Colour Documentation

Release 0.3.16

Colour Developers

May 29, 2022

CONTENTS

Colour is an open-source Python package providing a comprehensive number of algorithms and datasets for colour science.

It is freely available under the New BSD License terms.

Colour is an affiliated project of NumFOCUS, a 501(c)(3) nonprofit in the United States.

Table of Contents

The draft release notes from the develop branch are available at this url.

CHAPTER

ONE

2 SPONSORS

We are grateful for the support of our sponsors. If you'd like to join them, please consider becoming a sponsor on OpenCollective.

CHAPTER

TWO

3 FEATURES

Colour features a rich dataset and collection of objects, please see the features page for more information.

CHAPTER

THREE

4 INSTALLATION

Colour and its primary dependencies can be easily installed from the Python Package Index by issuing this command in a shell:

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

The detailed installation procedure for the secondary dependencies is described in the Installation Guide.

Colour is also available for Anaconda from *Continuum Analytics* via conda-forge:

```
$ conda install -c conda-forge colour-science
```

CHAPTER

FOUR

5 DOCUMENTATION

4.1 5.1 Tutorial

The static tutorial provides an introduction to **Colour**. An interactive version is available via Google Colab.

4.2 5.2 How-To Guide

The How-To guide for **Colour** shows various techniques to solve specific problems and highlights some interesting use cases.

4.3 5.3 API Reference

The main technical reference for **Colour** and its API is the Colour Manual.

4.3.1 Colour Manual

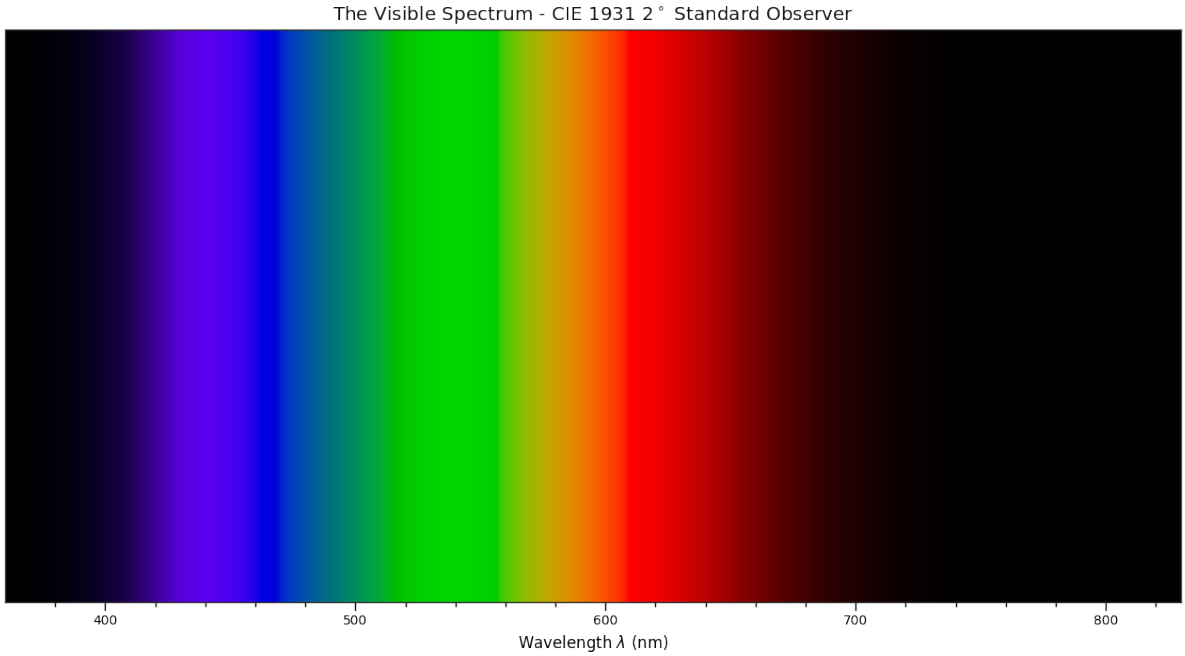
4.3.1.1 Tutorial

Note: An interactive version of the tutorial is available via Google Colab.

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()
```



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 correction, camera and display 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.
- *geometry*: Geometry primitives generation.
- *graph*: Graph for automatic colour conversions.
- *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

for sub_package in ('adaptation', 'algebra', 'appearance', 'biochemistry',
                    'blindness', 'characterisation', 'colorimetry',
                    'constants', 'continuous', 'contrast', 'corresponding',
                    'difference', 'geometry', 'graph', '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',
 'CAT_XYZ_SCALING',
 'CAT_VON_KRIES',
 'CAT_BRADFORD',
 'CAT_SHARP',
 '...']

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

Appearance
['InductionFactors_Hunt',
 'VIEWING_CONDITIONS_HUNT',
 'CAM_Specification_Hunt',
 'XYZ_to_Hunt',
 'CAM_Specification_ATD95',
 '...']

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

(continues on next page)

(continued from previous page)

```
Blindness
['CVD_MATRICES_MACHADO2010',
 'msds_cmfs_anomalous_trichromacy_Machado2009',
 'matrix_anomalous_trichromacy_Machado2009',
 'matrix_cvd_Machado2009',
 '...']

Characterisation
['RGB_CameraSensitivities',
 'RGB_DisplayPrimaries',
 'MSDS_ACES_RICD',
 'MSDS_CAMERA_SENSITIVITIES',
 'CCS_COLOURCHECKERS',
 '...']

Colorimetry
['SpectralShape',
 'SPECTRAL_SHAPE_DEFAULT',
 'SpectralDistribution',
 'MultiSpectralDistributions',
 'sds_and_msds_to_sds',
 '...']

Constants
['CONSTANT_K_M',
 'CONSTANT_KP_M',
 'CONSTANT_AVOGADRO',
 'CONSTANT_BOLTZMANN',
 'CONSTANT_LIGHT_SPEED',
 '...']

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

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

Corresponding
['BRENEMAN_EXPERIMENTS',
 'BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES',
 'CorrespondingColourDataset',
 'CorrespondingChromaticitiesPrediction',
 'corresponding_chromaticities_prediction_CIE1994',
 '...']

Difference
['delta_E_CAM02LCD',
 'delta_E_CAM02SCD',
 'delta_E_CAM02UCS',
 'delta_E_CAM16LCD',
```

(continues on next page)

(continued from previous page)

```
 'delta_E_CAM16SCD',
 '...']

Geometry
['PLANE_TO_AXIS_MAPPING',
 'primitive_grid',
 'primitive_cube',
 'PRIMITIVE_METHODS',
 'primitive',
 '...']

Graph
['CONVERSION_GRAPH',
 'CONVERSION_GRAPH_NODE_LABELS',
 'describe_conversion_path',
 'convert',
 '...']

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

Models
['Jab_to_JCh',
 'JCh_to_Jab',
 'COLOURSPACE_MODELS',
 'COLOURSPACE_MODELS_AXIS_LABELS',
 'XYZ_to_colourspace_model',
 '...']

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
['SD_ASTMG173_ETR',
 'SD_ASTMG173_GLOBAL_TILT',
 'SD_ASTMG173_DIRECT_CIRCUMSOLAR',
 'CONSTANTS_COLOUR_STYLE',
 'CONSTANTS_ARROW_STYLE',
 '...']
```

(continues on next page)

(continued from previous page)

```
Quality
['SDS_TCS',
 'SDS_VS',
 'ColourRendering_Specification_CRI',
 'colour_rendering_index',
 'ColourRendering_Specification_CQS',
 '...']

Recovery
['SPECTRAL_SHAPE_sRGB_MALLETT2019',
 'MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019',
 'SPECTRAL_SHAPE_OTSU2018',
 'BASIS_FUNCTIONS_OTSU2018',
 'CLUSTER_MEANS_OTSU2018',
 '...']

Temperature
['xy_to_CCT_CIE_D',
 'CCT_to_xy_CIE_D',
 'xy_to_CCT_Hernandez1999',
 'CCT_to_xy_Hernandez1999',
 'xy_to_CCT_Kang2002',
 '...']

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

Volume
['OPTIMAL_COLOUR_STIMULI_ILLUMINANTS',
 '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
----------
CCT_D_uv : ndarray
    Correlated colour temperature :math:`T_{cp}`, :math:`\Delta_{uv}`.
cmfs : XYZ_ColourMatchingFunctions, optional
    Standard observer colour matching functions.
Returns
-------
```

(continues on next page)

(continued from previous page)

```
ndarray
    *CIE UCS* colourspace *uv* chromaticity coordinates.
References
----------
:cite:`Ohno2014a`
Examples
--------
>>> from colour.colorimetry import (
...     SPECTRAL_SHAPE_DEFAULT, MSDS_CMFS_STANDARD_OBSERVER)
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 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:

```
pprint(colour.colorimetry.__all__)
```

```
['SpectralShape',
 'SPECTRAL_SHAPE_DEFAULT',
 'SpectralDistribution',
 'MultiSpectralDistributions',
 'sds_and_msds_to_sds',
 'sds_and_msds_to_msds',
 'sd_blackbody',
 'blackbody_spectral_radiance',
 'planck_law',
 'LMS_ConeFundamentals',
 'RGB_ColourMatchingFunctions',
 'XYZ_ColourMatchingFunctions',
 'MSDS_CMFS',
 'MSDS_CMFS_LMS',
 'MSDS_CMFS_RGB',
 'MSDS_CMFS_STANDARD_OBSERVER',
 'CCS_ILLUMINANTS',
 'SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES',
 'TVS_ILLUMINANTS_HUNTERLAB',
 'SDS_ILLUMINANTS',
 'CCS_LIGHT_SOURCES',
 'SDS_LIGHT_SOURCES',
 'SDS_LEFS',
 'SDS_LEFS_PHOTOPIC',
 'SDS_LEFS_SCOTOPIC',
 'sd_constant',
 'sd_zeros',
 'sd_ones',
 'msds_constant',
 'msds_zeros',
 'msds_ones',
 'SD_GAUSSIAN_METHODS',
 'sd_gaussian',
 'sd_gaussian_normal',
```

(continues on next page)

(continued from previous page)

```
 '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',
 'MSDS_TO_XYZ_METHODS',
 'sd_to_XYZ',
 'msds_to_XYZ',
 'SPECTRAL_SHAPE_ASTME308',
 'lagrange_coefficients_ASTME2022',
 'tristimulus_weighting_factors_ASTME2022',
 'adjust_tristimulus_weighting_factors_ASTME308',
 'sd_to_XYZ_integration',
 'sd_to_XYZ_tristimulus_weighting_factors_ASTME308',
 'sd_to_XYZ_ASTME308',
 'msds_to_XYZ_integration',
 'msds_to_XYZ_ASTME308',
 '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_ASTMD1535',
 'luminance_CIE1976',
 'luminance_Fairchild2010',
 'luminance_Fairchild2011',
 'intermediate_luminance_function_CIE1976',
 'dominant_wavelength',
 'complementary_wavelength',
 'excitation_purity',
 'colorimetric_purity',
 'luminous_flux',
 'luminous_efficiency',
 'luminous_efficacy',
 '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',
```

(continues on next page)

(continued from previous page)

```
 '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 quantity of datasets available in most sub-packages, for example the `colour.colorimetry.datasets` defines the following components:

```
pprint(colour.colorimetry.datasets.__all__)
```

```
['MSDS_CMFS',
 'MSDS_CMFS_LMS',
 'MSDS_CMFS_RGB',
 'MSDS_CMFS_STANDARD_OBSERVER',
 'CCS_ILLUMINANTS',
 'SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES',
 'TVS_ILLUMINANTS_HUNTERLAB',
 'SDS_ILLUMINANTS',
 'CCS_LIGHT_SOURCES',
 'SDS_LIGHT_SOURCES',
 'SDS_LEFS',
 'SDS_LEFS_PHOTOPIC',
 'SDS_LEFS_SCOTOPIC']
```

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.
data_sample = {
    380: 0.048,
    385: 0.051,
    390: 0.055,
    395: 0.060,
    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,
```

(continues on next page)

(continued from previous page)

```
    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,
    700: 0.282,
    705: 0.294,
    710: 0.305,
    715: 0.318,
    720: 0.334,
    725: 0.354,
    730: 0.372,
```

(continues on next page)

(continued from previous page)

```
    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(data_sample, 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],
                      [  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],
```

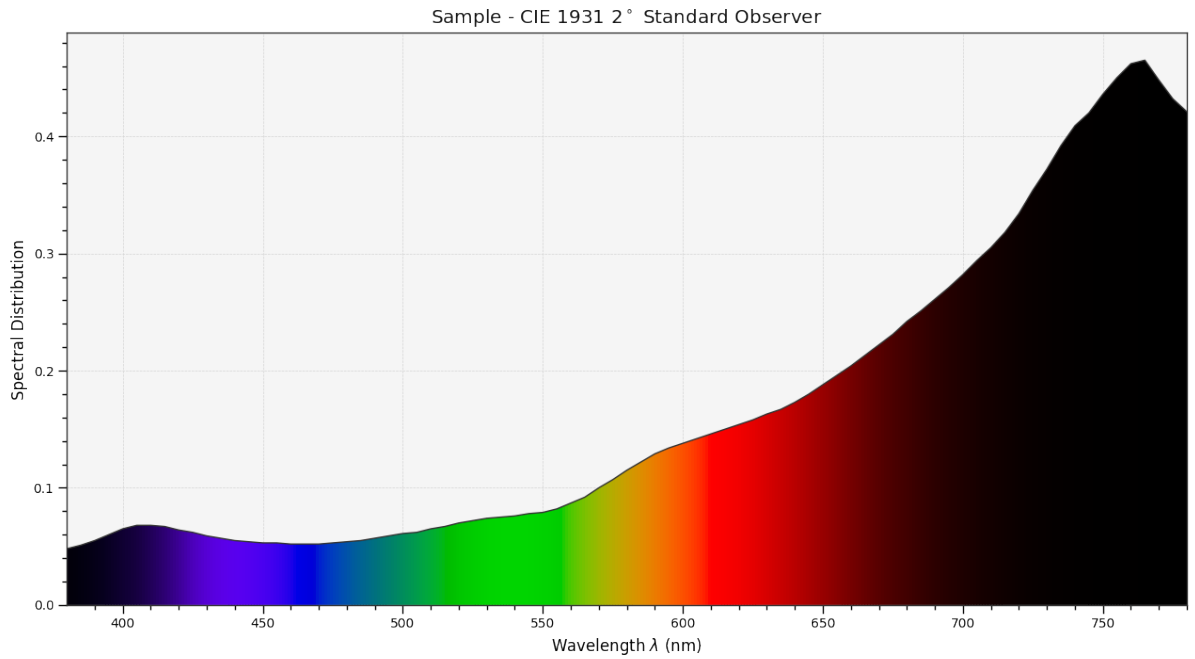
(continues on next page)

(continued from previous page)

```
                       [  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
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('\n"Zeros Filled Spectral Distribution"')
sd_zeros = colour.sd_zeros()
print(sd_zeros.shape)
print(sd_zeros[400])

# Defining a ones filled spectral distribution.
print('\n"Ones 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 [].

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

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

```
True
```

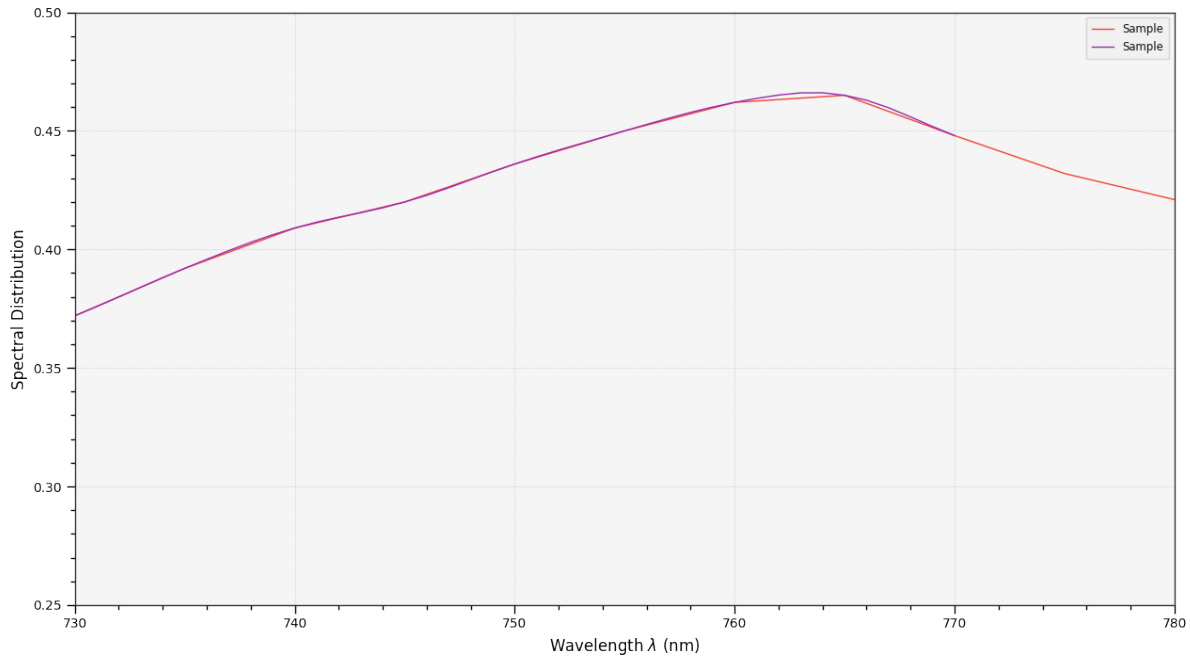
In this case, since the sample spectral distribution is uniform the interpolation defaults to the `colour.SpragueInterpolator` interpolator.

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()

# 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 []:

```
# 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',
 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],
```

(continues on next page)

(continued from previous page)

```
                       [  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]
```

```
(0.046999999999999348, 0.0)
```

Aligning a spectral distribution is a convenient way to first interpolates 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.]
```

A 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.]],
```

(continues on next page)

(continued from previous page)

```
       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])
```

Convert to Tristimulus Values

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

```
sd = colour.SpectralDistribution(data_sample)
cmfs = colour.MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
illuminant = colour.SDS_ILLUMINANTS['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]
```

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',
 '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',
```

(continues on next page)

(continued from previous page)

```
 'XYZ_to_UVW',
 'XYZ_to_hdr_CIELab',
 'XYZ_to_hdr_IPT',
 'XYZ_to_sRGB',
 'XYZ_to_xy',
 'XYZ_to_xyY',
 'XYZ_to_sd']
```

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_kwargs={'size': 'x-large'})
```

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.

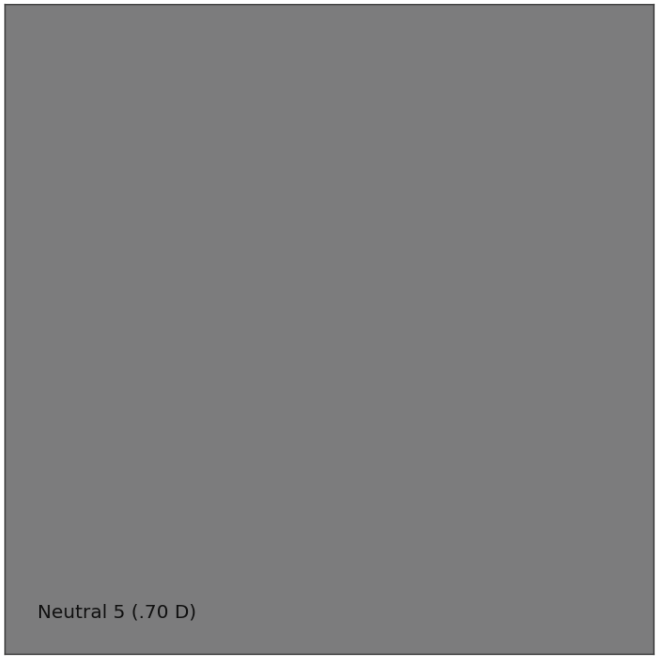
The `colour.characterisation` sub-package contains the dataset for various colour rendition charts:

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

# Colour rendition charts spectral distributions.
print(sorted(colour.characterisation.SDS_COLOURCHECKERS.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.SDS_COLOURCHECKERS['ColorChecker N Ohta'][patch_name]
XYZ = colour.sd_to_XYZ(patch_sd, cmfs, illuminant)
RGB = colour.XYZ_to_sRGB(XYZ / 100)

plot_single_colour_swatch(
    ColourSwatch(patch_name.title(), RGB),
    text_kwargs={'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_kwargs={'visible': False})
```

Convert to Chromaticity Coordinates

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

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

```
[ 0.31259787  0.32870029]
```

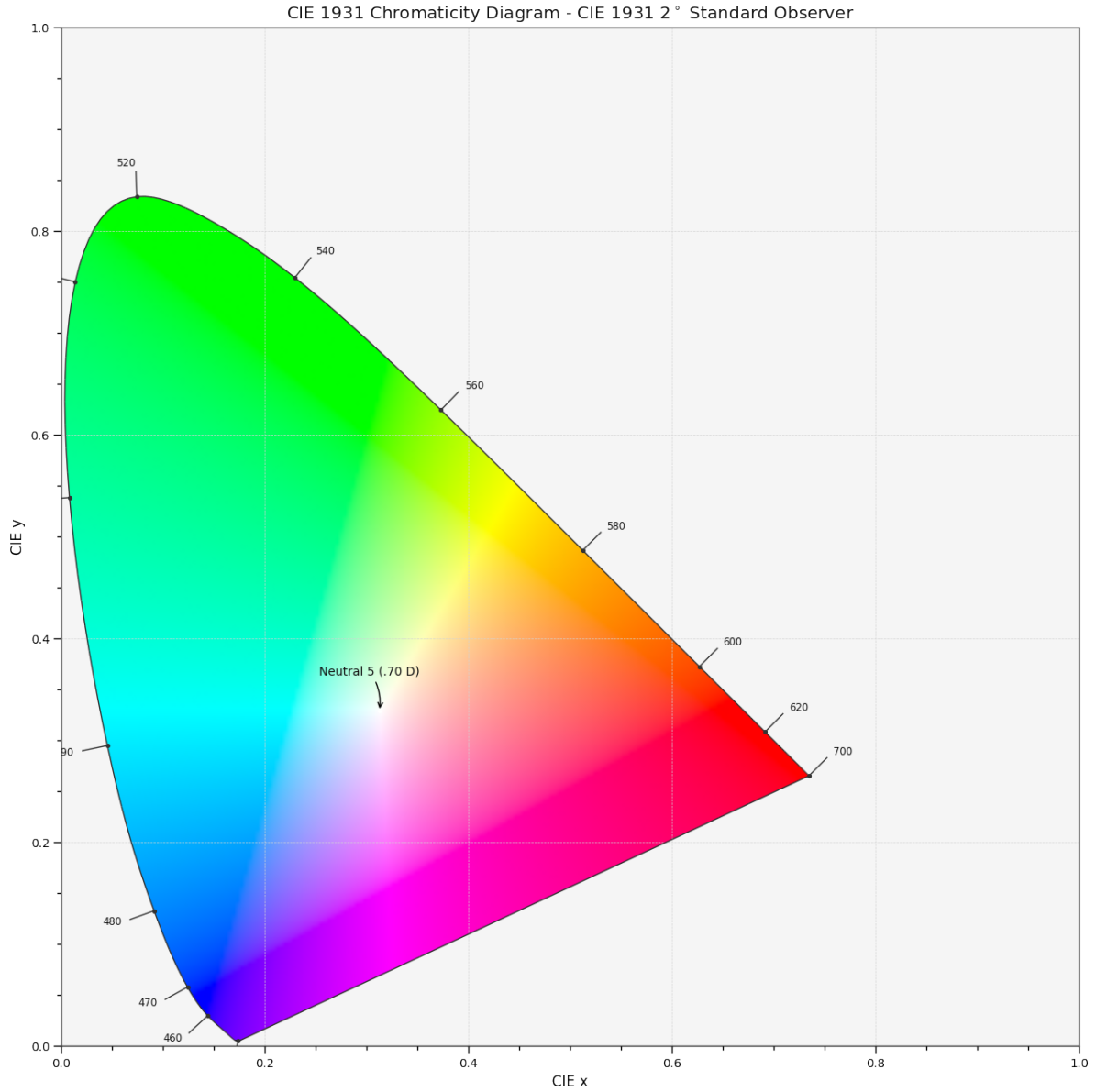
Chromaticity coordinates *CIE xy* can be plotted into the *CIE 1931 Chromaticity Diagram*:

```
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 *CIE 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',
```

(continues on next page)

(continued from previous page)

```
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3, rad=-0.2'))

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



See More

- The *Basics* page puts an emphasis on basic but important to understand concepts of **Colour**.
- The *Advanced* page describes some advanced usage scenarios of **Colour**.
- The How-To guide for **Colour** shows various techniques to solve specific problems and highlights some interesting use cases.

4.3.1.2 Basics

This page puts an emphasis on basic concepts of **Colour**, those are important to understand.

Table of Contents

Object Name Categorisation

The API tries to bundle the objects by categories by naming them with common prefixes which makes introspection and auto-completion easier.

For example, in IPython or Jupyter Notebook, most of the definitions pertaining to the spectral distribution handling can be found as follows:

```
In [1]: import colour

In [2]: colour.sd_
 sd_blackbody()                          sd_gaussian()                                sd_
↪rayleigh_scattering()                 sd_zeros
 sd_CIE_illuminant_D_series()            sd_mesopic_luminous_efficiency_function() sd_
↪single_led()
 sd_CIE_standard_illuminant_A()          sd_multi_leds()                              sd_
↪to_aces_relative_exposure_values()
 sd_constant()                           sd_ones()                                    sd_
↪to_XYZ
```

Likewise, for the spectral distribution handling related attributes:

```
In [2]: colour.SD
                    SD_GAUSSIAN_METHODS    SD_TO_XYZ_METHODS      SDS_ILLUMINANTS       SDS_
↪LIGHT_SOURCES
                    SD_MULTI_LEDS_METHODS SDS_COLOURCHECKERS     SDS_LEFS
                    SD_SINGLE_LED_METHODS SDS_FILTERS            SDS_LENSES
```

Similarly, all the RGB colourspaces can be individually accessed from the `colour.models` namespace:

```
In [2]: colour.models.RGB_COLOURSPACE
                          RGB_COLOURSPACE_ACES2065_1            RGB_COLOURSPACE_ACESPROXY  ␣
↪          RGB_COLOURSPACE_APPLE_RGB             RGB_COLOURSPACE_BT470_525
                          RGB_COLOURSPACE_ACESCC                RGB_COLOURSPACE_ADOBE_
↪RGB1998         RGB_COLOURSPACE_BEST_RGB               RGB_COLOURSPACE_BT470_625
                          RGB_COLOURSPACE_ACESCCT               RGB_COLOURSPACE_ADOBE_WIDE_
↪GAMUT_RGB RGB_COLOURSPACE_BETA_RGB              RGB_COLOURSPACE_BT709                 >
                          RGB_COLOURSPACE_ACESCG                RGB_COLOURSPACE_ALEXA_WIDE_
↪GAMUT      RGB_COLOURSPACE_BT2020                RGB_COLOURSPACE_CIE_RGB
```

Abbreviations

The following abbreviations are in use in Colour:

- **CAM** : Colour Appearance Model
- **CCS** : Chromaticity Coordinates
- **CCTF** : Colour Component Transfer Function
- **CCT** : Correlated Colour Temperature
- **CMY** : Cyan, Magenta, Yellow
- **CMYK** : Cyan, Magenta, Yellow, Black
- **CVD** : Colour Vision Deficiency
- **CV** : Code Value
- **EOTF** : Electro-Optical Transfer Function
- **IDT** : Input Device Transform
- **MSDS** : Multi-Spectral Distributions
- **OETF** : Optical-Electrical Transfer Function
- **OOTF** : Optical-Optical Transfer Function
- **SD** : Spectral Distribution
- **TVS** : Tristimulus Values

N-Dimensional Array Support

Most of **Colour** definitions are fully vectorised and support n-dimensional array by leveraging Numpy. While it is recommended to use ndarray 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:

