A': 10.5402377416545, 'B':
0.0729055341958355, 'X_BRK': 0.0078125, 'Y_BRK':
0.155251141552511})
```

Defines the *ACEScct* colourspace log decoding / electro-optical transfer function.

Parameters

- **ACEScct** (numeric or array_like) – *ACEScct* non-linear value.
- **constants** (*Structure*, optional) – *ACEScct* constants.

Returns *lin_AP1* value.

Return type numeric or ndarray

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16], [TheAoMPAaSciencesScienceaTCouncilAcademyC

Notes

Domain	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Examples

```
>>> log_decoding_ACEScct(0.413588402492442) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_ACESproxy

```
colour.models.log_encoding_ACESproxy(lin_AP1, bit_depth=10, out_int=False, constants={10:
    {'CV_max': 940, 'CV_min': 64, 'mid_CV_offset': 425,
     'mid_log_offset': 2.5, 'steps_per_stop': 50}, 12: {'CV_max':
     3760, 'CV_min': 256, 'mid_CV_offset': 1700, 'mid_log_offset':
     2.5, 'steps_per_stop': 200}})
```

Defines the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.

Parameters

- **lin_AP1** (numeric or array_like) – *lin_AP1* value.
- **bit_depth** (int, optional) – {10, 12}, *ACESproxy* bit depth.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- **constants** (Structure, optional) – *ACESproxy* constants.

Returns *ACESproxy* non-linear value.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_encoding_ACESproxy(0.18) # doctest: +ELLIPSIS
0.4164222...
>>> log_encoding_ACESproxy(0.18, out_int=True)
426
```


colour.models.log_decoding_ACESproxy

```
colour.models.log_decoding_ACESproxy(ACESproxy, bit_depth=10, in_int=False, constants={10:
    {'CV_max': 940, 'CV_min': 64, 'mid_CV_offset': 425,
     'mid_log_offset': 2.5, 'steps_per_stop': 50}, 12: {'CV_max':
     3760, 'CV_min': 256, 'mid_CV_offset': 1700, 'mid_log_offset':
     2.5, 'steps_per_stop': 200}})
```

Defines the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.

Parameters

- **ACESproxy** (numeric or array_like) – *ACESproxy* non-linear value.
- **bit_depth** (int, optional) – {10, 12}, *ACESproxy* bit depth.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **constants** (Structure, optional) – *ACESproxy* constants.

Returns *lin_AP1* value.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a],
 [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_decoding_ACESproxy(0.416422287390029) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_ACESproxy(426, in_int=True) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_ALEXALogC

`colour.models.log_encoding_ALEXALogC(x, firmware='SUP 3.x', method='Linear Scene Exposure Factor', EI=800)`

Defines the *ALEXA Log C* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data *x*.
- **firmware** (unicode, optional) – {'SUP 3.x', 'SUP 2.x'}, Alexa firmware version.
- **method** (unicode, optional) – {'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}, Conversion method.
- **EI** (`int`, optional) – *E_i*.

Returns *ALEXA Log C* encoded data *t*.

Return type numeric or ndarray

References

[ARR12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_ALEXALogC(0.18) # doctest: +ELLIPSIS
0.3910068...
```

colour.models.log_decoding_ALEXALogC

`colour.models.log_decoding_ALEXALogC(t, firmware='SUP 3.x', method='Linear Scene Exposure Factor', EI=800)`

Defines the *ALEXA Log C* log decoding curve / electro-optical transfer function.

Parameters

- **t** (numeric or array_like) – *ALEXA Log C* encoded data *t*.
- **firmware** (unicode, optional) – {'SUP 3.x', 'SUP 2.x'}, Alexa firmware version.
- **method** (unicode, optional) – {'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}, Conversion method.
- **EI** (`int`, optional) – *E_i*.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[ARR12]

Examples

```
>>> log_decoding_ALEXALogC(0.391006832034084) # doctest: +ELLIPSIS
0.18...
```

colour.models.log_encoding_CanonLog2

`colour.models.log_encoding_CanonLog2(x, bit_depth=10, out_legal=True, in_reflection=True)`
 Defines the *Canon Log 2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .
- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **out_legal** (`bool`, optional) – Whether the *Canon Log 2* non-linear data is encoded in legal range.
- **in_reflection** (`bool`, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log 2* non-linear data.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_encoding_CanonLog2(0.18) * 100 # doctest: +ELLIPSIS
39.8254694...
```

colour.models.log_decoding_CanonLog2

colour.models.log_decoding_CanonLog2(clog2, bit_depth=10, in_legal=True, out_reflection=True)
Defines the *Canon Log 2* log decoding curve / electro-optical transfer function.

Parameters

- **clog2** (numeric or array_like) – *Canon Log 2* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_decoding_CanonLog2(39.825469498316735 / 100) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_CanonLog3

`colour.models.log_encoding_CanonLog3(x, bit_depth=10, out_legal=True, in_reflection=True)`
 Defines the *Canon Log 3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .
- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **out_legal** (`bool`, optional) – Whether the *Canon Log 3* non-linear data is encoded in legal range.
- **in_reflection** (`bool`, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log 3* non-linear data.

Return type numeric or ndarray

Notes

- Introspection of the grafting points by Shaw, N. (2018) shows that the *Canon Log 3* IDT was likely derived from its encoding curve as the later is grafted at ± 0.014 :

```
>>> clog3 = 0.04076162
>>> (clog3 - 0.073059361) / 2.3069815
-0.014000000000000002
>>> clog3 = 0.105357102
>>> (clog3 - 0.073059361) / 2.3069815
0.013999999999999997
```

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog3	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_encoding_CanonLog3(0.18) * 100 # doctest: +ELLIPSIS
34.3389369...
```

colour.models.log_decoding_CanonLog3

`colour.models.log_decoding_CanonLog3(clog3, bit_depth=10, in_legal=True, out_reflection=True)`
Defines the *Canon Log 3* log decoding curve / electro-optical transfer function.

Parameters

- **clog3** (numeric or array_like) – *Canon Log 3* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog3	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_decoding_CanonLog3(34.338936938868677 / 100) # doctest: +ELLIPSIS
0.1800000...
```

colour.models.log_encoding_CanonLog

`colour.models.log_encoding_CanonLog(x, bit_depth=10, out_legal=True, in_reflection=True)`
Defines the *Canon Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the *Canon Log* non-linear data is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns *Canon Log* non-linear data.

Return type numeric or ndarray

References

[Tho12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_CanonLog(0.18) * 100 # doctest: +ELLIPSIS
34.3389651...
```

colour.models.log_decoding_CanonLog

`colour.models.log_decoding_CanonLog(clog, bit_depth=10, in_legal=True, out_reflection=True)`
 Defines the *Canon Log* log decoding curve / electro-optical transfer function.

Parameters

- **clog** (numeric or array_like) – *Canon Log* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the *Canon Log* non-linear data is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Tho12]

Examples

```
>>> log_decoding_CanonLog(34.338965172606912 / 100) # doctest: +ELLIPSIS
0.17999999...
```

colour.models.log_encoding_Cineon

colour.models.**log_encoding_Cineon**(*x*, *black_offset*=0.0107977516232771)
Defines the *Cineon* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data *x*.
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data *y*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Cineon(0.18) # doctest: +ELLIPSIS
0.4573196...
```

colour.models.log_decoding_Cineon

colour.models.**log_decoding_Cineon**(*y*, *black_offset*=0.0107977516232771)
Defines the *Cineon* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data *y*.

- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Cineon(0.457319613085418) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_ERIMMRGB

`colour.models.log_encoding_ERIMMRGB( $X$ ,  $bit\_depth=8$ ,  $out\_int=False$ ,  $E\_min=0.001$ ,  $E\_clip=316.2$ )`
 Defines the *ERIMM* RGB log encoding curve / opto-electronic transfer function (OETF / OECF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ERIMM} .
- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **out_int** (`bool`, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- **E_min** (numeric, optional) – Minimum exposure limit.
- **E_clip** (numeric, optional) – Maximum exposure limit.

Returns Non-linear data X'_{ERIMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

- * This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_encoding_ERIMMRGB(0.18) # doctest: +ELLIPSIS
0.4100523...
>>> log_encoding_ERIMMRGB(0.18, out_int=True)
105
```

colour.models.log_decoding_ERIMMRGB

`colour.models.log_decoding_ERIMMRGB( $X_p$ , bit_depth=8, in_int=False, E_min=0.001, E_clip=316.2)`

Defines the *ERIMM* RGB log decoding curve / electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ERIMM} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **E_min** (numeric, optional) – Minimum exposure limit.
- **E_clip** (numeric, optional) – Maximum exposure limit.

Returns Linear data X_{ERIMM} .

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

- * This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_decoding_ERIMMRGB(0.410052389492129) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_ERIMMRGB(105, in_int=True) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_Log3G10

colour.models.log_encoding_Log3G10(*x*, *legacy_curve=False*)

Defines the *Log3G10* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data *x*.
- **legacy_curve** (bool, optional) – Whether to use the v1 *Log3G10* log encoding curve. Default is *False*.

Returns Non-linear data *y*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

- The v1 *Log3G10* log encoding curve is the one used in *REDCINE-X beta 42*. *Resolve 12.5.2* also uses the v1 curve. *RED* is planning to use v2 *Log3G10* log encoding curve in the release version of the *RED SDK*. Use the *legacy_curve=True* argument to switch to the v1 curve for compatibility with the current (as of September 21, 2016) *RED SDK*.
- The intent of the v1 *Log3G10* log encoding curve is that zero maps to zero, 0.18 maps to 1/3, and 10 stops above 0.18 maps to 1.0. The name indicates this in a similar way to the naming conventions of *Sony HyperGamma* curves.

The constants used in the functions do not in fact quite hit these values, but rather than use corrected constants, the functions here use the official *RED* values, in order to match the output of the *RED SDK*.

For those interested, solving for constants which exactly hit 1/3 and 1.0 yields the following values:

```
B = 25 * (np.sqrt(4093.0) - 3) / 9
A = 1 / np.log10(B * 184.32 + 1)
```

where the function takes the form:

```
Log3G10(x) = A * np.log10(B * x + 1)
```

Similarly for *Log3G12*, the values which hit exactly 1/3 and 1.0 are:

```
B = 25 * (np.sqrt(16381.0) - 3) / 9
A = 1 / np.log10(B * 737.28 + 1)
```

References

[Nat16]

Examples

```
>>> log_encoding_Log3G10(0.18, legacy_curve=True) # doctest: +ELLIPSIS
0.3333336...
>>> log_encoding_Log3G10(0.0) # doctest: +ELLIPSIS
0.0915514...
```

colour.models.log_decoding_Log3G10

colour.models.**log_decoding_Log3G10**(y, legacy_curve=False)

Defines the *Log3G10* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data *y*.
- **legacy_curve** (bool, optional) – Whether to use the v1 *Log3G10* log encoding curve. Default is *False*.

Returns Linear data *x*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_decoding_Log3G10(1.0 / 3, legacy_curve=True) # doctest: +ELLIPSIS
0.1799994...
>>> log_decoding_Log3G10(1.0) # doctest: +ELLIPSIS
184.3223476...
```

colour.models.log_encoding_Log3G12

colour.models.log_encoding_Log3G12(*x*)

Defines the *Log3G12* log encoding curve / opto-electronic transfer function.

Parameters *x* (numeric or array_like) – Linear data *x*.

Returns Non-linear data *y*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>x</i>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>y</i>	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_encoding_Log3G12(0.18) # doctest: +ELLIPSIS
0.3333326...
```

colour.models.log_decoding_Log3G12

colour.models.log_decoding_Log3G12(*y*)

Defines the *Log3G12* log decoding curve / electro-optical transfer function.

Parameters *y* (numeric or array_like) – Non-linear data *y*.

Returns Linear data *x*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_decoding_Log3G12(1.0 / 3) # doctest: +ELLIPSIS
0.1800015...
```

colour.models.log_encoding_Panalog

`colour.models.log_encoding_Panalog(x, black_offset=0.04077184461038074)`

Defines the *Panalog* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y .

Return type numeric or ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Panalog(0.18) # doctest: +ELLIPSIS
0.3745767...
```

colour.models.log_decoding_Panalog

`colour.models.log_decoding_Panalog(y, black_offset=0.04077184461038074)`
 Defines the *Panalog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y .
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x .

Return type numeric or ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Panalog(0.374576791382298) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_PivotedLog

`colour.models.log_encoding_PivotedLog(x, log_reference=445, linear_reference=0.18, negative_gamma=0.6, density_per_code_value=0.002)`
 Defines the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .

- **log_reference** (numeric or array_like) – Log reference.
- **linear_reference** (numeric or array_like) – Linear reference.
- **negative_gamma** (numeric or array_like) – Negative gamma.
- **density_per_code_value** (numeric or array_like) – Density per code value.

Returns Non-linear data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[[SonyImageworks12](#)]

Examples

```
>>> log_encoding_PivotedLog(0.18) # doctest: +ELLIPSIS
0.4349951...
```

colour.models.log_decoding_PivotedLog

`colour.models.log_decoding_PivotedLog(y, log_reference=445, linear_reference=0.18, negative_gamma=0.6, density_per_code_value=0.002)`

Defines the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y .
- **log_reference** (numeric or array_like) – Log reference.
- **linear_reference** (numeric or array_like) – Linear reference.
- **negative_gamma** (numeric or array_like) – Negative gamma.
- **density_per_code_value** (numeric or array_like) – Density per code value.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_PivotedLog(0.434995112414467) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_Protune

colour.models.log_encoding_Protune(x)

Defines the *Protune* log encoding curve / opto-electronic transfer function.

Parameters x (numeric or array_like) – Linear data *x*.

Returns Non-linear data *y*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[GDM16]

Examples

```
>>> log_encoding_Protune(0.18) # doctest: +ELLIPSIS
0.6456234...
```

colour.models.log_decoding_Protune

colour.models.log_decoding_Protune(*y*)

Defines the *Protune* log decoding curve / electro-optical transfer function.

Parameters *y* (numeric or array_like) – Non-linear data *y*.

Returns Linear data *x*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>y</i>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>x</i>	[0, 1]	[0, 1]

References

[GDM16]

Examples

```
>>> log_decoding_Protune(0.645623486803636) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_REDLog

colour.models.log_encoding_REDLog(*x*, *black_offset*=0.009955040995908344)

Defines the *REDLog* log encoding curve / opto-electronic transfer function.

Parameters

- *x* (numeric or array_like) – Linear data *x*.
- *black_offset* (numeric or array_like) – Black offset.

Returns Non-linear data *y*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>x</i>	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>y</i>	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLog(0.18) # doctest: +ELLIPSIS
0.6376218...
```

colour.models.log_decoding_REDLog

`colour.models.log_decoding_REDLog(y, black_offset=0.009955040995908344)`
 Defines the *REDLog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y .
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLog(0.637621845988175) # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_REDLogFilm

`colour.models.log_encoding_REDLogFilm(x, black_offset=0.0107977516232771)`
 Defines the *REDLogFilm* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x .

- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLogFilm(0.18) # doctest: +ELLIPSIS
0.4573196...
```

colour.models.log_decoding_REDLogFilm

colour.models.**log_decoding_REDLogFilm**(y , *black_offset*=0.0107977516232771)

Defines the *REDLogFilm* log decoding curve / electro-optical transfer function.

Parameters

- y (numeric or array_like) – Non-linear data y .
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLogFilm(0.457319613085418) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_SLog

`colour.models.log_encoding_SLog(x, bit_depth=10, out_legal=True, in_reflection=True)`
 Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or *IRE*/100 input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log* data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog(0.18) # doctest: +ELLIPSIS
0.3849708...
>>> log_encoding_SLog(0.18, out_legal=False) # doctest: +ELLIPSIS
0.3765127...
>>> log_encoding_SLog(0.18, in_reflection=False) # doctest: +ELLIPSIS
0.3708204...
```

colour.models.log_decoding_SLog

`colour.models.log_decoding_SLog(y, bit_depth=10, in_legal=True, out_reflection=True)`
Defines the *Sony S-Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log* data y .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Reflection or *IRE*/100 input light level x to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog(0.384970815928670) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog(0.376512722254600, in_legal=False)
... # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog(0.370820482371268, out_reflection=False)
... # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_SLog2

`colour.models.log_encoding_SLog2(x, bit_depth=10, out_legal=True, in_reflection=True)`
Defines the *Sony S-Log2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or *IRE*/100 input light level x to a camera.

- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **out_legal** (`bool`, optional) – Whether the non-linear *Sony S-Log2* data y is encoded in legal range.
- **in_reflection** (`bool`, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log2* data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog2(0.18) # doctest: +ELLIPSIS
0.3395325...
>>> log_encoding_SLog2(0.18, out_legal=False) # doctest: +ELLIPSIS
0.3234495...
>>> log_encoding_SLog2(0.18, in_reflection=False) # doctest: +ELLIPSIS
0.3262865...
```

colour.models.log_decoding_SLog2

`colour.models.log_decoding_SLog2(y, bit_depth=10, in_legal=True, out_reflection=True)`

Defines the *Sony S-Log2* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log2* data y .
- **bit_depth** (`int`, optional) – Bit depth used for conversion.
- **in_legal** (`bool`, optional) – Whether the non-linear *Sony S-Log2* data y is encoded in legal range.
- **out_reflection** (`bool`, optional) – Whether the light level x to a camera is reflection.

Returns Reflection or *IRE*/100 input light level x to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog2(0.339532524633774) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog2(0.323449512215013, in_legal=False)
... # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog2(0.326286538946799, out_reflection=False)
... # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_SLog3

`colour.models.log_encoding_SLog3(x, bit_depth=10, out_legal=True, in_reflection=True)`

Defines the *Sony S-Log3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or *IRE*/100 input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Returns Non-linear *Sony S-Log3* data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_encoding_SLog3(0.18) # doctest: +ELLIPSIS
0.4105571...
>>> log_encoding_SLog3(0.18, out_legal=False) # doctest: +ELLIPSIS
0.4063926...
>>> log_encoding_SLog3(0.18, in_reflection=False) # doctest: +ELLIPSIS
0.3995079...
```

colour.models.log_decoding_SLog3

colour.models.log_decoding_SLog3(*y*, *bit_depth*=10, *in_legal*=True, *out_reflection*=True)

Defines the Sony *S-Log3* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear Sony *S-Log3* data *y*.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear Sony *S-Log3* data *y* is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level *x* to a camera is reflection.

Returns Reflection or *IRE*/100 input light level *x* to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_decoding_SLog3(0.410557184750733) # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog3(0.406392694063927, in_legal=False)
... # doctest: +ELLIPSIS
0.1...
>>> log_decoding_SLog3(0.399507939606216, out_reflection=False)
... # doctest: +ELLIPSIS
0.1...
```

colour.models.log_encoding_VLog

`colour.models.log_encoding_VLog(L_in, bit_depth=10, out_legal=True, in_reflection=True, constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206})`

Defines the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **L_in** (numeric or array_like) – Linear reflection data L_{in} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_legal** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded in legal range.
- **in_reflection** (bool, optional) – Whether the light level L_{in} to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.

Returns Non-linear data V_{out} .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_{in}	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_{out}	[0, 1]	[0, 1]

References

[Pan14]

Examples

```
>>> log_encoding_VLog(0.18) # doctest: +ELLIPSIS
0.4233114...
```

colour.models.log_decoding_VLog

`colour.models.log_decoding_VLog(V_out, bit_depth=10, in_legal=True, out_reflection=True, constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206})`

Defines the *Panasonic V-Log* log decoding curve / electro-optical transfer function.

Parameters

- **V_out** (numeric or array_like) – Non-linear data V_{out} .
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_legal** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded in legal range.
- **out_reflection** (bool, optional) – Whether the light level :math'L_{in}' to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.

Returns Linear reflection data :math'L_{in}'.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V_{out}	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_{in}	[0, 1]	[0, 1]

References

[Pan14]

Examples

```
>>> log_decoding_VLog(0.423311448760136) # doctest: +ELLIPSIS
0.1799999...
```

colour.models.log_encoding_ViperLog

`colour.models.log_encoding_ViperLog(x)`

Defines the *Viper Log* log encoding curve / opto-electronic transfer function.

Parameters x (numeric or array_like) – Linear data x .

Returns Non-linear data y .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_ViperLog(0.18) # doctest: +ELLIPSIS
0.6360080...
```

colour.models.log_decoding_ViperLog

colour.models.**log_decoding_ViperLog**(y)

Defines the *Viper Log* log decoding curve / electro-optical transfer function.

Parameters y (numeric or array_like) – Non-linear data *y*.

Returns Linear data *x*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_ViperLog(0.636008067010413) # doctest: +ELLIPSIS
0.1799999...
```

ACES Spectral Conversion

colour

<code>sd_to_aces_relative_exposure_values(sd[, ...])</code>	Converts given spectral distribution to <i>ACES2065-1</i> colourspace relative exposure values.
---	---

colour.sd_to_aces_relative_exposure_values

```
colour.sd_to_aces_relative_exposure_values(sd, illuminant=SpectralDistribution(name='D65',
...), apply_chromatic_adaptation=False, chromatic_adaptation_transform='CAT02')
```

Converts given spectral distribution to *ACES2065-1* colourspace relative exposure values.

Parameters

- **sd** (*SpectralDistribution*) – Spectral distribution.
- **illuminant** (*SpectralDistribution*, optional) – *Illuminant* spectral distribution.
- **apply_chromatic_adaptation** (*bool*, optional) – Whether to apply chromatic adaptation using given transform.
- **chromatic_adaptation_transform** (*unicode*, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, *Chromatic adaptation* transform.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- The chromatic adaptation method implemented here is a bit unusual as it involves building a new colourspace based on *ACES2065-1* colourspace primaries but using the whitepoint of the illuminant that the spectral distribution was measured under.

References

[For18], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> from colour import COLOURCHECKERS_SDS
>>> sd = COLOURCHECKERS_SDS['ColorChecker N Ohta']['dark skin']
>>> sd_to_aces_relative_exposure_values(sd) # doctest: +ELLIPSIS
array([ 0.1171785...,  0.0866347...,  0.0589707...])
```

(continues on next page)

(continued from previous page)

```
>>> sd_to_aces_relative_exposure_values(sd,
...     apply_chromatic_adaptation=True) # doctest: +ELLIPSIS
array([ 0.1180766...,  0.0869023...,  0.0589104...])
```

Ancillary Objects

colour.models

ACES_RICD

Implements support for the *CIE RGB* colour matching functions.

colour.models.ACES_RICD

colour.models.ACES_RICD = RGB_ColourMatchingFunctions(name='ACES RICD', ...)

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignal or MultiSpectralDistribution or array_like or dict_like, optional) – Data to be stored in the multi-spectral distribution.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (unicode, optional) – Multi-spectral distribution name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distribution labels for figures, default to colour.colorimetry.RGB_ColourMatchingFunctions.labels attribute value.

Colour Encodings

Y'CbCr Colour Encoding

colour

<code>RGB_to_YCbCr(RGB[, K, in_bits, in_legal, ...])</code>	Converts an array of <i>R'G'B'</i> values to the corresponding <i>YCbCr</i> colour encoding values array.
<code>YCbCr_to_RGB(YCbCr[, K, in_bits, in_legal, ...])</code>	Converts an array of <i>YCbCr</i> colour encoding values to the corresponding <i>R'G'B'</i> values array.
<code>YCBCR_WEIGHTS</code>	Implements a case-insensitive mutable mapping / dict object.
<code>RGB_to_YcCbCrc(RGB[, out_bits, out_legal, ...])</code>	Converts an array of <i>RGB</i> linear values to the corresponding <i>Yc'CbC'rc'</i> colour encoding values array.
<code>YcCbCrc_to_RGB(YcCbCrc[, in_bits, ...])</code>	Converts an array of <i>Yc'CbC'rc'</i> colour encoding values to the corresponding <i>RGB</i> array of linear values.

colour.RGB_to_YCbCr

`colour.RGB_to_YCbCr(RGB, K=array([ 0.2126, 0.0722]), in_bits=10, in_legal=False, in_int=False, out_bits=8, out_legal=True, out_int=False, **kwargs)`

Converts an array of *R'G'B'* values to the corresponding *YCbCr* colour encoding values array.

Parameters

- **RGB** (*array_like*) – Input *R'G'B'* array of floats or integer values.
- **K** (*array_like*, optional) – Luma weighting coefficients of red and blue. See `colour.YCBCR_WEIGHTS` for presets. Default is *(0.2126, 0.0722)*, the weightings for ITU-R BT.709.
- **in_bits** (*int*, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Default is *10*.
- **in_legal** (*bool*, optional) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (*bool*, optional) – Whether to treat the input values as *in_bits* integer code values. Default is *False*.
- **out_bits** (*int*, optional) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Ignored if *out_legal* and *out_int* are both *False*. Default is *8*.
- **out_legal** (*bool*, optional) – Whether to return legal range values. Default is *True*.
- **out_int** (*bool*, optional) – Whether to return values as *out_bits* integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like*, optional) – Array overriding the computed range such as *in_range = (RGB_min, RGB_max)*. If *in_range* is undefined, *RGB_min* and *RGB_max* will be computed using `colour.CV_range()` definition.
- **out_range** (*array_like*, optional) – Array overriding the computed range such as *out_range = (Y_min, Y_max, C_min, C_max)*. If “*out_range*” is undefined, **Y_min*, *Y_max*, *C_min* and *C_max* will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.

Returns *YCbCr* colour encoding array of integer or float values.

Return type ndarray

Warning: For *Recommendation ITU-R BT.2020*, `colour.RGB_to_YCbCr()` definition is only applicable to the non-constant luminance implementation. `colour.RGB_to_YcCbCrCrc()` definition should be used for the constant luminance case as per [InternationalTUnion15a].

Notes

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

- * This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.
- The default arguments, `**{'in_bits': 10, 'in_legal': False, 'in_int': False, 'out_bits': 8, 'out_legal': True, 'out_int': False}` transform a float *R'G'B'* input array normalised to domain [0, 1] (*in_bits* is ignored) to a float *Y'CbCr* output array where *Y'* is normalised to range [16 / 255, 235 / 255] and *Cb* and *Cr* are normalised to range [16 / 255, 240./255]. The float values are calculated based on an [0, 255] integer range, but no 8-bit quantisation or clamping are performed.

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wik04e]

Examples

```
>>> RGB = np.array([1.0, 1.0, 1.0])
>>> RGB_to_YCbCr(RGB) # doctest: +ELLIPSIS
array([ 0.9215686...,  0.5019607...,  0.5019607...])
```

Matching float output of The Foundry Nuke's Colorspace node set to YCbCr:

```
>>> RGB_to_YCbCr(RGB,
...               out_range=(16 / 255, 235 / 255, 15.5 / 255, 239.5 / 255))
... # doctest: +ELLIPSIS
array([ 0.9215686...,  0.5...,  0.5...])
```

Matching float output of The Foundry Nuke's Colorspace node set to YPbPr:

```
>>> RGB_to_YCbCr(RGB, out_legal=False, out_int=False)
... # doctest: +ELLIPSIS
array([ 1.,  0.,  0.])
```

Creating integer code values as per standard 10-bit SDI:


```
>>> RGB_to_YCbCr(RGB, out_legal=True, out_bits=10, out_int=True)
array([940, 512, 512])
```

For JFIF JPEG conversion as per ITU-T T.871 [InternationalTUnion11c]:

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_range=(0, 255),
...               out_range=(0, 255, 0, 256), out_int=True)
array([ 36, 136, 175])
```

Note the use of 256 for the max *Cb* / *Cr* value, which is required so that the *Cb* and *Cr* output is centered about 128. Using 255 centres it about 127.5, meaning that there is no integer code value to represent achromatic colours. This does however create the possibility of output integer codes with value of 256, which cannot be stored in 8-bit integer representation. Recommendation ITU-T T.871 specifies these should be clamped to 255.

These JFIF JPEG ranges are also obtained as follows:

```
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_bits=8,
...               in_int=True, out_legal=False, out_int=True)
array([ 36, 136, 175])
```

colour.YCbCr_to_RGB

`colour.YCbCr_to_RGB(YCbCr, K=array([ 0.2126, 0.0722]), in_bits=8, in_legal=True, in_int=False, out_bits=10, out_legal=False, out_int=False, **kwargs)`

Converts an array of *YCbCr* colour encoding values to the corresponding *R'G'B'* values array.

Parameters

- **YCbCr** (array_like) – Input *YCbCr* colour encoding array of integer or float values.
- **K** (array_like, optional) – Luma weighting coefficients of red and blue. See `colour.YCBCR_WEIGHTS` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (int, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Default is *10*.
- **in_legal** (bool, optional) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (bool, optional) – Whether to treat the input values as *in_bits* integer code values. Default is *False*.
- **out_bits** (int, optional) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Ignored if *out_legal* and *out_int* are both *False*. Default is *8*.
- **out_legal** (bool, optional) – Whether to return legal range values. Default is *True*.
- **out_int** (bool, optional) – Whether to return values as *out_bits* integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like, optional*) – Array overriding the computed range such as `in_range = (Y_min, Y_max, C_min, C_max)`. If `in_range` is undefined, `Y_min`, `Y_max`, `C_min` and `C_max` will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.
- **out_range** (*array_like, optional*) – Array overriding the computed range such as `out_range = (RGB_min, RGB_max)`. If `out_range` is undefined, `RGB_min` and `RGB_max` will be computed using `colour.CV_range()` definition.

Returns *R'G'B'* array of integer or float values.

Return type ndarray

Notes

Domain *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

- * This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: For *Recommendation ITU-R BT.2020*, `colour.YCbCr_to_RGB()` definition is only applicable to the non-constant luminance implementation. `colour.YcCbCrCrc_to_RGB()` definition should be used for the constant luminance case as per [InternationalUnion15a].

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wik04e]

Examples

```
>>> YCbCr = np.array([502, 512, 512])
>>> YCbCr_to_RGB(YCbCr, in_bits=10, in_legal=True, in_int=True)
array([ 0.5,  0.5,  0.5])
```

colour.YCBCR_WEIGHTS

`colour.YCBCR_WEIGHTS = CaseInsensitiveMapping({'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-R BT.2020': ...})`

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters `data` (*dict*) – *dict* of data to store into the mapping at initialisation.

Other Parameters ****kwargs** (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

```
colour.__setitem__()
colour.__getitem__()
colour.__delitem__()
colour.__contains__()
colour.__iter__()
colour.__len__()
colour.__eq__()
colour.__ne__()
colour.__repr__()
colour.copy()
colour.lower_items()
```

Warning: The keys are expected to be unicode or string-like objects.

References

[Rei]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

colour.RGB_to_YcCbCrC

`colour.RGB_to_YcCbCrC(`*RGB*`,` *out_bits=10*`,` *out_legal=True*`,` *out_int=False*`,` *is_12_bits_system=False*`,` ****kwargs**`)`

Converts an array of *RGB* linear values to the corresponding *YcCbCrC* colour encoding values array.

Parameters

- **RGB** (*array_like*) – Input *RGB* array of linear float values.
- **out_bits** (*int, optional*) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Ignored if *out_legal* and *out_int* are both *False*. Default is *10*.
- **out_legal** (*bool, optional*) – Whether to return legal range values. Default is *True*.
- **out_int** (*bool, optional*) – Whether to return values as *out_bits* integer code values. Default is *False*.

- **is_12_bits_system** (`bool`, optional) – Recommendation ITU-R BT.2020 OETF (OECF) adopts different parameters for 10 and 12 bit systems. Default is *False*.

Other Parameters **out_range** (*array_like*, optional) – Array overriding the computed range such as `out_range = (Y_min, Y_max, C_min, C_max)`. If `out_range` is undefined, `Y_min`, `Y_max`, `C_min` and `C_max` will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.

Returns *Yc'CbC'Cr'* colour encoding array of integer or float values.

Return type `ndarray`

Notes

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YcCbCrc	[0, 1]	[0, 1]

- * This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[[InternationalTUnion15a](#)], [[Wik04e](#)]

Examples

```
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> RGB_to_YcCbCrc(RGB, out_legal=True, out_bits=10, out_int=True,
...                 is_12_bits_system=False)
array([422, 512, 512])
```

colour.YcCbCrc_to_RGB

`colour.YcCbCrc_to_RGB(YcCbCrc, in_bits=10, in_legal=True, in_int=False, is_12_bits_system=False, **kwargs)`

Converts an array of *Yc'CbC'Cr'* colour encoding values to the corresponding *RGB* array of linear values.

Parameters

- **YcCbCrc** (*array_like*) – Input *Yc'CbC'Cr'* colour encoding array of linear float values.

- **in_bits** (*int*, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is $235 / 255$. Default is *10*.
- **in_legal** (*bool*, optional) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (*bool*, optional) – Whether to treat the input values as *in_bits* integer code values. Default is *False*.
- **is_12_bits_system** (*bool*, optional) – *Recommendation ITU-R BT.2020* EOTF (EOCF) adopts different parameters for 10 and 12 bit systems. Default is *False*.

Other Parameters **in_range** (*array_like*, optional) – Array overriding the computed range such as *in_range* = (*Y_min*, *Y_max*, *C_min*, *C_max*). If *in_range* is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using `colour.models.rgb.ycbcr.YCbCr_ranges()` definition.

Returns *RGB* array of linear float values.

Return type *ndarray*

Notes

Domain *	Scale - Reference	Scale - 1
YcCbCrc	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

- * This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[[InternationalTUnion15a](#)], [[Wik04e](#)]

Examples

```
>>> YcCbCrc = np.array([1689, 2048, 2048])
>>> YcCbCrc_to_RGB(YcCbCrc, in_legal=True, in_bits=12, in_int=True,
...                 is_12_bits_system=True)
... # doctest: +ELLIPSIS
array([ 0.1800903...,  0.1800903...,  0.1800903...])
```

Ancillary Objects

`colour`

<code>full_to_legal(CV[, bit_depth, in_int, out_int])</code>	Converts given code value <i>CV</i> or float equivalent of a code value at a given bit depth from full range (full swing) to legal range (studio swing).
<code>legal_to_full(CV[, bit_depth, in_int, out_int])</code>	Converts given code value <i>CV</i> or float equivalent of a code value at a given bit depth from legal range (studio swing) to full range (full swing).
<code>CV_range([bit_depth, is_legal, is_int])</code>	Returns the code value <i>CV</i> range for given bit depth, range legality and representation.

colour.full_to_legal

`colour.full_to_legal(CV, bit_depth=10, in_int=False, out_int=False)`

Converts given code value *CV* or float equivalent of a code value at a given bit depth from full range (full swing) to legal range (studio swing).

Parameters

- **CV** (array_like) – Full range code value *CV* or float equivalent of a code value at a given bit depth.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Legal range code value *CV* or float equivalent of a code value at a given bit depth.

Return type ndarray

Examples

```
>>> full_to_legal(0.0) # doctest: +ELLIPSIS
0.0625610...
>>> full_to_legal(1.0) # doctest: +ELLIPSIS
0.9188660...
>>> full_to_legal(0.0, out_int=True)
64
>>> full_to_legal(1.0, out_int=True)
940
>>> full_to_legal(0, in_int=True) # doctest: +ELLIPSIS
0.0625610...
>>> full_to_legal(1023, in_int=True) # doctest: +ELLIPSIS
0.9188660...
>>> full_to_legal(0, in_int=True, out_int=True)
64
>>> full_to_legal(1023, in_int=True, out_int=True)
940
```

colour.legal_to_full

`colour.legal_to_full(CV, bit_depth=10, in_int=False, out_int=False)`

Converts given code value *CV* or float equivalent of a code value at a given bit depth from legal range (studio swing) to full range (full swing).

Parameters

- **CV** (array_like) – Legal range code value *CV* or float equivalent of a code value at a given bit depth.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Full range code value *CV* or float equivalent of a code value at a given bit depth.

Return type ndarray

Examples

```

>>> legal_to_full(64 / 1023)
0.0
>>> legal_to_full(940 / 1023)
1.0
>>> legal_to_full(64 / 1023, out_int=True)
0
>>> legal_to_full(940 / 1023, out_int=True)
1023
>>> legal_to_full(64, in_int=True)
0.0
>>> legal_to_full(940, in_int=True)
1.0
>>> legal_to_full(64, in_int=True, out_int=True)
0
>>> legal_to_full(940, in_int=True, out_int=True)
1023

```

colour.CV_range

`colour.CV_range(bit_depth=10, is_legal=False, is_int=False)`

Returns the code value *CV* range for given bit depth, range legality and representation.

Parameters

- **bit_depth** (int, optional) – Bit depth of the code value *CV* range.
- **is_legal** (bool, optional) – Whether the code value *CV* range is legal.
- **is_int** (bool, optional) – Whether the code value *CV* range represents integer code values.

Returns Code value *CV* range.

Return type ndarray

Examples

```
>>> CV_range(8, True, True)
array([ 16, 235])
>>> CV_range(8, True, False) # doctest: +ELLIPSIS
array([ 0.0627451...,  0.9215686...])
>>> CV_range(10, False, False)
array([ 0.,  1.]
```

YCoCg Colour Encoding

colour

<code>RGB_to_YCoCg(RGB)</code>	Converts an array of $R'G'B'$ values to the corresponding YCoCg colour encoding values array.
<code>YCoCg_to_RGB(YCoCg)</code>	Converts an array of YCoCg colour encoding values to the corresponding $R'G'B'$ values array.

colour.RGB_to_YCoCg

colour.**RGB_to_YCoCg**(RGB)

Converts an array of $R'G'B'$ values to the corresponding YCoCg colour encoding values array.

Parameters RGB (array_like) – Input $R'G'B'$ array.

Returns YCoCg colour encoding array.

Return type ndarray

References

[MS03]

Examples

```
>>> RGB_to_YCoCg(np.array([1.0, 1.0, 1.0]))
array([ 1.,  0.,  0.])
>>> RGB_to_YCoCg(np.array([0.75, 0.5, 0.5]))
array([ 0.5625,  0.125 , -0.0625])
```

colour.YCoCg_to_RGB

colour.**YCoCg_to_RGB**(YCoCg)

Converts an array of YCoCg colour encoding values to the corresponding $R'G'B'$ values array.

Parameters YCoCg (array_like) – YCoCg colour encoding array.

Returns Output $R'G'B'$ array.

Return type ndarray

References

[MS03]

Examples

```
>>> YCoCg_to_RGB(np.array([1.0, 0.0, 0.0]))
array([ 1.,  1.,  1.])
>>> YCoCg_to_RGB(np.array([0.5625, 0.125, -0.0625]))
array([ 0.75,  0.5 ,  0.5 ])
```

IC_TC_P Colour Encoding

colour

<code>RGB_to ICTCP(RGB[, L_p])</code>	Converts from <i>ITU-R BT.2020</i> colourspace to IC_TC_P colour encoding.
<code>ICTCP_to_RGB(ICTCP[, L_p])</code>	Converts from IC_TC_P colour encoding to <i>ITU-R BT.2020</i> colourspace.

colour.RGB_to ICTCP

colour.RGB_to ICTCP(RGB, L_p=10000)

Converts from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.

Parameters

- **RGB** (array_like) – *ITU-R BT.2020* colourspace array.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding.

Returns IC_TC_P colour encoding array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ICTCP	I : [0, 1]	I : [0, 1]
	CT : [-1, 1]	CT : [-1, 1]
	CP : [-1, 1]	CP : [-1, 1]

References

[Dol16], [LPY+16]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to ICTCP(RGB) # doctest: +ELLIPSIS
array([ 0.0735136..., 0.0047525..., 0.0935159...])
```

colour.ICTCP_to_RGB

`colour.ICTCP_to_RGB(ICTCP, L_p=10000)`

Converts from $IC_T C_P$ colour encoding to *ITU-R BT.2020* colourspace.

Parameters

- **ICTCP** (array_like) – $IC_T C_P$ colour encoding array.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding.

Returns *ITU-R BT.2020* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
ICTCP	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Dol16], [LPY+16]

Examples

```
>>> ICTCP = np.array([0.07351364, 0.00475253, 0.09351596])
>>> ICTCP_to_RGB(ICTCP) # doctest: +ELLIPSIS
array([ 0.4562052..., 0.0308107..., 0.0409195...])
```

RGB Representations

Prismatic Colourspace

colour

<code>RGB_to_Prismatic(RGB)</code>	Converts from <i>RGB</i> colourspace to <i>Prismatic</i> $L\rho\gamma\beta$ colourspace array.
<code>Prismatic_to_RGB(Lrgb)</code>	Converts from <i>Prismatic</i> $L\rho\gamma\beta$ colourspace array to <i>RGB</i> colourspace.

colour.RGB_to_Prismatic

`colour.RGB_to_Prismatic(RGB)`

Converts from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.

Parameters `RGB` (array_like) – *RGB* colourspace array.

Returns *Prismatic* $L\rho\gamma\beta$ colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> RGB = np.array([0.25, 0.50, 0.75])
>>> RGB_to_Prismatic(RGB) # doctest: +ELLIPSIS
array([ 0.75...,  0.166666...,  0.333333...,  0.5...  ])
```

Adjusting saturation of given *RGB* colourspace array: `>>> saturation = 0.5 >>> Lrgb = RGB_to_Prismatic(RGB) >>> Lrgb[..., 1:] = 1 / 3 + saturation * (Lrgb[..., 1:] - 1 / 3) >>> Prismatic_to_RGB(Lrgb) # doctest: +ELLIPSIS array([ 0.45..., 0.6..., 0.75... ])`

colour.Prismatic_to_RGB

`colour.Prismatic_to_RGB(Lrgb)`

Converts from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* colourspace.

Parameters `Lrgb` (array_like) – *Prismatic* $L\rho\gamma\beta$ colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> Lrgb = np.array([0.75000000, 0.16666667, 0.33333333, 0.50000000])
>>> Prismatic_to_RGB(Lrgb) # doctest: +ELLIPSIS
array([ 0.25..., 0.4999999..., 0.75... ])
```

HSV Colourspace

colour

<code>RGB_to_HSV(</code> RGB <code>)</code>	Converts from <i>RGB</i> colourspace to <i>HSV</i> colourspace.
<code>HSV_to_RGB(</code> HSV <code>)</code>	Converts from <i>HSV</i> colourspace to <i>RGB</i> colourspace.

colour.RGB_to_HSV

colour.**RGB_to_HSV**(*RGB*)

Converts from *RGB* colourspace to *HSV* colourspace.

Parameters *RGB* (array_like) – *RGB* colourspace array.

Returns *HSV* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

References

[Eash], [Smi78], [Wik03a]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSV(RGB) # doctest: +ELLIPSIS
array([ 0.9960394..., 0.9324630..., 0.4562051...])
```

colour.HSV_to_RGB

`colour.HSV_to_RGB(HSV)`

Converts from *HSV* colourspace to *RGB* colourspace.

Parameters *HSV* (array_like) – *HSV* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Ease], [Smi78], [Wik03a]

Examples

```
>>> HSV = np.array([0.99603944, 0.93246304, 0.45620519])
>>> HSV_to_RGB(HSV) # doctest: +ELLIPSIS
array([ 0.4562051..., 0.0308107..., 0.0409195...])
```

HSL Colourspace

`colour`

<code>RGB_to_HSL(RGB)</code>	Converts from <i>RGB</i> colourspace to <i>HSL</i> colourspace.
<code>HSL_to_RGB(HSL)</code>	Converts from <i>HSL</i> colourspace to <i>RGB</i> colourspace.

colour.RGB_to_HSL

colour.**RGB_to_HSL**(*RGB*)

Converts from *RGB* colourspace to *HSL* colourspace.

Parameters *RGB* (array_like) – *RGB* colourspace array.

Returns *HSL* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

References

[Easg], [Smi78], [Wik03a]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSL(RGB) # doctest: +ELLIPSIS
array([ 0.9960394..., 0.8734714..., 0.2435079...])
```

colour.HSL_to_RGB

colour.**HSL_to_RGB**(*HSL*)

Converts from *HSL* colourspace to *RGB* colourspace.

Parameters *HSL* (array_like) – *HSL* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Easd], [Smi78], [Wik03a]

Examples

```
>>> HSL = np.array([0.99603944, 0.87347144, 0.24350795])
>>> HSL_to_RGB(HSL) # doctest: +ELLIPSIS
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

CMY Colourspace

colour

<code>RGB_to_CMY(RGB)</code>	Converts from <i>RGB</i> colourspace to <i>CMY</i> colourspace.
<code>CMY_to_RGB(CMY)</code>	Converts from <i>CMY</i> colourspace to <i>CMY</i> colourspace.
<code>CMY_to_CMYK(CMY)</code>	Converts from <i>CMY</i> colourspace to <i>CMYK</i> colourspace.
<code>CMYK_to_CMY(CMYK)</code>	Converts from <i>CMYK</i> colourspace to <i>CMY</i> colourspace.

colour.RGB_to_CMY

colour.**RGB_to_CMY**(*RGB*)

Converts from *RGB* colourspace to *CMY* colourspace.

Parameters *RGB* (array_like) – *RGB* colourspace array.

Returns *CMY* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[Easf]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_CMY(RGB) # doctest: +ELLIPSIS
array([ 0.5437948..., 0.9691892..., 0.9590804...])
```

colour.CMY_to_RGB

colour.CMY_to_RGB(*CMY*)

Converts from *CMY* colourspace to *CMY* colourspace.

Parameters *CMY* (array_like) – *CMY* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Easb]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_RGB(CMY) # doctest: +ELLIPSIS
array([ 0.4562051..., 0.0308107..., 0.0409195...])
```

colour.CMY_to_CMYK

colour.CMY_to_CMYK(*CMY*)

Converts from *CMY* colourspace to *CMYK* colourspace.

Parameters *CMY* (array_like) – *CMY* colourspace array.

Returns *CMYK* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

References

[Easa]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_CMYK(CMY) # doctest: +ELLIPSIS
array([ 0.          , 0.9324630..., 0.9103045..., 0.5437948...])
```

colour.CMYK_to_CMY

colour.CMYK_to_CMY(CMYK)

Converts from *CMYK* colour space to *CMY* colour space.

Parameters CMYK (array_like) – *CMYK* colour space array.

Returns *CMY* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[Easc]

Examples

```
>>> CMYK = np.array([0.50000000, 0.00000000, 0.74400000, 0.01960784])
>>> CMYK_to_CMY(CMYK) # doctest: +ELLIPSIS
array([ 0.5098039..., 0.0196078..., 0.7490196...])
```

Pointer's Gamut

colour

POINTER_GAMUT_BOUNDARIES
POINTER_GAMUT_DATA
POINTER_GAMUT_ILLUMINANT

colour.POINTER_GAMUT_BOUNDARIES

colour.POINTER_GAMUT_BOUNDARIES = array([[0.659, 0.316], [0.634, 0.351], [0.594, 0.391], [0.557, 0.427],

colour.POINTER_GAMUT_DATA

colour.POINTER_GAMUT_DATA = array([[15, 10, 0], [15, 15, 10], [15, 14, 20], ..., [90, 9, 330], [90, 4, 3

colour.POINTER_GAMUT_ILLUMINANT

colour.POINTER_GAMUT_ILLUMINANT = array([0.31005673, 0.3161457])

Colour Notation Systems

- *Munsell Renotation System*
- *Munsell Value*
 - *Priest, Gibson and MacNicholas (1920)*
 - *Munsell, Sloan and Godlove (1933)*
 - *Moon and Spencer (1943)*
 - *Saunderson and Milner (1944)*
 - *Ladd and Pinney (1955)*
 - *McCamy (1987)*
 - *ASTM D1535-08e1*
- *Hexadecimal Triplet Notation*

Munsell Renotation System

colour

<code>munsell_colour_to_xyY(munsell_colour)</code>	Converts given <i>Munsell</i> colour to <i>CIE xyY</i> colourspace.
--	---

Continued on next page

Table 227 – continued from previous page

<code>xyY_to_munsell_colour(xyY[, hue_decimals, ...])</code>	Converts from <i>CIE xyY</i> colourspace to <i>Munsell</i> colour.
--	--

colour.munsell_colour_to_xyY

`colour.munsell_colour_to_xyY(munsell_colour)`

Converts given *Munsell* colour to *CIE xyY* colourspace.

Parameters `munsell_colour` (unicode or array_like) – *Munsell* colour.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Cen], [Cen12]

Examples

```
>>> munsell_colour_to_xyY('4.2YR 8.1/5.3') # doctest: +ELLIPSIS
array([ 0.3873694...,  0.3575165...,  0.59362   ])
>>> munsell_colour_to_xyY('N8.9') # doctest: +ELLIPSIS
array([ 0.31006   ,  0.31616   ,  0.7461345...])
```

colour.xyY_to_munsell_colour

`colour.xyY_to_munsell_colour(xyY, hue_decimals=1, value_decimals=1, chroma_decimals=1)`

Converts from *CIE xyY* colourspace to *Munsell* colour.

Parameters

- `xyY` (array_like, (3,)) – *CIE xyY* colourspace array.
- `hue_decimals` (int) – Hue formatting decimals.
- `value_decimals` (int) – Value formatting decimals.
- `chroma_decimals` (int) – Chroma formatting decimals.

Returns *Munsell* colour.

Return type unicode

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[G_{en}], [G_{en}12]

Examples

```
>>> xyY = np.array([0.38736945, 0.35751656, 0.59362000])
>>> # Doctests skip for Python 2.x compatibility.
>>> xyY_to_munsell_colour(xyY) # doctest: +SKIP
'4.2YR 8.1/5.3'
```

Dataset

colour

MUNSELL_COLOURS

Aggregated *Munsell* colours.

colour.MUNSELL_COLOURS

colour.MUNSELL_COLOURS = CaseInsensitiveMapping({'Munsell Colours All': ..., 'Munsell Colours 1929': ..., 'Munsell Colours Real': ...})
Aggregated *Munsell* colours.

MUNSELL_COLOURS : CaseInsensitiveMapping

Aliases:

- 'all': 'Munsell Colours All'
- '1929': 'Munsell Colours 1929'
- 'real': 'Munsell Colours Real'

Munsell Value

colour

munsell_value(Y[, method])

Returns the *Munsell* value *V* of given *luminance* *Y* using given method.

MUNSELL_VALUE_METHODS

Supported *Munsell* value computation methods.

colour.munsell_value

colour.munsell_value(Y, method='ASTM D1535-08')

Returns the *Munsell* value *V* of given *luminance* *Y* using given method.

Parameters

- **Y** (numeric or array_like) – *luminance Y*.
- **method** (unicode, optional) – {'ASTM D1535-08', 'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987'}, Computation method.

Returns *Munsell* value *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[ASTMInternational89], [Wik07d]

Examples

```
>>> munsell_value(12.23634268) # doctest: +ELLIPSIS
4.0824437...
>>> munsell_value(12.23634268, method='Priest 1920') # doctest: +ELLIPSIS
3.4980484...
>>> munsell_value(12.23634268, method='Munsell 1933') # doctest: +ELLIPSIS
4.1627702...
>>> munsell_value(12.23634268, method='Moon 1943') # doctest: +ELLIPSIS
4.0688120...
>>> munsell_value(12.23634268, method='Saunderson 1944')
... # doctest: +ELLIPSIS
4.0444736...
>>> munsell_value(12.23634268, method='Ladd 1955') # doctest: +ELLIPSIS
4.0511633...
>>> munsell_value(12.23634268, method='McCamy 1987') # doctest: +ELLIPSIS
array(4.0814348...)
```

colour.MUNSELL_VALUE_METHODS

`colour.MUNSELL_VALUE_METHODS` = `CaseInsensitiveMapping`({'Priest 1920': ..., 'Munsell 1933': ..., 'Moon 1943': ...})
Supported *Munsell* value computation methods.

References

[ASTMInternational89], [Wik07d]

MUNSELL_VALUE_METHODS [`CaseInsensitiveMapping`] {'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987', 'ASTM D1535-08'}

Aliases:

- ‘astm2008’: ‘ASTM D1535-08’

Priest, Gibson and MacNicholas (1920)

`colour.notation`

<code>munsell_value_Priest1920(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using <i>Priest et al.(1920)</i> method..
--	--

`colour.notation.munsell_value_Priest1920`

`colour.notation.munsell_value_Priest1920(Y)`

Returns the *Munsell* value V of given *luminance* Y using *Priest et al. (1920)* method.

Parameters Y (numeric or array_like) – *luminance* Y .

Returns *Munsell* value V .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	$[0, 100]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
V	$[0, 10]$	$[0, 1]$

References

[Wik07d]

Examples

```
>>> munsell_value_Priest1920(12.23634268) # doctest: +ELLIPSIS
3.4980484...
```

Munsell, Sloan and Godlove (1933)

`colour.notation`

<code>munsell_value_Munsell1933(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using <i>Munsell et al.(1933)</i> method..
---	---

colour.notation.munsell_value_Munsell1933

colour.notation.munsell_value_Munsell1933(*Y*)

Returns the *Munsell* value *V* of given *luminance Y* using *Munsell et al. (1933)* method.

Parameters *Y* (numeric or array_like) – *luminance Y*.

Returns *Munsell* value *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
<i>Y</i>	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
<i>V</i>	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Munsell1933(12.23634268) # doctest: +ELLIPSIS
4.1627702...
```

Moon and Spencer (1943)

colour.notation

munsell_value_Moon1943(*Y*)

Returns the *Munsell* value *V* of given *luminance Y* using *Moon and Spencer (1943)* method.

colour.notation.munsell_value_Moon1943

colour.notation.munsell_value_Moon1943(*Y*)

Returns the *Munsell* value *V* of given *luminance Y* using *Moon and Spencer (1943)* method.

Parameters *Y* (numeric or array_like) – *luminance Y*.

Returns *Munsell* value *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Moon1943(12.23634268) # doctest: +ELLIPSIS
4.0688120...
```

Saunderson and Milner (1944)

colour.notation

<code>munsell_value_Saunderson1944(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using <i>Saunderson and Milner (1944)</i> method.
--	--

colour.notation.munsell_value_Saunderson1944

colour.notation.**munsell_value_Saunderson1944**(Y)
Returns the *Munsell* value V of given *luminance* Y using *Saunderson and Milner (1944)* method.

Parameters Y (numeric) – *luminance* Y .

Returns *Munsell* value V .

Return type numeric

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Saunderson1944(12.23634268) # doctest: +ELLIPSIS
4.0444736...
```

Ladd and Pinney (1955)

colour.notation

<code>munsell_value_Ladd1955(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using <i>Ladd and Pinney (1955)</i> method.
--	--

colour.notation.munsell_value_Ladd1955

colour.notation.**munsell_value_Ladd1955**(Y)

Returns the *Munsell* value V of given *luminance* Y using *Ladd and Pinney (1955)* method.

Parameters Y (numeric or array_like) – *luminance* Y .

Returns *Munsell* value V .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	$[0, 100]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
V	$[0, 10]$	$[0, 1]$

References

[Wik07d]

Examples

```
>>> munsell_value_Ladd1955(12.23634268) # doctest: +ELLIPSIS
4.0511633...
```

McCamy (1987)

`colour.notation`

<code>munsell_value_McCamy1987(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using <i>McCamy (1987)</i> method.
--	---

`colour.notation.munsell_value_McCamy1987`

`colour.notation.munsell_value_McCamy1987(Y)`

Returns the *Munsell* value V of given *luminance* Y using *McCamy (1987)* method.

Parameters Y (numeric or array_like) – *luminance* Y .

Returns *Munsell* value V .

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	$[0, 100]$	$[0, 1]$

Range	Scale - Reference	Scale - 1
V	$[0, 10]$	$[0, 1]$

References

[ASTMInternational89]

Examples

```
>>> munsell_value_McCamy1987(12.23634268) # doctest: +ELLIPSIS
array(4.0814348...)
```

ASTM D1535-08e1

`colour.notation`

<code>munsell_value_ASTMD153508(Y)</code>	Returns the <i>Munsell</i> value V of given <i>luminance</i> Y using a reverse lookup table from <i>ASTM D1535-08e1</i> method.
---	---

colour.notation.munsell_value_ASTMD153508

colour.notation.munsell_value_ASTMD153508(*Y*)

Returns the *Munsell* value *V* of given *luminance Y* using a reverse lookup table from *ASTM D1535-08e1* method.

Parameters *Y* (numeric or array_like) – *luminance Y*

Returns *Munsell* value *V*.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

- The *Munsell* value* computation with *ASTM D1535-08e1* method is only defined for domain [0, 100].

References

[ASTMInternational89]

Examples

```
>>> munsell_value_ASTMD153508(12.23634268) # doctest: +ELLIPSIS
4.0824437...
```

Hexadecimal Triplet Notation

colour.notation

RGB_to_HEX(<i>RGB</i>)	Converts from <i>RGB</i> colourspace to hexadecimal triplet representation.
HEX_to_RGB(<i>HEX</i>)	Converts from hexadecimal triplet representation to <i>RGB</i> colourspace.

colour.notation.RGB_to_HEX

colour.notation.RGB_to_HEX(*RGB*)

Converts from *RGB* colourspace to hexadecimal triplet representation.

Parameters *RGB* (array_like) – *RGB* colourspace array.

Returns Hexadecimal triplet representation.

Return type unicode

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.6666667, 0.8666667, 1.0000000])
>>> # Doctests skip for Python 2.x compatibility.
>>> RGB_to_HEX(RGB) # doctest: +SKIP
'#aaddff'
```

colour.notation.HEX_to_RGB

colour.notation.HEX_to_RGB(*HEX*)

Converts from hexadecimal triplet representation to *RGB* colourspace.

Parameters *HEX* (unicode or array_like) – Hexadecimal triplet representation.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Notes

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> HEX = '#aaddff'
>>> HEX_to_RGB(HEX) # doctest: +ELLIPSIS
array([ 0.6666666..., 0.8666666..., 1.          ])
```

Optical Phenomena

- *Rayleigh Scattering*

Rayleigh Scattering

colour

<code>rayleigh_scattering(wavelength[, ...])</code>	Returns the <i>Rayleigh</i> optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
<code>sd_rayleigh_scattering([shape, ...])</code>	Returns the <i>Rayleigh</i> spectral distribution for given spectral shape.
<code>scattering_cross_section(wavelength[, ...])</code>	Returns the scattering cross section per molecule σ of dry air as function of wavelength λ in centimeters (cm) using given CO_2 concentration in parts per million (ppm) and temperature $T[K]$ in kelvin degrees following <i>Van de Hulst (1957)</i> method.

colour.rayleigh_scattering

```
colour.rayleigh_scattering(wavelength, CO2_concentration=300, temperature=288.15,
                           pressure=101325, latitude=0, altitude=0, avo-
                           gadro_constant=6.02214179e+23, n_s=<function
                           air_refraction_index_Bodhaine1999>, F_air=<function
                           F_air_Bodhaine1999>)
```

Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric or array_like) – Surface pressure P of the measurement site.
- **latitude** (numeric or array_like, optional) – Latitude of the site in degrees.
- **altitude** (numeric or array_like, optional) – Altitude of the site in meters.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro's* number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type numeric or ndarray

Warning: Unlike most objects of `colour.phenomena.rayleigh` module, `colour.phenomena.rayleigh_optical_depth()` expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8) # doctest: +ELLIPSIS
0.1004070...
```

colour.sd_rayleigh_scattering

```
colour.sd_rayleigh_scattering(shape=SpectralShape(360, 780, 1), CO2_concentration=300,
                             temperature=288.15, pressure=101325, latitude=0, alti-
                             tude=0, avogadro_constant=6.02214179e+23, n_s=<function
                             air_refraction_index_Bodhaine1999>, F_air=<function
                             F_air_Bodhaine1999>)
```

Returns the *Rayleigh* spectral distribution for given spectral shape.

Parameters

- **shape** ([SpectralShape](#), optional) – Spectral shape used to create the *Rayleigh* scattering spectral distribution.
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric or array_like) – Surface pressure P of the measurement site.
- **latitude** (numeric or array_like, optional) – Latitude of the site in degrees.
- **altitude** (numeric or array_like, optional) – Altitude of the site in meters.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro's* number (molecules mol^{-1}).
- **n_s** ([object](#)) – Air refraction index n_s computation method.
- **F_air** ([object](#)) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth spectral distribution.

Return type [SpectralDistribution](#)

References

[BWDS99], [Wik01d]

Examples

```

>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_rayleigh_scattering() # doctest: +ELLIPSIS
SpectralDistribution([[ 360.      ,  0.5991013...],
 [ 361.      ,  0.5921706...],
 [ 362.      ,  0.5853410...],
 [ 363.      ,  0.5786105...],
 [ 364.      ,  0.5719774...],
 [ 365.      ,  0.5654401...],
 [ 366.      ,  0.5589968...],
 [ 367.      ,  0.5526460...],
 [ 368.      ,  0.5463860...],
 [ 369.      ,  0.5402153...],
 [ 370.      ,  0.5341322...],
 [ 371.      ,  0.5281354...],
 [ 372.      ,  0.5222234...],
 [ 373.      ,  0.5163946...],
 [ 374.      ,  0.5106476...],
 [ 375.      ,  0.5049812...],
 [ 376.      ,  0.4993939...],
 [ 377.      ,  0.4938844...],
 [ 378.      ,  0.4884513...],
 [ 379.      ,  0.4830934...],
 [ 380.      ,  0.4778095...],
 [ 381.      ,  0.4725983...],
 [ 382.      ,  0.4674585...],
 [ 383.      ,  0.4623891...],
 [ 384.      ,  0.4573889...],
 [ 385.      ,  0.4524566...],
 [ 386.      ,  0.4475912...],
 [ 387.      ,  0.4427917...],
 [ 388.      ,  0.4380568...],
 [ 389.      ,  0.4333856...],
 [ 390.      ,  0.4287771...],
 [ 391.      ,  0.4242302...],
 [ 392.      ,  0.4197439...],
 [ 393.      ,  0.4153172...],
 [ 394.      ,  0.4109493...],
 [ 395.      ,  0.4066391...],
 [ 396.      ,  0.4023857...],
 [ 397.      ,  0.3981882...],
 [ 398.      ,  0.3940458...],
 [ 399.      ,  0.3899576...],
 [ 400.      ,  0.3859227...],
 [ 401.      ,  0.3819402...],
 [ 402.      ,  0.3780094...],
 [ 403.      ,  0.3741295...],
 [ 404.      ,  0.3702996...],
 [ 405.      ,  0.366519 ...],
 [ 406.      ,  0.3627868...],
 [ 407.      ,  0.3591025...],
 [ 408.      ,  0.3554651...],
 [ 409.      ,  0.3518740...],
 [ 410.      ,  0.3483286...],
 [ 411.      ,  0.344828 ...],
 [ 412.      ,  0.3413716...],
 [ 413.      ,  0.3379587...],

```

(continues on next page)

(continued from previous page)

[414.	,	0.3345887...],
[415.	,	0.3312609...],
[416.	,	0.3279747...],
[417.	,	0.3247294...],
[418.	,	0.3215245...],
[419.	,	0.3183593...],
[420.	,	0.3152332...],
[421.	,	0.3121457...],
[422.	,	0.3090962...],
[423.	,	0.3060841...],
[424.	,	0.3031088...],
[425.	,	0.3001699...],
[426.	,	0.2972668...],
[427.	,	0.2943989...],
[428.	,	0.2915657...],
[429.	,	0.2887668...],
[430.	,	0.2860017...],
[431.	,	0.2832697...],
[432.	,	0.2805706...],
[433.	,	0.2779037...],
[434.	,	0.2752687...],
[435.	,	0.2726650...],
[436.	,	0.2700922...],
[437.	,	0.2675500...],
[438.	,	0.2650377...],
[439.	,	0.2625551...],
[440.	,	0.2601016...],
[441.	,	0.2576770...],
[442.	,	0.2552807...],
[443.	,	0.2529124...],
[444.	,	0.2505716...],
[445.	,	0.2482581...],
[446.	,	0.2459713...],
[447.	,	0.2437110...],
[448.	,	0.2414768...],
[449.	,	0.2392683...],
[450.	,	0.2370851...],
[451.	,	0.2349269...],
[452.	,	0.2327933...],
[453.	,	0.2306841...],
[454.	,	0.2285989...],
[455.	,	0.2265373...],
[456.	,	0.2244990...],
[457.	,	0.2224838...],
[458.	,	0.2204912...],
[459.	,	0.2185211...],
[460.	,	0.2165730...],
[461.	,	0.2146467...],
[462.	,	0.2127419...],
[463.	,	0.2108583...],
[464.	,	0.2089957...],
[465.	,	0.2071536...],
[466.	,	0.2053320...],
[467.	,	0.2035304...],
[468.	,	0.2017487...],
[469.	,	0.1999865...],

(continues on next page)

(continued from previous page)

[470.	,	0.1982436...],
[471.	,	0.1965198...],
[472.	,	0.1948148...],
[473.	,	0.1931284...],
[474.	,	0.1914602...],
[475.	,	0.1898101...],
[476.	,	0.1881779...],
[477.	,	0.1865633...],
[478.	,	0.1849660...],
[479.	,	0.1833859...],
[480.	,	0.1818227...],
[481.	,	0.1802762...],
[482.	,	0.1787463...],
[483.	,	0.1772326...],
[484.	,	0.1757349...],
[485.	,	0.1742532...],
[486.	,	0.1727871...],
[487.	,	0.1713365...],
[488.	,	0.1699011...],
[489.	,	0.1684809...],
[490.	,	0.1670755...],
[491.	,	0.1656848...],
[492.	,	0.1643086...],
[493.	,	0.1629468...],
[494.	,	0.1615991...],
[495.	,	0.1602654...],
[496.	,	0.1589455...],
[497.	,	0.1576392...],
[498.	,	0.1563464...],
[499.	,	0.1550668...],
[500.	,	0.1538004...],
[501.	,	0.1525470...],
[502.	,	0.1513063...],
[503.	,	0.1500783...],
[504.	,	0.1488628...],
[505.	,	0.1476597...],
[506.	,	0.1464687...],
[507.	,	0.1452898...],
[508.	,	0.1441228...],
[509.	,	0.1429675...],
[510.	,	0.1418238...],
[511.	,	0.1406916...],
[512.	,	0.1395707...],
[513.	,	0.1384610...],
[514.	,	0.1373624...],
[515.	,	0.1362747...],
[516.	,	0.1351978...],
[517.	,	0.1341316...],
[518.	,	0.1330759...],
[519.	,	0.1320306...],
[520.	,	0.1309956...],
[521.	,	0.1299707...],
[522.	,	0.1289559...],
[523.	,	0.1279511...],
[524.	,	0.1269560...],
[525.	,	0.1259707...],

(continues on next page)

(continued from previous page)

[526.	,	0.1249949...],
[527.	,	0.1240286...],
[528.	,	0.1230717...],
[529.	,	0.1221240...],
[530.	,	0.1211855...],
[531.	,	0.1202560...],
[532.	,	0.1193354...],
[533.	,	0.1184237...],
[534.	,	0.1175207...],
[535.	,	0.1166263...],
[536.	,	0.1157404...],
[537.	,	0.1148630...],
[538.	,	0.1139939...],
[539.	,	0.1131331...],
[540.	,	0.1122804...],
[541.	,	0.1114357...],
[542.	,	0.1105990...],
[543.	,	0.1097702...],
[544.	,	0.1089492...],
[545.	,	0.1081358...],
[546.	,	0.1073301...],
[547.	,	0.1065319...],
[548.	,	0.1057411...],
[549.	,	0.1049577...],
[550.	,	0.1041815...],
[551.	,	0.1034125...],
[552.	,	0.1026507...],
[553.	,	0.1018958...],
[554.	,	0.1011480...],
[555.	,	0.1004070...],
[556.	,	0.0996728...],
[557.	,	0.0989453...],
[558.	,	0.0982245...],
[559.	,	0.0975102...],
[560.	,	0.0968025...],
[561.	,	0.0961012...],
[562.	,	0.0954062...],
[563.	,	0.0947176...],
[564.	,	0.0940352...],
[565.	,	0.0933589...],
[566.	,	0.0926887...],
[567.	,	0.0920246...],
[568.	,	0.0913664...],
[569.	,	0.0907141...],
[570.	,	0.0900677...],
[571.	,	0.0894270...],
[572.	,	0.0887920...],
[573.	,	0.0881627...],
[574.	,	0.0875389...],
[575.	,	0.0869207...],
[576.	,	0.0863079...],
[577.	,	0.0857006...],
[578.	,	0.0850986...],
[579.	,	0.0845019...],
[580.	,	0.0839104...],
[581.	,	0.0833242...],

(continues on next page)

(continued from previous page)

[582.	,	0.0827430...],
[583.	,	0.082167 ...],
[584.	,	0.0815959...],
[585.	,	0.0810298...],
[586.	,	0.0804687...],
[587.	,	0.0799124...],
[588.	,	0.0793609...],
[589.	,	0.0788142...],
[590.	,	0.0782722...],
[591.	,	0.0777349...],
[592.	,	0.0772022...],
[593.	,	0.0766740...],
[594.	,	0.0761504...],
[595.	,	0.0756313...],
[596.	,	0.0751166...],
[597.	,	0.0746063...],
[598.	,	0.0741003...],
[599.	,	0.0735986...],
[600.	,	0.0731012...],
[601.	,	0.072608 ...],
[602.	,	0.0721189...],
[603.	,	0.0716340...],
[604.	,	0.0711531...],
[605.	,	0.0706763...],
[606.	,	0.0702035...],
[607.	,	0.0697347...],
[608.	,	0.0692697...],
[609.	,	0.0688087...],
[610.	,	0.0683515...],
[611.	,	0.0678981...],
[612.	,	0.0674485...],
[613.	,	0.0670026...],
[614.	,	0.0665603...],
[615.	,	0.0661218...],
[616.	,	0.0656868...],
[617.	,	0.0652555...],
[618.	,	0.0648277...],
[619.	,	0.0644033...],
[620.	,	0.0639825...],
[621.	,	0.0635651...],
[622.	,	0.0631512...],
[623.	,	0.0627406...],
[624.	,	0.0623333...],
[625.	,	0.0619293...],
[626.	,	0.0615287...],
[627.	,	0.0611312...],
[628.	,	0.0607370...],
[629.	,	0.0603460...],
[630.	,	0.0599581...],
[631.	,	0.0595733...],
[632.	,	0.0591917...],
[633.	,	0.0588131...],
[634.	,	0.0584375...],
[635.	,	0.0580649...],
[636.	,	0.0576953...],
[637.	,	0.0573286...],

(continues on next page)

(continued from previous page)

[638.	,	0.0569649...],
[639.	,	0.0566040...],
[640.	,	0.0562460...],
[641.	,	0.0558909...],
[642.	,	0.0555385...],
[643.	,	0.0551890...],
[644.	,	0.0548421...],
[645.	,	0.0544981...],
[646.	,	0.0541567...],
[647.	,	0.053818 ...],
[648.	,	0.0534819...],
[649.	,	0.0531485...],
[650.	,	0.0528176...],
[651.	,	0.0524894...],
[652.	,	0.0521637...],
[653.	,	0.0518405...],
[654.	,	0.0515198...],
[655.	,	0.0512017...],
[656.	,	0.0508859...],
[657.	,	0.0505726...],
[658.	,	0.0502618...],
[659.	,	0.0499533...],
[660.	,	0.0496472...],
[661.	,	0.0493434...],
[662.	,	0.0490420...],
[663.	,	0.0487428...],
[664.	,	0.0484460...],
[665.	,	0.0481514...],
[666.	,	0.0478591...],
[667.	,	0.0475689...],
[668.	,	0.0472810...],
[669.	,	0.0469953...],
[670.	,	0.0467117...],
[671.	,	0.0464302...],
[672.	,	0.0461509...],
[673.	,	0.0458737...],
[674.	,	0.0455986...],
[675.	,	0.0453255...],
[676.	,	0.0450545...],
[677.	,	0.0447855...],
[678.	,	0.0445185...],
[679.	,	0.0442535...],
[680.	,	0.0439905...],
[681.	,	0.0437294...],
[682.	,	0.0434703...],
[683.	,	0.0432131...],
[684.	,	0.0429578...],
[685.	,	0.0427044...],
[686.	,	0.0424529...],
[687.	,	0.0422032...],
[688.	,	0.0419553...],
[689.	,	0.0417093...],
[690.	,	0.0414651...],
[691.	,	0.0412226...],
[692.	,	0.0409820...],
[693.	,	0.0407431...],

(continues on next page)

(continued from previous page)

[694.	,	0.0405059...],
[695.	,	0.0402705...],
[696.	,	0.0400368...],
[697.	,	0.0398047...],
[698.	,	0.0395744...],
[699.	,	0.0393457...],
[700.	,	0.0391187...],
[701.	,	0.0388933...],
[702.	,	0.0386696...],
[703.	,	0.0384474...],
[704.	,	0.0382269...],
[705.	,	0.0380079...],
[706.	,	0.0377905...],
[707.	,	0.0375747...],
[708.	,	0.0373604...],
[709.	,	0.0371476...],
[710.	,	0.0369364...],
[711.	,	0.0367266...],
[712.	,	0.0365184...],
[713.	,	0.0363116...],
[714.	,	0.0361063...],
[715.	,	0.0359024...],
[716.	,	0.0357000...],
[717.	,	0.0354990...],
[718.	,	0.0352994...],
[719.	,	0.0351012...],
[720.	,	0.0349044...],
[721.	,	0.0347090...],
[722.	,	0.0345150...],
[723.	,	0.0343223...],
[724.	,	0.0341310...],
[725.	,	0.0339410...],
[726.	,	0.0337523...],
[727.	,	0.033565 ...],
[728.	,	0.0333789...],
[729.	,	0.0331941...],
[730.	,	0.0330106...],
[731.	,	0.0328284...],
[732.	,	0.0326474...],
[733.	,	0.0324677...],
[734.	,	0.0322893...],
[735.	,	0.0321120...],
[736.	,	0.0319360...],
[737.	,	0.0317611...],
[738.	,	0.0315875...],
[739.	,	0.0314151...],
[740.	,	0.0312438...],
[741.	,	0.0310737...],
[742.	,	0.0309048...],
[743.	,	0.0307370...],
[744.	,	0.0305703...],
[745.	,	0.0304048...],
[746.	,	0.0302404...],
[747.	,	0.0300771...],
[748.	,	0.0299149...],
[749.	,	0.0297538...],

(continues on next page)

(continued from previous page)

```

[ 750.      , 0.0295938...],
[ 751.      , 0.0294349...],
[ 752.      , 0.0292771...],
[ 753.      , 0.0291203...],
[ 754.      , 0.0289645...],
[ 755.      , 0.0288098...],
[ 756.      , 0.0286561...],
[ 757.      , 0.0285035...],
[ 758.      , 0.0283518...],
[ 759.      , 0.0282012...],
[ 760.      , 0.0280516...],
[ 761.      , 0.0279030...],
[ 762.      , 0.0277553...],
[ 763.      , 0.0276086...],
[ 764.      , 0.027463 ...],
[ 765.      , 0.0273182...],
[ 766.      , 0.0271744...],
[ 767.      , 0.0270316...],
[ 768.      , 0.0268897...],
[ 769.      , 0.0267487...],
[ 770.      , 0.0266087...],
[ 771.      , 0.0264696...],
[ 772.      , 0.0263314...],
[ 773.      , 0.0261941...],
[ 774.      , 0.0260576...],
[ 775.      , 0.0259221...],
[ 776.      , 0.0257875...],
[ 777.      , 0.0256537...],
[ 778.      , 0.0255208...],
[ 779.      , 0.0253888...],
[ 780.      , 0.0252576...]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})

```

colour.scattering_cross_section

```

colour.scattering_cross_section(wavelength, CO2_concentration=300, temperature=288.15,
                                avogadro_constant=6.02214179e+23, n_s=<function
                                air_refraction_index_Bodhaine1999>, F_air=<function
                                F_air_Bodhaine1999>)

```

Returns the scattering cross section per molecule σ of dry air as function of wavelength λ in centimeters (cm) using given CO_2 concentration in parts per million (ppm) and temperature $T[K]$ in kelvin degrees following *Van de Hulst (1957)* method.