```
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_kwargs={'text': 'sRGB to XYZ'})
```

Spectral Representation and Continuous Signal

Floating Point Wavelengths

Colour current representation of spectral data is atypical and has been influenced by the failures and shortcomings of the previous implementation that required less than ideal code to support floating point wavelengths. Wavelengths should not have to be defined as integer values and it is effectively common to get data from instruments whose domain is returned as floating point values.

For example, the data from an Ocean Insight (Optics) STS-VIS spectrometer is typically saved with 3 digits decimal precision:

```
Data from Subt2_14-36-15-210.txt Node

Date: Sat Nov 17 14:36:15 NZDT 2018
User: kelsolaar
Spectrometer: S12286
Trigger mode: 0
Resolution mode: 1024 pixels
Integration Time (sec): 5.000000E0
Scans to average: 3
Nonlinearity correction enabled: true
Boxcar width: 3
Baseline correction enabled: true
XAxis mode: Wavelengths
Number of Pixels in Spectrum: 1024
>>>>>Begin Spectral Data<<<<<
338.028     279.71
338.482     285.43
338.936     291.33
...
821.513     3112.65
822.008     3133.74
822.503     3107.11
```

A solution to the problem is to quantize the data at integer values but it is often non-desirable. The spectra representation implementation prior to **Colour 0.3.11** was relying on a custom mutable mapping which was allowing to retrieve decimal keys within a given precision:

```
data_1 = {0.1999999998: 'Nemo', 0.2000000000: 'John'}
apm_1 = ArbitraryPrecisionMapping(data_1, key_decimals=10)
tuple(apm_1.keys())
```

```
(0.1999999998, 0.2)
```

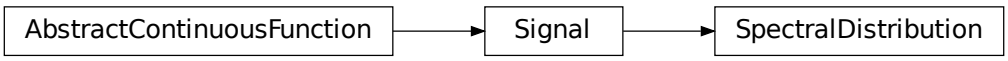
```
apm_2 = ArbitraryPrecisionMapping(data_1, key_decimals=7)
tuple(apm_2.keys())
```

```
(0.2,)
```

While functional, the approach was brittle and not elegant which triggered a significant amount of rework.

Continuous Signal

All the spectral distributions in **Colour** are instances of the `colour.SpectralDistribution` class (or its sub-classes), a sub-class of the `colour.continuous.Signal` class which is itself an implementation of the `colour.continuous.AbstractContinuousFunction` ABCMeta class:



Likewise, the multi-spectral distributions are instances `colour.MultiSpectralDistributions` class (or its sub-classes), a sub-class of the `colour.continuous.MultiSignals` class which is a container for multiple `colour.continuous.Signal` sub-class instances and also implements the `colour.continuous.AbstractContinuousFunction` ABCMeta class.

The `colour.continuous.Signal` 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.

Consequently, it is possible to get the value of a spectral distribution at any given wavelength:

```
data = {
    500: 0.0651,
```

(continues on next page)

(continued from previous page)

```
    520: 0.0705,
    540: 0.0772,
    560: 0.0870,
    580: 0.1128,
    600: 0.1360
}
sd = colour.SpectralDistribution(data)
sd[555.5]
```

```
0.083453673782958995
```

Getting, Setting, Indexing and Slicing

Attention: Indexing a spectral distribution (or multi-spectral distribution) with a numeric (or a numeric sequence) returns the corresponding value(s). Indexing a spectral distribution (or multi-spectral distribution) with a slice returns the values for the corresponding wavelength *indexes*.

While it is tempting to think that the `colour.SpectralDistribution` and `colour.MultiSpectralDistributions` classes behave like Numpy's `ndarray`, they do not entirely and some peculiarities exist that make them different.

An important difference lies in the behaviour with respect to getting and setting the values of the data.

Getting the value(s) for a single (or multiple wavelengths) is done by indexing the `colour.SpectralDistribution` (or `colour.MultiSpectralDistributions`) class with the a single numeric or array of numeric wavelengths, e.g. `sd[555.5]` or `sd[555.25, 555.25, 555.75]`.

However, if getting the values using a `slice` class instance, e.g. `sd[0:3]`, the underlying discrete values for the indexes represented by the `slice` class instance are returned instead.

As shown in the previous section, getting the value of a wavelength is done as follows:

```
data = {
    500: 0.0651,
    520: 0.0705,
    540: 0.0772,
    560: 0.0870,
    580: 0.1128,
    600: 0.1360
}
sd = colour.SpectralDistribution(data)
sd[555]
```

```
0.083135180664062502,
```

Multiple wavelength values can be retrieved as follows:

```
sd[(555.0, 556.25, 557.5, 558.75, 560.0)]
```

```
array([ 0.08313518,  0.08395997,  0.08488108,  0.085897  ,  0.087     ])
```

However, slices will return the values for the corresponding wavelength *indexes*:

```
sd[0:3]
```

```
array([ 0.0651,  0.0705,  0.0772])
```

```
sd[:]
```

```
array([ 0.0651,  0.0705,  0.0772,  0.087 ,  0.1128,  0.136 ])
```

Note: Indexing a multi-spectral distribution is achieved similarly, it can however be sliced along multiple axes because the data is2-dimensional, e.g. *msds[0:3, 0:2]*.

A *copy* of the underlying `colour.SpectralDistribution` and `colour.MultiSpectralDistributions` classes discretized data can be accessed via the `wavelengths` and `values` properties. However, it cannot be changed directly via the properties or slicing:

Attention: The data returned by the `wavelengths` and `values` properties is a *copy* of the underlying `colour.SpectralDistribution` and `colour.MultiSpectralDistributions` classes discretized data: It can only be changed indirectly.

```
data = {
    500: 0.0651,
    520: 0.0705,
    540: 0.0772,
    560: 0.0870,
    580: 0.1128,
    600: 0.1360
}
sd = colour.SpectralDistribution(data)
# Note: The wavelength 500nm is at index 0.
sd.values[0] = 0
sd[500]
```

```
0.065100000000000019
```

Instead, the values can be set indirectly:

```
values = sd.values
values[0] = 0
sd.values = values
sd.values
```

```
array([ 0.    ,  0.0705,  0.0772,  0.087 ,  0.1128,  0.136 ])
```

Domain-Range Scales

Note: This section contains 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 *360*.
- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.
- *Dimensionless* values are unaffected and are indicated with *DN*.
- *Unaffected* values are unaffected and are indicated with *UN*.

Warning: The conversion to **'1'** domain-range scale is a *soft* normalisation and similarly to the **'Reference'** domain-range scale it is normal to encounter values exceeding *1*, e.g. High Dynamic Range Imagery (HDRI) or negative values, e.g. out-of-gamut RGB colourspace values. Some definitions such as `colour.models.eotf_ST2084()` which decodes absolute luminance values are not affected by any domain-range scales and are indicated with *UN*.

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.

4.3.1.3 Advanced

This page describes some advanced usage scenarios of **Colour**.

Table of Contents

Environment

Various environment variables can be used to modify **Colour** behaviour at runtime:

- *COLOUR_SCIENCE__FLOAT_PRECISION*: Sets the float precision for most of **Colour** computations. Possible values are *float16*, *float32* and *float64* (default). Changing float precision might result in various **Colour** functionality breaking entirely. *With great power comes great responsibility*.
- *COLOUR_SCIENCE__INT_PRECISION*: Sets the integer precision for most of **Colour** computations. Possible values are *int8*, *int16*, *int32*, and *int64* (default). Changing integer precision *will almost certainly break* **Colour**! *With great power comes great responsibility*.
- *COLOUR_SCIENCE__COLOUR__SHOW_WARNINGS_WITH_TRACEBACK*: results in the `warnings.showwarning()` definition to be replaced with the `colour.utilities.show_warning()` definition and thus providing complete traceback from the point where the warning occurred.

Using Colour without Scipy

With the release of Colour 0.3.8, SciPy became a requirement.

Scipy is notoriously hard to compile, especially on Windows. Some Digital Content Creation (DCC) applications are shipping Python interpreters compiled with versions of Visual Studio such as 2011 or 2015. Those are incompatible with the Python Wheels commonly built with Visual Studio 2008 (Python 2.7) or Visual Studio 2017 (Python 3.6).

It is however possible to use **Colour** in a partially broken and mock **Scipy** by using the mock_for_colour.py module.

Assuming it is available for import, a typical usage would be as follows:

```
import sys
from mock_for_colour import MockModule

for module in ('scipy', 'scipy.interpolate', 'scipy.spatial',
               'scipy.spatial.distance', 'scipy.optimize'):
    sys.modules[str(module)] = MockModule(str(module))

import colour

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

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

Or directly using the `mock_scipy_for_colour` definition:

```
from mock_for_colour import mock_scipy_for_colour

mock_scipy_for_colour()

import colour

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

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

Anything relying on the spectral code will be unusable, but a great amount of useful functionality will still be available.

4.3.1.4 Reference

Colour

Chromatic Adaptation

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

Chromatic Adaptation

colour

chromatic_adaptation(XYZ, XYZ_w, XYZ_wr[, ...])	Adapts given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_METHODS	Supported chromatic adaptation methods.
VIEWING_CONDITIONS_CMCCAT2000	Reference *CMCCAT2000* 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 whitepoint.
- **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.
- **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', 'Inverse'}**, 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** (InductionFactors_CMCCAT2000, 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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, 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

[], [], [], [], [], []

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)
...
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)
...
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)
...
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)
...
array([ 0.2332526...,  0.2332455...,  0.7611593...])
```

colour.CHROMATIC_ADAPTATION_METHODS

colour.**CHROMATIC_ADAPTATION_METHODS = CaseInsensitiveMapping({'CIE 1994': ..., 'CMCCAT2000': ..., 'Fairchild 1990': ..., 'Von Kries': ...})**

Supported chromatic adaptation methods.

References

[], [], [], [], [], []

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

colour.VIEWING_CONDITIONS_CMCCAT2000

colour.**VIEWING_CONDITIONS_CMCCAT2000 = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[], []

VIEWING_CONDITIONS_CMCCAT2000 [CaseInsensitiveMapping] ('Average', 'Dim', 'Dark')

Dataset

colour

CHROMATIC_ADAPTATION_TRANSFORMS	Chromatic adaptation transforms.

colour.CHROMATIC_ADAPTATION_TRANSFORMS

colour.**CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Kries': ..., 'Bradford': ..., 'Sharp': ..., 'Fairchild': ..., 'CMCCAT97': ..., 'CMCCAT2000': ..., 'CAT02': ..., 'CAT02 Brill 2008': ..., 'Bianco 2010': ..., 'Bianco PC 2010': ...})**

Chromatic adaptation transforms.

References

[], [], [], [], [], [], [], []

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild, 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**

Fairchild (1990)

colour.adaptation

chromatic_adaptation_Fairchild1990(XYZ_1, ...)	Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *Fairchild (1990)* 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 whitepoint.
- **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

[], []

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)
...
array([ 23.3252634...,  23.3245581...,  76.1159375...])
```

CIE 1994

colour.adaptation

chromatic_adaptation_CIE1994(XYZ_1, xy_o1, ...)	Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *CIE 1994* 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

[]

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)
...
array([ 24.0337952...,  21.1562121...,  17.6430119...])
```

CMCCAT2000

colour.adaptation

chromatic_adaptation_CMCCAT2000(XYZ, XYZ_w, ...)	Adapts given stimulus *CIE XYZ* tristimulus values using given viewing conditions.
VIEWING_CONDITIONS_CMCCAT2000	Reference *CMCCAT2000* 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=InductionFactors_CMCCAT2000(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_inverse_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** (InductionFactors_CMCCAT2000, optional) – Surround viewing conditions induction factors.
- **direction** (unicode, optional) – **{'Forward', 'Inverse'}**, 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

[], []

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')
...
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

Using the *CMCCAT2000* inverse 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='Inverse')
...
array([ 22.48,  22.74,   8.54])
```

colour.adaptation.VIEWING_CONDITIONS_CMCCAT2000

colour.adaptation.**VIEWING_CONDITIONS_CMCCAT2000 = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[], []

VIEWING_CONDITIONS_CMCCAT2000 [CaseInsensitiveMapping] ('Average', 'Dim', 'Dark')

Ancillary Objects

`colour.adaptation`

`chromatic_adaptation_forward_CMCCAT2000`(XYZ, ...)	Adapts given stimulus *CIE XYZ* tristimulus values from test viewing conditions to reference viewing conditions using *CMCCAT2000* forward chromatic adaptation model.
`chromatic_adaptation_inverse_CMCCAT2000`(...)	Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* inverse chromatic adaptation model.
`InductionFactors_CMCCAT2000`(F)	*CMCCAT2000* 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=InductionFactors_CMCCAT2000(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 whitepoint.
- **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** (`InductionFactors_CMCCAT2000`, `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

[], []

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)
...
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

colour.adaptation.chromatic_adaptation_inverse_CMCCAT2000

colour.adaptation.**chromatic_adaptation_inverse_CMCCAT2000**(*XYZ_c*, *XYZ_w*, *XYZ_wr*, *L_A1*, *L_A2*, *surround=InductionFactors_CMCCAT2000(F=1)*)

Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* inverse 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 whitepoint.
- **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** (InductionFactors_CMCCAT2000, 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

[], []

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_inverse_CMCCAT2000(XYZ_c, XYZ_w, XYZ_wr, L_A1,
...                                         L_A2)
...
array([ 22.4839876...,  22.7419485...,   8.5393392...])
```

colour.adaptation.InductionFactors_CMCCAT2000

class colour.adaptation.**InductionFactors_CMCCAT2000**(F)

CMCCAT2000 chromatic adaptation model induction factors.

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

References

[], []

Create new instance of InductionFactors_CMCCAT2000(F,)

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

F	Alias for field number 0

Von Kries

colour.adaptation

chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)	Adapts given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_TRANSFORMS	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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, 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

[]

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)
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)
...
array([ 0.2166600...,  0.1260477...,  0.0385506...])
```

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS

colour.adaptation.**CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Kries': ..., 'Bradford': ..., 'Sharp': ..., 'Fairchild': ..., 'CMCCAT97': ..., 'CMCCAT2000': ..., 'CAT02': ..., 'CAT02 Brill 2008': ..., 'Bianco 2010': ..., 'Bianco PC 2010': ...})**

Chromatic adaptation transforms.

References

[], [], [], [], [], [], [], []

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild, 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**

Dataset

colour.adaptation

CAT_BRADFORD	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_BIANCO2010	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_PC_BIANCO2010	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CAT02_BRILL2008	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CAT02	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CMCCAT2000	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_CMCCAT97	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_FAIRCHILD	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_SHARP	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_VON_KRIES	ndarray(shape, dtype=float, buffer=None, offset=0,
CAT_XYZ_SCALING	ndarray(shape, dtype=float, buffer=None, offset=0,

colour.adaptation.CAT_BRADFORD

colour.adaptation.**CAT_BRADFORD = array([[0.8951, 0.2664, -0.1614], [-0.7502, 1.7135, 0.0367], [0.0389, -0.0685, 1.0296]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_BIANCO2010

colour.adaptation.**CAT_BIANCO2010 = array([[0.8752, 0.2787, -0.1539], [-0.8904, 1.8709, 0.0195], [-0.0061, 0.0162, 0.9899]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_PC_BIANCO2010

colour.adaptation.**CAT_PC_BIANCO2010 = array([[0.6489, 0.3915, -0.0404], [-0.3775, 1.3055, 0.072], [-0.0271, 0.0888, 0.9383]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_CAT02_BRILL2008

colour.adaptation.**CAT_CAT02_BRILL2008 = array([[0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [0. , 0. , 1.]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_CAT02

colour.adaptation.**CAT_CAT02 = array([[0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [0.003 , 0.0136, 0.9834]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_CMCCAT2000

colour.adaptation.**CAT_CMCCAT2000 = array([[7.98200000e-01, 3.38900000e-01, -1.37100000e-01], [-5.91800000e-01, 1.55120000e+00, 4.06000000e-02], [8.00000000e-04, 2.39000000e-02, 9.75300000e-01]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_CMCCAT97

colour.adaptation.**CAT_CMCCAT97 = array([[0.8951, -0.7502, 0.0389], [0.2664, 1.7135, 0.0685], [-0.1614, 0.0367, 1.0296]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_FAIRCHILD

colour.adaptation.**CAT_FAIRCHILD = array([[0.8562, 0.3372, -0.1934], [-0.836 , 1.8327, 0.0033], [0.0357, -0.0469, 1.0112]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_SHARP

colour.adaptation.**CAT_SHARP = array([[1.2694, -0.0988, -0.1706], [-0.8364, 1.8006, 0.0357], [0.0297, -0.0315, 1.0018]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_VON_KRIES

colour.adaptation.**CAT_VON_KRIES = array([[0.40024, 0.7076 , -0.08081], [-0.2263 , 1.16532, 0.0457], [0. , 0. , 0.91822]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time `(2 * 4)`.

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.adaptation.CAT_XYZ_SCALING

colour.adaptation.**CAT_XYZ_SCALING = array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

Ancillary Objects

`colour.adaptation`

`matrix_chromatic_adaptation_VonKries`(XYZ_w, ...)	Computes the *chromatic adaptation* matrix from test viewing conditions to reference viewing conditions.

colour.adaptation.matrix_chromatic_adaptation_VonKries

`colour.adaptation.`**`matrix_chromatic_adaptation_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 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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, 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

[]

Examples

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> matrix_chromatic_adaptation_VonKries(XYZ_w, XYZ_wr)
...
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'
>>> matrix_chromatic_adaptation_VonKries(XYZ_w, XYZ_wr, method)
...
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

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

colour.Extrapolator

class colour.**Extrapolator**(*interpolator=None*, *method='Linear'*, *left=None*, *right=None*, *dtype=None*)

Bases: object

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 < xi[0].
- **right** (numeric, optional) – Value to return for x > xi[-1].
- **dtype** (type) – Data type used for internal conversions.

Methods

- __init__()
- __class__()

Notes

- The interpolator must define *x* and *y* attributes.

References

[], []

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=None*)

property interpolator

Getter and setter property for the *Colour* or *scipy* interpolator class instance.

Parameters **value** (`callable`) – Value to set the *Colour* or *scipy* interpolator class instance with.

Returns *Colour* or *scipy* interpolator class instance.

Return type callable

__weakref__

list of weak references to the object (if defined)

property method

Getter and setter property for the extrapolation method.

Parameters **value** (`unicode`) – Value to set the extrapolation method. with.

Returns Extrapolation method.

Return type unicode

property left

Getter and setter property for left value to return for x < xi[0].

Parameters **value** (`numeric`) – Left value to return for x < xi[0].

Returns Left value to return for x < xi[0].

Return type numeric

property right

Getter and setter property for right value to return for x > xi[-1].

Parameters **value** (numeric) – Right value to return for x > xi[-1].

Returns Right value to return for x > xi[-1].

Return type numeric

__call__(*x*)

Evaluates the Extrapolator at given point(s).

Parameters **x** (numeric or array_like) – Point(s) to evaluate the Extrapolator at.

Returns Extrapolated points value(s).

Return type float or ndarray

Interpolation

colour

KernelInterpolator(x, y[, window, kernel, ...])	Kernel based interpolation of a 1-D function.
NearestNeighbourInterpolator(*args, **kwargs)	A nearest-neighbour interpolator.
LinearInterpolator(x, y[, dtype])	Linearly interpolates a 1-D function.
NullInterpolator(x, y[, absolute_tolerance, ...])	Performs 1-D function null interpolation, i.e. a call within given tolerances will return existing y variable values and default if outside tolerances.
PchipInterpolator(x, y, *args, **kwargs)	Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.
SpragueInterpolator(x, y[, dtype])	Constructs a fifth-order polynomial that passes through y dependent variable.

colour.KernelInterpolator

class colour.**KernelInterpolator**(*x*, *y*, *window=3*, *kernel=<function kernel_lanczos>*, *kernel_kwargs=None*, *padding_kwargs=None*, *dtype=None*)

Bases: object

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_kwargs** (dict, optional) – Arguments to use when calling the kernel.

- **padding_kwargs** (dict, optional) – Arguments to use when padding y variable values with the `np.pad()` definition.
- **dtype** (type) – Data type used for internal conversions.

Attributes

- x
- y
- window
- kernel
- kernel_kwargs
- padding_kwargs

Methods

- __init__()
- __call__()

References

[], []

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)
6.9411400...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75])
array([ 6.1806208...,  8.0823848...])
```

Using a different *lanczos* kernel:

```
>>> f = KernelInterpolator(x, y, kernel=kernel_sinc)
>>> f([0.25, 0.75])
array([ 6.5147317...,  8.3965466...])
```

Using a different window size:

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

(continues on next page)

(continued from previous page)

```
>>> f([0.25, 0.75])
array([ 5.3961792...,  5.6521093...])
```

__init__(*x*, *y*, *window=3*, *kernel=<function kernel_lanczos>*, *kernel_kwargs=None*, *padding_kwargs=None*, *dtype=None*)

property x

Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

property window

Getter and setter property for the window.

Parameters **value** (int) – Value to set the window with.

Returns Window.

Return type int

property kernel

Getter and setter property for the kernel callable.

Parameters **value** (callable) – Value to set the kernel callable.

Returns Kernel callable.

Return type callable

property kernel_kwargs

Getter and setter property for the kernel call time arguments.

Parameters **value** (dict) – Value to call the interpolation kernel with.

Returns Kernel call time arguments.

Return type dict

property padding_kwargs

Getter and setter property for the kernel call time arguments.

Parameters **value** (dict) – Value to call the interpolation kernel with.

Returns Kernel call time arguments.

Return type dict

__call__(*x*)

Evaluates the interpolator at given point(s).

Parameters **x** (numeric or array_like) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type float or ndarray

__weakref__
list of weak references to the object (if defined)

colour.NearestNeighbourInterpolator

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

Bases: colour.algebra.interpolation.KernelInterpolator

A nearest-neighbour interpolator.

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_kwargs** (dict, optional) – Arguments to use when padding y variable values with the np.pad() definition.
- **dtype** (type) – Data type used for internal conversions.

Methods

- __init__()

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

colour.LinearInterpolator

class colour.**LinearInterpolator**(*x*, *y*, *dtype=None*)

Bases: object

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.

Attributes

- x
- y

Methods

- `__init__()`
- `__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)
7.64...
```

Interpolating an *array_like* variable:

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

__init__(*x*, *y*, *dtype=None*)

property x

Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

__call__(*x*)

Evaluates the interpolating polynomial at given point(s).

Parameters **x** (numeric or array_like) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type float or ndarray

__weakref__

list of weak references to the object (if defined)

colour.NullInterpolator

class colour.**NullInterpolator**(*x*, *y*, *absolute_tolerance=1e-06*, *relative_tolerance=1e-06*, *default=nan*, *dtype=None*)

Bases: object

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.

Attributes

- x
- y
- relative_tolerance
- absolute_tolerance
- default

Methods

- __init__()
- __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)
9.3699999...
>>> f = NullInterpolator(x, y, absolute_tolerance=0.01)
>>> f(1.01)
9.3699999...
```

__weakref__

list of weak references to the object (if defined)

__init__(*x*, *y*, *absolute_tolerance=1e-06*, *relative_tolerance=1e-06*, *default=nan*, *dtype=None*)

property x

Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

property relative_tolerance

Getter and setter property for the relative tolerance.

Parameters **value** (numeric) – Value to set the relative tolerance with.

Returns Relative tolerance.

Return type numeric

property absolute_tolerance

Getter and setter property for the absolute tolerance.

Parameters **value** (numeric) – Value to set the absolute tolerance with.

Returns Absolute tolerance.

Return type numeric

property default

Getter and setter property for the default value for call outside tolerances.

Parameters **value** (numeric) – Value to set the default value with.

Returns Default value.

Return type numeric

__call__(x)

Evaluates the interpolator at given point(s).

Parameters **x** (numeric or array_like) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type float or ndarray

colour.PchipInterpolator

class colour.**PchipInterpolator**(*x*, *y*, **args*, ***kwargs*)

Bases: scipy.interpolate._cubic.PchipInterpolator

Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.

Attributes

- y

Methods

- __init__()

Notes

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

__init__(*x*, *y*, **args*, ***kwargs*)

property y

Getter property for the dependent and already known y variable.

Returns Dependent and already known y variable.

Return type array_like

colour.SpragueInterpolator

class colour.**SpragueInterpolator**(*x*, *y*, *dtype=None*)

Bases: object

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.

Attributes

- x
- y

Methods

- __init__()
- __call__()

Notes

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

References

[], []

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)
7.2185025...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75])
array([ 6.7295161...,  7.8140625...])
```

SPRAGUE_C_COEFFICIENTS = array([[884, -1960, 3033, -2648, 1080, -180], [508, -540, 488, -367, 144, -24], [-24, 144, -367, 488, -540, 508], [-180, 1080, -2648, 3033, -1960, 884]])

Defines the coefficients used to generate extra points for boundaries interpolation.

SPRAGUE_C_COEFFICIENTS : array_like, (4, 6)

References

[]

__init__(*x*, *y*, *dtype=None*)

property x

Getter and setter property for the independent x variable.

Parameters **value** (`array_like`) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (`array_like`) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

__call__(*x*)

Evaluates the interpolating polynomial at given point(s).

Parameters **x** (`numeric or array_like`) – Point(s) to evaluate the interpolant at.

Returns Interpolated value(s).

Return type numeric or ndarray

__weakref__

list of weak references to the object (if defined)

lagrange_coefficients(r[, n])	Computes the *Lagrange Coefficients* at given point r for degree n.
TABLE_INTERPOLATION_METHODS	Supported table interpolation methods.
table_interpolation(V_xyz, table[, method])	Performs interpolation of given V_{xyz} values using given interpolation table.

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.

Return type ndarray

References

[], []

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

[], []

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 (NxNxNx3) interpolation table.
- **method** (unicode, optional) – **{'Trilinear', 'Tetrahedral'}**, Interpolation method.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[], []

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.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)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation(V_xyz, table)
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')
...
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Interpolation Kernels

colour

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

[]

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

[]

Examples

```
>>> kernel_linear(np.linspace(0, 1, 10))
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

[]

Examples

```
>>> kernel_sinc(np.linspace(0, 1, 10))
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

[]

Examples

```
>>> kernel_lanczos(np.linspace(0, 1, 10))
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

[]

Examples

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

Ancillary Objects

colour.algebra

table_interpolation_trilinear(V_xyz, table)	Performs trilinear interpolation of given V_{xyz} values using given interpolation table.
table_interpolation_tetrahedral(V_xyz, table)	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 (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.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)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_trilinear(V_xyz, table)
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 (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.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)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_tetrahedral(V_xyz, table)
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Coordinates

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).
spherical_to_cartesian(a)	Transforms given spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to cartesian coordinates array xyz.
cartesian_to_polar(a)	Transforms given cartesian coordinates array xy to polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).
polar_to_cartesian(a)	Transforms given polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to cartesian coordinates array xy.
cartesian_to_cylindrical(a)	Transforms given cartesian coordinates array xyz to cylindrical coordinates array $\rho\phi z$ (radial distance, azimuth and height).
cylindrical_to_cartesian(a)	Transforms given cylindrical coordinates array $\rho\phi z$ (radial distance, azimuth 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 $\rho\theta\phi$, ρ is in range [0, +inf], θ is in range [0, pi] radians, i.e. [0, 180] degrees, and ϕ is in range [-pi, pi] radians, i.e. [-180, 180] degrees.

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_spherical(a)
array([ 6.7823299...,  0.4850497...,  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, ρ is in range [0, +inf], θ is in range [0, pi] radians, i.e. [0, 180] degrees, and ϕ is in range [-pi, pi] radians, i.e. [-180, 180] degrees.

Returns Cartesian coordinates array xyz.

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([6.78232998, 0.48504979, 0.32175055])
>>> spherical_to_cartesian(a)
array([ 3.0000000...,  0.9999999...,  5.9999999...])
```

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 $\rho\phi$, ρ is in range [0, +inf], ϕ is in range [-pi, pi] radians, i.e. [-180, 180] degrees.

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([3, 1])
>>> cartesian_to_polar(a)
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, ρ is in range [0, +inf], ϕ is in range [-pi, pi] radians i.e. [-180, 180] degrees.

Returns Cartesian coordinates array xy.

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([3.16227766, 0.32175055])
>>> polar_to_cartesian(a)
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 $\rho\phi z$ (radial distance, azimuth and height).

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

Returns Cylindrical coordinates array $\rho\phi z$, ρ is in range [0, +inf], ϕ is in range [-pi, pi] radians i.e. [-180, 180] degrees, z is in range [0, +inf].

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_cylindrical(a)
array([ 3.1622776...,  0.3217505...,  6.        ])
```

colour.algebra.cylindrical_to_cartesian

colour.algebra.**cylindrical_to_cartesian**(*a*)

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

Parameters **a** (array_like) – Cylindrical coordinates array $\rho\phi z$ to transform, ρ is in range [0, +inf], ϕ is in range [-pi, pi] radians i.e. [-180, 180] degrees, z is in range [0, +inf].

Returns Cartesian coordinates array xyz.

Return type ndarray

References

[], []

Examples

```
>>> a = np.array([3.16227766, 0.32175055, 6.00000000])
>>> cylindrical_to_cartesian(a)
array([ 3.        ,  0.9999999...,  6.        ])
```

Geometry

colour.algebra

normalise_vector(a)	Normalises given vector a.
euclidean_distance(a, b)	Returns the euclidean distance between point arrays a and b.
extend_line_segment(a, b[, distance])	Extends the line segment defined by point arrays a and b by given distance and return the new end point.
intersect_line_segments(l_1, l_2)	Computes l_1 line segments intersections with l_2 line segments.
ellipse_coefficients_general_form(coefficients)	Returns the general form ellipse coefficients from given canonical form ellipse coefficients.
ellipse_coefficients_canonical_form(coefficients)	Returns the canonical form ellipse coefficients from given general form ellipse coefficients.
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.
ELLIPSE_FITTING_METHODS	Supported ellipse fitting methods.
ellipse_fitting(a[, method])	Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a 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)
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)
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

[]

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)
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

[], []

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
array([[[        nan,         nan],
        [ 0.2279184...,  0.6006430...],
        [        nan,         nan]],

       [[ 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

[]

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:

$F(x, y) = \mathrm{ax\^{}2 + bxy + cy\^{}2 + dx + ey + f = 0`}$

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

[]

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_on_ellipse(45, coefficients)
array([ 1.,  2.])
```

colour.algebra.ELLIPSE_FITTING_METHODS

colour.algebra.**ELLIPSE_FITTING_METHODS = CaseInsensitiveMapping({'Halir 1998': ...})**

Supported ellipse fitting methods.

References

[]

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:

```
:math:`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.
- **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

[]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting(a)
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

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

colour.algebra.LineSegmentsIntersections_Specification

class colour.algebra.**LineSegmentsIntersections_Specification**(*xy*, *intersect*, *parallel*, *coincident*)

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__()

Methods

__init__()

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

coincident	Alias for field number 3
intersect	Alias for field number 1
parallel	Alias for field number 2
xy	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:`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

[]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting_Halir1998(a)
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

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

Random

colour.algebra

random_triplet_generator(size[, limits, ...])	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=RandomState(MT19937) at 0x7F0267854270*)

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 [].

Examples

```
>>> from pprint import pprint
>>> prng = np.random.RandomState(4)
>>> random_triplet_generator(10, random_state=prng)
...
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

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

[]

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

Common

colour.algebra

is_spow_enabled()	Returns whether *Colour* safe / symmetrical power function is enabled.
set_spow_enable(enable)	Sets *Colour* safe / symmetrical power function enabled state.
spow_enable(enable)	A context manager and decorator temporarily setting *Colour* safe / symmetrical power function enabled state.
spow(a, p)	Raises given array a to the power p as follows: $sign(a) * \|a\|^p$.
smoothstep_function(x[, a, b, clip])	Evaluates the *smoothstep* sigmoid-like function on array x.

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*)

Methods

`__init__`(enable)	

colour.algebra.spow

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

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

This definition avoids NaNs generation when array a is negative and the power p is fractional. This behaviour can be enabled or disabled with the `colour.algebra.set_spow_enable()` definition or with the `spow_enable()` context manager.

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)
-1.1095694...
>>> spow(0, 0)
0.0
```

colour.algebra.smoothstep_function

colour.algebra.**smoothstep_function**(*x*, *a=0*, *b=1*, *clip=False*)

Evaluates the *smoothstep* sigmoid-like function on array x.

Parameters

- **x** (numeric or array_like) – Array x.
- **a** (numeric, optional) – Low input domain limit, i.e. the left edge.
- **b** (numeric, optional) – High input domain limit, i.e. the right edge.
- **clip** (bool, optional) – Whether to scale, bias and clip input values to domain [0, 1].

Returns Array x after *smoothstep* sigmoid-like function evaluation.

Return type array_like

Examples

```
>>> x = np.linspace(-2, 2, 5)
>>> smoothstep_function(x, -2, 2, clip=True)
array([ 0.     ,  0.15625,  0.5    ,  0.84375,  1.     ])
```

Colour Appearance Models

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

ATD (1995)

colour

XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2[, sigma])	Computes the *ATD (1995)* colour vision model correlates.
CAM_Specification_ATD95(h, C, Q, A_1, T_1, ...)	Defines the *ATD (1995)* 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 *CAM_Specification_ATD95*

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_ATD95.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

[], []

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)
CAM_Specification_ATD95(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.CAM_Specification_ATD95

class colour.**CAM_Specification_ATD95**(*h*, *C*, *Q*, *A_1*, *T_1*, *D_1*, *A_2*, *T_2*, *D_2*)

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

[], []

Create new instance of CAM_Specification_ATD95(h, C, Q, A_1, T_1, D_1, A_2, T_2, D_2)

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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

XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b[, ...])	Computes the *CIECAM02* colour appearance model correlates from given *CIE XYZ* tristimulus values.
CIECAM02_to_XYZ(specification, XYZ_w, L_A, Y_b)	Converts from *CIECAM02* specification to *CIE XYZ* tristimulus values.
CAM_Specification_CIECAM02([J, C, h, s, Q, ...])	Defines the *CIECAM02* colour appearance model specification.
VIEWING_CONDITIONS_CIECAM02	Reference *CIECAM02* colour appearance model viewing conditions.

colour.XYZ_to_CIECAM02

colour.**XYZ_to_CIECAM02**(*XYZ*, *XYZ_w*, *L_A*, *Y_b*, *surround=InductionFactors_CIECAM02(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) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (InductionFactors_CIECAM02, 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 *CAM_Specification_CIECAM02*

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_CIECAM02.J	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.C	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.h	[0, 360]	[0, 1]
CAM_Specification_CIECAM02.s	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.Q	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.M	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.H	[0, 400]	[0, 1]

References

[], [], [], []

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 = VIEWING_CONDITIONS_CIECAM02['Average']
>>> XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
CAM_Specification_CIECAM02(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**(*specification*, *XYZ_w*, *L_A*, *Y_b*, *surround=InductionFactors_CIECAM02(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

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

This is the *inverse* implementation.

Parameters

- **specification** (CAM_Specification_CIECAM02) – *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) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (InductionFactors_CIECAM02, 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 CAM_Specification_CIECAM02 argument.

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

Notes

Domain	Scale - Reference	Scale - 1
CAM_Specification_CIECAM02.J	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.C	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.h	[0, 360]	[0, 1]
CAM_Specification_CIECAM02.s	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.Q	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.M	[0, 100]	[0, 1]
CAM_Specification_CIECAM02.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

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

- CAM_Specification_CIECAM02 can also be passed as a compatible argument to colour.utilities.as_namedtuple() definition.

References

[], [], [], []

Examples

```
>>> specification = CAM_Specification_CIECAM02(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)
array([ 19.01...,  20...  ,  21.78...])
```

colour.CAM_Specification_CIECAM02

class colour.**CAM_Specification_CIECAM02**(*J=None*, *C=None*, *h=None*, *s=None*, *Q=None*, *M=None*, *H=None*, *HC=None*)

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

[], [], [], []

Returns a new instance of the colour.CAM_Specification_CIECAM02 class.

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

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

colour.VIEWING_CONDITIONS_CIECAM02

colour.**VIEWING_CONDITIONS_CIECAM02 = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *CIECAM02* colour appearance model viewing conditions.

References

[], [], [], []

VIEWING_CONDITIONS_CIECAM02 [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Ancillary Objects

colour.appearance

InductionFactors_CIECAM02(F, c, N_c)	*CIECAM02* colour appearance model induction factors.

colour.appearance.InductionFactors_CIECAM02

class colour.appearance.**InductionFactors_CIECAM02**(*F*, *c*, *N_c*)

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

[], [], [], []

Create new instance of InductionFactors_CIECAM02(F, c, N_c)

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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(specification, XYZ_w, L_A, Y_b)	Converts from *CAM16* specification to *CIE XYZ* tristimulus values.
CAM_Specification_CAM16([J, C, h, s, Q, M, ...])	Defines the *CAM16* colour appearance model specification.
VIEWING_CONDITIONS_CAM16	Reference *CAM16* colour appearance model viewing conditions.

colour.XYZ_to_CAM16

colour.**XYZ_to_CAM16**(*XYZ*, *XYZ_w*, *L_A*, *Y_b*, *surround=InductionFactors_CIECAM02(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) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (InductionFactors_CAM16, 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 *CAM_Specification_CAM16*

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_CAM16.J	[0, 100]	[0, 1]
CAM_Specification_CAM16.C	[0, 100]	[0, 1]
CAM_Specification_CAM16.h	[0, 360]	[0, 1]
CAM_Specification_CAM16.s	[0, 100]	[0, 1]
CAM_Specification_CAM16.Q	[0, 100]	[0, 1]
CAM_Specification_CAM16.M	[0, 100]	[0, 1]
CAM_Specification_CAM16.H	[0, 400]	[0, 1]

References

[]

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 = VIEWING_CONDITIONS_CAM16['Average']
>>> XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround)
CAM_Specification_CAM16(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**(*specification*, *XYZ_w*, *L_A*, *Y_b*, *surround=InductionFactors_CIECAM02(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

Converts from *CAM16* specification to *CIE XYZ* tristimulus values.

This is the *inverse* implementation.

Parameters

- **specification** (CAM_Specification_CAM16) – *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) – Luminous factor of background Y_b such as $Y_b = 100xL_b/L_w$ where L_w is the luminance of the light source and L_b is the luminance of the background. For viewing images, Y_b can be the average Y value for the pixels in the entire image, or frequently, a Y value of 20, approximate an L^* of 50 is used.
- **surround** (InductionFactors_CAM16, 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 CAM_Specification_CAM16 argument.

Notes

Domain	Scale - Reference	Scale - 1
CAM_Specification_CAM16.J	[0, 100]	[0, 1]
CAM_Specification_CAM16.C	[0, 100]	[0, 1]
CAM_Specification_CAM16.h	[0, 360]	[0, 1]
CAM_Specification_CAM16.s	[0, 100]	[0, 1]
CAM_Specification_CAM16.Q	[0, 100]	[0, 1]
CAM_Specification_CAM16.M	[0, 100]	[0, 1]
CAM_Specification_CAM16.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

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

- `CAM_Specification_CAM16` can also be passed as a compatible argument to `colour.utilities.as_namedtuple()` definition.

References

[]

Examples

```
>>> specification = CAM_Specification_CAM16(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)
array([ 19.01...,  20...  ,  21.78...])
```

colour.CAM_Specification_CAM16

class colour.**CAM_Specification_CAM16**(*J=None*, *C=None*, *h=None*, *s=None*, *Q=None*, *M=None*, *H=None*, *HC=None*)

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

[]

Returns a new instance of the `colour.CAM_Specification_CAM16` class.

__init__()

Methods

`__init__`()	

`count`(value, /)	Return number of occurrences of value.
`index`(value[, start, stop])	Return first index of value.

Attributes

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

colour.VIEWING_CONDITIONS_CAM16

colour.**VIEWING_CONDITIONS_CAM16 = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *CAM16* colour appearance model viewing conditions.

References

[]

VIEWING_CONDITIONS_CAM16 [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Ancillary Objects

`colour.appearance`

`InductionFactors_CAM16`(F, c, N_c)	*CAM16* colour appearance model induction factors.

colour.appearance.InductionFactors_CAM16

class colour.appearance.**InductionFactors_CAM16**(*F*, *c*, *N_c*)

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

[]

Create new instance of InductionFactors_CAM16(F, c, N_c)

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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 *Hunt* colour appearance model correlates.
CAM_Specification_Hunt(J, C, h, s, Q, M, H, HC)	Defines the *Hunt* colour appearance model specification.
VIEWING_CONDITIONS_HUNT	Reference *Hunt* colour appearance model viewing conditions.

colour.XYZ_to_Hunt

colour.**XYZ_to_Hunt**(*XYZ*, *XYZ_w*, *XYZ_b*, *L_A*, *surround=InductionFactors_Hunt(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** (InductionFactors_Hunt, 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 *CAM_Specification_Hunt*

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
CAM_Specification_Hunt.h	[0, 360]	[0, 1]

References

[], []

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 = VIEWING_CONDITIONS_HUNT['Normal Scenes']
>>> CCT_w = 6504
>>> XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround, CCT_w=CCT_w)
...
CAM_Specification_Hunt(J=30.0462678..., C=0.1210508..., h=269.2737594..., s=0.
↪0199093..., Q=22.2097654..., M=0.1238964..., H=None, HC=None)
```

colour.CAM_Specification_Hunt

class colour.**CAM_Specification_Hunt**(*J*, *C*, *h*, *s*, *Q*, *M*, *H*, *HC*)

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

[], []

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

__init__()

Methods

`__init__`()	

`count`(value, /)	Return number of occurrences of value.
`index`(value[, start, stop])	Return first index of value.

Attributes

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

colour.VIEWING_CONDITIONS_HUNT

colour.**VIEWING_CONDITIONS_HUNT = CaseInsensitiveMapping({'Small Areas, Uniform Background & Surrounds': ..., 'Normal Scenes': ..., 'Television & CRT, Dim Surrounds': ..., 'Large Transparencies On Light Boxes': ..., 'Projected Transparencies, Dark Surrounds': ..., 'small_uniform': ..., 'normal': ..., 'tv_dim': ..., 'light_boxes': ..., 'projected_dark': ...})**

Reference *Hunt* colour appearance model viewing conditions.

References

[], []

VIEWING_CONDITIONS_HUNT [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

XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L[, surround])	Computes the *:math:`LLAB(l:c)`* colour appearance model correlates.
CAM_Specification_LLAB(J, C, h, s, M, HC, a, b)	Defines the *:math:`LLAB(l:c)`* colour appearance model specification.
VIEWING_CONDITIONS_LLAB	Reference $LLAB(l:c)$ colour appearance model viewing conditions.

colour.XYZ_to_LLAB

colour.**XYZ_to_LLAB**(*XYZ*, *XYZ_0*, *Y_b*, *L*, *surround=InductionFactors_LLAB(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** (InductionFactors_LLAB, optional) – Surround viewing conditions induction factors.

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

Return type *CAM_Specification_LLAB*

Notes

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

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

References

[], [], []

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 = VIEWING_CONDITIONS_LLAB['ref_average_4_minus']
>>> XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L, surround)
CAM_Specification_LLAB(J=37.3668650..., C=0.0089496..., h=270..., s=0.0002395...,␣
↪M=0.0190185..., HC=None, a=..., b=-0.0190185...)
```

colour.CAM_Specification_LLAB

class colour.**CAM_Specification_LLAB**(*J*, *C*, *h*, *s*, *M*, *HC*, *a*, *b*)

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

[], [], []

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

__init__()

Methods

__init__()	
count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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.VIEWING_CONDITIONS_LLAB

colour.**VIEWING_CONDITIONS_LLAB = 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': ..., 'ref_average_4_plus': ..., 'ref_average_4_minus': ..., 'tv_dim': ..., 'sheet_dim': ..., 'projected_dark': ...})**

Reference $LLAB(l : c)$ colour appearance model viewing conditions.

References

[], [], []

VIEWING_CONDITIONS_LLAB [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`

InductionFactors_LLAB(D, F_S, F_L, F_C)	*:math:`LLAB(l:c)`* colour appearance model induction factors.

colour.appearance.InductionFactors_LLAB

class colour.appearance.**InductionFactors_LLAB**(*D*, *F_S*, *F_L*, *F_C*)

: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

[], [], []

Create new instance of InductionFactors_LLAB(D, F_S, F_L, F_C)

__init__()

Methods

__init__()

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

D	Alias for field number 0
F_C	Alias for field number 3
F_L	Alias for field number 2
F_S	Alias for field number 1

Nayatani (1995)

colour

XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or)	Computes the *Nayatani (1995)* colour appearance model correlates.
CAM_Specification_Nayatani95(L_star_P, C, h, ...)	Defines the *Nayatani (1995)* 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 *CAM_Specification_Nayatani95*

Notes

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

Range	Scale - Reference	Scale - 1
CAM_Specification_Nayatani95.h	[0, 360]	[0, 1]

References

[], []

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)
CAM_Specification_Nayatani95(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.CAM_Specification_Nayatani95

class colour.**CAM_Specification_Nayatani95**(*L_star_P*, *C*, *h*, *s*, *Q*, *M*, *H*, *HC*, *L_star_N*)

Defines the *Nayatani (1995)* 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

- **L_star_P** (numeric or array_like) – Correlate of *achromatic Lightness* $L_p^\star$.
- **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* $L_n^\star$.

Notes

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

References

[], []

Create new instance of CAM_Specification_Nayatani95(L_star_P, C, h, s, Q, M, H, HC, L_star_N)

__init__()

Methods

`__init__`()	

`count`(value, /)	Return number of occurrences of value.
`index`(value[, start, stop])	Return first index of value.

Attributes

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

RLAB

`colour`

`XYZ_to_RLAB`(XYZ, XYZ_n, Y_n[, sigma, D])	Computes the *RLAB* model color appearance correlates.
`CAM_Specification_RLAB`(J, C, h, s, HC, a, b)	Defines the *RLAB* colour appearance model specification.
`VIEWING_CONDITIONS_RLAB`	Reference *RLAB* 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.VIEWING_CONDITIONS_RLAB` 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 *CAM_Specification_RLAB*

Notes

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

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

References

[], []

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_n = np.array([109.85, 100, 35.58])
>>> Y_n = 31.83
>>> sigma = VIEWING_CONDITIONS_RLAB['Average']
>>> D = D_FACTOR_RLAB['Hard Copy Images']
>>> XYZ_to_RLAB(XYZ, XYZ_n, Y_n, sigma, D)
CAM_Specification_RLAB(J=49.8347069..., C=54.8700585..., h=286.4860208..., s=1.
↪1010410..., HC=None, a=15.5711021..., b=-52.6142956...)
```

colour.CAM_Specification_RLAB

class colour.**CAM_Specification_RLAB**(*J*, *C*, *h*, *s*, *HC*, *a*, *b*)

Defines the *RLAB* 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^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

[], []

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

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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.VIEWING_CONDITIONS_RLAB

colour.**VIEWING_CONDITIONS_RLAB = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *RLAB* colour appearance model viewing conditions.

References

[], []

VIEWING_CONDITIONS_RLAB [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Ancillary Objects

colour.appearance

D_FACTOR_RLAB	*RLAB* colour appearance model *Discounting-the-Illuminant* factor values.

colour.appearance.D_FACTOR_RLAB

colour.appearance.**D_FACTOR_RLAB = CaseInsensitiveMapping({'Hard Copy Images': ..., 'Soft Copy Images': ..., 'Projected Transparencies, Dark Room': ..., 'hard_cp_img': ..., 'soft_cp_img': ..., 'projected_dark': ...})**

RLAB colour appearance model *Discounting-the-Illuminant* factor values.

References

[], []

D_FACTOR_RLAB [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'

Biochemistry

- *Michaelis–Menten Kinetics*

Michaelis–Menten Kinetics

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.
substrate_concentration_MichealisMenten(v, ...)	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

[]

Examples

```
>>> reaction_rate_MichealisMenten(0.5, 2.5, 0.8)
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

[]

Examples

```
>>> substrate_concentration_MichealisMenten(0.961538461538461, 2.5, 0.8)
...
0.4999999...
```

Colour Vision Deficiency

- *Machado, Oliveira and Fernandes (2009)*

Machado, Oliveira and Fernandes (2009)

colour

msds_cmfs_anomalous_trichromacy_Machado2009(...)	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.
matrix_anomalous_trichromacy_Machado2009(...)	Computes the *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.
matrix_cvd_Machado2009(deficiency, severity)	Computes *Machado et al. (2009) CVD* matrix for given deficiency and severity using the pre-computed matrices dataset.

colour.msds_cmfs_anomalous_trichromacy_Machado2009

colour.**msds_cmfs_anomalous_trichromacy_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

[], [], [], []

Examples

```
>>> from colour.colorimetry import MSDS_CMFS_LMS
>>> cmfs = MSDS_CMFS_LMS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> cmfs[450]
array([ 0.0498639,  0.0870524,  0.955393 ])
>>> msds_cmfs_anomalous_trichromacy_Machado2009(
...     cmfs, np.array([15, 0, 0]))[450]
array([ 0.0891288...,  0.0870524 ,  0.955393  ])
```

colour.matrix_anomalous_trichromacy_Machado2009

colour.**matrix_anomalous_trichromacy_Machado2009**(*cmfs*, *primaries*, *d_LMS*)

Computes the *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

[], [], [], []

Examples

```
>>> from colour.characterisation import MSDS_DISPLAY_PRIMARIES
>>> from colour.colorimetry import MSDS_CMFS_LMS
>>> cmfs = MSDS_CMFS_LMS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> d_LMS = np.array([15, 0, 0])
>>> primaries = MSDS_DISPLAY_PRIMARIES['Apple Studio Display']
>>> matrix_anomalous_trichromacy_Machado2009(cmfs, primaries, d_LMS)
...
array([[-0.2777465...,  2.6515008..., -1.3737543...],
       [ 0.2718936...,  0.2004786...,  0.5276276...],
       [ 0.0064404...,  0.2592157...,  0.7343437...]])
```

colour.matrix_cvd_Machado2009

colour.**matrix_cvd_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

[], [], [], []

Examples

```
>>> matrix_cvd_Machado2009('Protanomaly', 0.15)
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

ACES Spectral Conversion

colour

sd_to_aces_relative_exposure_values(sd[, ...])	Converts given spectral distribution to *ACES2065-1* 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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *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

[], [], [], []

Examples

```
>>> from colour import SDS_COLOURCHECKERS
>>> sd = SDS_COLOURCHECKERS['ColorChecker N Ohta']['dark skin']
>>> sd_to_aces_relative_exposure_values(sd)
array([ 0.1171814...,  0.0866360...,  0.0589726...])
>>> sd_to_aces_relative_exposure_values(sd,
...     apply_chromatic_adaptation=True)
array([ 0.1180779...,  0.0869031...,  0.0589125...])
```

Ancillary Objects

colour.characterisation

MSDS_ACES_RICD	Implements support for a camera *RGB* sensitivities.

colour.characterisation.MSDS_ACES_RICD

colour.characterisation.**MSDS_ACES_RICD = RGB_CameraSensitivities(name='ACES RICD', ...)**

Implements support for a camera *RGB* sensitivities.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.

- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.characterisation.RGB_CameraSensitivities.labels attribute value.

ACES Input Transform Computation

colour

matrix_idt(sensitivities, illuminant[, ...])	Computes an *Input Device Transform* (IDT) matrix for given camera *RGB* spectral sensitivities, illuminant, training data, standard observer colour matching functions and optimization settings according to *RAW to ACES* v1 and *P-2013-001* procedures.

colour.matrix_idt

colour.**matrix_idt**(*sensitivities*, *illuminant*, *training_data=None*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *optimisation_factory=<function optimisation_factory_rawtoaces_v1>*, *optimisation_kwargs=None*)

Computes an *Input Device Transform* (IDT) matrix for given camera *RGB* spectral sensitivities, illuminant, training data, standard observer colour matching functions and optimization settings according to *RAW to ACES* v1 and *P-2013-001* procedures.

Parameters

- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **training_data** (MultiSpectralDistributions, optional) – Training data multi-spectral distributions, defaults to using the *RAW to ACES* v1 190 patches.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **optimisation_factory** (callable, optional) – Callable producing the objective function and the *CIE XYZ* to optimisation colour model function.
- **optimisation_kwargs** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns *Input Device Transform* (IDT) matrix.

Return type ndarray

References

[], []

Examples

Computing the *Input Device Transform* (IDT) matrix for a *CANON EOS 5DMark II* and *CIE Illuminant D Series D55* using the method given in *RAW to ACES* v1:

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminant = SDS_ILLUMINANTS['D55']
>>> np.around(
...     matrix_idt(sensitivities, illuminant), 3)
array([[ 0.85 , -0.016,  0.151],
       [ 0.051,  1.126, -0.185],
       [ 0.02 , -0.194,  1.162]])
```

The *RAW to ACES* v1 matrix for the same camera and optimized by Ceres Solver is as follows:

0.864994 -0.026302 0.161308 0.056527 1.122997 -0.179524 0.023683 -0.202547 1.178864

```
>>> np.around(matrix_idt(
...     sensitivities, illuminant,
...     optimisation_factory=optimisation_factory_JzAzBz), 3)
array([[ 0.848, -0.016,  0.158],
       [ 0.053,  1.114, -0.175],
       [ 0.023, -0.225,  1.196]])
```

Ancillary Objects

`colour.characterisation`

read_training_data_rawtoaces_v1()	Reads the *RAW to ACES* v1 190 patches.
generate_illuminants_rawtoaces_v1()	Generates a series of illuminants according to *RAW to ACES* v1:
white_balance_multipliers(sensitivities, ...)	Computes the *RGB* white balance multipliers for given camera *RGB* spectral sensitivities and illuminant.
normalise_illuminant(illuminant, sensitivities)	Normalises given illuminant with given camera *RGB* spectral sensitivities.
training_data_sds_to_RGB(training_data, ...)	Converts given training data to *RGB* tristimulus values using given illuminant and given camera *RGB* spectral sensitivities.
training_data_sds_to_XYZ(training_data, ...)	Converts given training data to *CIE XYZ* tristimulus values using given illuminant and given standard observer colour matching functions.
best_illuminant(RGB_w, sensitivities, ...)	Select the best illuminant for given *RGB* white balance multipliers, and sensitivities in given series of illuminants.
optimisation_factory_rawtoaces_v1()	Factory that returns the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function according to *RAW to ACES* v1.
optimisation_factory_JzAzBz()	Factory that returns the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the $J_zA_zB_z$ colourspace.

colour.characterisation.read_training_data_rawtoaces_v1

colour.characterisation.**read_training_data_rawtoaces_v1**()

Reads the *RAW to ACES* v1 190 patches.

Returns *RAW to ACES* v1 190 patches.

Return type *MultiSpectralDistributions*

References

[]

Examples

```
>>> len(read_training_data_rawtoaces_v1().labels)
190
```

colour.characterisation.generate_illuminants_rawtoaces_v1

colour.characterisation.**generate_illuminants_rawtoaces_v1**()

Generates a series of illuminants according to *RAW to ACES* v1:

- *CIE Illuminant D Series* in range [4000, 25000] kelvin degrees.
- *Blackbodies* in range [1000, 3500] kelvin degrees.
- A.M.P.A.S. variant of *ISO 7589 Studio Tungsten*.

Returns Series of illuminants.

Return type *CaseInsensitiveMapping*

Notes

- This definition introduces a few differences compared to *RAW to ACES* v1: *CIE Illuminant D Series* are computed in range [4002.15, 7003.77] kelvin degrees and the C_2 change is not used in *RAW to ACES* v1.