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.

- **avogadro_constant** (numeric or array_like, optional) – *Avogadro's* number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(\text{air})$ or *King Factor* computation method.

Returns Scattering cross section per molecule σ of dry air.

Return type numeric or ndarray

Warning: Unlike most objects of `colour.phenomena.rayleigh` module, `colour.scattering_cross_section()` expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> scattering_cross_section(555 * 10e-8) # doctest: +ELLIPSIS
4.6613309...e-27
```

`colour.phenomena`

<code>rayleigh_optical_depth(wavelength[, ...])</code>	Returns the <i>Rayleigh</i> optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
--	---

`colour.phenomena.rayleigh_optical_depth`

`colour.phenomena.rayleigh_optical_depth(wavelength, CO2_concentration=300, temperature=288.15, pressure=101325, latitude=0, altitude=0, avogadro_constant=6.02214179e+23, n_s=<function air_refraction_index_Bodhaine1999>, F_air=<function F_air_Bodhaine1999>)`

Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric or array_like) – Surface pressure P of the measurement site.
- **latitude** (numeric or array_like, optional) – Latitude of the site in degrees.
- **altitude** (numeric or array_like, optional) – Altitude of the site in meters.

- **avogadro_constant** (numeric or array_like, optional) – *Avogadro's* number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6 + 3_p)/(6 - 7_p)$, the depolarisation term $F(\text{air})$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type numeric or ndarray

Warning: Unlike most objects of `colour.phenomena.rayleigh` module, `colour.phenomena.rayleigh_optical_depth()` expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8) # doctest: +ELLIPSIS
0.1004070...
```

Plotting

- *Common*
- *Colorimetry*
- *Colour Vision Deficiency*
- *Colour Characterisation*
- *Corresponding Chromaticities*
- *CIE Chromaticity Diagrams*
- *Colour Models*
- *Colour Notation Systems*
- *Optical Phenomena*
- *Colour Quality*
- *Colour Temperature & Correlated Colour Temperature*
- *Colour Models Volume*
- *Geometry Plotting Utilities*

Common

`colour.plotting`

<code>colour_style([use_style])</code>	Returns <i>Colour</i> plotting style.
<code>colour_cycle(**kwargs)</code>	Returns a colour cycle iterator using given colour map.
<code>artist(**kwargs)</code>	Returns the current figure and its axes or creates a new one.
<code>camera(**kwargs)</code>	Sets the camera settings.
<code>render(**kwargs)</code>	Renders the current figure while adjusting various settings such as the bounding box, the title or background transparency.
<code>label_rectangles(labels, rectangles[, ...])</code>	Add labels above given rectangles.
<code>uniform_axes3d(axes)</code>	Sets equal aspect ratio to given 3d axes.
<code>plot_single_colour_swatch(colour_swatch, ...)</code>	Plots given colour swatch.
<code>plot_multi_colour_swatches(colour_swatches)</code>	Plots given colours swatches.
<code>plot_single_function(function[, samples, ...])</code>	Plots given function.
<code>plot_multi_functions(functions[, samples, ...])</code>	Plots given functions.
<code>plot_image(image[, text_parameters, ...])</code>	Plots given image.

`colour.plotting.colour_style`

`colour.plotting.colour_style(use_style=True)`

Returns *Colour* plotting style.

Parameters `use_style` (`bool`, optional) – Whether to use the style and load it into *Matplotlib*.

Returns *Colour* style.

Return type `dict`

`colour.plotting.colour_cycle`

`colour.plotting.colour_cycle(**kwargs)`

Returns a colour cycle iterator using given colour map.

Other Parameters

- **colour_cycle_map** (*unicode* or *LinearSegmentedColormap*, optional) – *Matplotlib* colourmap name.
- **colour_cycle_count** (*int*, optional) – Colours count to pick in the colourmap.

Returns Colour cycle iterator.

Return type `cycle`

`colour.plotting.artist`

`colour.plotting.artist(**kwargs)`

Returns the current figure and its axes or creates a new one.

Other Parameters

- **axes** (*Axes, optional*) – Axes that will be passed through without creating a new figure.
- **uniform** (*unicode, optional*) – Whether to create the figure with an equal aspect ratio.

Returns Figure, axes.

Return type `tuple`

`colour.plotting.camera`

`colour.plotting.camera(**kwargs)`

Sets the camera settings.

Other Parameters

- **azimuth** (*numeric, optional*) – Camera azimuth.
- **camera_aspect** (*unicode, optional*) – Matplotlib axes aspect. Default is *equal*.
- **elevation** (*numeric, optional*) – Camera elevation.

Returns Current axes.

Return type `Axes`

`colour.plotting.render`

`colour.plotting.render(**kwargs)`

Renders the current figure while adjusting various settings such as the bounding box, the title or background transparency.

Other Parameters

- **figure** (*Figure, optional*) – Figure to apply the render elements onto.
- **axes** (*Axes, optional*) – Axes to apply the render elements onto.
- **filename** (*unicode, optional*) – Figure will be saved using given filename argument.
- **standalone** (*bool, optional*) – Whether to show the figure and call `plt.show()` definition.
- **aspect** (*unicode, optional*) – Matplotlib axes aspect.
- **axes_visible** (*bool, optional*) – Whether the axes are visible. Default is *True*.
- **bounding_box** (*array_like, optional*) – Array defining current axes limits such `bounding_box = (x min, x max, y min, y max)`.
- **tight_layout** (*bool, optional*) – Whether to invoke the `plt.tight_layout()` definition.
- **legend** (*bool, optional*) – Whether to display the legend. Default is *False*.
- **legend_columns** (*int, optional*) – Number of columns in the legend. Default is *1*.
- **transparent_background** (*bool, optional*) – Whether to turn off the background patch. Default is *False*.

- **title** (*unicode, optional*) – Figure title.
- **wrap_title** (*unicode, optional*) – Whether to wrap the figure title, the default is to wrap at a number of characters equal to the width of the figure multiplied by 10.
- **x_label** (*unicode, optional*) – X axis label.
- **y_label** (*unicode, optional*) – Y axis label.
- **x_ticker** (*bool, optional*) – Whether to display the X axis ticker. Default is *True*.
- **y_ticker** (*bool, optional*) – Whether to display the Y axis ticker. Default is *True*.

Returns Current figure and axes.

Return type `tuple`

`colour.plotting.label_rectangles`

`colour.plotting.label_rectangles(labels, rectangles, rotation='vertical', text_size=10, offset=None, **kwargs)`

Add labels above given rectangles.

Parameters

- **labels** (*array_like*) – Labels to display.
- **rectangles** (*object*) – Rectangles to used to set the labels value and position.
- **rotation** (*unicode, optional*) – {'horizontal', 'vertical'}, Labels orientation.
- **text_size** (*numeric, optional*) – Labels text size.
- **offset** (*array_like, optional*) – Labels offset as percentages of the largest rectangle dimensions.

Other Parameters **axes** (*Axes, optional*) – Axes to use for plotting.

Returns Definition success.

Return type `bool`

`colour.plotting.uniform_axes3d`

`colour.plotting.uniform_axes3d(axes)`

Sets equal aspect ratio to given 3d axes.

Parameters **axes** (*object*) – Axis to set the equal aspect ratio.

Returns Definition success.

Return type `bool`

`colour.plotting.plot_single_colour_swatch`

`colour.plotting.plot_single_colour_swatch(colour_swatch, **kwargs)`

Plots given colour swatch.

Parameters **colour_swatch** (*ColourSwatch*) – ColourSwatch.

Other Parameters

- ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.
- **width** (*numeric, optional*) – {`colour.plotting.plot_multi_colour_swatches()`}, Colour swatch width.
- **height** (*numeric, optional*) – {`colour.plotting.plot_multi_colour_swatches()`}, Colour swatch height.
- **spacing** (*numeric, optional*) – {`colour.plotting.plot_multi_colour_swatches()`}, Colour swatches spacing.
- **columns** (*int, optional*) – {`colour.plotting.plot_multi_colour_swatches()`}, Colour swatches columns count.
- **text_parameters** (*dict, optional*) – {`colour.plotting.plot_multi_colour_swatches()`}, Parameters for the `plt.text()` definition, offset can be set to define the text offset.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> RGB = ColourSwatch(RGB=(0.45620519, 0.03081071, 0.04091952))
>>> plot_single_colour_swatch(RGB) # doctest: +SKIP
```



colour.plotting.plot_multi_colour_swatches

```
colour.plotting.plot_multi_colour_swatches(colour_swatches, width=1, height=1, spacing=0,
                                           columns=None, text_parameters=None,
                                           background_colour=(1.0, 1.0, 1.0), compare_swatches=None,
                                           **kwargs)
```

Plots given colours swatches.

Parameters

- **colour_swatches** (*list*) – Colour swatch sequence.
- **width** (numeric, optional) – Colour swatch width.
- **height** (numeric, optional) – Colour swatch height.
- **spacing** (numeric, optional) – Colour swatches spacing.
- **columns** (*int*, optional) – Colour swatches columns count, defaults to the colour swatch count or half of it if comparing.
- **text_parameters** (*dict*, optional) – Parameters for the `plt.text()` definition, visible can be set to make the text visible, “offset” can be set to define the text offset.
- **background_colour** (array_like or unicode, optional) – Background colour.
- **compare_swatches** (unicode, optional) – {**None**, **‘Stacked’**, **‘Diagonal’**}, Whether to compare the swatches, in which case the colour swatch count must be an even number with alternating reference colour swatches and test colour swatches. *Stacked* will draw the test colour swatch in the center of the reference colour swatch, *Diagonal* will draw the reference colour swatch in the upper left diagonal area and the test colour swatch in the bottom right diagonal area.

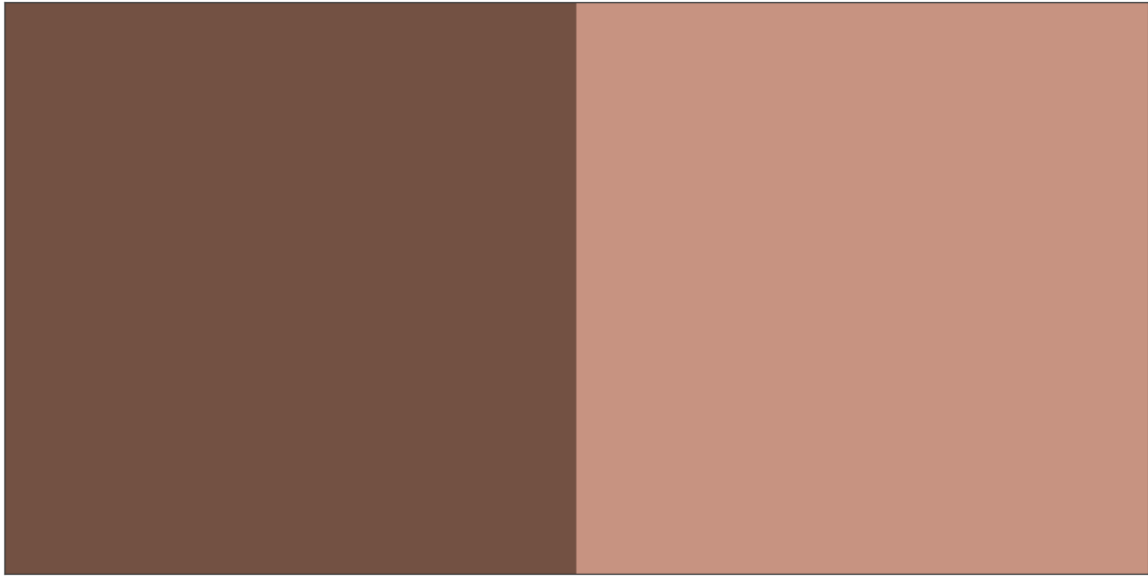
Other Parameters ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type *tuple*

Examples

```
>>> RGB_1 = ColourSwatch(
    RGB=(0.45293517, 0.31732158, 0.26414773))
>>> RGB_2 = ColourSwatch(
    RGB=(0.77875824, 0.57726450, 0.50453169))
>>> plot_multi_colour_swatches([RGB_1, RGB_2]) # doctest: +SKIP
```



`colour.plotting.plot_single_function`

`colour.plotting.plot_single_function(function, samples=None, log_x=None, log_y=None, **kwargs)`

Plots given function.

Parameters

- **function** (callable, optional) – Function to plot.
- **samples** (array_like, optional,) – Samples to evaluate the functions with.
- **log_x** (int, optional) – Log base to use for the x axis scale, if *None*, the x axis scale will be linear.
- **log_y** (int, optional) – Log base to use for the y axis scale, if *None*, the y axis scale will be linear.

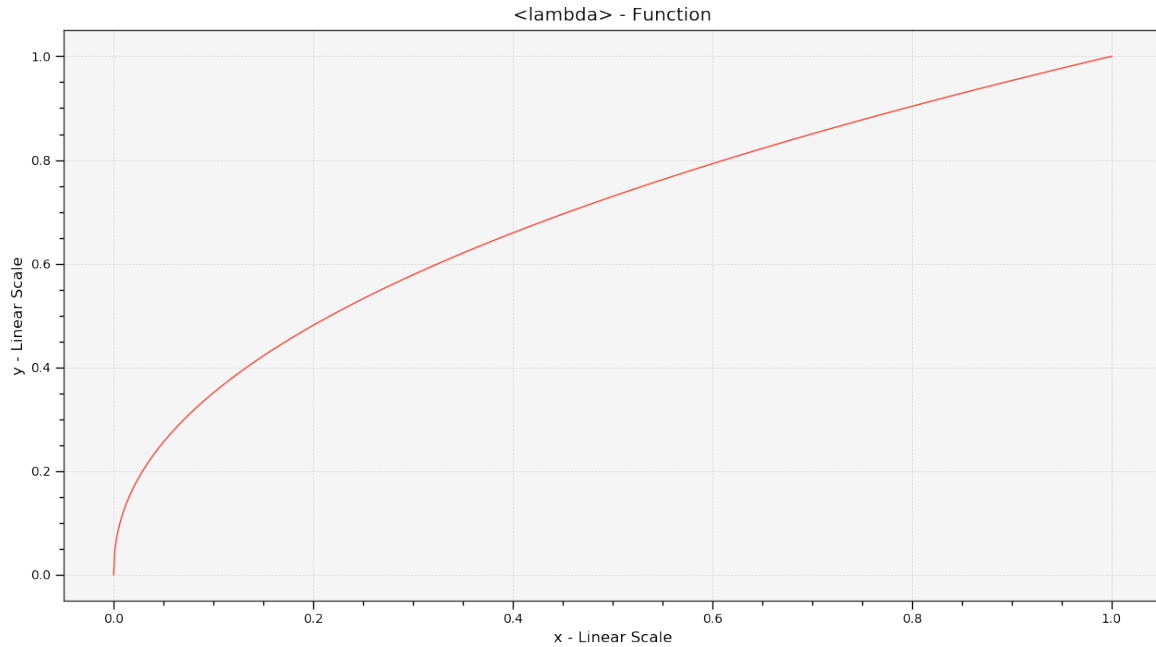
Other Parameters ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_function(partial(gamma_function, exponent=1 / 2.2))
... # doctest: +SKIP
```



colour.plotting.plot_multi_functions

`colour.plotting.plot_multi_functions(functions, samples=None, log_x=None, log_y=None, **kwargs)`

Plots given functions.

Parameters

- **functions** (`dict`) – Functions to plot.
- **samples** (`array_like`, optional,) – Samples to evaluate the functions with.
- **log_x** (`int`, optional) – Log base to use for the x axis scale, if `None`, the x axis scale will be linear.
- **log_y** (`int`, optional) – Log base to use for the y axis scale, if `None`, the y axis scale will be linear.

Other Parameters `**kwargs` (`dict`, optional) – `{colour.plotting.artist(), colour.plotting.render()}`, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

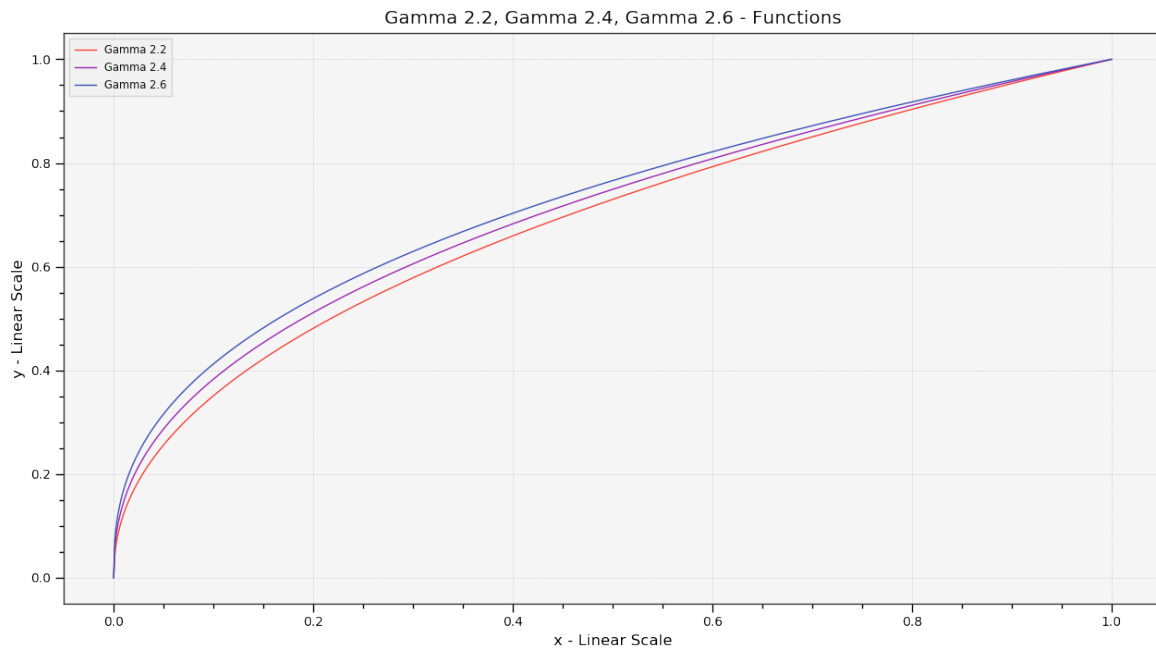
Examples

```
>>> functions = {
...     'Gamma 2.2' : lambda x: x ** (1 / 2.2),
...     'Gamma 2.4' : lambda x: x ** (1 / 2.4),
...     'Gamma 2.6' : lambda x: x ** (1 / 2.6),
... }
```

(continues on next page)

(continued from previous page)

```
>>> plot_multi_functions(functions)
... # doctest: +SKIP
```



colour.plotting.plot_image

```
colour.plotting.plot_image(image, text_parameters=None, interpolation='nearest',
                           colour_map=<matplotlib.colors.LinearSegmentedColormap object>,
                           **kwargs)
```

Plots given image.

Parameters

- **image** (array_like) – Image to plot.
- **text_parameters** (dict, optional) – Parameters for the `plt.text()` definition, offset can be set to define the text offset.
- **interpolation** (unicode, optional) – {'nearest', None, 'none', 'bilinear', 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos'} Image display interpolation.
- **colour_map** (unicode, optional) – Colour map used to display single channel images.

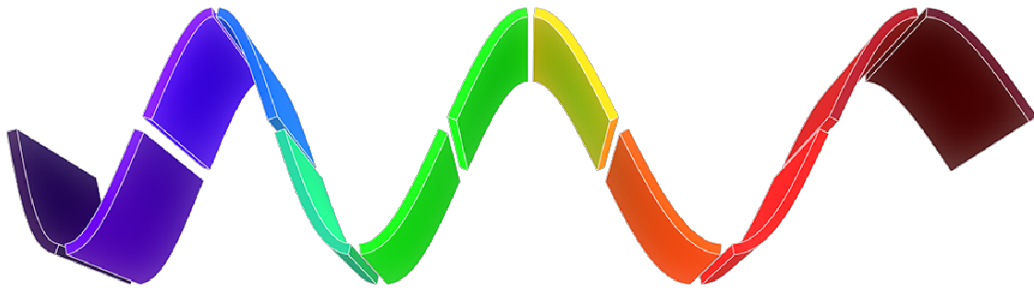
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> import os
>>> import colour
>>> from colour import read_image
>>> path = os.path.join(
...     colour.__path__[0], '..', 'docs', '_static', 'Logo_Medium_001.png')
>>> plot_image(read_image(path)) # doctest: +SKIP
```



Colorimetry

colour.plotting

<code>plot_single_sd(sd[, cmfs, ...])</code>	Plots given spectral distribution.
<code>plot_multi_sds(sds[, cmfs, use_sds_colours, ...])</code>	Plots given spectral distributions.
<code>plot_single_cmfs([cmfs])</code>	Plots given colour matching functions.
<code>plot_multi_cmfs([cmfs])</code>	Plots given colour matching functions.
<code>plot_single_illuminant_sd([illuminant, cmfs])</code>	Plots given single illuminant spectral distribution.
<code>plot_multi_illuminant_sds([illuminants])</code>	Plots given illuminants spectral distributions.
<code>plot_visible_spectrum([cmfs, ...])</code>	Plots the visible colours spectrum using given standard observer <i>CIE XYZ</i> colour matching functions.
<code>plot_single_lightness_function([function])</code>	Plots given <i>Lightness</i> function.
<code>plot_multi_lightness_functions([functions])</code>	Plots given <i>Lightness</i> functions.
<code>plot_single_luminance_function([function])</code>	Plots given <i>Luminance</i> function.
<code>plot_multi_luminance_functions([functions])</code>	Plots given <i>Luminance</i> functions.
<code>plot_blackbody_spectral_radiance([...])</code>	Plots given blackbody spectral radiance.
<code>plot_blackbody_colours([shape, cmfs])</code>	Plots blackbody colours.

colour.plotting.plot_single_sd

```
colour.plotting.plot_single_sd(sd, cmfs='CIE 1931 2 Degree Standard Observer', out_of_gamut_clipping=True, modulate_colours_with_sd_amplitude=False, equalize_sd_amplitude=False, **kwargs)
```

Plots given spectral distribution.

Parameters

- **sd** ([SpectralDistribution](#)) – Spectral distribution to plot.
- **out_of_gamut_clipping** ([bool](#), optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.
- **modulate_colours_with_sd_amplitude** ([bool](#), optional) – Whether to modulate the colours with the spectral distribution amplitude.
- **equalize_sd_amplitude** ([bool](#), optional) – Whether to equalize the spectral distribution amplitude. Equalization occurs after the colours modulation thus setting both arguments to *True* will generate a spectrum strip where each wavelength colour is modulated by the spectral distribution amplitude. The usual 5% margin above the spectral distribution is also omitted.
- **cmfs** (unicode) – Standard observer colour matching functions used for spectrum creation.

Other Parameters ****kwargs** (*dict*, optional) – {[colour.plotting.artist\(\)](#), [colour.plotting.render\(\)](#)}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

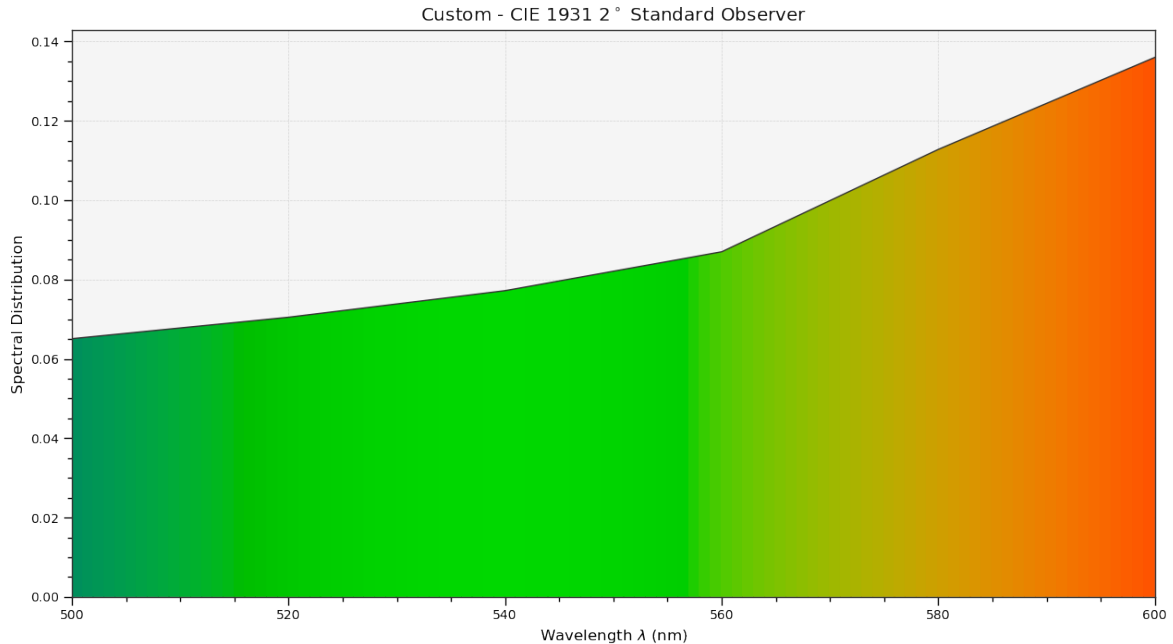
Return type [tuple](#)

References

[Spi15]

Examples

```
>>> from colour import SpectralDistribution
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data, name='Custom')
>>> plot_single_sd(sd) # doctest: +SKIP
```



colour.plotting.plot_multi_sds

`colour.plotting.plot_multi_sds(sds, cmfs='CIE 1931 2 Degree Standard Observer', use_sds_colours=False, normalise_sds_colours=False, **kwargs)`
 Plots given spectral distributions.

Parameters

- **sds** (array_like or `MultiSpectralDistribution`) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single `colour.MultiSpectralDistribution` class instance, a list of `colour.MultiSpectralDistribution` class instances or a list of `colour.SpectralDistribution` class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.
- **use_sds_colours** (bool, optional) – Whether to use spectral distributions colours.
- **normalise_sds_colours** (bool) – Whether to normalise spectral distributions colours.

Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

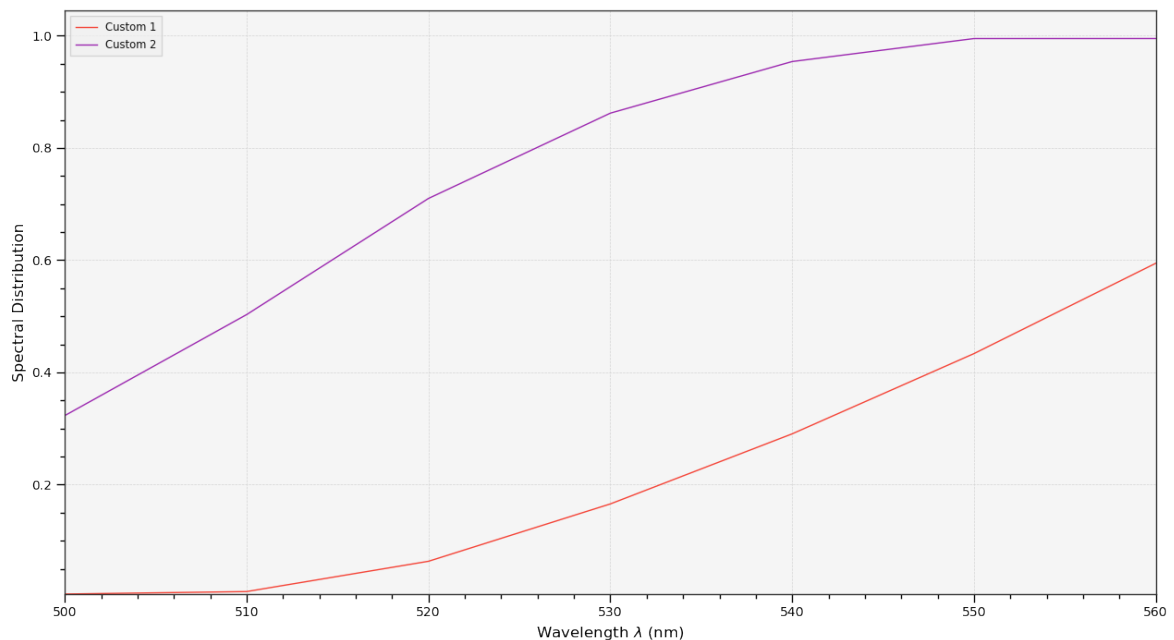
Return type tuple

Examples

```

>>> from colour import SpectralDistribution
>>> data_1 = {
...     500: 0.004900,
...     510: 0.009300,
...     520: 0.063270,
...     530: 0.165500,
...     540: 0.290400,
...     550: 0.433450,
...     560: 0.594500
... }
>>> data_2 = {
...     500: 0.323000,
...     510: 0.503000,
...     520: 0.710000,
...     530: 0.862000,
...     540: 0.954000,
...     550: 0.994950,
...     560: 0.995000
... }
>>> spd1 = SpectralDistribution(data_1, name='Custom 1')
>>> spd2 = SpectralDistribution(data_2, name='Custom 2')
>>> plot_multi_sds([spd1, spd2]) # doctest: +SKIP

```



colour.plotting.plot_single_cmfs

`colour.plotting.plot_single_cmfs(cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)`
 Plots given colour matching functions.

Parameters `cmfs` (unicode, optional) – Colour matching functions to plot.

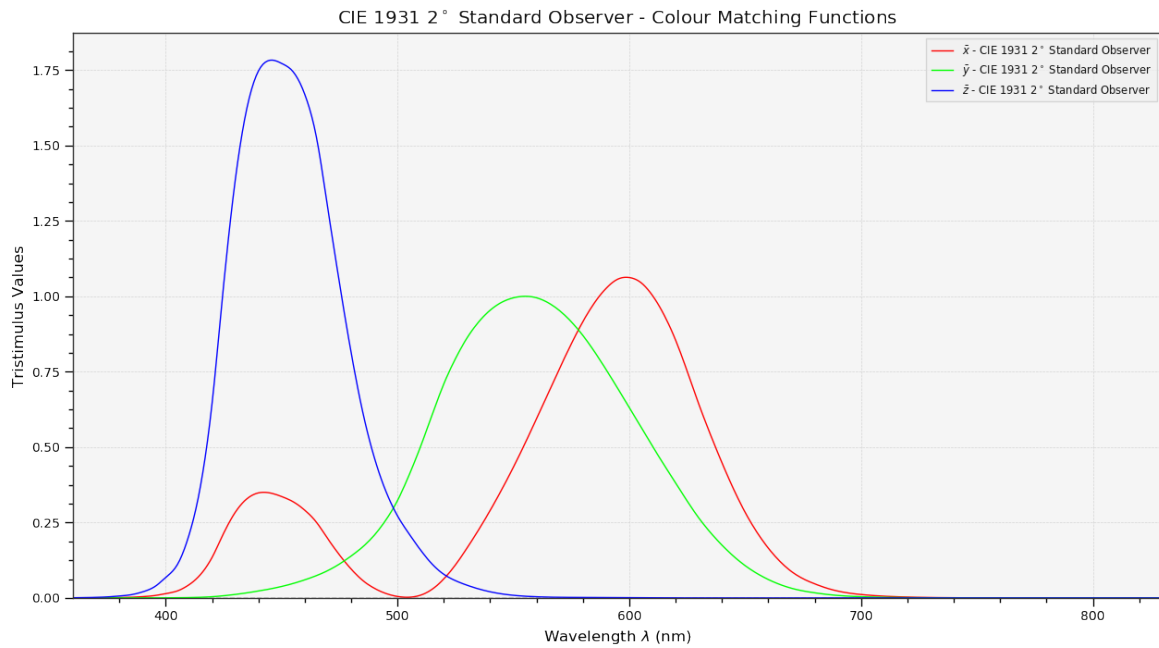
Other Parameters `**kwargs` (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_cmfs()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_cmfs('CIE 1931 2 Degree Standard Observer')
... # doctest: +SKIP
```



`colour.plotting.plot_multi_cmfs`

`colour.plotting.plot_multi_cmfs(cmfs=None, **kwargs)`

Plots given colour matching functions.

Parameters `cmfs` (`array_like`, optional) – Colour matching functions to plot.

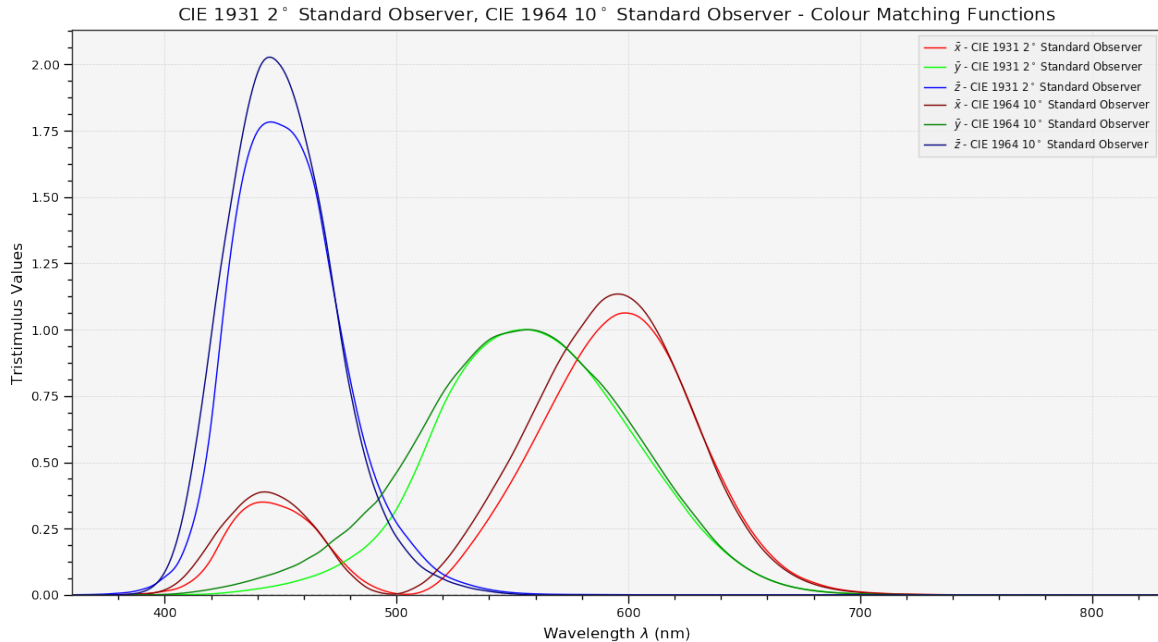
Other Parameters `**kwargs` (`dict`, optional) – `{colour.plotting.artist(), colour.plotting.render()}`, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> cmfs = ('CIE 1931 2 Degree Standard Observer',
...         'CIE 1964 10 Degree Standard Observer')
>>> plot_multi_cmfs(cmfs) # doctest: +SKIP
```



colour.plotting.plot_single_illuminant_sd

`colour.plotting.plot_single_illuminant_sd(illuminant='A', cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)`

Plots given single illuminant spectral distribution.

Parameters

- **illuminant** (unicode, optional) – Factory illuminant to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions to plot.

Other Parameters

- ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.
- **out_of_gamut_clipping** (bool, optional) – {`colour.plotting.plot_single_sd()`}, Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Returns Current figure and axes.

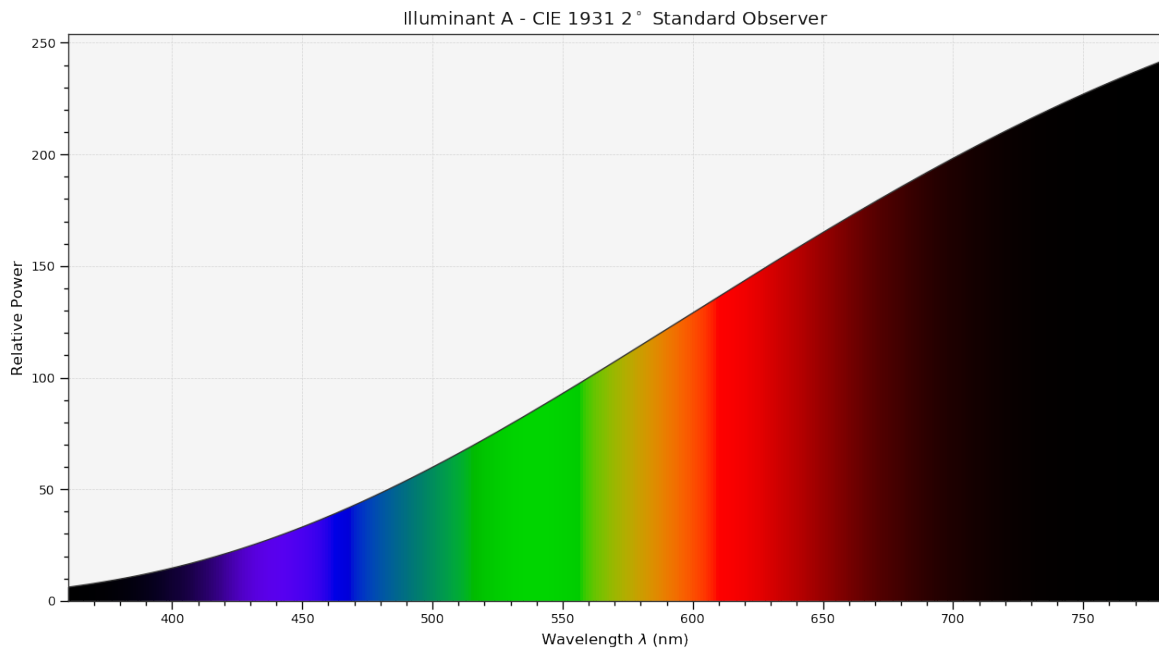
Return type tuple

References

[Spi15]

Examples

```
>>> plot_single_illuminant_sd('A') # doctest: +SKIP
```



colour.plotting.plot_multi_illuminant_sds

colour.plotting.plot_multi_illuminant_sds(illuminants=None, **kwargs)

Plots given illuminants spectral distributions.

Parameters `illuminants` (array_like, optional) – Factory illuminants to plot.

Other Parameters

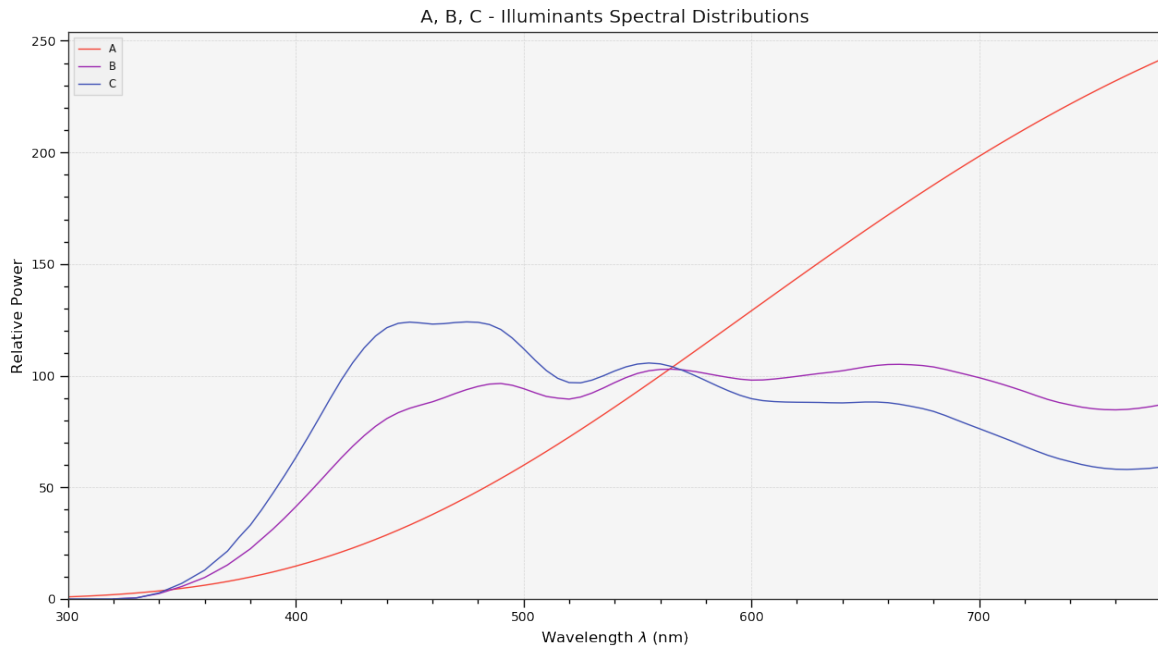
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_sds(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **use_sds_colours** (bool, optional) – {colour.plotting.plot_multi_sds()} Whether to use spectral distributions colours.
- **normalise_sds_colours** (bool) – {colour.plotting.plot_multi_sds()} Whether to normalise spectral distributions colours.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_illuminant_sds(['A', 'B', 'C']) # doctest: +SKIP
```



`colour.plotting.plot_visible_spectrum`

`colour.plotting.plot_visible_spectrum(cmfs='CIE 1931 2 Degree Standard Observer',
out_of_gamut_clipping=True, **kwargs)`

Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.
- **out_of_gamut_clipping** (bool, optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

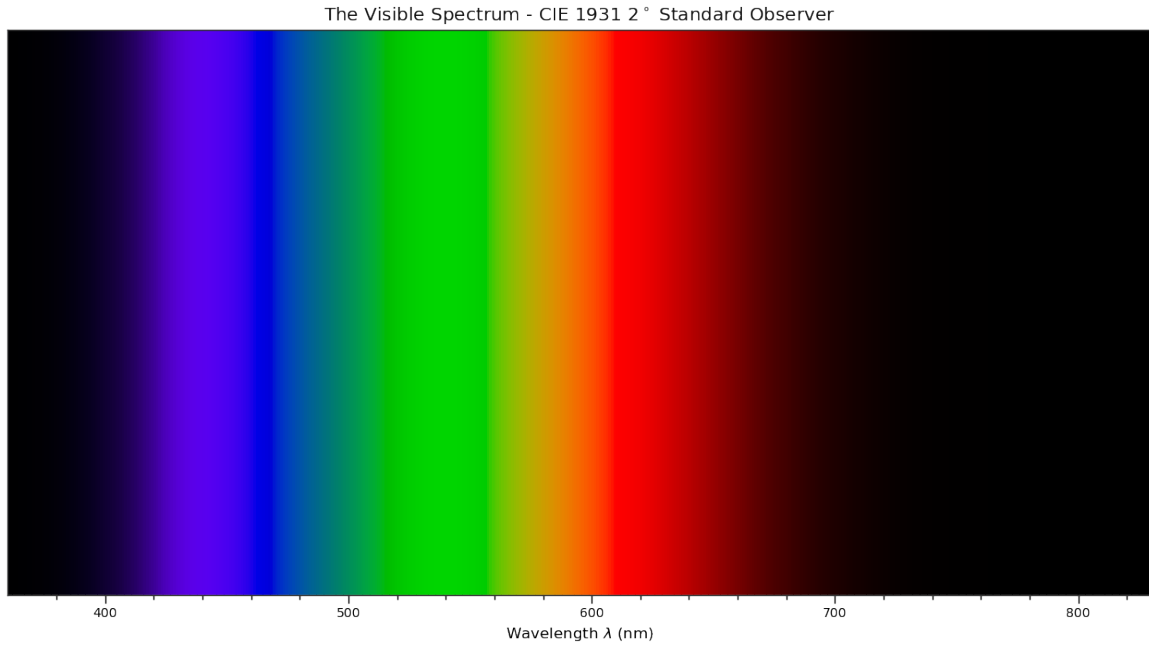
Return type tuple

References

[Spi15]

Examples

```
>>> plot_visible_spectrum() # doctest: +SKIP
```

`colour.plotting.plot_single_lightness_function`

`colour.plotting.plot_single_lightness_function(function='CIE 1976', **kwargs)`

Plots given *Lightness* function.

Parameters `function` (unicode, optional) – *Lightness* function to plot.

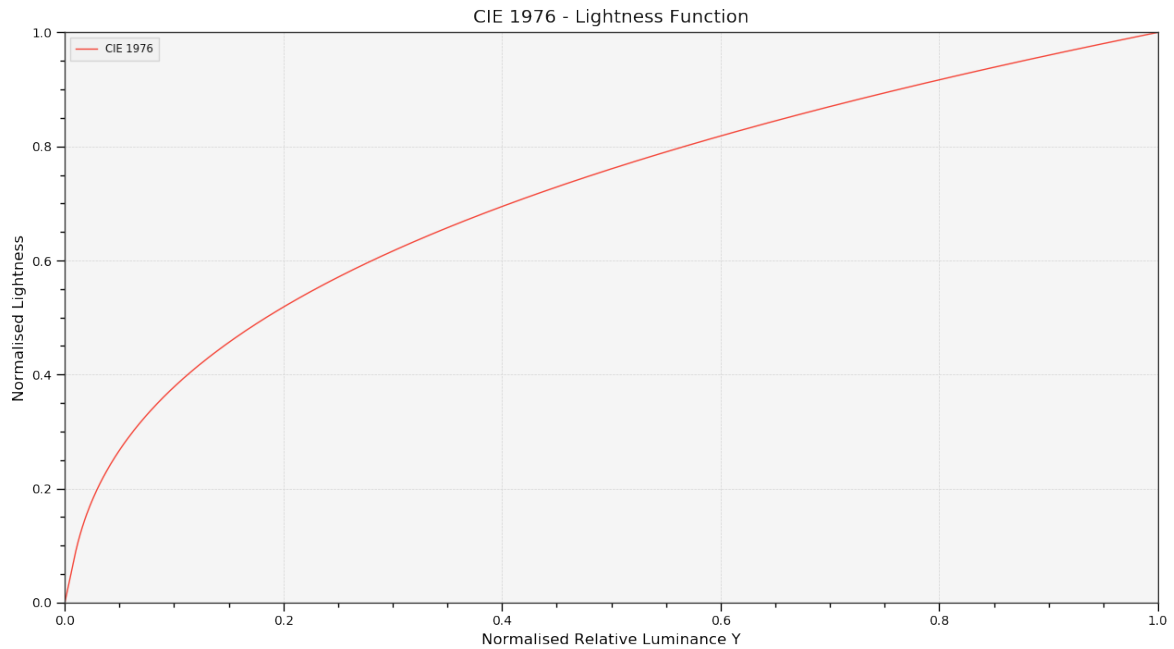
Other Parameters `**kwargs` (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_lightness_function('CIE 1976') # doctest: +SKIP
```



`colour.plotting.plot_multi_lightness_functions`

`colour.plotting.plot_multi_lightness_functions(functions=None, **kwargs)`

Plots given *Lightness* functions.

Parameters `functions` (array_like, optional) – *Lightness* functions to plot.

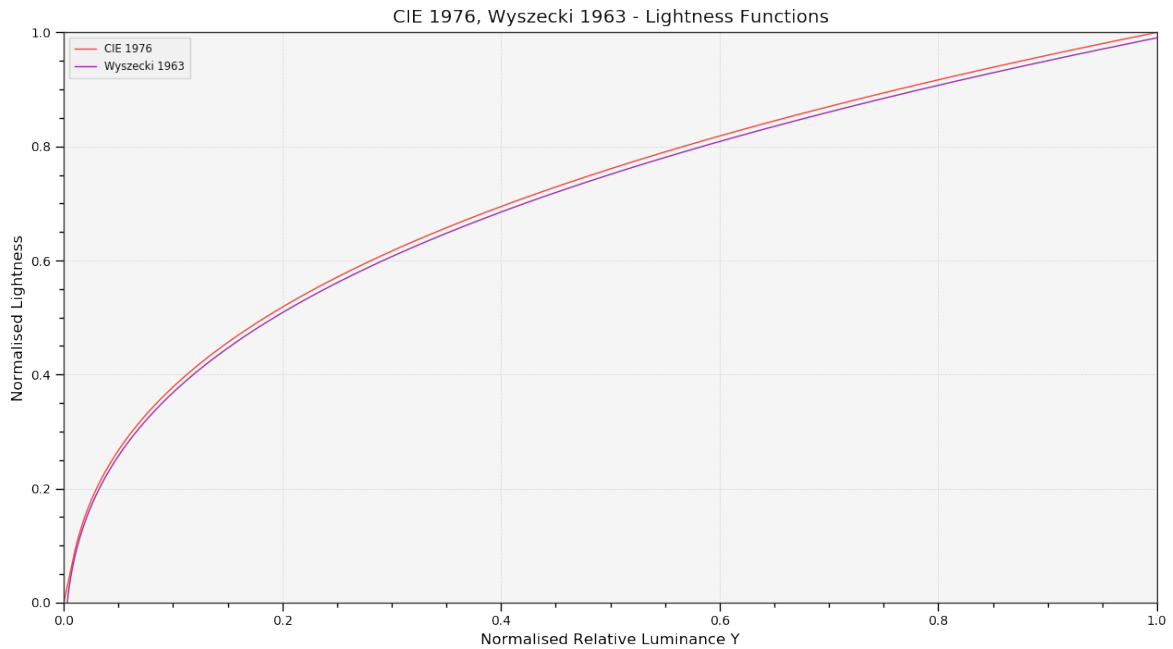
Other Parameters `**kwargs` (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_lightness_functions(['CIE 1976', 'Wyszecki 1963'])
... # doctest: +SKIP
```



`colour.plotting.plot_single_luminance_function`

`colour.plotting.plot_single_luminance_function(function='CIE 1976', **kwargs)`

Plots given *Luminance* function.

Parameters `function` (unicode, optional) – *Luminance* function to plot.

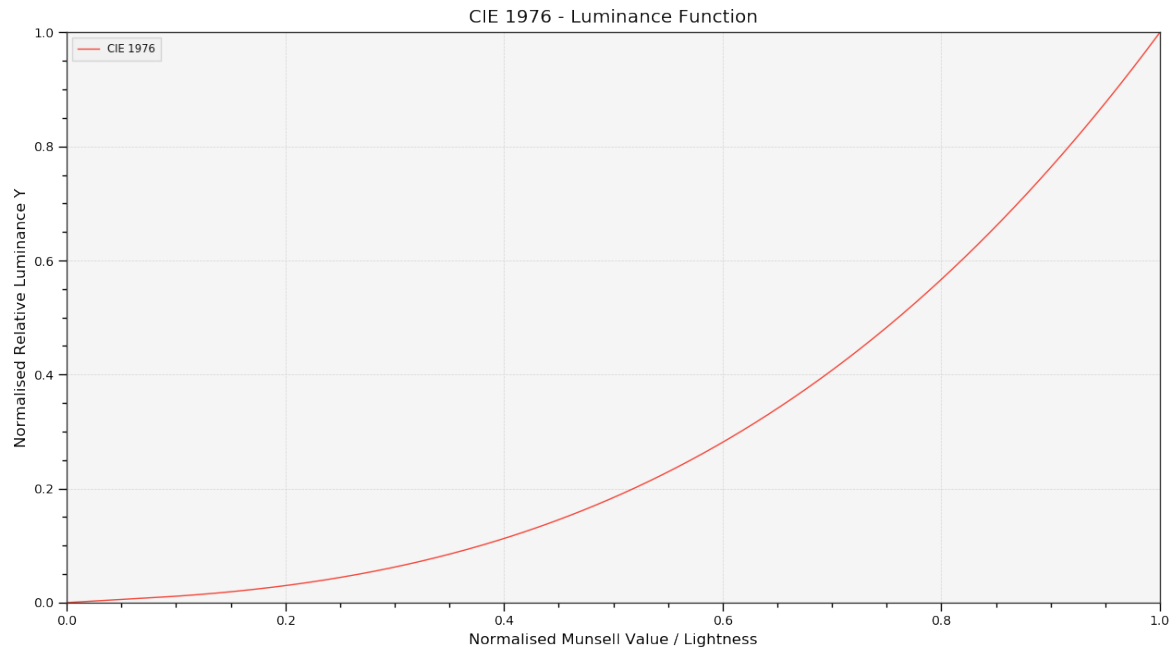
Other Parameters `**kwargs` (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_luminance_function('CIE 1976') # doctest: +SKIP
```



`colour.plotting.plot_multi_luminance_functions`

`colour.plotting.plot_multi_luminance_functions(functions=None, **kwargs)`

Plots given *Luminance* functions.

Parameters `functions` (array_like, optional) – *Luminance* functions to plot.

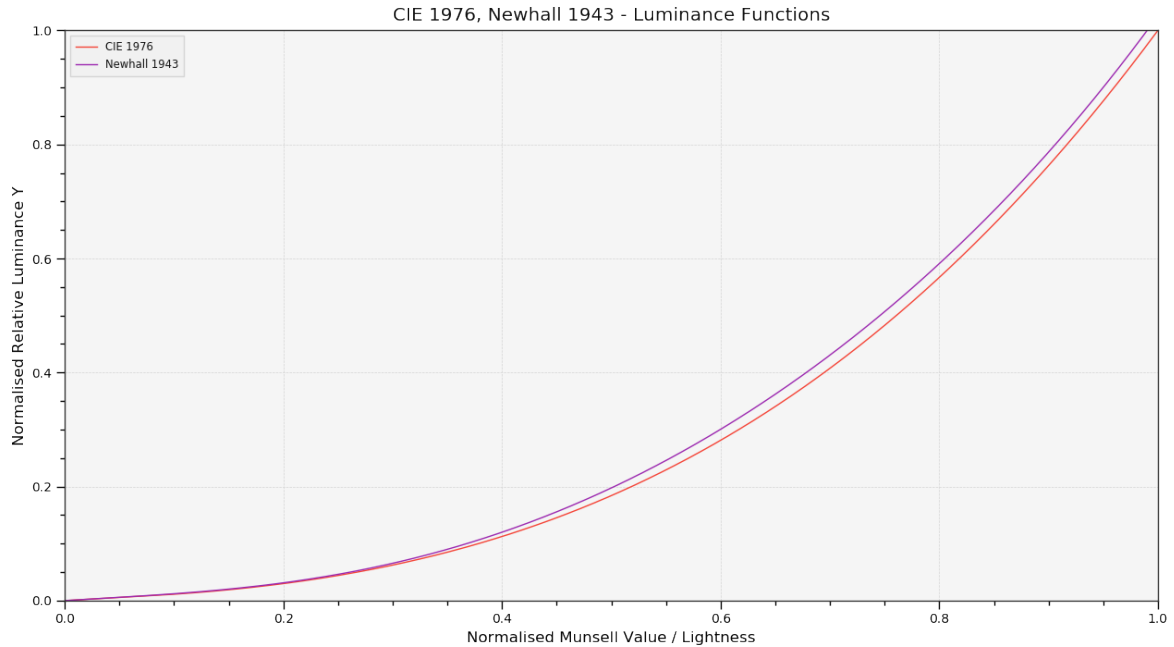
Other Parameters `**kwargs` (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_luminance_functions(['CIE 1976', 'Newhall 1943'])
... # doctest: +SKIP
```



`colour.plotting.plot_blackbody_spectral_radiance`

`colour.plotting.plot_blackbody_spectral_radiance(temperature=3500, cmfs='CIE 1931 2 Degree Standard Observer', blackbody='VY Canis Major', **kwargs)`

Plots given blackbody spectral radiance.

Parameters

- **temperature** (numeric, optional) – Blackbody temperature.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.
- **blackbody** (unicode, optional) – Blackbody name.

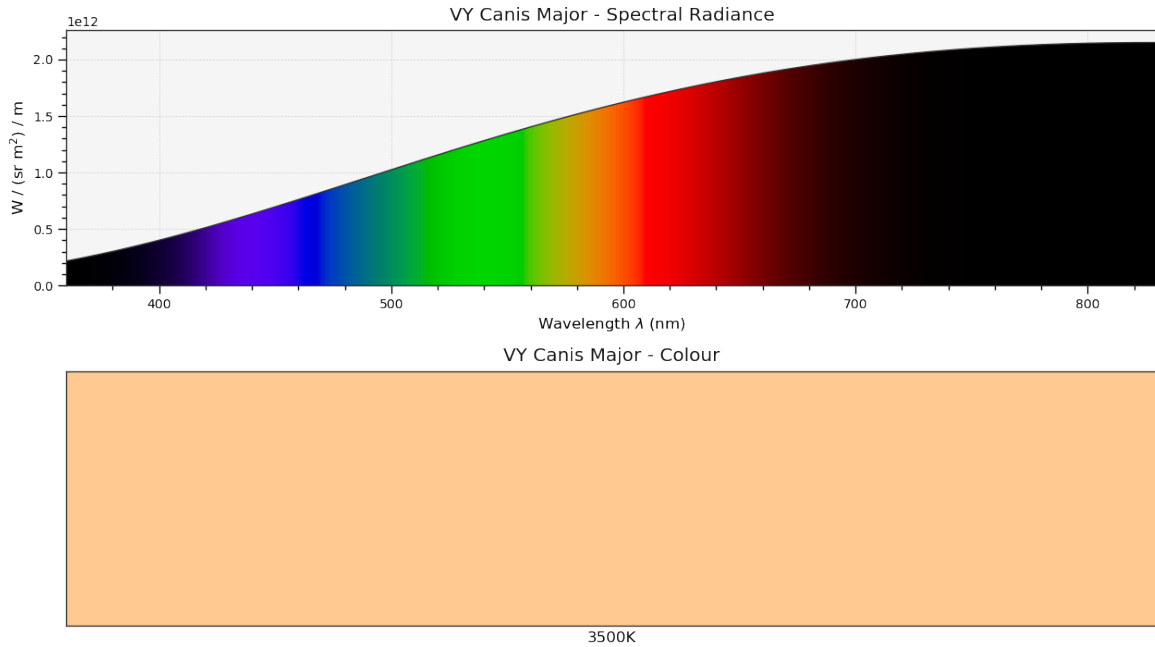
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_blackbody_spectral_radiance(3500, blackbody='VY Canis Major')
... # doctest: +SKIP
```



`colour.plotting.plot_blackbody_colours`

`colour.plotting.plot_blackbody_colours(shape=SpectralShape(150, 12500, 50), cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)`

Plots blackbody colours.

Parameters

- **shape** (`SpectralShape`, optional) – Spectral shape to use as plot boundaries.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.

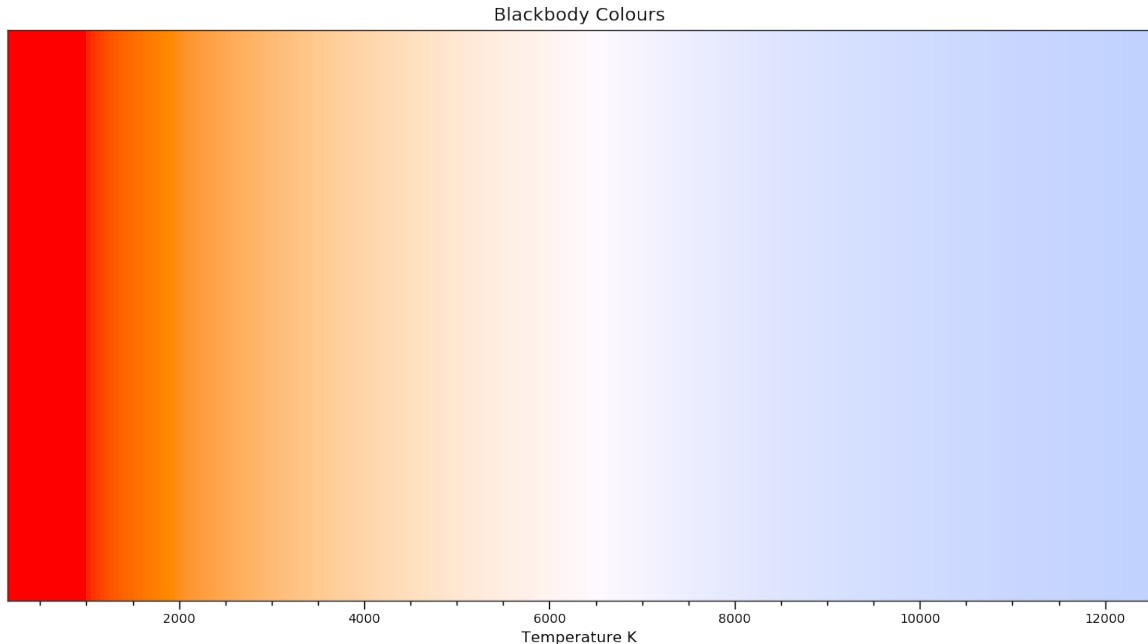
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_blackbody_colours(SpectralShape(150, 12500, 50)) # doctest: +SKIP
```



Colour Vision Deficiency

colour.plotting

<code>plot_cvd_simulation_Machado2009(</code> <code>RGB[, ...]</code> <code>)</code>	Performs colour vision deficiency simulation on given <i>RGB</i> colourspace array using <i>Machado et al. (2009)</i> model..
--	---

colour.plotting.plot_cvd_simulation_Machado2009

`colour.plotting.plot_cvd_simulation_Machado2009`(*RGB*, *deficiency*=*'Protanomaly'*, *severity*=*0.5*, *M_a*=*None*, ***kwargs*)
 Performs colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al. (2009)* model.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **deficiency** (unicode, optional) – {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'}
Colour blindness / vision deficiency type.
- **severity** (numeric, optional) – Severity of the colour vision deficiency in domain [0, 1].
- **M_a** (array_like, optional) – Anomalous trichromacy matrix to use instead of Machado (2010) pre-computed matrix.

Other Parameters ***kwargs* (*dict*, *optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_image()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Notes

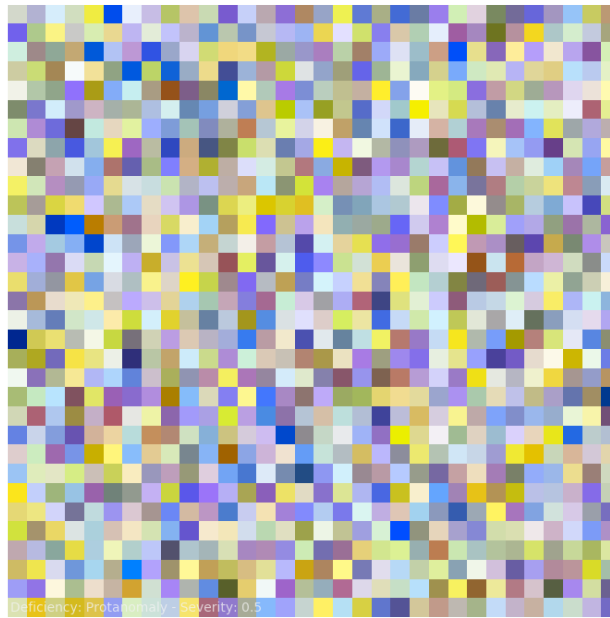
- Input *RGB* array is expected to be linearly encoded.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> import numpy as np
>>> RGB = np.random.rand(32, 32, 3)
>>> plot_cvd_simulation_Machado2009(RGB) # doctest: +SKIP
```



Colour Characterisation

`colour.plotting`

<code>plot_single_colour_checker([colour_checker])</code>	Plots given colour checker.
<code>plot_multi_colour_checkers([colour_checkers])</code>	Plots and compares given colour checkers.

`colour.plotting.plot_single_colour_checker`

`colour.plotting.plot_single_colour_checker(colour_checker='ColorChecker 2005', **kwargs)`
Plots given colour checker.

Parameters `colour_checker` (unicode, optional) – Color checker name.

Other Parameters `**kwargs` (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, Please refer to

the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_colour_checker('ColorChecker 2005') # doctest: +SKIP
```



`colour.plotting.plot_multi_colour_checkers`

`colour.plotting.plot_multi_colour_checkers(colour_checkers=None, **kwargs)`

Plots and compares given colour checkers.

Parameters `colour_checkers` (array_like, optional) – Color checker names, must be less than or equal to 2 names.

Other Parameters `**kwargs` (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_multi_colour_checkers(['ColorChecker 1976', 'ColorChecker 2005'])
... # doctest: +SKIP
```



Corresponding Chromaticities

`colour.plotting`

`plot_corresponding_chromaticities_prediction([. . .])` Plots given chromatic adaptation model corresponding chromaticities prediction.

`colour.plotting.plot_corresponding_chromaticities_prediction`

`colour.plotting.plot_corresponding_chromaticities_prediction(experiment=1, model='Von Kries', transform='CAT02', **kwargs)`

Plots given chromatic adaptation model corresponding chromaticities prediction.

Parameters

- **experiment** (`int`, optional) – Corresponding chromaticities prediction experiment number.
- **model** (unicode, optional) – Corresponding chromaticities prediction model name.
- **transform** (unicode, optional) – Transformation to use with *Von Kries* chromatic adaptation model.

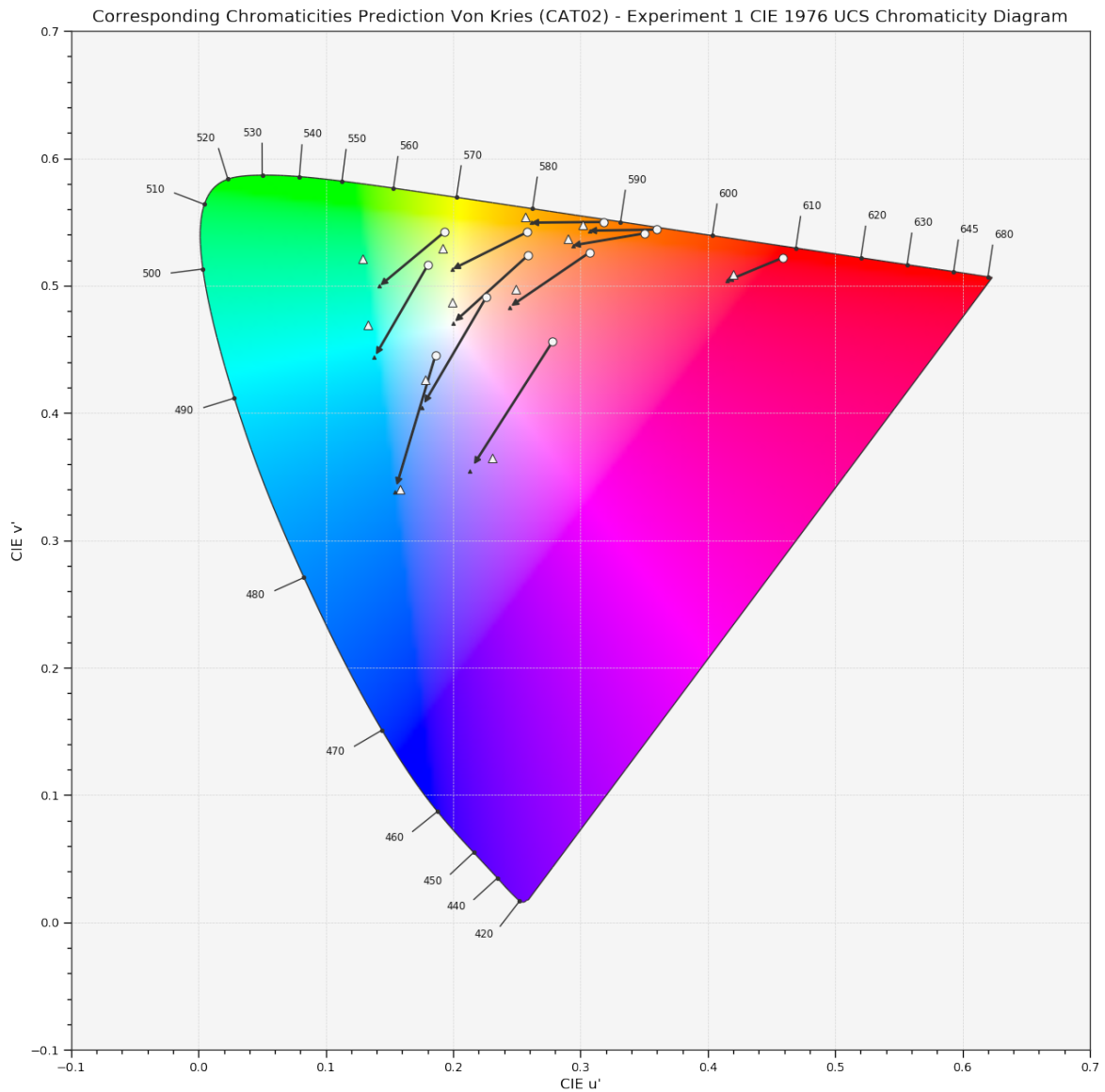
Other Parameters `**kwargs` (`dict`, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_corresponding_chromaticities_prediction(1, 'Von Kries', 'CAT02')
... # doctest: +SKIP
```



CIE Chromaticity Diagrams

colour.plotting

<code>plot_chromaticity_diagram_CIE1931([cmfs,...])</code>	Plots the <i>CIE 1931 Chromaticity Diagram</i> .
<code>plot_chromaticity_diagram_CIE1960UCS([cmfs,...])</code>	Plots the <i>CIE 1960 UCS Chromaticity Diagram</i> .