References

[]

Examples

```
>>> # Doctests skip for Python 2.x compatibility.
>>> list(sorted(generate_illuminants_rawtoaces_v1().keys()))
...
['1000K Blackbody', '1500K Blackbody', '2000K Blackbody', '2500K Blackbody', '3000K␣
↪Blackbody', '3500K Blackbody', 'D100', 'D105', 'D110', 'D115', 'D120', 'D125',
↪'D130', 'D135', 'D140', 'D145', 'D150', 'D155', 'D160', 'D165', 'D170', 'D175',
↪'D180', 'D185', 'D190', 'D195', 'D200', 'D205', 'D210', 'D215', 'D220', 'D225',
↪'D230', 'D235', 'D240', 'D245', 'D250', 'D40', 'D45', 'D50', 'D55', 'D60', 'D65',
↪'D70', 'D75', 'D80', 'D85', 'D90', 'D95', 'iso7589']
```

colour.characterisation.white_balance_multipliers

colour.characterisation.**white_balance_multipliers**(*sensitivities*, *illuminant*)

Computes the *RGB* white balance multipliers for given camera *RGB* spectral sensitivities and illuminant.

Parameters

- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.

Returns *RGB* white balance multipliers.

Return type ndarray

References

[]

Examples

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminant = SDS_ILLUMINANTS['D55']
>>> white_balance_multipliers(sensitivities, illuminant)
...
array([ 2.3414154...,  1.        ,  1.5163375...])
```

colour.characterisation.normalise_illuminant

colour.characterisation.**normalise_illuminant**(*illuminant*, *sensitivities*)

Normalises given illuminant with given camera *RGB* spectral sensitivities.

The multiplicative inverse scaling factor k is computed by multiplying the illuminant by the sensitivies channel with the maximum value.

Parameters

- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.

Returns Normalised illuminant.

Return type *SpectralDistribution*

Examples

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminant = SDS_ILLUMINANTS['D55']
>>> np.sum(illuminant.values)
7276.1490000...
>>> np.sum(normalise_illuminant(illuminant, sensitivities).values)
...
3.4390373...
```

colour.characterisation.training_data_sds_to_RGB

colour.characterisation.**training_data_sds_to_RGB**(*training_data*, *sensitivities*, *illuminant*)

Converts given training data to *RGB* tristimulus values using given illuminant and given camera *RGB* spectral sensitivities.

Parameters

- **training_data** (MultiSpectralDistributions) – Training data multi-spectral distributions.
- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.

Returns Training data *RGB* tristimulus values.

Return type ndarray

Examples

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminant = normalise_illuminant(
...     SDS_ILLUMINANTS['D55'], sensitivities)
>>> training_data = read_training_data_rawtoaces_v1()
>>> training_data_sds_to_RGB(training_data, sensitivities, illuminant)[:5]
...
array([[ 0.0207582...,  0.0196857...,  0.0213935...],
       [ 0.0895775...,  0.0891922...,  0.0891091...],
       [ 0.7810230...,  0.7801938...,  0.7764302...],
       [ 0.1995   ...,  0.1995   ...,  0.1995   ...],
       [ 0.5898478...,  0.5904015...,  0.5851076...]])
```

colour.characterisation.training_data_sds_to_XYZ

colour.characterisation.**training_data_sds_to_XYZ**(*training_data*, *cmfs*, *illuminant*)

Converts given training data to *CIE XYZ* tristimulus values using given illuminant and given standard observer colour matching functions.

Parameters

- **training_data** (MultiSpectralDistributions) – Training data multi-spectral distributions.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.

Returns Training data *CIE XYZ* tristimulus values.

Return type ndarray

Examples

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminant = normalise_illuminant(
...     SDS_ILLUMINANTS['D55'], sensitivities)
>>> training_data = read_training_data_rawtoaces_v1()
>>> training_data_sds_to_XYZ(training_data, cmfs, illuminant)[:5]
...
array([[ 0.0174353...,  0.0179504...,  0.0196109...],
       [ 0.0855607...,  0.0895735...,  0.0901703...],
       [ 0.7455880...,  0.7817549...,  0.7834356...],
       [ 0.1900528...,  0.1995   ...,  0.2012606...],
       [ 0.5626319...,  0.5914544...,  0.5894500...]])
```

colour.characterisation.best_illuminant

colour.characterisation.**best_illuminant**(*RGB_w*, *sensitivities*, *illuminants*)

Select the best illuminant for given *RGB* white balance multipliers, and sensitivities in given series of illuminants.

Parameters

- **RGB_w** (array_like) – *RGB* white balance multipliers.
- **sensitivities** (RGB_CameraSensitivities) – Camera *RGB* spectral sensitivities.
- **illuminants** (SpectralDistribution) – Illuminant spectral distributions to choose the best illuminant from.

Returns Best illuminant.

Return type *SpectralDistribution*

Examples

```
>>> path = os.path.join(
...     RESOURCES_DIRECTORY_RAWTOACES,
...     'CANON_EOS_5DMark_II_RGB_Sensitivities.csv')
>>> sensitivities = sds_and_msds_to_msds(
...     read_sds_from_csv_file(path).values())
>>> illuminants = generate_illuminants_rawtoaces_v1()
>>> RGB_w = white_balance_multipliers(
...     sensitivities, SDS_ILLUMINANTS['FL2'])
>>> # Doctests skip for Python 2.x compatibility.
>>> best_illuminant(RGB_w, sensitivities, illuminants).name
...
'D40'
```

colour.characterisation.optimisation_factory_rawtoaces_v1

colour.characterisation.**optimisation_factory_rawtoaces_v1**()

Factory that returns the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function according to *RAW to ACES* v1.

The objective function returns the euclidean distance between the training data *RGB* tristimulus values and the training data *CIE XYZ* tristimulus values** in *CIE L*a*b** colourspace.

Returns Objective function and *CIE XYZ* colourspace to *CIE L*a*b** colourspace function.

Return type tuple

Examples

```
>>> # Doctests skip for Python 2.x compatibility.
>>> optimisation_factory_rawtoaces_v1()
(<function optimisation_factory_rawtoaces_v1.<locals>.objective_function at 0x...>,
↪<function optimisation_factory_rawtoaces_v1.<locals>.XYZ_to_optimization_colour_
↪model at 0x...>)
```

colour.characterisation.optimisation_factory_JzAzBz

colour.characterisation.**optimisation_factory_JzAzBz**()

Factory that returns the objective function and *CIE XYZ* colourspace to optimisation colourspace/colour model function based on the $J_zA_zB_z$ colourspace.

The objective function returns the euclidean distance between the training data *RGB* tristimulus values and the training data *CIE XYZ* tristimulus values** in the $J_zA_zB_z$ colourspace.

Returns Objective function and *CIE XYZ* colourspace to $J_zA_zB_z$ colourspace function.

Return type tuple

Examples

```
>>> # Doctests skip for Python 2.x compatibility.
>>> optimisation_factory_JzAzBz()
(<function optimisation_factory_JzAzBz.<locals>.objective_function at 0x...>,
↪<function optimisation_factory_JzAzBz.<locals>.XYZ_to_optimization_colour_model at␣
↪0x...>)
```

Colour Fitting

colour

POLYNOMIAL_EXPANSION_METHODS	Supported polynomial expansion methods.
polynomial_expansion(a[, method])	Performs polynomial expansion of given a array.
MATRIX_COLOUR_CORRECTION_METHODS	Supported colour correction matrix methods.
matrix_colour_correction(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 *RGB* 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

[], [], [], []

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.
- **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.matrix_augmented_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

[], [], [], []

Examples

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

colour.MATRIX_COLOUR_CORRECTION_METHODS

colour.**MATRIX_COLOUR_CORRECTION_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ..., 'Vandermonde': ...})**

Supported colour correction matrix methods.

References

[], [], [], []

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

colour.matrix_colour_correction

colour.**matrix_colour_correction**(*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, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **method** (unicode, optional) – **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**, Computation method.
- **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.matrix_augmented_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, (n, 3)

References

[], [], [], []

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],
...      [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]]
... )
>>> matrix_colour_correction(M_T, M_R)
array([[ 0.6982266...,  0.0307162...,  0.1621042...],
       [ 0.0689349...,  0.6757961...,  0.1643038...],
```

(continues on next page)

(continued from previous page)

```
       [-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

[], [], [], []

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, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **method** (unicode, optional) – **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**, Computation method.
- **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.matrix_augmented_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

[], [], [], []

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],
...      [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],
```

(continues on next page)

(continued from previous page)

```
...       [0.10283975, 0.10424680, 0.10384975],
...       [0.04742204, 0.04772203, 0.04914226]]
... )
>>> colour_correction(RGB, M_T, M_R)
array([ 0.1334872...,  0.0843921...,  0.0599014...])
```

Ancillary Objects

colour.characterisation

matrix_augmented_Cheung2004(RGB[, terms])	Performs polynomial expansion of given *RGB* colourspace array using *Cheung et al. (2004)* method.
polynomial_expansion_Finlayson2015(RGB[, ...])	Performs polynomial expansion of given *RGB* colourspace array using *Finlayson et al. (2015)* method.
polynomial_expansion_Vandermonde(a[, degree])	Performs polynomial expansion of given a array using *Vandermonde* method.
matrix_colour_correction_Cheung2004(M_T, M_R)	Computes a colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.
matrix_colour_correction_Finlayson2015(M_T, M_R)	Computes a colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.
matrix_colour_correction_Vandermonde(M_T, M_R)	Computes a colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.
colour_correction_Cheung2004(RGB, M_T, M_R)	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.
colour_correction_Finlayson2015(RGB, M_T, M_R)	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.
colour_correction_Vandermonde(RGB, M_T, M_R)	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.

colour.characterisation.matrix_augmented_Cheung2004

colour.characterisation.**matrix_augmented_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 [] and [].

References

[], []

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> matrix_augmented_Cheung2004(RGB, terms=5)
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

[]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Finlayson2015(RGB, degree=2)
array([ 0.1722481...,  0.0917066...,  0.0641693...,  0.1256832...,  0.0767121...,
        0.1051335...])
```

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

[]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Vandermonde(RGB)
array([ 0.1722481 ,  0.0917066 ,  0.06416938,  1.        ])
```

colour.characterisation.matrix_colour_correction_Cheung2004

colour.characterisation.**matrix_colour_correction_Cheung2004**(*M_T*, *M_R*, *terms=3*)

Computes a colour correction matrix 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

[], []

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
>>> matrix_colour_correction_Cheung2004(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.matrix_colour_correction_Finlayson2015

colour.characterisation.**matrix_colour_correction_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, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – 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, (n, 3)

References

[]

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
>>> matrix_colour_correction_Finlayson2015(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.matrix_colour_correction_Vandermonde

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

Returns Colour correction matrix.

Return type ndarray, (n, 3)

References

[]

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
>>> matrix_colour_correction_Vandermonde(M_T, M_R)
array([[ 1.0300256...,  0.1141770..., -0.2621816...,  0.0418022...],
       [ 0.0670209...,  1.0221494..., -0.1166108...,  0.0128250...],
       [ 0.0744612..., -0.1872819...,  1.1278078..., -0.0318085...]])
```

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, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – 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

[], []

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)
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, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – 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

[]

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)
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, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – 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

[]

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)
array([ 0.2128689...,  0.1106242...,  0.036213 ...])
```

Colour Rendition Charts

Dataset

colour

CCS_COLOURCHECKERS	Chromaticity coordinates of the colour checkers.
SDS_COLOURCHECKERS	Spectral distributions of the colour checkers.

colour.CCS_COLOURCHECKERS

colour.**CCS_COLOURCHECKERS = CaseInsensitiveMapping({'ColorChecker 1976': ..., 'ColorChecker 2005': ..., 'BabelColor Average': ..., 'ColorChecker24 - Before November 2014': ..., 'ColorChecker24 - After November 2014': ..., 'babel_average': ..., 'cc2005': ..., 'ccb2014': ..., 'cca2014': ...})**

Chromaticity coordinates of the colour checkers.

References

[], [], []

CCS_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.SDS_COLOURCHECKERS

colour.**SDS_COLOURCHECKERS = CaseInsensitiveMapping({'BabelColor Average': ..., 'ColorChecker N Ohta': ..., 'babel_average': ..., 'cc_ohta': ..., 'ISO 17321-1': ...})**

Spectral distributions of the colour checkers.

References

[], [], [], [], []

SDS_COLOURCHECKERS [CaseInsensitiveMapping] **{'BabelColor Average', 'ColorChecker N Ohta', 'ISO 17321-1'}**

Notes

- Data from [] and [] has been verified to be the same.

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc_ohta': 'ColorChecker N Ohta'
- 'ISO 17321-1': 'ColorChecker N Ohta'

Ancillary Objects

colour.characterisation

ColourChecker(name, data, illuminant)	*Colour Checker* data.

colour.characterisation.ColourChecker

class colour.characterisation.**ColourChecker**(*name*, *data*, *illuminant*)

Colour Checker data.

Parameters

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

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

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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

RGB_CameraSensitivities([data, domain, labels])	Implements support for a camera *RGB* sensitivities.

colour.characterisation.RGB_CameraSensitivities

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

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for a camera *RGB* sensitivities.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.characterisation.RGB_CameraSensitivities.labels attribute value.

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Dataset

colour

MSDS_CAMERA_SENSITIVITIES	Multi-spectral distributions of camera sensitivities.

colour.MSDS_CAMERA_SENSITIVITIES

colour.**MSDS_CAMERA_SENSITIVITIES = CaseInsensitiveMapping({'Nikon 5100 (NPL)': ..., 'Sigma SDMerill (NPL)': ...})**

Multi-spectral distributions of camera sensitivities.

References

[]

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

Displays

colour.characterisation

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

colour.characterisation.RGB_DisplayPrimaries

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

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for a *RGB* display (such as a *CRT* or *LCD*) primaries multi-spectral distributions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.characterisation.RGB_DisplayPrimaries.labels attribute value.

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Dataset

colour

MSDS_DISPLAY_PRIMARIES	Primaries multi-spectral distributions of displays.

colour.MSDS_DISPLAY_PRIMARIES

colour.**MSDS_DISPLAY_PRIMARIES = CaseInsensitiveMapping({'Typical CRT Brainard 1997': ..., 'Apple Studio Display': ...})**

Primaries multi-spectral distributions of displays.

References

[], []

MSDS_DISPLAY_PRIMARIES [CaseInsensitiveMapping] **{Apple Studio Display, Typical CRT Brainard 1997}**

Filters

Dataset

colour

SDS_FILTERS	Spectral distributions of filters.

colour.SDS_FILTERS

colour.**SDS_FILTERS = CaseInsensitiveMapping({'ISO 7589 Diffuser': ...})**

Spectral distributions of filters.

References

[]

SDS_FILTERS : CaseInsensitiveMapping

Lenses

Dataset

colour

SDS_LENSES	Spectral distributions of lenses.

colour.SDS_LENSES

colour.**SDS_LENSES = CaseInsensitiveMapping({'ISO Standard Lens': ...})**

Spectral distributions of lenses.

References

[]

SDS_LENSES : CaseInsensitiveMapping

Colorimetry

Spectral Data Structure

`colour`

`SpectralShape`([start, end, interval])	Defines the base object for spectral distribution shape.
`SpectralDistribution`([data, domain])	Defines the spectral distribution: the base object for spectral computations.
`MultiSpectralDistributions`([data, domain, ...])	Defines the multi-spectral distributions: the base object for multi spectral computations.

colour.SpectralShape

class colour.**SpectralShape**(*start=None*, *end=None*, *interval=None*)

Bases: `object`

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.

Attributes

- start
- end
- interval
- boundaries

Methods

- __init__()
- __str__()
- __repr__()
- __hash__()
- __iter__()
- __contains__()
- __len__()
- __eq__()
- __ne__()
- range()

Examples

```
>>> SpectralShape(360, 830, 1)
SpectralShape(360, 830, 1)
```

__init__(*start=None*, *end=None*, *interval=None*)

property start

Getter and setter property for the spectral shape start.

Parameters **value** (numeric) – Value to set the spectral shape start with.

Returns Spectral shape start.

Return type numeric

property end

Getter and setter property for the spectral shape end.

Parameters **value** (numeric) – Value to set the spectral shape end with.

Returns Spectral shape end.

Return type numeric

property interval

Getter and setter property for the spectral shape interval.

Parameters **value** (numeric) – Value to set the spectral shape interval with.

Returns Spectral shape interval.

Return type numeric

property boundaries

Getter and setter property for the spectral shape boundaries.

Parameters **value** (array_like) – Value to set the spectral shape boundaries with.

Returns Spectral shape boundaries.

Return type tuple

__str__()

Returns a formatted string representation of the spectral shape.

Returns Formatted string representation.

Return type unicode

__repr__()

Returns an evaluable string representation of the spectral shape.

Returns Evaluable string representation.

Return type unicode

__hash__()

Returns the spectral shape hash.

Returns Object hash.

Return type int

__iter__()

Returns a generator for the spectral shape data.

Returns Spectral shape data generator.

Return type generator

Examples

```
>>> shape = SpectralShape(0, 10, 1)
>>> for wavelength in shape:
...     print(wavelength)
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
```

__contains__(*wavelength*)

Returns if the spectral shape contains given wavelength λ.

Parameters **wavelength** (numeric or array_like) – Wavelength λ.

Returns Is wavelength λ contained in the spectral shape.

Return type bool

Examples

```
>>> 0.5 in SpectralShape(0, 10, 0.1)
True
>>> 0.6 in SpectralShape(0, 10, 0.1)
True
>>> 0.51 in SpectralShape(0, 10, 0.1)
False
>>> np.array([0.5, 0.6]) in SpectralShape(0, 10, 0.1)
True
>>> np.array([0.51, 0.6]) in SpectralShape(0, 10, 0.1)
False
```

__len__()

Returns the spectral shape wavelength λ_n count.

Returns Spectral shape wavelength λ_n count.

Return type int

Examples

```
>>> len(SpectralShape(0, 10, 0.1))
101
```

__eq__(*shape*)

Returns the spectral shape equality with given other spectral shape.

Parameters **shape** (SpectralShape) – Spectral shape to compare for equality.

Returns Spectral shape equality.

Return type bool

Examples

```
>>> SpectralShape(0, 10, 0.1) == SpectralShape(0, 10, 0.1)
True
>>> SpectralShape(0, 10, 0.1) == SpectralShape(0, 10, 1)
False
```

__ne__(*shape*)

Returns the spectral shape inequality with given other spectral shape.

Parameters **shape** (SpectralShape) – Spectral shape to compare for inequality.

Returns Spectral shape inequality.

Return type bool

Examples

```
>>> SpectralShape(0, 10, 0.1) != SpectralShape(0, 10, 0.1)
False
>>> SpectralShape(0, 10, 0.1) != SpectralShape(0, 10, 1)
True
```

range(*dtype=None*)

Returns an iterable range for the spectral shape.

Parameters **dtype** (type) – Data type used to generate the range.

Returns Iterable range for the spectral distribution shape

Return type ndarray

Raises RuntimeError – If one of spectral shape *start*, *end* or *interval* attributes is not defined.

Examples

```
>>> SpectralShape(0, 10, 0.1).range()
array([  0. ,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,
         0.9,   1. ,   1.1,   1.2,   1.3,   1.4,   1.5,   1.6,   1.7,
         1.8,   1.9,   2. ,   2.1,   2.2,   2.3,   2.4,   2.5,   2.6,
         2.7,   2.8,   2.9,   3. ,   3.1,   3.2,   3.3,   3.4,   3.5,
         3.6,   3.7,   3.8,   3.9,   4. ,   4.1,   4.2,   4.3,   4.4,
         4.5,   4.6,   4.7,   4.8,   4.9,   5. ,   5.1,   5.2,   5.3,
         5.4,   5.5,   5.6,   5.7,   5.8,   5.9,   6. ,   6.1,   6.2,
         6.3,   6.4,   6.5,   6.6,   6.7,   6.8,   6.9,   7. ,   7.1,
         7.2,   7.3,   7.4,   7.5,   7.6,   7.7,   7.8,   7.9,   8. ,
         8.1,   8.2,   8.3,   8.4,   8.5,   8.6,   8.7,   8.8,   8.9,
         9. ,   9.1,   9.2,   9.3,   9.4,   9.5,   9.6,   9.7,   9.8,
         9.9,  10. ])
```

__weakref__

list of weak references to the object (if defined)

colour.SpectralDistribution

class colour.**SpectralDistribution**(*data=None*, *domain=None*, ***kwargs*)

Bases: colour.continuous.signal.Signal

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.

Important: Specific documentation about getting, setting, indexing and slicing the spectral power distribution values is available in the *Spectral Representation and Continuous Signal* section.

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.
- **name** (unicode, optional) – Spectral distribution name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (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.

Attributes

- strict_name
- wavelengths
- values
- shape

Methods

- __init__()
- interpolate()
- extrapolate()
- align()
- trim()
- normalise()

References

[], [], []

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
```

(continues on next page)

(continued from previous page)

```
... }
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
SpectralDistribution([[ 500.    ,    0.0651],
                      [ 520.    ,    0.0705],
                      [ 540.    ,    0.0772],
                      [ 560.    ,    0.087 ],
                      [ 580.    ,    0.1128],
                      [ 600.    ,    0.136 ]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
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_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
```

Instantiation with a *Pandas Series*:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import Series
...     print(SpectralDistribution(Series(data)))
[[  5.0000000...e+02   6.5100000...e-02]
 [  5.2000000...e+02   7.0500000...e-02]
 [  5.4000000...e+02   7.7200000...e-02]
 [  5.6000000...e+02   8.7000000...e-02]
 [  5.8000000...e+02   1.1280000...e-01]
 [  6.0000000...e+02   1.3600000...e-01]
 [  5.1000000...e+02   3.1416000...e-01]]
```

__init__(*data=None*, *domain=None*, ***kwargs*)

property strict_name

Getter and setter property for the spectral distribution strict name.

Parameters **value** (unicode) – Value to set the spectral distribution strict name with.

Returns Spectral distribution strict name.

Return type unicode

property wavelengths

Getter and setter property for the spectral distribution wavelengths λ_n.

Parameters **value** (array_like) – Value to set the spectral distribution wavelengths λ_n with.

Returns Spectral distribution wavelengths λ_n.

Return type ndarray

property values

Getter and setter property for the spectral distribution values.

Parameters **value** (array_like) – Value to set the spectral distribution wavelengths values with.

Returns Spectral distribution values.

Return type ndarray

property shape

Getter property for the spectral distribution shape.

Returns Spectral distribution shape.

Return type *SpectralShape*

Notes

- A spectral distribution with a non-uniformly spaced independent variable have multiple intervals, in that case colour.SpectralDistribution.shape attribute returns the *minimum* interval size.

Examples

Shape of a spectral distribution with a uniformly spaced independent variable:

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 20.0)
```

Shape of a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 10.0)
```

interpolate(*shape*, *interpolator=None*, *interpolator_kwargs=None*, ***kwargs*)

Interpolates the spectral distribution in-place according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when interpolator is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator = self.interpolator
elif self.is_uniform():
```

(continues on next page)

(continued from previous page)

```
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (`SpectralShape`, `optional`) – Spectral shape used for interpolation.
- **interpolator** (`object`, `optional`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict_like`, `optional`) – Arguments to use when instantiating the interpolating function.
- ****kwargs** (`dict`, `optional`) – Keywords arguments for deprecation management.

Returns Interpolated spectral distribution.

Return type *SpectralDistribution*

Notes

- Interpolation will be performed over boundaries range, if you need to extend the range of the spectral distribution use the `colour.SpectralDistribution.extrapolate()` or `colour.SpectralDistribution.align()` methods.

Warning:

- *Cubic Spline* interpolator requires at least 3 wavelengths λ_n for interpolation.
- *Sprague (1880)* interpolator requires at least 6 wavelengths λ_n for interpolation.

References

[]

Examples

Spectral distribution with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> 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
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0651   ...]
 [ 501.            0.0653522...]
 [ 502.            0.0656105...]
 [ 503.            0.0658715...]
 [ 504.            0.0661328...]
 [ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
 [ 508.            0.0671613...]
 [ 509.            0.0674150...]
 [ 510.            0.0676692...]
 [ 511.            0.0679253...]
 [ 512.            0.0681848...]
 [ 513.            0.0684491...]
 [ 514.            0.0687197...]
 [ 515.            0.0689975...]
 [ 516.            0.0692832...]
 [ 517.            0.0695771...]
 [ 518.            0.0698787...]
 [ 519.            0.0701870...]
 [ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
 [ 528.            0.0731051...]
 [ 529.            0.0734423...]
 [ 530.            0.0737808...]
 [ 531.            0.0741203...]
 [ 532.            0.0744603...]
 [ 533.            0.0748006...]
 [ 534.            0.0751409...]
 [ 535.            0.0754813...]
 [ 536.            0.0758220...]
 [ 537.            0.0761633...]
 [ 538.            0.0765060...]
```

(continues on next page)

(continued from previous page)

```
[ 539.            0.0768511...]
[ 540.            0.0772   ...]
[ 541.            0.0775527...]
[ 542.            0.0779042...]
[ 543.            0.0782507...]
[ 544.            0.0785908...]
[ 545.            0.0789255...]
[ 546.            0.0792576...]
[ 547.            0.0795917...]
[ 548.            0.0799334...]
[ 549.            0.0802895...]
[ 550.            0.0806671...]
[ 551.            0.0810740...]
[ 552.            0.0815176...]
[ 553.            0.0820049...]
[ 554.            0.0825423...]
[ 555.            0.0831351...]
[ 556.            0.0837873...]
[ 557.            0.0845010...]
[ 558.            0.0852763...]
[ 559.            0.0861110...]
[ 560.            0.087    ...]
[ 561.            0.0879383...]
[ 562.            0.0889300...]
[ 563.            0.0899793...]
[ 564.            0.0910876...]
[ 565.            0.0922541...]
[ 566.            0.0934760...]
[ 567.            0.0947487...]
[ 568.            0.0960663...]
[ 569.            0.0974220...]
[ 570.            0.0988081...]
[ 571.            0.1002166...]
[ 572.            0.1016394...]
[ 573.            0.1030687...]
[ 574.            0.1044972...]
[ 575.            0.1059186...]
[ 576.            0.1073277...]
[ 577.            0.1087210...]
[ 578.            0.1100968...]
[ 579.            0.1114554...]
[ 580.            0.1128   ...]
[ 581.            0.1141333...]
[ 582.            0.1154495...]
[ 583.            0.1167424...]
[ 584.            0.1180082...]
[ 585.            0.1192452...]
[ 586.            0.1204536...]
[ 587.            0.1216348...]
[ 588.            0.1227915...]
[ 589.            0.1239274...]
[ 590.            0.1250465...]
[ 591.            0.1261531...]
[ 592.            0.1272517...]
[ 593.            0.1283460...]
[ 594.            0.1294393...]
```

(continues on next page)

(continued from previous page)

```
 [ 595.            0.1305340...]
 [ 596.            0.1316310...]
 [ 597.            0.1327297...]
 [ 598.            0.1338277...]
 [ 599.            0.1349201...]
 [ 600.            0.136    ...]]
```

Spectral distribution with a non-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> sd = SpectralDistribution(data)
>>> sd[510] = np.pi / 10
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0651   ...]
 [ 501.            0.1365202...]
 [ 502.            0.1953263...]
 [ 503.            0.2423724...]
 [ 504.            0.2785126...]
 [ 505.            0.3046010...]
 [ 506.            0.3214916...]
 [ 507.            0.3300387...]
 [ 508.            0.3310962...]
 [ 509.            0.3255184...]
 [ 510.            0.3141592...]
 [ 511.            0.2978729...]
 [ 512.            0.2775135...]
 [ 513.            0.2539351...]
 [ 514.            0.2279918...]
 [ 515.            0.2005378...]
 [ 516.            0.1724271...]
 [ 517.            0.1445139...]
 [ 518.            0.1176522...]
 [ 519.            0.0926962...]
 [ 520.            0.0705   ...]
 [ 521.            0.0517370...]
 [ 522.            0.0363589...]
 [ 523.            0.0241365...]
 [ 524.            0.0148407...]
 [ 525.            0.0082424...]
 [ 526.            0.0041126...]
 [ 527.            0.0022222...]
 [ 528.            0.0023421...]
 [ 529.            0.0042433...]
 [ 530.            0.0076966...]
 [ 531.            0.0124729...]
 [ 532.            0.0183432...]
 [ 533.            0.0250785...]
 [ 534.            0.0324496...]
 [ 535.            0.0402274...]
 [ 536.            0.0481829...]
 [ 537.            0.0560870...]
 [ 538.            0.0637106...]
 [ 539.            0.0708246...]
 [ 540.            0.0772   ...]
```

(continues on next page)

(continued from previous page)