Continued on next page

Table 245 – continued from previous page

<code>plot_chromaticity_diagram_CIE1976UCS([cmfs, ...])</code>	Plots the <i>CIE 1976 UCS Chromaticity Diagram</i> .
<code>plot_sds_in_chromaticity_diagram_CIE1931(sds)</code>	Plots given spectral distribution chromaticity coordinates into the <i>CIE 1931 Chromaticity Diagram</i> .
<code>plot_sds_in_chromaticity_diagram_CIE1960UCS(sds)</code>	Plots given spectral distribution chromaticity coordinates into the <i>CIE 1960 UCS Chromaticity Diagram</i> .
<code>plot_sds_in_chromaticity_diagram_CIE1976UCS(sds)</code>	Plots given spectral distribution chromaticity coordinates into the <i>CIE 1976 UCS Chromaticity Diagram</i> .

colour.plotting.plot_chromaticity_diagram_CIE1931

`colour.plotting.plot_chromaticity_diagram_CIE1931(cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, show_spectral_locus=True, **kwargs)`

Plots the *CIE 1931 Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

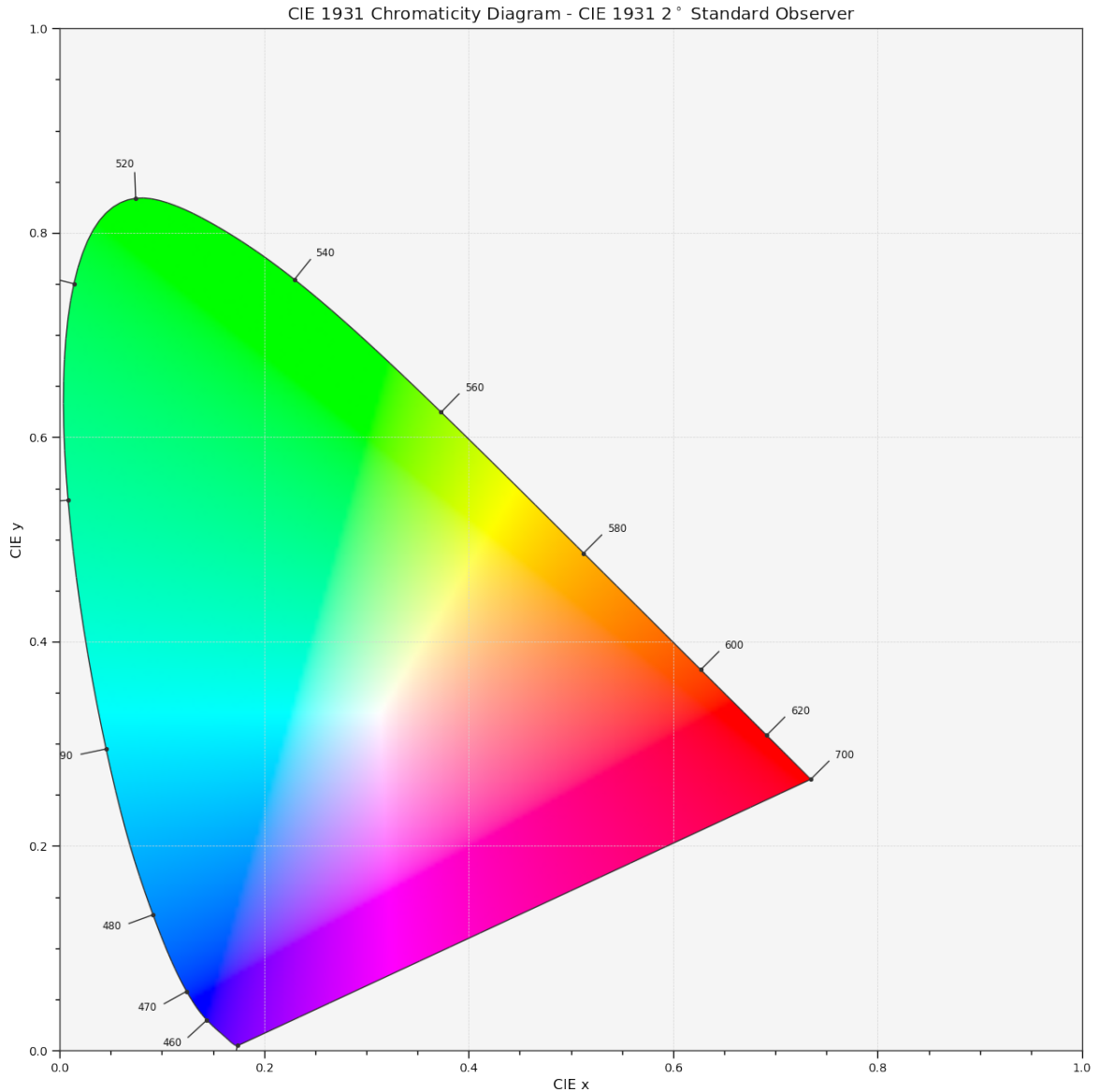
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1931() # doctest: +SKIP
```



`colour.plotting.plot_chromaticity_diagram_CIE1960UCS`

`colour.plotting.plot_chromaticity_diagram_CIE1960UCS(cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, show_spectral_locus=True, **kwargs)`

Plots the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

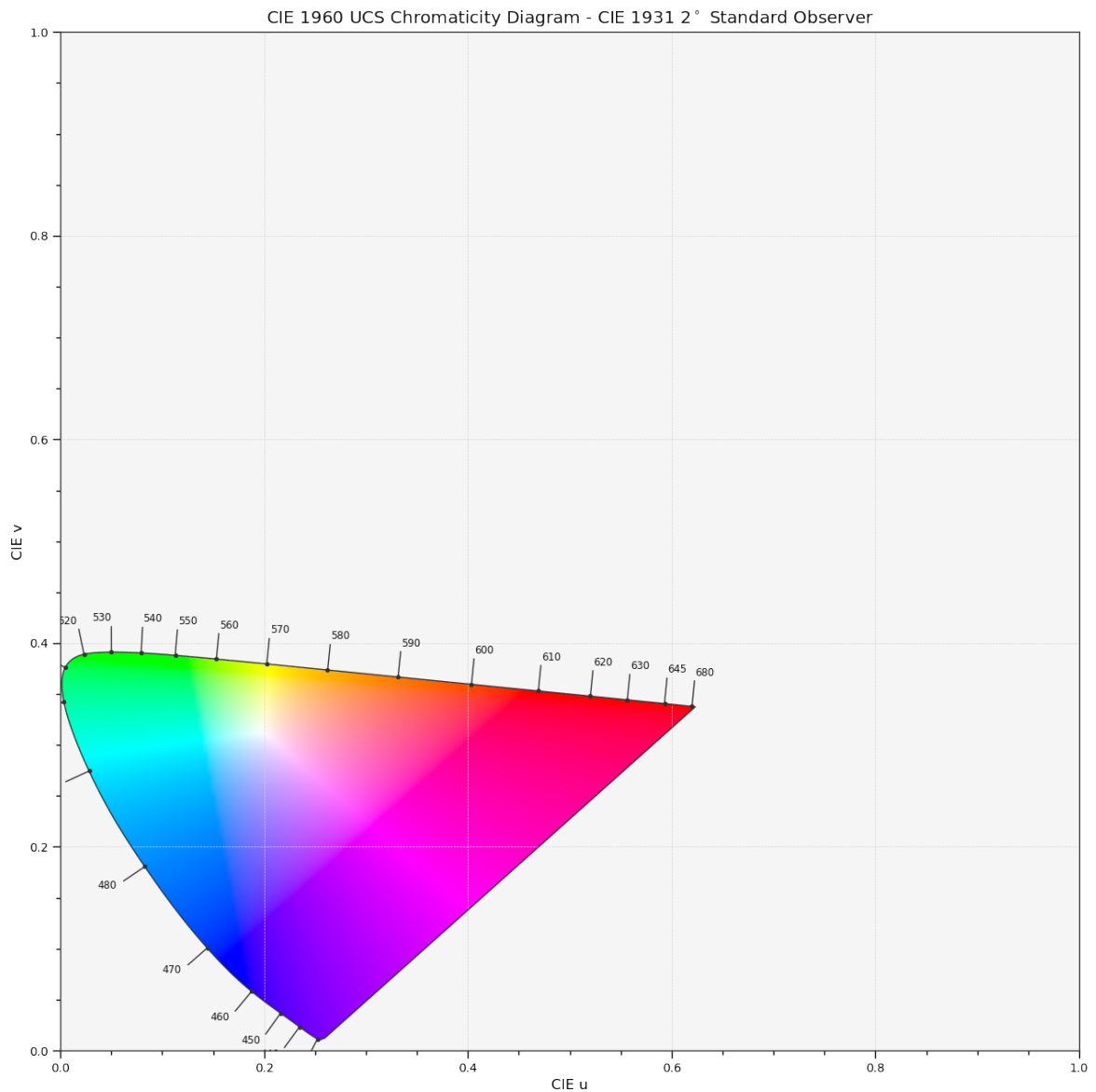
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`},
Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_chromaticity_diagram_CIE1960UCS() # doctest: +SKIP
```



colour.plotting.plot_chromaticity_diagram_CIE1976UCS

```
colour.plotting.plot_chromaticity_diagram_CIE1976UCS(cmfs='CIE 1931 2 Degree Standard Ob-
server', show_diagram_colours=True,
show_spectral_locus=True, **kwargs)
```

Plots the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

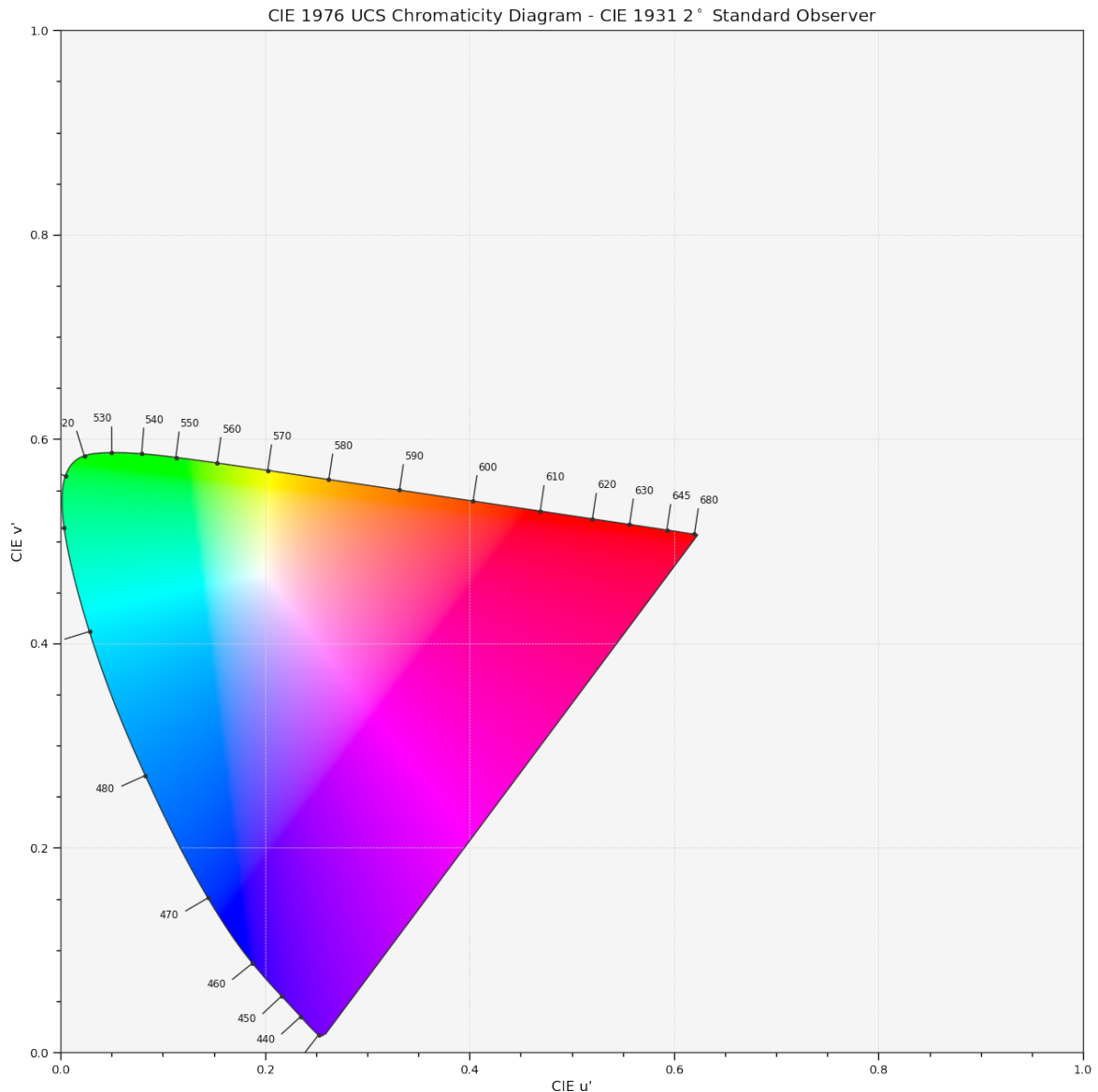
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1976UCS() # doctest: +SKIP
```



`colour.plotting.plot_sds_in_chromaticity_diagram_CIE1931`

```
colour.plotting.plot_sds_in_chromaticity_diagram_CIE1931(sds, cmfs='CIE 1931 2 Degree Standard Observer',
    annotate_parameters=None, chromaticity_diagram_callable_CIE1931=<function
    plot_chromaticity_diagram_CIE1931>, **kwargs)
```

Plots given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Parameters

- **sds** (array_like, optional) – Spectral distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **annotate_parameters** (`dict` or `array_like`, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to `True`. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

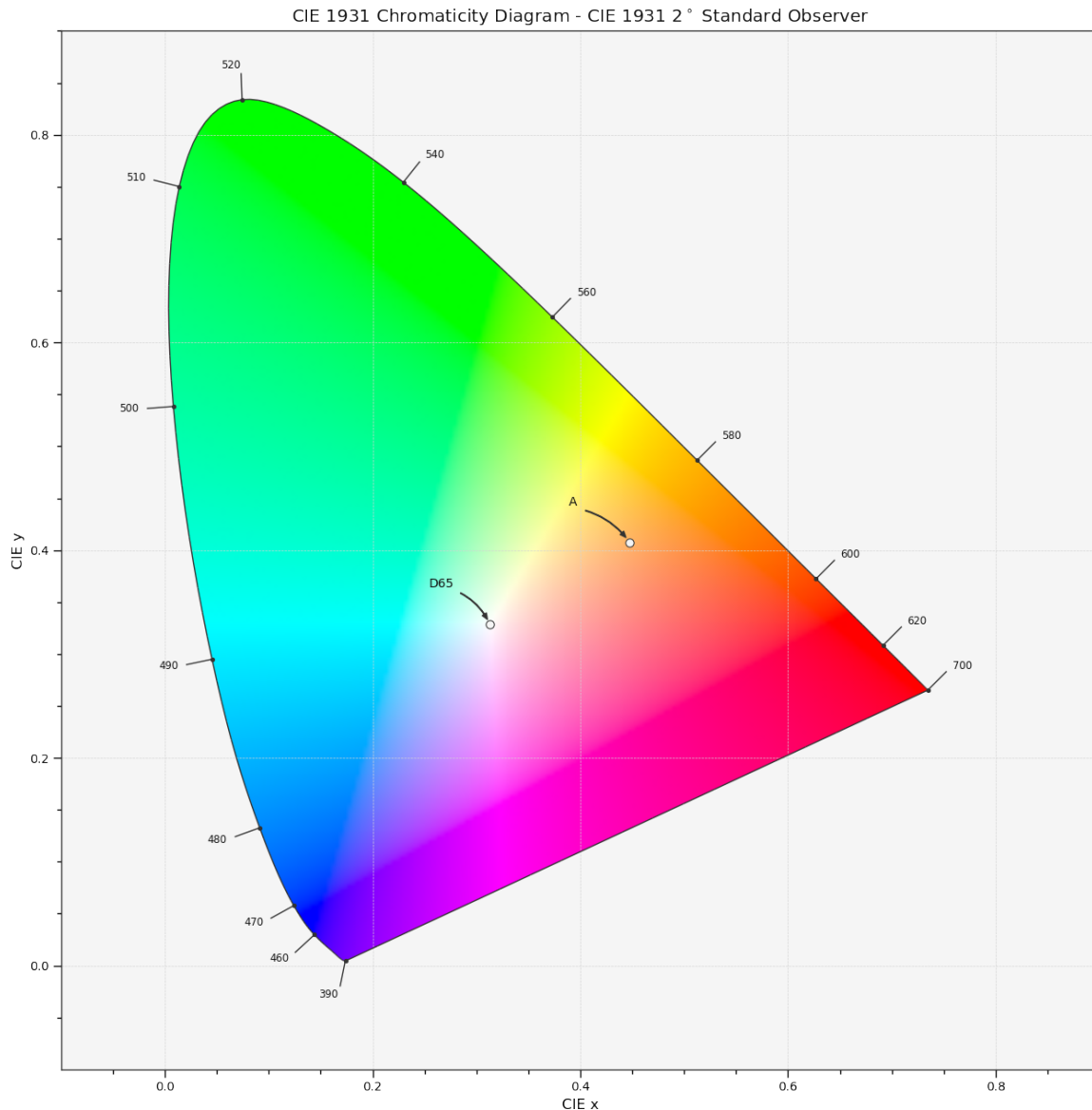
Other Parameters ****kwargs** (`dict`, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1931([A, D65]) # doctest: +SKIP
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1960UCS

```
colour.plotting.plot_sds_in_chromaticity_diagram_CIE1960UCS(sds, cmfs='CIE 1931 2 Degree Standard Observer',
    annotate_parameters=None,
    chromaticity_diagram_callable_CIE1960UCS=<function
    plot_chromaticity_diagram_CIE1960UCS>,
    **kwargs)
```

Plots given spectral distribution chromaticity coordinates into the CIE 1960 UCS Chromaticity Diagram.

Parameters

- **sds** (array_like, optional) – Spectral distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for

Chromaticity Diagram bounds.

- **annotate_parameters** (`dict` or `array_like`, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to `True`. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

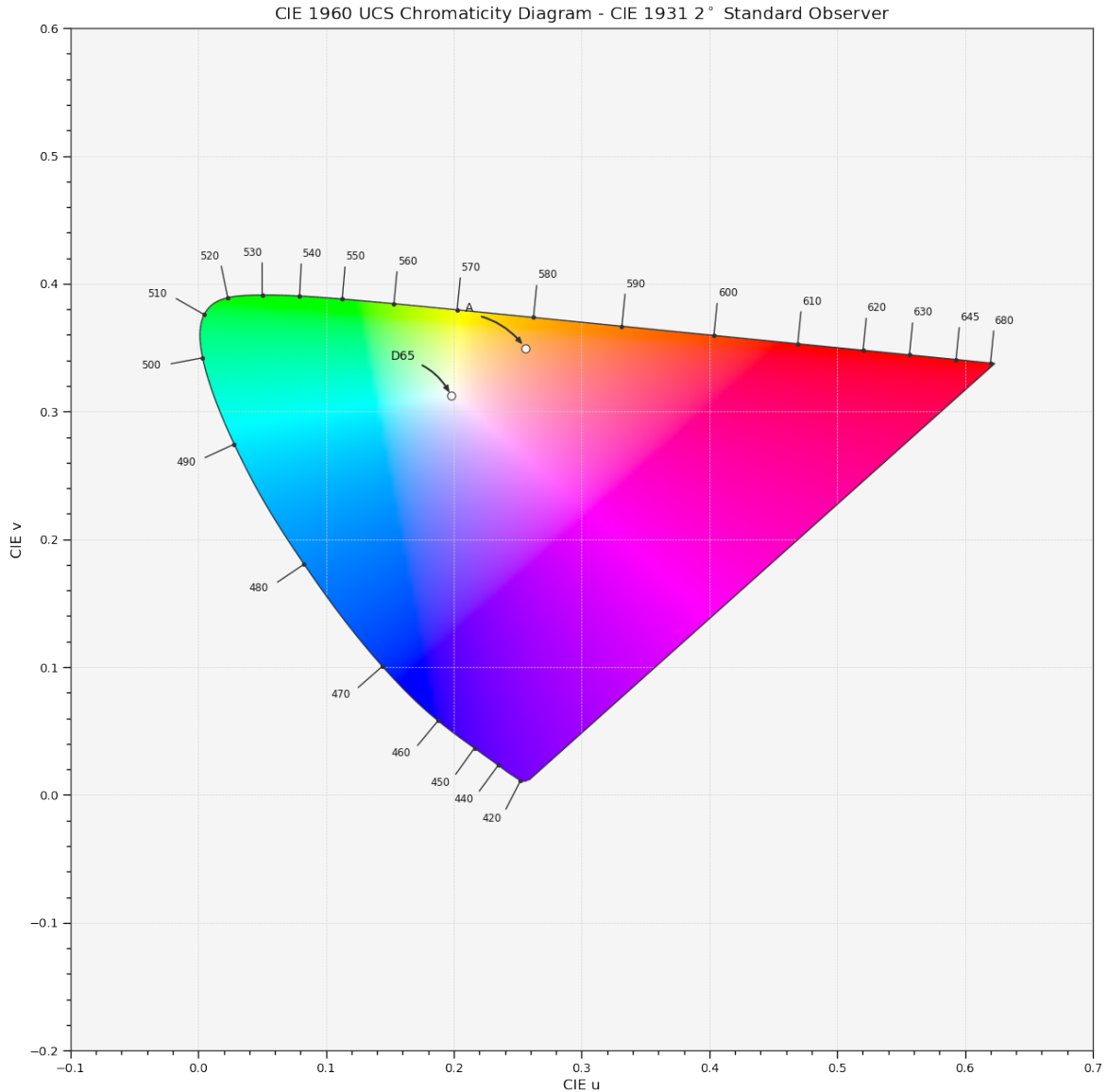
Other Parameters ****kwargs** (`dict`, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1960UCS([A, D65])
... # doctest: +SKIP
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1976UCS

```
colour.plotting.plot_sds_in_chromaticity_diagram_CIE1976UCS(sds, cmfs='CIE 1931 2 Degree Standard Observer',
    annotate_parameters=None,
    chromaticity_diagram_callable_CIE1976UCS=<function
    plot_chromaticity_diagram_CIE1976UCS>,
    **kwargs)
```

Plots given spectral distribution chromaticity coordinates into the CIE 1976 UCS Chromaticity Diagram.

Parameters

- **sds** (array_like, optional) – Spectral distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for

Chromaticity Diagram bounds.

- **annotate_parameters** (`dict` or `array_like`, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to `True`. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

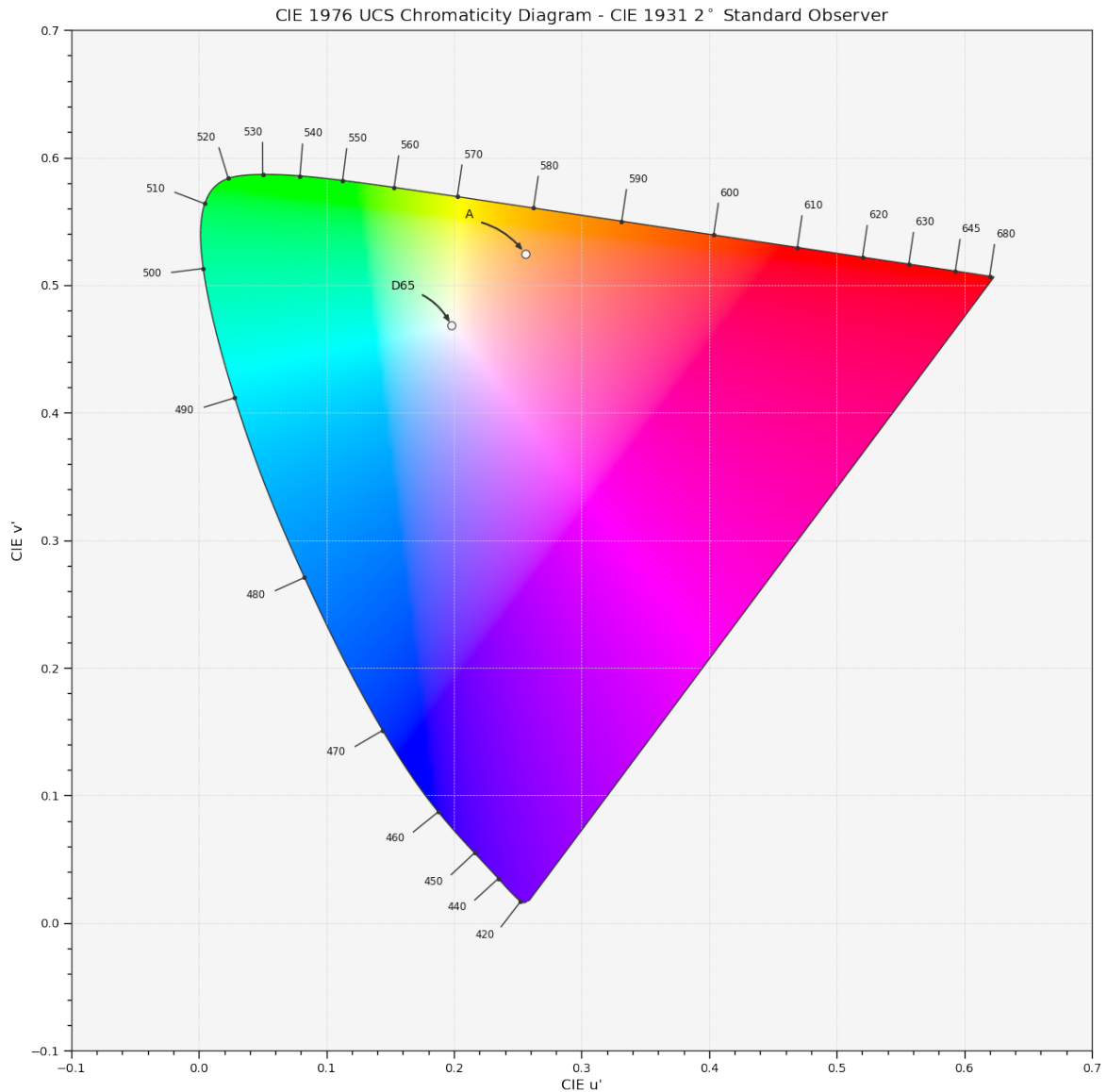
Other Parameters ****kwargs** (`dict`, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1976UCS([A, D65])
... # doctest: +SKIP
```



Ancillary Objects

`colour.plotting.diagrams`

<code>plot_spectral_locus([cmfs, ...])</code>	Plots the <i>Spectral Locus</i> according to given method.
<code>plot_chromaticity_diagram_colours([samples, ...])</code>	Plots the <i>Chromaticity Diagram</i> colours according to given method.
<code>plot_chromaticity_diagram([cmfs, ...])</code>	Plots the <i>Chromaticity Diagram</i> according to given method.
<code>plot_sds_in_chromaticity_diagram(sds[, ...])</code>	Plots given spectral distribution chromaticity coordinates into the <i>Chromaticity Diagram</i> using given method.

colour.plotting.diagrams.plot_spectral_locus

```
colour.plotting.diagrams.plot_spectral_locus(cmfs='CIE 1931 2 Degree Standard Ob-
server', spectral_locus_colours=None, spec-
tral_locus_labels=None, method='CIE 1931',
**kwargs)
```

Plots the *Spectral Locus* according to given method.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions defining the *Spectral Locus*.
- **spectral_locus_colours** (array_like or unicode, optional) – *Spectral Locus* colours, if `spectral_locus_colours` is set to *RGB*, the colours will be computed according to the corresponding chromaticity coordinates.
- **spectral_locus_labels** (array_like, optional) – Array of wavelength labels used to customise which labels will be drawn around the spectral locus. Passing an empty array will result in no wavelength labels being drawn.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

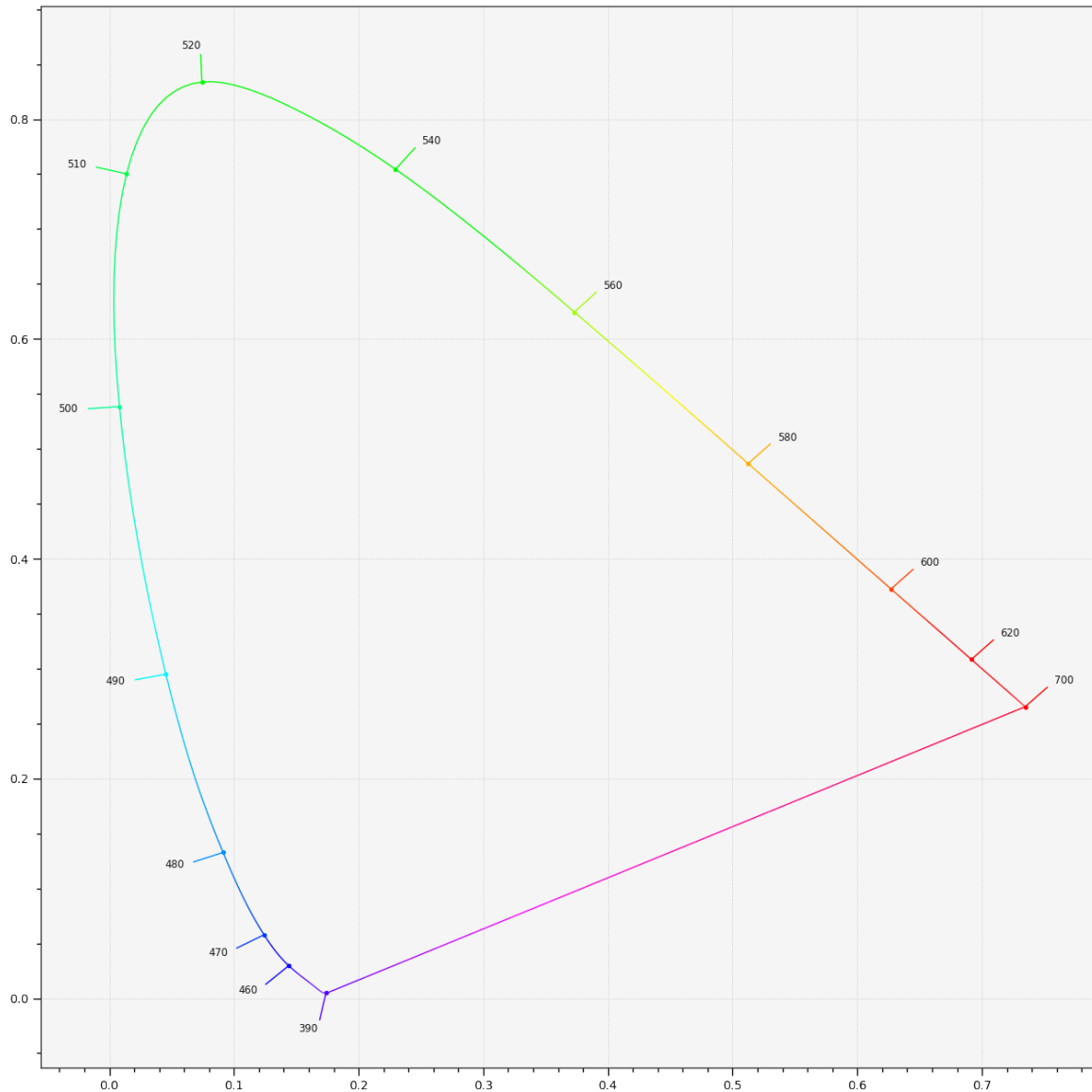
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_spectral_locus(spectral_locus_colours='RGB') # doctest: +SKIP
```



colour.plotting.diagrams.plot_chromaticity_diagram_colours

```
colour.plotting.diagrams.plot_chromaticity_diagram_colours(samples=256,          dia-
                                                             gram_opacity=1.0,      dia-
                                                             gram_clipping_path=None,
                                                             cmfs='CIE 1931 2 Degree Standard
Observer', method='CIE 1931',
                                                             **kwargs)
```

Plots the *Chromaticity Diagram* colours according to given method.

Parameters

- **samples** (numeric, optional) – Samples count on one axis.
- **diagram_opacity** (numeric, optional) – Opacity of the *Chromaticity Diagram* colours.

- **diagram_clipping_path** (array_like, optional) – Path of points used to clip the *Chromaticity Diagram* colours.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_chromaticity_diagram_colours() # doctest: +SKIP
```



`colour.plotting.diagrams.plot_chromaticity_diagram`

```
colour.plotting.diagrams.plot_chromaticity_diagram(cmfs='CIE 1931 2 Degree Standard Observer',  
                                                    show_diagram_colours=True,  
                                                    show_spectral_locus=True,    method='CIE 1931', **kwargs)
```

Plots the *Chromaticity Diagram* according to given method.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

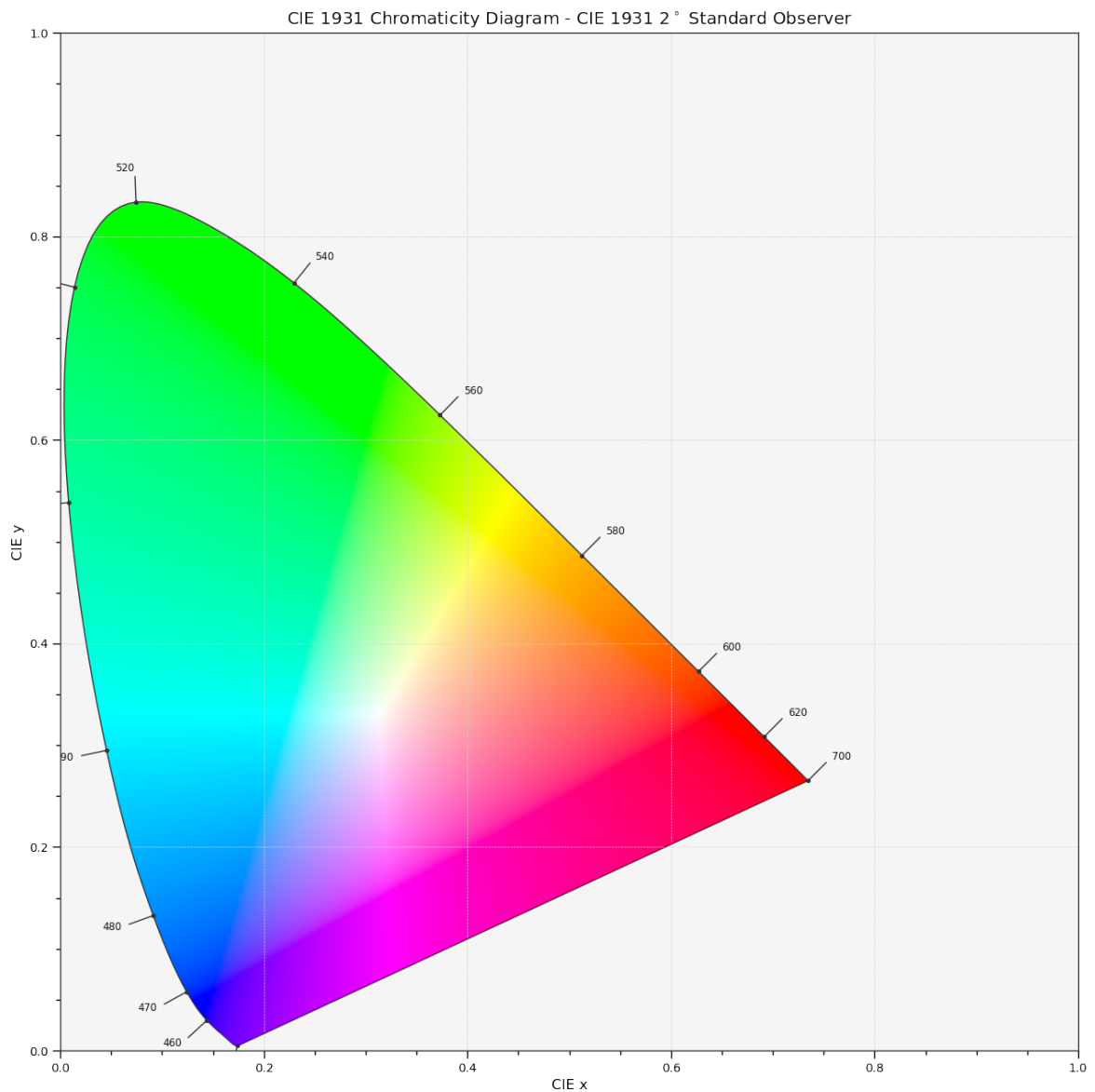
Other Parameters ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_spectral_locus(), colour.plotting.diagrams.plot_chromaticity_diagram_colours(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram() # doctest: +SKIP
```



colour.plotting.diagrams.plot_sds_in_chromaticity_diagram

```
colour.plotting.diagrams.plot_sds_in_chromaticity_diagram(sds, cmfs='CIE 1931 2 Degree Standard Observer', annotate_parameters=None, chromaticity_diagram_callable=<function plot_chromaticity_diagram>, method='CIE 1931', **kwargs)
```

Plots given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

Parameters

- **sds** (array_like, optional) – Spectral distributions to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to `True`. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

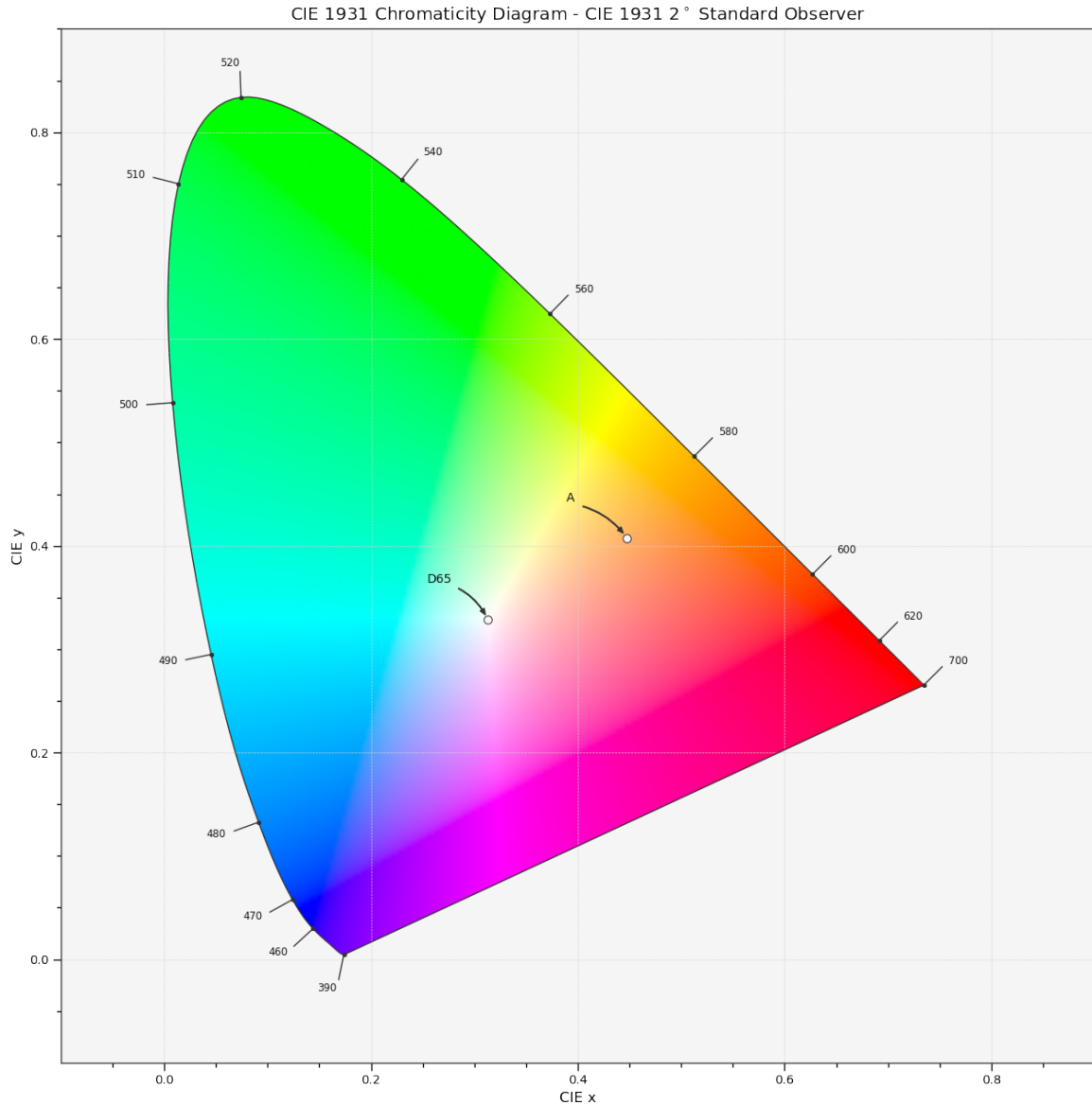
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram([A, D65]) # doctest: +SKIP
```



Colour Models

`colour.plotting`

`plot_RGB_colourspace_in_chromaticity_diagram_CIE1931` Plots given RGB colourspace in the CIE 1931 Chromaticity Diagram.

`plot_RGB_colourspace_in_chromaticity_diagram_CIE1960` Plots given RGB colourspace in the CIE 1960 UCS Chromaticity Diagram.

`plot_RGB_colourspace_in_chromaticity_diagram_CIE1976` Plots given RGB colourspace in the CIE 1976 UCS Chromaticity Diagram.

`plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931` Plots given RGB colourspace array in the CIE 1931 Chromaticity Diagram.

Continued on next page

Table 247 – continued from previous page

<code>plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931</code>	Plots given <i>RGB</i> colourspace array in the <i>CIE 1960 UCS Chromaticity Diagram</i> .
<code>plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976</code>	Plots given <i>RGB</i> colourspace array in the <i>CIE 1976 UCS Chromaticity Diagram</i> .
<code>plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931</code>	Plots <i>MacAdam (1942) Ellipses (Observer PGN)</i> in the <i>CIE 1931 Chromaticity Diagram</i> .
<code>plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960</code>	Plots <i>MacAdam (1942) Ellipses (Observer PGN)</i> in the <i>CIE 1960 UCS Chromaticity Diagram</i> .
<code>plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976</code>	Plots <i>MacAdam (1942) Ellipses (Observer PGN)</i> in the <i>CIE 1976 UCS Chromaticity Diagram</i> .
<code>plot_single_cctf([cctf, decoding_cctf])</code>	Plots given colourspace colour component transfer function.
<code>plot_multi_cctfs([cctfs, decoding_cctf])</code>	Plots given colour component transfer functions.

`colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931`

`colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931` (*colourspaces*=None, *cmfs*='CIE 1931 2 Degree Standard Observer', *chromaticity_diagram_callable_CIE1931*=<function *plot_chromaticity_diagram_CIE1931*>, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters ***kwargs* (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
... # doctest: +SKIP
```



`colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS`

```
colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS(colourspaces=None,
                                                                           cmfs='CIE 1931
                                                                           2 Degree Stan-
                                                                           dard Observer',
                                                                           chromatic-
                                                                           ity_diagram_callable_CIE1960UCS=<fun
                                                                           ction>,
                                                                           plot_chromaticity_diagram_CIE1960UCS=
                                                                           **kwargs)
```

Plots given *RGB* colourspace in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

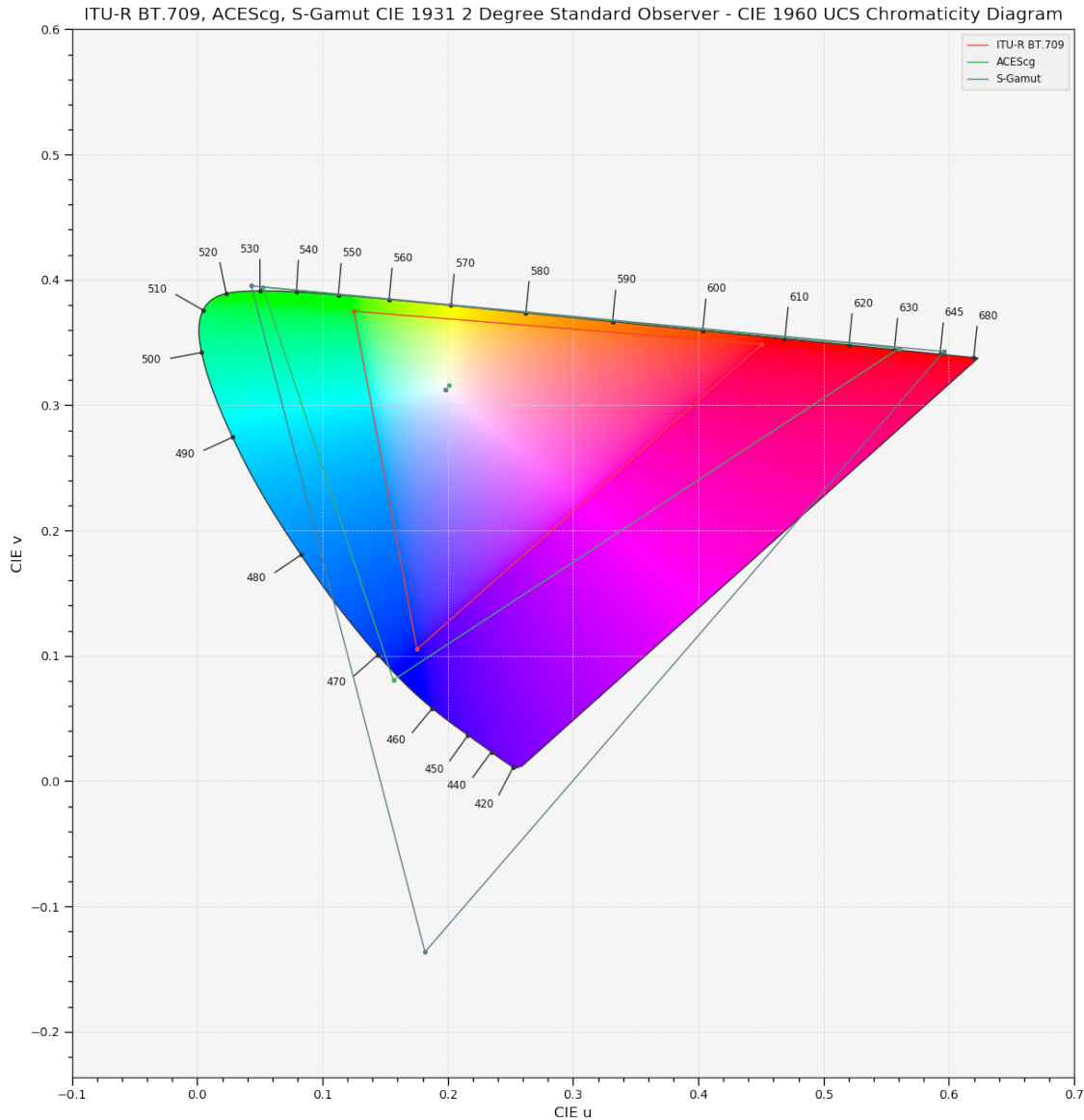
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_RGB_colourspace_in_chromaticity_diagram_CIE1960UCS((
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
... # doctest: +SKIP
```

`colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS`

```
colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS(colourspaces=None,
                                                                           cmfs='CIE 1931
                                                                           2 Degree Stan-
                                                                           dard Observer',
                                                                           chromatic-
                                                                           ity_diagram_callable_CIE1976UCS=<fun
                                                                           ction>,
                                                                           plot_chromaticity_diagram_CIE1976UCS:
                                                                           **kwargs)
```

Plots given *RGB* colourspace in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

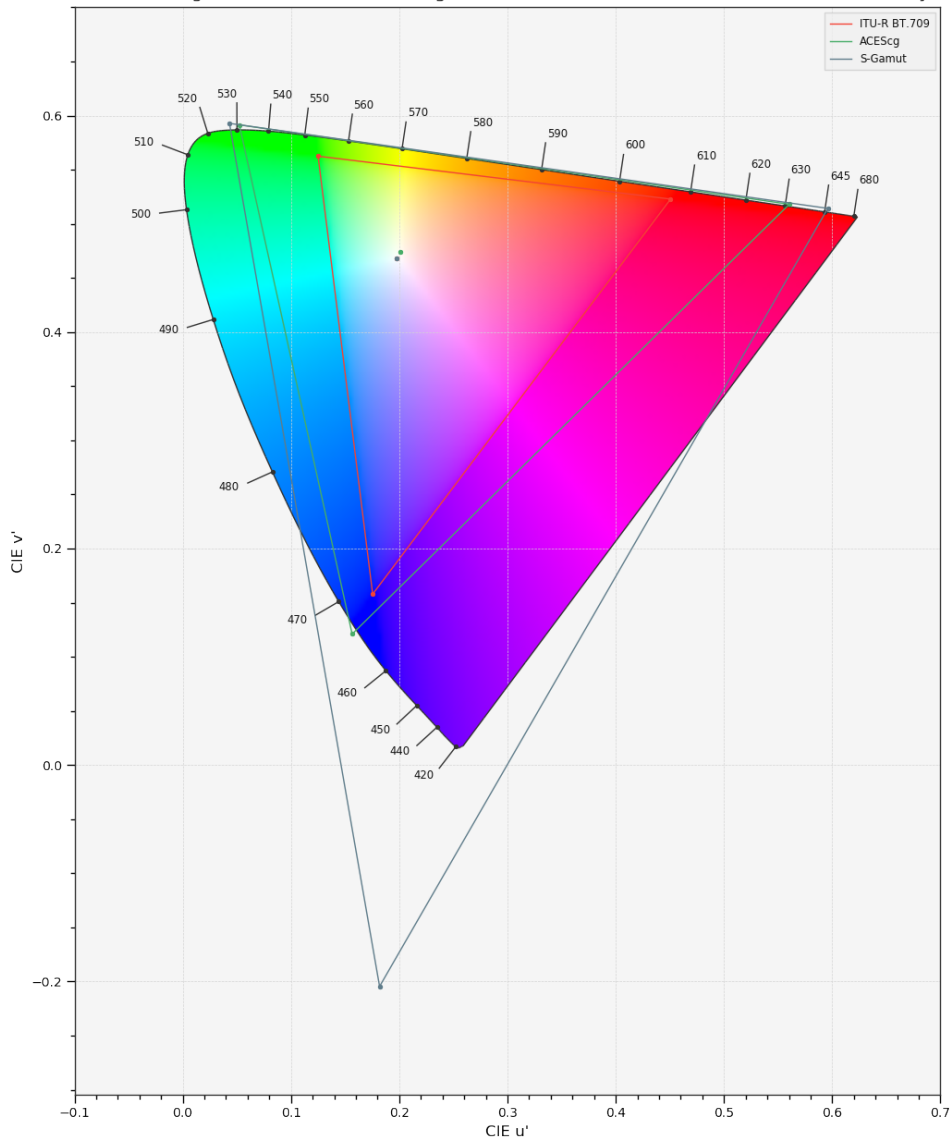
Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_RGB_colourspace_in_chromaticity_diagram_CIE1976UCS((
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
... # doctest: +SKIP
```

ITU-R BT.709, ACEScg, S-Gamut CIE 1931 2 Degree Standard Observer - CIE 1976 UCS Chromaticity Diagram



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931

```
colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
    RGB,
    colourspace='sRGB',
    chromaticity_diagram_callable_CIE1931=<function
    plot_RGB_colourspace_in_chromaticity_diagram_CIE1931>,
    scatter_parameters=None,
    **kwargs)
```

Plots given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.

Parameters

- *RGB* (array_like) – *RGB* colourspace array.

- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the `plt.scatter()` definition, if `c` is set to *RGB*, the scatter will use given *RGB* colours.

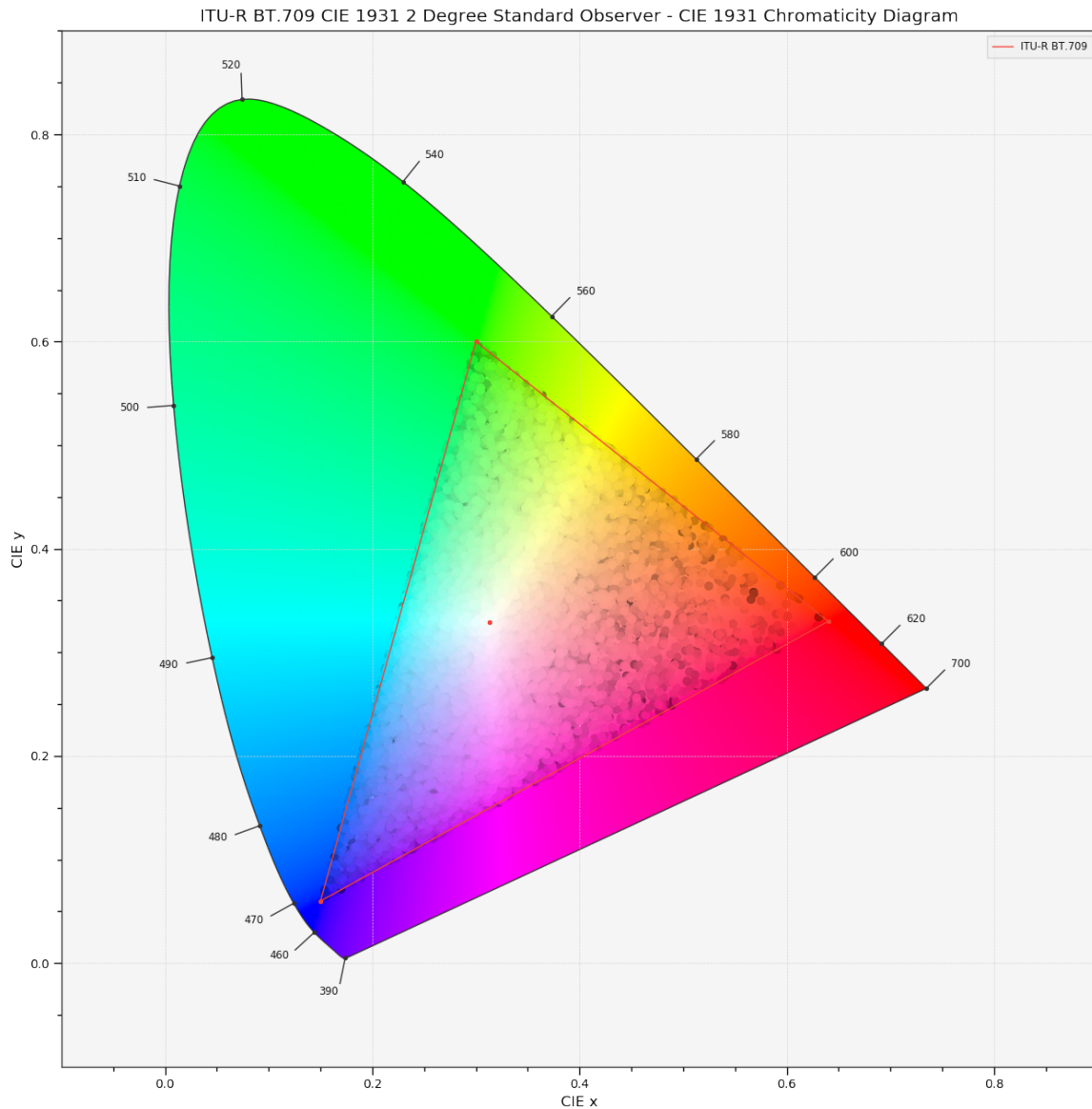
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.diagrams.plot_RGB_colourspace_in_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB, 'ITU-R BT.709')
... # doctest: +SKIP
```



`colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS`

```
colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(RGB,
    colourspace='sRGB',
    chromaticity_diagram_callable_CIE1960UCS=<j
    plot_RGB_colourspace_in_chromaticity
    scatter_parameters=None,
    **kwargs)
```

Plots given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- *RGB* (*array_like*) – *RGB* colourspace array.

- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the `plt.scatter()` definition, if `c` is set to *RGB*, the scatter will use given *RGB* colours.

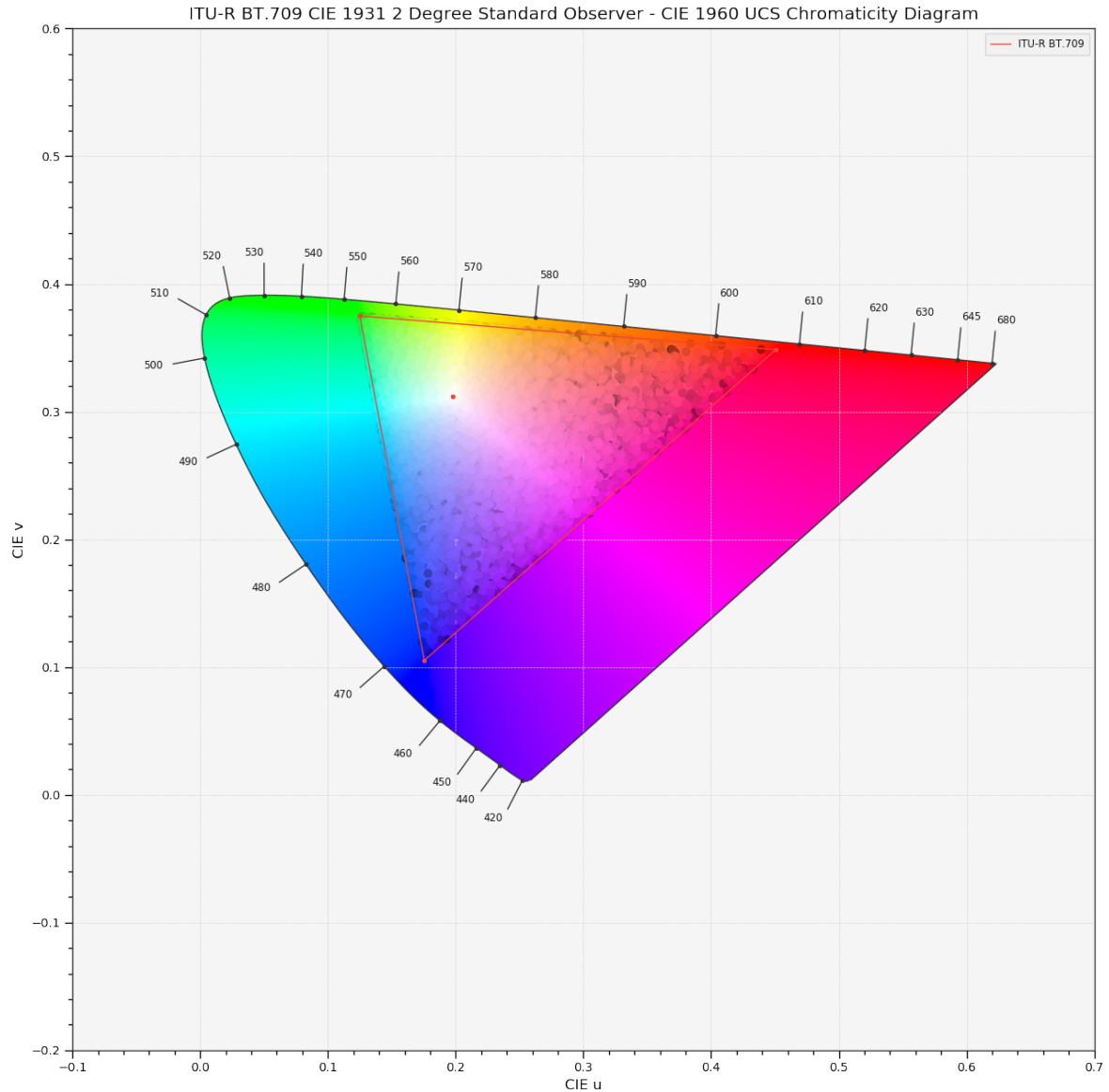
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.diagrams.plot_RGB_colourspace_in_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(
...     RGB, 'ITU-R BT.709')
... # doctest: +SKIP
```



`colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS`

```
colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(RGB,
    colourspace='sRGB',
    chromaticity_diagram_callable_CIE1976UCS=<j
    plot_RGB_colourspace_in_chromaticity
    scatter_parameters=None,
    **kwargs)
```

Plots given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- *RGB* (*array_like*) – *RGB* colourspace array.

- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the `plt.scatter()` definition, if `c` is set to *RGB*, the scatter will use given *RGB* colours.

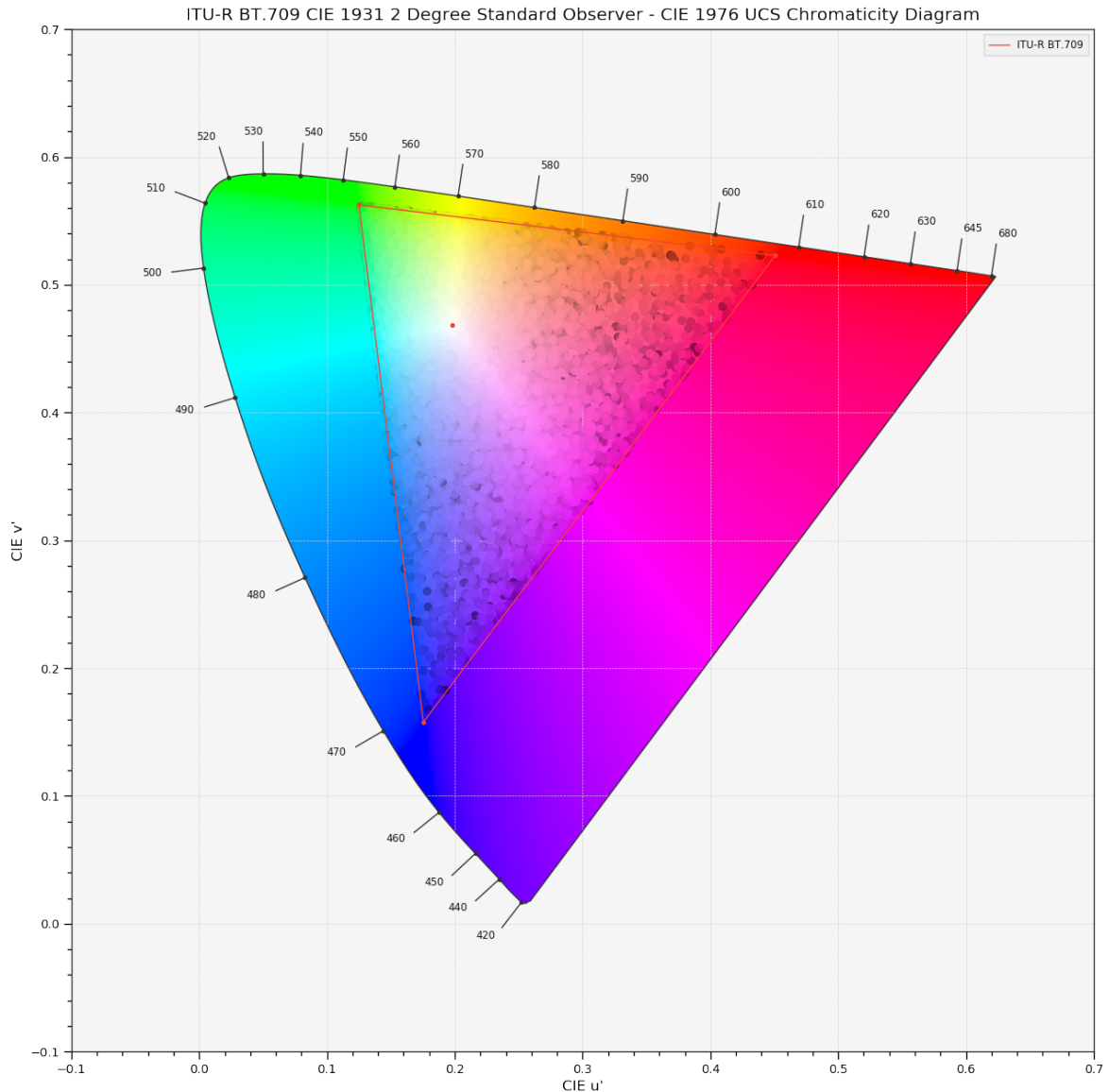
Other Parameters ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.diagrams.plot_RGB_colourspace_in_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(
...     RGB, 'ITU-R BT.709')
... # doctest: +SKIP
```

`colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931`

`colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931`(*chromaticity_diagram_callable_CIE1931*
plot_chromaticity_diagram_CIE1931>,
chromaticity_diagram_clipping=False,
ellipse_parameters=None,
***kwargs*)

Plots MacAdam (1942) Ellipses (Observer PGN) in the CIE 1931 Chromaticity Diagram.

Parameters

- **`chromaticity_diagram_callable_CIE1931`** (callable, optional) – Callable responsible for drawing the CIE 1931 Chromaticity Diagram.

- **chromaticity_diagram_clipping** (`bool`, optional,) – Whether to clip the *CIE 1931 Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (`dict` or `array_like`, optional) – Parameters for the `Ellipse` class, `ellipse_parameters` can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

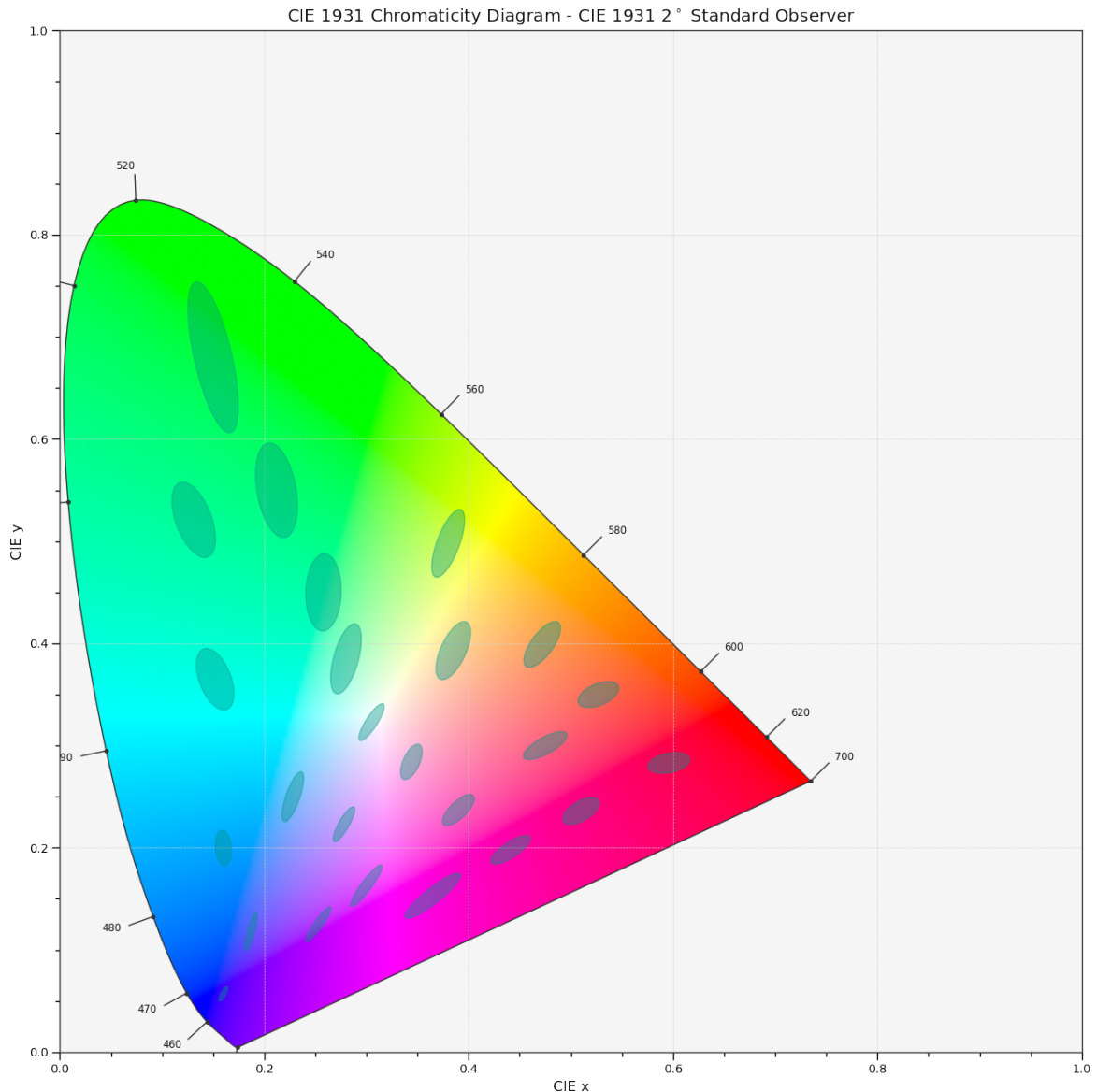
Other Parameters ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()`}, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931()
... # doctest: +SKIP
```



`colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS`

```
colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS(chromaticity_diagram_callable_CIE1960UCS,
                                                                              plot_chromaticity_diagram_CIE1960UCS=True,
                                                                              chromaticity_diagram_clipping=False,
                                                                              ellipse_parameters=None,
                                                                              **kwargs)
```

Plots MacAdam (1942) Ellipses (Observer PGN) in the CIE 1960 UCS Chromaticity Diagram.

Parameters

- **`chromaticity_diagram_callable_CIE1960UCS`** (callable, optional) – Callable responsible for drawing the CIE 1960 UCS Chromaticity Diagram.

- **chromaticity_diagram_clipping** (`bool`, optional,) – Whether to clip the *CIE 1960 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (`dict` or `array_like`, optional) – Parameters for the `Ellipse` class, `ellipse_parameters` can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

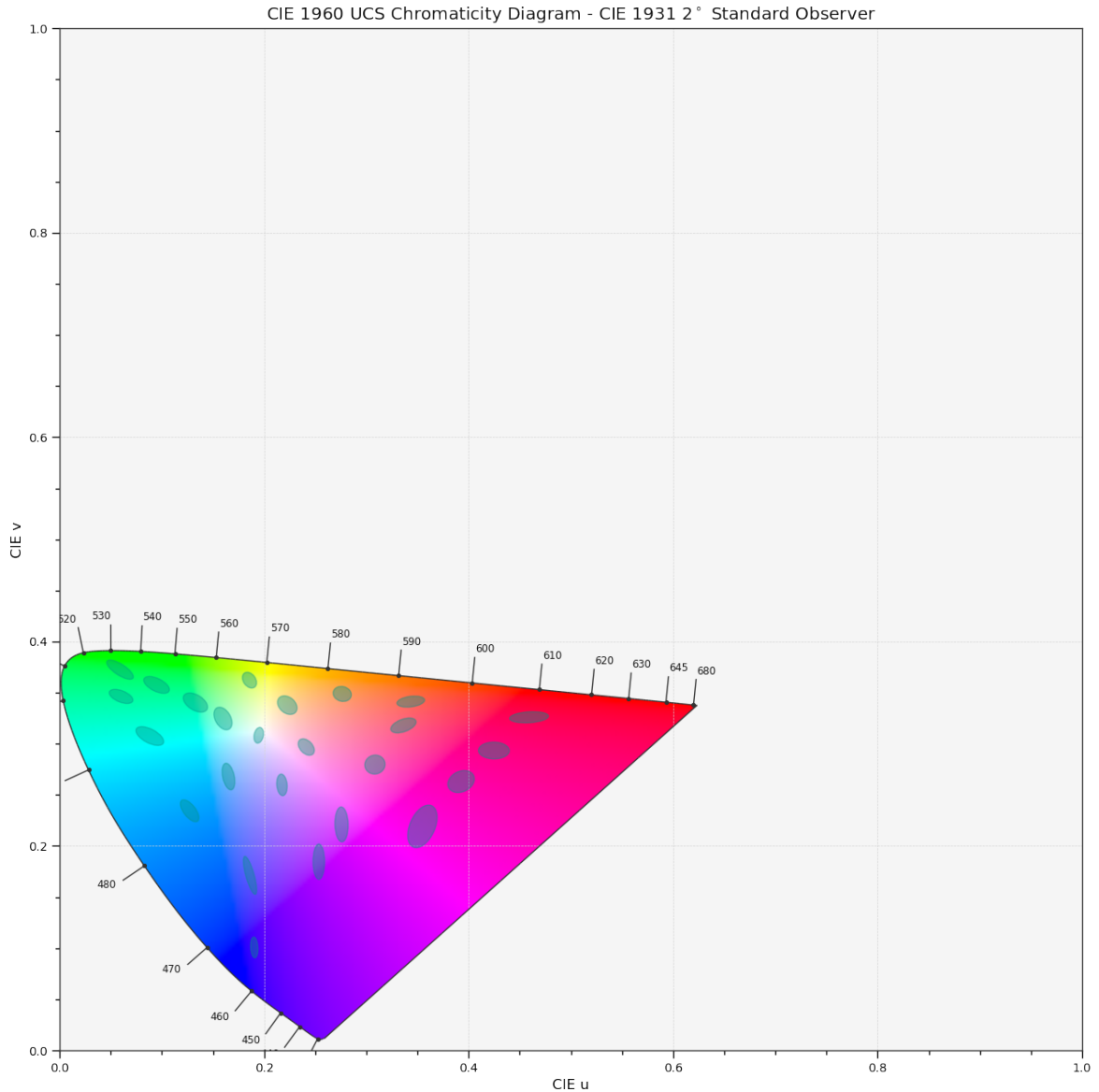
Other Parameters ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()`}, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS()  
... # doctest: +SKIP
```



`colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS`

`colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS(chromaticity_diagram_callable_CIE1976UCS, plot_chromaticity_diagram_CIE1976UCS, chromaticity_diagram_clipping=False, ellipse_parameters=None, **kwargs)`

Plots MacAdam (1942) Ellipses (Observer PGN) in the CIE 1976 UCS Chromaticity Diagram.

Parameters

- **`chromaticity_diagram_callable_CIE1976UCS`** (callable, optional) – Callable responsible for drawing the CIE 1976 UCS Chromaticity Diagram.

- **chromaticity_diagram_clipping** (`bool`, optional,) – Whether to clip the *CIE 1976 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (`dict` or `array_like`, optional) – Parameters for the `Ellipse` class, `ellipse_parameters` can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

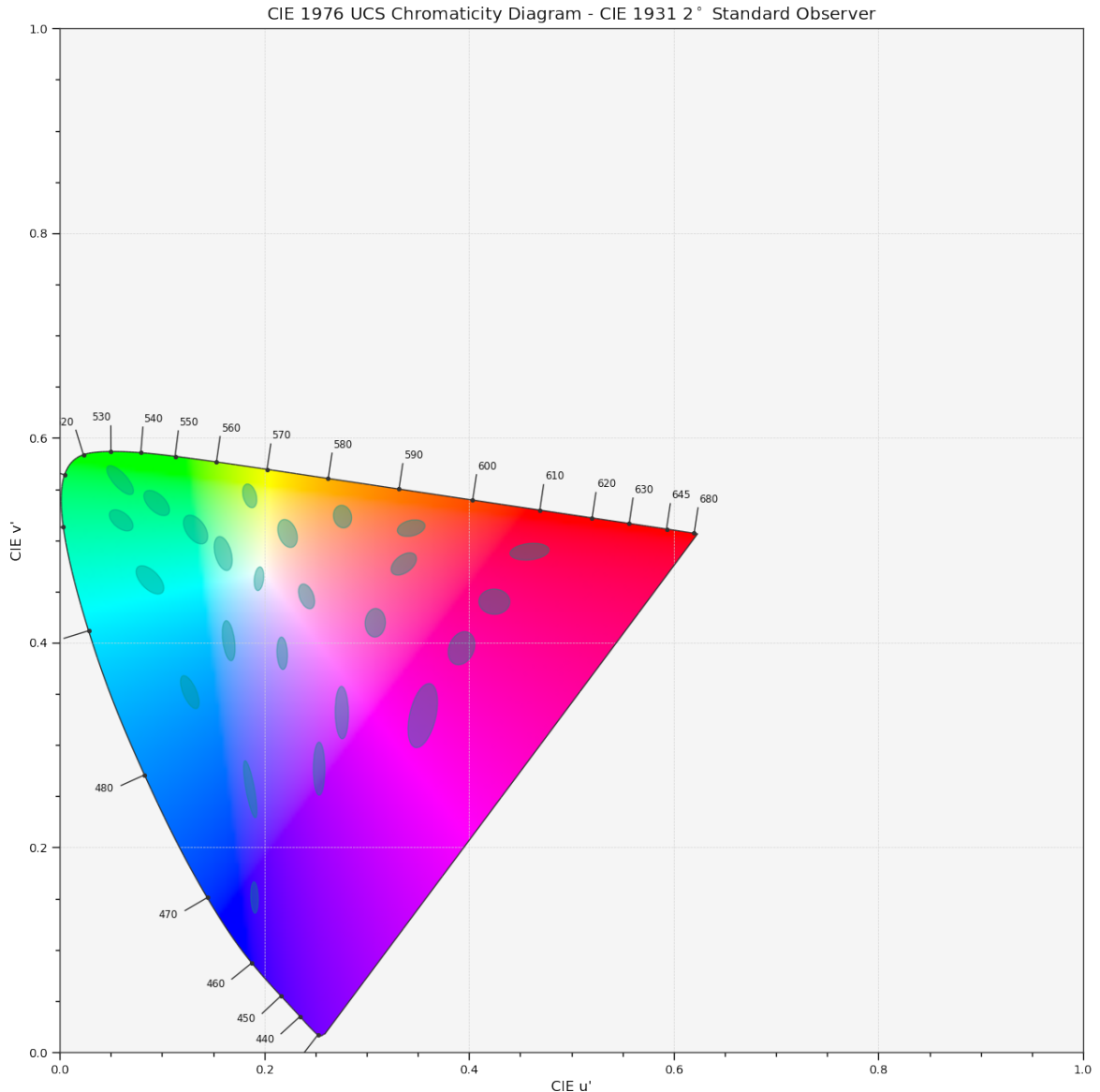
Other Parameters ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()`}, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS()  
... # doctest: +SKIP
```



`colour.plotting.plot_single_cctf`

`colour.plotting.plot_single_cctf(cctf='ITU-R BT.709', decoding_cctf=False, **kwargs)`
 Plots given colourspace colour component transfer function.

Parameters

- **cctf** (unicode, optional) – Colour component transfer function to plot.
- **decoding_cctf** (bool) – Plot the decoding colour component transfer function instead.

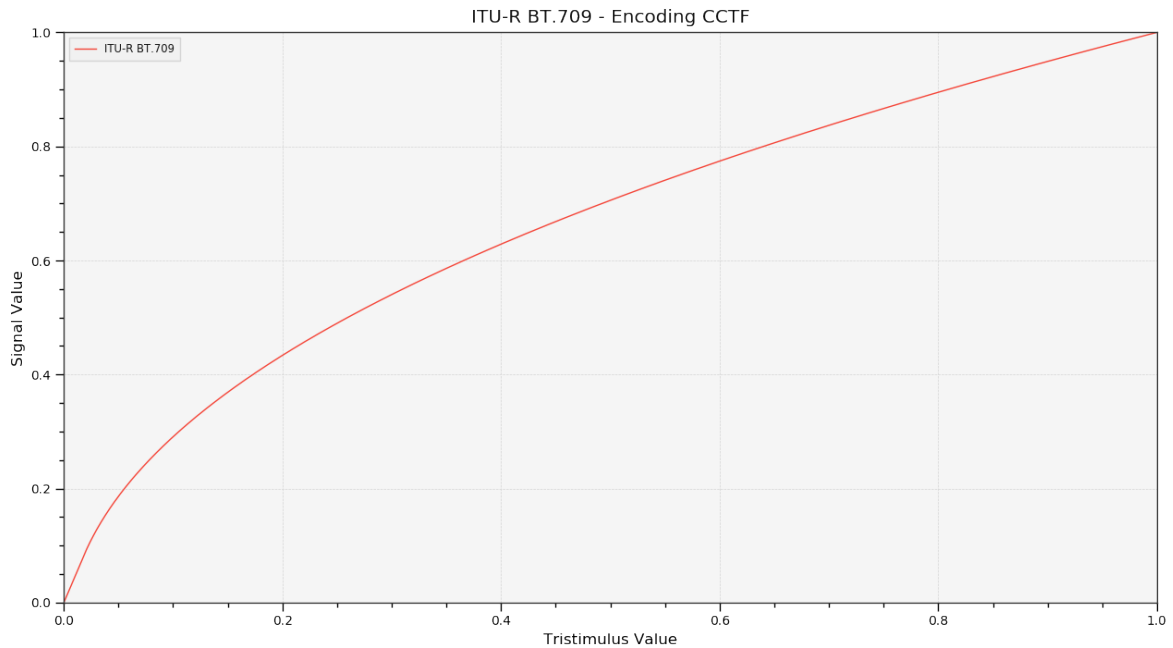
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_cctf('ITU-R BT.709') # doctest: +SKIP
```



`colour.plotting.plot_multi_cctfs`

`colour.plotting.plot_multi_cctfs(cctfs=None, decoding_cctf=False, **kwargs)`

Plots given colour component transfer functions.

Parameters

- **cctfs** (`array_like`, optional) – Colour component transfer function to plot.
- **decoding_cctf** (`bool`) – Plot the decoding colour component transfer function instead.

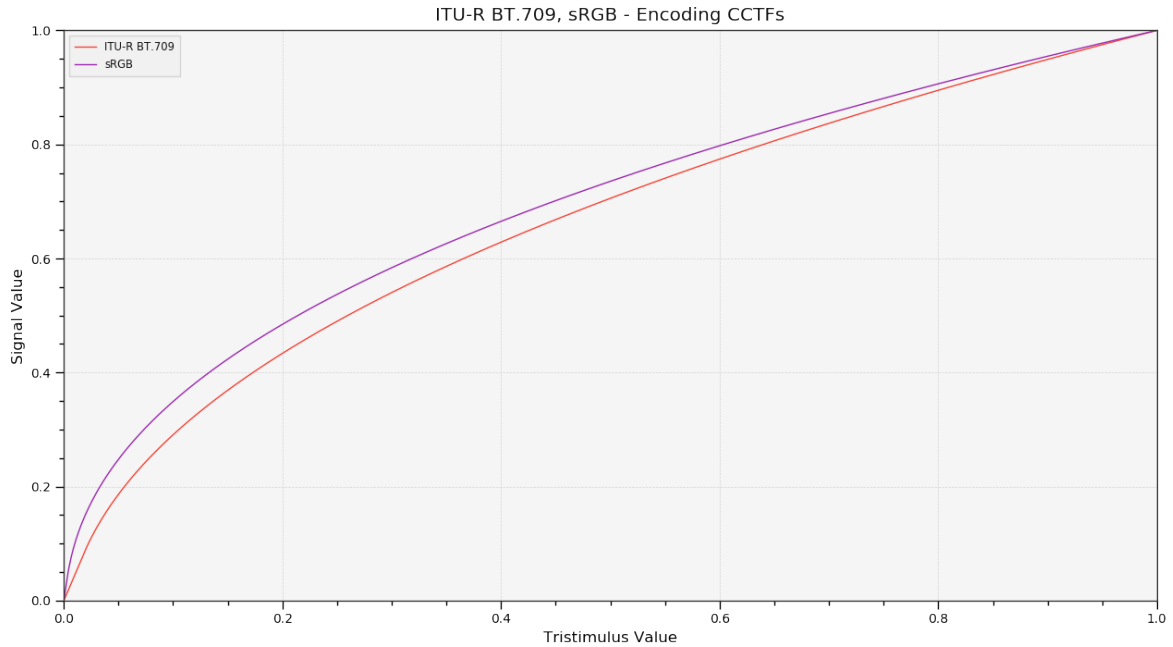
Other Parameters `**kwargs` (`dict`, optional) – `{colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}`, Please refer to the documentation of the previously listed definitions.`

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_multi_cctfs(['ITU-R BT.709', 'sRGB']) # doctest: +SKIP
```

Ancillary Objects

`colour.plotting.models`

<code>plot_pointer_gamut([method])</code>	Plots <i>Pointer's Gamut</i> according to given method.
<code>plot_RGB_colourspace_in_chromaticity_diagram([Plot])</code>	Plots given <i>RGB</i> colourspaces in the <i>Chromaticity Diagram</i> according to given method.
<code>plot_RGB_chromaticities_in_chromaticity_diagram([RGB])</code>	Plots given <i>RGB</i> colourspace array in the <i>Chromaticity Diagram</i> according to given method.

`colour.plotting.models.plot_pointer_gamut`

`colour.plotting.models.plot_pointer_gamut(method='CIE 1931', **kwargs)`

Plots *Pointer's Gamut* according to given method.

Parameters `method` (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, Plotting method.

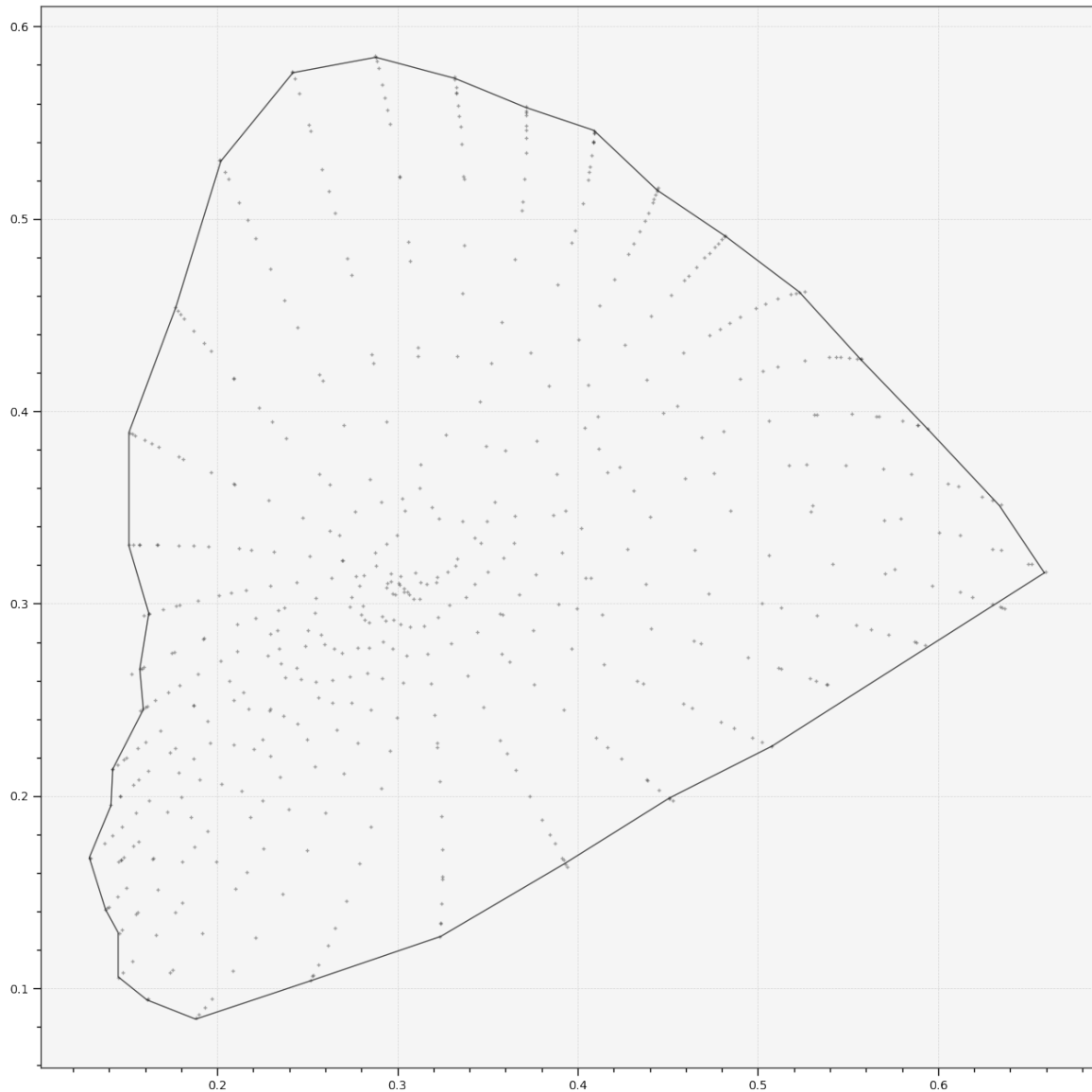
Other Parameters `**kwargs` (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_pointer_gamut() # doctest: +SKIP
```



`colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram`

```
colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram(colourspace=None,
                                                                    cmfs='CIE 1931 2
                                                                    Degree Standard
                                                                    Observer', chromatic-
                                                                    ity_diagram_callable=<function
                                                                    plot_chromaticity_diagram>,
                                                                    method='CIE 1931',
                                                                    show_whitepoints=True,
                                                                    show_pointer_gamut=False,
                                                                    **kwargs)
```

Plots given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.

Parameters

- **colourspace**s (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.
- **show_whitepoints** (bool, optional) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool, optional) – Whether to display the *Pointer's Gamut*.

Other Parameters ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.plot_pointer_gamut(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspace_in_chromaticity_diagram(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
... # doctest: +SKIP
```



colour.plotting.models.plot_RGB_chromaticities_in_chromaticity_diagram

```
colour.plotting.models.plot_RGB_chromaticities_in_chromaticity_diagram(
    RGB,
    colourspace='sRGB',
    chromaticity_diagram_callable=<function
    plot_RGB_colourspace_in_chromaticity_diagram>,
    method='CIE 1931',
    scatter_parameters=None,
    **kwargs)
```

Plots given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.

- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.
- **scatter_parameters** (dict, optional) – Parameters for the `plt.scatter()` definition, if `c` is set to *RGB*, the scatter will use given *RGB* colours.

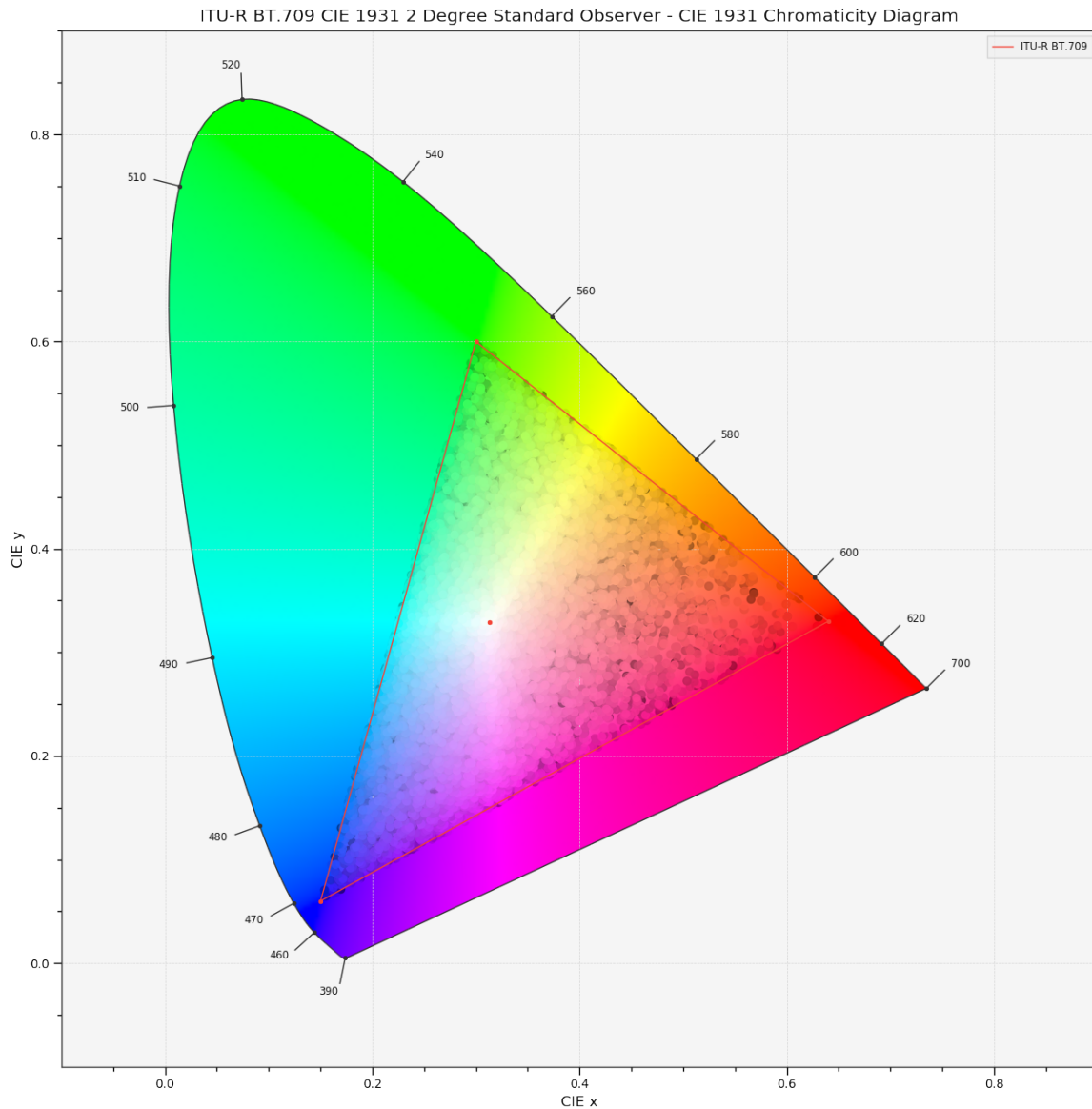
Other Parameters ****kwargs** (*dict*, *optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.diagrams.plot_RGB_colourspaces_in_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram(
...     RGB, 'ITU-R BT.709')
... # doctest: +SKIP
```



Colour Notation Systems

`colour.plotting`

`plot_single_munsell_value_function([function])` Plots given *Lightness* function.

`plot_multi_munsell_value_functions([functions])` Plots given *Munsell* value functions.

`colour.plotting.plot_single_munsell_value_function`

`colour.plotting.plot_single_munsell_value_function(function='ASTM D1535-08', **kwargs)`

Plots given *Lightness* function.

Parameters `function` (unicode, optional) – *Munsell* value function to plot.

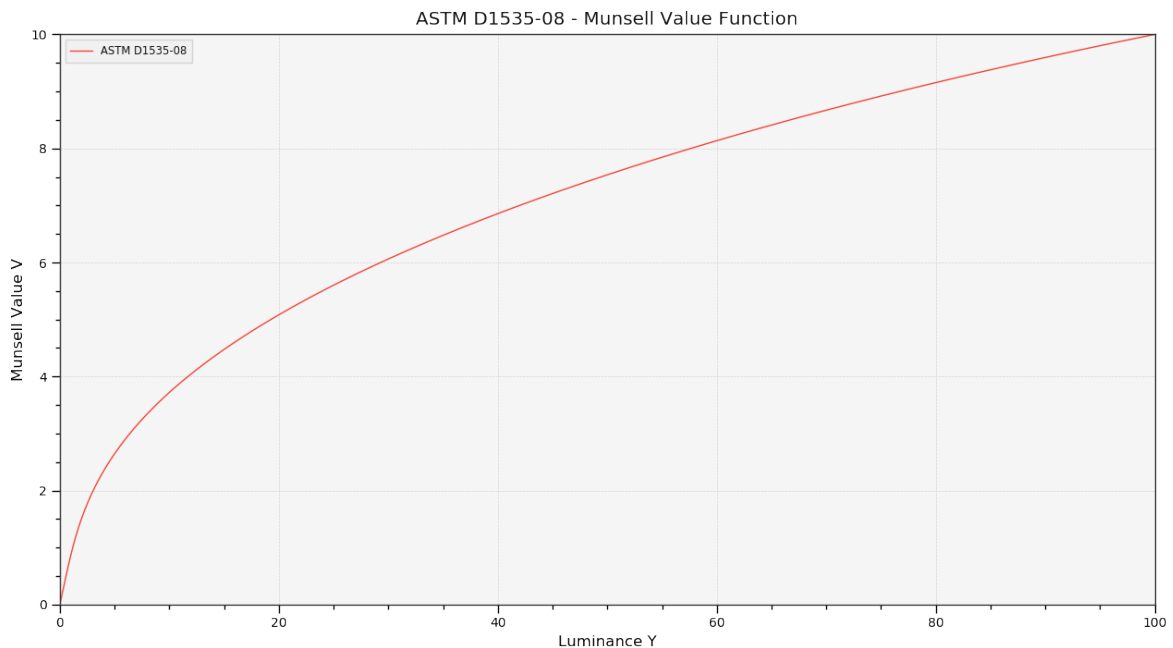
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_munsell_value_function('ASTM D1535-08') # doctest: +SKIP
```



`colour.plotting.plot_multi_munsell_value_functions`

`colour.plotting.plot_multi_munsell_value_functions`(*functions=None, **kwargs*)

Plots given *Munsell* value functions.

Parameters **functions** (*array_like, optional*) – *Munsell* value functions to plot.

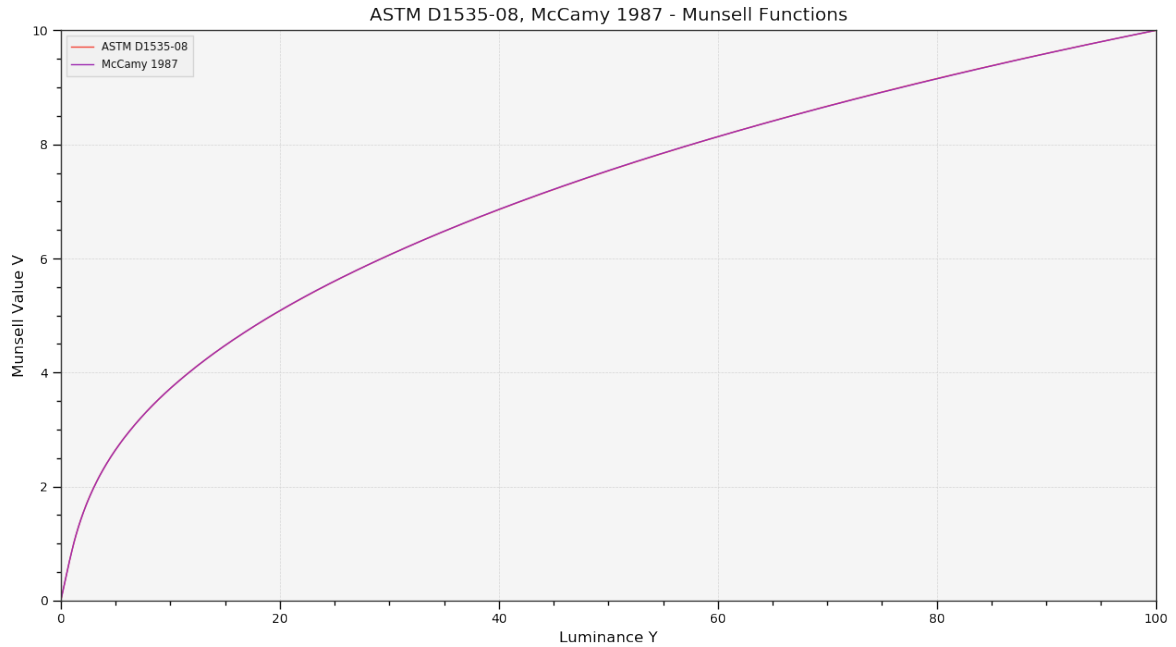
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_multi_munsell_value_functions(['ASTM D1535-08', 'McCamy 1987'])
... # doctest: +SKIP
```



Optical Phenomena

`colour.plotting`

<code>plot_single_sd_rayleigh_scattering([...])</code>	Plots a single <i>Rayleigh</i> scattering spectral distribution.
<code>plot_the_blue_sky([cmfs])</code>	Plots the blue sky.

`colour.plotting.plot_single_sd_rayleigh_scattering`

`colour.plotting.plot_single_sd_rayleigh_scattering(CO2_concentration=300, temperature=288.15, pressure=101325, latitude=0, altitude=0, cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)`

Plots a single *Rayleigh* scattering spectral distribution.

Parameters

- **CO2_concentration** (numeric, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric) – Surface pressure P of the measurement site.
- **latitude** (numeric, optional) – Latitude of the site in degrees.
- **altitude** (numeric, optional) – Altitude of the site in meters.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.

Other Parameters

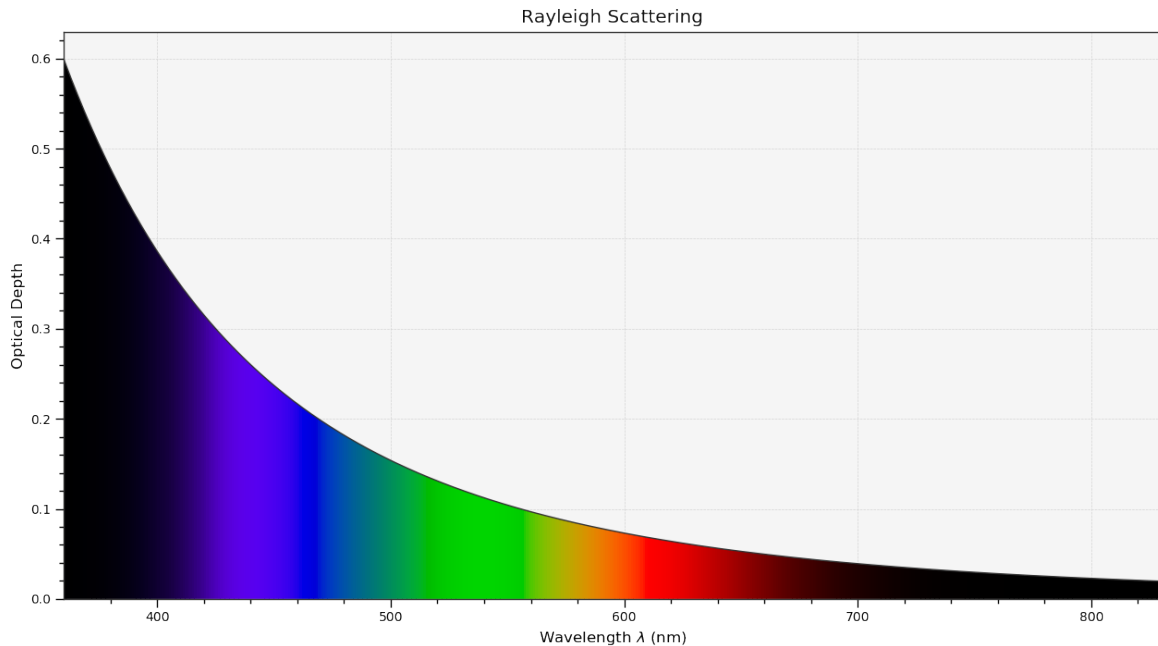
- ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.
- **out_of_gamut_clipping** (*bool, optional*) – {`colour.plotting.plot_single_sd()`}, Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_single_sd_rayleigh_scattering() # doctest: +SKIP
```



`colour.plotting.plot_the_blue_sky`

`colour.plotting.plot_the_blue_sky(cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)`
Plots the blue sky.

Parameters `cmfs` (*unicode, optional*) – Standard observer colour matching functions.

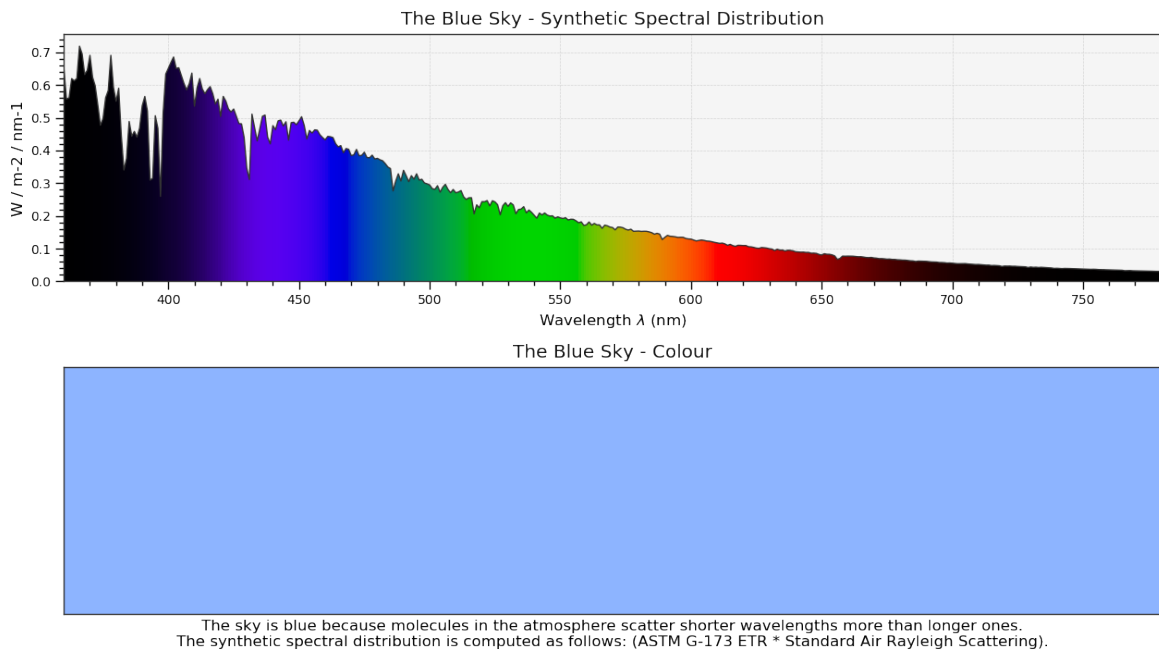
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_single_sd()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_the_blue_sky() # doctest: +SKIP
```



Colour Quality

colour.plotting

<code>plot_single_sd_colour_rendering_indexBars(sd, ...)</code>	Plots the <i>Colour Rendering Index</i> (CRI) of given illuminant or light source spectral distribution.
<code>plot_multi_sds_colour_rendering_indexesBars(sds)</code>	Plots the <i>Colour Rendering Index</i> (CRI) of given illuminants or light sources spectral distributions.
<code>plot_single_sd_colour_quality_scaleBars(sd)</code>	Plots the <i>Colour Quality Scale</i> (CQS) of given illuminant or light source spectral distribution.
<code>plot_multi_sds_colour_quality_scalesBars(sds)</code>	Plots the <i>Colour Quality Scale</i> (CQS) of given illuminants or light sources spectral distributions.

colour.plotting.plot_single_sd_colour_rendering_indexBars

colour.plotting.plot_single_sd_colour_rendering_indexBars(sd, **kwargs)

Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.

Parameters `sd` (`SpectralDistribution`) – Illuminant or light source spectral distribution to plot the *Colour Rendering Index* (CRI).

Other Parameters

- ****kwargs** (`dict`, *optional*) – {`colour.plotting.artist()`, `colour.plotting.quality.plot_colour_qualityBars()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

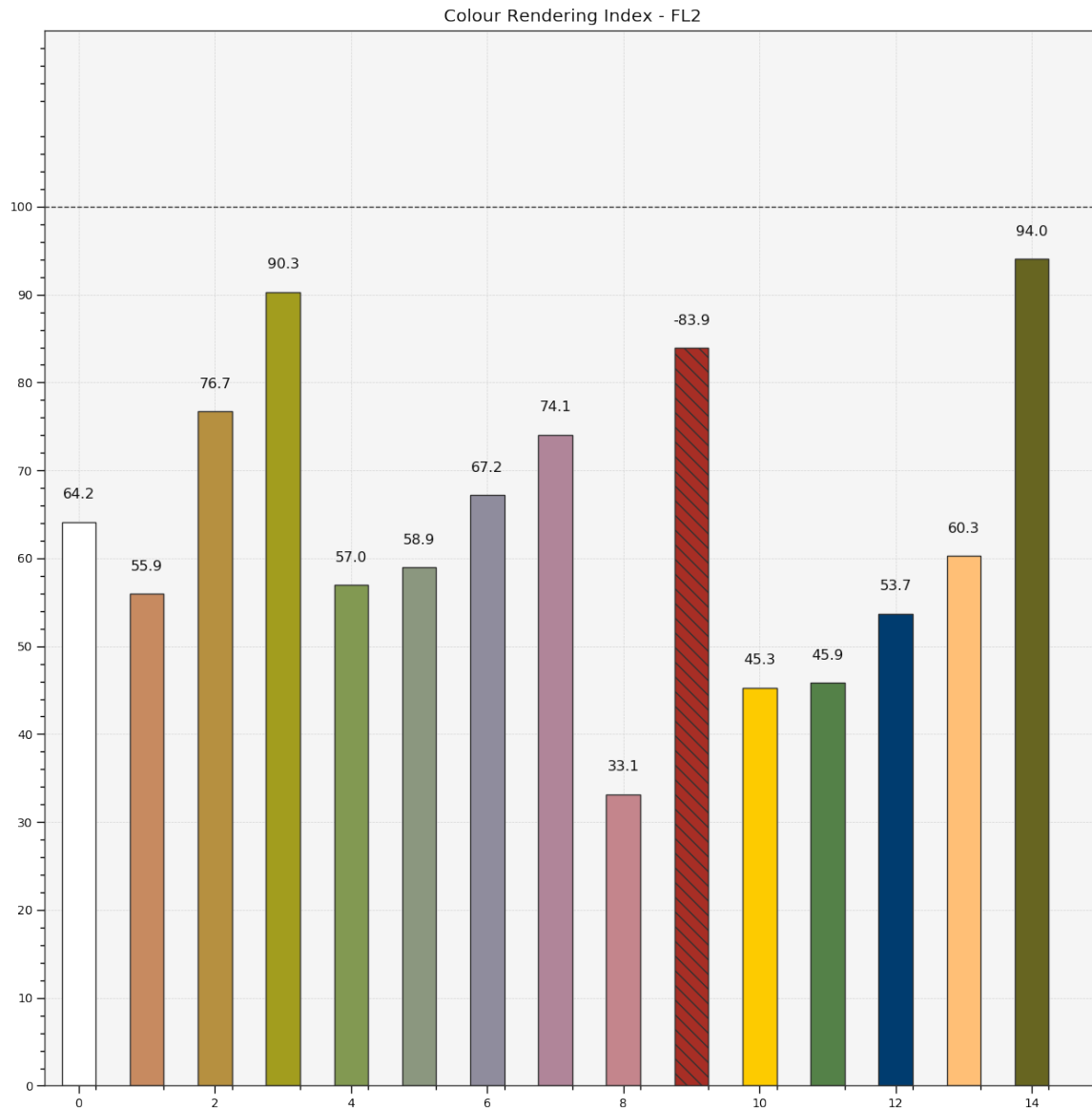
- **labels** (*bool, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> plot_single_sd_colour_rendering_index_bars(illuminant)
... # doctest: +SKIP
```



`colour.plotting.plot_multi_sds_colour_rendering_indexes_bars`

`colour.plotting.plot_multi_sds_colour_rendering_indexes_bars(sds, **kwargs)`

Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.

Parameters `sds` (*array_like*) – Array of illuminants or light sources spectral distributions to plot the *Colour Rendering Index* (CRI).

Other Parameters

- ****kwargs** (*dict, optional*) – `{colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}`, Please refer to the documentation of the previously listed definitions.
- **labels** (*bool, optional*) – `{colour.plotting.quality.plot_colour_quality_bars()}`, Add labels above bars.

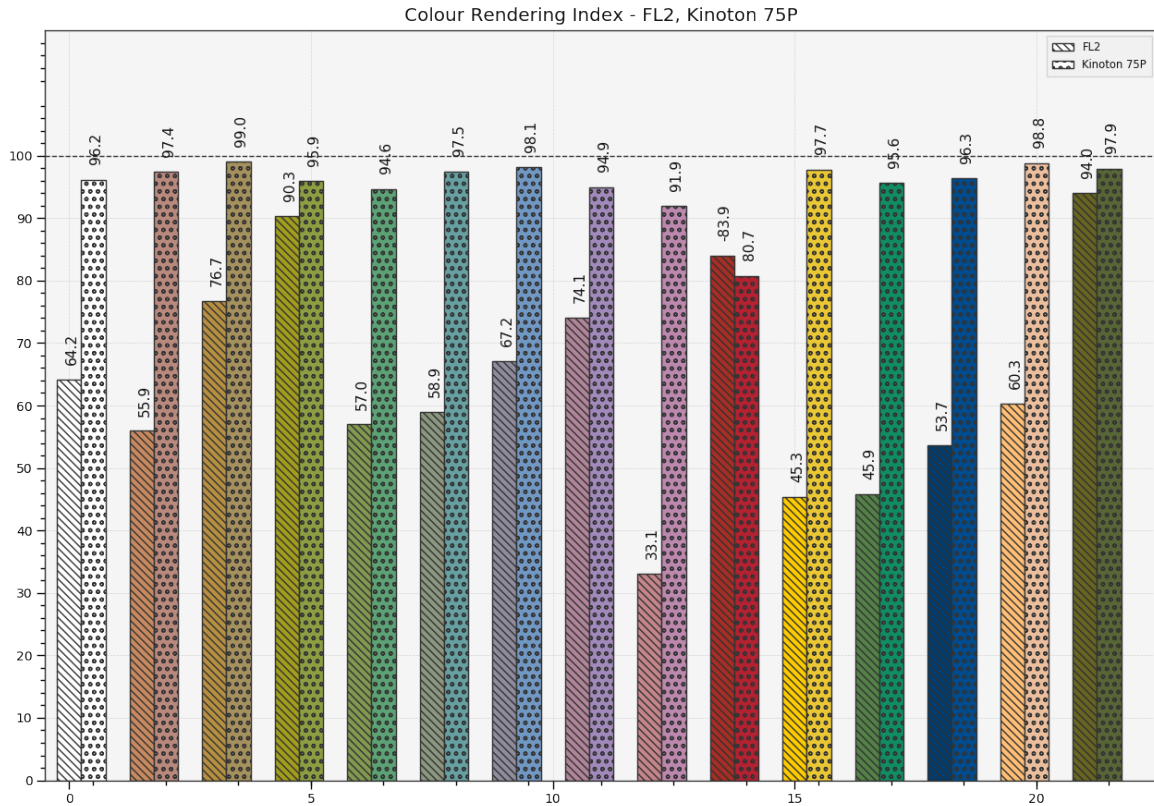
- **hatching** (*bool or None, optional*) – `{colour.plotting.quality.plot_colour_qualityBars()}`, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – `{colour.plotting.quality.plot_colour_qualityBars()}`, Hatching pattern repeat.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> plot_multi_sds_colour_rendering_indexes_bars(
...     [illuminant, light_source]) # doctest: +SKIP
```



colour.plotting.plot_single_sd_colour_quality_scale_bars

```
colour.plotting.plot_single_sd_colour_quality_scale_bars(sd, method='NIST CQS 7.4',
**kwargs)
```

Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.

Parameters

- **sd** (*SpectralDistribution*) – Illuminant or light source spectral distribution to plot the *Colour Quality Scale* (CQS).
- **method** (unicode, optional) – {**NIST CQS 7.4**}, *Colour Quality Scale* (CQS) computation method.

Other Parameters

- ****kwargs** (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.`

`quality.plot_colour_quality_bars()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

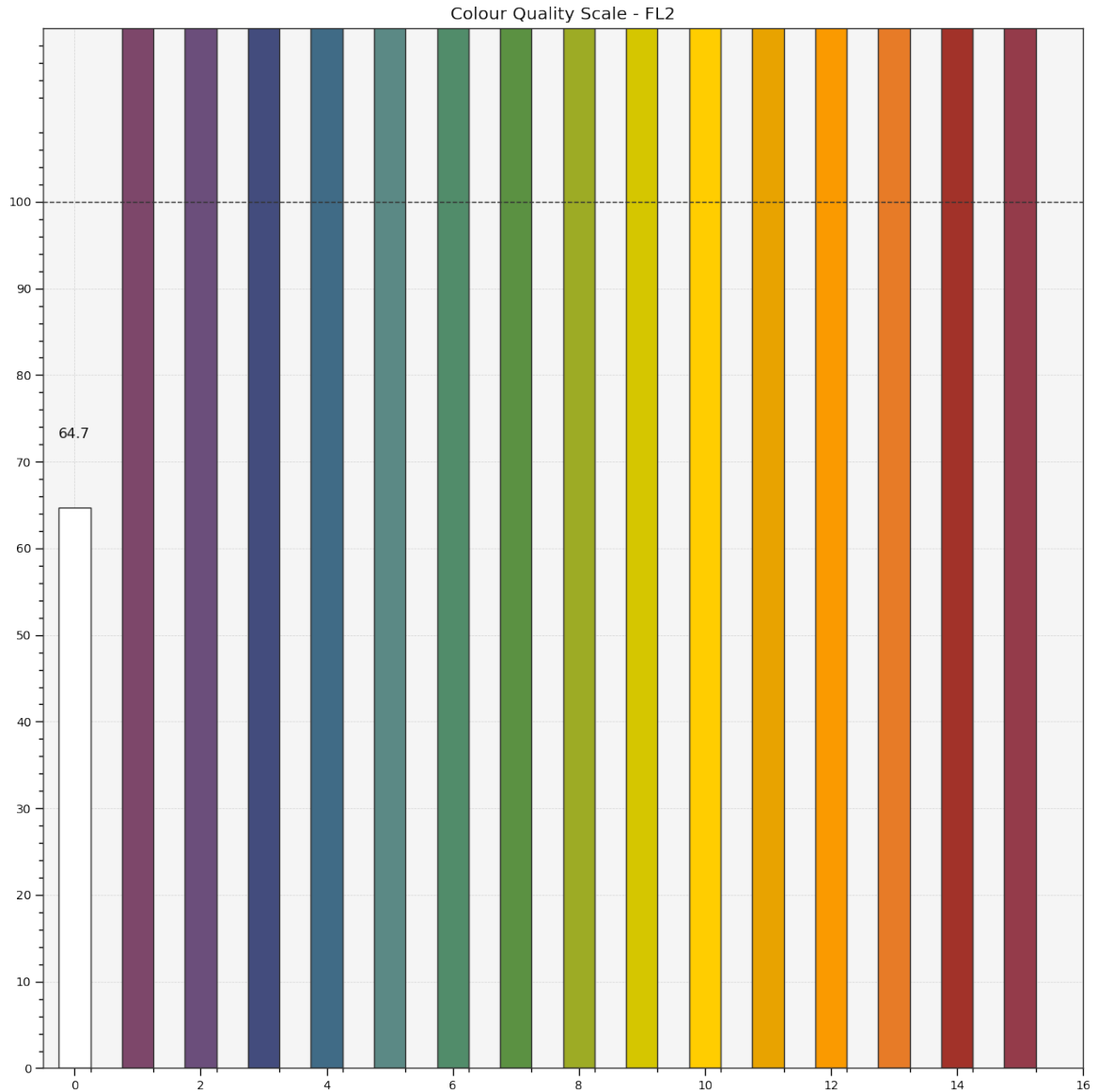
- **labels** (*bool, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Hatching pattern repeat.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> plot_single_sd_colour_quality_scale_bars(illuminant)
... # doctest: +SKIP
```



`colour.plotting.plot_multi_sds_colour_quality_scales_bars`

`colour.plotting.plot_multi_sds_colour_quality_scales_bars(sds, method='NIST CQS 7.4', **kwargs)`

Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

Parameters

- **sds** (array_like) – Array of illuminants or light sources spectral distributions to plot the *Colour Quality Scale* (CQS).
- **method** (unicode, optional) – {**NIST CQS 7.4**}, *Colour Quality Scale* (CQS) computation method.

Other Parameters

- ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.`

`quality.plot_colour_quality_bars()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

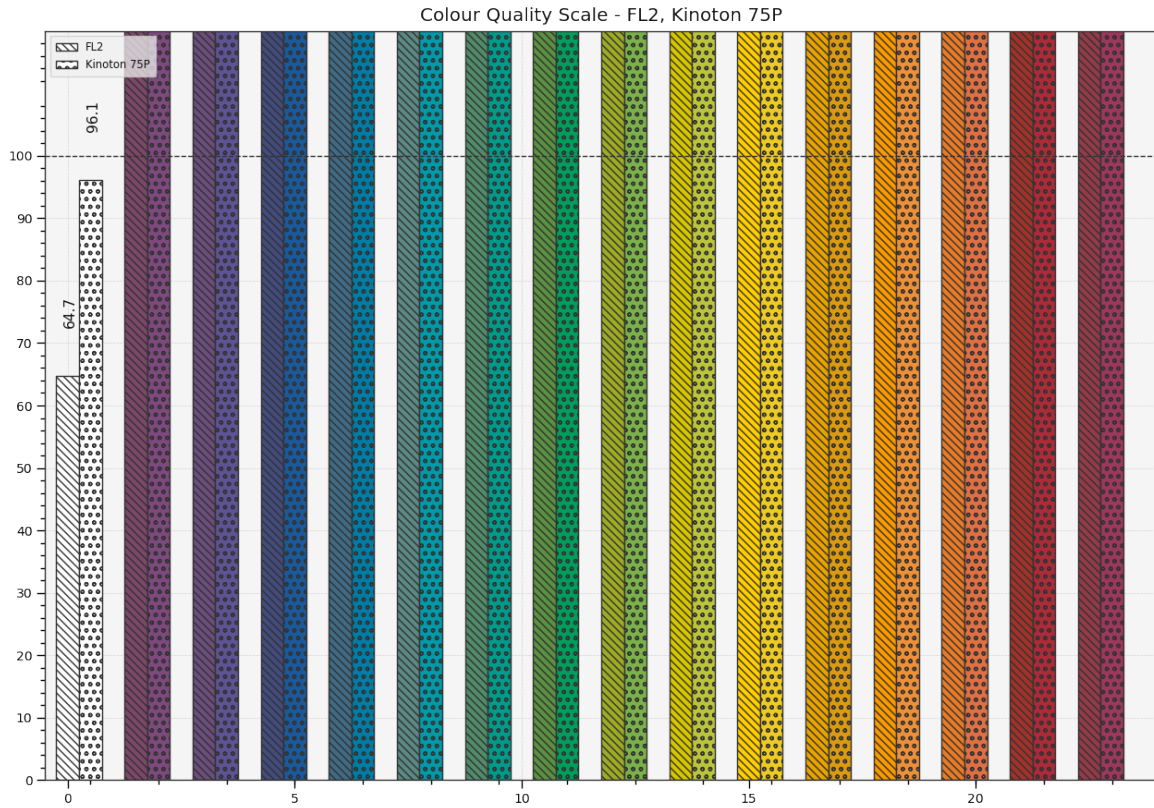
- **labels** (*bool, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {`colour.plotting.quality.plot_colour_quality_bars()`}, Hatching pattern repeat.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> plot_multi_sds_colour_quality_scales_bars([illuminant, light_source])
... # doctest: +SKIP
```



Ancillary Objects

`colour.plotting.quality`

`plot_colour_qualityBars(specifications[, ...])`

Plots the colour quality data of given illuminants or light sources colour quality specifications.

`colour.plotting.quality.plot_colour_qualityBars`

`colour.plotting.quality.plot_colour_qualityBars(specifications, labels=True, hatching=None, hatching_repeat=2, **kwargs)`

Plots the colour quality data of given illuminants or light sources colour quality specifications.

Parameters

- **specifications** (array_like) – Array of illuminants or light sources colour quality

specifications.

- **labels** (*bool*, optional) – Add labels above bars.
- **hatching** (*bool* or *None*, optional) – Use hatching for the bars.
- **hatching_repeat** (*int*, optional) – Hatching pattern repeat.

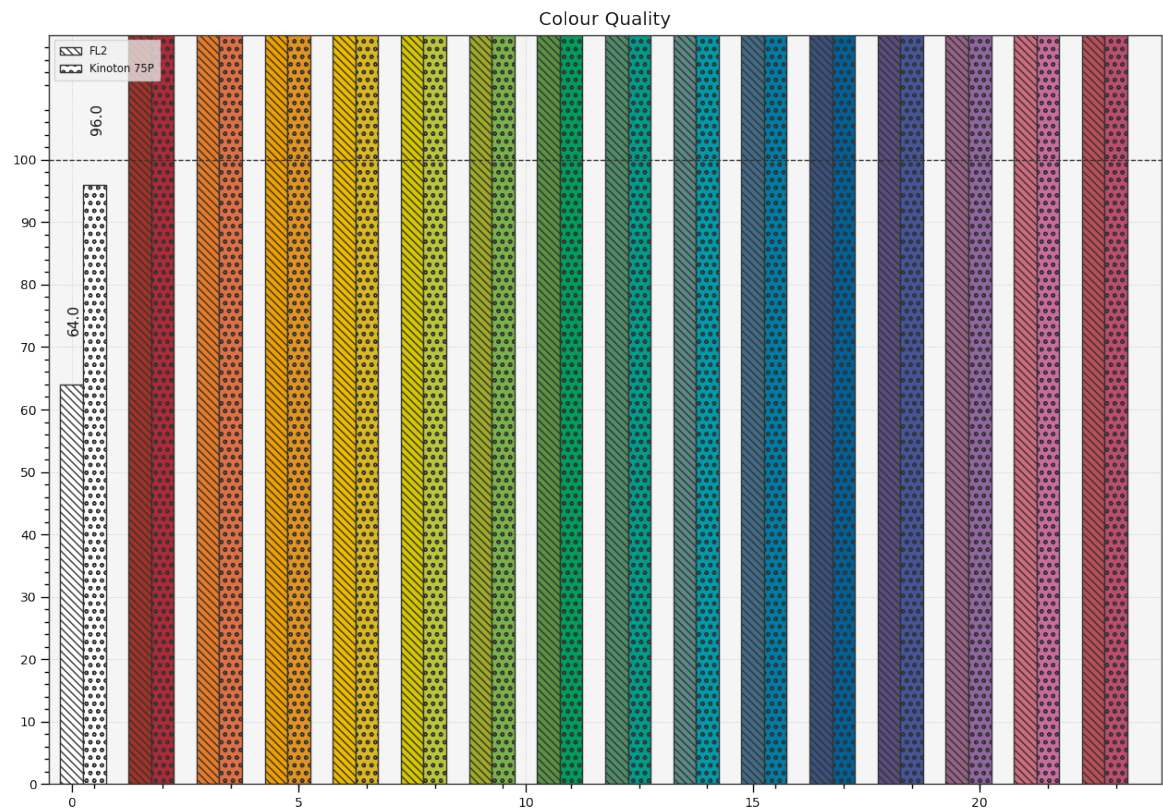
Other Parameters ***kwargs* (*dict*, optional) – {`colour.plotting.artist()`, `colour.plotting.quality.plot_colour_quality_bars()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type *tuple*

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS, SpectralShape)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> light_source = light_source.copy().align(SpectralShape(360, 830, 1))
>>> cqs_i = colour_quality_scale(illuminant, additional_data=True)
>>> cqs_l = colour_quality_scale(light_source, additional_data=True)
>>> plot_colour_quality_bars([cqs_i, cqs_l]) # doctest: +SKIP
```



Colour Temperature & Correlated Colour Temperature

colour.plotting

<code>plot_planckian_locus_in_chromaticity_diagram_CIE1931</code>	Plot the Planckian Locus and given illuminants in CIE 1931 Chromaticity Diagram.
<code>plot_planckian_locus_in_chromaticity_diagram_CIE1960</code>	Plot the Planckian Locus and given illuminants in CIE 1960 UCS Chromaticity Diagram.

colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931

```
colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931(illuminants=None,
                                                                    anno-
                                                                    tate_parameters=None,
                                                                    chromatic-
                                                                    ity_diagram_callable_CIE1931=<function
                                                                    plot_chromaticity_diagram_CIE1931>,
                                                                    **kwargs)
```

Plots the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

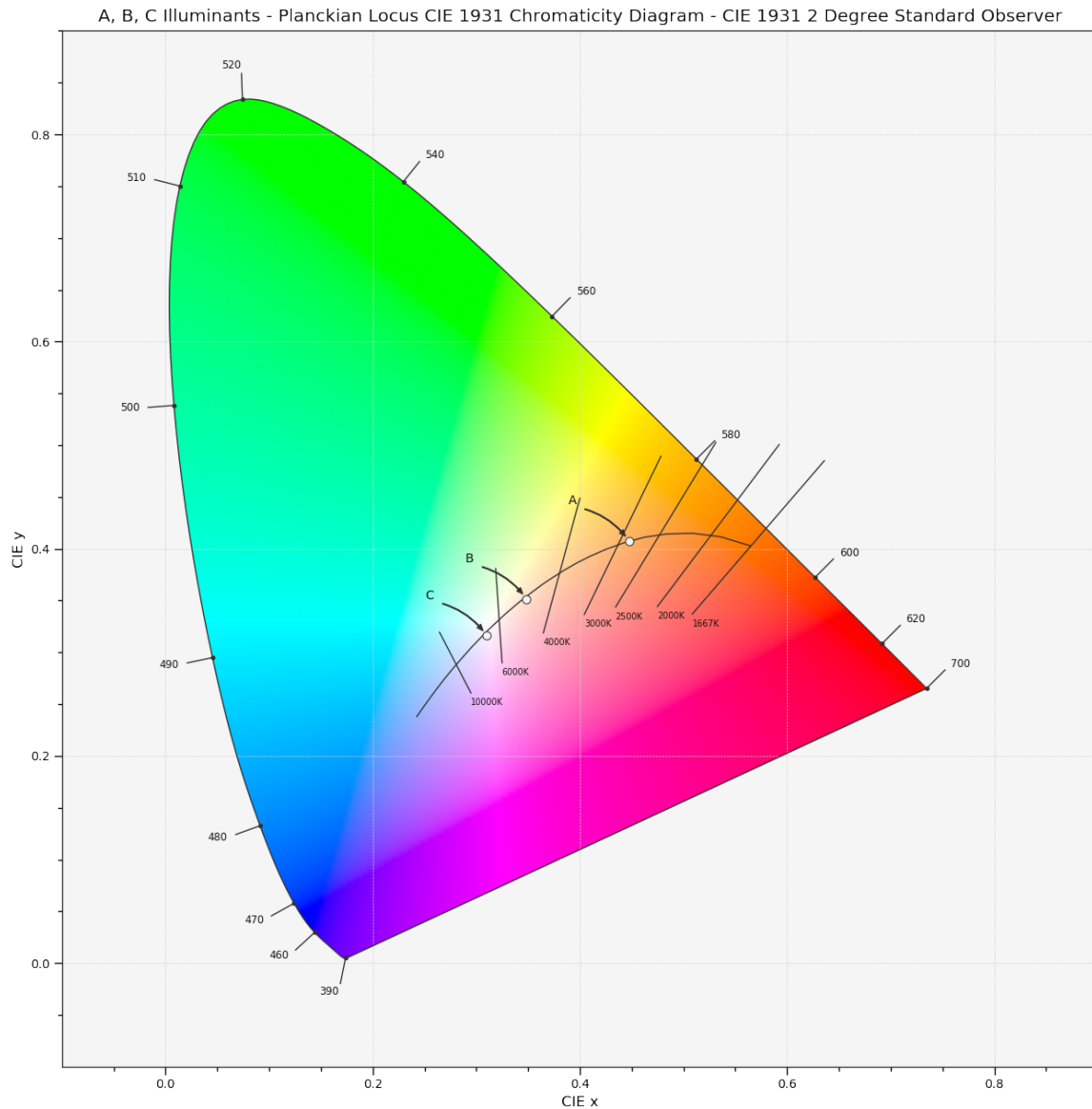
Other Parameters ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.temperature.plot_planckian_locus(), colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1931(['A', 'B', 'C'])
... # doctest: +SKIP
```



colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS

```
colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(illuminants=None,
                                                                           anno-
                                                                           tate_parameters=None,
                                                                           chromatic-
                                                                           ity_diagram_callable_CIE1960UCS=<func
                                                                           plot_chromaticity_diagram_CIE1960UCS>
                                                                           **kwargs)
```

Plots the *Planckian Locus* and given illuminants in CIE 1960 UCS Chromaticity Diagram.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.

`annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if `annotate` is set to *True*. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.

- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

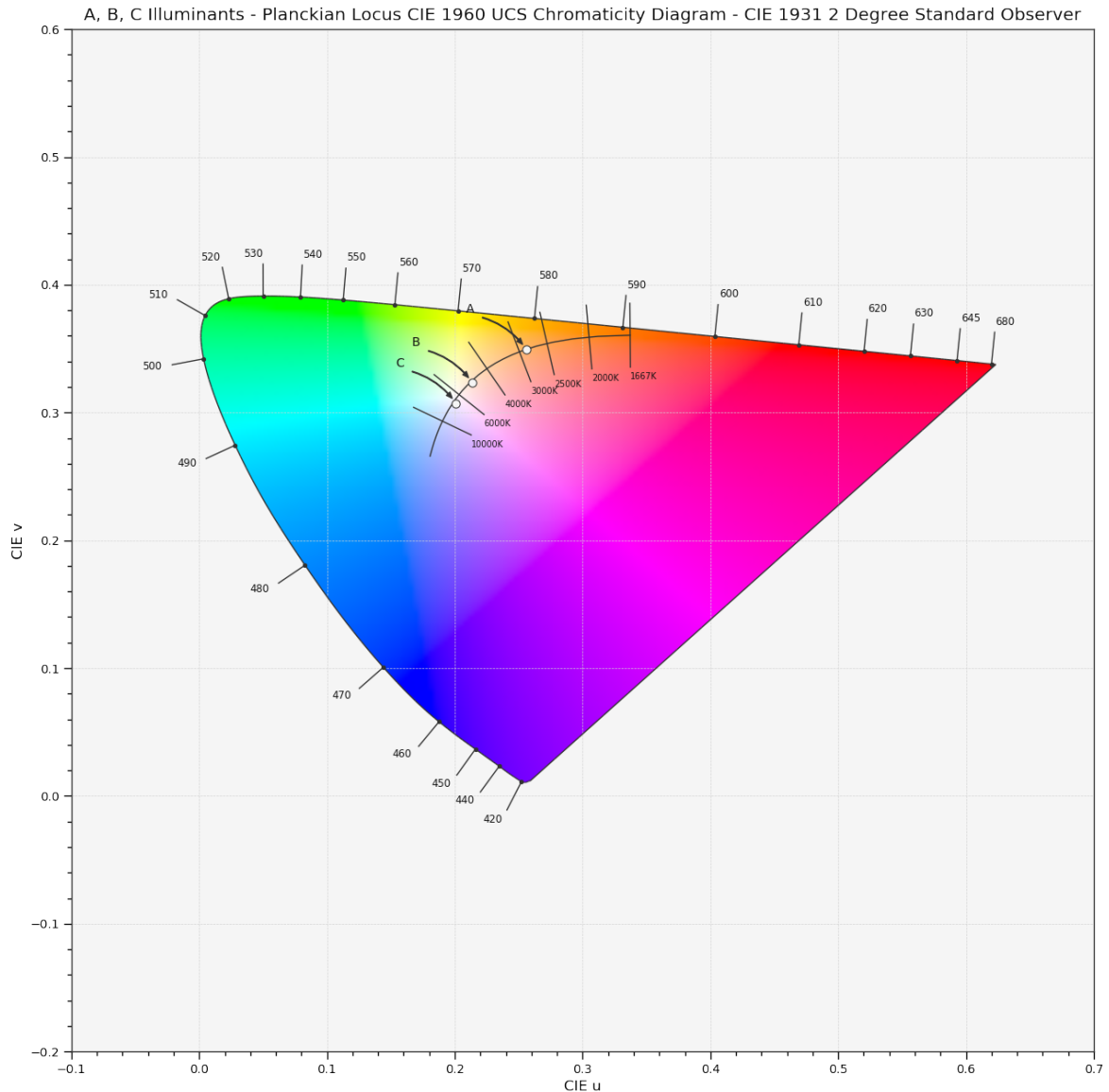
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.temperature.plot_planckian_locus()`, `colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(
...     ['A', 'C', 'E']) # doctest: +SKIP
```

Ancillary Objects

`colour.plotting.temperature`

`plot_planckian_locus([...])`

Plots the *Planckian Locus* according to given method.

`plot_planckian_locus_in_chromaticity_diagram([...])`

Plots the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given method.

`colour.plotting.temperature.plot_planckian_locus`

`colour.plotting.temperature.plot_planckian_locus(planckian_locus_colours=None, method='CIE 1931', **kwargs)`

Plots the *Planckian Locus* according to given method.

Parameters

- **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

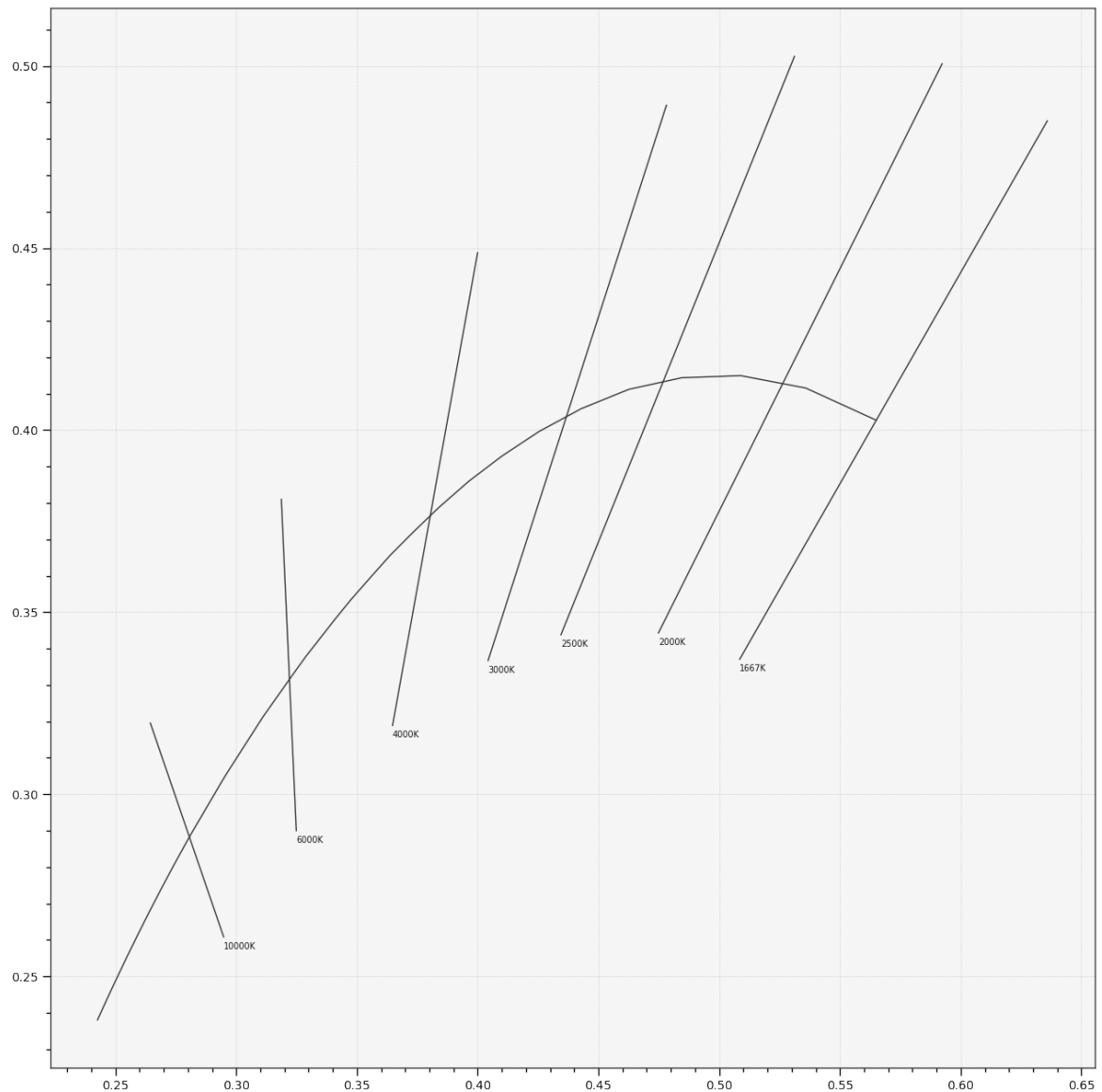
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_planckian_locus() # doctest: +SKIP
```



`colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram`

```
colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram(illuminants=None,
                                                                           anno-
                                                                           tate_parameters=None,
                                                                           chromatic-
                                                                           ity_diagram_callable=<function
                                                                           plot_chromaticity_diagram>,
                                                                           method='CIE
                                                                           1931', **kwargs)
```

Plots the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given method.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.

- **annotate_parameters** (`dict` or `array_like`, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if `annotate` is set to `True`. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – {'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}, *Chromaticity Diagram* method.