```
[ 541.            0.0826564...]
[ 542.            0.0872086...]
[ 543.            0.0909203...]
[ 544.            0.0938549...]
[ 545.            0.0960760...]
[ 546.            0.0976472...]
[ 547.            0.0986321...]
[ 548.            0.0990942...]
[ 549.            0.0990971...]
[ 550.            0.0987043...]
[ 551.            0.0979794...]
[ 552.            0.0969861...]
[ 553.            0.0957877...]
[ 554.            0.0944480...]
[ 555.            0.0930304...]
[ 556.            0.0915986...]
[ 557.            0.0902161...]
[ 558.            0.0889464...]
[ 559.            0.0878532...]
[ 560.            0.087    ...]
[ 561.            0.0864371...]
[ 562.            0.0861623...]
[ 563.            0.0861600...]
[ 564.            0.0864148...]
[ 565.            0.0869112...]
[ 566.            0.0876336...]
[ 567.            0.0885665...]
[ 568.            0.0896945...]
[ 569.            0.0910020...]
[ 570.            0.0924735...]
[ 571.            0.0940936...]
[ 572.            0.0958467...]
[ 573.            0.0977173...]
[ 574.            0.0996899...]
[ 575.            0.1017491...]
[ 576.            0.1038792...]
[ 577.            0.1060649...]
[ 578.            0.1082906...]
[ 579.            0.1105408...]
[ 580.            0.1128   ...]
[ 581.            0.1150526...]
[ 582.            0.1172833...]
[ 583.            0.1194765...]
[ 584.            0.1216167...]
[ 585.            0.1236884...]
[ 586.            0.1256760...]
[ 587.            0.1275641...]
[ 588.            0.1293373...]
[ 589.            0.1309798...]
[ 590.            0.1324764...]
[ 591.            0.1338114...]
[ 592.            0.1349694...]
[ 593.            0.1359349...]
[ 594.            0.1366923...]
[ 595.            0.1372262...]
[ 596.            0.1375211...]
```

(continues on next page)

(continued from previous page)

```
[ 597.            0.1375614...]
[ 598.            0.1373316...]
[ 599.            0.1368163...]
[ 600.            0.136    ...]]
```

extrapolate(*shape*, *extrapolator=None*, *extrapolator_kwargs=None*, ***kwargs*)

Extrapolates the spectral distribution in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (SpectralShape) – Spectral shape used for extrapolation.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Extrapolated spectral distribution.

Return type *SpectralDistribution*

References

[], []

Examples

```
>>> 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
... }
>>> sd = SpectralDistribution(data)
>>> sd.extrapolate(SpectralShape(400, 700)).shape
SpectralShape(400.0, 700.0, 20.0)
>>> with numpy_print_options(suppress=True):
...     print(sd)
[[ 400.        0.0651]
 [ 420.        0.0651]
 [ 440.        0.0651]
 [ 460.        0.0651]
 [ 480.        0.0651]
 [ 500.        0.0651]
 [ 520.        0.0705]
 [ 540.        0.0772]
 [ 560.        0.087 ]
 [ 580.        0.1128]
 [ 600.        0.136 ]
 [ 620.        0.136 ]
```

(continues on next page)

(continued from previous page)

```
 [ 640.        0.136 ]
 [ 660.        0.136 ]
 [ 680.        0.136 ]
 [ 700.        0.136 ]]
```

align(*shape*, *interpolator=None*, *interpolator_kwargs=None*, *extrapolator=None*, *extrapolator_kwargs=None*, ***kwargs*)

Aligns the spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Interpolation is performed according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (SpectralShape) – Spectral shape used for alignment.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Aligned spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> 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
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.align(SpectralShape(505, 565, 1)))
...
[[ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
 [ 508.            0.0671613...]
 [ 509.            0.0674150...]
 [ 510.            0.0676692...]
 [ 511.            0.0679253...]
 [ 512.            0.0681848...]
 [ 513.            0.0684491...]
 [ 514.            0.0687197...]
 [ 515.            0.0689975...]
 [ 516.            0.0692832...]
 [ 517.            0.0695771...]
 [ 518.            0.0698787...]
 [ 519.            0.0701870...]
 [ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
 [ 528.            0.0731051...]
 [ 529.            0.0734423...]
 [ 530.            0.0737808...]
 [ 531.            0.0741203...]
 [ 532.            0.0744603...]
 [ 533.            0.0748006...]
 [ 534.            0.0751409...]
 [ 535.            0.0754813...]
 [ 536.            0.0758220...]
 [ 537.            0.0761633...]
 [ 538.            0.0765060...]
 [ 539.            0.0768511...]
 [ 540.            0.0772   ...]
 [ 541.            0.0775527...]
 [ 542.            0.0779042...]
 [ 543.            0.0782507...]
 [ 544.            0.0785908...]
 [ 545.            0.0789255...]
 [ 546.            0.0792576...]
```

(continues on next page)

(continued from previous page)

```
 [ 547.            0.0795917...]
 [ 548.            0.0799334...]
 [ 549.            0.0802895...]
 [ 550.            0.0806671...]
 [ 551.            0.0810740...]
 [ 552.            0.0815176...]
 [ 553.            0.0820049...]
 [ 554.            0.0825423...]
 [ 555.            0.0831351...]
 [ 556.            0.0837873...]
 [ 557.            0.0845010...]
 [ 558.            0.0852763...]
 [ 559.            0.0861110...]
 [ 560.            0.087    ...]
 [ 561.            0.0879383...]
 [ 562.            0.0889300...]
 [ 563.            0.0899793...]
 [ 564.            0.0910876...]
 [ 565.            0.0922541...]]
```

trim(*shape*)

Trims the spectral distribution wavelengths to given spectral shape.

Parameters **shape** (SpectralShape) – Spectral shape used for trimming.

Returns Trimmed spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> 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
... }
>>> sd = SpectralDistribution(data)
>>> sd = sd.interpolate(SpectralShape(interval=1))
>>> with numpy_print_options(suppress=True):
...     print(sd.trim(SpectralShape(520, 580, 5)))
...
[[ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
 [ 528.            0.0731051...]
 [ 529.            0.0734423...]
 [ 530.            0.0737808...]
```

(continues on next page)

(continued from previous page)

```
[ 531.           0.0741203...]
[ 532.           0.0744603...]
[ 533.           0.0748006...]
[ 534.           0.0751409...]
[ 535.           0.0754813...]
[ 536.           0.0758220...]
[ 537.           0.0761633...]
[ 538.           0.0765060...]
[ 539.           0.0768511...]
[ 540.           0.0772   ...]
[ 541.           0.0775527...]
[ 542.           0.0779042...]
[ 543.           0.0782507...]
[ 544.           0.0785908...]
[ 545.           0.0789255...]
[ 546.           0.0792576...]
[ 547.           0.0795917...]
[ 548.           0.0799334...]
[ 549.           0.0802895...]
[ 550.           0.0806671...]
[ 551.           0.0810740...]
[ 552.           0.0815176...]
[ 553.           0.0820049...]
[ 554.           0.0825423...]
[ 555.           0.0831351...]
[ 556.           0.0837873...]
[ 557.           0.0845010...]
[ 558.           0.0852763...]
[ 559.           0.0861110...]
[ 560.           0.087    ...]
[ 561.           0.0879383...]
[ 562.           0.0889300...]
[ 563.           0.0899793...]
[ 564.           0.0910876...]
[ 565.           0.0922541...]
[ 566.           0.0934760...]
[ 567.           0.0947487...]
[ 568.           0.0960663...]
[ 569.           0.0974220...]
[ 570.           0.0988081...]
[ 571.           0.1002166...]
[ 572.           0.1016394...]
[ 573.           0.1030687...]
[ 574.           0.1044972...]
[ 575.           0.1059186...]
[ 576.           0.1073277...]
[ 577.           0.1087210...]
[ 578.           0.1100968...]
[ 579.           0.1114554...]
[ 580.           0.1128   ...]]
```

normalise(*factor=1*)

Normalises the spectral distribution using given normalization factor.

Parameters **factor** (numeric, optional) – Normalization factor

Returns Normalised spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> 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
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.normalise())
[[ 500.            0.4786764...]
 [ 520.            0.5183823...]
 [ 540.            0.5676470...]
 [ 560.            0.6397058...]
 [ 580.            0.8294117...]
 [ 600.            1.        ...]]
```

__repr__()

Returns an ellipsis string representation of the continuous signal for documentation purposes.

Returns Ellipsis string representation.

Return type unicode

colour.MultiSpectralDistributions

class colour.**MultiSpectralDistributions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.continuous.multi_signals.MultiSignals

Defines the multi-spectral distributions: the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc...

The multi-spectral distributions 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.

Important: Specific documentation about getting, setting, indexing and slicing the multi-spectral power distributions values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.MultiSpectralDistributions.labels attribute value.

Attributes

- strict_name
- strict_labels
- wavelengths
- values
- shape

Methods

- __init__()
- interpolate()
- extrapolate()
- align()
- trim()
- normalise()
- to_sds()

References

[], [], []

Examples

Instantiating the multi-spectral distributions 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):
...     MultiSpectralDistributions(data, labels=labels)
...
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_kwargs={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_kwargs={...})
```

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):
...     MultiSpectralDistributions(data, labels=labels)
...
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_kwargs={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_kwargs={...})
```

Instantiation with a *Pandas DataFrame*:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     x_bar = [data[key][0] for key in sorted(data.keys())]
...     y_bar = [data[key][1] for key in sorted(data.keys())]
```

(continues on next page)

(continued from previous page)

```
...     z_bar = [data[key][2] for key in sorted(data.keys())]
...     print(MultiSignals(
...         DataFrame(
...             dict(zip(labels, [x_bar, y_bar, z_bar])), data.keys())))
[[  5.0000000...e+02   4.9000000...e-03   3.2300000...e-01   2.7200000...e-01]
 [  5.1000000...e+02   9.3000000...e-03   5.0300000...e-01   1.5820000...e-01]
 [  5.2000000...e+02   3.1400000...e-03   3.1416000...e-01   3.1420000...e-02]
 [  5.3000000...e+02   6.3270000...e-02   7.1000000...e-01   7.8250000...e-02]
 [  5.4000000...e+02   1.6550000...e-01   8.6200000...e-01   4.2160000...e-02]
 [  5.5000000...e+02   2.9040000...e-01   9.5400000...e-01   2.0300000...e-02]
 [  5.6000000...e+02   4.3345000...e-01   9.9495000...e-01   8.7500000...e-03]
 [  5.1100000...e+02   5.9450000...e-01   9.9500000...e-01   3.9000000...e-03]]
```

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

property strict_name

Getter and setter property for the multi-spectral distributions strict name.

Parameters **value** (unicode) – Value to set the multi-spectral distributions strict name with.

Returns Multi-spectral distributions strict name.

Return type unicode

property strict_labels

Getter and setter property for the multi-spectral distributions strict labels.

Parameters **value** (array_like) – Value to set the multi-spectral distributions strict labels with.

Returns Multi-spectral distributions strict labels.

Return type array_like

property wavelengths

Getter and setter property for the multi-spectral distributions wavelengths λ_n.

Parameters **value** (array_like) – Value to set the multi-spectral distributions wavelengths λ_n with.

Returns Multi-spectral distributions wavelengths λ_n.

Return type ndarray

property values

Getter and setter property for the multi-spectral distributions values.

Parameters **value** (array_like) – Value to set the multi-spectral distributions wavelengths values with.

Returns Multi-spectral distributions values.

Return type ndarray

property shape

Getter property for the multi-spectral distributions shape.

Returns Multi-spectral distributions shape.

Return type *SpectralShape*

Notes

- Multi-spectral distributions with a non-uniformly spaced independent variable have multiple intervals, in that case `colour.MultiSpectralDistributions.shape` attribute returns the *minimum* interval size.

Examples

Shape of the multi-spectral distributions 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)
... }
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 10.0)
```

Shape of the multi-spectral distributions with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 1.0)
```

interpolate(*shape*, *interpolator=None*, *interpolator_kwargs=None*, ***kwargs*)

Interpolates the multi-spectral distributions in-place according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (`SpectralShape`, `optional`) – Spectral shape used for interpolation.
- **interpolator** (`object`, `optional`) – Interpolator class type to use as interpolating function.

- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Interpolated multi-spectral distributions.

Return type *MultiSpectralDistributions*

Notes

- See colour.SpectralDistribution.interpolate() method notes section.

Warning: See colour.SpectralDistribution.interpolate() method warning section.

References

[]

Examples

Multi-spectral distributions with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> 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)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0049    ...    0.323     ...    0.272     ...]
 [ 501.            0.0043252...    0.3400642...    0.2599848...]
 [ 502.            0.0037950...    0.3572165...    0.2479849...]
 [ 503.            0.0033761...    0.3744030...    0.2360688...]
 [ 504.            0.0031397...    0.3916650...    0.2242878...]
 [ 505.            0.0031582...    0.4091067...    0.2126801...]
 [ 506.            0.0035019...    0.4268629...    0.2012748...]
 [ 507.            0.0042365...    0.4450668...    0.1900968...]
 [ 508.            0.0054192...    0.4638181...    0.1791709...]
 [ 509.            0.0070965...    0.4831505...    0.1685260...]
 [ 510.            0.0093    ...    0.503     ...    0.1582    ...]
 [ 511.            0.0120562...    0.5232543...    0.1482365...]
 [ 512.            0.0154137...    0.5439717...    0.1386625...]
 [ 513.            0.0193991...    0.565139  ...    0.1294993...]
 [ 514.            0.0240112...    0.5866255...    0.1207676...]
 [ 515.            0.0292289...    0.6082226...    0.1124864...]
```

(continues on next page)

(continued from previous page)

```
 [ 516.           0.0350192...    0.6296821...    0.1046717...]
 [ 517.           0.0413448...    0.6507558...    0.0973361...]
 [ 518.           0.0481727...    0.6712346...    0.0904871...]
 [ 519.           0.0554816...    0.6909873...    0.0841267...]
 [ 520.           0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.           0.0715642...    0.7283456...    0.0728614...]
 [ 522.           0.0803970...    0.7459679...    0.0680051...]
 [ 523.           0.0897629...    0.7628184...    0.0636823...]
 [ 524.           0.0996227...    0.7789004...    0.0598449...]
 [ 525.           0.1099142...    0.7942533...    0.0564111...]
 [ 526.           0.1205637...    0.8089368...    0.0532822...]
 [ 527.           0.1314973...    0.8230153...    0.0503588...]
 [ 528.           0.1426523...    0.8365417...    0.0475571...]
 [ 529.           0.1539887...    0.8495422...    0.0448253...]
 [ 530.           0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.           0.1772055...    0.8738585...    0.0395936...]
 [ 532.           0.1890877...    0.8850940...    0.0371046...]
 [ 533.           0.2011304...    0.8957073...    0.0346733...]
 [ 534.           0.2133310...    0.9057092...    0.0323006...]
 [ 535.           0.2256968...    0.9151181...    0.0300011...]
 [ 536.           0.2382403...    0.9239560...    0.0277974...]
 [ 537.           0.2509754...    0.9322459...    0.0257131...]
 [ 538.           0.2639130...    0.9400080...    0.0237668...]
 [ 539.           0.2770569...    0.9472574...    0.0219659...]
 [ 540.           0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.           0.3039194...    0.9602409...    0.0187414...]
 [ 542.           0.3175893...    0.9660106...    0.0172748...]
 [ 543.           0.3314022...    0.9713260...    0.0158947...]
 [ 544.           0.3453666...    0.9761850...    0.0146001...]
 [ 545.           0.3595019...    0.9805731...    0.0133933...]
 [ 546.           0.3738324...    0.9844703...    0.0122777...]
 [ 547.           0.3883818...    0.9878583...    0.0112562...]
 [ 548.           0.4031674...    0.9907270...    0.0103302...]
 [ 549.           0.4181943...    0.9930817...    0.0094972...]
 [ 550.           0.43345  ...    0.99495  ...    0.00875  ...]
 [ 551.           0.4489082...    0.9963738...    0.0080748...]
 [ 552.           0.4645599...    0.9973682...    0.0074580...]
 [ 553.           0.4803950...    0.9979568...    0.0068902...]
 [ 554.           0.4963962...    0.9981802...    0.0063660...]
 [ 555.           0.5125410...    0.9980910...    0.0058818...]
 [ 556.           0.5288034...    0.9977488...    0.0054349...]
 [ 557.           0.5451560...    0.9972150...    0.0050216...]
 [ 558.           0.5615719...    0.9965479...    0.0046357...]
 [ 559.           0.5780267...    0.9957974...    0.0042671...]
 [ 560.           0.5945   ...    0.995    ...    0.0039   ...]]
```

Multi-spectral distributions with a non-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(interval=1)))
...
[[ 500.           0.0049   ...    0.323    ...    0.272    ...]
 [ 501.           0.0300110...    0.9455153...    0.5985102...]
```

(continues on next page)

(continued from previous page)

```
[ 502.          0.0462136...    1.3563103...    0.8066498...]
[ 503.          0.0547925...    1.5844039...    0.9126502...]
[ 504.          0.0570325...    1.6588148...    0.9327429...]
[ 505.          0.0542183...    1.6085619...    0.8831594...]
[ 506.          0.0476346...    1.4626640...    0.7801312...]
[ 507.          0.0385662...    1.2501401...    0.6398896...]
[ 508.          0.0282978...    1.0000089...    0.4786663...]
[ 509.          0.0181142...    0.7412892...    0.3126925...]
[ 510.          0.0093   ...    0.503    ...    0.1582   ...]
[ 511.          0.00314  ...    0.31416  ...    0.03142  ...]
[ 512.          0.0006228...    0.1970419...   -0.0551709...]
[ 513.          0.0015528...    0.1469341...   -0.1041165...]
[ 514.          0.0054381...    0.1523785...   -0.1217152...]
[ 515.          0.0117869...    0.2019173...   -0.1142659...]
[ 516.          0.0201073...    0.2840925...   -0.0880670...]
[ 517.          0.0299077...    0.3874463...   -0.0494174...]
[ 518.          0.0406961...    0.5005208...   -0.0046156...]
[ 519.          0.0519808...    0.6118579...    0.0400397...]
[ 520.          0.06327  ...    0.71     ...    0.07825  ...]
[ 521.          0.0741690...    0.7859059...    0.1050384...]
[ 522.          0.0846726...    0.8402033...    0.1207164...]
[ 523.          0.0948728...    0.8759363...    0.1269173...]
[ 524.          0.1048614...    0.8961496...    0.1252743...]
[ 525.          0.1147305...    0.9038874...    0.1174207...]
[ 526.          0.1245719...    0.9021942...    0.1049899...]
[ 527.          0.1344776...    0.8941145...    0.0896151...]
[ 528.          0.1445395...    0.8826926...    0.0729296...]
[ 529.          0.1548497...    0.8709729...    0.0565668...]
[ 530.          0.1655   ...    0.862    ...    0.04216  ...]
[ 531.          0.1765618...    0.858179 ...    0.0309976...]
[ 532.          0.1880244...    0.8593588...    0.0229897...]
[ 533.          0.1998566...    0.8647493...    0.0177013...]
[ 534.          0.2120269...    0.8735601...    0.0146975...]
[ 535.          0.2245042...    0.8850011...    0.0135435...]
[ 536.          0.2372572...    0.8982820...    0.0138044...]
[ 537.          0.2502546...    0.9126126...    0.0150454...]
[ 538.          0.2634650...    0.9272026...    0.0168315...]
[ 539.          0.2768572...    0.9412618...    0.0187280...]
[ 540.          0.2904   ...    0.954    ...    0.0203   ...]
[ 541.          0.3040682...    0.9647869...    0.0211987...]
[ 542.          0.3178617...    0.9736329...    0.0214207...]
[ 543.          0.3317865...    0.9807080...    0.0210486...]
[ 544.          0.3458489...    0.9861825...    0.0201650...]
[ 545.          0.3600548...    0.9902267...    0.0188525...]
[ 546.          0.3744103...    0.9930107...    0.0171939...]
[ 547.          0.3889215...    0.9947048...    0.0152716...]
[ 548.          0.4035944...    0.9954792...    0.0131685...]
[ 549.          0.4184352...    0.9955042...    0.0109670...]
[ 550.          0.43345  ...    0.99495  ...    0.00875  ...]
[ 551.          0.4486447...    0.9939867...    0.0065999...]
[ 552.          0.4640255...    0.9927847...    0.0045994...]
[ 553.          0.4795984...    0.9915141...    0.0028313...]
[ 554.          0.4953696...    0.9903452...    0.0013781...]
[ 555.          0.5113451...    0.9894483...    0.0003224...]
[ 556.          0.5275310...    0.9889934...   -0.0002530...]
[ 557.          0.5439334...    0.9891509...   -0.0002656...]
```

(continues on next page)

(continued from previous page)

```
 [ 558.           0.5605583...    0.9900910...    0.0003672...]
 [ 559.           0.5774118...    0.9919840...    0.0017282...]
 [ 560.           0.5945   ...    0.995    ...    0.0039   ...]]
```

__repr__()

Returns an ellipsis string representation of the continuous signal for documentation purposes.

Returns Ellipsis string representation.

Return type unicode

extrapolate(*shape*, *extrapolator=None*, *extrapolator_kwargs=None*, ***kwargs*)

Extrapolates the multi-spectral distributions in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (SpectralShape) – Spectral shape used for extrapolation.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Extrapolated multi-spectral distributions.

Return type *MultiSpectralDistributions*

References

[], []

Examples

```
>>> 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)
... }
>>> msds = MultiSpectralDistributions(data)
>>> msds.extrapolate(SpectralShape(400, 700)).shape
SpectralShape(400.0, 700.0, 10.0)
>>> with numpy_print_options(suppress=True):
...     print(msds)
[[ 400.         0.0049     0.323      0.272  ]
 [ 410.         0.0049     0.323      0.272  ]
 [ 420.         0.0049     0.323      0.272  ]
 [ 430.         0.0049     0.323      0.272  ]
 [ 440.         0.0049     0.323      0.272  ]
 [ 450.         0.0049     0.323      0.272  ]
```

(continues on next page)

(continued from previous page)

```
 [ 460.          0.0049      0.323       0.272  ]
 [ 470.          0.0049      0.323       0.272  ]
 [ 480.          0.0049      0.323       0.272  ]
 [ 490.          0.0049      0.323       0.272  ]
 [ 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 ]
 [ 570.          0.5945      0.995       0.0039 ]
 [ 580.          0.5945      0.995       0.0039 ]
 [ 590.          0.5945      0.995       0.0039 ]
 [ 600.          0.5945      0.995       0.0039 ]
 [ 610.          0.5945      0.995       0.0039 ]
 [ 620.          0.5945      0.995       0.0039 ]
 [ 630.          0.5945      0.995       0.0039 ]
 [ 640.          0.5945      0.995       0.0039 ]
 [ 650.          0.5945      0.995       0.0039 ]
 [ 660.          0.5945      0.995       0.0039 ]
 [ 670.          0.5945      0.995       0.0039 ]
 [ 680.          0.5945      0.995       0.0039 ]
 [ 690.          0.5945      0.995       0.0039 ]
 [ 700.          0.5945      0.995       0.0039 ]]
```

align(*shape*, *interpolator=None*, *interpolator_kwargs=None*, *extrapolator=None*, *extrapolator_kwargs=None*, ***kwargs*)

Aligns the multi-spectral distributions in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Interpolation is performed according to *CIE 167:2005* recommendation (if the interpolator has not been changed at instantiation time) or given interpolation arguments.

The logic for choosing the interpolator class when `interpolator` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator = self.interpolator
elif self.is_uniform():
    interpolator = SpragueInterpolator
else:
    interpolator = CubicSplineInterpolator
```

The logic for choosing the interpolator keyword arguments when `interpolator_kwargs` is not given is as follows:

```
if self.interpolator not in (SpragueInterpolator,
                             CubicSplineInterpolator):
    interpolator_kwargs = self.interpolator_kwargs
else:
    interpolator_kwargs = {}
```

Parameters

- **shape** (SpectralShape) – Spectral shape used for alignment.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.

- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Aligned multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> 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)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.align(SpectralShape(505, 565, 1)))
...
[[ 505.            0.0031582...    0.4091067...    0.2126801...]
 [ 506.            0.0035019...    0.4268629...    0.2012748...]
 [ 507.            0.0042365...    0.4450668...    0.1900968...]
 [ 508.            0.0054192...    0.4638181...    0.1791709...]
 [ 509.            0.0070965...    0.4831505...    0.1685260...]
 [ 510.            0.0093   ...    0.503    ...    0.1582   ...]
 [ 511.            0.0120562...    0.5232543...    0.1482365...]
 [ 512.            0.0154137...    0.5439717...    0.1386625...]
 [ 513.            0.0193991...    0.565139 ...    0.1294993...]
 [ 514.            0.0240112...    0.5866255...    0.1207676...]
 [ 515.            0.0292289...    0.6082226...    0.1124864...]
 [ 516.            0.0350192...    0.6296821...    0.1046717...]
 [ 517.            0.0413448...    0.6507558...    0.0973361...]
 [ 518.            0.0481727...    0.6712346...    0.0904871...]
 [ 519.            0.0554816...    0.6909873...    0.0841267...]
 [ 520.            0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.            0.0715642...    0.7283456...    0.0728614...]
 [ 522.            0.0803970...    0.7459679...    0.0680051...]
 [ 523.            0.0897629...    0.7628184...    0.0636823...]
 [ 524.            0.0996227...    0.7789004...    0.0598449...]
 [ 525.            0.1099142...    0.7942533...    0.0564111...]
 [ 526.            0.1205637...    0.8089368...    0.0532822...]
 [ 527.            0.1314973...    0.8230153...    0.0503588...]
 [ 528.            0.1426523...    0.8365417...    0.0475571...]
 [ 529.            0.1539887...    0.8495422...    0.0448253...]
 [ 530.            0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.            0.1772055...    0.8738585...    0.0395936...]
```

(continues on next page)

(continued from previous page)

```
 [ 532.            0.1890877...    0.8850940...    0.0371046...]
 [ 533.            0.2011304...    0.8957073...    0.0346733...]
 [ 534.            0.2133310...    0.9057092...    0.0323006...]
 [ 535.            0.2256968...    0.9151181...    0.0300011...]
 [ 536.            0.2382403...    0.9239560...    0.0277974...]
 [ 537.            0.2509754...    0.9322459...    0.0257131...]
 [ 538.            0.2639130...    0.9400080...    0.0237668...]
 [ 539.            0.2770569...    0.9472574...    0.0219659...]
 [ 540.            0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.            0.3039194...    0.9602409...    0.0187414...]
 [ 542.            0.3175893...    0.9660106...    0.0172748...]
 [ 543.            0.3314022...    0.9713260...    0.0158947...]
 [ 544.            0.3453666...    0.9761850...    0.0146001...]
 [ 545.            0.3595019...    0.9805731...    0.0133933...]
 [ 546.            0.3738324...    0.9844703...    0.0122777...]
 [ 547.            0.3883818...    0.9878583...    0.0112562...]
 [ 548.            0.4031674...    0.9907270...    0.0103302...]
 [ 549.            0.4181943...    0.9930817...    0.0094972...]
 [ 550.            0.43345  ...    0.99495  ...    0.00875  ...]
 [ 551.            0.4489082...    0.9963738...    0.0080748...]
 [ 552.            0.4645599...    0.9973682...    0.0074580...]
 [ 553.            0.4803950...    0.9979568...    0.0068902...]
 [ 554.            0.4963962...    0.9981802...    0.0063660...]
 [ 555.            0.5125410...    0.9980910...    0.0058818...]
 [ 556.            0.5288034...    0.9977488...    0.0054349...]
 [ 557.            0.5451560...    0.9972150...    0.0050216...]
 [ 558.            0.5615719...    0.9965479...    0.0046357...]
 [ 559.            0.5780267...    0.9957974...    0.0042671...]
 [ 560.            0.5945   ...    0.995    ...    0.0039   ...]
 [ 561.            0.5945   ...    0.995    ...    0.0039   ...]
 [ 562.            0.5945   ...    0.995    ...    0.0039   ...]
 [ 563.            0.5945   ...    0.995    ...    0.0039   ...]
 [ 564.            0.5945   ...    0.995    ...    0.0039   ...]
 [ 565.            0.5945   ...    0.995    ...    0.0039   ...]]
```

trim(*shape*)

Trims the multi-spectral distributions wavelengths to given shape.

Parameters **shape** (SpectralShape) – Spectral shape used for trimming.

Returns Trimmed multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> 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)
... }
```

(continues on next page)

(continued from previous page)

```
>>> msds = MultiSpectralDistributions(data)
>>> msds = msds.interpolate(SpectralShape(interval=1))
>>> with numpy_print_options(suppress=True):
...     print(msds.trim(SpectralShape(520, 580, 5)))
...
[[ 520.            0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.            0.0715642...    0.7283456...    0.0728614...]
 [ 522.            0.0803970...    0.7459679...    0.0680051...]
 [ 523.            0.0897629...    0.7628184...    0.0636823...]
 [ 524.            0.0996227...    0.7789004...    0.0598449...]
 [ 525.            0.1099142...    0.7942533...    0.0564111...]
 [ 526.            0.1205637...    0.8089368...    0.0532822...]
 [ 527.            0.1314973...    0.8230153...    0.0503588...]
 [ 528.            0.1426523...    0.8365417...    0.0475571...]
 [ 529.            0.1539887...    0.8495422...    0.0448253...]
 [ 530.            0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.            0.1772055...    0.8738585...    0.0395936...]
 [ 532.            0.1890877...    0.8850940...    0.0371046...]
 [ 533.            0.2011304...    0.8957073...    0.0346733...]
 [ 534.            0.2133310...    0.9057092...    0.0323006...]
 [ 535.            0.2256968...    0.9151181...    0.0300011...]
 [ 536.            0.2382403...    0.9239560...    0.0277974...]
 [ 537.            0.2509754...    0.9322459...    0.0257131...]
 [ 538.            0.2639130...    0.9400080...    0.0237668...]
 [ 539.            0.2770569...    0.9472574...    0.0219659...]
 [ 540.            0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.            0.3039194...    0.9602409...    0.0187414...]
 [ 542.            0.3175893...    0.9660106...    0.0172748...]
 [ 543.            0.3314022...    0.9713260...    0.0158947...]
 [ 544.            0.3453666...    0.9761850...    0.0146001...]
 [ 545.            0.3595019...    0.9805731...    0.0133933...]
 [ 546.            0.3738324...    0.9844703...    0.0122777...]
 [ 547.            0.3883818...    0.9878583...    0.0112562...]
 [ 548.            0.4031674...    0.9907270...    0.0103302...]
 [ 549.            0.4181943...    0.9930817...    0.0094972...]
 [ 550.            0.43345  ...    0.99495  ...    0.00875  ...]
 [ 551.            0.4489082...    0.9963738...    0.0080748...]
 [ 552.            0.4645599...    0.9973682...    0.0074580...]
 [ 553.            0.4803950...    0.9979568...    0.0068902...]
 [ 554.            0.4963962...    0.9981802...    0.0063660...]
 [ 555.            0.5125410...    0.9980910...    0.0058818...]
 [ 556.            0.5288034...    0.9977488...    0.0054349...]
 [ 557.            0.5451560...    0.9972150...    0.0050216...]
 [ 558.            0.5615719...    0.9965479...    0.0046357...]
 [ 559.            0.5780267...    0.9957974...    0.0042671...]
 [ 560.            0.5945   ...    0.995    ...    0.0039   ...]]
```

normalise(*factor=1*)

Normalises the multi-spectral distributions with given normalization factor.

Parameters **factor** (numeric, optional) – Normalization factor

Returns Normalised multi- spectral distribution.

Return type *MultiSpectralDistributions*

Notes

- The implementation uses the maximum value for each `colour.SpectralDistribution` class instances.

Examples

```
>>> 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)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.normalise())
[[ 500.            0.0082422...    0.3246231...    1.        ...]
 [ 510.            0.0156434...    0.5055276...    0.5816176...]
 [ 520.            0.1064255...    0.7135678...    0.2876838...]
 [ 530.            0.2783852...    0.8663316...    0.155     ...]
 [ 540.            0.4884777...    0.9587939...    0.0746323...]
 [ 550.            0.7291000...    0.9999497...    0.0321691...]
 [ 560.            1.        ...   1.        ...   0.0143382...]]
```

to_sds()

Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.

Returns List of spectral distributions.

Return type list

Examples

```
>>> 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)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     for sd in msds.to_sds():
...         print(sd)
[[ 500.         0.0049 ...]
 [ 510.         0.0093 ...]
 [ 520.         0.06327...]
 [ 530.         0.1655 ...]
```

(continues on next page)

(continued from previous page)

```
 [ 540.         0.2904 ...]
 [ 550.         0.43345...]
 [ 560.         0.5945 ...]]
[[ 500.         0.323  ...]
 [ 510.         0.503  ...]
 [ 520.         0.71   ...]
 [ 530.         0.862  ...]
 [ 540.         0.954  ...]
 [ 550.         0.99495...]
 [ 560.         0.995  ...]]
[[ 500.         0.272  ...]
 [ 510.         0.1582 ...]
 [ 520.         0.07825...]
 [ 530.         0.04216...]
 [ 540.         0.0203 ...]
 [ 550.         0.00875...]
 [ 560.         0.0039 ...]]
```

SPECTRAL_SHAPE_ASTME308	(360, 780, 1).
SPECTRAL_SHAPE_DEFAULT	(360, 780, 1).

colour.SPECTRAL_SHAPE_ASTME308

colour.**SPECTRAL_SHAPE_ASTME308 = SpectralShape(360, 780, 1)**

(360, 780, 1).

References

[]

SPECTRAL_SHAPE_ASTME308 : SpectralShape

Type Shape for *ASTM E308-15* practise

colour.SPECTRAL_SHAPE_DEFAULT

colour.**SPECTRAL_SHAPE_DEFAULT = SpectralShape(360, 780, 1)**

(360, 780, 1).

References

[]

SPECTRAL_SHAPE_ASTME308 : SpectralShape

Type Shape for *ASTM E308-15* practise

Spectral Data Generation

colour

sd_CIE_standard_illuminant_A([shape])	*CIE Standard Illuminant A* is intended to represent typical, domestic, tungsten-filament lighting.
sd_CIE_illuminant_D_series(xy[, M1_M2_rounding])	Returns the spectral distribution of given *CIE Illuminant D Series* using given *CIE xy* chromaticity coordinates.
sd_blackbody(temperature[, shape, c1, c2, n])	Returns the spectral distribution of the planckian radiator for given temperature $T[K]$ with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$).
sd_constant(k[, shape, dtype])	Returns a spectral distribution of given spectral shape filled with constant k values.
sd_ones([shape])	Returns a spectral distribution of given spectral shape filled with ones.
sd_zeros([shape])	Returns a spectral distribution of given spectral shape filled with zeros.
msds_constant(k, labels[, shape, dtype])	Returns the multi-spectral distributions with given labels and given spectral shape filled with constant k values.
msds_ones(labels[, shape])	Returns the multi-spectral distributionss with given labels and given spectral shape filled with ones.
msds_zeros(labels[, shape])	Returns the multi-spectral distributionss with given labels and given spectral shape filled with zeros.
SD_GAUSSIAN_METHODS	Supported gaussian spectral distribution computation methods.
sd_gaussian(mu_peak_wavelength, sigma_fwhm)	Returns a gaussian spectral distribution of given spectral shape using given method.
SD_SINGLE_LED_METHODS	Supported single *LED* spectral distribution computation methods.
sd_single_led(peak_wavelength, fwhm[, ...])	Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to given method.
SD_MULTI_LEDS_METHODS	Supported multi *LED* spectral distribution computation methods.
sd_multi_leds(peak_wavelengths, fwhm[, ...])	Returns a multi *LED* 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

[]

Examples

```
>>> from colour import SpectralShape
>>> sd_CIE_standard_illuminant_A(SpectralShape(400, 700, 10))
...
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...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
```

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 *CIE xy* chromaticity coordinates.

Parameters

- **xy** (array_like) – *CIE 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 *CIE 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

[], []

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)
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...],
                      [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 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...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                    [ 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=LinearInterpolator,
                   interpolator_kwargs={},
                   extrapolator=Extrapolator,
                   extrapolator_kwargs={...})
```

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 $T[K]$ with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$).

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 $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (numeric, optional) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the cur-

rent International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et al., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.

- **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 with values in *watts per steradian per square metre per nanometer* ($W/sr/m^2/nm$).

Return type *SpectralDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_blackbody(5000)
SpectralDistribution([[   360.        ,   6654.2782706...],
                      [   361.        ,   6709.6052792...],
                      [   362.        ,   6764.8251215...],
                      [   363.        ,   6819.9330786...],
                      [   364.        ,   6874.9244898...],
                      [   365.        ,   6929.7947526...],
                      [   366.        ,   6984.5393232...],
                      [   367.        ,   7039.1537166...],
                      [   368.        ,   7093.6335071...],
                      [   369.        ,   7147.9743284...],
                      [   370.        ,   7202.1718736...],
                      [   371.        ,   7256.2218956...],
                      [   372.        ,   7310.1202073...],
                      [   373.        ,   7363.8626816...],
                      [   374.        ,   7417.4452515...],
                      [   375.        ,   7470.8639102...],
                      [   376.        ,   7524.1147113...],
                      [   377.        ,   7577.1937686...],
                      [   378.        ,   7630.0972565...],
                      [   379.        ,   7682.8214094...],
                      [   380.        ,   7735.3625224...],
                      [   381.        ,   7787.7169506...],
                      [   382.        ,   7839.8811097...],
                      [   383.        ,   7891.8514754...],
                      [   384.        ,   7943.6245836...],
                      [   385.        ,   7995.1970300...],
                      [   386.        ,   8046.5654705...],
                      [   387.        ,   8097.7266205...],
                      [   388.        ,   8148.6772551...],
                      [   389.        ,   8199.4142089...],
                      [   390.        ,   8249.9343757...],
                      [   391.        ,   8300.2347083...],
                      [   392.        ,   8350.3122185...],
                      [   393.        ,   8400.1639766...],
                      [   394.        ,   8449.7871113...],
                      [   395.        ,   8499.1788096...],
                      [   396.        ,   8548.3363163...],
                      [   397.        ,   8597.2569337...],
                      [   398.        ,   8645.9380216...],
                      [   399.        ,   8694.3769968...],
```

(continues on next page)

(continued from previous page)

```
                       [   400.        ,   8742.5713329...],
                       [   401.        ,   8790.5185599...],
                       [   402.        ,   8838.2162638...],
                       [   403.        ,   8885.6620864...],
                       [   404.        ,   8932.8537251...],
                       [   405.        ,   8979.7889322...],
                       [   406.        ,   9026.4655149...],
                       [   407.        ,   9072.8813344...],
                       [   408.        ,   9119.0343064...],
                       [   409.        ,   9164.9223997...],
                       [   410.        ,   9210.5436366...],
                       [   411.        ,   9255.8960922...],
                       [   412.        ,   9300.9778938...],
                       [   413.        ,   9345.7872209...],
                       [   414.        ,   9390.3223045...],
                       [   415.        ,   9434.5814267...],
                       [   416.        ,   9478.5629206...],
                       [   417.        ,   9522.2651692...],
                       [   418.        ,   9565.6866057...],
                       [   419.        ,   9608.8257125...],
                       [   420.        ,   9651.6810212...],
                       [   421.        ,   9694.2511118...],
                       [   422.        ,   9736.5346124...],
                       [   423.        ,   9778.5301986...],
                       [   424.        ,   9820.2365935...],
                       [   425.        ,   9861.6525666...],
                       [   426.        ,   9902.7769336...],
                       [   427.        ,   9943.6085564...],
                       [   428.        ,   9984.1463416...],
                       [   429.        ,  10024.3892411...],
                       [   430.        ,  10064.3362510...],
                       [   431.        ,  10103.9864112...],
                       [   432.        ,  10143.3388051...],
                       [   433.        ,  10182.3925589...],
                       [   434.        ,  10221.1468414...],
                       [   435.        ,  10259.6008633...],
                       [   436.        ,  10297.7538768...],
                       [   437.        ,  10335.6051749...],
                       [   438.        ,  10373.1540914...],
                       [   439.        ,  10410.3999999...],
                       [   440.        ,  10447.3423137...],
                       [   441.        ,  10483.9804852...],
                       [   442.        ,  10520.3140051...],
                       [   443.        ,  10556.3424025...],
                       [   444.        ,  10592.0652439...],
                       [   445.        ,  10627.4821331...],
                       [   446.        ,  10662.5927104...],
                       [   447.        ,  10697.3966524...],
                       [   448.        ,  10731.8936712...],
                       [   449.        ,  10766.0835144...],
                       [   450.        ,  10799.9659640...],
                       [   451.        ,  10833.5408365...],
                       [   452.        ,  10866.8079821...],
                       [   453.        ,  10899.7672843...],
                       [   454.        ,  10932.4186594...],
                       [   455.        ,  10964.7620561...],
```

(continues on next page)

(continued from previous page)

```
                       [   456.        ,  10996.7974551...],
                       [   457.        ,  11028.5248683...],
                       [   458.        ,  11059.9443388...],
                       [   459.        ,  11091.0559402...],
                       [   460.        ,  11121.8597759...],
                       [   461.        ,  11152.3559791...],
                       [   462.        ,  11182.5447121...],
                       [   463.        ,  11212.4261658...],
                       [   464.        ,  11242.0005596...],
                       [   465.        ,  11271.2681403...],
                       [   466.        ,  11300.2291822...],
                       [   467.        ,  11328.8839867...],
                       [   468.        ,  11357.2328813...],
                       [   469.        ,  11385.2762197...],
                       [   470.        ,  11413.0143813...],
                       [   471.        ,  11440.4477705...],
                       [   472.        ,  11467.5768165...],
                       [   473.        ,  11494.4019726...],
                       [   474.        ,  11520.9237164...],
                       [   475.        ,  11547.1425485...],
                       [   476.        ,  11573.0589928...],
                       [   477.        ,  11598.6735959...],
                       [   478.        ,  11623.9869264...],
                       [   479.        ,  11648.9995750...],
                       [   480.        ,  11673.7121534...],
                       [   481.        ,  11698.1252948...],
                       [   482.        ,  11722.2396526...],
                       [   483.        ,  11746.0559008...],
                       [   484.        ,  11769.5747329...],
                       [   485.        ,  11792.7968621...],
                       [   486.        ,  11815.7230205...],
                       [   487.        ,  11838.3539591...],
                       [   488.        ,  11860.6904469...],
                       [   489.        ,  11882.7332712...],
                       [   490.        ,  11904.4832366...],
                       [   491.        ,  11925.9411650...],
                       [   492.        ,  11947.1078953...],
                       [   493.        ,  11967.9842826...],
                       [   494.        ,  11988.5711984...],
                       [   495.        ,  12008.8695298...],
                       [   496.        ,  12028.8801795...],
                       [   497.        ,  12048.6040651...],
                       [   498.        ,  12068.0421192...],
                       [   499.        ,  12087.1952887...],
                       [   500.        ,  12106.0645344...],
                       [   501.        ,  12124.6508312...],
                       [   502.        ,  12142.9551672...],
                       [   503.        ,  12160.9785437...],
                       [   504.        ,  12178.7219748...],
                       [   505.        ,  12196.1864870...],
                       [   506.        ,  12213.3731190...],
                       [   507.        ,  12230.2829214...],
                       [   508.        ,  12246.9169563...],
                       [   509.        ,  12263.2762971...],
                       [   510.        ,  12279.3620282...],
                       [   511.        ,  12295.1752445...],
```

(continues on next page)

(continued from previous page)

```
                       [   512.        ,  12310.7170514...],
                       [   513.        ,  12325.9885643...],
                       [   514.        ,  12340.9909086...],
                       [   515.        ,  12355.7252189...],
                       [   516.        ,  12370.1926394...],
                       [   517.        ,  12384.3943230...],
                       [   518.        ,  12398.3314315...],
                       [   519.        ,  12412.0051350...],
                       [   520.        ,  12425.4166118...],
                       [   521.        ,  12438.5670483...],
                       [   522.        ,  12451.4576382...],
                       [   523.        ,  12464.0895830...],
                       [   524.        ,  12476.4640911...],
                       [   525.        ,  12488.5823780...],
                       [   526.        ,  12500.4456657...],
                       [   527.        ,  12512.0551828...],
                       [   528.        ,  12523.4121640...],
                       [   529.        ,  12534.5178499...],
                       [   530.        ,  12545.3734871...],
                       [   531.        ,  12555.9803275...],
                       [   532.        ,  12566.3396282...],
                       [   533.        ,  12576.4526517...],
                       [   534.        ,  12586.3206651...],
                       [   535.        ,  12595.9449403...],
                       [   536.        ,  12605.3267534...],
                       [   537.        ,  12614.4673849...],
                       [   538.        ,  12623.3681194...],
                       [   539.        ,  12632.0302452...],
                       [   540.        ,  12640.4550541...],
                       [   541.        ,  12648.6438417...],
                       [   542.        ,  12656.5979064...],
                       [   543.        ,  12664.3185499...],
                       [   544.        ,  12671.8070768...],
                       [   545.        ,  12679.0647943...],
                       [   546.        ,  12686.0930120...],
                       [   547.        ,  12692.8930419...],
                       [   548.        ,  12699.4661982...],
                       [   549.        ,  12705.8137971...],
                       [   550.        ,  12711.9371564...],
                       [   551.        ,  12717.8375957...],
                       [   552.        ,  12723.5164362...],
                       [   553.        ,  12728.9750001...],
                       [   554.        ,  12734.2146109...],
                       [   555.        ,  12739.2365933...],
                       [   556.        ,  12744.0422724...],
                       [   557.        ,  12748.6329745...],
                       [   558.        ,  12753.0100260...],
                       [   559.        ,  12757.1747541...],
                       [   560.        ,  12761.1284859...],
                       [   561.        ,  12764.8725489...],
                       [   562.        ,  12768.4082704...],
                       [   563.        ,  12771.7369777...],
                       [   564.        ,  12774.8599976...],
                       [   565.        ,  12777.7786567...],
                       [   566.        ,  12780.4942809...],
                       [   567.        ,  12783.0081955...],
```

(continues on next page)

(continued from previous page)

```
                       [   568.        ,  12785.3217250...],
                       [   569.        ,  12787.4361930...],
                       [   570.        ,  12789.3529220...],
                       [   571.        ,  12791.0732335...],
                       [   572.        ,  12792.5984474...],
                       [   573.        ,  12793.9298826...],
                       [   574.        ,  12795.0688562...],
                       [   575.        ,  12796.0166840...],
                       [   576.        ,  12796.7746799...],
                       [   577.        ,  12797.3441559...],
                       [   578.        ,  12797.7264224...],
                       [   579.        ,  12797.9227874...],
                       [   580.        ,  12797.9345572...],
                       [   581.        ,  12797.7630356...],
                       [   582.        ,  12797.4095241...],
                       [   583.        ,  12796.8753220...],
                       [   584.        ,  12796.1617260...],
                       [   585.        ,  12795.2700302...],
                       [   586.        ,  12794.2015261...],
                       [   587.        ,  12792.9575025...],
                       [   588.        ,  12791.5392453...],
                       [   589.        ,  12789.9480374...],
                       [   590.        ,  12788.1851590...],
                       [   591.        ,  12786.2518870...],
                       [   592.        ,  12784.1494952...],
                       [   593.        ,  12781.8792543...],
                       [   594.        ,  12779.4424316...],
                       [   595.        ,  12776.8402910...],
                       [   596.        ,  12774.0740932...],
                       [   597.        ,  12771.1450952...],
                       [   598.        ,  12768.0545506...],
                       [   599.        ,  12764.8037091...],
                       [   600.        ,  12761.3938171...],
                       [   601.        ,  12757.8261171...],
                       [   602.        ,  12754.1018476...],
                       [   603.        ,  12750.2222435...],
                       [   604.        ,  12746.1885357...],
                       [   605.        ,  12742.0019511...],
                       [   606.        ,  12737.6637126...],
                       [   607.        ,  12733.1750389...],
                       [   608.        ,  12728.5371449...],
                       [   609.        ,  12723.7512409...],
                       [   610.        ,  12718.8185333...],
                       [   611.        ,  12713.7402241...],
                       [   612.        ,  12708.5175109...],
                       [   613.        ,  12703.1515870...],
                       [   614.        ,  12697.6436414...],
                       [   615.        ,  12691.9948585...],
                       [   616.        ,  12686.2064183...],
                       [   617.        ,  12680.2794963...],
                       [   618.        ,  12674.2152632...],
                       [   619.        ,  12668.0148855...],
                       [   620.        ,  12661.6795247...],
                       [   621.        ,  12655.2103378...],
                       [   622.        ,  12648.6084770...],
                       [   623.        ,  12641.8750899...],
```

(continues on next page)

(continued from previous page)

```
                       [   624.        ,  12635.0113192...],
                       [   625.        ,  12628.0183029...],
                       [   626.        ,  12620.8971740...],
                       [   627.        ,  12613.6490609...],
                       [   628.        ,  12606.2750869...],
                       [   629.        ,  12598.7763704...],
                       [   630.        ,  12591.1540251...],
                       [   631.        ,  12583.4091595...],
                       [   632.        ,  12575.5428771...],
                       [   633.        ,  12567.5562766...],
                       [   634.        ,  12559.4504515...],
                       [   635.        ,  12551.2264904...],
                       [   636.        ,  12542.8854766...],
                       [   637.        ,  12534.4284886...],
                       [   638.        ,  12525.8565997...],
                       [   639.        ,  12517.1708779...],
                       [   640.        ,  12508.3723863...],
                       [   641.        ,  12499.4621828...],
                       [   642.        ,  12490.4413201...],
                       [   643.        ,  12481.3108457...],
                       [   644.        ,  12472.0718019...],
                       [   645.        ,  12462.7252260...],
                       [   646.        ,  12453.2721498...],
                       [   647.        ,  12443.7136000...],
                       [   648.        ,  12434.0505982...],
                       [   649.        ,  12424.2841606...],
                       [   650.        ,  12414.4152982...],
                       [   651.        ,  12404.4450167...],
                       [   652.        ,  12394.3743166...],
                       [   653.        ,  12384.2041931...],
                       [   654.        ,  12373.9356362...],
                       [   655.        ,  12363.5696306...],
                       [   656.        ,  12353.1071555...],
                       [   657.        ,  12342.5491851...],
                       [   658.        ,  12331.8966883...],
                       [   659.        ,  12321.1506285...],
                       [   660.        ,  12310.3119640...],
                       [   661.        ,  12299.3816476...],
                       [   662.        ,  12288.3606272...],
                       [   663.        ,  12277.2498448...],
                       [   664.        ,  12266.0502378...],
                       [   665.        ,  12254.7627377...],
                       [   666.        ,  12243.3882711...],
                       [   667.        ,  12231.9277592...],
                       [   668.        ,  12220.3821179...],
                       [   669.        ,  12208.7522577...],
                       [   670.        ,  12197.0390841...],
                       [   671.        ,  12185.2434970...],
                       [   672.        ,  12173.3663914...],
                       [   673.        ,  12161.4086567...],
                       [   674.        ,  12149.3711771...],
                       [   675.        ,  12137.2548318...],
                       [   676.        ,  12125.0604945...],
                       [   677.        ,  12112.7890338...],
                       [   678.        ,  12100.4413128...],
                       [   679.        ,  12088.0181898...],
```

(continues on next page)

(continued from previous page)

```
                       [   680.        ,  12075.5205176...],
                       [   681.        ,  12062.9491438...],
                       [   682.        ,  12050.3049109...],
                       [   683.        ,  12037.5886562...],
                       [   684.        ,  12024.8012117...],
                       [   685.        ,  12011.9434044...],
                       [   686.        ,  11999.016056 ...],
                       [   687.        ,  11986.0199830...],
                       [   688.        ,  11972.9559971...],
                       [   689.        ,  11959.8249045...],
                       [   690.        ,  11946.6275064...],
                       [   691.        ,  11933.3645990...],
                       [   692.        ,  11920.0369733...],
                       [   693.        ,  11906.6454152...],
                       [   694.        ,  11893.1907055...],
                       [   695.        ,  11879.6736202...],
                       [   696.        ,  11866.0949300...],
                       [   697.        ,  11852.4554007...],
                       [   698.        ,  11838.7557929...],
                       [   699.        ,  11824.9968625...],
                       [   700.        ,  11811.1793602...],
                       [   701.        ,  11797.3040317...],
                       [   702.        ,  11783.3716180...],
                       [   703.        ,  11769.3828548...],
                       [   704.        ,  11755.3384733...],
                       [   705.        ,  11741.2391993...],
                       [   706.        ,  11727.0857541...],
                       [   707.        ,  11712.878854 ...],
                       [   708.        ,  11698.6192103...],
                       [   709.        ,  11684.3075296...],
                       [   710.        ,  11669.9445138...],
                       [   711.        ,  11655.5308596...],
                       [   712.        ,  11641.0672593...],
                       [   713.        ,  11626.5544002...],
                       [   714.        ,  11611.9929648...],
                       [   715.        ,  11597.3836310...],
                       [   716.        ,  11582.7270720...],
                       [   717.        ,  11568.0239562...],
                       [   718.        ,  11553.2749471...],
                       [   719.        ,  11538.4807040...],
                       [   720.        ,  11523.6418811...],
                       [   721.        ,  11508.7591283...],
                       [   722.        ,  11493.8330905...],
                       [   723.        ,  11478.8644085...],
                       [   724.        ,  11463.8537180...],
                       [   725.        ,  11448.8016505...],
                       [   726.        ,  11433.7088327...],
                       [   727.        ,  11418.5758871...],
                       [   728.        ,  11403.4034314...],
                       [   729.        ,  11388.1920788...],
                       [   730.        ,  11372.9424382...],
                       [   731.        ,  11357.6551141...],
                       [   732.        ,  11342.3307063...],
                       [   733.        ,  11326.9698104...],
                       [   734.        ,  11311.5730175...],
                       [   735.        ,  11296.1409144...],
```

(continues on next page)

(continued from previous page)

```
                    [   736.        ,  11280.6740836...],
                    [   737.        ,  11265.1731031...],
                    [   738.        ,  11249.6385466...],
                    [   739.        ,  11234.0709837...],
                    [   740.        ,  11218.4709796...],
                    [   741.        ,  11202.8390952...],
                    [   742.        ,  11187.1758873...],
                    [   743.        ,  11171.4819083...],
                    [   744.        ,  11155.7577065...],
                    [   745.        ,  11140.0038261...],
                    [   746.        ,  11124.2208070...],
                    [   747.        ,  11108.4091852...],
                    [   748.        ,  11092.5694922...],
                    [   749.        ,  11076.7022559...],
                    [   750.        ,  11060.8079996...],
                    [   751.        ,  11044.8872430...],
                    [   752.        ,  11028.9405016...],
                    [   753.        ,  11012.9682867...],
                    [   754.        ,  10996.9711059...],
                    [   755.        ,  10980.9494627...],
                    [   756.        ,  10964.9038566...],
                    [   757.        ,  10948.8347833...],
                    [   758.        ,  10932.7427345...],
                    [   759.        ,  10916.6281979...],
                    [   760.        ,  10900.4916576...],
                    [   761.        ,  10884.3335937...],
                    [   762.        ,  10868.1544824...],
                    [   763.        ,  10851.9547962...],
                    [   764.        ,  10835.7350038...],
                    [   765.        ,  10819.4955701...],
                    [   766.        ,  10803.2369563...],
                    [   767.        ,  10786.9596199...],
                    [   768.        ,  10770.6640145...],
                    [   769.        ,  10754.3505902...],
                    [   770.        ,  10738.0197934...],
                    [   771.        ,  10721.6720668...],
                    [   772.        ,  10705.3078497...],
                    [   773.        ,  10688.9275774...],
                    [   774.        ,  10672.5316819...],
                    [   775.        ,  10656.1205916...],
                    [   776.        ,  10639.6947313...],
                    [   777.        ,  10623.2545223...],
                    [   778.        ,  10606.8003824...],
                    [   779.        ,  10590.3327259...],
                    [   780.        ,  10573.8519636...]],
                   interpolator=SpragueInterpolator,
                   interpolator_kwargs={},
                   extrapolator=Extrapolator,
                   extrapolator_kwargs={...})
```

colour.sd_constant

colour.**sd_constant**(*k*, *shape=SpectralShape(360, 780, 1)*, *dtype=None*)

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 filled spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT 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.SPECTRAL_SHAPE_DEFAULT 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.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_zeros()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
0.0
```

colour.msds_constant

colour.**msds_constant**(*k*, *labels*, *shape=SpectralShape(360, 780, 1)*, *dtype=None*)

Returns the multi-spectral distributions with given labels and given spectral shape filled with constant k values.

Parameters

- **k** (numeric) – Constant k to fill the multi-spectral distributions with.
- **labels** (array_like) – Names to use for the colour.SpectralDistribution class instances.
- **shape** (SpectralShape, optional) – Spectral shape used to create the multi-spectral distributions.
- **dtype** (type) – Data type used for the multi-spectral distributions.

Returns Constant k filled multi-spectral distributions.

Return type *MultiSpectralDistributions*

Notes

- By default, the multi-spectral distributions will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.

Examples

```
>>> msds = msds_constant(100, labels=['a', 'b', 'c'])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 100.,  100.,  100.])
>>> msds.labels
['a', 'b', 'c']
```

colour.msds_ones

`colour.`**`msds_ones`**(*labels*, *shape=SpectralShape(360, 780, 1)*)

Returns the multi-spectral distributionss with given labels and given spectral shape filled with ones.

Parameters

- **labels** (`array_like`) – Names to use for the `colour.SpectralDistribution` class instances.
- **shape** (`SpectralShape`, `optional`) – Spectral shape used to create the multi-spectral distributions.

Returns Ones filled multi-spectral distributions.

Return type *MultiSpectralDistributions*

Notes

- By default, the multi-spectral distributions will use the shape given by `colour.SPECTRAL_SHAPE_DEFAULT` attribute.

Examples

```
>>> msds = msds_ones(labels=['a', 'b', 'c'])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 1.,  1.,  1.])
>>> msds.labels
['a', 'b', 'c']
```

colour.msds_zeros

colour.**msds_zeros**(*labels*, *shape=SpectralShape(360, 780, 1)*)

Returns the multi-spectral distributionss with given labels and given spectral shape filled with zeros.

Parameters

- **labels** (array_like) – Names to use for the colour.SpectralDistribution class instances.
- **shape** (SpectralShape, optional) – Spectral shape used to create the multi-spectral distributions.

Returns Zeros filled multi-spectral distributions.

Return type *MultiSpectralDistributions*

Notes

- By default, the multi-spectral distributions will use the shape given by colour.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> msds = msds_zeros(labels=['a', 'b', 'c'])
>>> msds.shape
SpectralShape(360.0, 780.0, 1.0)
>>> msds[400]
array([ 0.,  0.,  0.])
>>> msds.labels
['a', 'b', 'c']
```

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.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
>>> sd[530]
0.6065306...
>>> sd = sd_gaussian(555, 25, method='FWHM')
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
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.SPECTRAL_SHAPE_DEFAULT` attribute.