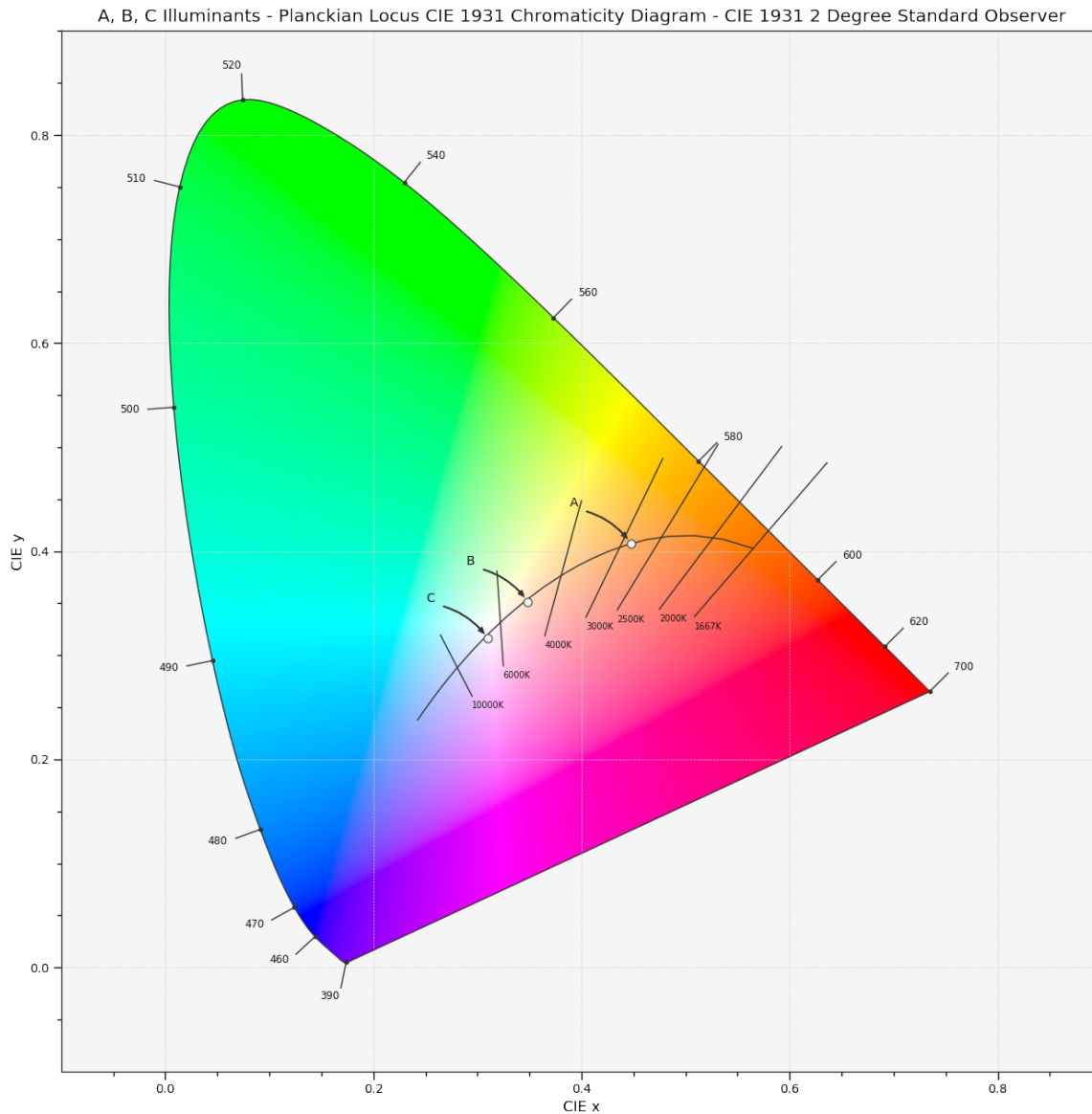
Other Parameters ****kwargs** (`dict`, optional) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.temperature.plot_planckian_locus()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram(['A', 'B', 'C'])
... # doctest: +SKIP
```



Colour Models Volume

colour.plotting

<code>plot_RGB_colourspace_gamuts([colourspace, ...])</code>	Plots given <i>RGB</i> colourspaces gamuts in given reference colourspace.
<code>plot_RGB_scatter(RGB, colourspace[, ...])</code>	Plots given <i>RGB</i> colourspace array in a scatter plot.

colour.plotting.plot_RGB_colourspaces_gamuts

```
colour.plotting.plot_RGB_colourspaces_gamuts(colourspaces=None, reference_colourspace='CIE
xyY', segments=8, show_grid=True,
grid_segments=10, show_spectral_locus=False,
spectral_locus_colour=None, cmfs='CIE 1931 2
Degree Standard Observer', **kwargs)
```

Plots given *RGB* colourspace gamuts in given reference colourspace.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot the gamuts.
- **reference_colourspace** (unicode, optional) – {'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}, Reference colourspace to plot the gamuts into.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool, optional) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **show_spectral_locus** (bool, optional) – Whether to show the spectral locus.
- **spectral_locus_colour** (array_like, optional) – Spectral locus colour.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

Other Parameters

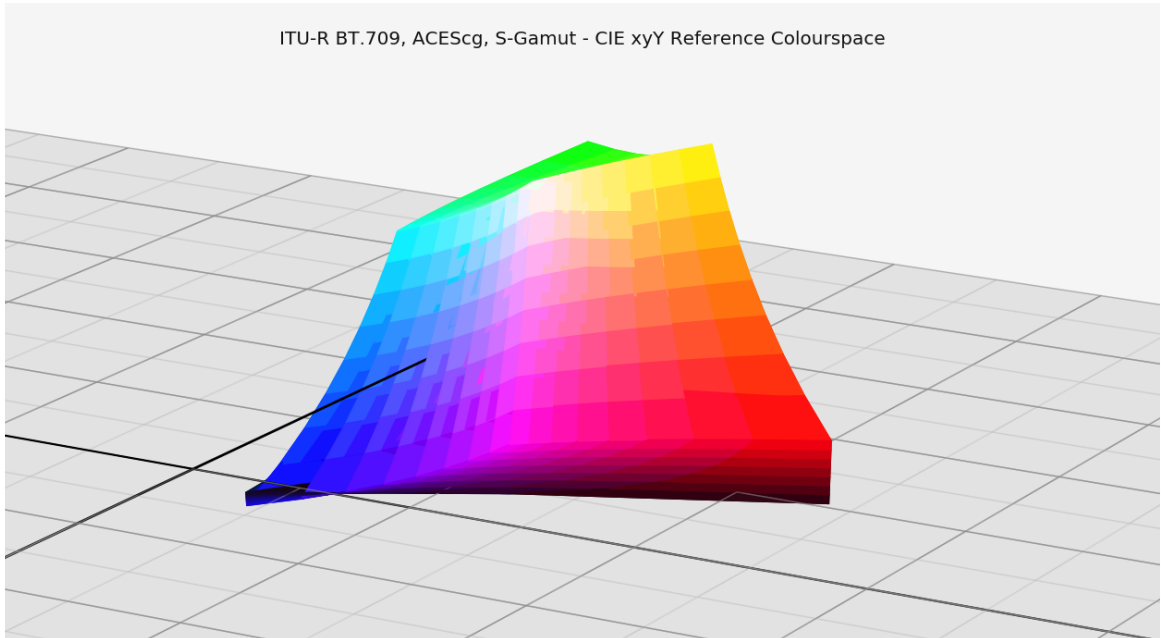
- ****kwargs** (dict, optional) – {`colour.plotting.artist()`, `colour.plotting.volume.nadir_grid()`}, Please refer to the documentation of the previously listed definitions.
- **face_colours** (array_like, optional) – Face colours array such as `face_colours = (None, (0.5, 0.5, 1.0))`.
- **edge_colours** (array_like, optional) – Edge colours array such as `edge_colours = (None, (0.5, 0.5, 1.0))`.
- **face_alpha** (numeric, optional) – Face opacity value such as `face_alpha = (0.5, 1.0)`.
- **edge_alpha** (numeric, optional) – Edge opacity value such as `edge_alpha = (0.0, 1.0)`.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_gamuts(['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
... # doctest: +SKIP
```



colour.plotting.plot_RGB_scatter

```
colour.plotting.plot_RGB_scatter(
    RGB,          colourspace,          reference_colourspace='CIE xyY',
    colourspace=None, segments=8, show_grid=True,
    grid_segments=10, show_spectral_locus=False, spectral_locus_colour=None,
    points_size=12, cmfs='CIE 1931 2 Degree Standard Observer', **kwargs)
```

Plots given *RGB* colourspace array in a scatter plot.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (*RGB_Colourspace*) – *RGB* colourspace of the *RGB* array.
- **reference_colourspace** (unicode, optional) – {'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}, Reference colourspace for colour conversion.
- **colourspace** (array_like, optional) – *RGB* colourspace to plot the gamuts.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool, optional) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **show_spectral_locus** (bool, optional) – Whether to show the spectral locus.
- **spectral_locus_colour** (array_like, optional) – Spectral locus colour.
- **points_size** (numeric, optional) – Scatter points size.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

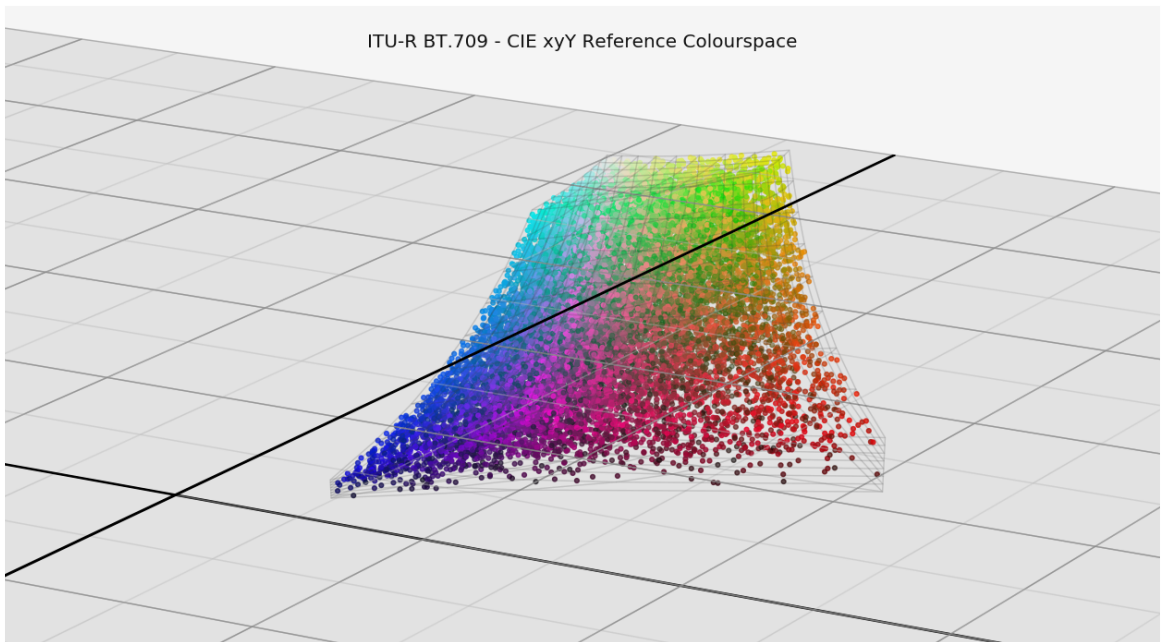
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_RGB_colourspaces_gamuts()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type `tuple`

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_scatter(RGB, 'ITU-R BT.709') # doctest: +SKIP
```



Geometry Plotting Utilities

`colour.plotting`

<code>quad([plane, origin, width, height, depth])</code>	Returns the vertices of a quad geometric element in counter-clockwise order.
<code>grid([plane, origin, width, height, depth, ...])</code>	Returns the vertices of a grid made of quads.
<code>cube([plane, origin, width, height, depth, ...])</code>	Returns the vertices of a cube made of grids.

`colour.plotting.quad`

`colour.plotting.quad(plane='xy', origin=None, width=1, height=1, depth=0)`
Returns the vertices of a quad geometric element in counter-clockwise order.

Parameters

- **plane** (*array_like, optional*) – {'xy', 'xz', 'yz'}, Construction plane of the quad.

- **origin** (array_like, optional) – Quad origin on the construction plane.
- **width** (numeric, optional) – Quad width.
- **height** (numeric, optional) – Quad height.
- **depth** (numeric, optional) – Quad depth.

Returns Quad vertices.

Return type ndarray

Examples

```
>>> quad()
array([[0, 0, 0],
       [1, 0, 0],
       [1, 1, 0],
       [0, 1, 0]])
```

colour.plotting.grid

`colour.plotting.grid(plane='xy', origin=None, width=1, height=1, depth=0, width_segments=1, height_segments=1)`

Returns the vertices of a grid made of quads.

Parameters

- **plane** (array_like, optional) – {'xy', 'xz', 'yz'}, Construction plane of the grid.
- **origin** (array_like, optional) – Grid origin on the construction plane.
- **width** (numeric, optional) – Grid width.
- **height** (numeric, optional) – Grid height.
- **depth** (numeric, optional) – Grid depth.
- **width_segments** (int, optional) – Grid segments, quad counts along the width.
- **height_segments** (int, optional) – Grid segments, quad counts along the height.

Returns Grid vertices.

Return type ndarray

Examples

```
>>> grid(width_segments=2, height_segments=2)
array([[ 0. ,  0. ,  0. ],
       [ 0.5,  0. ,  0. ],
       [ 0.5,  0.5,  0. ],
       [ 0. ,  0.5,  0. ]],
<BLANKLINE>
       [[ 0. ,  0.5,  0. ],
       [ 0.5,  0.5,  0. ],
       [ 0.5,  1. ,  0. ],
       [ 0. ,  1. ,  0. ]],
<BLANKLINE>
```

(continues on next page)

(continued from previous page)

```
[[ 0.5, 0. , 0. ],
 [ 1. , 0. , 0. ],
 [ 1. , 0.5, 0. ],
 [ 0.5, 0.5, 0. ]],
<BLANKLINE>
[[ 0.5, 0.5, 0. ],
 [ 1. , 0.5, 0. ],
 [ 1. , 1. , 0. ],
 [ 0.5, 1. , 0. ]]])
```

colour.plotting.cube

`colour.plotting.cube(plane=None, origin=None, width=1, height=1, depth=1, width_segments=1, height_segments=1, depth_segments=1)`

Returns the vertices of a cube made of grids.

Parameters

- **plane** (array_like, optional) – Any combination of {'+x', '-x', '+y', '-y', '+z', '-z'}, Included grids in the cube construction.
- **origin** (array_like, optional) – Cube origin.
- **width** (numeric, optional) – Cube width.
- **height** (numeric, optional) – Cube height.
- **depth** (numeric, optional) – Cube depth.
- **width_segments** (int, optional) – Cube segments, quad counts along the width.
- **height_segments** (int, optional) – Cube segments, quad counts along the height.
- **depth_segments** (int, optional) – Cube segments, quad counts along the depth.

Returns Cube vertices.

Return type ndarray

Examples

```
>>> cube()
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]],
<BLANKLINE>
       [[ 0.,  0.,  1.],
       [ 1.,  0.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  1.,  1.]],
<BLANKLINE>
       [[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  1.],
       [ 0.,  0.,  1.]],
<BLANKLINE>)
```

(continues on next page)

(continued from previous page)

```
[[ 0., 1., 0.],
 [ 1., 1., 0.],
 [ 1., 1., 1.],
 [ 0., 1., 1.]],
<BLANKLINE>
[[ 0., 0., 0.],
 [ 0., 1., 0.],
 [ 0., 1., 1.],
 [ 0., 0., 1.]],
<BLANKLINE>
[[ 1., 0., 0.],
 [ 1., 1., 0.],
 [ 1., 1., 1.],
 [ 1., 0., 1.]]])
```

Colour Quality

- *Colour Rendering Index*
- *Colour Quality Scale*

Colour Rendering Index

colour

<code>colour_rendering_index(sd_test[, ...])</code>	Returns the <i>Colour Rendering Index</i> (CRI) Q_a of given spectral distribution.
---	---

colour.colour_rendering_index

`colour.colour_rendering_index(sd_test, additional_data=False)`
Returns the *Colour Rendering Index* (CRI) Q_a of given spectral distribution.

Parameters

- **sd_test** (*SpectralDistribution*) – Test spectral distribution.
- **additional_data** (*bool*, optional) – Whether to output additional data.

Returns *Colour Rendering Index* (CRI).

Return type numeric or *CRI_Specification*

References

[OD08]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> sd = ILLUMINANTS_SDS['FL2']
>>> colour_rendering_index(sd) # doctest: +ELLIPSIS
64.1515202...
```

colour.quality

CRI_Specification	Defines the <i>Colour Rendering Index</i> (CRI) colour quality specification.
-------------------	---

colour.quality.CRI_Specification

class colour.quality.CRI_Specification

Defines the *Colour Rendering Index* (CRI) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **Q_a** (numeric) – *Colour Rendering Index* (CRI) Q_a .
- **Q_as** (dict) – Individual *colour rendering indexes* data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

[OD08]

Create new instance of CRI_Specification(name, Q_a, Q_as, colorimetry_data)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

Q_a	Alias for field number 1
Q_as	Alias for field number 2
colorimetry_data	Alias for field number 3
name	Alias for field number 0

Colour Quality Scale

colour

<code>COLOUR_QUALITY_SCALE_METHODS</code>	Supported <i>Colour Quality Scale</i> (CQS) computation methods.
<code>colour_quality_scale(sd_test[, ...])</code>	Returns the <i>Colour Quality Scale</i> (CQS) of given spectral distribution using given method.

colour.COLOUR_QUALITY_SCALE_METHODS

`colour.COLOUR_QUALITY_SCALE_METHODS = ('NIST CQS 7.4', 'NIST CQS 9.0')`
Supported *Colour Quality Scale* (CQS) computation methods.

References

[DO10], [OD08], [OD13]

`COLOUR_QUALITY_SCALE_METHODS` [tuple] { 'NIST CQS 9.0', 'NIST CQS 7.4'}

colour.colour_quality_scale

`colour.colour_quality_scale(sd_test, additional_data=False, method='NIST CQS 9.0')`
Returns the *Colour Quality Scale* (CQS) of given spectral distribution using given method.

Parameters

- **sd_test** (*SpectralDistribution*) – Test spectral distribution.
- **additional_data** (*bool*, optional) – Whether to output additional data.
- **method** (*unicode*, optional) – {'NIST CQS 9.0', 'NIST CQS 7.4'}, Computation method.

Returns Color quality scale.

Return type numeric or *CQS_Specification*

References

[DO10], [OD08], [OD13]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> sd = ILLUMINANTS_SDS['FL2']
>>> colour_quality_scale(sd) # doctest: +ELLIPSIS
64.0172835...
```

`colour.quality`

<i>CQS_Specification</i>	Defines the <i>Colour Quality Scale</i> (CQS) colour quality specification.
--------------------------	---

colour.quality.CQS_Specification**class** colour.quality.CQS_Specification

Defines the *Colour Quality Scale* (CQS) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **Q_a** (numeric) – Colour quality scale Q_a .
- **Q_f** (numeric) – Colour fidelity scale Q_f intended to evaluate the fidelity of object colour appearances (compared to the reference illuminant of the same correlated colour temperature and illuminance).
- **Q_p** (numeric) – Colour preference scale Q_p similar to colour quality scale Q_a but placing additional weight on preference of object colour appearance, set to *None* in *NIST CQS 9.0* method. This metric is based on the notion that increases in chroma are generally preferred and should be rewarded.
- **Q_g** (numeric) – Gamut area scale Q_g representing the relative gamut formed by the (a^*, b^*) coordinates of the 15 samples illuminated by the test light source in the *CIE $L^*a^*b^*$* object colourspace.
- **Q_d** (numeric) – Relative gamut area scale Q_d , set to *None* in *NIST CQS 9.0* method.
- **Q_as** (dict) – Individual *Colour Quality Scale* (CQS) data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

[DO10], [OD08], [OD13]

Create new instance of CQS_Specification(name, Q_a, Q_f, Q_p, Q_g, Q_d, Q_as, colorimetry_data)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

count	Return number of occurrences of value.
index	Return first index of value.

Attributes

Q_a	Alias for field number 1
Q_as	Alias for field number 6
Q_d	Alias for field number 5
Q_f	Alias for field number 2
Q_g	Alias for field number 4
Q_p	Alias for field number 3
colorimetry_data	Alias for field number 7
name	Alias for field number 0

Reflectance Recovery

- *CIE XYZ Colourspace to Spectral*
- *Smits (1999)*
- *Meng, Simon and Hanika (2015)*

CIE XYZ Colourspace to Spectral

colour

<code>XYZ_to_sd(XYZ[, method])</code>	Recovers the spectral distribution of given <i>CIE XYZ</i> tristimulus values using given method.
<code>XYZ_TO_SD_METHODS</code>	Supported spectral distribution recovery methods.

colour.XYZ_to_sd

`colour.XYZ_to_sd(XYZ, method='Meng 2015', **kwargs)`

Recovers the spectral distribution of given *CIE XYZ* tristimulus values using given method.

Parameters

- **XYZ** (*array_like*) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **method** (*unicode, optional*) – {'**Meng 2015**', '**Smits 1999**'}, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions*) – {`colour.recovery.XYZ_to_sd_Meng2015()`}, Standard observer colour matching functions.
- **interval** (*numeric, optional*) – {`colour.recovery.XYZ_to_sd_Meng2015()`}, Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **optimisation_parameters** (*dict_like, optional*) – {`colour.recovery.XYZ_to_sd_Meng2015()`}, Parameters for `scipy.optimize.minimize()` definition.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- *Smits (1999)* method will internally convert given *CIE XYZ* tristimulus values to *RGB* colourspace array assuming equal energy illuminant *E*.

References

[MSHD15], [Smi99]

Examples

Meng (2015) reflectance recovery:

```
>>> import numpy as np
>>> from colour.utilities import numpy_print_options
>>> from colour.colorimetry import sd_to_XYZ_integration
>>> XYZ = np.array([0.21781186, 0.12541048, 0.04697113])
>>> sd = XYZ_to_sd(XYZ, interval=10)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd # doctest: +SKIP
SpectralDistribution([[ 360.      ,  0.0741540...],
                    [ 370.      ,  0.0741409...],
                    [ 380.      ,  0.0741287...],
                    [ 390.      ,  0.0740876...],
                    [ 400.      ,  0.0740215...],
                    [ 410.      ,  0.0738692...],
                    [ 420.      ,  0.0731412...],
                    [ 430.      ,  0.0705798...],
                    [ 440.      ,  0.0647359...],
                    [ 450.      ,  0.0551962...],
                    [ 460.      ,  0.0425597...],
                    [ 470.      ,  0.0283678...],
                    [ 480.      ,  0.0147370...],
                    [ 490.      ,  0.0044271...],
                    [ 500.      ,  0.0000302...],
                    [ 510.      ,  0.      ...],
                    [ 520.      ,  0.      ...],
                    [ 530.      ,  0.      ...],
                    [ 540.      ,  0.0051962...],
                    [ 550.      ,  0.0289516...],
                    [ 560.      ,  0.0687006...],
                    [ 570.      ,  0.1204130...],
                    [ 580.      ,  0.1789378...],
                    [ 590.      ,  0.2383451...],
                    [ 600.      ,  0.2930157...],
                    [ 610.      ,  0.3387433...],
                    [ 620.      ,  0.3734033...],
                    [ 630.      ,  0.3972820...],
                    [ 640.      ,  0.4125508...],
                    [ 650.      ,  0.4215782...],
                    [ 660.      ,  0.4265503...],
                    [ 670.      ,  0.4292647...],
                    [ 680.      ,  0.4307000...],
                    [ 690.      ,  0.4313993...],
                    [ 700.      ,  0.4316316...],
                    [ 710.      ,  0.4317109...],
                    [ 720.      ,  0.4317684...],
                    [ 730.      ,  0.4317864...],
                    [ 740.      ,  0.4317972...],
                    [ 750.      ,  0.4318385...],
                    [ 760.      ,  0.4318576...],
```

(continues on next page)

(continued from previous page)

```

[ 770.      , 0.4318455...],
[ 780.      , 0.4317877...],
[ 790.      , 0.4318119...],
[ 800.      , 0.4318070...],
[ 810.      , 0.4318089...],
[ 820.      , 0.4317781...],
[ 830.      , 0.4317733...]],
interpolator=SpragueInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100 # doctest: +ELLIPSIS
array([ 0.2178552..., 0.1254142..., 0.0470105...])

```

Smits (1999) reflectance recovery:

```

>>> sd = XYZ_to_sd(XYZ, method='Smits 1999')
>>> with numpy_print_options(suppress=True):
...     sd # doctest: +ELLIPSIS
SpectralDistribution([[ 380.      , 0.07691923],
[ 417.7778 , 0.0587005 ],
[ 455.5556 , 0.03943195],
[ 493.3333 , 0.03024978],
[ 531.1111 , 0.02750692],
[ 568.8889 , 0.02808645],
[ 606.6667 , 0.34298985],
[ 644.4444 , 0.41185795],
[ 682.2222 , 0.41185795],
[ 720.      , 0.41180754]],
interpolator=LinearInterpolator,
interpolator_args={},
extrapolator=Extrapolator,
extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100 # doctest: +ELLIPSIS
array([ 0.2004540..., 0.1105632..., 0.0420963...])

```

colour.XYZ_TO_SD_METHODS

`colour.XYZ_TO_SD_METHODS = CaseInsensitiveMapping({'Meng 2015': ..., 'Smits 1999': ...})`
Supported spectral distribution recovery methods.

References

[MSHD15], [Smi99]

`XYZ_TO_SD_METHODS` [CaseInsensitiveMapping] {'Meng 2015', 'Smits 1999'}

Smits (1999)

`colour.recovery`

<code>RGB_to_sd_Smits1999(</code> <code>RGB)</code>	Recovers the spectral distribution of given <i>RGB</i> colourspace array using <i>Smits (1999)</i> method.
<code>SMITS_1999_SDS</code>	<i>Smits (1999)</i> spectral distributions.

colour.recovery.RGB_to_sd_Smits1999

`colour.recovery.RGB_to_sd_Smits1999`(*RGB*)

Recovers the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.

Parameters *RGB* (array_like, (3,)) – *RGB* colourspace array to recover the spectral distribution from.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Smi99]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> RGB = np.array([0.40639599, 0.02752894, 0.03982193])
>>> with numpy_print_options(suppress=True):
...     RGB_to_sd_Smits1999(RGB) # doctest: +ELLIPSIS
SpectralDistribution([[ 380.      ,  0.0769192...],
                    [ 417.7778 ,  0.0587004...],
                    [ 455.5556 ,  0.0394319...],
                    [ 493.3333 ,  0.0302497...],
                    [ 531.1111 ,  0.0275069...],
                    [ 568.8889 ,  0.0280864...],
                    [ 606.6667 ,  0.3429898...],
                    [ 644.4444 ,  0.4118579...],
                    [ 682.2222 ,  0.4118579...],
                    [ 720.      ,  0.4118075...]],
                    interpolator=LinearInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
```

colour.recovery.SMITS_1999_SDS

`colour.recovery.SMITS_1999_SDS` = `CaseInsensitiveMapping`({'white': ..., 'cyan': ..., 'magenta': ..., 'yellow': ...})
Smits (1999) spectral distributions.

References

[Smi99]

SMITS_1999_SDS : CaseInsensitiveMapping

Meng, Simon and Hanika (2015)

`colour.recovery`

<code>XYZ_to_sd_Meng2015(XYZ[, cmfs, interval, ...])</code>	Recovers the spectral distribution of given <i>CIE XYZ</i> tristimulus values using <i>Meng et al.(2015)</i> method..
---	---

`colour.recovery.XYZ_to_sd_Meng2015`

`colour.recovery.XYZ_to_sd_Meng2015(XYZ, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), interval=5, optimisation_parameters=None)`

Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al. (2015)* method.

Parameters

- **XYZ** (`array_like`, (3,)) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (`XYZ_ColourMatchingFunctions`) – Standard observer colour matching functions.
- **interval** (`numeric`, `optional`) – Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **optimisation_parameters** (`dict_like`, `optional`) – Parameters for `scipy.optimize.minimize()` definition.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- The definition used to convert spectrum to *CIE XYZ* tristimulus values is `colour.colorimetry.spectral_to_XYZ_integration()` definition because it processes any measurement interval opposed to `colour.colorimetry.sd_to_XYZ_ASTME30815()` definition that handles only measurement interval of 1, 5, 10 or 20nm.

References

[MSHD15]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> sd = XYZ_to_sd_Meng2015(XYZ, interval=10)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd # doctest: +SKIP
SpectralDistribution([[ 360.      ,  0.0780368...],
                    [ 370.      ,  0.0780387...],
                    [ 380.      ,  0.0780469...],
                    [ 390.      ,  0.0780894...],
                    [ 400.      ,  0.0780285...],
                    [ 410.      ,  0.0777034...],
                    [ 420.      ,  0.0769175...],
                    [ 430.      ,  0.0746243...],
                    [ 440.      ,  0.0691410...],
                    [ 450.      ,  0.0599949...],
                    [ 460.      ,  0.04779   ...],
                    [ 470.      ,  0.0337270...],
                    [ 480.      ,  0.0196952...],
                    [ 490.      ,  0.0078056...],
                    [ 500.      ,  0.0004368...],
                    [ 510.      ,  0.0000065...],
                    [ 520.      ,  0.          ...],
                    [ 530.      ,  0.          ...],
                    [ 540.      ,  0.0124283...],
                    [ 550.      ,  0.0389186...],
                    [ 560.      ,  0.0774087...],
                    [ 570.      ,  0.1246716...],
                    [ 580.      ,  0.1765055...],
                    [ 590.      ,  0.2281652...],
                    [ 600.      ,  0.2751726...],
                    [ 610.      ,  0.3141208...],
                    [ 620.      ,  0.3434564...],
                    [ 630.      ,  0.3636521...],
                    [ 640.      ,  0.3765182...],
                    [ 650.      ,  0.3841561...],
                    [ 660.      ,  0.3884648...],
                    [ 670.      ,  0.3906975...],
                    [ 680.      ,  0.3918679...],
                    [ 690.      ,  0.3924590...],
                    [ 700.      ,  0.3927439...],
                    [ 710.      ,  0.3928570...],
                    [ 720.      ,  0.3928867...],
                    [ 730.      ,  0.3929099...],
                    [ 740.      ,  0.3928997...],
                    [ 750.      ,  0.3928827...],
                    [ 760.      ,  0.3928579...],
                    [ 770.      ,  0.3927857...],
                    [ 780.      ,  0.3927272...],
                    [ 790.      ,  0.3926867...],
                    [ 800.      ,  0.3926441...],
                    [ 810.      ,  0.3926385...],
                    [ 820.      ,  0.3926247...],
                    [ 830.      ,  0.3926105...]],
                    interpolator=SpragueInterpolator,
```

(continues on next page)

(continued from previous page)

```

        interpolator_args={},
        extrapolator=Extrapolator,
        extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100 # doctest: +ELLIPSIS
array([ 0.2065817...,  0.1219754...,  0.0514131...])

```

Colour Temperature

- *Correlated Colour Temperature*
 - *Robertson (1968)*
 - *Krystek (1985)*
 - *Ohno (2013)*
 - *Hernandez-Andres, Lee and Romero (1999)*
 - *Kang, Moon, Hong, Lee, Cho and Kim (2002)*
 - *CIE Illuminant D Series*

Correlated Colour Temperature

colour

<code>CCT_to_uv(CCT_D_uv[, method])</code>	Returns the <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
<code>CCT_TO_UV_METHODS</code>	Supported correlated colour temperature T_{cp} to <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates computation methods.
<code>uv_to_CCT(uv[, method])</code>	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates using given method.
<code>UV_TO_CCT_METHODS</code>	Supported <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
<code>CCT_to_xy(CCT[, method])</code>	Returns the <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
<code>CCT_TO_XY_METHODS</code>	Supported correlated colour temperature T_{cp} to <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates computation methods.
<code>xy_to_CCT(xy[, method])</code>	Returns the correlated colour temperature T_{cp} from given <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates using given method.
<code>XY_TO_CCT_METHODS</code>	Supported <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

colour.CCT_to_uv

`colour.CCT_to_uv(CCT_D_uv, method='Ohno 2013', **kwargs)`

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp} , Δ_{uv} .
- **method** (unicode, optional) – {'Ohno 2013', 'Robertson 1968', 'Krystek 1985'}, Computation method.

Other Parameters **cmfs** (*XYZ_ColourMatchingFunctions*, optional) – {`colour.temperature.CCT_to_uv_Ohno2013()`}, Standard observer colour matching functions.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[AdobeSystems13a], [AdobeSystems13b], [Kry85], [Ohn14], [WS00e]

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> CCT_D_uv = np.array([6507.47380460, 0.00322335])
>>> CCT_to_uv(CCT_D_uv, cmfs=cmfs) # doctest: +ELLIPSIS
array([ 0.1977999...,  0.3121999...])
```

colour.CCT_TO_UV_METHODS

`colour.CCT_TO_UV_METHODS = CaseInsensitiveMapping({'Ohno 2013': ..., 'Robertson 1968': ..., 'Krystek 1985': ...})`
Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.

References

[AdobeSystems13a], [AdobeSystems13b], [Kry85], [Ohn14], [WS00e]

CCT_TO_UV_METHODS [CaseInsensitiveMapping] {'Ohno 2013', 'Robertson 1968', 'Krystek 1985'}

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.uv_to_CCT

`colour.uv_to_CCT(uv, method='Ohno 2013', **kwargs)`

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.

Parameters

- **uv** (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **method** (unicode, optional) – {'Ohno 2013', 'Robertson 1968'}, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions*, optional) – {`colour.temperature.uv_to_CCT_Ohno2013()`}, Standard observer colour matching functions.
- **start** (*numeric*, optional) – {`colour.temperature.uv_to_CCT_Ohno2013()`}, Temperature range start in kelvins.
- **end** (*numeric*, optional) – {`colour.temperature.uv_to_CCT_Ohno2013()`}, Temperature range end in kelvins.
- **count** (*int*, optional) – {`colour.temperature.uv_to_CCT_Ohno2013()`}, Temperatures count in the planckian tables.
- **iterations** (*int*, optional) – {`colour.temperature.uv_to_CCT_Ohno2013()`}, Number of planckian tables to generate.

Returns Correlated colour temperature T_{cp} , Δ_{uv} .

Return type ndarray

References

[AdobeSystems13a], [AdobeSystems13b], [Ohn14], [WS00e]

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT(uv, cmfs=cmfs) # doctest: +ELLIPSIS
array([ 6.5074738...e+03,  3.2233461...e-03])
```

colour.UV_TO_CCT_METHODS

`colour.UV_TO_CCT_METHODS = CaseInsensitiveMapping({'Ohno 2013': ..., 'Robertson 1968': ..., 'ohno2013': ...})`

Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[AdobeSystems13a], [AdobeSystems13b], [Ohn14], [WS00e]

UV_TO_CCT_METHODS [CaseInsensitiveMapping] {'Ohno 2013', 'Robertson 1968'}

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.CCT_to_xy

`colour.CCT_to_xy(CCT, method='Kang 2002')`

Returns the *CIE XYZ* tristimulus values *xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp} .
- **method** (unicode, optional) – {'Kang 2002', 'CIE Illuminant D Series'}, Computation method.

Returns *xy* chromaticity coordinates.

Return type ndarray

References

[KMH+02], [Wik01b], [WS00d]

colour.CCT_TO_XY_METHODS

`colour.CCT_TO_XY_METHODS = CaseInsensitiveMapping({'Kang 2002': ..., 'CIE Illuminant D Series': ..., 'kang2`

Supported correlated colour temperature T_{cp} to *CIE XYZ* tristimulus values *xy* chromaticity coordinates computation methods.

References

[KMH+02], [Wik01b], [WS00d]

CCT_TO_XY_METHODS [CaseInsensitiveMapping] {'Kang 2002', 'CIE Illuminant D Series'}

Aliases:

- 'kang2002': 'Kang 2002'
- 'cie_d': 'Hernandez 1999'

colour.xy_to_CCT

`colour.xy_to_CCT(xy, method='McCamy 1992')`

Returns the correlated colour temperature T_{cp} from given *CIE XYZ* tristimulus values *xy* chromaticity coordinates using given method.

Parameters

- **xy** (array_like) – *xy* chromaticity coordinates.

- **method** (unicode, optional) – {'McCamy 1992', 'Hernandez 1999'}, Computation method.

Returns Correlated colour temperature T_{cp} .

Return type numeric or ndarray

References

[HernandezAndresLR99], [Wik01a], [Wik01b]

colour.XY_TO_CCT_METHODS

`colour.XY_TO_CCT_METHODS = CaseInsensitiveMapping({'McCamy 1992': ..., 'Hernandez 1999': ..., 'mccamy1992': ...})`
Supported *CIE XYZ* tristimulus values *xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[HernandezAndresLR99], [Wik01a], [Wik01b]

XY_TO_CCT_METHODS [CaseInsensitiveMapping] {'McCamy 1992', 'Hernandez 1999'}

Aliases:

- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

Robertson (1968)

`colour.temperature`

<code>CCT_to_uv_Robertson1968(CCT_D_uv)</code>	Returns the <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using <i>Roberston (1968)</i> method.
<code>uv_to_CCT_Robertson1968(uv)</code>	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates using <i>Roberston (1968)</i> method.

colour.temperature.CCT_to_uv_Robertson1968

`colour.temperature.CCT_to_uv_Robertson1968(CCT_D_uv)`

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

Parameters **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp} , Δ_{uv} .

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[AdobeSystems13b], [WS00e]

Examples

```
>>> CCT_D_uv = np.array([6500.0081378199056, 0.008333331244225])
>>> CCT_to_uv_Robertson1968(CCT_D_uv) # doctest: +ELLIPSIS
array([ 0.1937413..., 0.3152210...])
```

colour.temperature.uv_to_CCT_Robertson1968

colour.temperature.uv_to_CCT_Robertson1968(*uv*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.

Parameters *uv* (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.

Returns Correlated colour temperature T_{cp} , Δ_{uv} .

Return type ndarray

References

[AdobeSystems13a], [WS00e]

Examples

```
>>> uv = np.array([0.193741375998230, 0.315221043940594])
>>> uv_to_CCT_Robertson1968(uv) # doctest: +ELLIPSIS
array([ 6.5000162...e+03, 8.3333289...e-03])
```

Krystek (1985)

colour.temperature

CCT_to_uv_Krystek1985(CCT)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

colour.temperature.CCT_to_uv_Krystek1985

colour.temperature.CCT_to_uv_Krystek1985(*CCT*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

Parameters *CCT* (numeric) – Correlated colour temperature T_{cp} .

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[Kry85]

Examples

```
>>> CCT_to_uv_Krystek1985(6504.38938305) # doctest: +ELLIPSIS
array([ 0.1837669...,  0.3093443...])
```

Ohno (2013)

colour.temperature

<code>CCT_to_uv_Ohno2013(CCT_D_uv[, cmfs])</code>	Returns the <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates from given correlated colour temperature T_{cp} , Δ_{uv} and colour matching functions using <i>Ohno (2013)</i> method.
<code>uv_to_CCT_Ohno2013(uv[, cmfs, start, end, ...])</code>	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given <i>CIE UCS</i> colourspace <i>uv</i> chromaticity coordinates, colour matching functions and temperature range using <i>Ohno (2013)</i> method.

colour.temperature.CCT_to_uv_Ohno2013

colour.temperature.CCT_to_uv_Ohno2013(*CCT_D_uv*, *cmfs*=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...))

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} , Δ_{uv} and colour matching functions using *Ohno (2013)* method.

Parameters

- **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp} , Δ_{uv} .
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[Ohn14]

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs) # doctest: +ELLIPSIS
array([ 0.1977999..., 0.3122004...])
```

colour.temperature.uv_to_CCT_Ohno2013

colour.temperature.uv_to_CCT_Ohno2013(*uv*, *cmfs*=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), *start*=1000, *end*=100000, *count*=10, *iterations*=6)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given CIE UCS colour space *uv* chromaticity coordinates, colour matching functions and temperature range using *Ohno (2013)* method.

The iterations parameter defines the calculations precision: The higher its value, the more planckian tables will be generated through cascade expansion in order to converge to the exact solution.

Parameters

- **uv** (array_like) – CIE UCS colour space *uv* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **start** (numeric, optional) – Temperature range start in kelvins.
- **end** (numeric, optional) – Temperature range end in kelvins.
- **count** (int, optional) – Temperatures count in the planckian tables.
- **iterations** (int, optional) – Number of planckian tables to generate.

Returns Correlated colour temperature T_{cp} , Δ_{uv} .

Return type ndarray

References

[Ohn14]

Examples

```
>>> from colour import STANDARD_OBSERVERS_CMFS
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT_Ohno2013(uv, cmfs) # doctest: +ELLIPSIS
array([ 6.5074738...e+03, 3.2233461...e-03])
```

Hernandez-Andres, Lee and Romero (1999)

colour.temperature

<code>xy_to_CCT_Hernandez1999(xy)</code>	Returns the correlated colour temperature T_{cp} from given <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates using <i>Hernandez-Andres et al.(1999)</i> method..
--	--

`colour.temperature.xy_to_CCT_Hernandez1999`

`colour.temperature.xy_to_CCT_Hernandez1999(xy)`
Returns the correlated colour temperature T_{cp} from given *CIE XYZ* tristimulus values *xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.

Parameters *xy* (array_like) – *xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp} .

Return type numeric

References

[HernandezAndresLR99]

Examples

```
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_Hernandez1999(xy) # doctest: +ELLIPSIS
6500.7420431...
```

Kang, Moon, Hong, Lee, Cho and Kim (2002)

`colour.temperature`

<code>CCT_to_xy_Kang2002(CCT)</code>	Returns the <i>CIE XYZ</i> tristimulus values <i>xy</i> chromaticity coordinates from given correlated colour temperature T_{cp} using <i>Kang et al.(2002)</i> method..
--------------------------------------	--

`colour.temperature.CCT_to_xy_Kang2002`

`colour.temperature.CCT_to_xy_Kang2002(CCT)`
Returns the *CIE XYZ* tristimulus values *xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al. (2002)* method.

Parameters *CCT* (numeric or array_like) – Correlated colour temperature T_{cp} .

Returns *xy* chromaticity coordinates.

Return type ndarray

Raises `ValueError` – If the correlated colour temperature is not in appropriate domain.

References

[KMH+02]

Examples

```
>>> CCT_to_xy_Kang2002(6504.38938305) # doctest: +ELLIPSIS
array([ 0.313426 ..., 0.3235959...])
```

CIE Illuminant D Series

colour.temperature

CCT_to_xy_CIE_D(CCT)	Converts from the correlated colour temperature T_{cp} of a <i>CIE Illuminant D Series</i> to the chromaticity of that <i>CIE Illuminant D Series</i> illuminant.
----------------------	---

colour.temperature.CCT_to_xy_CIE_D

colour.temperature.CCT_to_xy_CIE_D(CCT)

Converts from the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* to the chromaticity of that *CIE Illuminant D Series* illuminant.

Parameters CCT (numeric or array_like) – Correlated colour temperature T_{cp} .

Returns xy chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

[WS00d]

Examples

```
>>> CCT_to_xy_CIE_D(6504.38938305) # doctest: +ELLIPSIS
array([ 0.3127077..., 0.3291128...])
```

Utilities

- *Common*
- *Array*
- *Metrics*

- *Data Structures*
- *Verbose*

Common

colour

<code>domain_range_scale(scale)</code>	A context manager and decorator temporarily setting <i>Colour</i> domain-range scale.
<code>get_domain_range_scale()</code>	Returns the current <i>Colour</i> domain-range scale.
<code>set_domain_range_scale([scale])</code>	Sets the current <i>Colour</i> domain-range scale.

colour.domain_range_scale

class `colour.domain_range_scale(scale)`

A context manager and decorator temporarily setting *Colour* domain-range scale. The following scales are available:

- **‘Reference’**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc. . .
- **‘1’**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters `scale` (unicode) – {‘Reference’, ‘1’}, *Colour* domain-range scale to set.

`__init__(scale)`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__(scale)</code>	Initialize self.
------------------------------	------------------

colour.get_domain_range_scale

`colour.get_domain_range_scale()`

Returns the current *Colour* domain-range scale. The following scales are available:

- **‘Reference’**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc. . .
- **‘1’**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Returns *Colour* domain-range scale.

Return type unicode

colour.set_domain_range_scale

`colour.set_domain_range_scale(scale='Reference')`

Sets the current *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc. . .
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters `scale` (unicode or int) – {'Reference', '1'}, *Colour* domain-range scale to set.

colour.utilities

<code>handle_numpy_errors(**kwargs)</code>	Decorator for handling <i>Numpy</i> errors.
<code>ignore_numpy_errors(function)</code>	Wrapper for given function.
<code>raise_numpy_errors(function)</code>	Wrapper for given function.
<code>print_numpy_errors(function)</code>	Wrapper for given function.
<code>warn_numpy_errors(function)</code>	Wrapper for given function.
<code>ignore_python_warnings(function)</code>	Decorator for ignoring <i>Python</i> warnings.
<code>batch(iterable[, k])</code>	Returns a batch generator from given iterable.
<code>disable_multiprocessing</code>	A context manager and decorator temporarily disabling <i>Colour</i> multiprocessing.
<code>multiprocessing_pool(*args, **kwargs)</code>	A context manager providing a multiprocessing pool.
<code>is_openimageio_installed([raise_exception])</code>	Returns if <i>OpenImageIO</i> is installed and available.
<code>is_pandas_installed([raise_exception])</code>	Returns if <i>Pandas</i> is installed and available.
<code>is_iterable(a)</code>	Returns if given <i>a</i> variable is iterable.
<code>is_string(a)</code>	Returns if given <i>a</i> variable is a <i>string</i> like variable.
<code>is_numeric(a)</code>	Returns if given <i>a</i> variable is a number.
<code>is_integer(a)</code>	Returns if given <i>a</i> variable is an integer under given threshold.
<code>is_sibling(element, mapping)</code>	Returns whether given element type is present in given mapping types.
<code>filter_kwargs(function, **kwargs)</code>	Filters keyword arguments incompatible with the given function signature.
<code>filter_mapping(mapping, filterers[, ...])</code>	Filters given mapping with given filterers.
<code>first_item(a)</code>	Return the first item of an iterable.
<code>to_domain_1(a[, scale_factor, dtype])</code>	Scales given array <i>a</i> to domain '1'.
<code>to_domain_10(a[, scale_factor, dtype])</code>	Scales given array <i>a</i> to domain '10', used by <i>Munsell Renotation System</i> .
<code>to_domain_100(a[, scale_factor, dtype])</code>	Scales given array <i>a</i> to domain '100'.
<code>to_domain_degrees(a[, scale_factor, dtype])</code>	Scales given array <i>a</i> to degrees domain.
<code>to_domain_int(a[, bit_depth, dtype])</code>	Scales given array <i>a</i> to int domain.
<code>from_range_1(a[, scale_factor])</code>	Scales given array <i>a</i> from range '1'.
<code>from_range_10(a[, scale_factor])</code>	Scales given array <i>a</i> from range '10', used by <i>Munsell Renotation System</i> .
<code>from_range_100(a[, scale_factor])</code>	Scales given array <i>a</i> from range '100'.
<code>from_range_degrees(a[, scale_factor])</code>	Scales given array <i>a</i> from degrees range.
<code>from_range_int(a[, bit_depth, dtype])</code>	Scales given array <i>a</i> from int range.

`colour.utilities.handle_numpy_errors`

`colour.utilities.handle_numpy_errors(**kwargs)`

Decorator for handling *Numpy* errors.

Other Parameters `**kwargs` (*dict, optional*) – Keywords arguments.

Returns

Return type `object`

References

[KPK11]

Examples

```
>>> import numpy
>>> @handle_numpy_errors(all='ignore')
... def f():
...     1 / numpy.zeros(3)
>>> f()
```

`colour.utilities.ignore_numpy_errors`

`colour.utilities.ignore_numpy_errors(function)`

Wrapper for given function.

`colour.utilities.raise_numpy_errors`

`colour.utilities.raise_numpy_errors(function)`

Wrapper for given function.

`colour.utilities.print_numpy_errors`

`colour.utilities.print_numpy_errors(function)`

Wrapper for given function.

`colour.utilities.warn_numpy_errors`

`colour.utilities.warn_numpy_errors(function)`

Wrapper for given function.

`colour.utilities.ignore_python_warnings`

`colour.utilities.ignore_python_warnings(function)`

Decorator for ignoring *Python* warnings.

Parameters `function` (`object`) – Function to decorate.

Returns

Return type `object`

Examples

```
>>> @ignore_python_warnings
... def f():
...     warnings.warn('This is an ignored warning!')
>>> f()
```

colour.utilities.batch

colour.utilities.**batch**(iterable, k=3)

Returns a batch generator from given iterable.

Parameters

- **iterable** (iterable) – Iterable to create batches from.
- **k** (integer) – Batches size.

Returns Is *string_like* variable.

Return type `bool`

Examples

```
>>> batch(tuple(range(10))) # doctest: +ELLIPSIS
<generator object batch at 0x...>
```

colour.utilities.disable_multiprocessing

class colour.utilities.**disable_multiprocessing**

A context manager and decorator temporarily disabling *Colour* multiprocessing.

__init__()

Initialize self. See help(type(self)) for accurate signature.

colour.utilities.multiprocessing_pool

colour.utilities.**multiprocessing_pool**(*args, **kwargs)

A context manager providing a multiprocessing pool.

Other Parameters

- ***args** (list, optional) – Arguments.
- ****kwargs** (dict, optional) – Keywords arguments.

Examples

```
>>> from functools import partial
>>> def _add(a, b):
...     return a + b
>>> with multiprocessing_pool() as pool:
...     pool.map(partial(_add, b=2), range(10))
... # doctest: +SKIP
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

colour.utilities.is_openimageio_installed

colour.utilities.is_openimageio_installed(*raise_exception=False*)

Returns if *OpenImageIO* is installed and available.

Parameters *raise_exception* (*bool*) – Raise exception if *OpenImageIO* is unavailable.

Returns Is *OpenImageIO* installed.

Return type *bool*

Raises *ImportError* – If *OpenImageIO* is not installed.

colour.utilities.is_pandas_installed

colour.utilities.is_pandas_installed(*raise_exception=False*)

Returns if *Pandas* is installed and available.

Parameters *raise_exception* (*bool*) – Raise exception if *Pandas* is unavailable.

Returns Is *Pandas* installed.

Return type *bool*

Raises *ImportError* – If *Pandas* is not installed.

colour.utilities.is_iterable

colour.utilities.is_iterable(*a*)

Returns if given *a* variable is iterable.

Parameters *a* (*object*) – Variable to check the iterability.

Returns *a* variable iterability.

Return type *bool*

Examples

```
>>> is_iterable([1, 2, 3])
True
>>> is_iterable(1)
False
```

colour.utilities.is_string

colour.utilities.is_string(*a*)

Returns if given *a* variable is a *string* like variable.

Parameters *a* (*object*) – Data to test.

Returns Is *a* variable a *string* like variable.

Return type *bool*

Examples

```
>>> is_string("I'm a string!")
True
>>> is_string(["I'm a string!"])
False
```

colour.utilities.is_numeric

colour.utilities.is_numeric(*a*)

Returns if given *a* variable is a number.

Parameters *a* (*object*) – Variable to check.

Returns Is *a* variable a number.

Return type *bool*

Examples

```
>>> is_numeric(1)
True
>>> is_numeric((1,))
False
```

colour.utilities.is_integer

colour.utilities.is_integer(*a*)

Returns if given *a* variable is an integer under given threshold.

Parameters *a* (*object*) – Variable to check.

Returns Is *a* variable an integer.

Return type *bool*

Notes

- The determination threshold is defined by the colour.algebra.common.INTEGER_THRESHOLD attribute.

Examples

```
>>> is_integer(1)
True
>>> is_integer(1.01)
False
```

colour.utilities.is_sibling

`colour.utilities.is_sibling(element, mapping)`

Returns whether given element type is present in given mapping types.

Parameters

- **element** (`object`) – Element to check if its type is present in the mapping types.
- **mapping** (`dict`) – Mapping.

Returns Whether given element type is present in given mapping types.

Return type `bool`

colour.utilities.filter_kwargs

```
colour.utilities.filter_kwargs(function, **kwargs)
```

Filters keyword arguments incompatible with the given function signature.

Parameters **function** (callable) – Callable to filter the incompatible keyword arguments.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Filtered keyword arguments.

Return type `dict`

Examples

```
>>> def fn_a(a):
...     return a
>>> def fn_b(a, b=0):
...     return a, b
>>> def fn_c(a, b=0, c=0):
...     return a, b, c
>>> fn_a(1, **filter_kwargs(fn_a, b=2, c=3))
1
>>> fn_b(1, **filter_kwargs(fn_b, b=2, c=3))
(1, 2)
>>> fn_c(1, **filter_kwargs(fn_c, b=2, c=3))
(1, 2, 3)
```

colour.utilities.filter_mapping

```
colour.utilities.filter_mapping(mapping, filterers, anchors=True,
                               flags=<RegexFlag.IGNORECASE: 2>)
Filters given mapping with given filterers.
```

Parameters

- **mapping** (dict_like) – Mapping to filter.
- **filterers** (unicode or `object` or array_like) – Filterer pattern for given mapping elements or a list of filterers.
- **anchors** (bool, optional) – Whether to use Regex line anchors, i.e. `^` and `$` are added, surrounding the filterer pattern.
- **flags** (int, optional) – Regex flags.

Returns Filtered mapping elements.

Return type OrderedDict

Notes

- To honour the filterers ordering, the return value is an OrderedDict class instance.

Examples

```
>>> class Element(object):
...     pass
>>> mapping = {
...     'Element A': Element(),
...     'Element B': Element(),
...     'Element C': Element(),
...     'Not Element C': Element(),
... }
>>> # Doctests skip for Python 2.x compatibility.
>>> filter_mapping(mapping, '\w+\s+A') # doctest: +SKIP
{'Element A': <colour.utilities.common.Element object at 0x...>}
>>> # Doctests skip for Python 2.x compatibility.
>>> sorted(filter_mapping(mapping, 'Element.*')) # doctest: +SKIP
[u'Element A', u'Element B', u'Element C']
```

colour.utilities.first_item

`colour.utilities.first_item(a)`

Return the first item of an iterable.

Parameters `a` (`object`) – Iterable to get the first item from.

Returns

Return type `object`

Raises `StopIteration` – If the iterable is empty.

Examples

```
>>> a = range(10)
>>> first_item(a)
0
```

colour.utilities.to_domain_1

colour.utilities.to_domain_1(a, scale_factor=100, dtype=<class 'numpy.float64'>)

Scales given array *a* to domain '1'. The behaviour is as follows:

- If *Colour* domain-range scale is '**Reference**' or '**1**', the definition is almost entirely by-passed and will just conveniently convert array *a* to np.ndarray.
- If *Colour* domain-range scale is '**100**' (currently unsupported private value only used for unit tests), array *a* is divided by scale_factor, typically 100.

Parameters

- **a** (array_like) – *a* to scale to domain '1'.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain '1'.
- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns *a* scaled to domain '1'.

Return type ndarray

Examples

With *Colour* domain-range scale set to '**Reference**':

```
>>> with domain_range_scale('Reference'):
...     to_domain_1(1)
array(1.0)
```

With *Colour* domain-range scale set to '**1**':

```
>>> with domain_range_scale('1'):
...     to_domain_1(1)
array(1.0)
```

With *Colour* domain-range scale set to '**100**' (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_1(1)
array(0.01)
```

colour.utilities.to_domain_10

colour.utilities.to_domain_10(a, scale_factor=10, dtype=<class 'numpy.float64'>)

Scales given array *a* to domain '**10**', used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is '**Reference**', the definition is almost entirely by-passed and will just conveniently convert array *a* to np.ndarray.
- If *Colour* domain-range scale is '**1**', array *a* is multiplied by scale_factor, typically 10.
- If *Colour* domain-range scale is '**100**' (currently unsupported private value only used for unit tests), array *a* is divided by scale_factor, typically 10.

Parameters

- **a** (array_like) – *a* to scale to domain ‘10’.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain ‘10’.
- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns *a* scaled to domain ‘10’.

Return type ndarray

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     to_domain_10(1)
array(1.0)
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     to_domain_10(1)
array(10.0)
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_10(1)
array(0.1)
```

colour.utilities.to_domain_100

colour.utilities.to_domain_100(*a*, scale_factor=100, dtype=<class 'numpy.float64'>)

Scales given array *a* to domain ‘100’. The behaviour is as follows:

- If *Colour* domain-range scale is ‘Reference’ or ‘100’ (currently unsupported private value only used for unit tests), the definition is almost entirely by-passed and will just conveniently convert array *a* to np.ndarray.
- If *Colour* domain-range scale is ‘1’, array *a* is multiplied by scale_factor, typically 100.

Parameters

- **a** (array_like) – *a* to scale to domain ‘100’.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain ‘100’.
- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns *a* scaled to domain ‘100’.

Return type ndarray

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     to_domain_100(1)
array(1.0)
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     to_domain_100(1)
array(100.0)
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_100(1)
array(1.0)
```

colour.utilities.to_domain_degrees

`colour.utilities.to_domain_degrees(a, scale_factor=360, dtype=<class 'numpy.float64'>)`

Scales given array *a* to degrees domain. The behaviour is as follows:

- If *Colour* domain-range scale is ‘Reference’, the definition is almost entirely by-passed and will just conveniently convert array *a* to `np.ndarray`.
- If *Colour* domain-range scale is ‘1’, array *a* is multiplied by `scale_factor`, typically 360.
- If *Colour* domain-range scale is ‘100’ (currently unsupported private value only used for unit tests), array *a* is multiplied by `scale_factor / 100`, typically 360 / 100.

Parameters

- **a** (*array_like*) – *a* to scale to degrees domain.
- **scale_factor** (*numeric* or *array_like*, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to degrees domain.
- **dtype** (*object*, optional) – Data type used for the conversion to `np.ndarray`.

Returns *a* scaled to degrees domain.

Return type `ndarray`

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     to_domain_degrees(1)
array(1.0)
```

With *Colour* domain-range scale set to ‘1’:


```
>>> with domain_range_scale('1'):
...     to_domain_degrees(1)
array(360.0)
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_degrees(1)
array(3.6)
```

colour.utilities.to_domain_int

`colour.utilities.to_domain_int(a, bit_depth=8, dtype=<class 'numpy.float64'>)`

Scales given array *a* to int domain. The behaviour is as follows:

- If *Colour* domain-range scale is ‘**Reference**’, the definition is almost entirely by-passed and will just conveniently convert array *a* to `np.ndarray`.
- If *Colour* domain-range scale is ‘1’, array *a* is multiplied by $2^{bit_depth} - 1$.
- If *Colour* domain-range scale is ‘100’ (currently unsupported private value only used for unit tests), array *a* is multiplied by $2^{bit_depth} - 1$.

Parameters

- **a** (*array_like*) – *a* to scale to int domain.
- **bit_depth** (numeric or *array_like*, optional) – Bit depth, usually *int* but can be an *array_like* if some axis need different scaling to be brought to int domain.
- **dtype** (*object*, optional) – Data type used for the conversion to `np.ndarray`.

Returns *a* scaled to int domain.

Return type `ndarray`

Notes

- To avoid precision issues and rounding, the scaling is performed on floating-point numbers.

Examples

With *Colour* domain-range scale set to ‘**Reference**’:

```
>>> with domain_range_scale('Reference'):
...     to_domain_int(1)
array(1.0)
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     to_domain_int(1)
array(255.0)
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_int(1)
array(2.55)
```

colour.utilities.from_range_1

colour.utilities.from_range_1(*a*, scale_factor=100)

Scales given array *a* from range '1'. The behaviour is as follows:

- If *Colour* domain-range scale is '**Reference**' or '1', the definition is entirely by-passed.
- If *Colour* domain-range scale is '100' (currently unsupported private value only used for unit tests), array *a* is multiplied by *scale_factor*, typically 100.

Parameters

- **a** (array_like) – *a* to scale from range '1'.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range '1'.

Returns *a* scaled from range '1'.

Return type ndarray

Examples

With *Colour* domain-range scale set to '**Reference**':

```
>>> with domain_range_scale('Reference'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to '1':

```
>>> with domain_range_scale('1'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to '100' (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_1(1)
100
```

colour.utilities.from_range_10

colour.utilities.from_range_10(*a*, scale_factor=10)

Scales given array *a* from range '10', used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is '**Reference**', the definition is entirely by-passed.
- If *Colour* domain-range scale is '1', array *a* is divided by *scale_factor*, typically 10.

- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array *a* is multiplied by *scale_factor*, typically 10.

Parameters

- **a** (array_like) – *a* to scale from range **'10'**.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range **'10'**.

Returns *a* scaled from range **'10'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_10(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_10(1)
0.1
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_10(1)
10
```

colour.utilities.from_range_100

colour.utilities.**from_range_100**(*a*, *scale_factor*=100)

Scales given array *a* from range **'100'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'100'** (currently unsupported private value only used for unit tests), the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array *a* is divided by *scale_factor*, typically 100.

Parameters

- **a** (array_like) – *a* to scale from range **'100'**.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range **'100'**.

Returns *a* scaled from range **'100'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     from_range_100(1)
1
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     from_range_100(1)
0.01
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_100(1)
1
```

colour.utilities.from_range_degrees

colour.utilities.from_range_degrees(*a*, *scale_factor*=360)

Scales given array *a* from degrees range. The behaviour is as follows:

- If *Colour* domain-range scale is ‘Reference’, the definition is entirely by-passed.
- If *Colour* domain-range scale is ‘1’, array *a* is divided by *scale_factor*, typically 360.
- If *Colour* domain-range scale is ‘100’ (currently unsupported private value only used for unit tests), array *a* is divided by *scale_factor* / 100, typically 360 / 100.

Parameters

- **a** (array_like) – *a* to scale from degrees range.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from degrees range.

Returns *a* scaled from degrees range.

Return type ndarray

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     from_range_degrees(1)
1
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     from_range_degrees(1) # doctest: +ELLIPSIS
0.002777...
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_degrees(1) # doctest: +ELLIPSIS
0.277777...
```

colour.utilities.from_range_int

`colour.utilities.from_range_int(a, bit_depth=8, dtype=<class 'numpy.float64'>)`

Scales given array *a* from int range. The behaviour is as follows:

- If *Colour* domain-range scale is ‘Reference’, the definition is entirely by-passed.
- If *Colour* domain-range scale is ‘1’, array *a* is converted to `np.ndarray` and divided by $2^{bit_depth} - 1$.
- If *Colour* domain-range scale is ‘100’ (currently unsupported private value only used for unit tests), array *a* is converted to `np.ndarray` and divided by $2^{bit_depth} - 1$.

Parameters

- **a** (array_like) – *a* to scale from int range.
- **bit_depth** (numeric or array_like, optional) – Bit depth, usually *int* but can be an *array_like* if some axis need different scaling to be brought from int range.
- **dtype** (object, optional) – Data type used for the conversion to `np.ndarray`.

Returns *a* scaled from int range.

Return type `ndarray`

Notes

- To avoid precision issues and rounding, the scaling is performed on floating-point numbers.

Examples

With *Colour* domain-range scale set to ‘Reference’:

```
>>> with domain_range_scale('Reference'):
...     from_range_int(1)
1
```

With *Colour* domain-range scale set to ‘1’:

```
>>> with domain_range_scale('1'):
...     from_range_int(1) # doctest: +ELLIPSIS
array(0.0039215...)
```

With *Colour* domain-range scale set to ‘100’ (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_int(1) # doctest: +ELLIPSIS
array(0.3921568...)
```

Array

colour.utilities

<code>as_array(a[, dtype])</code>	Converts given <i>a</i> variable to <i>ndarray</i> with given type.
<code>as_int_array(a)</code>	Converts given <i>a</i> variable to <i>ndarray</i> using the type defined by <code>colour.constant.DEFAULT_INT_DTYPE</code> attribute.
<code>as_float_array(a)</code>	Converts given <i>a</i> variable to <i>ndarray</i> using the type defined by <code>colour.constant.DEFAULT_FLOAT_DTYPE</code> attribute.
<code>as_numeric(a[, dtype])</code>	Converts given <i>a</i> variable to <i>numeric</i> .
<code>as_int(a)</code>	Converts given <i>a</i> variable to <i>numeric</i> using the type defined by <code>colour.constant.DEFAULT_INT_DTYPE</code> attribute.
<code>as_float(a)</code>	Converts given <i>a</i> variable to <i>numeric</i> using the type defined by <code>colour.constant.DEFAULT_FLOAT_DTYPE</code> attribute.
<code>as_namedtuple(a, named_tuple)</code>	Converts given <i>a</i> variable to given <i>namedtuple</i> class instance.
<code>closest_indexes(a, b)</code>	Returns the <i>a</i> variable closest element indexes to reference <i>b</i> variable elements.
<code>closest(a, b)</code>	Returns the <i>a</i> variable closest elements to reference <i>b</i> variable elements.
<code>normalise_maximum(a[, axis, factor, clip])</code>	Normalises given <i>array_like</i> <i>a</i> variable values by <i>a</i> variable maximum value and optionally clip them between.
<code>interval(distribution[, unique])</code>	Returns the interval size of given distribution.
<code>is_uniform(distribution)</code>	Returns if given distribution is uniform.
<code>in_array(a, b[, tolerance])</code>	Tests whether each element of an array is also present in a second array within given tolerance.
<code>tstack(a[, dtype])</code>	Stacks arrays in sequence along the last axis (tail).
<code>tsplit(a[, dtype])</code>	Splits arrays in sequence along the last axis (tail).
<code>row_as_diagonal(a)</code>	Returns the per row diagonal matrices of the given array.
<code>dot_vector(m, v)</code>	Convenient wrapper around <code>np.einsum()</code> with the following subscripts: <code>'...ij,...j->...i'</code> .
<code>dot_matrix(a, b)</code>	Convenient wrapper around <code>np.einsum()</code> with the following subscripts: <code>'...ij,...jk->...ik'</code> .
<code>orient(a, orientation)</code>	Orient given array according to given orientation value.
<code>centroid(a)</code>	Computes the centroid indexes of given <i>a</i> array.
<code>linear_conversion(a, old_range, new_range)</code>	Performs a simple linear conversion of given array between the old and new ranges.
<code>lerp(a, b, c)</code>	Performs a simple linear interpolation between given array <i>a</i> and array <i>b</i> using <i>c</i> value.
<code>fill_nan(a[, method, default])</code>	Fills given array NaNs according to given method.
<code>ndarray_write(a)</code>	A context manager setting given array writeable to perform an operation and then read-only.

colour.utilities.as_array

`colour.utilities.as_array(a, dtype=<class 'numpy.float64'>)`
 Converts given *a* variable to *ndarray* with given type.

Parameters

- **a** (*object*) – Variable to convert.
- **dtype** (*object*) – Type to use for conversion.

Returns *a* variable converted to *ndarray*.

Return type *ndarray*

Examples

```
>>> as_array([1, 2, 3])
array([ 1.,  2.,  3.])
>>> as_array([1, 2, 3], dtype=DEFAULT_INT_DTYPE)
array([1, 2, 3])
```

colour.utilities.as_int_array

`colour.utilities.as_int_array(a)`
 Converts given *a* variable to *ndarray* using the type defined by `colour.constant.DEFAULT_INT_DTYPE` attribute.

Parameters **a** (*object*) – Variable to convert.

Returns *a* variable converted to *ndarray*.

Return type *ndarray*

Examples

```
>>> as_int_array([1.0, 2.0, 3.0])
array([1, 2, 3])
```

colour.utilities.as_float_array

`colour.utilities.as_float_array(a)`
 Converts given *a* variable to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Parameters **a** (*object*) – Variable to convert.

Returns *a* variable converted to *ndarray*.

Return type *ndarray*

Examples

```
>>> as_float_array([1, 2, 3])
array([ 1.,  2.,  3.]
```

colour.utilities.as_numeric

colour.utilities.as_numeric(*a*, dtype=<class 'numpy.float64'>)

Converts given *a* variable to *numeric*. In the event where *a* cannot be converted, it is passed as is.

Parameters

- **a** (*object*) – Variable to convert.
- **dtype** (*object*) – Type to use for conversion.

Returns *a* variable converted to *numeric*.

Return type ndarray

Examples

```
>>> as_numeric(np.array([1]))
1.0
>>> as_numeric(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.]
```

colour.utilities.as_int

colour.utilities.as_int(*a*)

Converts given *a* variable to *numeric* using the type defined by colour.constant.DEFAULT_INT_DTYPE attribute. In the event where *a* cannot be converted, it is converted to *ndarray* using the type defined by colour.constant.DEFAULT_INT_DTYPE attribute.

Parameters **a** (*object*) – Variable to convert.

Returns *a* variable converted to *numeric*.

Return type ndarray

Warning: The behaviour of this definition is different than colour.utilities.as_numeric() definition when it comes to conversion failure: the former will forcibly convert *a* variable to *ndarray* using the type defined by colour.constant.DEFAULT_INT_DTYPE attribute while the later will pass the *a* variable as is.

Examples

```
>>> as_int(np.array([1]))
1
>>> as_int(np.arange(10))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```


colour.utilities.as_float

colour.utilities.**as_float**(*a*)

Converts given *a* variable to *numeric* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute. In the event where *a* cannot be converted, it is converted to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Parameters *a* (`object`) – Variable to convert.

Returns *a* variable converted to *numeric*.

Return type ndarray

Warning: The behaviour of this definition is different than `colour.utilities.as_numeric()` definition when it comes to conversion failure: the former will forcibly convert *a* variable to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute while the later will pass the *a* variable as is.

Examples

```
>>> as_float(np.array([1]))
1.0
>>> as_float(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.]
```

colour.utilities.as_namedtuple

colour.utilities.**as_namedtuple**(*a*, *named_tuple*)

Converts given *a* variable to given *namedtuple* class instance.

a can be either a *Numpy* structured array, a *namedtuple*, a *mapping*, or an *array_like* object. The definition will attempt to convert it to given *namedtuple*.

Parameters

- *a* (`object`) – Variable to convert.
- *named_tuple* (`namedtuple`) – *namedtuple* class.

Returns *a* variable converted to *namedtuple*.

Return type namedtuple

Examples

```
>>> from collections import namedtuple
>>> a_a = 1
>>> a_b = 2
>>> a_c = 3
>>> NamedTuple = namedtuple('NamedTuple', 'a b c')
>>> as_namedtuple(NamedTuple(a=1, b=2, c=3), NamedTuple)
NamedTuple(a=1, b=2, c=3)
>>> as_namedtuple({'a': a_a, 'b': a_b, 'c': a_c}, NamedTuple)
```

(continues on next page)

(continued from previous page)

```
NamedTuple(a=1, b=2, c=3)
>>> as_namedtuple([a_a, a_b, a_c], NamedTuple)
NamedTuple(a=1, b=2, c=3)
```

colour.utilities.closest_indexes

colour.utilities.**closest_indexes**(*a*, *b*)

Returns the *a* variable closest element indexes to reference *b* variable elements.

Parameters

- **a** (array_like) – Variable to search for the closest element indexes.
- **b** (numeric) – Reference variable.

Returns Closest *a* variable element indexes.

Return type numeric

Examples

```
>>> a = np.array([24.31357115, 63.62396289, 55.71528816,
...              62.70988028, 46.84480573, 25.40026416])
>>> print(closest_indexes(a, 63))
[3]
>>> print(closest_indexes(a, [63, 25]))
[3 5]
```

colour.utilities.closest

colour.utilities.**closest**(*a*, *b*)

Returns the *a* variable closest elements to reference *b* variable elements.

Parameters

- **a** (array_like) – Variable to search for the closest elements.
- **b** (numeric) – Reference variable.

Returns Closest *a* variable elements.

Return type numeric

Examples

```
>>> a = np.array([24.31357115, 63.62396289, 55.71528816,
...              62.70988028, 46.84480573, 25.40026416])
>>> closest(a, 63)
array([ 62.70988028])
>>> closest(a, [63, 25])
array([ 62.70988028,  25.40026416])
```

colour.utilities.normalise_maximum

`colour.utilities.normalise_maximum(a, axis=None, factor=1, clip=True)`

Normalises given *array_like* *a* variable values by *a* variable maximum value and optionally clip them between.

Parameters

- **a** (*array_like*) – *a* variable to normalise.
- **axis** (numeric, optional) – Normalization axis.
- **factor** (numeric, optional) – Normalization factor.
- **clip** (*bool*, optional) – Clip values to domain [0, 'factor'].

Returns Maximum normalised *a* variable.

Return type ndarray

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> normalise_maximum(a) # doctest: +ELLIPSIS
array([ 1.          ,  0.6564384...,  0.4576822...])
```

colour.utilities.interval

`colour.utilities.interval(distribution, unique=True)`

Returns the interval size of given distribution.

Parameters

- **distribution** (*array_like*) – Distribution to retrieve the interval.
- **unique** (*bool*, optional) – Whether to return unique intervals if the distribution is non-uniformly spaced or the complete intervals

Returns Distribution interval.

Return type ndarray

Examples

Uniformly spaced variable:

```
>>> y = np.array([1, 2, 3, 4, 5])
>>> interval(y)
array([ 1.])
>>> interval(y, False)
array([ 1.,  1.,  1.,  1.])
```

Non-uniformly spaced variable:

```
>>> y = np.array([1, 2, 3, 4, 8])
>>> interval(y)
array([ 1.,  4.])
>>> interval(y, False)
array([ 1.,  1.,  1.,  4.])
```

colour.utilities.is_uniform

colour.utilities.is_uniform(*distribution*)

Returns if given distribution is uniform.

Parameters *distribution* (array_like) – Distribution to check for uniformity.

Returns Is distribution uniform.

Return type bool

Examples

Uniformly spaced variable:

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> is_uniform(a)
True
```

Non-uniformly spaced variable:

```
>>> a = np.array([1, 2, 3.1415, 4, 5])
>>> is_uniform(a)
False
```

colour.utilities.in_array

colour.utilities.in_array(*a*, *b*, *tolerance*=2.2204460492503131e-16)

Tests whether each element of an array is also present in a second array within given tolerance.

Parameters

- **a** (array_like) – Array to test the elements from.
- **b** (array_like) – The values against which to test each value of array *a*.
- **tolerance** (numeric, optional) – Tolerance value.

Returns A boolean array with *a* shape describing whether an element of *a* is present in *b* within given tolerance.

Return type ndarray

References

[Yor14]

Examples

```
>>> a = np.array([0.50, 0.60])
>>> b = np.linspace(0, 10, 101)
>>> np.in1d(a, b)
array([ True, False], dtype=bool)
>>> in_array(a, b)
array([ True,  True], dtype=bool)
```

colour.utilities.tstack

colour.utilities.**tstack**(a, dtype=<class 'numpy.float64'>)

Stacks arrays in sequence along the last axis (tail).

Rebuilds arrays divided by colour.utilities.tsplit().

Parameters

- **a** (array_like) – Array to perform the stacking.
- **dtype** (object) – Type to use for initial conversion to ndarray.

Returns

Return type ndarray

Examples

```
>>> a = 0
>>> tstack([a, a, a])
array([ 0.,  0.,  0.])
>>> a = np.arange(0, 6)
>>> tstack([a, a, a])
array([[ 0.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 2.,  2.,  2.],
       [ 3.,  3.,  3.],
       [ 4.,  4.,  4.],
       [ 5.,  5.,  5.]])
>>> a = np.reshape(a, (1, 6))
>>> tstack([a, a, a])
array([[[ 0.,  0.,  0.],
        [ 1.,  1.,  1.],
        [ 2.,  2.,  2.],
        [ 3.,  3.,  3.],
        [ 4.,  4.,  4.],
        [ 5.,  5.,  5.]]])
>>> a = np.reshape(a, (1, 1, 6))
>>> tstack([a, a, a])
array([[[[ 0.,  0.,  0.],
          [ 1.,  1.,  1.],
          [ 2.,  2.,  2.],
          [ 3.,  3.,  3.],
          [ 4.,  4.,  4.],
          [ 5.,  5.,  5.]]]])
```

colour.utilities.tsplit

colour.utilities.**tsplit**(a, dtype=<class 'numpy.float64'>)

Splits arrays in sequence along the last axis (tail).

Parameters

- **a** (array_like) – Array to perform the splitting.
- **dtype** (object) – Type to use for initial conversion to ndarray.

Returns

Return type ndarray

Examples

```
>>> a = np.array([0, 0, 0])
>>> tsplit(a)
array([ 0., 0., 0.])
>>> a = np.array(
...     [[0, 0, 0],
...      [1, 1, 1],
...      [2, 2, 2],
...      [3, 3, 3],
...      [4, 4, 4],
...      [5, 5, 5]]
... )
>>> tsplit(a)
array([[ 0., 1., 2., 3., 4., 5.],
       [ 0., 1., 2., 3., 4., 5.],
       [ 0., 1., 2., 3., 4., 5.]])
>>> a = np.array(
...     [[[0, 0, 0],
...       [1, 1, 1],
...       [2, 2, 2],
...       [3, 3, 3],
...       [4, 4, 4],
...       [5, 5, 5]]]
... )
>>> tsplit(a)
array([[[ 0., 1., 2., 3., 4., 5.],
<BLANKLINE>
       [[ 0., 1., 2., 3., 4., 5.],
<BLANKLINE>
       [[ 0., 1., 2., 3., 4., 5.]])])
```

colour.utilities.row_as_diagonal

colour.utilities.**row_as_diagonal**(a)

Returns the per row diagonal matrices of the given array.

Parameters **a** (array_like) – Array to perform the diagonal matrices computation.

Returns

Return type ndarray

References

[Cas14]

Examples

```
>>> a = np.array(
...     [[0.25891593, 0.07299478, 0.36586996],
...      [0.30851087, 0.37131459, 0.16274825],
...      [0.71061831, 0.67718718, 0.09562581],
...      [0.71588836, 0.76772047, 0.15476079],
...      [0.92985142, 0.22263399, 0.88027331]]
... )
>>> row_as_diagonal(a)
array([[ 0.25891593,  0.          ,  0.          ],
       [ 0.          ,  0.07299478,  0.          ],
       [ 0.          ,  0.          ,  0.36586996]],
<BLANKLINE>
       [[ 0.30851087,  0.          ,  0.          ],
       [ 0.          ,  0.37131459,  0.          ],
       [ 0.          ,  0.          ,  0.16274825]],
<BLANKLINE>
       [[ 0.71061831,  0.          ,  0.          ],
       [ 0.          ,  0.67718718,  0.          ],
       [ 0.          ,  0.          ,  0.09562581]],
<BLANKLINE>
       [[ 0.71588836,  0.          ,  0.          ],
       [ 0.          ,  0.76772047,  0.          ],
       [ 0.          ,  0.          ,  0.15476079]],
<BLANKLINE>
       [[ 0.92985142,  0.          ,  0.          ],
       [ 0.          ,  0.22263399,  0.          ],
       [ 0.          ,  0.          ,  0.88027331]]])
```

colour.utilities.dot_vector

colour.utilities.dot_vector(*m*, *v*)

Convenient wrapper around `np.einsum()` with the following subscripts: ‘...*ij*,...*j*->...*i*’.

It performs the dot product of two arrays where *m* parameter is expected to be an array of 3x3 matrices and parameter *v* an array of vectors.

Parameters

- **m** (array_like) – Array of 3x3 matrices.
- **v** (array_like) – Array of vectors.

Returns

Return type ndarray

Examples

```
>>> m = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> m = np.reshape(np.tile(m, (6, 1)), (6, 3, 3))
>>> v = np.array([0.20654008, 0.12197225, 0.05136952])
>>> v = np.tile(v, (6, 1))
>>> dot_vector(m, v) # doctest: +ELLIPSIS
array([[ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...]])
```

colour.utilities.dot_matrix

colour.utilities.dot_matrix(*a*, *b*)

Convenient wrapper around `np.einsum()` with the following subscripts: ‘...*ij*,...*jk*->...*ik*’.

It performs the dot product of two arrays where *a* parameter is expected to be an array of 3x3 matrices and parameter *b* another array of of 3x3 matrices.

Parameters

- **a** (array_like) – Array of 3x3 matrices.
- **b** (array_like) – Array of 3x3 matrices.

Returns

Return type ndarray

Examples

```
>>> a = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> a = np.reshape(np.tile(a, (6, 1)), (6, 3, 3))
>>> b = a
>>> dot_matrix(a, b) # doctest: +ELLIPSIS
array([[[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],
       <BLANKLINE>
        [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],
       <BLANKLINE>
        [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],
       <BLANKLINE>
        [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],
       <BLANKLINE>
        [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]]])
```

(continues on next page)

(continued from previous page)

```

[[ 0.2342420...,  1.0418482..., -0.2760903...],
 [-1.7099407...,  2.5793226...,  0.1306181...],
 [-0.0044203...,  0.0377490...,  0.9666713...]],
<BLANKLINE>
[[ 0.2342420...,  1.0418482..., -0.2760903...],
 [-1.7099407...,  2.5793226...,  0.1306181...],
 [-0.0044203...,  0.0377490...,  0.9666713...]],
<BLANKLINE>
[[ 0.2342420...,  1.0418482..., -0.2760903...],
 [-1.7099407...,  2.5793226...,  0.1306181...],
 [-0.0044203...,  0.0377490...,  0.9666713...]]])

```

colour.utilities.orient

colour.utilities.**orient**(*a*, *orientation*)

Orient given array according to given orientation value.

Parameters

- **a** (array_like) – Array to perform the orientation onto.
- **orientation** (unicode, optional) – {'Flip', 'Flop', '90 CW', '90 CCW', '180'} Orientation to perform.

Returns Oriented array.

Return type ndarray

Examples

```

>>> a = np.tile(np.arange(5), (5, 1))
>>> a
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
>>> orient(a, '90 CW')
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
>>> orient(a, 'Flip')
array([[4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0]])

```

colour.utilities.centroid

colour.utilities.**centroid**(*a*)

Computes the centroid indexes of given *a* array.

Parameters *a* (array_like) – *a* array to compute the centroid indexes.

Returns *a* array centroid indexes.

Return type ndarray

Examples

```
>>> a = np.tile(np.arange(0, 5), (5, 1))
>>> centroid(a)
array([2, 3])
```

colour.utilities.linear_conversion

colour.utilities.**linear_conversion**(*a*, *old_range*, *new_range*)

Performs a simple linear conversion of given array between the old and new ranges.

Parameters

- **a** (array_like) – Array to perform the linear conversion onto.
- **old_range** (array_like) – Old range.
- **new_range** (array_like) – New range.

Returns Linear conversion result.

Return type ndarray

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> linear_conversion(a, np.array([0, 1]), np.array([1, 10]))
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]
```

colour.utilities.lerp

colour.utilities.**lerp**(*a*, *b*, *c*)

Performs a simple linear interpolation between given array *a* and array *b* using *c* value.

Parameters

- **a** (array_like) – Array *a*, the start of the range in which to interpolate.
- **b** (array_like) – Array *b*, the end of the range in which to interpolate.
- **c** (array_like) – Array *c* value to use to interpolate between array *a* and array *b*.

Returns Linear interpolation result.

Return type ndarray

Examples

```
>>> a = 0
>>> b = 2
>>> lerp(a, b, 0.5)
1.0
```

colour.utilities.fill_nan

colour.utilities.**fill_nan**(a, method='Interpolation', default=0)

Fills given array NaNs according to given method.

Parameters

- **a** (array_like) – Array to fill the NaNs of.
- **method** (unicode) – {'Interpolation', 'Constant'}, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (numeric) – Value to use with the *Constant* method.

Returns NaNs filled array.

Return type ndarray

Examples

```
>>> a = np.array([0.1, 0.2, np.nan, 0.4, 0.5])
>>> fill_nan(a)
array([ 0.1, 0.2, 0.3, 0.4, 0.5])
>>> fill_nan(a, method='Constant')
array([ 0.1, 0.2, 0. , 0.4, 0.5])
```

colour.utilities.ndarray_write

colour.utilities.**ndarray_write**(a)

A context manager setting given array writeable to perform an operation and then read-only.

Parameters **a** (array_like) – Array to perform an operation.

Returns Array.

Return type ndarray

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> a.setflags(write=False)
>>> try:
...     a += 1
... except ValueError:
...     pass
>>> with ndarray_write(a):
...     a +=1
```

Metrics

`colour.utilities`

<code>metric_mse(a, b)</code>	Computes the mean squared error (MSE) or mean squared deviation (MSD) between given <i>array_like</i> <i>a</i> and <i>b</i> variables.
<code>metric_psnr(a, b[, max_a])</code>	Computes the peak signal-to-noise ratio (PSNR) between given <i>array_like</i> <i>a</i> and <i>b</i> variables.

`colour.utilities.metric_mse`

`colour.utilities.metric_mse(a, b)`

Computes the mean squared error (MSE) or mean squared deviation (MSD) between given *array_like* *a* and *b* variables.

Parameters

- **a** (*array_like*) – *a* variable.
- **b** (*array_like*) – *b* variable.

Returns Mean squared error (MSE).

Return type `float`

References

[Wik03d]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_mse(a, b) # doctest: +ELLIPSIS
0.0012714...
```

`colour.utilities.metric_psnr`

`colour.utilities.metric_psnr(a, b, max_a=1)`

Computes the peak signal-to-noise ratio (PSNR) between given *array_like* *a* and *b* variables.

Parameters

- **a** (*array_like*) – *a* variable.
- **b** (*array_like*) – *b* variable.
- **max_a** (numeric, optional) – Maximum possible pixel value of the *a* variable.

Returns Peak signal-to-noise ratio (PSNR).

Return type `float`

References

[Wik04a]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_psnr(a, b) # doctest: +ELLIPSIS
28.9568515...
```

Data Structures

colour.utilities

<code>CaseInsensitiveMapping([data])</code>	Implements a case-insensitive mutable mapping / <i>dict</i> object.
<code>Lookup</code>	Extends <i>dict</i> type to provide a lookup by value(s).
<code>Structure(*args, **kwargs)</code>	Defines an object similar to C/C++ structured type.

colour.utilities.CaseInsensitiveMapping

class colour.utilities.CaseInsensitiveMapping(*data=None, **kwargs*)

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters *data* (*dict*) – *dict* of data to store into the mapping at initialisation.

Other Parameters ***kwargs* (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

```
__setitem__()
__getitem__()
__delitem__()
__contains__()
__iter__()
__len__()
__eq__()
__ne__()
__repr__()
copy()
lower_items()
```

Warning: The keys are expected to be unicode or string-like objects.

References

[Rei]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

__init__(data=None, **kwargs)
Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__</code> ([data])	Initialize self.
<code>clear</code> ()	
<code>copy</code> ()	Returns a copy of the mapping.
<code>get</code> (k[,d])	
<code>items</code> ()	
<code>keys</code> ()	
<code>lower_items</code> ()	Iterates over the lower items names.
<code>pop</code> (k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised.
<code>popitem</code> ()	as a 2-tuple; but raise KeyError if D is empty.
<code>setdefault</code> (k[,d])	
<code>update</code> ([E,]**F)	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
<code>values</code> ()	

Attributes

data	Getter and setter property for the data.
------	--

colour.utilities.Lookup

class colour.utilities.Lookup
Extends dict type to provide a lookup by value(s).

keys_from_value()

first_key_from_value()

References

[[Mana](#)]

Examples

```
>>> person = Lookup(first_name='Doe', last_name='John', gender='male')
>>> person.first_key_from_value('Doe')
'first_name'
>>> persons = Lookup(John='Doe', Jane='Doe', Luke='Skywalker')
>>> sorted(persons.keys_from_value('Doe'))
['Jane', 'John']
```

`__init__()`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>clear()</code>	
<code>copy()</code>	
<code>first_key_from_value(value)</code>	Gets the first key with given value.
<code>fromkeys</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get</code>	Return the value for key if key is in the dictionary, else default.
<code>items()</code>	
<code>keys()</code>	
<code>keys_from_value(value)</code>	Gets the keys with given value.
<code>pop(k[,d])</code>	If key is not found, d is returned if given, otherwise <code>KeyError</code> is raised
<code>popitem()</code>	2-tuple; but raise <code>KeyError</code> if D is empty.
<code>setdefault</code>	Insert key with a value of default if key is not in the dictionary.
<code>update([E,]**F)</code>	If E is present and has a <code>.keys()</code> method, then does: for k in E: D[k] = E[k] If E is present and lacks a <code>.keys()</code> method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
<code>values()</code>	

`colour.utilities.Structure`

class `colour.utilities.Structure(*args, **kwargs)`

Defines an object similar to C/C++ structured type.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Key / Value pairs.

References

[Manb]

Examples

```
>>> person = Structure(first_name='Doe', last_name='John', gender='male')
>>> # Doctests skip for Python 2.x compatibility.
>>> person.first_name # doctest: +SKIP
'Doe'
>>> sorted(person.keys())
['first_name', 'gender', 'last_name']
>>> # Doctests skip for Python 2.x compatibility.
>>> person['gender'] # doctest: +SKIP
'male'
```

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

Methods

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>clear()</code>	
<code>copy()</code>	
<code>fromkeys</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get</code>	Return the value for key if key is in the dictionary, else default.
<code>items()</code>	
<code>keys()</code>	
<code>pop(k[,d])</code>	If key is not found, d is returned if given, otherwise <code>KeyError</code> is raised
<code>popitem()</code>	2-tuple; but raise <code>KeyError</code> if D is empty.
<code>setdefault</code>	Insert key with a value of default if key is not in the dictionary.
<code>update([E,]**F)</code>	If E is present and has a <code>.keys()</code> method, then does: for k in E: D[k] = E[k] If E is present and lacks a <code>.keys()</code> method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
<code>values()</code>	

Verbose

`colour.utilities`

<code>message_box(message[, width, padding, ...])</code>	Prints a message inside a box.
<code>warning(*args, **kwargs)</code>	Issues a warning.
<code>filter_warnings([state, colour_warnings, ...])</code>	Filters <i>Colour</i> and also optionally overall Python warnings.

Continued on next page

Table 285 – continued from previous page

<code>suppress_warnings([colour_warnings, ...])</code>	A context manager filtering <i>Colour</i> and also optionally overall Python warnings.
<code>numpy_print_options(*args, **kwargs)</code>	A context manager implementing context changes to <i>Numpy</i> print behaviour.
<code>describe_environment([runtime_packages, ...])</code>	Describes <i>Colour</i> running environment, i.e.

colour.utilities.message_box

`colour.utilities.message_box(message, width=79, padding=3, print_callable=<built-in function print>)`

Prints a message inside a box.

Parameters

- **message** (unicode) – Message to print.
- **width** (int, optional) – Message box width.
- **padding** (unicode, optional) – Padding on each sides of the message.
- **print_callable** (callable, optional) – Callable used to print the message box.

Returns Definition success.

Return type bool

Examples

```
>>> message = ('Lorem ipsum dolor sit amet, consectetur adipiscing elit, '
...           'sed do eiusmod tempor incididunt ut labore et dolore magna '
...           'aliqua.')
>>> message_box(message, width=75)
=====
*                                                                 *
*  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do  *
*  eiusmod tempor incididunt ut labore et dolore magna aliqua.      *
*                                                                 *
=====
True
>>> message_box(message, width=60)
=====
*                                                                 *
*  Lorem ipsum dolor sit amet, consectetur adipiscing                *
*  elit, sed do eiusmod tempor incididunt ut labore et              *
*  dolore magna aliqua.                                              *
*                                                                 *
=====
True
>>> message_box(message, width=75, padding=16)
=====
*                                                                 *
*           Lorem ipsum dolor sit amet, consectetur                *
*           adipiscing elit, sed do eiusmod tempor                  *
*           incididunt ut labore et dolore magna                    *
*           aliqua.                                                  *
*                                                                 *
=====
```

(continues on next page)

(continued from previous page)

```
=====
True
```

colour.utilities.warning

`colour.utilities.warning(*args, **kwargs)`

Issues a warning.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Definition success.

Return type `bool`

Examples

```
>>> warning('This is a warning!') # doctest: +SKIP
```

colour.utilities.filter_warnings

`colour.utilities.filter_warnings(state=True, colour_warnings=True, colour_runtime_warnings=False, colour_usage_warnings=False, python_warnings=False)`

Filters *Colour* and also optionally overall Python warnings.

Parameters

- **state** (`bool`, optional) – Warnings filter state.
- **colour_warnings** (`bool`, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings.
- **colour_runtime_warnings** (`bool`, optional) – Whether to filter *Colour* runtime warnings.
- **colour_usage_warnings** (`bool`, optional) – Whether to filter *Colour* usage warnings.
- **python_warnings** (`bool`, optional) – Whether to filter *Python* warnings.

Returns Definition success.

Return type `bool`

Examples

```
# Filtering Colour warnings: >>> filter_warnings() True
# Filtering Colour runtime warnings: >>> filter_warnings(colour_warnings=False,
colour_runtime_warnings=True) True
```

```
# Filtering Colour usage warnings: >>> filter_warnings(colour_warnings=False,
colour_usage_warnings=True) True

# Filtering Colour and also Python warnings: >>> filter_warnings(python_warnings=True) True
```

colour.utilities.suppress_warnings

```
colour.utilities.suppress_warnings(colour_warnings=True, colour_runtime_warnings=False,
colour_usage_warnings=False, python_warnings=False)
```

A context manager filtering *Colour* and also optionally overall Python warnings.

Parameters

- **colour_warnings** (*bool*, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings.
- **colour_runtime_warnings** (*bool*, optional) – Whether to filter *Colour* runtime warnings.
- **colour_usage_warnings** (*bool*, optional) – Whether to filter *Colour* usage warnings.
- **python_warnings** (*bool*, optional) – Whether to filter *Python* warnings.

colour.utilities.numpy_print_options

```
colour.utilities.numpy_print_options(*args, **kwargs)
```

A context manager implementing context changes to *Numpy* print behaviour.

Other Parameters

- ***args** (*list*, optional) – Arguments.
- ****kwargs** (*dict*, optional) – Keywords arguments.

Examples

```
>>> np.array([np.pi]) # doctest: +ELLIPSIS
array([ 3.1415926...])
>>> with numpy_print_options(formatter={'float': '{:0.1f}'.format}):
...     np.array([np.pi])
array([3.1])
```

colour.utilities.describe_environment

```
colour.utilities.describe_environment(runtime_packages=True, development_packages=False,
print_environment=True, **kwargs)
```

Describes *Colour* running environment, i.e. interpreter, runtime and development packages.

Parameters

- **runtime_packages** (*bool*, optional) – Whether to return the runtime packages versions.
- **development_packages** (*bool*, optional) – Whether to return the development packages versions.

- **print_environment** (*bool*, optional) – Whether to print the environment.

Other Parameters

- **padding** (*unicode*, optional) – {colour.utilities.message_box()}, Padding on each sides of the message.
- **print_callable** (*callable*, optional) – {colour.utilities.message_box()}, Callable used to print the message box.
- **width** (*int*, optional) – {colour.utilities.message_box()}, Message box width.

Returns Environment.

Return type defaultdict

Examples

```
>>> environment = describe_environment(width=75) # doctest: +SKIP
=====
*                                     *
* Interpreter :                       *
*   python : 2.7.14 | packaged by conda-forge | (default, Dec 25 *
*   2017, 01:18:54)                 *
*   [GCC 4.2.1 Compatible Apple LLVM 6.1.0 *
*   (clang-602.0.53)]                *
*                                     *
* colour-science.org :              *
*   colour : v0.3.11-323-g380c1838    *
*                                     *
* Runtime :                          *
*   numpy : 1.14.3                   *
*   scipy : 1.0.0                    *
*   pandas : 0.22.0                  *
*   matplotlib : 2.2.2               *
*   notebook : 5.4.0                *
*   ipywidgets : 7.2.1               *
*                                     *
=====
>>> environment = describe_environment(True, True, width=75)
... # doctest: +SKIP
=====
*                                     *
* Interpreter :                       *
*   python : 2.7.14 | packaged by conda-forge | (default, Dec 25 *
*   2017, 01:18:54)                 *
*   [GCC 4.2.1 Compatible Apple LLVM 6.1.0 *
*   (clang-602.0.53)]                *
*                                     *
* colour-science.org :              *
*   colour : v0.3.11-323-g380c1838    *
*                                     *
* Runtime :                          *
*   numpy : 1.14.3                   *
*   scipy : 1.0.0                    *
*   pandas : 0.22.0                  *
*   matplotlib : 2.2.2               *
*   notebook : 5.4.0                *
*                                     *
```

(continues on next page)

(continued from previous page)

```

*      ipywidgets : 7.2.1
*
*      Development :
*      coverage : 4.5.1
*      flake8 : 3.5.0
*      invoke : 0.22.1
*      mock : 2.0.0
*      nose : 1.3.7
*      restructuredtext_lint : 1.1.3
*      six : 1.11.0
*      sphinx : 1.7.5
*      sphinx_rtd_theme : 0.2.4
*      twine : 1.10.0
*      yapf : 0.20.2
*
=====

```

Ancillary Objects

`colour.utilities`

<code>ColourWarning</code>	This is the base class of <i>Colour</i> warnings.
<code>ColourUsageWarning</code>	This is the base class of <i>Colour</i> usage warnings.
<code>ColourRuntimeWarning</code>	This is the base class of <i>Colour</i> runtime warnings.

`colour.utilities.ColourWarning`

exception `colour.utilities.ColourWarning`

This is the base class of *Colour* warnings. It is a subclass of `Warning` class.

`colour.utilities.ColourUsageWarning`

exception `colour.utilities.ColourUsageWarning`

This is the base class of *Colour* usage warnings. It is a subclass of `colour.utilities.ColourWarning` class.

`colour.utilities.ColourRuntimeWarning`

exception `colour.utilities.ColourRuntimeWarning`

This is the base class of *Colour* runtime warnings. It is a subclass of `colour.utilities.ColourWarning` class.

Colour Volume

- *Optimal Colour Stimuli - MacAdam Limits*
- *Mesh Volume*
- *Pointer's Gamut*

- *RGB Volume*
- *Visible Spectrum*

Optimal Colour Stimuli - MacAdam Limits

colour

<code>is_within_macadam_limits(xyY, illuminant[, ...])</code>	Returns if given <i>CIE xyY</i> colourspace array is within MacAdam limits of given illuminant.
<code>ILLUMINANTS_OPTIMAL_COLOUR_STIMULI</code>	Illuminants <i>Optimal Colour Stimuli</i> .

colour.is_within_macadam_limits

`colour.is_within_macadam_limits(xyY, illuminant, tolerance=None)`

Returns if given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.

Parameters

- **xyY** (array_like) – *CIE xyY* colourspace array.
- **illuminant** (unicode) – Illuminant.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within MacAdam limits.

Return type `bool`

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Examples

```
>>> is_within_macadam_limits(np.array([0.3205, 0.4131, 0.51]), 'A')
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51],
...               [0.0005, 0.0031, 0.001]])
>>> is_within_macadam_limits(a, 'A')
array([ True, False], dtype=bool)
```

colour.ILLUMINANTS_OPTIMAL_COLOUR_STIMULI

`colour.ILLUMINANTS_OPTIMAL_COLOUR_STIMULI = CaseInsensitiveMapping({'A': ..., 'C': ..., 'D65': ...})`
Illuminants *Optimal Colour Stimuli*.

References

[Wik04b]

ILLUMINANTS_OPTIMAL_COLOUR_STIMULI [CaseInsensitiveMapping] {'A', 'C', 'D65'}

Mesh Volume

colour

<code>is_within_mesh_volume(points, mesh[, tolerance])</code>	Returns if given points are within given mesh volume using Delaunay triangulation.
---	--

colour.is_within_mesh_volume

`colour.is_within_mesh_volume(points, mesh, tolerance=None)`

Returns if given points are within given mesh volume using Delaunay triangulation.

Parameters

- **points** (array_like) – Points to check if they are within mesh volume.
- **mesh** (array_like) – Points of the volume used to generate the Delaunay triangulation.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within mesh volume.

Return type `bool`

Examples

```
>>> mesh = np.array(
...     [[-1.0, -1.0, 1.0],
...      [1.0, -1.0, 1.0],
...      [1.0, -1.0, -1.0],
...      [-1.0, -1.0, -1.0],
...      [0.0, 1.0, 0.0]]
... )
>>> is_within_mesh_volume(np.array([0.0005, 0.0031, 0.0010]), mesh)
array(True, dtype=bool)
>>> a = np.array([[0.0005, 0.0031, 0.0010],
...               [0.3205, 0.4131, 0.5100]])
>>> is_within_mesh_volume(a, mesh)
array([ True, False], dtype=bool)
```

Pointer's Gamut

colour

<code>is_within_pointer_gamut(XYZ[, tolerance])</code>	Returns if given <i>CIE XYZ</i> tristimulus values are within Pointer's Gamut volume.
--	---

colour.is_within_pointer_gamut

`colour.is_within_pointer_gamut(XYZ, tolerance=None)`

Returns if given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within Pointer's Gamut.

Return type `bool`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_pointer_gamut(np.array([0.3205, 0.4131, 0.5100]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.5100], [0.0005, 0.0031, 0.0010]])
>>> is_within_pointer_gamut(a)
array([ True, False], dtype=bool)
```

RGB Volume

colour

<code>RGB_colourspace_limits(colourspace[, illuminant])</code>	Computes given <i>RGB</i> colourspace volume limits in <i>Lab</i> colourspace.
<code>RGB_colourspace_pointer_gamut_coverage_MonteCarlo(s, g)</code>	Returns given <i>RGB</i> colourspace percentage coverage of Pointer's Gamut volume using <i>Monte Carlo</i> method.
<code>RGB_colourspace_visible_spectrum_coverage_MonteCarlo(s, g)</code>	Returns given <i>RGB</i> colourspace percentage coverage of visible spectrum volume using <i>Monte Carlo</i> method.
<code>RGB_colourspace_volume_MonteCarlo(colourspace)</code>	Performs given <i>RGB</i> colourspace volume computation using <i>Monte Carlo</i> method and multiprocessing.
<code>RGB_colourspace_volume_coverage_MonteCarlo(...)</code>	Returns given <i>RGB</i> colourspace percentage coverage of an arbitrary volume.

colour.RGB_colourspace_limits

`colour.RGB_colourspace_limits(colourspace, illuminant=array([ 0.3127, 0.329 ]))`

Computes given *RGB* colourspace volume limits in *Lab* colourspace.

Parameters

- **colourspace** (*RGB_Colourspace*) – *RGB* colourspace to compute the volume of.
- **illuminant** (array_like, optional) – *Lab* colourspace *illuminant* chromaticity coordinates.

Returns *RGB* colourspace volume limits.

Return type ndarray

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> RGB_colourspace_limits(sRGB) # doctest: +ELLIPSIS
array([[ 0. ..., 100. ...],
       [-86.182855 ..., 98.2563272...],
       [-107.8503557..., 94.4894974...]])
```

colour.RGB_colourspace_pointer_gamut_coverage_MonteCarlo

`colour.RGB_colourspace_pointer_gamut_coverage_MonteCarlo(colourspace, samples=10000000.0, random_generator=<function random_triplet_generator>, random_state=None)`

Returns given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.

Parameters

- **colourspace** (*RGB_Colourspace*) – *RGB* colourspace to compute the *Pointer's Gamut* coverage percentage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of *Pointer's Gamut* volume.

Return type float

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_pointer_gamut_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng) # doctest: +ELLIPSIS
81...
```

colour.RGB_colourspace_visible_spectrum_coverage_MonteCarlo

```
colour.RGB_colourspace_visible_spectrum_coverage_MonteCarlo(colourspace,          sam-
                                                             ples=10000000.0,      ran-
                                                             dom_generator=<function
                                                             random_triplet_generator>,
                                                             random_state=None)
```

Returns given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.

Parameters

- **colourspace** (*RGB_Colourspace*) – *RGB* colourspace to compute the visible spectrum coverage percentage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (*RandomState*, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of visible spectrum volume.

Return type *float*

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_visible_spectrum_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng) # doctest: +ELLIPSIS
47...
```

colour.RGB_colourspace_volume_MonteCarlo

```
colour.RGB_colourspace_volume_MonteCarlo(colourspace, samples=10000000.0, limits=array([[
    0., 100.], [-150., 150.], [-150., 150.]]), il-
luminant_Lab=array([ 0.3127, 0.329  ]),
chromatic_adaptation_method='CAT02',      ran-
dom_generator=<function random_triplet_generator>,
random_state=None)
```

Performs given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.

Parameters

- **colourspace** (*RGB_Colourspace*) – *RGB* colourspace to compute the volume of.
- **samples** (numeric, optional) – Samples count.
- **limits** (array_like, optional) – *Lab* colourspace volume.
- **illuminant_Lab** (array_like, optional) – *Lab* colourspace *illuminant* chromaticity coordinates.

- **chromatic_adaptation_method** (unicode, optional) – {'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}, *Chromatic adaptation method*.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples within the *Lab* colourspace volume.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns *RGB* colourspace volume.

Return type `float`

Notes

- The doctest is assuming that `np.random.RandomState()` definition will return the same sequence no matter which *OS* or *Python* version is used. There is however no formal promise about the *prng* sequence reproducibility of either *Python* or *Numpy* implementations: Laurent. (2012). Reproducibility of python pseudo-random numbers across systems and versions? Retrieved January 20, 2015, from <http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions>

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> from colour.utilities import disable_multiprocessing
>>> prng = np.random.RandomState(2)
>>> with disable_multiprocessing():
...     RGB_colourspace_volume_MonteCarlo(sRGB, 10e3, random_state=prng)
... # doctest: +ELLIPSIS
8...
```

`colour.RGB_colourspace_volume_coverage_MonteCarlo`

```
colour.RGB_colourspace_volume_coverage_MonteCarlo(colourspace, coverage_sampler,
                                                    samples=10000000.0,      ran-
                                                    dom_generator=<function      ran-
                                                    dom_triplet_generator>,      ran-
                                                    dom_state=None)
```

Returns given *RGB* colourspace percentage coverage of an arbitrary volume.

Parameters

- **colourspace** (`RGB_Colourspace`) – *RGB* colourspace to compute the volume coverage percentage.
- **coverage_sampler** (`object`) – Python object responsible for checking the volume coverage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of volume.

Return type `float`

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_volume_coverage_MonteCarlo(
...     sRGB, is_within_pointer_gamut, 10e3, random_state=prng)
... # doctest: +ELLIPSIS
81...
```

Visible Spectrum

colour

<code>is_within_visible_spectrum(XYZ[, interval, ...])</code>	Returns if given <i>CIE XYZ</i> tristimulus values are within visible spectrum volume / given colour matching functions volume.
---	---

colour.is_within_visible_spectrum

`colour.is_within_visible_spectrum(XYZ, interval=10, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralDistribution(name='1 Constant', ...), tolerance=None)`

Returns if given *CIE XYZ* tristimulus values are within visible spectrum volume / given colour matching functions volume.

Parameters

- **XYZ** (`array_like`) – *CIE XYZ* tristimulus values.
- **interval** (`int`, optional) – Wavelength λ_i range interval used to compute the pulse waves for the *CIE XYZ* colourspace outer surface.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.
- **illuminant** (`SpectralDistribution`, optional) – Illuminant spectral distribution.
- **tolerance** (`numeric`, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within visible spectrum.

Return type `bool`

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_visible_spectrum(np.array([0.3205, 0.4131, 0.51]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51],
...               [-0.0005, 0.0031, 0.001]])
>>> is_within_visible_spectrum(a)
array([ True, False], dtype=bool)
```

Ancillary Objects

colour.volume

<code>generate_pulse_waves(bins)</code>	Generates the pulse waves of given number of bins necessary to totally stimulate the colour matching functions.
<code>XYZ_outer_surface([interval, cmfs, illuminant])</code>	Generates the <i>CIE XYZ</i> colourspace outer surface for given colour matching functions using multi-spectral conversion of pulse waves to <i>CIE XYZ</i> tristimulus values.

colour.volume.generate_pulse_waves

colour.volume.**generate_pulse_waves**(bins)

Generates the pulse waves of given number of bins necessary to totally stimulate the colour matching functions.

Assuming 5 bins, a first set of SPDs would be as follows:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

The second one:

```
1 1 0 0 0
0 1 1 0 0
0 0 1 1 0
0 0 0 1 1
1 0 0 0 1
```

The third:

```
1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 1
```

Etc...

Parameters bins (*int*) – Number of bins of the pulse waves.

Returns Pulse waves.

Return type ndarray

References

[Lin15], [Man18]

Examples

```
>>> generate_pulse_waves(5)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 1.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  1.,  1.],
       [ 1.,  0.,  0.,  1.,  1.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  1.],
       [ 1.,  0.,  1.,  1.,  1.],
       [ 1.,  1.,  0.,  1.,  1.],
       [ 1.,  1.,  1.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
```

colour.volume.XYZ_outer_surface

`colour.volume.XYZ_outer_surface(interval=10, cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...), illuminant=SpectralDistribution(name='1 Constant', ...))`

Generates the *CIE XYZ* colourspace outer surface for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

Parameters

- **interval** (`int`, optional) – Wavelength λ_i range interval used to compute the pulse waves.
- **cmfs** (`XYZ_ColourMatchingFunctions`, optional) – Standard observer colour matching functions.
- **illuminant** (`SpectralDistribution`, optional) – Illuminant spectral distribution.

Returns Outer surface *CIE XYZ* tristimulus values.

Return type `ndarray`

References

[Lin15], [Man18]

Examples

```
>>> XYZ_outer_surface(84) # doctest: +ELLIPSIS
array([[ 0.0000000...e+00,  0.0000000...e+00,  0.0000000...e+00],
       [ 1.4766924...e-03,  4.1530347...e-05,  6.9884362...e-03],
       [ 1.6281275...e-01,  3.7114387...e-02,  9.0151471...e-01],
       [ 1.8650894...e-01,  5.6617464...e-01,  9.1355179...e-02],
       [ 6.1555347...e-01,  3.8427775...e-01,  4.7422070...e-04],
       [ 3.3622045...e-02,  1.2354556...e-02,  0.0000000...e+00],
       [ 1.0279500...e-04,  3.7121158...e-05,  0.0000000...e+00],
       [ 1.6428945...e-01,  3.7155917...e-02,  9.0850314...e-01],
       [ 3.4932169...e-01,  6.0328903...e-01,  9.9286989...e-01],
       [ 8.0206241...e-01,  9.5045240...e-01,  9.1829399...e-02],
       [ 6.4917552...e-01,  3.9663231...e-01,  4.7422070...e-04],
       [ 3.3724840...e-02,  1.2391678...e-02,  0.0000000...e+00],
       [ 1.5794874...e-03,  7.8651505...e-05,  6.9884362...e-03],
       [ 3.5079839...e-01,  6.0333056...e-01,  9.9985832...e-01],
       [ 9.6487517...e-01,  9.8756679...e-01,  9.9334411...e-01],
       [ 8.3568446...e-01,  9.6280696...e-01,  9.1829399...e-02],
       [ 6.4927831...e-01,  3.9666943...e-01,  4.7422070...e-04],
       [ 3.5201532...e-02,  1.2433208...e-02,  6.9884362...e-03],
       [ 1.6439224...e-01,  3.7193038...e-02,  9.0850314...e-01],
       [ 9.6635186...e-01,  9.8760832...e-01,  1.0003325...e+00],
       [ 9.9849722...e-01,  9.9992134...e-01,  9.9334411...e-01],
       [ 8.3578726...e-01,  9.6284408...e-01,  9.1829399...e-02],
       [ 6.5075501...e-01,  3.9671096...e-01,  7.4626569...e-03],
       [ 1.9801429...e-01,  4.9547595...e-02,  9.0850314...e-01],
       [ 3.5090118...e-01,  6.0336768...e-01,  9.9985832...e-01],
       [ 9.9997391...e-01,  9.9996287...e-01,  1.0003325...e+00],
       [ 9.9860001...e-01,  9.9995847...e-01,  9.9334411...e-01],
       [ 8.3726395...e-01,  9.6288561...e-01,  9.8817836...e-02],
       [ 8.1356776...e-01,  4.3382535...e-01,  9.0897737...e-01],
       [ 3.8452323...e-01,  6.1572224...e-01,  9.9985832...e-01],
       [ 9.6645466...e-01,  9.8764544...e-01,  1.0003325...e+00],
       [ 1.0000767...e+00,  1.0000000...e+00,  1.0003325...e+00]])
```

5.1.3.2 Indices and tables

- [genindex](#)
- [search](#)

5.1.4 Bibliography

5.1.4.1 Indirect References

Some extra references used in the codebase but not directly part of the public api:

- [\[Cen14e\]](#)
- [\[Cen14k\]](#)
- [\[Cen14h\]](#)
- [\[Cen14c\]](#)
- [\[Cen14j\]](#)

- [\[Cen14i\]](#)
- [\[Cen14g\]](#)
- [\[Cen14d\]](#)
- [\[Cen14f\]](#)
- [\[Cen14b\]](#)
- [\[Cen14a\]](#)
- [\[CIET13805d\]](#)
- [\[Dji17\]](#)
- [\[FiLMiCInc17\]](#)
- [\[Hou15\]](#)
- [\[Lau12\]](#)
- [\[Mac35\]](#)
- [\[Mac42\]](#)
- [\[MunsellCScienceb\]](#)
- [\[Poi80\]](#)
- [\[RenewableRDCenter03\]](#)
- [\[SWD05\]](#)
- [\[Sir18\]](#)
- [\[SHF00\]](#)
- [\[WEV02\]](#)
- [\[War16\]](#)
- [\[WS00b\]](#)
- [\[WS00j\]](#)
- [\[WS00l\]](#)
- [\[WS00h\]](#)

5.2 1.5.1 Examples

Most of the objects are available from the colour namespace:

```
>>> import colour
```

5.2.1 1.5.1.1 Chromatic Adaptation - `colour.adaptation`

```
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> D65 = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
>>> A = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['A']
>>> colour.chromatic_adaptation(
```

(continues on next page)

(continued from previous page)

```
... XYZ, colour.xy_to_XYZ(D65), colour.xy_to_XYZ(A))
array([ 0.2533053 ,  0.13765138,  0.01543307])
>>> sorted(colour.CHROMATIC_ADAPTATION_METHODS.keys())
['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries']
```

5.2.2 1.5.1.2 Algebra - colour.algebra

5.2.2.1 1.5.1.2.1 Kernel Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.KernelInterpolator(x, y)([0.25, 0.75, 5.50])
array([ 6.18062083,  8.08238488, 57.85783403])
```

5.2.2.2 1.5.1.2.2 Sprague (1880) Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.SpragueInterpolator(x, y)([0.25, 0.75, 5.50])
array([ 6.72951612,  7.81406251, 43.77379185])
```

5.2.3 1.5.1.3 Colour Appearance Models - colour.appearance

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b)
CIECAM02_Specification(J=34.434525727858997, C=67.365010921125915, h=22.279164147957076, s=62.
↳814855853327131, Q=177.47124941102123, M=70.024939419291385, H=2.689608534423904, HC=None)
```

5.2.4 1.5.1.4 Colour Blindness - colour.blindness

```
>>> import colour
>>> cmfs = colour.LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> colour.anomalous_trichromacy_cmfs_Machado2009(cmfs, np.array([15, 0, 0]))[450]
array([ 0.08912884,  0.0870524 ,  0.955393  ])
>>> primaries = colour.DISPLAYS_RGB_PRIMARIES['Apple Studio Display']
>>> d_LMS = (15, 0, 0)
>>> colour.anomalous_trichromacy_matrix_Machado2009(cmfs, primaries, d_LMS)
array([[ -0.27774652,  2.65150084, -1.37375432],
       [ 0.27189369,  0.20047862,  0.52762768],
       [ 0.00644047,  0.25921579,  0.73434374]])
```

5.2.5 1.5.1.5 Colour Correction - colour characterisation

```
>>> import numpy as np
>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> M_T = np.random.random((24, 3))
>>> M_R = M_T + (np.random.random((24, 3)) - 0.5) * 0.5
>>> colour.colour_correction(RGB, M_T, M_R)
array([ 0.15205429, 0.08974029, 0.04141435])
>>> sorted(colour.COLOUR_CORRECTION_METHODS.keys())
['Cheung 2004', 'Finlayson 2015', 'Vandermonde']
```

5.2.6 1.5.1.6 Colorimetry - colour.colorimetry

5.2.6.1 1.5.1.6.1 Spectral Computations

```
>>> colour.sd_to_XYZ(colour.LIGHT_SOURCES_SDS['Neodimium Incandescent'])
array([ 36.94726204, 32.62076174, 13.0143849 ])
>>> sorted(colour.SPECTRAL_TO_XYZ_METHODS.keys())
['ASTM E308-15', 'Integration', 'astm2015']
```

5.2.6.2 1.5.1.6.2 Multi-Spectral Computations

```
>>> msd = np.array([
...     [[0.01367208, 0.09127947, 0.01524376, 0.02810712, 0.19176012, 0.04299992],
...       [0.00959792, 0.25822842, 0.41388571, 0.22275120, 0.00407416, 0.37439537],
...       [0.01791409, 0.29707789, 0.56295109, 0.23752193, 0.00236515, 0.58190280],
...       [0.01492332, 0.10421912, 0.02240025, 0.03735409, 0.57663846, 0.32416266],
...       [0.04180972, 0.26402685, 0.03572137, 0.00413520, 0.41808194, 0.24696727],
...       [0.00628672, 0.11454948, 0.02198825, 0.39906919, 0.63640803, 0.01139849],
...       [0.04325933, 0.26825359, 0.23732357, 0.05175860, 0.01181048, 0.08233768],
...       [0.02484169, 0.12027161, 0.00541695, 0.00654612, 0.18603799, 0.36247808],
...       [0.03102159, 0.16815442, 0.37186235, 0.08610666, 0.00413520, 0.78492409],
...       [0.11682307, 0.78883040, 0.74468607, 0.83375293, 0.90571451, 0.70054168],
...       [0.06321812, 0.41898224, 0.15190357, 0.24591440, 0.55301750, 0.00657664],
...       [0.00305180, 0.11288624, 0.11357290, 0.12924391, 0.00195315, 0.21771573]],
... ])
>>> colour.multi_sds_to_XYZ(msd, colour.SpectralShape(400, 700, 60),
...                           cmfs, illuminant))
[[[ 9.73192501  5.02105851  3.22790699]
 [ 16.08032168 24.47303359 10.28681006]
 [ 17.73513774 29.61865582 12.10713449]]
 [[ 25.69298792 11.72611193  3.70187275]
 [ 18.51208526  8.03720984  9.30361825]
 [ 48.55945054 32.30885571  4.09223401]]
 [[ 5.7743232 10.10692925 10.08461311]
 [ 8.81306527 3.65394599  4.20783881]
 [ 8.06007398 15.87077693  7.02551086]]
 [[ 90.88877129 81.82966846 29.86765971]
 [ 38.64801062 26.70860262 15.08396538]
 [ 8.77151115 10.56330761  4.28940206]]]
>>> sorted(colour.MULTI_SPECTRAL_TO_XYZ_METHODS.keys())
['Integration']
```

5.2.6.3 1.5.1.6.3 Blackbody Spectral Radiance Computation

```
>>> colour.sd_blackbody(5000)
SpectralDistribution([[ 3.60000000e+02,  6.65427827e+12],
                    [ 3.61000000e+02,  6.70960528e+12],
                    [ 3.62000000e+02,  6.76482512e+12],
                    ...
                    [ 7.78000000e+02,  1.06068004e+13],
                    [ 7.79000000e+02,  1.05903327e+13],
                    [ 7.80000000e+02,  1.05738520e+13]],
                    interpolator=SpragueInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={'right': None, 'method': 'Constant', 'left': None})
```

5.2.6.4 1.5.1.6.4 Dominant, Complementary Wavelength & Colour Purity Computation

```
>>> xy = [0.54369557, 0.32107944]
>>> xy_n = [0.31270000, 0.32900000]
>>> colour.dominant_wavelength(xy, xy_n)
(array(616.0),
 array([ 0.68354746,  0.31628409]),
 array([ 0.68354746,  0.31628409]))
```

5.2.6.5 1.5.1.6.5 Lightness Computation

```
>>> colour.lightness(12.19722535)
41.527875844653451
>>> sorted(colour.LIGHTNESS_METHODS.keys())
['CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Glasser 1958',
 'Lstar1976',
 'Wyszecki 1963']
```

5.2.6.6 1.5.1.6.6 Luminance Computation

```
>>> colour.luminance(41.52787585)
12.197225353400775
>>> sorted(colour.LUMINANCE_METHODS.keys())
['ASTM D1535-08',
 'CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Newhall 1943',
 'astm2008',
 'cie1976']
```

5.2.6.7 1.5.1.6.7 Whiteness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> XYZ_0 = [94.80966767, 100.00000000, 107.30513595]
>>> colour.whiteness(XYZ, XYZ_0)
array([ 93.756      , -1.33000001])
>>> sorted(colour.WHITENESS_METHODS.keys())
['ASTM E313',
 'Berger 1959',
 'CIE 2004',
 'Ganz 1979',
 'Stensby 1968',
 'Taube 1960',
 'cie2004']
```

5.2.6.8 1.5.1.6.8 Yellowness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> colour.yellowness(XYZ)
11.065000000000003
>>> sorted(colour.YELLOWNESS_METHODS.keys())
['ASTM D1925', 'ASTM E313']
```

5.2.6.9 1.5.1.6.9 Luminous Flux, Efficiency & Efficacy Computation

```
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_flux(sd)
23807.655527367202
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_efficiency(sd)
0.19943935624521045
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_efficacy(sd)
136.21708031547874
```

5.2.7 1.5.1.7 Contrast Sensitivity Function - colour.contrast

```
>>> colour.contrast_sensitivity_function(u=4, X_0=60, E=65)
358.51180789884984
>>> sorted(colour.CONTRAST_SENSITIVITY_METHODS.keys())
['Barten 1999']
```

5.2.8 1.5.1.8 Colour Difference - colour.difference

```
>>> Lab_1 = [100.00000000, 21.57210357, 272.22819350]
>>> Lab_2 = [100.00000000, 426.67945353, 72.39590835]
>>> colour.delta_E(Lab_1, Lab_2)
94.035649026659485
>>> sorted(colour.DELTA_E_METHODS.keys())
```

(continues on next page)

(continued from previous page)

```
[ 'CAM02-LCD',
  'CAM02-SCD',
  'CAM02-UCS',
  'CAM16-LCD',
  'CAM16-SCD',
  'CAM16-UCS',
  'CIE 1976',
  'CIE 1994',
  'CIE 2000',
  'CMC',
  'DIN99',
  'cie1976',
  'cie1994',
  'cie2000']
```

5.2.9 1.5.1.9 IO - colour.io

5.2.9.1 1.5.1.9.1 Images

```
>>> RGB = colour.read_image('Ishihara_Colour_Blindness_Test_Plate_3.png')
>>> RGB.shape
(276, 281, 3)
```

5.2.9.2 1.5.1.9.2 Look Up Table (LUT) Data

```
>>> LUT = colour.read_LUT('ACES_Proxy_10_to_ACES.cube')
>>> print(LUT)
LUT3x1D - ACES Proxy 10 to ACES
-----
Dimensions : 2
Domain      : [[0 0 0]
               [1 1 1]]
Size       : (32, 3)

>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> LUT.apply(RGB)
array([ 0.00575674,  0.00181493,  0.00121419])
```

5.2.10 1.5.1.10 Colour Models - colour.models

5.2.10.1 1.5.1.10.1 CIE xyY Colourspace

```
>>> colour.XYZ_to_xyY([0.20654008, 0.12197225, 0.05136952])
array([ 0.54369557,  0.32107944,  0.12197225])
```

5.2.10.2 1.5.1.10.2 CIE L*a*b* Colourspace

```
>>> colour.XYZ_to_Lab([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  52.63858304,  26.92317922])
```

5.2.10.3 1.5.1.10.3 CIE L*u*v* Colourspace

```
>>> colour.XYZ_to_Luv([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  96.83626054,  17.75210149])
```

5.2.10.4 1.5.1.10.4 CIE 1960 UCS Colourspace

```
>>> colour.XYZ_to_UCS([0.20654008, 0.12197225, 0.05136952])
array([ 0.13769339,  0.12197225,  0.1053731 ])
```

5.2.10.5 1.5.1.10.5 CIE 1964 U*V*W* Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_UVW(XYZ)
array([ 94.55035725,  11.55536523,  40.54757405])
```

5.2.10.6 1.5.1.10.6 Hunter L,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_Hunter_Lab(XYZ)
array([ 34.92452577,  47.06189858,  14.38615107])
```

5.2.10.7 1.5.1.10.7 Hunter Rd,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_Hunter_Rdab(XYZ)
array([ 12.197225 ,  57.12537874,  17.46241341])
```

5.2.10.8 1.5.1.10.8 CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = colour.XYZ_to_CIECAM02(
    XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 47.16899898,  38.72623785,  15.8663383 ])
```

5.2.10.9 1.5.1.10.9 CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.CAM16_VIEWING_CONDITIONS['Average']
>>> specification = colour.XYZ_to_CAM16(
    XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CAM16_to_CAM16UCS(JMh)
array([ 46.55542238,  40.22460974,  14.25288392])
```

5.2.10.10 1.5.1.10.10 IPT Colourspace

```
>>> colour.XYZ_to_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 0.38426191,  0.38487306,  0.1886838])
```

5.2.10.11 1.5.1.10.11 DIN99 Colourspace

```
>>> Lab = [41.52787529, 52.63858304, 26.92317922]
>>> colour.Lab_to_DIN99(Lab)
array([ 53.22821988,  28.41634656,   3.89839552])
```

5.2.10.12 1.5.1.10.12 hdr-CIELAB Colourspace

```
>>> colour.XYZ_to_hdr_CIELab([0.20654008, 0.12197225, 0.05136952])
array([ 51.87002062,  60.4763385 ,  32.14551912])
```

5.2.10.13 1.5.1.10.13 hdr-IPT Colourspace

```
>>> colour.XYZ_to_hdr_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 25.18261761, -22.62111297,   3.18511729])
```

5.2.10.14 1.5.1.10.14 OSA UCS Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> colour.XYZ_to_OSA_UCS(XYZ)
array([-3.0049979 ,  2.99713697, -9.66784231])
```

5.2.10.15 1.5.1.10.15 JzAzBz Colourspace

```
>>> colour.XYZ_to_JzAzBz([0.20654008, 0.12197225, 0.05136952])
array([ 0.00535048,  0.00924302,  0.00526007])
```

5.2.10.16 1.5.1.10.16 Y'CbCr Colour Encoding

```
>>> colour.RGB_to_YCbCr([1.0, 1.0, 1.0])
array([ 0.92156863,  0.50196078,  0.50196078])
```

5.2.10.17 1.5.1.10.17 YCoCg Colour Encoding

```
>>> colour.RGB_to_YCoCg([0.75, 0.75, 0.0])
array([ 0.5625,  0.375 ,  0.1875])
```

5.2.10.18 1.5.1.10.18 ICTCP Colour Encoding

```
>>> colour.RGB_to_ICTCP([0.45620519, 0.03081071, 0.04091952])
array([ 0.07351364,  0.00475253,  0.09351596])
```

5.2.10.19 1.5.1.10.19 HSV Colourspace

```
>>> colour.RGB_to_HSV([0.45620519, 0.03081071, 0.04091952])
array([ 0.99603944,  0.93246304,  0.45620519])
```

5.2.10.20 1.5.1.10.20 Prismatic Colourspace

```
>>> colour.RGB_to_Prismatic([0.25, 0.50, 0.75])
array([ 0.75      ,  0.16666667,  0.33333333,  0.5      ])
```

5.2.10.21 1.5.1.10.21 RGB Colourspace and Transformations

```
>>> XYZ = [0.21638819, 0.12570000, 0.03847493]
>>> illuminant_XYZ = [0.34570, 0.35850]
>>> illuminant_RGB = [0.31270, 0.32900]
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = [
    [3.24062548, -1.53720797, -0.49862860],
    [-0.96893071, 1.87575606, 0.04151752],
    [0.05571012, -0.20402105, 1.05699594]]
>>> colour.XYZ_to_RGB(
    XYZ,
    illuminant_XYZ,
    illuminant_RGB,
    XYZ_to_RGB_matrix,
    chromatic_adaptation_transform)
array([ 0.45595571,  0.03039702,  0.04087245])
```

5.2.10.22 1.5.1.10.22 RGB Colourspace Derivation


```
>>> p = [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
>>> w = [0.32168, 0.33767]
>>> colour.normalised_primary_matrix(p, w)
array([[ 9.52552396e-01,  0.00000000e+00,  9.36786317e-05],
       [ 3.43966450e-01,  7.28166097e-01, -7.21325464e-02],
       [ 0.00000000e+00,  0.00000000e+00,  1.00882518e+00]])
```

5.2.10.23 1.5.1.10.23 RGB Colourspaces

```
>>> sorted(colour.RGB_COLOURSPACES.keys())
['ACES2065-1',
 'ACEScc',
 'ACEScct',
 'ACEScg',
 'ACESproxy',
 'ALEXA Wide Gamut',
 'Adobe RGB (1998)',
 'Adobe Wide Gamut RGB',
 'Apple RGB',
 'Best RGB',
 'Beta RGB',
 'CIE RGB',
 'Cinema Gamut',
 'ColorMatch RGB',
 'DCDM XYZ',
 'DCI-P3',
 'DCI-P3+',
 'DJI D-Gamut',
 'DRAGONcolor',
 'DRAGONcolor2',
 'Don RGB 4',
 'ECI RGB v2',
 'ERIMM RGB',
 'Ekta Space PS 5',
 'FilmLight E-Gamut',
 'ITU-R BT.2020',
 'ITU-R BT.470 - 525',
 'ITU-R BT.470 - 625',
 'ITU-R BT.709',
 'Max RGB',
 'NTSC',
 'P3-D65',
 'Pal/Secam',
 'ProPhoto RGB',
 'Protune Native',
 'REDWideGamutRGB',
 'REDcolor',
 'REDcolor2',
 'REDcolor3',
 'REDcolor4',
 'RIMM RGB',
 'ROMM RGB',
 'Russell RGB',
 'S-Gamut',
 'S-Gamut3',
```

(continues on next page)

(continued from previous page)

```
'S-Gamut3.Cine',  
'SMPTE 240M',  
'Sharp RGB',  
'V-Gamut',  
'Xtreme RGB',  
'aces',  
'adobe1998',  
'prophoto',  
'sRGB']
```

5.2.10.24 1.5.1.10.24 OETFs

```
>>> sorted(colour.OETFs.keys())  
['ARIB STD-B67',  
'DICOM GSDF',  
'ITU-R BT.2020',  
'ITU-R BT.2100 HLG',  
'ITU-R BT.2100 PQ',  
'ITU-R BT.601',  
'ITU-R BT.709',  
'ProPhoto RGB',  
'RIMM RGB',  
'ROMM RGB',  
'SMPTE 240M',  
'ST 2084',  
'sRGB']
```

5.2.10.25 1.5.1.10.25 OETFs Reverse

```
>>> sorted(colour.OETFs_REVERSE.keys())  
['ARIB STD-B67',  
'ITU-R BT.2100 HLD',  
'ITU-R BT.2100 PQ',  
'ITU-R BT.601',  
'ITU-R BT.709',  
'sRGB']
```

5.2.10.26 1.5.1.10.26 EOTFs

```
>>> sorted(colour.EOTFs.keys())  
['DCDM',  
'DICOM GSDF',  
'ITU-R BT.1886',  
'ITU-R BT.2020',  
'ITU-R BT.2100 HLG',  
'ITU-R BT.2100 PQ',  
'ProPhoto RGB',  
'RIMM RGB',  
'ROMM RGB',  
'SMPTE 240M',  
'ST 2084']
```

5.2.10.27 1.5.1.10.27 EOTFs Reverse

```
>>> sorted(colour.EOTFS_REVERSE.keys())
['DCDM', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

5.2.10.28 1.5.1.10.28 OOTFs

```
>>> sorted(colour.OOTFS.keys())
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

5.2.10.29 1.5.1.10.29 OOTFs Reverse

```
>>> sorted(colour.OOTFS_REVERSE.keys())
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

5.2.10.30 1.5.1.10.30 Log Encoding / Decoding Curves

```
>>> sorted(colour.LOG_ENCODING_CURVES.keys())
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'ERIMM RGB',
 'Filmic Pro 6',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'T-Log',
 'V-Log',
 'ViperLog']
```

5.2.11 1.5.1.11 Colour Notation Systems - colour.notation

5.2.11.1 1.5.1.11.1 Munsell Value

```
>>> colour.munsell_value(12.23634268)
4.0824437076525664
>>> sorted(colour.MUNSELL_VALUE_METHODS.keys())
['ASTM D1535-08',
 'Ladd 1955',
 'McCamy 1987',
 'Moon 1943',
 'Munsell 1933',
 'Priest 1920',
 'Saunderson 1944',
 'astm2008']
```

5.2.11.2 1.5.1.11.2 Munsell Colour

```
>>> colour.xyY_to_munsell_colour([0.38736945, 0.35751656, 0.59362000])
'4.2YR 8.1/5.3'
>>> colour.munsell_colour_to_xyY('4.2YR 8.1/5.3')
array([ 0.38736945,  0.35751656,  0.59362    ])
```

5.2.12 1.5.1.12 Optical Phenomena - colour.phenomena

```
>>> colour.rayleigh_scattering_sd()
SpectralDistribution([[ 3.60000000e+02,  5.99101337e-01],
 [ 3.61000000e+02,  5.92170690e-01],
 [ 3.62000000e+02,  5.85341006e-01],
 ...,
 [ 7.78000000e+02,  2.55208377e-02],
 [ 7.79000000e+02,  2.53887969e-02],
 [ 7.80000000e+02,  2.52576106e-02]],
 interpolator=SpragueInterpolator,
 interpolator_args={},
 extrapolator=Extrapolator,
 extrapolator_args={'right': None, 'method': 'Constant', 'left': None})
```

5.2.13 1.5.1.13 Light Quality - colour.quality

5.2.13.1 1.5.1.13.1 Colour Rendering Index

```
>>> colour.colour_quality_scale(colour.ILLUMINANTS_SDS['FL2'])
64.017283509280588
>>> colour.COLOUR_QUALITY_SCALE_METHODS
('NIST CQS 7.4', 'NIST CQS 9.0')
```

5.2.13.2 1.5.1.13.2 Colour Quality Scale

```
>>> colour.colour_rendering_index(colour.ILLUMINANTS_SDS['FL2'])
64.151520202968015
```

5.2.14 1.5.1.14 Spectral Up-sampling & Reflectance Recovery - `colour.recovery`

```
>>> colour.XYZ_to_sd([0.20654008, 0.12197225, 0.05136952])
SpectralDistribution([[ 3.60000000e+02,  7.73462151e-02],
                    [ 3.65000000e+02,  7.73632975e-02],
                    [ 3.70000000e+02,  7.74299705e-02],
                    ...,
                    [ 8.20000000e+02,  3.93126353e-01],
                    [ 8.25000000e+02,  3.93158148e-01],
                    [ 8.30000000e+02,  3.93163548e-01]],
                    interpolator=SpragueInterpolator,
                    interpolator_args={},
                    extrapolator=Extrapolator,
                    extrapolator_args={'right': None, 'method': 'Constant', 'left': None})

>>> sorted(colour.REFLECTANCE_RECOVERY_METHODS.keys())
['Meng 2015', 'Smits 1999']
```

5.2.15 1.5.1.15 Correlated Colour Temperature Computation Methods - `colour.temperature`

```
>>> colour.uv_to_CCT([0.1978, 0.3122])
array([ 6.50751282e+03,  3.22335875e-03])
>>> sorted(colour.UV_TO_CCT_METHODS.keys())
['Ohno 2013', 'Robertson 1968', 'ohno2013', 'robertson1968']
>>> sorted(colour.UV_TO_CCT_METHODS.keys())
['Krystek 1985',
 'Ohno 2013',
 'Robertson 1968',
 'ohno2013',
 'robertson1968']
>>> sorted(colour.XY_TO_CCT_METHODS.keys())
['Hernandez 1999', 'McCamy 1992', 'hernandez1999', 'mccamy1992']
>>> sorted(colour.CCT_TO_XY_METHODS.keys())
['CIE Illuminant D Series', 'Kang 2002', 'cie_d', 'kang2002']
```

5.2.16 1.5.1.16 Colour Volume - `colour.volume`

```
>>> colour.RGB_colourspace_volume_MonteCarlo(colour.RGB_COLOURSPACE['sRGB'])
821958.300000000005
```

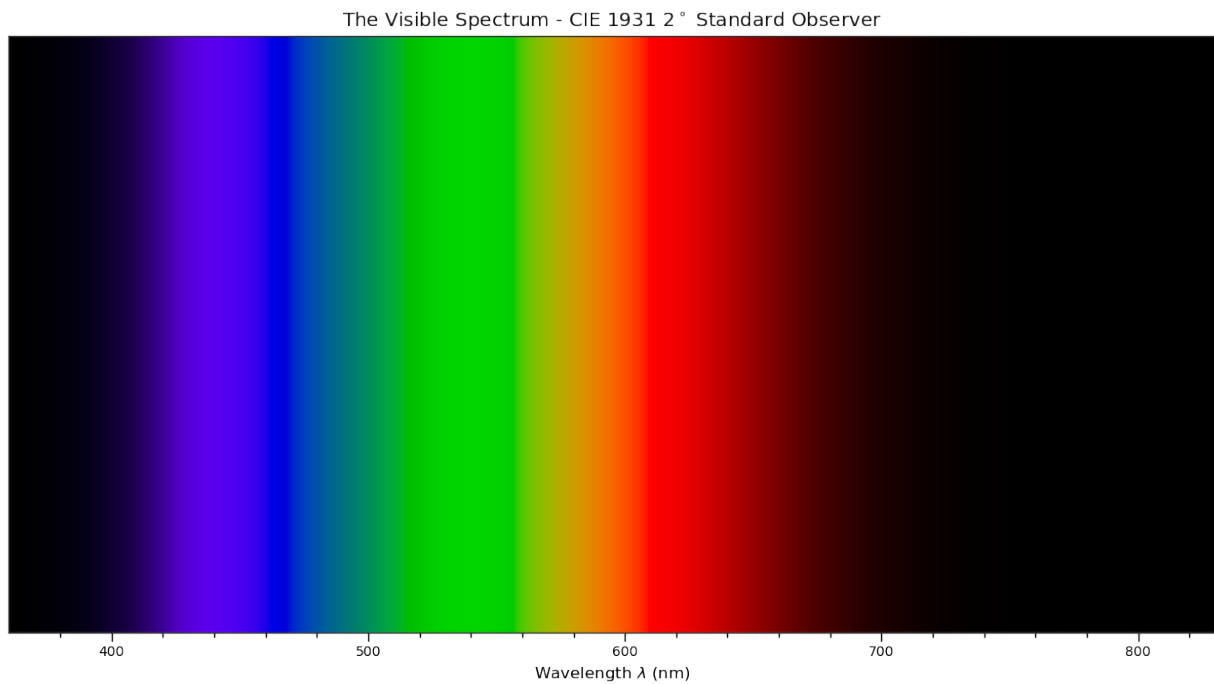
5.2.17 1.5.1.17 Plotting - `colour.plotting`

Most of the objects are available from the `colour.plotting` namespace:

```
>>> from colour.plotting import *
>>> colour_style()
```

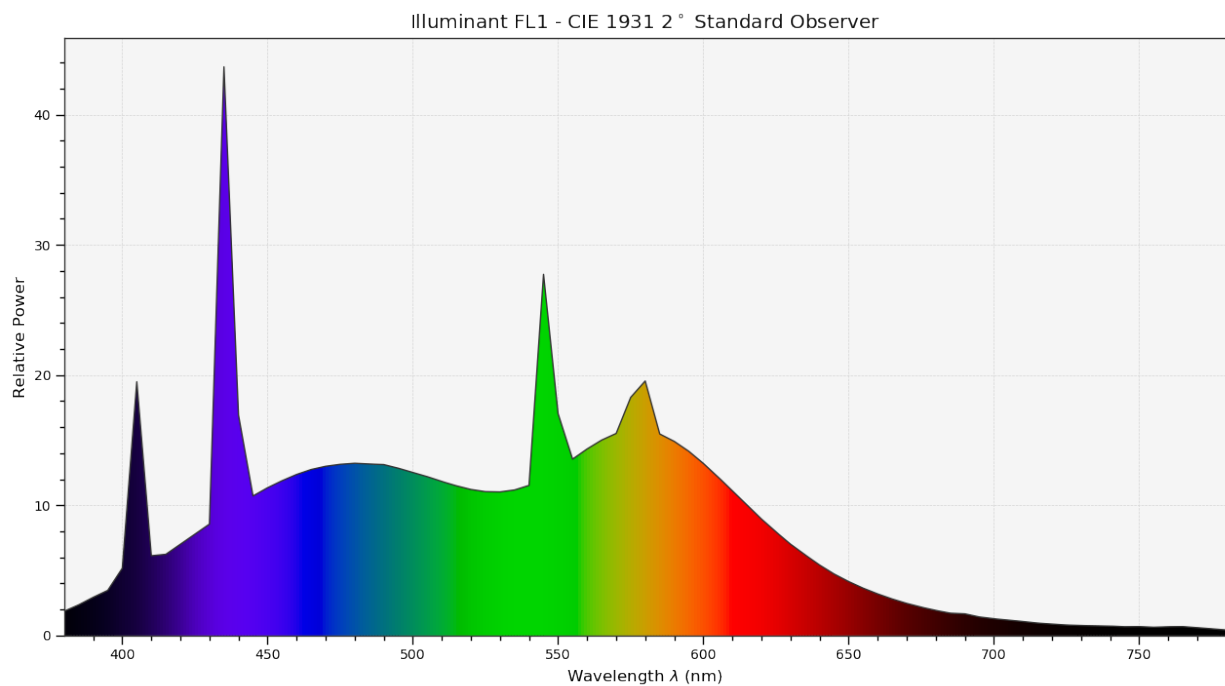
5.2.17.1 1.5.1.17.1 Visible Spectrum

```
>>> plot_visible_spectrum('CIE 1931 2 Degree Standard Observer')
```



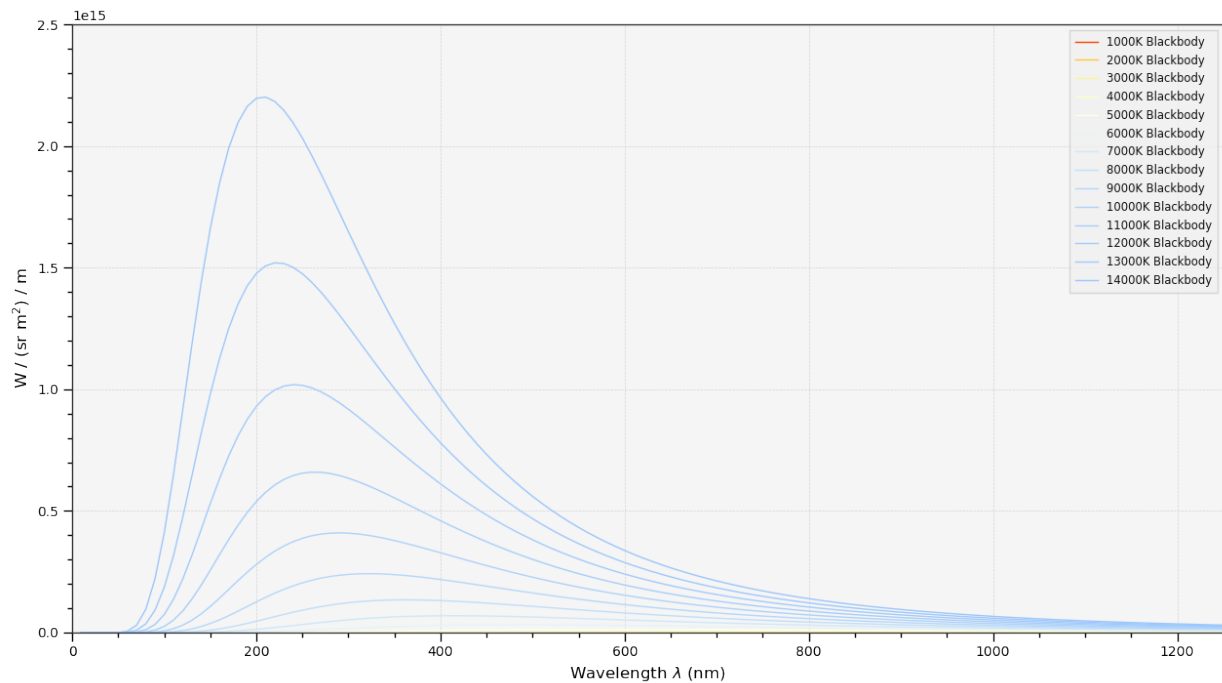
5.2.17.2 1.5.1.17.2 Spectral Distribution

```
>>> plot_single_illuminant_sd('FL1')
```