References

[], []

Examples

```
>>> sd = sd_single_led(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
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.SPECTRAL_SHAPE_DEFAULT` attribute.

References

[], []

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]
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 *CIE xy* chromaticity coordinates.
`sd_gaussian_normal`(mu, sigma[, shape])	Returns a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation $sigma$.
`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 *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to *Ohno (2005)* method.
`sd_multi_leds_Ohno2005`(peak_wavelengths, fwhm)	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.
`sds_and_msds_to_sds`(sds)	Converts given spectral and multi-spectral distributions to a flat list of spectral distributions.
`sds_and_msds_to_msds`(sds)	Converts given spectral and multi-spectral distributions to multi-spectral distributions.

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 $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (numeric or array_like, optional) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et aI., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **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* $(W/sr/m^2)$.

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* $(W/sr/m^2)$.

References

[]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

colour.colorimetry.daylight_locus_function

colour.colorimetry.**daylight_locus_function**(*x_D*)

Returns the daylight locus as *CIE xy* chromaticity coordinates.

Parameters **x_D** (numeric or array_like) – *x* chromaticity coordinates

Returns Daylight locus as *CIE xy* chromaticity coordinates.

Return type numeric or array_like

References

[]

Examples

```
>>> daylight_locus_function(0.31270)
0.3291051...
```

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.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian_normal(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
>>> sd[530]
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.SPECTRAL_SHAPE_DEFAULT attribute.

Examples

```
>>> sd = sd_gaussian_fwhm(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
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.SPECTRAL_SHAPE_DEFAULT` attribute.

References

[], []

Examples

```
>>> sd = sd_single_led_Ohno2005(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
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.SPECTRAL_SHAPE_DEFAULT` attribute.

References

[], []

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]
0.1295132...
```

colour.colorimetry.sds_and_msds_to_sds

colour.colorimetry.**sds_and_msds_to_sds**(*sds*)

Converts given spectral and multi-spectral distributions to a flat list of spectral distributions.

Parameters **sds** (array_like) – Spectral and multi-spectral distributions to convert to a flat list of spectral distributions.

Returns Flat list of spectral distributions.

Return type list

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> 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)
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> len(sds_and_msds_to_sds([sd_1, sd_2, multi_sds_1, multi_sds_2]))
8
```

colour.colorimetry.sds_and_msds_to_msds

colour.colorimetry.**sds_and_msds_to_msds**(*sds*)

Converts given spectral and multi-spectral distributions to multi-spectral distributions.

The spectral and multi-spectral distributions will be aligned to the intersection of their spectral shapes.

Parameters **sds** (array_like) – Spectral and multi-spectral distributions to convert to multi-spectral distributions.

Returns Multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> 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)
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True, linewidth=160):
...     sds_and_msds_to_msds(
...         [sd_1, sd_2, multi_sds_1, multi_sds_2])
MultiSpectralDistributions([[ 500.        ,    0.0651  ...,0.0651  ...,    0.0049 ␣
↪ ...,    0.323   ...,    0.272   ...,0.0049  ...,    0.323   ...,    0.272   .
↪..],
                            [ 510.        ,    0.0676692...,0.0676692...,    0.0093 ␣
↪ ...,    0.503   ...,    0.1582  ...,0.0093  ...,    0.503   ...,    0.1582  .
↪..],
                            [ 520.        ,    0.0705  ...,0.0705  ...,    0.06327␣
↪ ...,    0.71    ...,    0.07825 ...,0.06327 ...,    0.71    ...,    0.07825 .
↪..],
                            [ 530.        ,    0.0737808...,0.0737808...,    0.1655 ␣
↪ ...,    0.862   ...,    0.04216 ...,0.1655  ...,    0.862   ...,    0.04216 .
↪..],
                            [ 540.        ,    0.0772  ...,0.0772  ...,    0.2904 ␣
↪ ...,    0.954   ...,    0.0203  ...,0.2904  ...,    0.954   ...,    0.0203  .
↪..],
                            [ 550.        ,    0.0806671...,0.0806671...,    0.43345␣
↪ ...,    0.99495 ...,    0.00875 ...,0.43345 ...,    0.99495 ...,    0.00875 .
↪..],
```

(continues on next page)

(continued from previous page)

```
                           [ 560.        ,    0.087    ...,0.087    ...,    0.5945 ␣
↪ ...,    0.995    ...,    0.0039   ...,0.5945   ...,    0.995    ...,    0.0039   .
↪..]],
                          labels=['SpectralDistribution (...)',
↪'SpectralDistribution (...)', '0 - SpectralDistribution (...)', '1 -␣
↪SpectralDistribution (...)', '2 - SpectralDistribution (...)', '0 -␣
↪SpectralDistribution (...)', '1 - SpectralDistribution (...)', '2 -␣
↪SpectralDistribution (...)'],
                          interpolator=SpragueInterpolator,
                          interpolator_kwargs={},
                          extrapolator=Extrapolator,
                          extrapolator_kwargs={...})
```

Aliases

`colour.colorimetry`

`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.

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 $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (numeric or array_like, optional) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et aI., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **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* ($W/sr/m^2$).

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* ($W/sr/m^2$).

References

[]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

Conversion to Tristimulus Values

colour

sd_to_XYZ(sd[, cmfs, illuminant, k, method])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.
SD_TO_XYZ_METHODS	Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.
msds_to_XYZ(msds[, cmfs, illuminant, k, method])	Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.
MSDS_TO_XYZ_METHODS	Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.
wavelength_to_XYZ(wavelength[, cmfs])	Converts given wavelength λ to *CIE XYZ* 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'*, ***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 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

- **method** (unicode, optional) – **{'ASTM E308', 'Integration'}**, Computation method.
- **mi_5nm_omission_method** (bool, optional) – {colour.colorimetry.sd_to_XYZ_ASTME308()}, 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_ASTME308()}, 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_ASTME308()}, 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

[], [], []

Examples

```
>>> from colour import (
...     MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution)
>>> cmfs = MSDS_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 = SDS_ILLUMINANTS['D65']
>>> sd_to_XYZ(sd, cmfs, illuminant)
...
array([ 10.8401953...,   9.6841740...,   6.2158913...])
>>> sd_to_XYZ(sd, cmfs, illuminant, use_practice_range=False)
...
array([ 10.8402774...,   9.6841967...,   6.2158838...])
>>> sd_to_XYZ(sd, cmfs, illuminant, method='Integration')
...
array([ 10.8404805...,   9.6838697...,   6.2115722...])
```

colour.SD_TO_XYZ_METHODS

colour.**SD_TO_XYZ_METHODS = CaseInsensitiveMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015': ...})**

Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.

References

[], [], []

SD_TO_XYZ_METHODS [CaseInsensitiveMapping] **{'ASTM E308', 'Integration'}**

Aliases:

- 'astm2015': 'ASTM E308'

colour.msds_to_XYZ

colour.**msds_to_XYZ**(*msds*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*, *method='ASTM E308'*, ***kwargs*)

Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant. For the *Integration* method, the multi-spectral distributions can be either a `colour.MultiSpectralDistributions` class instance or an *array_like* in which case the shape must be passed.

Parameters

- **msds** (`MultiSpectralDistributions or array_like`) – Multi-spectral distributions, if an *array_like* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, `msds` shape should be (384, 512, 77).
- **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 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

- **method** (unicode, optional) – **{'ASTM E308', 'Integration'}**, Computation method.
- **use_practice_range** (bool, optional) – {colour.colorimetry.msds_to_XYZ_ASTME308()}, 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) – {colour.colorimetry.msds_to_XYZ_ASTME308()}, 5 nm measurement intervals multi-spectral distributions 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.msds_to_XYZ_ASTME308()}, 20 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **shape** (SpectralShape, optional) – {colour.colorimetry.msds_to_XYZ_integration()}, Spectral shape of the multi-spectral distributions array $msds$, `cmfs` and `illuminant` will be aligned to it.

Returns *CIE XYZ* tristimulus values, for a 512x384 multi-spectral image with 77 wavelengths, the output shape will be (384, 512, 3).

Return type array_like

Notes

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

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a colour.MultiSpectralDistributions class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[], [], []

Examples

```
>>> shape = SpectralShape(400, 700, 60)
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
>>> msds_to_XYZ(msds, method='Integration', shape=shape)
...
array([[  8.2415862...,   4.2543993...,   7.6100842...],
       [ 29.6144619...,  16.1158465...,  25.9015472...],
       [ 16.6799560...,  27.2350547...,  22.9413337...],
       [ 12.5597688...,   9.0667136...,   5.9670327...],
       [ 18.5804689...,  33.6618109...,  26.9249733...],
       [ 47.7113308...,  40.4573249...,  39.6439145...],
       [  7.830207 ...,  12.3689624...,  23.3742655...],
       [ 24.1695370...,  20.0629815...,   7.2718670...],
       [  7.2333751...,   2.7982097...,  10.0688374...],
       [ 48.7358074...,  30.2417164...,  10.6753233...],
       [  8.3231013...,  18.6791507...,  15.8228184...],
       [ 24.6452277...,  26.0809382...,  27.7106399...]])
>>> msds = 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],
...     ],
... ])
>>> msds_to_XYZ(msds, method='Integration', shape=shape)
...
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...]],

      [[  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.MSDS_TO_XYZ_METHODS

colour.**MSDS_TO_XYZ_METHODS = CaseInsensitiveMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015': ...})**

Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.

References

[], [], []

MSDS_TO_XYZ_METHODS [CaseInsensitiveMapping] **{'ASTM E308', 'Integration'}**

Aliases:

- 'astm2015': 'ASTM E308'

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 MSDS_CMFS
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> wavelength_to_XYZ(480, cmfs)
array([ 0.09564  ,  0.13902  ,  0.8129501...])
>>> wavelength_to_XYZ(480.5, cmfs)
array([ 0.0914287...,  0.1418350...,  0.7915726...])
```

ASTM E308-15

colour.colorimetry

sd_to_XYZ_ASTME308(sd[, cmfs, illuminant, ...])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.
msds_to_XYZ_ASTME308(msds[, cmfs, ...])	Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

colour.colorimetry.sd_to_XYZ_ASTME308

colour.colorimetry.**sd_to_XYZ_ASTME308**(*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 $lm \cdot W^{-1}$) 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

[]

Examples

```
>>> from colour import (
...     MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution)
>>> cmfs = MSDS_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 = SDS_ILLUMINANTS['D65']
>>> sd_to_XYZ_ASTME308(sd, cmfs, illuminant)
```

(continues on next page)

(continued from previous page)

```
...
array([ 10.8401953...,   9.6841740...,   6.2158913...])
```

colour.colorimetry.msds_to_XYZ_ASTME308

colour.colorimetry.**msds_to_XYZ_ASTME308**(*msds*, *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 multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **msds** (MultiSpectralDistributions or array_like) – Multi-spectral distributions.
- **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 multi-spectral distributions 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 multi-spectral distributions 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 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **shape** (SpectralShape, optional) – Spectral shape of the multi-spectral distributions, cmfs and illuminant will be aligned to it.

Returns *CIE XYZ* tristimulus values.

Return type array_like

Notes

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

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a `colour.MultiSpectralDistributions` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> shape = SpectralShape(400, 700, 60)
>>> D65 = SDS_ILLUMINANTS['D65']
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
>>> msds = msds.align(SpectralShape(400, 700, 20))
>>> msds_to_XYZ_ASTME308(msds, illuminant=D65)
...
array([[  7.5052758...,   3.9557516...,   8.38929  ...],
       [ 26.9408494...,  15.0987746...,  28.6631260...],
       [ 16.7047370...,  28.2089815...,  25.6556751...],
       [ 11.5711808...,   8.6445071...,   6.5587827...],
       [ 18.7428858...,  35.0626352...,  30.1778517...],
       [ 45.1224886...,  39.6238997...,  43.5813345...],
       [  8.1786985...,  13.0950215...,  25.9326459...],
       [ 22.4462888...,  19.3115133...,   7.9304333...],
       [  6.5764361...,   2.5305945...,  11.07253  ...],
       [ 43.9113380...,  28.0003541...,  11.6852531...],
       [  8.5496209...,  19.6913570...,  17.7400079...],
       [ 23.8866733...,  26.2147704...,  30.6297684...]])
```

Ancillary Objects

`colour.colorimetry`

sd_to_XYZ_tristimulus_weighting_factors_ASTME308(sd, ...)	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.
adjust_tristimulus_weighting_factors_ASTME308(W, ...)	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.
lagrange_coefficients_ASTME2022([interval, ...])	Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.
tristimulus_weighting_factors_ASTME2022(...)	Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

colour.colorimetry.sd_to_XYZ_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**sd_to_XYZ_tristimulus_weighting_factors_ASTME308**(*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 $lm \cdot W^{-1}$) 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

[]

Examples

```
>>> from colour import (
...     MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution)
>>> cmfs = MSDS_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 = SDS_ILLUMINANTS['D65']
>>> sd_to_XYZ_tristimulus_weighting_factors_ASTME308(
...     sd, cmfs, illuminant)
array([ 10.8405832...,   9.6844909...,   6.2155622...])
```

colour.colorimetry.adjust_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**adjust_tristimulus_weighting_factors_ASTME308**(*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

[]

Examples

```
>>> from colour import (MSDS_CMFS, sd_CIE_standard_illuminant_A,
...     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = MSDS_CMFS['CIE 1964 10 Degree Standard Observer']
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> W = tristimulus_weighting_factors_ASTME2022(
...     cmfs, A, SpectralShape(360, 830, 20))
>>> with numpy_print_options(suppress=True):
...     adjust_tristimulus_weighting_factors_ASTME308(
...         W,  SpectralShape(360, 830, 20), SpectralShape(400, 700, 20))
...
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_ASTME2022

colour.colorimetry.**lagrange_coefficients_ASTME2022**(*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

[]

Examples

```
>>> lagrange_coefficients_ASTME2022(10, 'inner')
...
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_ASTME2022(10, 'boundary')
...
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_ASTME2022

colour.colorimetry.**tristimulus_weighting_factors_ASTME2022**(*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 $lm \cdot W^{-1}$) 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.

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

[]

Examples

```
>>> from colour import (MSDS_CMFS, sd_CIE_standard_illuminant_A,
...     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = MSDS_CMFS['CIE 1964 10 Degree Standard Observer']
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> with numpy_print_options(suppress=True):
...     tristimulus_weighting_factors_ASTME2022(
...         cmfs, A, SpectralShape(360, 830, 20))
...
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

sd_to_XYZ_integration(sd[, cmfs, illuminant, k])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.
msds_to_XYZ_integration(msds[, cmfs, ...])	Converts given multi-spectral distributions to *CIE XYZ* 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 $lm \cdot W^{-1}$) 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

[]

Examples

```
>>> from colour import (
...     MSDS_CMFS, SDS_ILLUMINANTS, SpectralDistribution)
>>> cmfs = MSDS_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 = SDS_ILLUMINANTS['D65']
>>> sd_to_XYZ_integration(sd, cmfs, illuminant)
...
array([ 10.8404805...,   9.6838697...,   6.2115722...])
```

colour.colorimetry.msds_to_XYZ_integration

colour.colorimetry.**msds_to_XYZ_integration**(*msds*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*, *shape=SpectralShape(360, 780, 1)*)

Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant. The multi-spectral distribution can be either a colour.MultiSpectralDistributions class instance or an *array_like* in which case the shape must be passed.

Parameters

- **msds** (MultiSpectralDistributions or array_like) – Multi-spectral distributions, if an *array_like* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, msds shape should be (384, 512, 77).
- **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 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **shape** (`SpectralShape, optional`) – Spectral shape of the multi-spectral distributions, `cmfs` and `illuminant` will be aligned to it.

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]

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a `colour.MultiSpectralDistributions` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> shape = SpectralShape(400, 700, 60)
>>> D65 = SDS_ILLUMINANTS['D65']
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
```

(continues on next page)

(continued from previous page)

```
>>> msds_to_XYZ_integration(msds, illuminant=D65, shape=shape)
...
array([[  7.5029651...,   3.9487840...,   8.4034770...],
       [ 26.925986 ...,  15.0724738...,  28.7058153...],
       [ 16.7031140...,  28.2172235...,  25.6456293...],
       [ 11.5767146...,   8.6401095...,   6.5768486...],
       [ 18.7313077...,  35.0750086...,  30.1457629...],
       [ 45.1657291...,  39.6137391...,  43.6784025...],
       [  8.1755520...,  13.0934236...,  25.9421257...],
       [ 22.4676530...,  19.3099303...,   7.9637645...],
       [  6.5780111...,   2.5254943...,  11.0930902...],
       [ 43.9146821...,  27.9803874...,  11.7292796...],
       [  8.5363407...,  19.7029458...,  17.7051147...],
       [ 23.9088530...,  26.2129842...,  30.6763518...]])
>>> msds = 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],
...     ],
... ])
>>> msds_to_XYZ_integration(msds, illuminant=D65, shape=shape)
...
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...]],

       [[  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

bandpass_correction(sd[, method])	Implements spectral bandpass dependence correction on given spectral distribution using given method.
BANDPASS_CORRECTION_METHODS	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*

References

[], []

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,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction(SpectralDistribution(data))
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
```

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

[], []

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,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction_Stearns1988(SpectralDistribution(data))
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
```

(continues on next page)

(continued from previous page)

```
                    extrapolator=Extrapolator,
                    extrapolator_kwargs={...})
```

Colour Matching Functions

colour.colorimetry

LMS_ConeFundamentals([data, domain, labels])	Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.
RGB_ColourMatchingFunctions([data, domain, ...])	Implements support for the *CIE RGB* colour matching functions.
XYZ_ColourMatchingFunctions([data, domain, ...])	Implements support for the *CIE* Standard Observers *XYZ* colour matching functions.

colour.colorimetry.LMS_ConeFundamentals

class colour.colorimetry.**LMS_ConeFundamentals**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.colorimetry.LMS_ConeFundamentals.labels attribute value.

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

colour.colorimetry.RGB_ColourMatchingFunctions

class colour.colorimetry.**RGB_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.colorimetry.RGB_ColourMatchingFunctions.labels attribute value.

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

colour.colorimetry.XYZ_ColourMatchingFunctions

class colour.colorimetry.**XYZ_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the *CIE* Standard Observers *XYZ* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.colorimetry.XYZ_ColourMatchingFunctions.labels attribute value.

__init__(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Dataset

colour

MSDS_CMFS	Multi-spectral distributions of the colour matching functions.

colour.MSDS_CMFS

colour.**MSDS_CMFS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 10 Degree Cone Fundamentals': ..., 'Smith & Pokorny 1975 Normal Trichromats': ..., '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': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Multi-spectral distributions of the colour matching functions.

References

[], [], [], [], [], [], []

MSDS_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'}**

Ancillary Objects

colour.colorimetry

MSDS_CMFS_LMS	Multi-spectral distributions of the *LMS* colour matching functions.
MSDS_CMFS_RGB	Multi-spectral distributions of the *RGB* colour matching functions.
MSDS_CMFS_STANDARD_OBSERVER	Multi-spectral distributions of the *CIE* Standard Observer colour matching functions.

colour.colorimetry.MSDS_CMFS_LMS

colour.colorimetry.**MSDS_CMFS_LMS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sharpe 10 Degree Cone Fundamentals': ..., 'Smith & Pokorny 1975 Normal Trichromats': ...})**

Multi-spectral distributions of the *LMS* colour matching functions.

References

[], []

MSDS_CMFS_LMS [CaseInsensitiveMapping] {'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Smith & Pokorny 1975 Normal Trichromats'}

colour.colorimetry.MSDS_CMFS_RGB

colour.colorimetry.**MSDS_CMFS_RGB = CaseInsensitiveMapping({'Wright & Guild 1931 2 Degree RGB CMFs': ..., 'Stiles & Burch 1955 2 Degree RGB CMFs': ..., 'Stiles & Burch 1959 10 Degree RGB CMFs': ...})**

Multi-spectral distributions of the *RGB* colour matching functions.

References

[], [], []

MSDS_CMFS_RGB [CaseInsensitiveMapping] **{'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs'}**

colour.colorimetry.MSDS_CMFS_STANDARD_OBSERVER

colour.colorimetry.**MSDS_CMFS_STANDARD_OBSERVER = 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': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Multi-spectral distributions of the *CIE* Standard Observer colour matching functions.