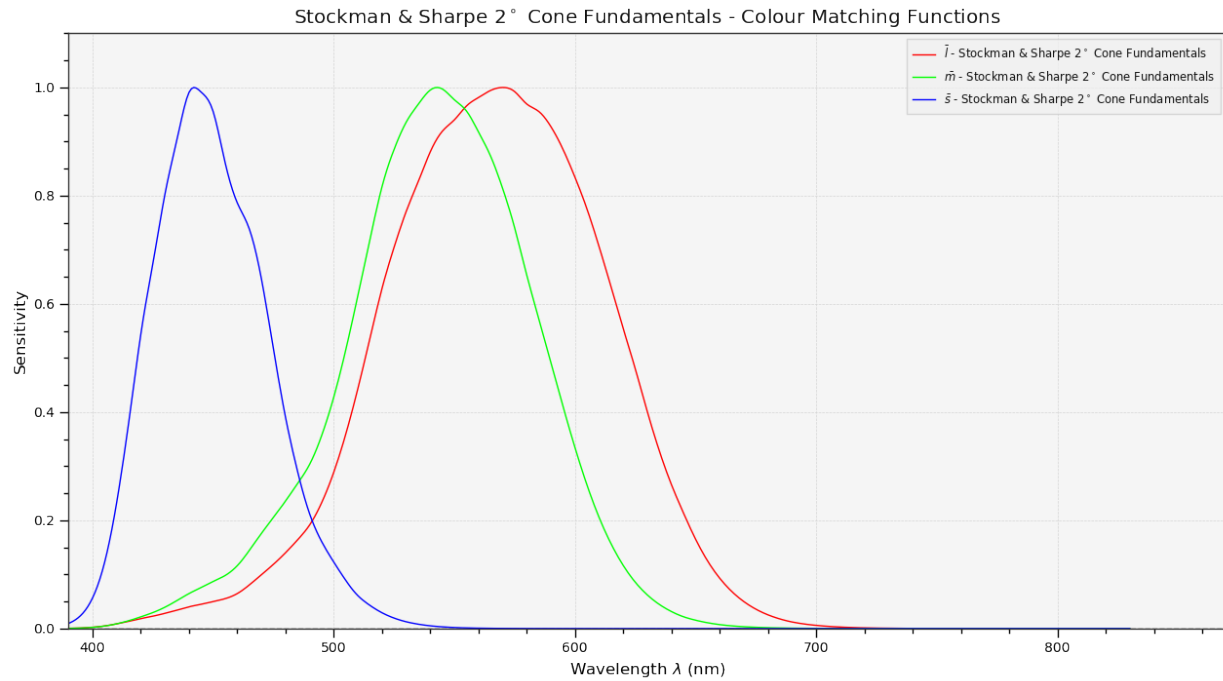
5.2.17.3 1.5.1.17.3 Blackbody

```
>>> blackbody_sds = [
...     colour.sd_blackbody(i, colour.SpectralShape(0, 10000, 10))
...     for i in range(1000, 15000, 1000)
... ]
>>> plot_multi_sds(
...     blackbody_sds,
...     y_label='W / (sr m$^2$) / m',
...     use_sds_colours=True,
...     normalise_sds_colours=True,
...     legend_location='upper right',
...     bounding_box=(0, 1250, 0, 2.5e15))
```



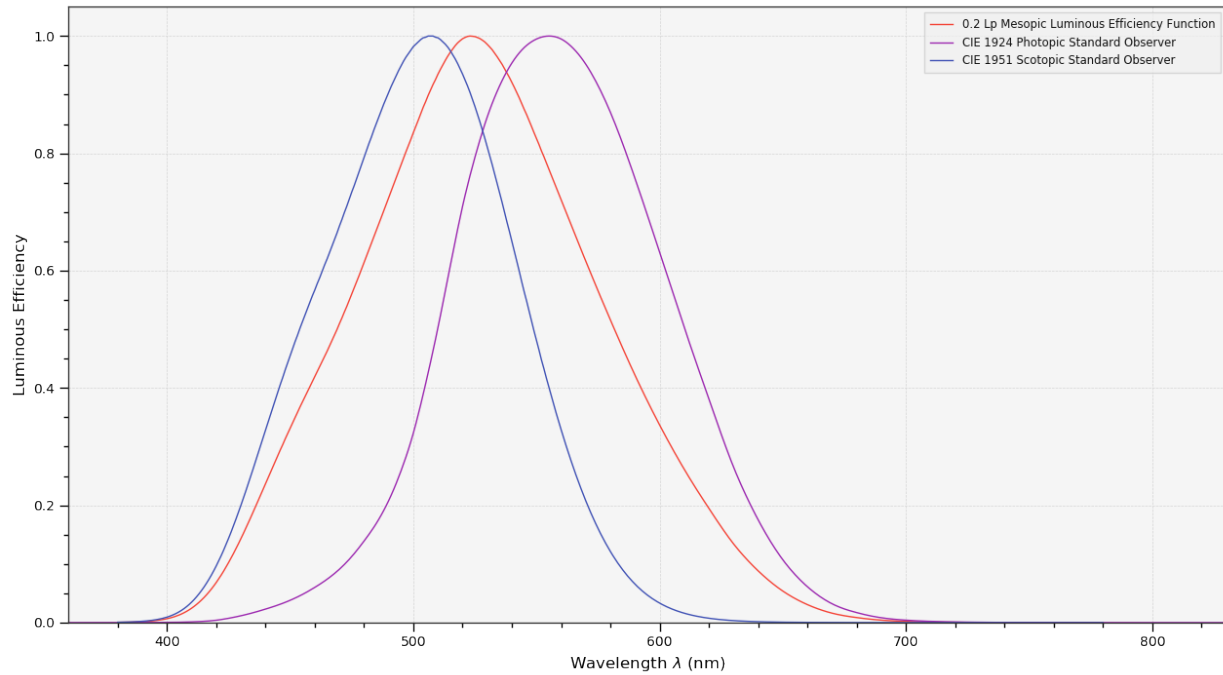
5.2.17.4 1.5.1.17.4 Colour Matching Functions

```
>>> plot_single_cmfs(
...     'Stockman & Sharpe 2 Degree Cone Fundamentals',
...     y_label='Sensitivity',
...     bounding_box=(390, 870, 0, 1.1))
```



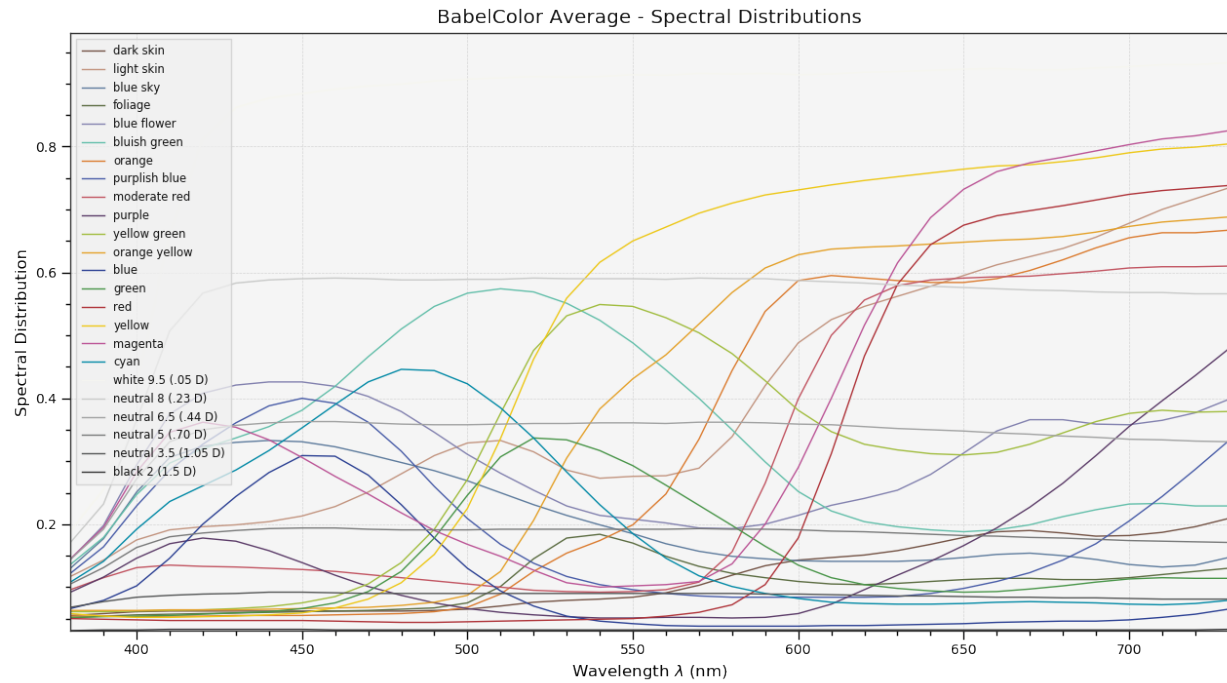
5.2.17.5 1.5.1.17.5 Luminous Efficiency

```
>>> sd_mesopic_luminous_efficiency_function = (
...     colour.sd_mesopic_luminous_efficiency_function(0.2))
>>> plot_multi_sds(
...     (sd_mesopic_luminous_efficiency_function,
...      colour.PHOTOPIC_LEFS['CIE 1924 Photopic Standard Observer'],
...      colour.SCOTOPIC_LEFS['CIE 1951 Scotopic Standard Observer']),
...     y_label='Luminous Efficiency',
...     legend_location='upper right',
...     y_tighten=True,
...     margins=(0, 0, 0, .1))
```

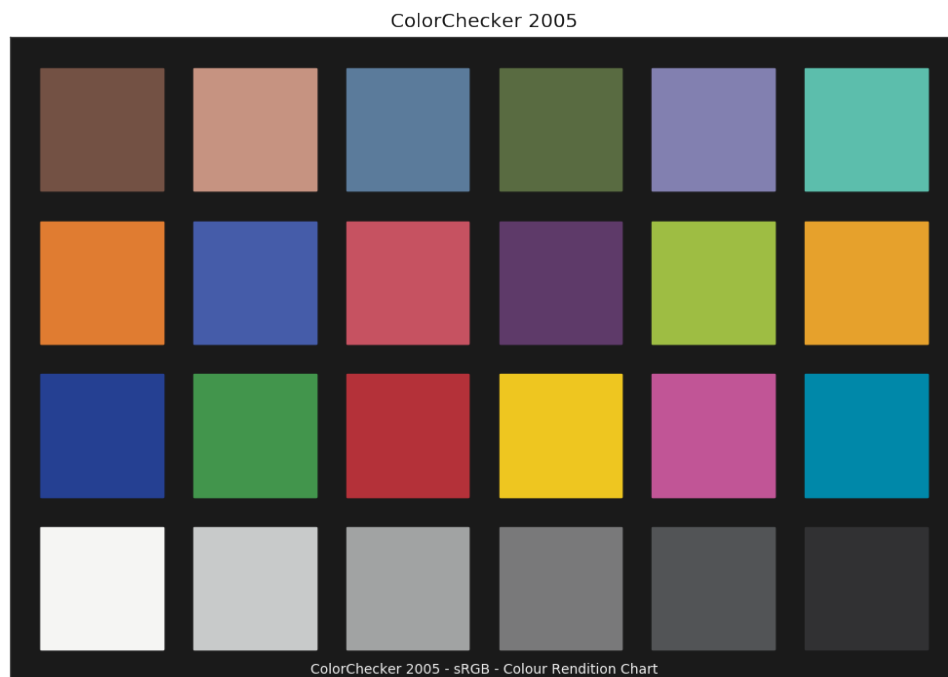



5.2.17.6 1.5.1.17.6 Colour Checker

```
>>> from colour.characterisation.dataset.colour_checkers.sds import (
...     COLOURCHECKER_INDEXES_TO_NAMES_MAPPING)
>>> plot_multi_sds(
...     [
...         colour.COLOURCHECKERS_SDS['BabelColor Average'][value]
...         for key, value in sorted(
...             COLOURCHECKER_INDEXES_TO_NAMES_MAPPING.items())
...     ],
...     use_sds_colours=True,
...     title=('BabelColor Average - '
...            'Spectral Distributions'))
```

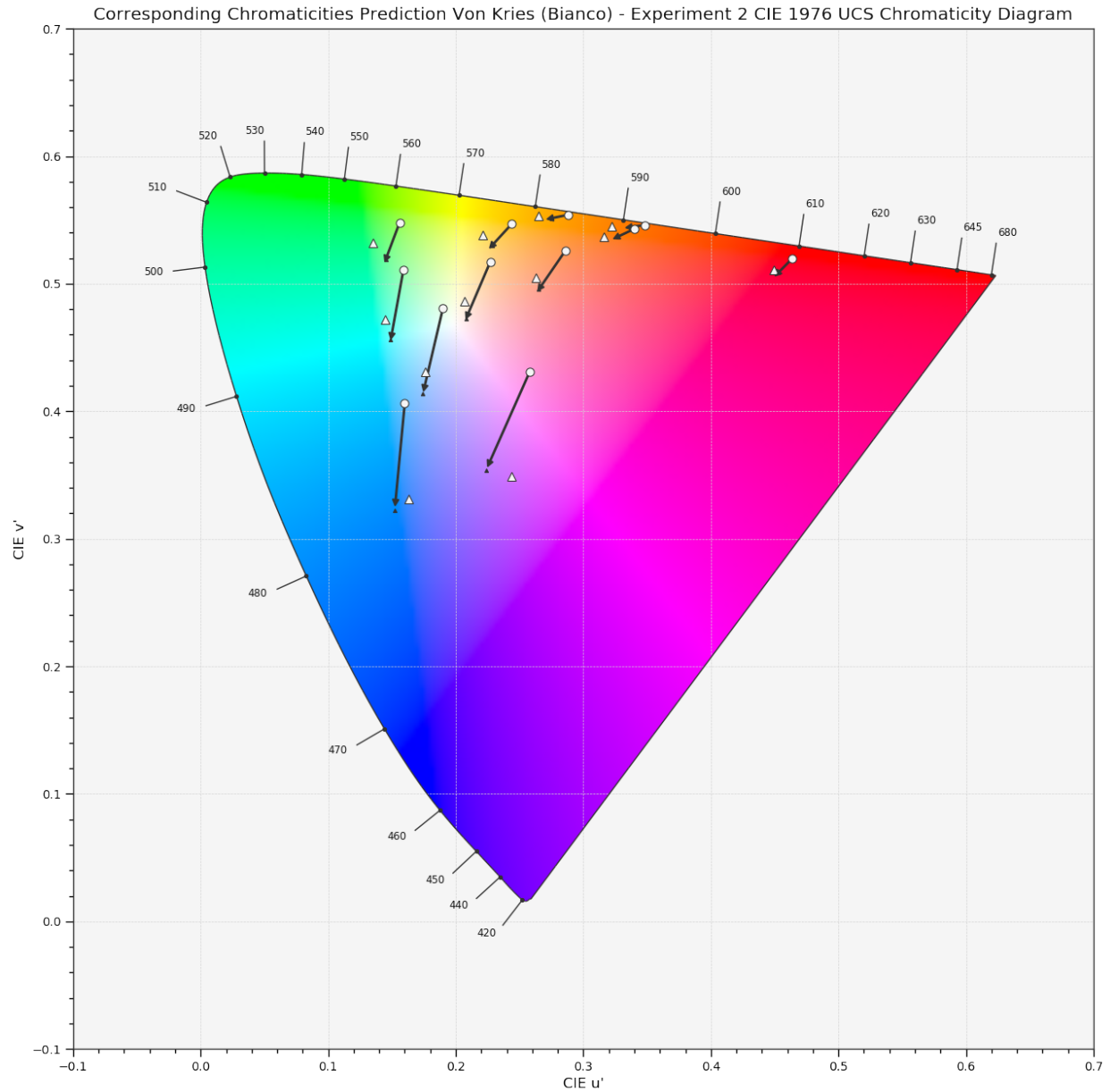


```
>>> plot_single_colour_checker('ColorChecker 2005', text_parameters={'visible': False})
```



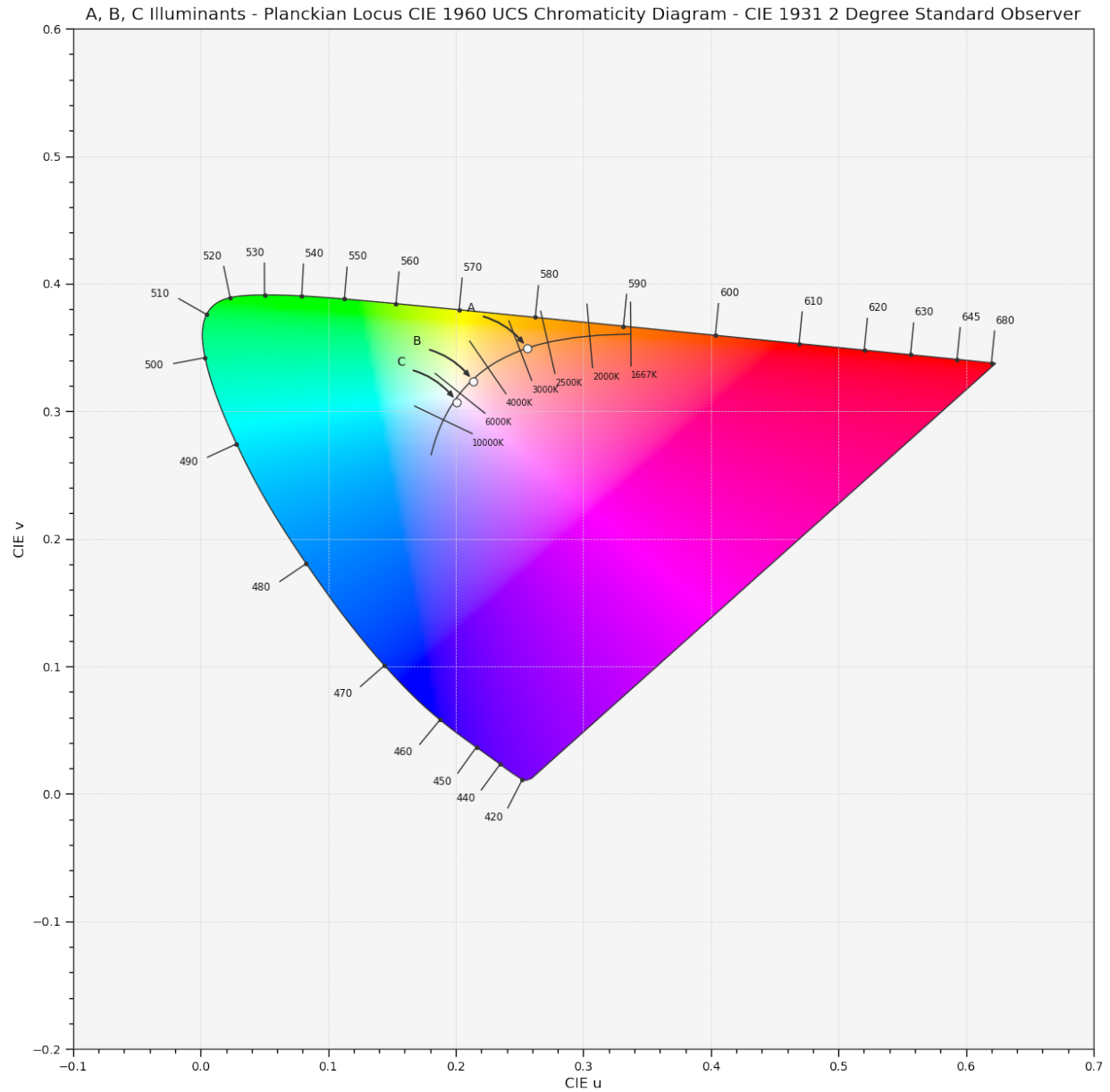
5.2.17.7 1.5.1.17.7 Chromaticities Prediction

```
>>> plot_corresponding_chromaticities_prediction(2, 'Von Kries', 'Bianco')
```



5.2.17.8 1.5.1.17.8 Colour Temperature

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(['A', 'B', 'C'])
```



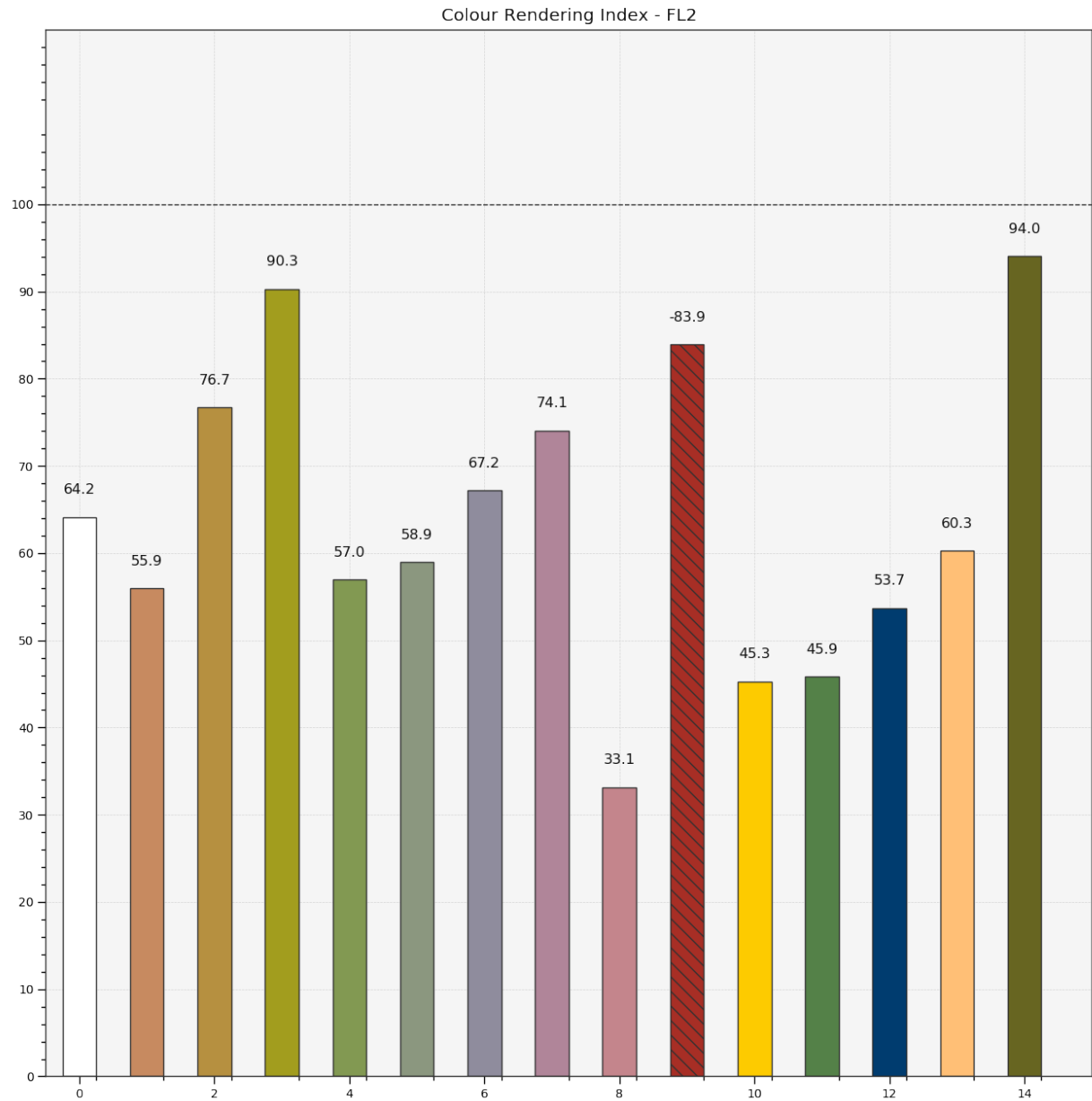
5.2.17.9 1.5.1.17.9 Chromaticities

```
>>> import numpy as np
>>> RGB = np.random.random((32, 32, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB, 'ITU-R BT.709', colourspaces=['ACEScg', 'S-Gamut'], show_pointer_gamut=True)
```



5.2.17.10 1.5.1.17.10 Colour Rendering Index

```
>>> plot_single_sd_colour_rendering_index_bars(
...     colour.ILLUMINANTS_SDS['FL2'])
```



CHAPTER 6

1.6 Contributing

If you would like to contribute to **Colour**, please refer to the following [Contributing](#) guide.

CHAPTER 7

1.7 Changes

The changes are viewable on the [Releases](#) page.

CHAPTER 8

1.8 Bibliography

The bibliography is available on the [Bibliography](#) page.

It is also viewable directly from the repository in [BibTeX](#) format.

Here is a list of notable colour science packages sorted by languages:

Python

- [Colorio](#) by Schlömer, N.
- [ColorPy](#) by Kness, M.
- [Colorspacious](#) by Smith, N. J., et al.
- [python-colormath](#) by Taylor, G., et al.

.NET

- [Colourful](#) by Pažourek, T., et al.

Julia

- [Colors.jl](#) by Holy, T., et al.

Matlab & Octave

- [COLORLAB](#) by Malo, J., et al.
- [Psychtoolbox](#) by Brainard, D., et al.
- [The Munsell and Kubelka-Munk Toolbox](#) by Centore, P.

CHAPTER 10

1.10 About

Colour by Colour Developers - 2013-2019

Copyright © 2013-2019 – Colour Developers – colour-science@googlegroups.com

This software is released under terms of New BSD License: <https://opensource.org/licenses/BSD-3-Clause>
<https://github.com/colour-science/colour>

Bibliography

- [ANS03] ANSI. Specification of ROMM RGB. 2003. URL: <http://www.color.org/ROMMRGB.pdf>.
- [ARR12] ARRI. ALEXA - Log C Curve - Usage in VFX. 2012. URL: https://drive.google.com/open?id=1t73fAG_QpV7hJxoQPYZDWvOojYkYDgvn.
- [Bab12a] BabelColor. ColorChecker RGB and spectra. 2012. URL: http://www.babelcolor.com/download/ColorChecker_RGB_and_spectra.xls.
- [Bab12b] BabelColor. The ColorChecker (since 1976!). 2012. URL: http://www.babelcolor.com/main_level/ColorChecker.htm.
- [Bar99] Peter G. Barten. *Contrast Sensitivity of the Human Eye and Its Effects on Image Quality*. Number 1999. SPIE, December 1999. ISBN 9780819478498. URL: <http://ebooks.spiedigitallibrary.org/book.aspx?doi=10.1117/3.353254><https://spiedigitallibrary.org/ebooks/PM/Contrast-Sensitivity-of-the-Human-Eye-and-Its-Effects-on/eISBN-9780819478498/10.1117/3.353254>, doi:10.1117/3.353254.
- [Bar03] Peter G. J. Barten. Formula for the contrast sensitivity of the human eye. In Yoichi Miyake and D. Rene Rasmussen, editors, *Proceedings of SPIE*, volume 5294, 231–238. December 2003. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed\T1\textbackslash{}&cmd=Retrieve\T1\textbackslash{}&dopt=AbstractPlus\T1\textbackslash{}&list_uids=15798324032973700240related:kPzpPwHsPtsJ<http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.135956><http://proceedings.spiedigitallibrary,doi:10.1117/12.537476>.
- [BS10] S. Bianco and R. Schettini. Two new von Kries based chromatic adaptation transforms found by numerical optimization. *Color Research & Application*, 35(3):184–192, June 2010. URL: <http://doi.wiley.com/10.1002/col.20573>, doi:10.1002/col.20573.
- [BWDS99] Barry A. Bodhaine, Norman B. Wood, Ellsworth G. Dutton, and James R. Slusser. On Rayleigh Optical Depth Calculations. *Journal of Atmospheric and Oceanic Technology*, 16(11):1854–1861, November 1999. URL: <http://journals.ametsoc.org/doi/abs/10.1175/1520-0426%281999%29016%3C1854%3AORODC%3E2.0.CO%3B2>, doi:10.1175/1520-0426(1999)016<1854:ORODC>2.0.CO;2.
- [Bor17] Tim Borer. Private Discussion with Mansencal, T. and Shaw, N. 2017.
- [Boua] Paul Bourke. Intersection point of two line segments in 2 dimensions. URL: <http://paulbourke.net/geometry/pointlineplane/>.

- [Boub] Paul Bourke. Trilinear Interpolation. URL: <http://paulbourke.net/miscellaneous/interpolation/>.
- [Bre87] Edwin J. Breneman. Corresponding chromaticities for different states of adaptation to complex visual fields. *Journal of the Optical Society of America A*, 4(6):1115, June 1987. URL: <https://www.osapublishing.org/abstract.cfm?URI=josaa-4-6-1115><http://www.opticsinfobase.org/josaa/fulltext.cfm?uri=josaa-4-6-1115\T1\textbackslash{}&id=2783>, doi:10.1364/JOSAA.4.001115.
- [BS08] Michael H. Brill and Sabine Susstrunk. Repairing gamut problems in CIECAM02: A progress report. *Color Research & Application*, 33(5):424–426, October 2008. URL: <http://doi.wiley.com/10.1002/col.20432>, doi:10.1002/col.20432.
- [Bro09] A. D. Broadbent. Calculation from the Original Experimental Data of the Cie 1931 RGB Standard Observer Spectral Chromaticity Co-Ordinates and Color Matching Functions. 2009. URL: http://www.cis.rit.edu/research/mcsl2/research/broadbent/CIE1931_RGB.pdf<http://www.cis.rit.edu/mcsl/research/1931.php>.
- [BB09] Wilhelm Burger and Mark James Burge. *Principles of Digital Image Processing*. Undergraduate Topics in Computer Science. Springer London, London, 2009. ISBN 978-1-84800-194-7. URL: <http://link.springer.com/10.1007/978-1-84800-195-4>, doi:10.1007/978-1-84800-195-4.
- [Can14] Canon. EOS C500 Firmware Update. 2014. URL: <https://www.usa.canon.com/internet/portal/us/home/explore/product-showcases/cameras-and-lenses/cinema-eos-firmware/c500>.
- [Can16] Canon. EOS C300 Mark II - EOS C300 Mark II Input Transform Version 2.0 (for Cinema Gamut / BT.2020). 2016. URL: <https://www.usa.canon.com/internet/portal/us/home/support/details/cameras/cinema-eos/eos-c300-mark-ii>.
- [CTS13] Renbo Cao, H Joel Trussell, and Renzo Shamey. Comparison of the performance of inverse transformation methods from OSA-UCS to CIEXYZ. *Journal of the Optical Society of America A*, 30(8):1508, August 2013. URL: <https://www.osapublishing.org/abstract.cfm?URI=josaa-30-8-1508>, doi:10.1364/JOSAA.30.001508.
- [Cas14] Saullo Castro. Numpy: Fastest way of computing diagonal for each row of a 2d array. 2014. URL: <http://stackoverflow.com/questions/26511401/numpy-fastest-way-of-computing-diagonal-for-each-row-of-a-2d-array/26517247#26517247>.
- [Cen] Paul Centore. The Munsell and Kubelka-Munk Toolbox. URL: <http://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html>.
- [Cen12] Paul Centore. An open-source inversion algorithm for the Munsell renotation. *Color Research & Application*, 37(6):455–464, December 2012. URL: <http://doi.wiley.com/10.1002/col.20715>, doi:10.1002/col.20715.
- [Cen14a] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - GeneralRoutines/CIELABtoApproxMunsellSpec.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14b] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/ChromDiagHueAngleToMunsellHue.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14c] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/FindHueOnRenotationOvoid.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.

- [Cen14d] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MaxChromaForExtrapolatedRenotation.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14e] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToASTMHue.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14f] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToChromDiagHueAngle.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14g] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyForIntegerMunsellValue.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14h] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyY.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14i] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/xyYtoMunsell.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14j] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/BoundingRenotationHues.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cen14k] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/LinearVsRadialInterpOnRenotationOvoid.m. 2014. URL: <https://github.com/colour-science/MunsellAndKubelkaMunkToolbox>.
- [Cha15] Peter Chamberlain. LUT documentation (to create from another program). 2015. URL: <https://forum.blackmagicdesign.com/viewtopic.php?f=21&t=40284#p232952>.
- [CWCR04] Vien Cheung, Stephen Westland, David Connah, and Caterina Ripamonti. A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms. *Coloration Technology*, 120(1):19–25, 2004. doi:10.1111/j.1478-4408.2004.tb00201.x.
- [CIE] CIE. CIE Spectral Data. URL: <http://files.cie.co.at/204.xls>.
- [CIE04] CIE. CIE 15:2004 Tables Data. 2004. URL: <https://law.resource.org/pub/us/cfr/ibr/003/cie.15.2004.tables.xls>.
- [Cola] Colblindor. Deuteranopia - Red-Green Color Blindness. URL: <http://www.color-blindness.com/deuteranopia-red-green-color-blindness/>.
- [Colb] Colblindor. Protanopia - Red-Green Color Blindness. URL: <http://www.color-blindness.com/protanopia-red-green-color-blindness/>.
- [Colc] Colblindor. Tritanopia - Blue-Yellow Color Blindness. URL: <http://www.color-blindness.com/tritanopia-blue-yellow-color-blindness/>.
- [Cot] Russell Cottrell. The Russell RGB working color space. URL: <http://www.russellcottrell.com/photo/downloads/RussellRGB.icc>.
- [CKM+04] Matthew Cowan, Glenn Kennel, Thomas Maier, Brad Walker, and Matthew Cowan. Constant Sensitivity Experiment to Determine the Bit Depth for Digital Cinema. 2004. URL: <http://car.france3.mars.free.fr/FormationINAHD/HDTV/HDTV2007v35/SMPTEnormesetconfs/Contrastm.pdf>.

- [CVRa] CVRL. CIE (2012) 10-deg XYZ “physiologically-relevant” colour matching functions. URL: <http://www.cvrl.org/database/text/cienewxyz/cie2012xyz10.htm>.
- [CVRb] CVRL. CIE (2012) 2-deg XYZ “physiologically-relevant” colour matching functions. URL: <http://www.cvrl.org/database/text/cienewxyz/cie2012xyz2.htm>.
- [CVRc] CVRL. Luminous efficiency. URL: <http://www.cvrl.org/lumindex.htm>.
- [CVRd] CVRL. New CIE XYZ functions transformed from the CIE (2006) LMS functions. URL: <http://cvrl.ioo.ucl.ac.uk/ciexyzpr.htm>.
- [CVRe] CVRL. Older CIE Standards. URL: <http://cvrl.ioo.ucl.ac.uk/cie.htm>.
- [CVRf] CVRL. Stiles & Burch individual 10-deg colour matching data. URL: http://www.cvrl.org/stilesburch10_ind.htm.
- [CVRg] CVRL. Stiles & Burch individual 2-deg colour matching data. URL: http://www.cvrl.org/stilesburch2_ind.htm.
- [DFGM15] Maryam Mohammadzadeh Darrodi, Graham Finlayson, Teresa Goodman, and Michal Mackiewicz. Reference data set for camera spectral sensitivity estimation. *Journal of the Optical Society of America A*, 32(3):381, March 2015. URL: <https://www.osapublishing.org/abstract.cfm?URI=josaa-32-3-381>, doi:10.1364/JOSAA.32.000381.
- [DO10] Wendy Davis and Yoshiro Ohno. Color quality scale. *Optical Engineering*, 49(3):033602, March 2010. URL: <http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.3360335>, doi:10.1117/1.3360335.
- [Dji17] Dji. White Paper on D-Log and D-Gamut of DJI Cinema Color System. 2017. URL: https://dl.djicdn.com/downloads/zenmuse+x7/20171010/D-Log_D-Gamut_Whitepaper.pdf.
- [Dol16] Dolby. WHAT IS ICTCP? - INTRODUCTION. 2016. URL: <https://www.dolby.com/us/en/technologies/dolby-vision/ICtCp-white-paper.pdf>.
- [Easa] EasyRGB. CMY $\rightarrow$ CMYK. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=13#text13>.
- [Easb] EasyRGB. CMY $\rightarrow$ RGB. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=12#text12>.
- [Easc] EasyRGB. CMYK $\rightarrow$ CMY. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=14#text14>.
- [Easd] EasyRGB. HSL $\rightarrow$ RGB. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=19#text19>.
- [Ease] EasyRGB. HSV $\rightarrow$ RGB. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=21#text21>.
- [Easf] EasyRGB. RGB $\rightarrow$ CMY. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=11#text11>.
- [Easg] EasyRGB. RGB $\rightarrow$ HSL. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=18#text18>.
- [Eash] EasyRGB. RGB $\rightarrow$ HSV. URL: <http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=20#text20>.
- [Erda] U. Murat Erdem. Fast Line Segment Intersection. URL: <http://www.mathworks.com/matlabcentral/fileexchange/27205-fast-line-segment-intersection>.
- [Erdb] Turan Erdogan. How to Calculate Luminosity, Dominant Wavelength, and Excitation Purity. URL: http://www.semrock.com/Data/Sites/1/semrockpdfs/whitepaper_howtocalculateluminositywavelengthandpurity.pdf.

- [FW98] M. Fairchild and D. Wyble. Colorimetric Characterization of The Apple Studio Display (flat panel LCD). 1998. URL: <http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1922\T1\textbackslash{}&context=article>.
- [FC11] Mark D Fairchild and Ping-hsu Chen. Brightness, lightness, and specifying color in high-dynamic-range scenes and images. In Susan P. Farnand and Frans Gaykema, editors, *Proc. SPIE 7867, Image Quality and System Performance VIII*, 78670O. January 2011. URL: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.872075>, doi:10.1117/12.872075.
- [Fai] Mark D. Fairchild. Fairchild YSh. URL: <http://rit-mcsl.org/fairchild//files/FairchildYSh.zip>.
- [Fai91] Mark D. Fairchild. Formulation and testing of an incomplete-chromatic-adaptation model. *Color Research & Application*, 16(4):243–250, August 1991. URL: <http://doi.wiley.com/10.1002/col.5080160406>, doi:10.1002/col.5080160406.
- [Fai96] Mark D. Fairchild. Refinement of the RLAB color space. *Color Research & Application*, 21(5):338–346, October 1996. URL: [http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199610%2921%3A5%3C338%3A%3AAID-COL3%3E3.0.CO%3B2-Z,doi:10.1002/\(SICI\)1520-6378\(199610\)21:5<338::AID-COL3>3.0.CO;2-Z](http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199610%2921%3A5%3C338%3A%3AAID-COL3%3E3.0.CO%3B2-Z,doi:10.1002/(SICI)1520-6378(199610)21:5<338::AID-COL3>3.0.CO;2-Z).
- [Fai04] Mark D. Fairchild. CIECAM02. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter CIECAM02, pages 289–301. Wiley, 2 edition, 2004.
- [Fai13a] Mark D. Fairchild. ATD Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.2, pages 5852–5991. Wiley, 3 edition, 2013.
- [Fai13b] Mark D. Fairchild. Chromatic Adaptation Models. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4179–4252. Wiley, 3 edition, 2013.
- [Fai13c] Mark D. Fairchild. FAIRCHILD’S 1990 MODEL. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4418–4495. Wiley, 3 edition, 2013.
- [Fai13d] Mark D. Fairchild. IPT Colourspace. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 20.3, pages 6197–6223. Wiley, 3 edition, 2013.
- [Fai13e] Mark D. Fairchild. LLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.3, pages 6025–6178. Wiley, 3 edition, 2013.
- [Fai13f] Mark D. Fairchild. The Hunt Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 12, pages 5094–5556. Wiley, 3 edition, 2013.
- [Fai13g] Mark D. Fairchild. The Nayatani et al. Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4810–5085. Wiley, 3 edition, 2013.
- [Fai13h] Mark D. Fairchild. The RLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 13, pages 5563–5824. Wiley, 3 edition, 2013.
- [FW10] Mark D. Fairchild and David R. Wyble. hdr-CIELAB and hdr-IPT: Simple Models for Describing the Color of High-Dynamic-Range and Wide-Color-Gamut Images. In *Proc. of Color and Imaging Conference*, 322–326. 2010. URL: <http://www.ingentaconnect.com/content/ist/cic/2010/00002010/00000001/art00057>.
- [Fai85] Hugh S. Fairman. The calculation of weight factors for tristimulus integration. *Color Research & Application*, 10(4):199–203, 1985. URL: <http://doi.wiley.com/10.1002/col.5080100407>, doi:10.1002/col.5080100407.

- [FBH97] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research & Application*, 22(1):11–23, February 1997. URL: [http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199702%2922%3A1%3C11%3A%3AAID-COL4%3E3.0.CO%3B2-7, doi:10.1002/\(SICI\)1520-6378\(199702\)22:1<11::AID-COL4>3.0.CO;2-7](http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199702%2922%3A1%3C11%3A%3AAID-COL4%3E3.0.CO%3B2-7, doi:10.1002/(SICI)1520-6378(199702)22:1<11::AID-COL4>3.0.CO;2-7).
- [FMH15] Graham D. Finlayson, Michal MacKiewicz, and Anya Hurlbert. Color Correction Using Root-Polynomial Regression. *IEEE Transactions on Image Processing*, 24(5):1460–1470, May 2015. URL: <http://ieeexplore.ieee.org/document/7047834/>, doi:10.1109/TIP.2015.2405336.
- [For18] Alex Forsythe. Private Discussion with Mansencal, T. 2018.
- [GDY+] Hugo Gaggioni, Patel Dhanendra, Jin Yamashita, N. Kawada, K. Endo, and Curtis Clark. S-Log: A new LUT for digital production mastering and interchange applications. URL: http://pro.sony.com/bbsccms/assets/files/mkt/cinema/solutions/slog_manual.pdf.
- [GMRS58] L. G. Glasser, A. H. McKinney, C. D. Reilly, and P. D. Schnelle. Cube-Root Color Coordinate System. *Journal of the Optical Society of America*, 48(10):736, October 1958. URL: <https://www.osapublishing.org/abstract.cfm?URI=josa-48-10-736>, doi:10.1364/JOSA.48.000736.
- [GDM16] GoPro, Haarm-Pieter Duiker, and Thomas Mansencal. Gopro.py. 2016. URL: https://github.com/hpd/OpenColorIO-Configs/blob/master/aces_1.0.3/python/aces_ocio/colourspace/gopro.py.
- [Gut95] S. Lee Guth. Further applications of the ATD model for color vision. In Eric Walowitz, editor, *Proc. SPIE 2414, Device-Independent Color Imaging II*, volume 2414, 12–26. April 1995. URL: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=991324>, doi:10.1117/12.206546.
- [HF98] Radim Halir and Jan Flusser. Numerically Stable Direct Least Squares Fitting Of Ellipses. 1998.
- [HernandezAndresLR99] Javier Hernández-Andrés, Raymond L. Lee, and Javier Romero. Calculating correlated color temperatures across the entire gamut of daylight and skylight chromaticities. *Applied Optics*, 38(27):5703, September 1999. URL: <https://www.osapublishing.org/abstract.cfm?URI=ao-38-27-5703>, doi:10.1364/AO.38.005703.
- [Hol] Joseph Holmes. Ekta Space PS 5. URL: https://www.josephholmes.com/userfiles/Ekta_Space_PS5_JHolmes.zip.
- [Hou15] Jim Houston. Private Discussion with Mansencal, T. 2015.
- [Hun04] R.W.G. Hunt. *The Reproduction of Colour*. The Wiley-IS&T Series in Imaging Science and Technology. John Wiley & Sons, Ltd, Chichester, UK, 6 edition, September 2004. ISBN 9780470024270. URL: <http://doi.wiley.com/10.1002/0470024275, doi:10.1002/0470024275>.
- [Hun08a] HunterLab. Hunter L,a,b Color Scale. 2008. URL: <http://www.hunterlab.se/wp-content/uploads/2012/11/Hunter-L-a-b.pdf>.
- [Hun08b] HunterLab. Illuminant Factors in Universal Software and EasyMatch Coatings. 2008. URL: https://support.hunterlab.com/hc/en-us/article_attachments/201437785/an02_02.pdf.
- [Hun12] HunterLab. Hunter Rd,a,b Color Scale – History and Application. 2012. URL: <https://hunterlabdotcom.files.wordpress.com/2012/07/an-1016-hunter-rd-a-b-color-scale-update-12-07-03.pdf>.
- [Huta] HutchColor. BestRGB (4 K). URL: <http://www.hutchcolor.com/profiles/BestRGB.zip>.
- [Hutb] HutchColor. DonRGB4 (4 K). URL: <http://www.hutchcolor.com/profiles/DonRGB4.zip>.
- [Hutc] HutchColor. MaxRGB (4 K). URL: <http://www.hutchcolor.com/profiles/MaxRGB.zip>.
- [Hutd] HutchColor. XtremeRGB (4 K). URL: <http://www.hutchcolor.com/profiles/XtremeRGB.zip>.

- [KMH+02] Bongsoon Kang, Ohak Moon, Changhee Hong, Honam Lee, Bonghwan Cho, and Youngsun Kim. Design of advanced color: Temperature control system for HDTV applications. *Journal of the Korean Physical Society*, 41(6):865–871, 2002. URL: <http://cat.inist.fr/?aModele=afficheN\T1\textbackslash{}&cpsidt=14448733>.
- [KPK11] Paul Kienzle, Nikunj Patel, and James Krycka. refl1d.numpyerrors - Refl1D v0.6.19 documentation. 2011. URL: http://www.reflectometry.org/danse/docs/refl1d/_modules/refl1d/numpyerrors.html.
- [Kir06] Richard Kirk. Truelight Software Library 2.0. 2006. URL: <https://www.filmlight.ltd.uk/pdf/whitepapers/FL-TL-TN-0057-SoftwareLib.pdf>.
- [Kry85] M Krystek. An algorithm to calculate correlated colour temperature. *Color Research & Application*, 10(1):38–40, 1985. URL: <http://doi.wiley.com/10.1002/col.5080100109>, doi:10.1002/col.5080100109.
- [Lau12] Laurent. Reproducibility of python pseudo-random numbers across systems and versions? 2012. URL: <http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions>.
- [LLW+17] Changjun Li, Zhiqiang Li, Zhifeng Wang, Yang Xu, Ming Ronnier Luo, Guihua Cui, Manuel Melgosa, Michael H Brill, and Michael Pointer. Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS. *Color Research & Application*, 42(6):703–718, December 2017. URL: <http://doi.wiley.com/10.1002/col.22131>, doi:10.1002/col.22131.
- [LLRH02] Changjun Li, Ming Ronnier Luo, Bryan Rigg, and Robert W. G. Hunt. CMC 2000 chromatic adaptation transform: CMCCAT2000. *Color Research & Application*, 27(1):49–58, February 2002. URL: <http://doi.wiley.com/10.1002/col.10005>, doi:10.1002/col.10005.
- [LPLMv07] Changjun Li, Esther Perales, Ming Ronnier Luo, and Francisco Martinez-verdu. The Problem with CAT02 and Its Correction. 2007. URL: <https://pdfs.semanticscholar.org/b5a9/0215ad9a1fb6b01f310b3d64305f7c9feb3a.pdf>.
- [Lin03a] Bruce Lindbloom. Delta E (CIE 1976). 2003. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE76.html.
- [Lin03b] Bruce Lindbloom. XYZ to xyY. 2003. URL: http://www.brucelindbloom.com/Eqn_XYZ_to_xyY.html.
- [Lin09a] Bruce Lindbloom. Chromatic Adaptation. 2009. URL: http://brucelindbloom.com/Eqn_ChromAdapt.html.
- [Lin09b] Bruce Lindbloom. Delta E (CIE 2000). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE2000.html.
- [Lin09c] Bruce Lindbloom. Delta E (CMC). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CMC.html.
- [Lin09d] Bruce Lindbloom. xyY to XYZ. 2009. URL: http://www.brucelindbloom.com/Eqn_xyY_to_XYZ.html.
- [Lin11] Bruce Lindbloom. Delta E (CIE 1994). 2011. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE94.html.
- [Lin14] Bruce Lindbloom. RGB Working Space Information. 2014. URL: <http://www.brucelindbloom.com/WorkingSpaceInfo.html>.
- [Lin15] Bruce Lindbloom. About the Lab Gamut. 2015. URL: <http://www.brucelindbloom.com/LabGamutDisplayHelp.html>.
- [LPY+16] Taoran Lu, Fangjun Pu, Peng Yin, Tao Chen, Walt Husak, Jaclyn Pytlarz, Robin Atkins, Jan Froehlich, and Guan-Ming Su. ITP Colour Space and Its Compression Performance for High Dy-

- namic Range and Wide Colour Gamut Video Distribution. *ZTE Communications*, 14(1):32–38, 2016. URL: <http://www.cnki.net/kcms/detail/34.1294.TN.20160205.1903.006.html>.
- [LCL06] M. Ronnier Luo, Guihua Cui, and Changjun Li. Uniform colour spaces based on CIECAM02 colour appearance model. *Color Research & Application*, 31(4):320–330, August 2006. URL: <http://doi.wiley.com/10.1002/col.20227>, doi:10.1002/col.20227.
- [LL13] Ming Ronnier Luo and Changjun Li. CIECAM02 and Its Recent Developments. In Christine Fernandez-Maloigne, editor, *Advanced Color Image Processing and Analysis*, pages 19–58. Springer New York, New York, NY, 2013. URL: <http://link.springer.com/10.1007/978-1-4419-6190-7>, doi:10.1007/978-1-4419-6190-7.
- [LLK96] Ming Ronnier Luo, Mei-Chun Lo, and Wen-Guey Kuo. The LLAB (l:c) colour model. *Color Research & Application*, 21(6):412–429, December 1996. URL: [http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199612%2921%3A6%3C412%3A%3AAID-COL4%3E3.0.CO%3B2-Z](http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199612%2921%3A6%3C412%3A%3AAID-COL4%3E3.0.CO%3B2-Z, doi:10.1002/(SICI)1520-6378(199612)21:6<412::AID-COL4>3.0.CO;2-Z), doi:10.1002/(SICI)1520-6378(199612)21:6<412::AID-COL4>3.0.CO;2-Z.
- [LM96] Ming Ronnier Luo and Ján Morovic. Two Unsolved Issues in Colour Management – Colour Appearance and Gamut Mapping. In *Conference: 5th International Conference on High Technology: Imaging Science and Technology – Evolution & Promise*, 136–147. 1996. URL: http://www.researchgate.net/publication/236348295_Two_Unsolved_Issues_in_Colour_Management_Colour_Appearance_and_Gamut_Mapping.
- [Mac35] David L. MacAdam. Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America*, 25(11):361–367, November 1935. URL: [http://www.opticsinfobase.org/abstract.cfm?URI=josa-25-11-361](http://www.opticsinfobase.org/abstract.cfm?URI=josa-25-11-361, doi:10.1364/JOSA.25.000361), doi:10.1364/JOSA.25.000361.
- [Mac42] David L. Macadam. Visual Sensitivities to Color Differences in Daylight. *Journal of the Optical Society of America*, 32(5):28, 1942. doi:10.1364/JOSA.32.000247.
- [MOF09] G.M. Machado, M.M. Oliveira, and L. Fernandes. A Physiologically-based Model for Simulation of Color Vision Deficiency. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1291–1298, November 2009. URL: [http://ieeexplore.ieee.org/document/5290741/](http://ieeexplore.ieee.org/document/5290741/, doi:10.1109/TVCG.2009.113), doi:10.1109/TVCG.2009.113.
- [Mac10] Gustavo Mello Machado. A model for simulation of color vision deficiency and a color contrast enhancement technique for dichromats. 2010. URL: <http://www.lume.ufrgs.br/handle/10183/26950>.
- [MS03] Henrique Malvar and Gary Sullivan. YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range. 2003. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/Malvar_Sullivan_YCoCg-R_JVT-I014r3-2.pdf.
- [Mana] Thomas Mansencal. Lookup. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.
- [Manb] Thomas Mansencal. Structure. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.
- [Man15] Thomas Mansencal. RED Colourspace Derivation. 2015. URL: <https://www.colour-science.org/posts/red-colourspace-derivation>.
- [Man18] Thomas Mansencal. How is the visible gamut bounded? 2018. URL: <https://stackoverflow.com/a/48396021/931625>.
- [Mel13] Manuel Melgosa. CIE / ISO new standard: CIEDE2000. 2013. URL: http://www.color.org/events/colorimetry/Melgosa_CIEDE2000_Workshop-July4.pdf.
- [MSHD15] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically Meaningful Rendering using Tristimulus Colours. *Computer Graphics Forum*, 34(4):31–40, July 2015. URL: [http://doi.wiley.com/10.1111/cgf.12676](http://doi.wiley.com/10.1111/cgf.12676, doi:10.1111/cgf.12676), doi:10.1111/cgf.12676.

- [Mil14] Scott Miller. A Perceptual EOTF for Extended Dynamic Range Imagery. 2014. URL: <https://www.smpite.org/sites/default/files/2014-05-06-EOTF-Miller-1-2-handout.pdf>.
- [Mor03] Nathan Moroney. A radial sampling of the OSA uniform color scales. *Color and Imaging Conference*, pages 1–14, 2003. URL: <http://www.ingentaconnect.com/content/ist/cic/2003/00002003/00000001/art00031>.
- [MFH+02] Nathan Moroney, Mark D. Fairchild, Robert W. G. Hunt, Changjun Li, Ming Ronnier Luo, and Todd Newman. The CIECAM02 color appearance model. *Color and Imaging Conference*, pages 23–27, 2002. URL: <http://www.ingentaconnect.com/content/ist/cic/2002/00002002/00000001/art00006>.
- [Nat16] Graeme Nattress. Private Discussion with Shaw, N. 2016.
- [NSY95] Yoshinobu Nayatani, Hiroaki Sobagaki, and Kenjiro Hashimoto Tadashi Yano. Lightness dependency of chroma scales of a nonlinear color-appearance model and its latest formulation. *Color Research & Application*, 20(3):156–167, June 1995. URL: <http://doi.wiley.com/10.1002/col.5080200305>, doi:10.1002/col.5080200305.
- [NNJ43] Sidney M. Newhall, Dorothy Nickerson, and Deane B. Judd. Final Report of the OSA Subcommittee on the Spacing of the Munsell Colors. *Journal of the Optical Society of America*, 33(7):385, July 1943. URL: <https://www.osapublishing.org/abstract.cfm?URI=josa-33-7-385>, doi:10.1364/JOSA.33.000385.
- [Ohn05] Yoshi Ohno. Spectral design considerations for white LED color rendering. *Optical Engineering*, 44(11):111302, 2005. URL: <http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.2130694>, doi:10.1117/1.2130694.
- [Ohn14] Yoshiro Ohno. Practical Use and Calculation of CCT and Duv. *LEUKOS*, 10(1):47–55, January 2014. URL: <http://www.tandfonline.com/doi/abs/10.1080/15502724.2014.839020>, doi:10.1080/15502724.2014.839020.
- [OD08] Yoshiro Ohno and Wendy Davis. NIST CQS simulation 7.4. 2008. URL: <https://drive.google.com/file/d/1PsuU6QjUJjCX6tQyCud6ul2Tbs8rYWW9/view?usp=sharing>.
- [OD13] Yoshiro Ohno and Wendy Davis. NIST CQS simulation 9.0. 2013. URL: <https://www.researchgate.net/file.PostFileLoader.html?id=5541c498f15bc7cc2c8b4578\T1\textbackslash{}&assetKey=AS%3A273582771376136%401442238623549>.
- [Oht97] N. Ohta. The basis of color reproduction engineering. 1997.
- [Pan14] Panasonic. VARICAM V-Log/V-Gamut. 2014. URL: http://pro-av.panasonic.net/en/varicam/common/pdf/VARICAM_V-Log_V-Gamut.pdf.
- [Poi80] Michael R. Pointer. Pointer’s Gamut Data. 1980. URL: <http://www.cis.rit.edu/research/mcsl2/online/PointerData.xls>.
- [Rei] Kenneth Reitz. CaseInsensitiveDict. URL: <https://github.com/kennethreitz/requests/blob/v1.2.3/requests/structures.py#L37>.
- [Sae] Saeedn. Extend a line segment a specific distance. URL: <http://stackoverflow.com/questions/7740507/extend-a-line-segment-a-specific-distance>.
- [SCKL17] Muhammad Safdar, Guihua Cui, Youn Jin Kim, and Ming Ronnier Luo. Perceptually uniform color space for image signals including high dynamic range and wide gamut. *Optics Express*, 25(13):15131, June 2017. URL: <https://www.osapublishing.org/abstract.cfm?URI=oe-25-13-15131>, doi:10.1364/OE.25.015131.
- [Sas] Sastanin. How to make scipy.interpolate give an extrapolated result beyond the input range? URL: <http://stackoverflow.com/a/2745496/931625>.

- [SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, February 2005. URL: <http://doi.wiley.com/10.1002/col.20070>, doi:10.1002/col.20070.
- [SH15] Peter Shirley and David Hart. The prismatic color space for rgb computations. 2015.
- [Sir18] Daniele Siragusano. Private Discussion with Shaw, Nick. 2018.
- [Smi78] Alvy Ray Smith. Color gamut transform pairs. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78*, SIGGRAPH '78, 12–19. New York, New York, USA, 1978. ACM Press. URL: <http://portal.acm.org/citation.cfm?doid=800248.807361>, doi:10.1145/800248.807361.
- [Smi99] Brian Smits. An RGB-to-Spectrum Conversion for Reflectances. *Journal of Graphics Tools*, 4(4):11–22, January 1999. URL: <http://www.tandfonline.com/doi/abs/10.1080/10867651.1999.10487511>, doi:10.1080/10867651.1999.10487511.
- [SWG00] K E Spaulding, G J Woolfe, and E J Giorgianni. Reference Input/Output Medium Metric RGB Color Encodings (RIMM/ROMM RGB). 2000. URL: <http://www.photo-lovers.org/pdf/color/romm.pdf>.
- [Spi15] Nick Spiker. Private Discussion with Mansencal, T. 2015. URL: <http://www.invisiblelightimages.com/>.
- [SS88] E. I. Stearns and R. E. Stearns. An example of a method for correcting radiance data for Bandpass error. *Color Research & Application*, 13(4):257–259, August 1988. URL: <http://doi.wiley.com/10.1002/col.5080130410>, doi:10.1002/col.5080130410.
- [SS00] Andrew Stockman and Lindsay T. Sharpe. Cone Fundamentals. 2000. URL: <http://www.cvrl.org/cones.htm>.
- [SBS99] Sabine Susstrunk, Robert Buckley, and Steve Swen. Standard RGB Color Spaces. 1999.
- [SHF00] Sabine E. Susstrunk, Jack M. Holm, and Graham D. Finlayson. Chromatic adaptation performance of different RGB sensors. In Reiner Eschbach and Gabriel G. Marcu, editors, *Photonics West 2001 - Electronic Imaging*, volume 4300, 172–183. December 2000. URL: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=903890>, doi:10.1117/12.410788.
- [Tho12] Larry Thorpe. CANON-LOG TRANSFER CHARACTERISTIC. 2012. URL: http://downloads.canon.com/CDLC/Canon-Log_Transfer_Characteristic_6-20-2012.pdf.
- [Tri15] Tashi Trieu. Private Discussion with Mansencal, T. 2015.
- [War16] Greg Ward. Private Discussion with Mansencal, T. 2016.
- [WEV02] Greg Ward and Elena Eydelberg-Vileshin. Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries. *Eurographics workshop on Rendering*, pages 117–124, 2002. URL: <http://portal.acm.org/citation.cfm?id=581896.581913%5Cnpapers2://publication/uuid/72F077CB-2366-40E4-901E-5FE35B85BAD6http://dl.acm.org/citation.cfm?id=581913>, doi:10.2312/EGWR/EGWR02/117-124.
- [WR04] Stephen Westland and Caterina Ripamonti. Table 8.2. In *Computational Colour Science Using MATLAB*, pages 137. John Wiley & Sons, Ltd, Chichester, UK, 1 edition, March 2004. URL: <http://doi.wiley.com/10.1002/0470020326>, doi:10.1002/0470020326.
- [WRC12a] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT2000. In *Computational Colour Science Using MATLAB*, chapter 6.2.3, pages 83–86. 2 edition, 2012.
- [WRC12b] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT97. In *Computational Colour Science Using MATLAB*, chapter 6.2.2, pages 80. 2 edition, 2012.

- [WRC12c] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Correction for Spectral Bandpass. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.
- [WRC12d] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Extrapolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.
- [WRC12e] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Interpolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.3, pages 29–37. 2 edition, 2012.
- [Wik] Wikipedia. Ellipse. URL: <https://en.wikipedia.org/wiki/Ellipse>.
- [Wik01a] Wikipedia. Approximation. 2001. URL: http://en.wikipedia.org/wiki/Color_temperature#Approximation.
- [Wik01b] Wikipedia. Color temperature. 2001. URL: http://en.wikipedia.org/wiki/Color_temperature.
- [Wik01c] Wikipedia. Luminance. 2001. URL: <https://en.wikipedia.org/wiki/Luminance>.
- [Wik01d] Wikipedia. Rayleigh scattering. 2001. URL: http://en.wikipedia.org/wiki/Rayleigh_scattering.
- [Wik03a] Wikipedia. HSL and HSV. 2003. URL: http://en.wikipedia.org/wiki/HSL_and_HSV.
- [Wik03b] Wikipedia. Lagrange polynomial - Definition. 2003. URL: https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition.
- [Wik03c] Wikipedia. Luminosity function. 2003. URL: https://en.wikipedia.org/wiki/Luminosity_function#Details.
- [Wik03d] Wikipedia. Mean squared error. 2003. URL: https://en.wikipedia.org/wiki/Mean_squared_error.
- [Wik03e] Wikipedia. Michaelis–Menten kinetics. 2003. URL: https://en.wikipedia.org/wiki/Michaelis-Menten_kinetics.
- [Wik03f] Wikipedia. Vandermonde matrix. 2003. URL: https://en.wikipedia.org/wiki/Vandermonde_matrix.
- [Wik04a] Wikipedia. Peak signal-to-noise ratio. 2004. URL: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.
- [Wik04b] Wikipedia. Surfaces. 2004. URL: <http://en.wikipedia.org/wiki/Gamut#Surfaces>.
- [Wik04c] Wikipedia. Whiteness. 2004. URL: <http://en.wikipedia.org/wiki/Whiteness>.
- [Wik04d] Wikipedia. Wide-gamut RGB color space. 2004. URL: http://en.wikipedia.org/wiki/Wide-gamut_RGB_color_space.
- [Wik04e] Wikipedia. YCbCr. 2004. URL: <https://en.wikipedia.org/wiki/YCbCr>.
- [Wik05a] Wikipedia. CIE 1931 color space. 2005. URL: http://en.wikipedia.org/wiki/CIE_1931_color_space.
- [Wik05b] Wikipedia. ISO 31-11. 2005. URL: https://en.wikipedia.org/wiki/ISO_31-11.
- [Wik05c] Wikipedia. Lanczos resampling. 2005. URL: https://en.wikipedia.org/wiki/Lanczos_resampling.
- [Wik05d] Wikipedia. Luminous Efficacy. 2005. URL: https://en.wikipedia.org/wiki/Luminous_efficacy.
- [Wik05e] Wikipedia. Mesopic weighting function. 2005. URL: http://en.wikipedia.org/wiki/Mesopic_vision#Mesopic_weighting_function.
- [Wik06a] Wikipedia. List of common coordinate transformations. 2006. URL: http://en.wikipedia.org/wiki/List_of_common_coordinate_transformations.

- [Wik06b] Wikipedia. White points of standard illuminants. 2006. URL: http://en.wikipedia.org/wiki/Standard_illuminant#White_points_of_standard_illuminants.
- [Wik07a] Wikipedia. CAT02. 2007. URL: <http://en.wikipedia.org/wiki/CIECAM02#CAT02>.
- [Wik07b] Wikipedia. CIECAM02. 2007. URL: <http://en.wikipedia.org/wiki/CIECAM02>.
- [Wik07c] Wikipedia. CIELUV. 2007. URL: <http://en.wikipedia.org/wiki/CIELUV>.
- [Wik07d] Wikipedia. Lightness. 2007. URL: <http://en.wikipedia.org/wiki/Lightness>.
- [Wik07e] Wikipedia. The reverse transformation. 2007. URL: http://en.wikipedia.org/wiki/CIELUV#The_reverse_transformation.
- [Wik08a] Wikipedia. CIE 1960 color space. 2008. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space.
- [Wik08b] Wikipedia. CIE 1964 color space. 2008. URL: http://en.wikipedia.org/wiki/CIE_1964_color_space.
- [Wik08c] Wikipedia. Color difference. 2008. URL: http://en.wikipedia.org/wiki/Color_difference.
- [Wik08d] Wikipedia. Relation to CIE XYZ. 2008. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space#Relation_to_CIE_XYZ.
- [Wys63] Günther Wyszecki. Proposal for a New Color-Difference Formula. *Journal of the Optical Society of America*, 53(11):1318, November 1963. URL: <https://www.osapublishing.org/abstract.cfm?URI=josa-53-11-1318>, doi:10.1364/JOSA.53.001318.
- [WS00a] Günther Wyszecki and W S Stiles. Equation I(1.2.1). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 8. Wiley, 2000.
- [WS00b] Günther Wyszecki and W S Stiles. Table 2(5.4.1) MacAdam Ellipses (Observer PGN) Observed and Calculated on the Basis of a Normal Distribution of Color Matches about a Color Center (Silberstein and MacAdam, 1945). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 309. Wiley, 2000.
- [WS00c] Günther Wyszecki and W. S. Stiles. CIE 1976 ($L^*u^*v^*$)-Space and Color-Difference Formula. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 167. Wiley, 2000.
- [WS00d] Günther Wyszecki and W. S. Stiles. CIE Method of Calculating D-Illuminants. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 145–146. Wiley, 2000.
- [WS00e] Günther Wyszecki and W. S. Stiles. DISTRIBUTION TEMPERATURE, COLOR TEMPERATURE, AND CORRELATED COLOR TEMPERATURE. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 224–229. Wiley, 2000.
- [WS00f] Günther Wyszecki and W. S. Stiles. Integration Replaced by Summation. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 158–163. Wiley, 2000.
- [WS00g] Günther Wyszecki and W. S. Stiles. Standard Photometric Observers. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 256–259,395. Wiley, 2000.
- [WS00h] Günther Wyszecki and W. S. Stiles. Table 1(3.11) Isotemperature Lines. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 228. Wiley, 2000.
- [WS00i] Günther Wyszecki and W. S. Stiles. Table 1(3.3.3). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 138–139. Wiley, 2000.
- [WS00j] Günther Wyszecki and W. S. Stiles. Table I(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 776–777. Wiley, 2000.

- [WS00k] Günther Wyszecki and W. S. Stiles. Table I(6.5.3) Whiteness Formulae (Whiteness Measure Denoted by W). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 837–839. Wiley, 2000.
- [WS00l] Günther Wyszecki and W. S. Stiles. Table II(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 778–779. Wiley, 2000.
- [WS00m] Günther Wyszecki and W. S. Stiles. The CIE 1964 Standard Observer. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 141. Wiley, 2000.
- [XR15] X-Rite. New color specifications for ColorChecker SG and Classic Charts. 2015. URL: http://xritephoto.com/ph_product_overview.aspx?ID=938\T1\textbackslash{}&Action=Support\T1\textbackslash{}&SupportID=5884#.
- [XRP12] X-Rite and Pantone. Color iQC and Color iMatch Color Calculations Guide. 2012. URL: https://www.xrite.com/-/media/xrite/files/apps_engineering_techdocuments/c/09_color_calculations_en.pdf.
- [Yor14] Rory Yorke. Python: Change format of np.array or allow tolerance in in1d function. 2014. URL: <http://stackoverflow.com/a/23521245/931625>.
- [AdobeSystems05] Adobe Systems. Adobe RGB (1998) Color Image Encoding. 2005. URL: <http://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf>.
- [AdobeSystems13a] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::Set_xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.
- [AdobeSystems13b] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.
- [AdobeSystems13c] Adobe Systems. Cube LUT Specification. 2013. URL: https://drive.google.com/open?id=143Eh08ZYncAMwJ1q4gWxVOqR_OSWYvs.
- [AssociationoRIaBusinesses15] Association of Radio Industries and Businesses. Essential Parameter Values for the Extended Image Dynamic Range Television (EIDRTV) System for Programme Production. 2015. URL: https://www.arib.or.jp/english/std_tr/broadcasting/desc/std-b67.html.
- [ASTMInternational89] ASTM International. ASTM D1535-89 - Standard Practice for Specifying Color by the Munsell System. 1989. URL: <http://www.astm.org/DATABASE.CART/HISTORICAL/D1535-89.htm>.
- [ASTMInternational07] ASTM International. ASTM D2244-07 - Standard Practice for Calculation of Color Tolerances and Color Differences from Instrumentally Measured Color Coordinates. 2007.
- [ASTMInternational08] ASTM International. ASTM D1535-08e1 - Standard Practice for Specifying Color by the Munsell System. 2008. doi:10.1520/D1535-08E01.
- [ASTMInternational11] ASTM International. ASTM E2022-11 - Standard Practice for Calculation of Weighting Factors for Tristimulus Integration. 2011. doi:10.1520/E2022-11.
- [ASTMInternational15] ASTM International. ASTM E308-15 - Standard Practice for Computing the Colors of Objects by Using the CIE System. 2015. doi:10.1520/E0308-15.
- [CIET13294] CIE TC 1-32. CIE 109-1994 A Method of Predicting Corresponding Colours under Different Chromatic and Illuminance Adaptations. 1994. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=34.
- [CIET13606] CIE TC 1-36. CIE 170-1:2006 Fundamental Chromaticity Diagram with Physiological Axes - Part 1. 2006. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=48.

- [CIET13805a] CIE TC 1-38. 9. INTERPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9, pages 14–19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.
- [CIET13805b] CIE TC 1-38. 9.2.4 Method of interpolation for uniformly spaced independent variable. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9.2.4, pages 1–27. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.
- [CIET13805c] CIE TC 1-38. EXTRAPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 10, pages 19–20. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.
- [CIET13805d] CIE TC 1-38. Table V. Values of the c-coefficients of Equ.s 6 and 7. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter Table V, pages 19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.
- [CIET14804a] CIE TC 1-48. 3.1 Recommendations concerning standard physical data of illuminants. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 3.1, pages 12–13. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804b] CIE TC 1-48. 9.1 Dominant wavelength and purity. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.1, pages 32–33. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804c] CIE TC 1-48. APPENDIX E. INFORMATION ON THE USE OF PLANCK'S EQUATION FOR STANDARD AIR. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter APPENDIX E, pages 77–82. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804d] CIE TC 1-48. *CIE 015:2004 Colorimetry, 3rd Edition*. Commission internationale de l'éclairage, 2004. ISBN 978-3-901-90633-6. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804e] CIE TC 1-48. CIE 1976 uniform chromaticity scale diagram (UCS diagram). In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.1, pages 24. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804f] CIE TC 1-48. CIE 1976 uniform colour spaces. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.2, pages 24. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804g] CIE TC 1-48. EXPLANATORY COMMENTS - 5. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 68–68. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804h] CIE TC 1-48. Extrapolation. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 7.2.2.1, pages 24. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [CIET14804i] CIE TC 1-48. The evaluation of whiteness. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.4, pages 24. 2004. URL: <http://www.cie.co.at/publications/colorimetry-3rd-edition>.
- [DigitalCInitiatives07] Digital Cinema Initiatives. Digital Cinema System Specification - Version 1.1. 2007. URL: http://www.dcinovies.com/archives/spec_v1_1/DCI_DCinema_System_Spec_v1_1.pdf.
- [EuropeanCInitiative02] European Color Initiative. ECI RGB v2. 2002. URL: http://www.eci.org/_media/downloads/icc_profiles_from_eci/ecirgbv20.zip.
- [FiLMiCInc17] FiLMiC Inc. FiLMiC Pro - User Manual v6 - Revision 1. 2017. URL: <http://www.filmicpro.com/FilmicProUserManualv6.pdf>.
- [HewlettPDCompany09] Hewlett-Packard Development Company. Understanding the HP DreamColor LP2480zx DCI-P3 Emulation Color Space. 2009. URL: <http://www.hp.com/united-states/campaigns/workstations/pdfs/lp2480zx-dci-p3-emulation.pdf>.

- [IESCommitteeTM2714WGroup14] IES Computer Committee and TM-27-14 Working Group. IES Standard Format for the Electronic Transfer of Spectral Data Electronic Transfer of Spectral Data. 2014.
- [InternationalECommission99] International Electrotechnical Commission. IEC 61966-2-1:1999 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. 1999. URL: <https://webstore.iec.ch/publication/6169>.
- [InternationalTUnion98] International Telecommunication Union. Recommendation ITU-R BT.470-6 - CONVENTIONAL TELEVISION SYSTEMS. 1998. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.470-6-199811-S!!PDF-E.pdf.
- [InternationalTUnion11a] International Telecommunication Union. Recommendation ITU-R BT.1886 - Reference electro-optical transfer function for flat panel displays used in HDTV studio production BT Series Broadcasting service. 2011. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1886-0-201103-I!!PDF-E.pdf.
- [InternationalTUnion11b] International Telecommunication Union. Recommendation ITU-R BT.601-7 - Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. 2011. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf.
- [InternationalTUnion11c] International Telecommunication Union. Recommendation ITU-T T.871 - Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF). 2011. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e\T1\textbackslash{}&id=T-REC-T.871-201105-I!!PDF-E\T1\textbackslash{}&type=items.
- [InternationalTUnion15a] International Telecommunication Union. Recommendation ITU-R BT.2020 - Parameter values for ultra-high definition television systems for production and international programme exchange. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2020-2-201510-I!!PDF-E.pdf.
- [InternationalTUnion15b] International Telecommunication Union. Recommendation ITU-R BT.709-6 - Parameter values for the HDTV standards for production and international programme exchange BT Series Broadcasting service. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf.
- [InternationalTUnion15c] International Telecommunication Union. Report ITU-R BT.2246-4 - The present state of ultra-high definition television BT Series Broadcasting service. 2015.
- [InternationalTUnion16] International Telecommunication Union. Recommendation ITU-R BT.2100-1 - Image parameter values for high dynamic range television for use in production and international programme exchange. 2016. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2100-1-201706-I!!PDF-E.pdf.
- [MunsellCSsciencea] Munsell Color Science. Macbeth Colorchecker. URL: <http://www.rit-mcsl.org/UsefulData/MacbethColorChecker.xls>.
- [MunsellCSscienceb] Munsell Color Science. Munsell Colours Data. URL: <http://www.cis.rit.edu/research/mcsl2/online/munsell.php>.
- [NationalEMAssociation04] National Electrical Manufacturers Association. Digital Imaging and Communications in Medicine (DICOM) Part 14: Grayscale Standard Display Function. 2004. URL: http://medical.nema.org/dicom/2004/04_14PU.PDF.
- [RenewableRDCenter03] Renewable Resource Data Center. Reference Solar Spectral Irradiance: ASTM G-173. 2003. URL: <http://rredc.nrel.gov/solar/spectra/am1.5/ASTMG173/ASTMG173.html>.
- [RisingSResearch] Rising Sun Research. cineSpace LUT Library. URL: <https://sourceforge.net/projects/cinespacelutlib/>.

- [SocietyoMPaTEngineers93] Society of Motion Picture and Television Engineers. *RP 177:1993 : Derivation of Basic Television Color Equations*. Volume RP 177:199. The Society of Motion Picture and Television Engineers, January 1993. ISBN 978-1-61482-191-5. URL: <http://standards.smpte.org/lookup/doi/10.5594/S9781614821915>, doi:10.5594/S9781614821915.
- [SocietyoMPaTEngineers99] Society of Motion Picture and Television Engineers. ANSI/SMPTE 240M-1995 - Signal Parameters - 1125-Line High-Definition Production Systems. 1999. URL: <http://car.france3.mars.free.fr/HD/INA-26jan06/SMPTEnormesetconfs/s240m.pdf>.
- [SocietyoMPaTEngineers04] Society of Motion Picture and Television Engineers. *RP 145:2004: SMPTE C Color Monitor Colorimetry*. Volume RP 145:200. The Society of Motion Picture and Television Engineers, January 2004. ISBN 978-1-61482-164-9. URL: <http://standards.smpte.org/lookup/doi/10.5594/S9781614821649>, doi:10.5594/S9781614821649.
- [SocietyoMPaTEngineers14] Society of Motion Picture and Television Engineers. SMPTE ST 2084:2014 - Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. 2014. URL: <http://www.techstreet.com/products/1883436>, doi:10.5594/SMPTE.ST2084.2014.
- [SonyCorporationa] Sony Corporation. S-Gamut3_S-Gamut3Cine_Matrix.xlsx. URL: https://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/3/S-Gamut3_S-Gamut3Cine_Matrix.xlsx.
- [SonyCorporationb] Sony Corporation. S-Log Whitepaper. URL: http://www.theodoropoulos.info/attachments/076_onS-Log.pdf.
- [SonyCorporationc] Sony Corporation. Technical Summary for S-Gamut3.Cine/S-Log3 and S-Gamut3/S-Log3. URL: http://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/2/TechnicalSummary_for_S-Gamut3Cine_S-Gamut3_S-Log3_V1_00.pdf.
- [SonyCorporation12] Sony Corporation. S-Log2 Technical Paper. 2012. URL: https://pro.sony.com/bbsccms/assets/files/micro/dmpc/training/S-Log2_Technical_PaperV1_0.pdf.
- [SonyImageworks12] Sony Imageworks. Make.py. 2012. URL: <https://github.com/imageworks/OpenColorIO-Configs/blob/master/nuke-default/make.py>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSSubcommittee] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Academy Color Encoding System. URL: <http://www.oscars.org/science-technology/council/projects/aces.html>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSSubcommittee14a] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2013-001 - ACESproxy, an Integer Log Encoding of ACES Image Data. 2014. URL: <https://github.com/ampas/aces-dev/tree/master/documents>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSSubcommittee14b] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-003 - ACEScc, A Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2014. URL: <https://github.com/ampas/aces-dev/tree/master/documents>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSSubcommittee14c] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-004 - Informative Notes on SMPTE ST 2065-1 – Academy Color Encoding Specification (ACES). 2014. URL: <https://github.com/ampas/aces-dev/tree/master/documents>.

- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-012 - Academy Color Encoding System Version 1.0 Component Names. 2014. URL: <https://github.com/ampas/aces-dev/tree/master/documents>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-004 - ACEScg – A Working Space for CGI Render and Compositing. 2015. URL: <https://github.com/ampas/aces-dev/tree/master/documents>.
- [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project. Specification S-2016-001 - ACESct, A Quasi-Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2016. URL: <https://github.com/ampas/aces-dev/tree/v1.0.3/documents>.

Symbols

- `__add__()` (*colour.continuous.AbstractContinuousFunction* method), 246
- `__call__()` (*colour.KernelInterpolator* method), 57
- `__call__()` (*colour.LinearInterpolator* method), 59
- `__call__()` (*colour.NullInterpolator* method), 60
- `__call__()` (*colour.SpragueInterpolator* method), 62
- `__class__()` (*colour.Extrapolator* method), 55
- `__contains__()` (*colour.SpectralShape* method), 132
- `__contains__()` (*colour.continuous.AbstractContinuousFunction* method), 245
- `__contains__()` (*colour.continuous.MultiSignal* method), 252
- `__contains__()` (*colour.continuous.Signal* method), 248
- `__contains__()` (*colour.utilities.CaseInsensitiveMapping* method), 642
- `__contains__()` (in module *colour*), 456
- `__delitem__()` (*colour.LUTSequence* method), 282
- `__delitem__()` (*colour.utilities.CaseInsensitiveMapping* method), 642
- `__delitem__()` (in module *colour*), 456
- `__div__()` (*colour.continuous.AbstractContinuousFunction* method), 246
- `__eq__()` (*colour.LUTSequence* method), 282
- `__eq__()` (*colour.SpectralShape* method), 132
- `__eq__()` (*colour.continuous.AbstractContinuousFunction* method), 245
- `__eq__()` (*colour.continuous.MultiSignal* method), 252
- `__eq__()` (*colour.continuous.Signal* method), 248
- `__eq__()` (*colour.utilities.CaseInsensitiveMapping* method), 642
- `__eq__()` (in module *colour*), 456
- `__getitem__()` (*colour.LUTSequence* method), 282
- `__getitem__()` (*colour.continuous.AbstractContinuousFunction* method), 245
- `__getitem__()` (*colour.continuous.MultiSignal* method), 252
- `__getitem__()` (*colour.continuous.Signal* method), 248
- `__getitem__()` (*colour.utilities.CaseInsensitiveMapping* method), 642
- `__getitem__()` (in module *colour*), 456
- `__hash__()` (*colour.continuous.AbstractContinuousFunction* method), 245
- `__hash__()` (*colour.continuous.MultiSignal* method), 252
- `__hash__()` (*colour.continuous.Signal* method), 248
- `__iadd__()` (*colour.continuous.AbstractContinuousFunction* method), 245
- `__idiv__()` (*colour.continuous.AbstractContinuousFunction* method), 246
- `__imul__()` (*colour.continuous.AbstractContinuousFunction* method), 246
- `__init__()` (*colour.ATD95_Specification* method), 85
- `__init__()` (*colour.CAM16_Specification* method), 93
- `__init__()` (*colour.CIECAM02_Specification* method), 89
- `__init__()` (*colour.Extrapolator* method), 55
- `__init__()` (*colour.Hunt_Specification* method), 97
- `__init__()` (*colour.KernelInterpolator* method), 58
- `__init__()` (*colour.LLAB_Specification* method), 99
- `__init__()` (*colour.LUT1D* method), 277
- `__init__()` (*colour.LUT3D* method), 281
- `__init__()` (*colour.LUT3x1D* method), 279
- `__init__()` (*colour.LUTSequence* method), 283
- `__init__()` (*colour.LinearInterpolator* method), 59
- `__init__()` (*colour.MultiSpectralDistribution* method), 137
- `__init__()` (*colour.Nayatani95_Specification* method), 103
- `__init__()` (*colour.NearestNeighbourInterpolator* method), 58
- `__init__()` (*colour.NullInterpolator* method), 60
- `__init__()` (*colour.PchipInterpolator* method), 61
- `__init__()` (*colour.RGB_Colourspace* method), 361
- `__init__()` (*colour.RLAB_Specification* method), 105
- `__init__()` (*colour.SpectralDistribution* method),

133, 134

`__init__()` (`colour.SpectralDistribution_IESTM2714` method), 300

`__init__()` (`colour.SpectralShape` method), 132

`__init__()` (`colour.SpragueInterpolator` method), 63

`__init__()` (`colour.adaptation.CMCCAT2000_InductionFactors` method), 49

`__init__()` (`colour.algebra.LineSegmentsIntersections_Specification` method), 78

`__init__()` (`colour.algebra.spow_enable` method), 82

`__init__()` (`colour.appearance.CAM16_InductionFactors` method), 94

`__init__()` (`colour.appearance.CIECAM02_InductionFactors` method), 90

`__init__()` (`colour.appearance.LLAB_InductionFactors` method), 101

`__init__()` (`colour.characterisation.ColourChecker` method), 123

`__init__()` (`colour.characterisation.RGB_DisplayPrimaries` method), 128

`__init__()` (`colour.characterisation.RGB_SpectralSensitivities` method), 124

`__init__()` (`colour.colorimetry.LMS_ConeFundamentals` method), 182

`__init__()` (`colour.colorimetry.RGB_ColourMatchingFunctions` method), 184

`__init__()` (`colour.colorimetry.XYZ_ColourMatchingFunctions` method), 187

`__init__()` (`colour.continuous.AbstractContinuousFunction` method), 246

`__init__()` (`colour.continuous.MultiSignal` method), 254

`__init__()` (`colour.continuous.Signal` method), 250

`__init__()` (`colour.domain_range_scale` method), 611

`__init__()` (`colour.io.AbstractLUTSequenceOperator` method), 286

`__init__()` (`colour.io.ImageAttribute_Specification` method), 275

`__init__()` (`colour.quality.CQS_Specification` method), 594

`__init__()` (`colour.quality.CRI_Specification` method), 592

`__init__()` (`colour.utilities.CaseInsensitiveMapping` method), 643

`__init__()` (`colour.utilities.Lookup` method), 644

`__init__()` (`colour.utilities.Structure` method), 645

`__init__()` (`colour.utilities.disable_multiprocessing` method), 614

`__ipow__()` (`colour.continuous.AbstractContinuousFunction` method), 246

`__isub__()` (`colour.continuous.AbstractContinuousFunction` method), 246

`__iter__()` (`colour.SpectralShape` method), 131

`__iter__()` (`colour.utilities.CaseInsensitiveMapping` method), 642

`__iter__()` (in module `colour`), 456

`__len__()` (`colour.LUTSequence` method), 282

`__len__()` (`colour.SpectralShape` method), 132

`__len__()` (`colour.continuous.AbstractContinuousFunction` method), 245

`__len__()` (`colour.utilities.CaseInsensitiveMapping` method), 642

`__len__()` (in module `colour`), 456

`__mul__()` (`colour.continuous.AbstractContinuousFunction` method), 246

`__ne__()` (`colour.LUTSequence` method), 282

`__ne__()` (`colour.SpectralShape` method), 132

`__ne__()` (`colour.continuous.AbstractContinuousFunction` method), 245

`__ne__()` (`colour.continuous.MultiSignal` method), 252

`__ne__()` (`colour.continuous.Signal` method), 248

`__ne__()` (`colour.utilities.CaseInsensitiveMapping` method), 642

`__ne__()` (in module `colour`), 456

`__pow__()` (`colour.continuous.AbstractContinuousFunction` method), 246

`__repr__()` (`colour.LUTSequence` method), 282

`__repr__()` (`colour.RGB_Colourspace` method), 360

`__repr__()` (`colour.SpectralShape` method), 131

`__repr__()` (`colour.continuous.AbstractContinuousFunction` method), 245

`__repr__()` (`colour.continuous.MultiSignal` method), 252

`__repr__()` (`colour.continuous.Signal` method), 248

`__repr__()` (`colour.utilities.CaseInsensitiveMapping` method), 642

`__repr__()` (in module `colour`), 456

`__setitem__()` (`colour.LUTSequence` method), 282

`__setitem__()` (`colour.continuous.AbstractContinuousFunction` method), 245

`__setitem__()` (`colour.continuous.MultiSignal` method), 252

`__setitem__()` (`colour.continuous.Signal` method), 248

`__setitem__()` (`colour.utilities.CaseInsensitiveMapping` method), 642

`__setitem__()` (in module `colour`), 456

`__str__()` (`colour.LUTSequence` method), 282

`__str__()` (`colour.RGB_Colourspace` method), 360

`__str__()` (`colour.SpectralShape` method), 131

`__str__()` (`colour.continuous.AbstractContinuousFunction` method), 245

`__str__()` (`colour.continuous.MultiSignal` method), 252

`__str__()` (`colour.continuous.Signal` method), 248

`__sub__()` (`colour.continuous.AbstractContinuousFunction` method), 246

A

- `absolute_tolerance` (*colour.NullInterpolator* attribute), 60
- `AbstractContinuousFunction` (class in *colour.continuous*), 245
- `AbstractLUTSequenceOperator` (class in *colour.io*), 286
- `ACES_2065_1_COLOURSPACE` (in module *colour.models*), 363
- `ACES_CC_COLOURSPACE` (in module *colour.models*), 364
- `ACES_CCT_COLOURSPACE` (in module *colour.models*), 364
- `ACES_CG_COLOURSPACE` (in module *colour.models*), 364
- `ACES_PROXY_COLOURSPACE` (in module *colour.models*), 364
- `ACES_RICD` (in module *colour.models*), 451
- `adjust_tristimulus_weighting_factors_ASTME30815` (in module *colour.colorimetry*), 173
- `ADOBE_RGB_1998_COLOURSPACE` (in module *colour.models*), 365
- `ADOBE_WIDE_GAMUT_RGB_COLOURSPACE` (in module *colour.models*), 365
- `ALEXA_WIDE_GAMUT_COLOURSPACE` (in module *colour.models*), 365
- `align()` (*colour.MultiSpectralDistribution* method), 136
- `align()` (*colour.SpectralDistribution* method), 133
- `anomalous_trichromacy_cmfs_Machado2009()` (in module *colour*), 108
- `anomalous_trichromacy_matrix_Machado2009()` (in module *colour*), 109
- `APPLE_RGB_COLOURSPACE` (in module *colour.models*), 365
- `apply()` (*colour.io.AbstractLUTSequenceOperator* method), 286
- `apply()` (*colour.LUT1D* method), 276
- `apply()` (*colour.LUT3D* method), 280
- `apply()` (*colour.LUT3x1D* method), 278
- `apply()` (*colour.LUTSequence* method), 282
- `arithmetical_operation()` (*colour.continuous.AbstractContinuousFunction* method), 246
- `arithmetical_operation()` (*colour.continuous.MultiSignal* method), 252
- `arithmetical_operation()` (*colour.continuous.Signal* method), 248
- `artist()` (in module *colour.plotting*), 495
- `as_array()` (in module *colour.utilities*), 628
- `as_float()` (in module *colour.utilities*), 630
- `as_float_array()` (in module *colour.utilities*), 628
- `as_int()` (in module *colour.utilities*), 629
- `as_int_array()` (in module *colour.utilities*), 628
- `as_LUT()` (*colour.LUT1D* method), 276
- `as_LUT()` (*colour.LUT3D* method), 280
- `as_LUT()` (*colour.LUT3x1D* method), 278
- `as_namedtuple()` (in module *colour.utilities*), 630
- `as_numeric()` (in module *colour.utilities*), 629
- `ASTME30815_PRACTISE_SHAPE` (in module *colour*), 140
- `ATD95_Specification` (class in *colour*), 85
- `augmented_matrix_Cheung2004()` (in module *colour.characterisation*), 117
- `AVOGADRO_CONSTANT` (in module *colour.constants*), 235

B

- `bandpass_correction()` (in module *colour*), 179
- `BANDPASS_CORRECTION_METHODS` (in module *colour*), 180
- `bandpass_correction_Stearns1988()` (in module *colour.colorimetry*), 180
- `bandwidth_corrected` (*colour.SpectralDistribution_IESTM2714* attribute), 299
- `bandwidth_FWHM` (*colour.SpectralDistribution_IESTM2714* attribute), 299
- `batch()` (in module *colour.utilities*), 614
- `BEST_RGB_COLOURSPACE` (in module *colour.models*), 365
- `BETA_RGB_COLOURSPACE` (in module *colour.models*), 366
- `blackbody_spectral_radiance()` (in module *colour.colorimetry*), 159
- `BOLTZMANN_CONSTANT` (in module *colour.constants*), 235
- `boundaries` (*colour.SpectralShape* attribute), 131
- `BRADFORD_CAT` (in module *colour.adaptation*), 51
- `BRENEMAN_EXPERIMENTS` (in module *colour*), 257
- `BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES` (in module *colour*), 258
- `BS_CAT` (in module *colour.adaptation*), 52
- `BS_PC_CAT` (in module *colour.adaptation*), 52
- `BT2020_COLOURSPACE` (in module *colour.models*), 367
- `BT470_525_COLOURSPACE` (in module *colour.models*), 366
- `BT470_625_COLOURSPACE` (in module *colour.models*), 366
- `BT709_COLOURSPACE` (in module *colour.models*), 366

C

- `CAM02LCD_to_JMh_CIECAM02()` (in module *colour*), 330
- `CAM02SCD_to_JMh_CIECAM02()` (in module *colour*), 331
- `CAM02UCS_to_JMh_CIECAM02()` (in module *colour*), 333
- `CAM16_InductionFactors` (class in *colour.appearance*), 94
- `CAM16_Specification` (class in *colour*), 93
- `CAM16_to_XYZ()` (in module *colour*), 91

- CAM16_VIEWING_CONDITIONS (in module colour), 93
- CAM16LCD_to_JMh_CAM16() (in module colour), 335
- CAM16SCD_to_JMh_CAM16() (in module colour), 336
- CAM16UCS_to_JMh_CAM16() (in module colour), 338
- camera() (in module colour.plotting), 495
- CAMERAS_RGB_SPECTRAL_SENSITIVITIES (in module colour), 127
- cartesian_to_cylindrical() (in module colour.algebra), 71
- cartesian_to_polar() (in module colour.algebra), 70
- cartesian_to_spherical() (in module colour.algebra), 69
- CaseInsensitiveMapping (class in colour.utilities), 642
- CAT02_BRILL_CAT (in module colour.adaptation), 52
- CAT02_CAT (in module colour.adaptation), 52
- CCT_to_uv() (in module colour), 602
- CCT_to_uv_Krystek1985() (in module colour.temperature), 606
- CCT_TO_UV_METHODS (in module colour), 602
- CCT_to_uv_Ohno2013() (in module colour.temperature), 607
- CCT_to_uv_Robertson1968() (in module colour.temperature), 605
- CCT_to_xy() (in module colour), 604
- CCT_to_xy_CIE_D() (in module colour.temperature), 610
- CCT_to_xy_Kang2002() (in module colour.temperature), 609
- CCT_TO_XY_METHODS (in module colour), 604
- centroid() (in module colour.utilities), 638
- chromatic_adaptation() (in module colour), 40
- chromatic_adaptation_CIE1994() (in module colour.adaptation), 44
- chromatic_adaptation_CMCCAT2000() (in module colour.adaptation), 45
- chromatic_adaptation_Fairchild1990() (in module colour.adaptation), 43
- chromatic_adaptation_forward_CMCCAT2000() (in module colour.adaptation), 47
- chromatic_adaptation_matrix_VonKries() (in module colour.adaptation), 53
- CHROMATIC_ADAPTATION_METHODS (in module colour), 42
- chromatic_adaptation_reverse_CMCCAT2000() (in module colour.adaptation), 48
- CHROMATIC_ADAPTATION_TRANSFORMS (in module colour), 42
- CHROMATIC_ADAPTATION_TRANSFORMS (in module colour.adaptation), 51
- chromatic_adaptation_VonKries() (in module colour.adaptation), 50
- chromatically_adapt() (colour.RGB_Colourspace method), 360
- chromatically_adapted_primaries() (in module colour), 356
- CIE_RGB_COLOURSPACE (in module colour.models), 367
- CIECAM02_InductionFactors (class in colour.appearance), 90
- CIECAM02_Specification (class in colour), 88
- CIECAM02_to_XYZ() (in module colour), 87
- CIECAM02_VIEWING_CONDITIONS (in module colour), 89
- CINEMA_GAMUT_COLOURSPACE (in module colour.models), 367
- closest() (in module colour.utilities), 631
- closest_indexes() (in module colour.utilities), 631
- CMCCAT2000_CAT (in module colour.adaptation), 52
- CMCCAT2000_InductionFactors (class in colour.adaptation), 49
- CMCCAT2000_VIEWING_CONDITIONS (in module colour), 42
- CMCCAT2000_VIEWING_CONDITIONS (in module colour.adaptation), 47
- CMCCAT97_CAT (in module colour.adaptation), 52
- CMFS (in module colour), 189
- CMY_to_CMYK() (in module colour), 470
- CMY_to_RGB() (in module colour), 469
- CMYK_to_CMY() (in module colour), 470
- COLOR_MATCH_RGB_COLOURSPACE (in module colour.models), 367
- colorimetric_purity() (in module colour), 198
- colour_correction() (in module colour), 114
- colour_correction_Cheung2004() (in module colour.characterisation), 120
- colour_correction_Finlayson2015() (in module colour.characterisation), 121
- colour_correction_matrix() (in module colour), 112
- colour_correction_matrix_Cheung2004() (in module colour.characterisation), 118
- colour_correction_matrix_Finlayson2015() (in module colour.characterisation), 119
- COLOUR_CORRECTION_MATRIX_METHODS (in module colour), 112
- colour_correction_matrix_Vandermonde() (in module colour.characterisation), 120
- COLOUR_CORRECTION_METHODS (in module colour), 114
- colour_correction_Vandermonde() (in module colour.characterisation), 122
- colour_cycle() (in module colour.plotting), 495
- colour_quality_scale() (in module colour), 593
- COLOUR_QUALITY_SCALE_METHODS (in module colour), 593
- colour_rendering_index() (in module colour), 591
- colour_style() (in module colour.plotting), 494
- ColourChecker (class in colour.characterisation), 123
- COLOURCHECKERS (in module colour), 122

- COLOURCHECKERS_SDS (in module colour), 123
 ColourRuntimeWarning, 650
 ColourUsageWarning, 650
 ColourWarning, 650
 complementary_wavelength() (in module colour), 197
 contrast_sensitivity_function() (in module colour), 237
 contrast_sensitivity_function_Barten1999() (in module colour.contrast), 239
 CONTRAST_SENSITIVITY_METHODS (in module colour), 238
 copy() (colour.continuous.AbstractContinuousFunction method), 246
 copy() (colour.LUTSequence method), 282
 copy() (colour.RGB_Colourspace method), 360
 copy() (colour.utilities.CaseInsensitiveMapping method), 642
 copy() (in module colour), 456
 corresponding_chromaticities_prediction() (in module colour), 256
 corresponding_chromaticities_prediction_CIE1994() (in module colour.corresponding), 259
 corresponding_chromaticities_prediction_CMCCAT2000() (in module colour.corresponding), 260
 corresponding_chromaticities_prediction_Fairchild1990() (in module colour.corresponding), 258
 CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS (in module colour), 257
 corresponding_chromaticities_prediction_VonKries() (in module colour.corresponding), 261
 CQS_Specification (class in colour.quality), 594
 CRI_Specification (class in colour.quality), 592
 cube() (in module colour.plotting), 590
 CV_range() (in module colour), 460
 CVD_MATRICES_MACHADO2010 (in module colour), 110
 cvd_matrix_Machado2009() (in module colour), 110
 cylindrical_to_cartesian() (in module colour.algebra), 72
- ## D
- daylight_locus_function() (in module colour.colorimetry), 160
 DCDM_XYZ_COLOURSPACE (in module colour.models), 367
 DCI_P3_COLOURSPACE (in module colour.models), 368
 DCI_P3_P_COLOURSPACE (in module colour.models), 368
 decoding_cctf (colour.RGB_Colourspace attribute), 360
 decoding_cctf() (in module colour), 375
 DECODING_CCTFS (in module colour), 376
 default (colour.NullInterpolator attribute), 60
 DEFAULT_FLOAT_DTYPE (in module colour.constants), 236
 DEFAULT_SPECTRAL_SHAPE (in module colour), 139
 delta_E() (in module colour), 262
 delta_E_CAM02LCD() (in module colour.difference), 268
 delta_E_CAM02SCD() (in module colour.difference), 269
 delta_E_CAM02UCS() (in module colour.difference), 269
 delta_E_CAM16LCD() (in module colour.difference), 270
 delta_E_CAM16SCD() (in module colour.difference), 271
 delta_E_CAM16UCS() (in module colour.difference), 272
 delta_E_CIE1976() (in module colour.difference), 264
 delta_E_CIE1994() (in module colour.difference), 264
 delta_E_CIE2000() (in module colour.difference), 266
 delta_E_CMC() (in module colour.difference), 267
 delta_E_DIN99() (in module colour.difference), 273
 DELTA_E_METHODS (in module colour), 263
 describe_environment() (in module colour.utilities), 648
 DIN99_to_Lab() (in module colour), 328
 dlopen() (in module colour.utilities), 614
 DISPLAYS_RGB_PRIMARIES (in module colour), 130
 domain (colour.continuous.AbstractContinuousFunction attribute), 245
 domain (colour.continuous.MultiSignal attribute), 251
 domain (colour.continuous.Signal attribute), 248
 domain_distance() (colour.continuous.AbstractContinuousFunction method), 246
 domain_range_scale (class in colour), 611
 dominant_wavelength() (in module colour), 196
 DON_RGB_4_COLOURSPACE (in module colour.models), 368
 dot_matrix() (in module colour.utilities), 637
 dot_vector() (in module colour.utilities), 636
 DRAGON_COLOR_2_COLOURSPACE (in module colour.models), 371
 DRAGON_COLOR_COLOURSPACE (in module colour.models), 371
 dtype (colour.continuous.MultiSignal attribute), 251
 dtype (colour.continuous.Signal attribute), 248
- ## E
- ECI_RGB_V2_COLOURSPACE (in module colour.models), 368
 EKTA_SPACE_PS_5_COLOURSPACE (in module colour.models), 369
 ellipse_coefficients_canonical_form() (in module colour.algebra), 76

- `ellipse_coefficients_general_form()` (in module `colour.algebra`), 75
 - `ellipse_fitting()` (in module `colour.algebra`), 77
 - `ellipse_fitting_Halir1998()` (in module `colour.algebra`), 78
 - `ELLIPSE_FITTING_METHODS` (in module `colour.algebra`), 77
 - `encoding_cctf` (`colour.RGB_Colourspace` attribute), 360
 - `encoding_cctf()` (in module `colour`), 374
 - `ENCODING_CCTFS` (in module `colour`), 375
 - `end` (`colour.SpectralShape` attribute), 131
 - `eotf()` (in module `colour`), 394
 - `eotf_BT1886()` (in module `colour.models`), 399
 - `eotf_BT2020()` (in module `colour.models`), 401
 - `eotf_BT2100_HLG()` (in module `colour.models`), 401
 - `eotf_BT2100_PQ()` (in module `colour.models`), 403
 - `eotf_DCDM()` (in module `colour.models`), 397
 - `eotf_DICOMGSDF()` (in module `colour.models`), 398
 - `eotf_ProPhotoRGB()` (in module `colour.models`), 404
 - `eotf_reverse()` (in module `colour`), 395
 - `eotf_reverse_BT1886()` (in module `colour.models`), 400
 - `eotf_reverse_BT2100_HLG()` (in module `colour.models`), 402
 - `eotf_reverse_BT2100_PQ()` (in module `colour.models`), 404
 - `eotf_reverse_DCDM()` (in module `colour.models`), 398
 - `eotf_RIMMRGB()` (in module `colour.models`), 405
 - `eotf_ROMMRGB()` (in module `colour.models`), 406
 - `eotf_SMPTE240M()` (in module `colour.models`), 407
 - `eotf_ST2084()` (in module `colour.models`), 407
 - `EOTFS` (in module `colour`), 395
 - `EOTFS_REVERSE` (in module `colour`), 396
 - `EPSILON` (in module `colour.constants`), 236
 - `ERIMM_RGB_COLOURSPACE` (in module `colour.models`), 372
 - `euclidean_distance()` (in module `colour.algebra`), 73
 - `excitation_purity()` (in module `colour`), 198
 - `extend_line_segment()` (in module `colour.algebra`), 73
 - `extrapolate()` (`colour.MultiSpectralDistribution` method), 136
 - `extrapolate()` (`colour.SpectralDistribution` method), 133
 - `Extrapolator` (class in `colour`), 54
 - `extrapolator` (`colour.continuous.AbstractContinuousFunction` attribute), 245
 - `extrapolator` (`colour.continuous.MultiSignal` attribute), 251
 - `extrapolator` (`colour.continuous.Signal` attribute), 248
 - `extrapolator_args` (`colour.continuous.AbstractContinuousFunction` attribute), 245
 - `extrapolator_args` (`colour.continuous.MultiSignal` attribute), 251
 - `extrapolator_args` (`colour.continuous.Signal` attribute), 248
- ## F
- `FAIRCHILD_CAT` (in module `colour.adaptation`), 52
 - `fill_nan()` (`colour.continuous.AbstractContinuousFunction` method), 246
 - `fill_nan()` (`colour.continuous.MultiSignal` method), 252
 - `fill_nan()` (`colour.continuous.Signal` method), 248
 - `fill_nan()` (in module `colour.utilities`), 640
 - `filter_kwargs()` (in module `colour.utilities`), 617
 - `filter_mapping()` (in module `colour.utilities`), 617
 - `filter_warnings()` (in module `colour.utilities`), 647
 - `first_item()` (in module `colour.utilities`), 618
 - `first_key_from_value()` (`colour.utilities.Lookup` method), 643
 - `FLOATING_POINT_NUMBER_PATTERN` (in module `colour.constants`), 236
 - `from_range_1()` (in module `colour.utilities`), 623
 - `from_range_10()` (in module `colour.utilities`), 623
 - `from_range_100()` (in module `colour.utilities`), 624
 - `from_range_degrees()` (in module `colour.utilities`), 625
 - `from_range_int()` (in module `colour.utilities`), 626
 - `full_to_legal()` (in module `colour`), 459
 - `function` (`colour.continuous.AbstractContinuousFunction` attribute), 245
 - `function` (`colour.continuous.MultiSignal` attribute), 251
 - `function` (`colour.continuous.Signal` attribute), 248
- ## G
- `gamma_function()` (in module `colour`), 393
 - `generate_pulse_waves()` (in module `colour.volume`), 658
 - `get_domain_range_scale()` (in module `colour`), 611
 - `grid()` (in module `colour.plotting`), 589
- ## H
- `handle_numpy_errors()` (in module `colour.utilities`), 613
 - `HDR_CIELAB_METHODS` (in module `colour`), 343
 - `hdr_CIELab_to_XYZ()` (in module `colour`), 342
 - `HDR_IPT_METHODS` (in module `colour`), 345
 - `hdr_IPT_to_XYZ()` (in module `colour`), 344
 - `header` (`colour.SpectralDistribution_IESTM2714` attribute), 299
 - `HEX_to_RGB()` (in module `colour.notation`), 481
 - `HSL_to_RGB()` (in module `colour`), 468
 - `HSV_to_RGB()` (in module `colour`), 466
 - `Hunt_Specification` (class in `colour`), 96

- HUNT_VIEWING_CONDITIONS (in module colour), 97
 Hunter_Lab_to_XYZ() (in module colour), 323
 Hunter_Rdab_to_XYZ() (in module colour), 326
 HUNTERLAB_ILLUMINANTS (in module colour), 195
- I**
- ICTCP_to_RGB() (in module colour), 463
 ignore_numpy_errors() (in module colour.utilities), 613
 ignore_python_warnings() (in module colour.utilities), 613
 ILLUMINANTS (in module colour), 194
 ILLUMINANTS_OPTIMAL_COLOUR_STIMULI (in module colour), 651
 ILLUMINANTS_SDS (in module colour), 194
 ImageAttribute_Specification (class in colour.io), 275
 in_array() (in module colour.utilities), 633
 insert() (colour.LUTSequence method), 282
 INTEGER_THRESHOLD (in module colour.constants), 236
 intermediate_lightness_function_CIE1976() (in module colour.colorimetry), 214
 intermediate_luminance_function_CIE1976() (in module colour.colorimetry), 220
 interpolate() (colour.MultiSpectralDistribution method), 136
 interpolate() (colour.SpectralDistribution method), 133
 interpolator (colour.continuous.AbstractContinuousFunction attribute), 245
 interpolator (colour.continuous.MultiSignal attribute), 251
 interpolator (colour.continuous.Signal attribute), 248
 interpolator_args (colour.continuous.AbstractContinuousFunction attribute), 245
 interpolator_args (colour.continuous.MultiSignal attribute), 251
 interpolator_args (colour.continuous.Signal attribute), 248
 intersect_line_segments() (in module colour.algebra), 74
 interval (colour.SpectralShape attribute), 131
 interval() (in module colour.utilities), 632
 IPT_hue_angle() (in module colour), 340
 IPT_to_XYZ() (in module colour), 339
 is_domain_explicit() (colour.LUT1D method), 276
 is_domain_explicit() (colour.LUT3D method), 280
 is_domain_explicit() (colour.LUT3x1D method), 278
 is_identity() (in module colour.algebra), 79
 is_integer() (in module colour.utilities), 616
 is_iterable() (in module colour.utilities), 615
 is_numeric() (in module colour.utilities), 616
 is_openimageio_installed() (in module colour.utilities), 615
 is_pandas_installed() (in module colour.utilities), 615
 is_sibling() (in module colour.utilities), 617
 is_spow_enabled() (in module colour.algebra), 82
 is_string() (in module colour.utilities), 616
 is_uniform() (colour.continuous.AbstractContinuousFunction method), 246
 is_uniform() (in module colour.utilities), 633
 is_within_macadam_limits() (in module colour), 651
 is_within_mesh_volume() (in module colour), 652
 is_within_pointer_gamut() (in module colour), 653
 is_within_visible_spectrum() (in module colour), 657
- J**
- JMh_CAM16_to_CAM16LCD() (in module colour), 334
 JMh_CAM16_to_CAM16SCD() (in module colour), 336
 JMh_CAM16_to_CAM16UCS() (in module colour), 337
 JMh_CIECAM02_to_CAM02LCD() (in module colour), 329
 JMh_CIECAM02_to_CAM02SCD() (in module colour), 331
 JMh_CIECAM02_to_CAM02UCS() (in module colour), 332
 JzAzBz_to_XYZ() (in module colour), 349
- K**
- K_M (in module colour.constants), 234
 kernel (colour.KernelInterpolator attribute), 57
 kernel_args (colour.KernelInterpolator attribute), 57
 kernel_cardinal_spline() (in module colour), 67
 kernel_fanucos() (in module colour), 66
 kernel_linear() (in module colour), 65
 kernel_nearest_neighbour() (in module colour), 65
 kernel_sinc() (in module colour), 66
 KernelInterpolator (class in colour), 56
 keys_from_value() (colour.utilities.Lookup method), 643
 KP_M (in module colour.constants), 234
- L**
- Lab_to_DIN99() (in module colour), 327
 Lab_to_LCHab() (in module colour), 310
 Lab_to_XYZ() (in module colour), 309
 label_rectangles() (in module colour.plotting), 496
 labels (colour.continuous.MultiSignal attribute), 251
 lagrange_coefficients() (in module colour), 63
 lagrange_coefficients_ASTME202211() (in module colour.colorimetry), 174
 LCHab_to_Lab() (in module colour), 311
 LCHuv_to_Luv() (in module colour), 314

least_square_mapping_MoorePenrose() (in module *colour.algebra*), 81
 LEFS (in module *colour*), 209
 legal_to_full() (in module *colour*), 460
 lerp() (in module *colour.utilities*), 639
 LIGHT_SOURCES (in module *colour*), 195
 LIGHT_SOURCES_SDS (in module *colour*), 195
 LIGHT_SPEED (in module *colour.constants*), 235
 lightness() (in module *colour*), 210
 lightness_CIE1976() (in module *colour.colorimetry*), 213
 lightness_Fairchild2010() (in module *colour.colorimetry*), 215
 lightness_Fairchild2011() (in module *colour.colorimetry*), 216
 lightness_Glasser1958() (in module *colour.colorimetry*), 211
 LIGHTNESS_METHODS (in module *colour*), 211
 lightness_Wyszecki1963() (in module *colour.colorimetry*), 212
 linear_conversion() (in module *colour.utilities*), 639
 linear_function() (in module *colour*), 394
 linear_table() (*colour.LUT1D* method), 276
 linear_table() (*colour.LUT3D* method), 280
 linear_table() (*colour.LUT3x1D* method), 278
 LinearInterpolator (class in *colour*), 59
 LineSegmentsIntersections_Specification (class in *colour.algebra*), 78
 LLAB_InductionFactors (class in *colour.appearance*), 100
 LLAB_Specification (class in *colour*), 99
 LLAB_VIEWING_CONDITIONS (in module *colour*), 100
 LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs() (in module *colour.colorimetry*), 193
 LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs() (in module *colour.colorimetry*), 192
 LMS_CMFS (in module *colour*), 189
 LMS_ConeFundamentals (class in *colour.colorimetry*), 181
 log_decoding_ACEScc() (in module *colour.models*), 419
 log_decoding_ACEScct() (in module *colour.models*), 420
 log_decoding_ACESproxy() (in module *colour.models*), 422
 log_decoding_ALEXAlogC() (in module *colour.models*), 423
 log_decoding_CanonLog() (in module *colour.models*), 428
 log_decoding_CanonLog2() (in module *colour.models*), 425
 log_decoding_CanonLog3() (in module *colour.models*), 427
 log_decoding_Cineon() (in module *colour.models*), 430
 log_decoding_curve() (in module *colour*), 415
 LOG_DECODING_CURVES (in module *colour*), 416
 log_decoding_ERIMMRGB() (in module *colour.models*), 431
 log_decoding_Log3G10() (in module *colour.models*), 433
 log_decoding_Log3G12() (in module *colour.models*), 435
 log_decoding_Panalog() (in module *colour.models*), 436
 log_decoding_PivotedLog() (in module *colour.models*), 437
 log_decoding_Protune() (in module *colour.models*), 439
 log_decoding_REDLog() (in module *colour.models*), 440
 log_decoding_REDLogFilm() (in module *colour.models*), 441
 log_decoding_SLog() (in module *colour.models*), 443
 log_decoding_SLog2() (in module *colour.models*), 444
 log_decoding_SLog3() (in module *colour.models*), 446
 log_decoding_ViperLog() (in module *colour.models*), 449
 log_decoding_VLog() (in module *colour.models*), 448
 log_encoding_ACEScc() (in module *colour.models*), 418
 log_encoding_ACEScct() (in module *colour.models*), 419
 log_encoding_ACESproxy() (in module *colour.models*), 421
 log_encoding_ALEXAlogC() (in module *colour.models*), 423
 log_encoding_CanonLog() (in module *colour.models*), 427
 log_encoding_CanonLog2() (in module *colour.models*), 424
 log_encoding_CanonLog3() (in module *colour.models*), 426
 log_encoding_Cineon() (in module *colour.models*), 429
 log_encoding_curve() (in module *colour*), 414
 LOG_ENCODING_CURVES (in module *colour*), 415
 log_encoding_ERIMMRGB() (in module *colour.models*), 430
 log_encoding_Log3G10() (in module *colour.models*), 432
 log_encoding_Log3G12() (in module *colour.models*), 434
 log_encoding_Panalog() (in module *colour.models*), 435

- [log_encoding_PivotedLog\(\)](#) (in module [colour.models](#)), 437
[log_encoding_Protune\(\)](#) (in module [colour.models](#)), 438
[log_encoding_REDLog\(\)](#) (in module [colour.models](#)), 439
[log_encoding_REDLogFilm\(\)](#) (in module [colour.models](#)), 441
[log_encoding_SLog\(\)](#) (in module [colour.models](#)), 442
[log_encoding_SLog2\(\)](#) (in module [colour.models](#)), 444
[log_encoding_SLog3\(\)](#) (in module [colour.models](#)), 445
[log_encoding_ViperLog\(\)](#) (in module [colour.models](#)), 449
[log_encoding_VLog\(\)](#) (in module [colour.models](#)), 447
[Lookup](#) (class in [colour.utilities](#)), 643
[lower_items\(\)](#) ([colour.utilities.CaseInsensitiveMapping](#) method), 642
[lower_items\(\)](#) (in module [colour](#)), 456
[luminance\(\)](#) (in module [colour](#)), 217
[luminance_ASTMD153508\(\)](#) (in module [colour.colorimetry](#)), 221
[luminance_CIE1976\(\)](#) (in module [colour.colorimetry](#)), 219
[luminance_Fairchild2010\(\)](#) (in module [colour.colorimetry](#)), 222
[luminance_Fairchild2011\(\)](#) (in module [colour.colorimetry](#)), 223
[LUMINANCE_METHODS](#) (in module [colour](#)), 218
[luminance_Newhall1943\(\)](#) (in module [colour.colorimetry](#)), 218
[luminous_efficacy\(\)](#) (in module [colour](#)), 199
[luminous_efficiency\(\)](#) (in module [colour](#)), 200
[luminous_flux\(\)](#) (in module [colour](#)), 200
[LUT1D](#) (class in [colour](#)), 276
[LUT3D](#) (class in [colour](#)), 280
[LUT3x1D](#) (class in [colour](#)), 278
[LUT_to_LUT\(\)](#) (in module [colour.io](#)), 286
[LUTSequence](#) (class in [colour](#)), 282
[Luv_to_LCHuv\(\)](#) (in module [colour](#)), 313
[Luv_to_uv\(\)](#) (in module [colour](#)), 315
[Luv_to_XYZ\(\)](#) (in module [colour](#)), 313
[Luv_uv_to_xy\(\)](#) (in module [colour](#)), 316
- ## M
- [mapping](#) ([colour.SpectralDistribution_IESTM2714](#) attribute), 299
[MAX_RGB_COLOURSPACE](#) (in module [colour.models](#)), 369
[maximum_angular_size_Barten1999\(\)](#) (in module [colour.contrast](#)), 244
[message_box\(\)](#) (in module [colour.utilities](#)), 646
[metric_mse\(\)](#) (in module [colour.utilities](#)), 641
[metric_psnr\(\)](#) (in module [colour.utilities](#)), 641
[MULTI_SD_TO_XYZ_METHODS](#) (in module [colour](#)), 168
[multi_sds_to_XYZ\(\)](#) (in module [colour](#)), 166
[multi_sds_to_XYZ_integration\(\)](#) (in module [colour.colorimetry](#)), 178
[multi_signal_unpack_data\(\)](#) ([colour.continuous.MultiSignal](#) method), 252
[multiprocessing_pool\(\)](#) (in module [colour.utilities](#)), 614
[MultiSignal](#) (class in [colour.continuous](#)), 251
[MultiSpectralDistribution](#) (class in [colour](#)), 136
[munsell_colour_to_xyY\(\)](#) (in module [colour](#)), 472
[MUNSELL_COLOURS](#) (in module [colour](#)), 473
[munsell_value\(\)](#) (in module [colour](#)), 474
[munsell_value_ASTMD153508\(\)](#) (in module [colour.notation](#)), 480
[munsell_value_Ladd1955\(\)](#) (in module [colour.notation](#)), 478
[munsell_value_McCamy1987\(\)](#) (in module [colour.notation](#)), 479
[MUNSELL_VALUE_METHODS](#) (in module [colour](#)), 475
[munsell_value_Moon1943\(\)](#) (in module [colour.notation](#)), 477
[munsell_value_Munsell1933\(\)](#) (in module [colour.notation](#)), 476
[munsell_value_Priest1920\(\)](#) (in module [colour.notation](#)), 475
[munsell_value_Saunderson1944\(\)](#) (in module [colour.notation](#)), 478
- ## N
- [name](#) ([colour.continuous.AbstractContinuousFunction](#) attribute), 245
[name](#) ([colour.RGB_Colourspace](#) attribute), 359
[Nayatani95_Specification](#) (class in [colour](#)), 102
[ndarray_write\(\)](#) (in module [colour.utilities](#)), 640
[NearestNeighbourInterpolator](#) (class in [colour](#)), 58
[normalise\(\)](#) ([colour.MultiSpectralDistribution](#) method), 136
[normalise\(\)](#) ([colour.SpectralDistribution](#) method), 133
[normalise_maximum\(\)](#) (in module [colour.utilities](#)), 632
[normalise_vector\(\)](#) (in module [colour.algebra](#)), 73
[normalised_primary_matrix\(\)](#) (in module [colour](#)), 355
[NTSC_COLOURSPACE](#) (in module [colour.models](#)), 369
[NullInterpolator](#) (class in [colour](#)), 60
[numpy_print_options\(\)](#) (in module [colour.utilities](#)), 648
- ## O
- [oetf\(\)](#) (in module [colour](#)), 377
[oetf_ARIBSTDB67\(\)](#) (in module [colour.models](#)), 379

- `oetf_BT2020()` (in module `colour.models`), 382
 - `oetf_BT2100_HLG()` (in module `colour.models`), 382
 - `oetf_BT2100_PQ()` (in module `colour.models`), 384
 - `oetf_BT601()` (in module `colour.models`), 385
 - `oetf_BT709()` (in module `colour.models`), 386
 - `oetf_DICOMGSDF()` (in module `colour.models`), 381
 - `oetf_ProPhotoRGB()` (in module `colour.models`), 387
 - `oetf_reverse()` (in module `colour`), 378
 - `oetf_reverse_ARIBSTDB67()` (in module `colour.models`), 380
 - `oetf_reverse_BT2100_HLG()` (in module `colour.models`), 383
 - `oetf_reverse_BT2100_PQ()` (in module `colour.models`), 384
 - `oetf_reverse_BT601()` (in module `colour.models`), 386
 - `oetf_reverse_BT709()` (in module `colour.models`), 387
 - `oetf_reverse_sRGB()` (in module `colour.models`), 392
 - `oetf_RIMMRGB()` (in module `colour.models`), 388
 - `oetf_ROMMRGB()` (in module `colour.models`), 389
 - `oetf_SMPTE240M()` (in module `colour.models`), 390
 - `oetf_sRGB()` (in module `colour.models`), 391
 - `oetf_ST2084()` (in module `colour.models`), 390
 - `OETFS` (in module `colour`), 378
 - `OETFS_REVERSE` (in module `colour`), 378
 - `ootf()` (in module `colour`), 409
 - `ootf_BT2100_HLG()` (in module `colour.models`), 410
 - `ootf_BT2100_PQ()` (in module `colour.models`), 412
 - `ootf_reverse()` (in module `colour`), 409
 - `ootf_reverse_BT2100_HLG()` (in module `colour.models`), 411
 - `ootf_reverse_BT2100_PQ()` (in module `colour.models`), 413
 - `OOTFS` (in module `colour`), 409
 - `OOTFS_REVERSE` (in module `colour`), 410
 - `optical_MTF_Barten1999()` (in module `colour.contrast`), 241
 - `orient()` (in module `colour.utilities`), 638
 - `OSA_UCS_to_XYZ()` (in module `colour`), 347
- ## P
- `P3_D65_COLOURSPACE` (in module `colour.models`), 369
 - `padding_args` (`colour.KernelInterpolator` attribute), 57
 - `PAL_SECAM_COLOURSPACE` (in module `colour.models`), 370
 - `path` (`colour.SpectralDistribution_IESTM2714` attribute), 299
 - `PchipInterpolator` (class in `colour`), 61
 - `PHOTOPIC_LEFS` (in module `colour`), 209
 - `PLANCK_CONSTANT` (in module `colour.constants`), 235
 - `planck_law()` (in module `colour.colorimetry`), 160
 - `plot_blackbody_colours()` (in module `colour.plotting`), 515
 - `plot_blackbody_spectral_radiance()` (in module `colour.plotting`), 514
 - `plot_chromaticity_diagram()` (in module `colour.plotting.diagrams`), 535
 - `plot_chromaticity_diagram_CIE1931()` (in module `colour.plotting`), 521
 - `plot_chromaticity_diagram_CIE1960UCS()` (in module `colour.plotting`), 522
 - `plot_chromaticity_diagram_CIE1976UCS()` (in module `colour.plotting`), 524
 - `plot_chromaticity_diagram_colours()` (in module `colour.plotting.diagrams`), 533
 - `plot_colour_qualityBars()` (in module `colour.plotting.quality`), 575
 - `plot_corresponding_chromaticities_prediction()` (in module `colour.plotting`), 519
 - `plot_cvd_simulation_Machado2009()` (in module `colour.plotting`), 516
 - `plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931()` (in module `colour.plotting`), 550
 - `plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS()` (in module `colour.plotting`), 552
 - `plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS()` (in module `colour.plotting`), 554
 - `plot_image()` (in module `colour.plotting`), 501
 - `plot_multi_cctfs()` (in module `colour.plotting`), 557
 - `plot_multi_cmfs()` (in module `colour.plotting`), 506
 - `plot_multi_colour_checkers()` (in module `colour.plotting`), 518
 - `plot_multi_colour_swatches()` (in module `colour.plotting`), 498
 - `plot_multi_functions()` (in module `colour.plotting`), 500
 - `plot_multi_illuminant_sds()` (in module `colour.plotting`), 508
 - `plot_multi_lightness_functions()` (in module `colour.plotting`), 511
 - `plot_multi_luminance_functions()` (in module `colour.plotting`), 513
 - `plot_multi_munsell_value_functions()` (in module `colour.plotting`), 564
 - `plot_multi_sds()` (in module `colour.plotting`), 504
 - `plot_multi_sds_colour_quality_scalesBars()` (in module `colour.plotting`), 573
 - `plot_multi_sds_colour_rendering_indexesBars()` (in module `colour.plotting`), 569
 - `plot_planckian_locus()` (in module `colour.plotting.temperature`), 581
 - `plot_planckian_locus_in_chromaticity_diagram()` (in module `colour.plotting.temperature`), 583
 - `plot_planckian_locus_in_chromaticity_diagram_CIE1931()` (in module `colour.plotting`), 578

[plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS\(\)](#) (in module [colour.plotting](#)), 579
[plot_pointer_gamut\(\)](#) (in module [plot_the_blue_sky\(\)](#) (in module [colour.plotting](#)), [colour.plotting.models](#)), 558
[plot_RGB_chromaticities_in_chromaticity_diagram\(\)](#) (in module [colour.plotting.models](#)), 561
[plot_RGB_chromaticities_in_chromaticity_diagram_POE1931\(\)](#) (in module [colour.plotting](#)), 544
[plot_RGB_chromaticities_in_chromaticity_diagram_POE1960UCS\(\)](#) (in module [colour.plotting](#)), 546
[plot_RGB_chromaticities_in_chromaticity_diagram_POE1976UCS\(\)](#) (in module [colour.plotting](#)), 548
[plot_RGB_colourspace_gamuts\(\)](#) (in module [colour.plotting](#)), 586
[plot_RGB_colourspace_in_chromaticity_diagram\(\)](#) (in module [colour.plotting.models](#)), 559
[plot_RGB_colourspace_in_chromaticity_diagram_CIE1931\(\)](#) (in module [colour.plotting](#)), 539
[plot_RGB_colourspace_in_chromaticity_diagram_CIE1960UCS\(\)](#) (in module [colour.plotting](#)), 540
[plot_RGB_colourspace_in_chromaticity_diagram_CIE1976UCS\(\)](#) (in module [colour.plotting](#)), 542
[plot_RGB_scatter\(\)](#) (in module [colour.plotting](#)), 587
[plot_sds_in_chromaticity_diagram\(\)](#) (in module [colour.plotting.diagrams](#)), 537
[plot_sds_in_chromaticity_diagram_CIE1931\(\)](#) (in module [colour.plotting](#)), 525
[plot_sds_in_chromaticity_diagram_CIE1960UCS\(\)](#) (in module [colour.plotting](#)), 527
[plot_sds_in_chromaticity_diagram_CIE1976UCS\(\)](#) (in module [colour.plotting](#)), 529
[plot_single_cctf\(\)](#) (in module [colour.plotting](#)), 556
[plot_single_cmfs\(\)](#) (in module [colour.plotting](#)), 505
[plot_single_colour_checker\(\)](#) (in module [colour.plotting](#)), 517
[plot_single_colour_swatch\(\)](#) (in module [colour.plotting](#)), 497
[plot_single_function\(\)](#) (in module [colour.plotting](#)), 499
[plot_single_illuminant_sd\(\)](#) (in module [colour.plotting](#)), 507
[plot_single_lightness_function\(\)](#) (in module [colour.plotting](#)), 510
[plot_single_luminance_function\(\)](#) (in module [colour.plotting](#)), 512
[plot_single_munsell_value_function\(\)](#) (in module [colour.plotting](#)), 563
[plot_single_sd\(\)](#) (in module [colour.plotting](#)), 503
[plot_single_sd_colour_quality_scaleBars\(\)](#) (in module [colour.plotting](#)), 571
[plot_single_sd_colour_rendering_indexBars\(\)](#) (in module [colour.plotting](#)), 567
[plot_single_sd_rayleigh_scattering\(\)](#) (in module [colour.plotting](#)), 565
[plot_visible_spectrum\(\)](#) (in module [colour.plotting](#)), 509
[plot_visible_spectrum_angle_on_ellipse\(\)](#) (in module [colour.algebra](#)), 76
[POE1931_BOUNDARIES](#) (in module [colour](#)), 471
[POE1931_GAMUT_DATA](#) (in module [colour](#)), 471
[POE1931_ILLUMINANT](#) (in module [colour](#)), 471
[polar_to_cartesian\(\)](#) (in module [colour.algebra](#)), 71
[polynomial_expansion\(\)](#) (in module [colour](#)), 111
[polynomial_expansion_Finlayson2015\(\)](#) (in module [colour.characterisation](#)), 117
[POLYNOMIAL_EXPANSION_METHODS](#) (in module [colour](#)), 117
[polynomial_expansion_Vandermonde\(\)](#) (in module [colour.characterisation](#)), 118
[primaries](#) ([colour.RGB_Colourspace](#) attribute), 359
[print_numpy_errors\(\)](#) (in module [colour.utilities](#)), 613
[Prismatic_to_RGB\(\)](#) (in module [colour](#)), 465
[PROPHOTO_RGB_COLOURSPACE](#) (in module [colour.models](#)), 372
[PROTUNE_NATIVE_COLOURSPACE](#) (in module [colour.models](#)), 369
[pupil_diameter_Barten1999\(\)](#) (in module [colour.contrast](#)), 242

Q

[quad\(\)](#) (in module [colour.plotting](#)), 588

R

[raise_numpy_errors\(\)](#) (in module [colour.utilities](#)), 613
[random_triplet_generator\(\)](#) (in module [colour.algebra](#)), 80
[range](#) ([colour.continuous.AbstractContinuousFunction](#) attribute), 245
[range](#) ([colour.continuous.MultiSignal](#) attribute), 251
[range](#) ([colour.continuous.Signal](#) attribute), 248
[range\(\)](#) ([colour.SpectralShape](#) method), 132
[rayleigh_optical_depth\(\)](#) (in module [colour.phenomena](#)), 493
[rayleigh_scattering\(\)](#) (in module [colour](#)), 482
[reaction_rate_MichealisMenten\(\)](#) (in module [colour.biochemistry](#)), 106
[read\(\)](#) ([colour.SpectralDistribution_IESTM2714](#) method), 299
[read_image\(\)](#) (in module [colour](#)), 274
[read_LUT\(\)](#) (in module [colour](#)), 284
[read_LUT_Cinespace\(\)](#) (in module [colour.io](#)), 288

- `read_LUT_IridasCube()` (in module `colour.io`), 289
 - `read_LUT_SonySPI1D()` (in module `colour.io`), 291
 - `read_LUT_SonySPI3D()` (in module `colour.io`), 293
 - `read_sds_from_csv_file()` (in module `colour`), 294
 - `read_sds_from_xrite_file()` (in module `colour`), 302
 - `read_spectral_data_from_csv_file()` (in module `colour`), 296
 - `RED_COLOR_2_COLOURSPACE` (in module `colour.models`), 370
 - `RED_COLOR_3_COLOURSPACE` (in module `colour.models`), 370
 - `RED_COLOR_4_COLOURSPACE` (in module `colour.models`), 370
 - `RED_COLOR_COLOURSPACE` (in module `colour.models`), 370
 - `RED_WIDE_GAMUT_RGB_COLOURSPACE` (in module `colour.models`), 371
 - `reflection_geometry` (`colour.SpectralDistribution_IESTIMPL` attribute), 299
 - `relative_tolerance` (`colour.NullInterpolator` attribute), 60
 - `render()` (in module `colour.plotting`), 495
 - `retinal_illuminance_Barten1999()` (in module `colour.contrast`), 243
 - `RGB_10_degree_cmfs_to_LMS_10_degree_cmfs()` (in module `colour.colorimetry`), 192
 - `RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs()` (in module `colour.colorimetry`), 191
 - `RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs()` (in module `colour.colorimetry`), 191
 - `RGB_CMFS` (in module `colour`), 189
 - `RGB_ColourMatchingFunctions` (class in `colour.colorimetry`), 184
 - `RGB_Colourspace` (class in `colour`), 358
 - `RGB_colourspace_limits()` (in module `colour`), 654
 - `RGB_colourspace_pointer_gamut_coverage_MonteCarlo()` (in module `colour`), 654
 - `RGB_colourspace_visible_spectrum_coverage_MonteCarlo()` (in module `colour`), 655
 - `RGB_colourspace_volume_coverage_MonteCarlo()` (in module `colour`), 656
 - `RGB_colourspace_volume_MonteCarlo()` (in module `colour`), 655
 - `RGB_COLOURSPACES` (in module `colour`), 362
 - `RGB_DisplayPrimaries` (class in `colour.characterisation`), 127
 - `RGB_luminance()` (in module `colour`), 357
 - `RGB_luminance_equation()` (in module `colour`), 358
 - `RGB_SpectralSensitivities` (class in `colour.characterisation`), 124
 - `RGB_to_CMY()` (in module `colour`), 468
 - `RGB_to_HEX()` (in module `colour.notation`), 481
 - `RGB_to_HSL()` (in module `colour`), 467
 - `RGB_to_HSV()` (in module `colour`), 465
 - `RGB_to_ICTCP()` (in module `colour`), 462
 - `RGB_to_Prismatic()` (in module `colour`), 464
 - `RGB_to_RGB()` (in module `colour`), 352
 - `RGB_to_RGB_matrix()` (in module `colour`), 353
 - `RGB_to_sd_Smits1999()` (in module `colour.recovery`), 598
 - `RGB_to_XYZ()` (in module `colour`), 351
 - `RGB_to_XYZ_matrix` (`colour.RGB_Colourspace` attribute), 360
 - `RGB_to_YCbCr()` (in module `colour`), 452
 - `RGB_to_YcBcCrC()` (in module `colour`), 456
 - `RGB_to_YCoCg()` (in module `colour`), 461
 - `RIMM_RGB_COLOURSPACE` (in module `colour.models`), 372
 - `RLAB_D_FACTOR` (in module `colour`), 104
 - `RLAB_Specification` (class in `colour`), 105
 - `RLAB_VIEWING_CONDITIONS` (in module `colour`), 106
 - `ROMM_RGB_COLOURSPACE` (in module `colour.models`), 371
 - `row_as_diagonal()` (in module `colour.utilities`), 635
 - `RUSSELL_RGB_COLOURSPACE` (in module `colour.models`), 372
- ## S
- `S_GAMUT3_CINE_COLOURSPACE` (in module `colour.models`), 373
 - `S_GAMUT3_COLOURSPACE` (in module `colour.models`), 373
 - `S_GAMUT_COLOURSPACE` (in module `colour.models`), 373
 - `scattering_cross_section()` (in module `colour`), 492
 - `SCOTOPIC_LEFS` (in module `colour`), 209
 - `sd_blackbody()` (in module `colour`), 144
 - `sd_CIE_illuminant_D_series()` (in module `colour`), 142
 - `sd_CIE_standard_illuminant_A()` (in module `colour`), 141
 - `sd_constant()` (in module `colour`), 154
 - `sd_gaussian()` (in module `colour`), 155
 - `sd_gaussian_fwhm()` (in module `colour.colorimetry`), 161
 - `SD_GAUSSIAN_METHODS` (in module `colour`), 155
 - `sd_gaussian_normal()` (in module `colour.colorimetry`), 161
 - `sd_mesopic_luminous_efficiency_function()` (in module `colour`), 201
 - `sd_multi_leds()` (in module `colour`), 157
 - `SD_MULTI_LEDS_METHODS` (in module `colour`), 157
 - `sd_multi_leds_Ohno2005()` (in module `colour.colorimetry`), 163
 - `sd_ones()` (in module `colour`), 154
 - `sd_rayleigh_scattering()` (in module `colour`), 483
 - `sd_single_led()` (in module `colour`), 156

- SD_SINGLE_LED_METHODS (in module colour), 156
sd_single_led_Ohno2005() (in module colour.colorimetry), 162
sd_to_aces_relative_exposure_values() (in module colour), 450
sd_to_XYZ() (in module colour), 164
sd_to_XYZ_ASTME30815() (in module colour.colorimetry), 169
sd_to_XYZ_integration() (in module colour.colorimetry), 176
SD_TO_XYZ_METHODS (in module colour), 166
sd_to_XYZ_tristimulus_weighting_factors_ASTME30815() (in module colour.colorimetry), 171
sd_zeros() (in module colour), 155
sequence (colour.LUTSequence attribute), 282
set_domain_range_scale() (in module colour), 612
set_spow_enable() (in module colour.algebra), 82
shape (colour.MultiSpectralDistribution attribute), 136
shape (colour.SpectralDistribution attribute), 133
SHARP_CAT (in module colour.adaptation), 52
sigma_Barten1999() (in module colour.contrast), 242
Signal (class in colour.continuous), 247
signal_type (colour.continuous.MultiSignal attribute), 252
signal_unpack_data() (colour.continuous.Signal method), 248
signals (colour.continuous.MultiSignal attribute), 251
SMITS_1999_SDS (in module colour.recovery), 598
SMPTE_240M_COLOURSPACE (in module colour.models), 372
spectral_quantity (colour.SpectralDistribution_IESTM2714 attribute), 299
SpectralDistribution (class in colour), 132
SpectralDistribution_IESTM2714 (class in colour), 298
SpectralShape (class in colour), 131
spherical_to_cartesian() (in module colour.algebra), 70
spow() (in module colour.algebra), 83
spow_enable (class in colour.algebra), 82
SpragueInterpolator (class in colour), 62
sRGB_COLOURSPACE (in module colour.models), 373
sRGB_to_XYZ() (in module colour), 354
STANDARD_OBSERVERS_CMFS (in module colour), 190
start (colour.SpectralShape attribute), 131
strict_labels (colour.MultiSpectralDistribution attribute), 136
strict_name (colour.MultiSpectralDistribution attribute), 136
strict_name (colour.SpectralDistribution attribute), 133
Structure (class in colour.utilities), 644
substrate_concentration_MichealisMenten() (in module colour.biochemistry), 107
suppress_warnings() (in module colour.utilities), 648
- ## T
- table_interpolation() (in module colour), 64
TABLE_INTERPOLATION_METHODS (in module colour), 64
table_interpolation_tetrahedral() (in module colour.algebra), 68
table_interpolation_trilinear() (in module colour.algebra), 68
to_dataframe() (colour.continuous.MultiSignal method), 252
to_domain_1() (in module colour.utilities), 619
to_domain_10() (in module colour.utilities), 619
to_domain_100() (in module colour.utilities), 620
to_domain_degrees() (in module colour.utilities), 621
to_domain_int() (in module colour.utilities), 622
to_sds() (colour.MultiSpectralDistribution method), 136
to_series() (colour.continuous.Signal method), 248
transmission_geometry (colour.SpectralDistribution_IESTM2714 attribute), 299
trim() (colour.MultiSpectralDistribution method), 136
trim() (colour.SpectralDistribution method), 133
tristimulus_weighting_factors_ASTME202211() (in module colour.colorimetry), 174
truncate() (in module colour.utilities), 635
tstack() (in module colour.utilities), 634
- ## U
- UCS_to_uv() (in module colour), 319
UCS_to_XYZ() (in module colour), 318
UCS_uv_to_xy() (in module colour), 320
uniform_axes3d() (in module colour.plotting), 497
use_derived_RGB_to_XYZ_matrix (colour.RGB_Colourspace attribute), 360
use_derived_transformation_matrices() (colour.RGB_Colourspace method), 360
use_derived_XYZ_to_RGB_matrix (colour.RGB_Colourspace attribute), 360
uv_to_CCT() (in module colour), 603
UV_TO_CCT_METHODS (in module colour), 603
uv_to_CCT_Ohno2013() (in module colour.temperature), 608
uv_to_CCT_Robertson1968() (in module colour.temperature), 606
uv_to_Luv() (in module colour), 315
uv_to_UCS() (in module colour), 319

UVW_to_XYZ() (in module colour), 322

V

V_GAMUT_COLOURSPACE (in module colour.models), 374

values (colour.MultiSpectralDistribution attribute), 136

values (colour.SpectralDistribution attribute), 133

VON_KRIES_CAT (in module colour.adaptation), 52

W

warn_numpy_errors() (in module colour.utilities), 613

warning() (in module colour.utilities), 647

wavelength_to_XYZ() (in module colour), 168

wavelengths (colour.MultiSpectralDistribution attribute), 136

wavelengths (colour.SpectralDistribution attribute), 133

whiteness() (in module colour), 224

whiteness_ASTME313() (in module colour.colorimetry), 228

whiteness_Berger1959() (in module colour.colorimetry), 225

whiteness_CIE2004() (in module colour.colorimetry), 230

whiteness_Ganz1979() (in module colour.colorimetry), 229

WHITENESS_METHODS (in module colour), 225

whiteness_Stensby1968() (in module colour.colorimetry), 227

whiteness-Taube1960() (in module colour.colorimetry), 226

whitepoint (colour.RGB_Colourspace attribute), 359

whitepoint_name (colour.RGB_Colourspace attribute), 360

window (colour.KernelInterpolator attribute), 57

write() (colour.SpectralDistribution_IESTM2714 method), 299

write_image() (in module colour), 274

write_LUT() (in module colour), 285

write_LUT_Cinespace() (in module colour.io), 288

write_LUT_IridasCube() (in module colour.io), 290

write_LUT_SonySPI1D() (in module colour.io), 292

write_LUT_SonySPI3D() (in module colour.io), 293

write_sds_to_csv_file() (in module colour), 298

X

x (colour.KernelInterpolator attribute), 57

x (colour.LinearInterpolator attribute), 59

x (colour.NullInterpolator attribute), 60

x (colour.SpragueInterpolator attribute), 62

XTREME_RGB_COLOURSPACE (in module colour.models), 374

xy_to_CCT() (in module colour), 604

xy_to_CCT_Hernandez1999() (in module colour.temperature), 609

XY_TO_CCT_METHODS (in module colour), 605

xy_to_Luv_uv() (in module colour), 317

xy_to_UCS_uv() (in module colour), 320

xy_to_xyY() (in module colour), 307

xy_to_XYZ() (in module colour), 306

xyY_to_munsell_colour() (in module colour), 473

xyY_to_xy() (in module colour), 307

xyY_to_XYZ() (in module colour), 305

XYZ_ColourMatchingFunctions (class in module colour.colorimetry), 186

XYZ_outer_surface() (in module colour.volume), 659

XYZ_SCALING_CAT (in module colour.adaptation), 52

XYZ_to_ATD95() (in module colour), 84

XYZ_to_CAM16() (in module colour), 90

XYZ_to_CIECAM02() (in module colour), 86

XYZ_to_hdr_CIELab() (in module colour), 341

XYZ_to_hdr_IPT() (in module colour), 344

XYZ_to_Hunt() (in module colour), 95

XYZ_to_Hunter_Lab() (in module colour), 323

XYZ_to_Hunter_Rdab() (in module colour), 325

XYZ_to_IPT() (in module colour), 339

XYZ_to_JzAzBz() (in module colour), 348

XYZ_to_K_ab_HunterLab1966() (in module colour), 324

XYZ_to_Lab() (in module colour), 308

XYZ_to_LLAB() (in module colour), 98

XYZ_to_Luv() (in module colour), 312

XYZ_to_Nayatani95() (in module colour), 101

XYZ_to_OSA_UCS() (in module colour), 346

XYZ_to_RGB() (in module colour), 350

XYZ_to_RGB_matrix (colour.RGB_Colourspace attribute), 360

XYZ_to_RLAB() (in module colour), 104

XYZ_to_sd() (in module colour), 595

XYZ_to_sd_Meng2015() (in module colour.recovery), 599

XYZ_TO_SD_METHODS (in module colour), 597

XYZ_to_sRGB() (in module colour), 353

XYZ_to_UCS() (in module colour), 317

XYZ_to_UVW() (in module colour), 321

XYZ_to_xy() (in module colour), 305

XYZ_to_xyY() (in module colour), 304

Y

y (colour.KernelInterpolator attribute), 57

y (colour.LinearInterpolator attribute), 59

y (colour.NullInterpolator attribute), 60

y (colour.PchipInterpolator attribute), 61

y (colour.SpragueInterpolator attribute), 62

YCbCr_to_RGB() (in module colour), 454

YCBCR_WEIGHTS (in module colour), 456

YcCbCrc_to_RGB() (in module colour), 458

YCoCg_to_RGB() (*in module colour*), [462](#)
yellowness() (*in module colour*), [231](#)
yellowness_ASTMD1925() (in module
 colour:colorimetry), [232](#)
yellowness_ASTME313() (in module
 colour:colorimetry), [233](#)
YELLOWNESS_METHODS (*in module colour*), [232](#)