References

[], []

MSDS_CMFS_STANDARD_OBSERVER [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`

`RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs`(...)	Converts *Wright & Guild 1931 2 Degree RGB CMFs* colour matching functions into the *CIE 1931 2 Degree Standard Observer* colour matching functions.
`RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs`(...)	Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *CIE 1964 10 Degree Standard Observer* colour matching functions.
`RGB_10_degree_cmfs_to_LMS_10_degree_cmfs`(...)	Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral sensitivity functions.
`LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs`(...)	Converts *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 2 Degree Standard Observer* colour matching functions.
`LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs`(...)	Converts *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 10 Degree Standard Observer* 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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(700)
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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(700)
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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(700)
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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
array([ 0.0098162...,  0.0037761...,  0.        ])
```

Illuminants and Light Sources

Dataset

colour

CCS_ILLUMINANTS	Chromaticity coordinates of the illuminants.
SDS_ILLUMINANTS	Spectral distributions of the illuminants.
CCS_LIGHT_SOURCES	Chromaticity coordinates of the light sources.
SDS_LIGHT_SOURCES	Spectral distributions of the light sources.
TVS_ILLUMINANTS_HUNTERLAB	Tristimulus values of the *HunterLab* illuminants.

colour.CCS_ILLUMINANTS

colour.**CCS_ILLUMINANTS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Chromaticity coordinates of the illuminants.

> **Warning:** *DCI-P3* illuminant has no associated spectral distribution. *DCI* has no official reference spectral measurement for this whitepoint. The closest matching spectral distribution is *Kinoton 75P* projector.

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)
colour.XYZ_to_xy(
    colour.sd_to_XYZ(
        sd, colour.MSDS_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 colourspaces litterature. Their chromaticity coordinates as given in [] are (0.34567, 0.35851) and (0.31272, 0.32903) respectively.
- *CIE* illuminants with chromaticity coordinates not defined in the reference [] have been calculated using their correlated colour temperature and `colour.temperature.CCT_to_xy_CIE_D()` `colour.sd_CIE_illuminant_D_series()` and / or `colour.sd_to_XYZ()` definitions.
- *ICC D50* chromaticity coordinates were computed with `colour.XYZ_to_xy()` definition from the *CIE XYZ* tristimulus values as given by *ICC*: [96.42, 100.00, 82.49].

References

[], [], [], [], [], []

CCS_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.SDS_ILLUMINANTS

colour.**SDS_ILLUMINANTS = CaseInsensitiveMapping({'A': ..., 'B': ..., 'C': ..., 'D50': ..., 'D55': ..., 'D60': ..., 'D65': ..., 'D75': ..., 'E': ..., 'FL1': ..., 'FL2': ..., 'FL3': ..., 'FL4': ..., 'FL5': ..., 'FL6': ..., 'FL7': ..., 'FL8': ..., 'FL9': ..., 'FL10': ..., 'FL11': ..., 'FL12': ..., 'FL3.1': ..., 'FL3.2': ..., 'FL3.3': ..., 'FL3.4': ..., 'FL3.5': ..., 'FL3.6': ..., 'FL3.7': ..., 'FL3.8': ..., 'FL3.9': ..., 'FL3.10': ..., 'FL3.11': ..., 'FL3.12': ..., 'FL3.13': ..., 'FL3.14': ..., 'FL3.15': ..., 'HP1': ..., 'HP2': ..., 'HP3': ..., 'HP4': ..., 'HP5': ..., 'LED-B1': ..., 'LED-B2': ..., 'LED-B3': ..., 'LED-B4': ..., 'LED-B5': ..., 'LED-BH1': ..., 'LED-RGB1': ..., 'LED-V1': ..., 'LED-V2': ..., 'ID65': ..., 'ID50': ..., 'ISO 7589 Photographic Daylight': ..., 'ISO 7589 Sensitometric Daylight': ..., 'ISO 7589 Studio Tungsten': ..., 'ISO 7589 Sensitometric Studio Tungsten': ..., 'ISO 7589 Photoflood': ..., 'ISO 7589 Sensitometric Photoflood': ..., 'ISO 7589 Sensitometric Printer': ...})**

Spectral distributions of the illuminants.

References

[], [], [], []

SDS_ILLUMINANTS : CaseInsensitiveMapping

colour.CCS_LIGHT_SOURCES

colour.**CCS_LIGHT_SOURCES = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Chromaticity coordinates of the light sources.

CCS_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.SDS_LIGHT_SOURCES

colour.**SDS_LIGHT_SOURCES = CaseInsensitiveMapping({'Natural': ..., 'Philips TL-84': ..., 'SA': ..., 'SC': ..., 'T8 Luxline Plus White': ..., 'T8 Polylux 3000': ..., 'T8 Polylux 4000': ..., 'Thorn Kolor-rite': ..., 'Cool White FL': ..., 'Daylight FL': ..., 'HPS': ..., 'Incandescent': ..., 'LPS': ..., 'Mercury': ..., 'Metal Halide': ..., 'Neodimium Incandescent': ..., 'Super HPS': ..., 'Triphosphor FL': ..., '3-LED-1 (457/540/605)': ..., '3-LED-2 (473/545/616)': ..., '3-LED-2 Yellow': ..., '3-LED-3 (465/546/614)': ..., '3-LED-4 (455/547/623)': ..., '4-LED No Yellow': ..., '4-LED Yellow': ..., '4-LED-1 (461/526/576/624)': ..., '4-LED-2 (447/512/573/627)': ..., 'Luxeon WW 2880': ..., 'PHOS-1': ..., 'PHOS-2': ..., 'PHOS-3': ..., 'PHOS-4': ..., 'Phosphor LED YAG': ..., '60 A/W (Soft White)': ..., 'C100S54 (HPS)': ..., 'C100S54C (HPS)': ..., 'F32T8/TL830 (Triphosphor)': ..., 'F32T8/TL835 (Triphosphor)': ..., 'F32T8/TL841 (Triphosphor)': ..., 'F32T8/TL850 (Triphosphor)': ..., 'F32T8/TL865 /PLUS (Triphosphor)': ..., 'F34/CW/RS/EW (Cool White FL)': ..., 'F34T12/LW/RS /EW': ..., 'F34T12WW/RS /EW (Warm White FL)': ..., 'F40/C50 (Broadband FL)': ..., 'F40/C75 (Broadband FL)': ..., 'F40/CWX (Broadband FL)': ..., 'F40/DX (Broadband FL)': ..., 'F40/DXTP (Delux FL)': ..., 'F40/N (Natural FL)': ..., 'H38HT-100 (Mercury)': ..., 'H38JA-100/DX (Mercury DX)': ..., 'MHC100/U/MP /3K': ..., 'MHC100/U/MP /4K': ..., 'SDW-T 100W/LV (Super HPS)': ..., 'Kinoton 75P': ...})**

Spectral distributions of the light sources.

References

[], [], []

SDS_LIGHT_SOURCES : CaseInsensitiveMapping

colour.TVS_ILLUMINANTS_HUNTERLAB

colour.**TVS_ILLUMINANTS_HUNTERLAB = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Degree Standard Observer': ..., 'cie_2_1931': ..., 'cie_10_1964': ...})**

Tristimulus values of the *HunterLab* illuminants.

References

[], []

TVS_ILLUMINANTS_HUNTERLAB [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'

Ancillary Objects

colour.colorimetry

SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES	*CIE Illuminant D Series* $S_n(\lambda)$ spectral distributions.

colour.colorimetry.SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES

colour.colorimetry.**SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES = CaseInsensitiveMapping({'S0': ..., 'S1': ..., 'S2': ...})**

CIE Illuminant D Series $S_n(\lambda)$ spectral distributions.

References

[], []

SDS_BASIS_FUNCTIONS_CIE_ILLUMINANT_D_SERIES [CaseInsensitiveMapping] **{'S0', 'S1', 'S1'}**

Dominant Wavelength and Purity

colour

dominant_wavelength(xy, xy_n[, cmfs, inverse])	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.
complementary_wavelength(xy, xy_n[, cmfs])	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.
excitation_purity(xy, xy_n[, cmfs])	Returns the *excitation purity* P_e for given colour stimulus xy.
colorimetric_purity(xy, xy_n[, cmfs])	Returns the *colorimetric purity* 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', ...)*, *inverse=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 *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **inverse** (bool, optional) – Inverse the computation direction to retrieve the *complementary wavelength*.

Returns *Dominant wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type tuple

References

[], []

Examples

Dominant wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(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))
(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 *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Complementary wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type tuple

References

[], []

Examples

Complementary wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.37605506, 0.24452225])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(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))
(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 *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Excitation purity* P_e.

Return type numeric or array_like

References

[], []

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> excitation_purity(xy, xy_n, cmfs)
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 *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Colorimetric purity* P_c.

Return type numeric or array_like

References

[], []

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> colorimetric_purity(xy, xy_n, cmfs)
0.6135828...
```

Luminous Efficiency Functions

colour

luminous_efficacy(sd[, lef])	Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.
luminous_efficiency(sd[, lef])	Returns the *luminous efficiency* of given spectral distribution using given luminous efficiency function.
luminous_flux(sd[, lef, K_m])	Returns the *luminous flux* for given spectral distribution using given luminous efficiency function.
sd_mesopic_luminous_efficiency_function(Lp)	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

[]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> luminous_efficacy(sd)
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

[]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> luminous_efficiency(sd)
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

[]

Examples

```
>>> from colour import SDS_LIGHT_SOURCES
>>> sd = SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> luminous_flux(sd)
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', ...)*, *scotopic_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

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_mesopic_luminous_efficiency_function(0.2)
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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 388.        ,    0.0011923...],
                       [ 389.        ,    0.0013703...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 444.        ,    0.2753387...],
                       [ 445.        ,    0.2848520...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                    [ 500.        ,    0.8352251...],
                    [ 501.        ,    0.8456342...],
                    [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 556.        ,    0.7627808...],
                       [ 557.        ,    0.7522710...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 612.        ,    0.2471885...],
                       [ 613.        ,    0.2403570...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 668.        ,    0.0183056...],
                       [ 669.        ,    0.0171216...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 724.        ,    0.0003971...],
                       [ 725.        ,    0.0003704...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                    [ 780.        ,    0.0000075...]],
                   interpolator=SpragueInterpolator,
                   interpolator_kwargs={},
                   extrapolator=Extrapolator,
                   extrapolator_kwargs={...})
```

Dataset

`colour`

SDS_LEFS	Spectral distributions of the luminous efficiency functions.

colour.SDS_LEFS

colour.**SDS_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_2_1924': ..., 'cie_10_1964': ..., 'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': ...})**

Spectral distributions of the luminous efficiency functions.

References

[], [], []

SDS_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'}**

Ancillary Objects

`colour.colorimetry`

SDS_LEFS_PHOTOPIC	Spectral distributions of the photopic luminous efficiency functions.
SDS_LEFS_SCOTOPIC	Spectral distributions of the scotopic luminous efficiency functions.

colour.colorimetry.SDS_LEFS_PHOTOPIC

colour.colorimetry.**SDS_LEFS_PHOTOPIC = 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_2_1924': ..., 'cie_10_1964': ...})**

Spectral distributions of the photopic luminous efficiency functions.

References

[], []

SDS_LEFS_PHOTOPIC [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.colorimetry.SDS_LEFS_SCOTOPIC

colour.colorimetry.**SDS_LEFS_SCOTOPIC = CaseInsensitiveMapping({'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': ...})**

Spectral distributions of the scotopic luminous efficiency functions.

References

[]

SDS_LEFS_SCOTOPIC [CaseInsensitiveMapping] **{'CIE 1951 Scotopic Standard Observer', }**

Aliases:

- 'cie_1951': 'CIE 1951 Scotopic Standard Observer'

Lightness Computation

colour

lightness(Y[, method])	Returns the *Lightness* L of given *luminance* Y using given method.
LIGHTNESS_METHODS	Supported *Lightness* 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.
- **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

[], [], [], [], [], [], []

Examples

```
>>> lightness(12.19722535)
41.5278758...
>>> lightness(12.19722535, Y_n=100)
41.5278758...
>>> lightness(12.19722535, Y_n=95)
42.5199307...
>>> lightness(12.19722535, method='Glasser 1958')
39.8351264...
>>> lightness(12.19722535, method='Wyszecki 1963')
40.5475745...
>>> lightness(12.19722535, epsilon=0.710, method='Fairchild 2011')
...
29.8295108...
```

colour.LIGHTNESS_METHODS

colour.**LIGHTNESS_METHODS = CaseInsensitiveMapping({'Glasser 1958': ..., 'Wyszecki 1963': ..., 'CIE 1976': ..., 'Fairchild 2010': ..., 'Fairchild 2011': ..., 'Lstar1976': ...})**

Supported *Lightness* computation methods.

References

[], [], [], [], [], []

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

lightness_Glasser1958(Y)	Returns the *Lightness* L of given *luminance* Y using *Glasser et al. (1958)* 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

[]

Examples

```
>>> lightness_Glasser1958(12.19722535)
39.8351264...
```

Wyszecki (1963)

colour.colorimetry

lightness_Wyszecki1963(Y)	Returns the *Lightness* W of given *luminance* Y using *Wyszecki (1963)* 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

[]

Examples

```
>>> lightness_Wyszecki1963(12.19722535)
40.5475745...
```

CIE 1976

colour.colorimetry

lightness_CIE1976(Y[, Y_n])	Returns the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.
intermediate_lightness_function_CIE1976(Y[, Y_n])	Returns the intermediate value $f(Y/Yn)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* 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

[], []

Examples

```
>>> lightness_CIE1976(12.19722535)
41.5278758...
```

colour.colorimetry.intermediate_lightness_function_CIE1976

colour.colorimetry.**intermediate_lightness_function_CIE1976**(*Y*, *Y_n=100*)

Returns the intermediate value $f(Y/Yn)$ 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 $f(Y/Yn)$.

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

[], []

Examples

```
>>> intermediate_lightness_function_CIE1976(12.19722535)
...
0.4959299...
>>> intermediate_lightness_function_CIE1976(12.19722535, 95)
...
0.5044821...
```

Fairchild and Wyble (2010)

colour.colorimetry

lightness_Fairchild2010(Y[, epsilon])	Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* 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

[]

Examples

```
>>> lightness_Fairchild2010(12.19722535 / 100)
31.9963902...
```

Fairchild and Chen (2011)

colour.colorimetry

lightness_Fairchild2011(Y[, epsilon, method])	Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michealis-Menten* 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

[]

Examples

```
>>> lightness_Fairchild2011(12.19722535 / 100)
51.8529584...
>>> lightness_Fairchild2011(12.19722535 / 100, method='hdr-IPT')
...
51.6431084...
```

Luminance Computation

colour

luminance(LV[, method])	Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.
LUMINANCE_METHODS	Supported *luminance* 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', 'Fairchild 2010', 'Fairchild 2011'}**, Computation method.
- **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

[], [], [], [], [], [], []

Examples

```
>>> luminance(41.527875844653451)
12.1972253...
>>> luminance(41.527875844653451, Y_n=100)
12.1972253...
>>> luminance(42.51993072812094, Y_n=95)
12.1972253...
>>> luminance(4.08244375 * 10, method='Newhall 1943')
...
12.5500788...
>>> luminance(4.08244375 * 10, method='ASTM D1535')
...
12.2363426...
>>> luminance(29.829510892279330, epsilon=0.710, method='Fairchild 2011')
...
12.1972253...
```

colour.LUMINANCE_METHODS

colour.**LUMINANCE_METHODS = CaseInsensitiveMapping({'Newhall 1943': ..., 'ASTM D1535': ..., 'CIE 1976': ..., 'Fairchild 2010': ..., 'Fairchild 2011': ..., 'astm2008': ..., 'cie1976': ...})**

Supported *luminance* computation methods.

References

[], [], [], [], [], []

LUMINANCE_METHODS [CaseInsensitiveMapping] **{'Newhall 1943', 'ASTM D1535', 'CIE 1976', 'Fairchild 2010'}**

Aliases:

- 'astm2008': 'ASTM D1535'
- 'cie1976': 'CIE 1976'

Newhall, Nickerson and Judd (1943)

colour.colorimetry

luminance_Newhall1943(V)	Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et al. (1943)* 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

[]

Examples

```
>>> luminance_Newhall1943(4.08244375)
12.5500788...
```

CIE 1976

colour.colorimetry

luminance_CIE1976(L_star[, Y_n])	Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.
intermediate_luminance_function_CIE1976(f_Y_Yn)	Returns the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Yn)$ using given reference white *luminance* 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

[], []

Examples

```
>>> luminance_CIE1976(41.527875844653451)
12.1972253...
>>> luminance_CIE1976(41.527875844653451, 95)
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 $f(Y/Yn)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **f_Y_Y_n** (numeric or array_like) – Intermediate value $f(Y/Yn)$.
- **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

[], []

Examples

```
>>> intermediate_luminance_function_CIE1976(0.495929964178047)
...
12.1972253...
>>> intermediate_luminance_function_CIE1976(0.504482161449319, 95)
...
12.1972253...
```

ASTM D1535-08e1

colour.colorimetry

luminance_ASTMD1535(V)	Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

colour.colorimetry.luminance_ASTMD1535

colour.colorimetry.**luminance_ASTMD1535**(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

[]

Examples

```
>>> luminance_ASTMD1535(4.08244375)
12.2363426...
```

Fairchild and Wyble (2010)

colour.colorimetry

luminance_Fairchild2010(L_hdr[, epsilon])	Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* 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

[]

Examples

```
>>> luminance_Fairchild2010(31.996390226262736, 1.836)
...
0.1219722...
```

Fairchild and Chen (2011)

colour.colorimetry

luminance_Fairchild2011(L_hdr[, epsilon, method])		Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michealis-Menten* 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

[]

Examples

```
>>> luminance_Fairchild2011(51.852958445912506)
0.1219722...
>>> luminance_Fairchild2011(51.643108411718522, method='hdr-IPT')
...
0.1219722...
```

Whiteness Computation

colour

whiteness(XYZ, XYZ_0[, method])	Returns the *whiteness* W using given method.
WHITENESS_METHODS	Supported *whiteness* 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.
- **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

[], [], [], []

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)
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')
91.4071738...
```

colour.WHITENESS_METHODS

colour.**WHITENESS_METHODS = CaseInsensitiveMapping({'Berger 1959': ..., 'Taube 1960': ..., 'Stensby 1968': ..., 'ASTM E313': ..., 'Ganz 1979': ..., 'CIE 2004': ..., 'cie2004': ...})**

Supported *whiteness* computation methods.

References

[], []

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

whiteness_Berger1959(XYZ, XYZ_0)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Berger (1959)* 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

[]

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)
30.3638017...
```

Taube (1960)

colour.colorimetry

whiteness_Taube1960(XYZ, XYZ_0)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Taube (1960)* 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

[]

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)
91.4071738...
```

Stensby (1968)

colour.colorimetry

whiteness_Stensby1968(Lab)	Returns the *whiteness* index WI of given sample *CIE L*a*b** colourspace array using *Stensby (1968)* 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

[]

Examples

```
>>> import numpy as np
>>> Lab = np.array([100.00000000, -2.46875131, -16.72486654])
>>> whiteness_Stensby1968(Lab)
142.7683456...
```

ASTM E313

colour.colorimetry

whiteness_ASTME313(XYZ)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *ASTM E313* 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> whiteness_ASTME313(XYZ)
55.7400000...
```

Ganz and Griesser (1979)

colour.colorimetry

whiteness_Ganz1979(xy, Y)	Returns the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

colour.colorimetry.whiteness_Ganz1979

colour.colorimetry.**whiteness_Ganz1979**(*xy*, *Y*)

Returns the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *CIE 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

[]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> whiteness_Ganz1979(xy, 100)
array([ 85.6003766...,   0.6789003...])
```

CIE 2004

colour.colorimetry

whiteness_CIE2004(xy, Y, xy_n[, observer])	Returns the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *CIE xy* chromaticity coordinates using *CIE 2004* 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 *CIE xy* chromaticity coordinates using *CIE 2004* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *CIE 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 5Y10 - 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

[]

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)
array([ 93.85...,  -1.305...])
```

Yellowness Computation

colour

yellowness(XYZ[, method])	Returns the *yellowness* W using given method.
YELLOWNESS_METHODS	Supported *yellowness* 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness(XYZ)
11.0650000...
>>> method = 'ASTM D1925'
>>> yellowness(XYZ, method=method)
10.2999999...
```

colour.YELLOWNESS_METHODS

colour.**YELLOWNESS_METHODS = CaseInsensitiveMapping({'ASTM D1925': ..., 'ASTM E313': ...})**

Supported *yellowness* computation methods.

References

[]

YELLOWNESS_METHODS [CaseInsensitiveMapping] **{'ASTM E313', 'ASTM D1925'}**

ASTM D1925

colour.colorimetry

yellowness_ASTMD1925(XYZ)	Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM D1925* 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTMD1925(XYZ)
10.2999999...
```

ASTM E313

colour.colorimetry

yellowness_ASTME313(XYZ)	Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM E313* 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTME313(XYZ)
11.0650000...
```

Constants

- *CIE*
- *CODATA*
- *Common*

CIE

colour.constants

CONSTANT_K_M	Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.
CONSTANT_KP_M	Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

colour.constants.CONSTANT_K_M

colour.constants.**CONSTANT_K_M = 683.0**

Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.

CONSTANT_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

[]

colour.constants.CONSTANT_KP_M

colour.constants.**CONSTANT_KP_M = 1700.0**

Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

CONSTANT_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

[]

CODATA

colour.constants

CONSTANT_AVOGADRO	Avogadro constant.
CONSTANT_BOLTZMANN	Boltzmann constant.
CONSTANT_LIGHT_SPEED	Speed of light in vacuum.
CONSTANT_PLANCK	Planck constant.

colour.constants.CONSTANT_AVOGADRO

colour.constants.**CONSTANT_AVOGADRO = 6.02214179e+23**

Avogadro constant.

CONSTANT_AVOGADRO : numeric

colour.constants.CONSTANT_BOLTZMANN

colour.constants.**CONSTANT_BOLTZMANN = 1.38065e-23**

Boltzmann constant.

CONSTANT_BOLTZMANN : numeric

colour.constants.CONSTANT_LIGHT_SPEED

colour.constants.**CONSTANT_LIGHT_SPEED = 299792458.0**

Speed of light in vacuum.

CONSTANT_LIGHT_SPEED : numeric

colour.constants.CONSTANT_PLANCK

colour.constants.**CONSTANT_PLANCK = 6.62607e-34**

Planck constant.

CONSTANT_PLANCK : numeric

Common

colour.constants

DEFAULT_FLOAT_DTYPE	alias of numpy.float64
DEFAULT_INT_DTYPE	alias of numpy.int64
EPSILON	Double-precision floating-point number type, compatible with Python *float* and C double.
FLOATING_POINT_NUMBER_PATTERN	str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
INTEGER_THRESHOLD	Integer threshold value when checking if a floating point number is almost an integer.

colour.constants.DEFAULT_FLOAT_DTYPE

colour.constants.**DEFAULT_FLOAT_DTYPE**

alias of numpy.float64

colour.constants.DEFAULT_INT_DTYPE

colour.constants.**DEFAULT_INT_DTYPE**

alias of numpy.int64

colour.constants.EPSILON

colour.constants.**EPSILON = 2.2204460492503131e-16**

Double-precision floating-point number type, compatible with Python *float* and C double.

Character code 'd'

Canonical name *numpy.double*

Alias *numpy.float_*

Alias on this platform (Linux x86_64) *numpy.float64*: 64-bit precision floating-point number type: sign bit, 11 bits exponent, 52 bits mantissa.

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 when checking if a floating point number is almost an integer.

INTEGER_THRESHOLD : numeric

Contrast Sensitivity

- *Contrast Sensitivity*
- *Barten (1999) Contrast Sensitivity Function*

Contrast Sensitivity

colour

contrast_sensitivity_function([method])	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.
CONTRAST_SENSITIVITY_METHODS	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.
- **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 $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **phi_0** (numeric or array_like, optional) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spectral density ϕ_0 in $seconds degrees^2$ 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 $cycles \div degrees$ above which the lateral inhibition ceases.

Returns Contrast sensitivity S.

Return type ndarray

References

[], [], [], [],

Examples

```
>>> contrast_sensitivity_function(u=4)
360.8691122...
>>> contrast_sensitivity_function('Barten 1999', u=4)
360.8691122...
```

colour.CONTRAST_SENSITIVITY_METHODS

colour.**CONTRAST_SENSITIVITY_METHODS = CaseInsensitiveMapping({'Barten 1999': ...})**

Supported contrast sensitivity methods.

References

[], [], [], [],

CONTRAST_SENSITIVITY_METHODS [CaseInsensitiveMapping] **{'Barten 1999'}**

Barten (1999) Contrast Sensitivity Function

colour.contrast

contrast_sensitivity_function_Barten1999(u)	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

colour.contrast.contrast_sensitivity_function_Barten1999

colour.contrast.**contrast_sensitivity_function_Barten1999**(*u*, *sigma=0.008791157173230634*, *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=66.08231606052992*, *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 $seconds degrees^2$ 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 $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ 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 [] 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

[], [], [], [],

Examples

```
>>> contrast_sensitivity_function_Barten1999(4)
360.8691122...
```

Reproducing *Figure 31* in [] 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)
...         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))
```

(continues on next page)

(continued from previous page)

```
...
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`

optical_MTF_Barten1999(u[, sigma])	Returns the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.
pupil_diameter_Barten1999(L[, X_0, Y_0])	Returns the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.
sigma_Barten1999([sigma_0, C_ab, d])	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.
retinal_illuminance_Barten1999(L[, d, ...])	Returns the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.
maximum_angular_size_Barten1999(u[, X_0, ...])	Returns the maximum angular size X of the object considered using *Barten (1999)* 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

[], [], [], [],

Examples

```
>>> optical_MTF_Barten1999(4, 0.01)
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

[], [], [], [],

Examples

```
>>> pupil_diameter_Barten1999(100, 60, 60)
2.0777571...
```

colour.contrast.sigma_Barten1999

colour.contrast.**sigma_Barten1999**(*sigma_0=0.008333333333333333*, *C_ab=0.001333333333333333*, *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 $degrees \div mm$.
- **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 $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ respectively, thus they need to be divided by 60.

References

[], [], [], [],

Examples

```
>>> sigma_Barten1999(0.5 / 60, 0.08 / 60, 2.1)
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

[], [], [], [],

Examples

```
>>> retinal_illuminance_Barten1999(100, 2.1)
330.4115803...
>>> retinal_illuminance_Barten1999(100, 2.1, False)
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

[], [], [], [],

Examples

```
>>> maximum_angular_size_Barten1999(4)
3.5729480...
```

Continuous Signal

- *Continuous Signal*

Continuous Signal

colour.continuous

AbstractContinuousFunction([name])	Defines the base class for abstract continuous function.
Signal([data, domain])	Defines the base class for continuous signal.
MultiSignals([data, domain, labels])	Defines the base class for multi-continuous signals, a container for multiple colour.continuous.Signal sub-class instances.

colour.continuous.AbstractContinuousFunction

class colour.continuous.**AbstractContinuousFunction**(*name=None*)

Bases: object

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.

Attributes

- name
- domain
- range
- interpolator
- interpolator_kwargs
- extrapolator
- extrapolator_kwargs
- function

Methods

- __init__()
- __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*)

property name

Getter and setter property for the abstract continuous function name.

Parameters **value** (unicode) – Value to set the abstract continuous function name with.

Returns Abstract continuous function name.

Return type unicode

abstract property domain

Getter and setter property for the abstract continuous function independent domain x variable, must be reimplemented by sub-classes.

Parameters **value** (array_like) – Value to set the abstract continuous function independent domain x variable with.

Returns Abstract continuous function independent domain x variable.

Return type ndarray

abstract property range

Getter and setter property for the abstract continuous function corresponding range y variable, must be reimplemented by sub-classes.

Parameters **value** (array_like) – Value to set the abstract continuous function corresponding range y variable with.

Returns Abstract continuous function corresponding range y variable.

Return type ndarray

abstract property interpolator

Getter and setter property for the abstract continuous function interpolator type, must be reimplemented by sub-classes.

Parameters **value** (type) – Value to set the abstract continuous function interpolator type with.

Returns Abstract continuous function interpolator type.

Return type type

abstract property interpolator_kwargs

Getter and setter property for the abstract continuous function interpolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters **value** (dict) – Value to set the abstract continuous function interpolator instantiation time arguments to.

Returns Abstract continuous function interpolator instantiation time arguments.

Return type dict

abstract property extrapolator

Getter and setter property for the abstract continuous function extrapolator type, must be reimplemented by sub-classes.

Parameters **value** (type) – Value to set the abstract continuous function extrapolator type with.

Returns Abstract continuous function extrapolator type.

Return type type

abstract property extrapolator_kwargs

Getter and setter property for the abstract continuous function extrapolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters **value** (dict) – Value to set the abstract continuous function extrapolator instantiation time arguments to.

Returns Abstract continuous function extrapolator instantiation time arguments.

Return type dict

abstract property function

Getter and setter property for the abstract continuous function callable, must be reimplemented by sub-classes.

Parameters **value** (object) – Attribute value.

Returns Abstract continuous function callable.

Return type callable

abstract __str__()

Returns a formatted string representation of the abstract continuous function, must be reimplemented by sub-classes.

Returns Formatted string representation.

Return type unicode

abstract __repr__()

Returns an evaluable string representation of the abstract continuous function, must be reimplemented by sub-classes.

Returns Evaluable string representation.

Return type unicode

abstract __hash__()

Returns the abstract continuous function hash.

Returns Object hash.

Return type int

abstract __getitem__(*x*)

Returns the corresponding range y variable for independent domain x variable, must be reimplemented by sub-classes.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns math:y range value.

Return type numeric or ndarray

abstract __setitem__(*x*, *y*)

Sets the corresponding range y variable for independent domain x variable, must be reimplemented by sub-classes.

Parameters

- **x** (numeric, array_like or slice) – Independent domain x variable.
- **y** (numeric or ndarray) – Corresponding range y variable.

abstract __contains__(*x*)

Returns whether the abstract continuous function contains given independent domain x variable, must be reimplemented by sub-classes.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns Is x domain value contained.

Return type bool

__len__()

Returns the abstract continuous function independent domain x variable elements count.

Returns Independent domain x variable elements count.

Return type int

abstract __eq__(*other*)

Returns whether the abstract continuous function is equal to given other object, must be reimplemented by sub-classes.

Parameters **other** (object) – Object to test whether it is equal to the abstract continuous function.

Returns Is given object equal to the abstract continuous function.

Return type bool

abstract __ne__(*other*)

Returns whether the abstract continuous function is not equal to given other object, must be reimplemented by sub-classes.

Parameters **other** (object) – Object to test whether it is not equal to the abstract continuous function.

Returns Is given object not equal to the abstract continuous function.

Return type bool

__add__(*a*)

Implements support for addition.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to add.

Returns Variable added abstract continuous function.

Return type *AbstractContinuousFunction*

__iadd__(*a*)

Implements support for in-place addition.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to add in-place.

Returns In-place variable added abstract continuous function.

Return type *AbstractContinuousFunction*

__sub__(*a*)

Implements support for subtraction.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to subtract.

Returns Variable subtracted abstract continuous function.

Return type *AbstractContinuousFunction*

__isub__(*a*)

Implements support for in-place subtraction.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to subtract in-place.

Returns In-place variable subtracted abstract continuous function.

Return type *AbstractContinuousFunction*

__mul__(*a*)

Implements support for multiplication.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to multiply by.

Returns Variable multiplied abstract continuous function.

Return type *AbstractContinuousFunction*

__imul__(*a*)

Implements support for in-place multiplication.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to multiply by in-place.

Returns In-place variable multiplied abstract continuous function.

Return type *AbstractContinuousFunction*

__div__(*a*)

Implements support for division.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to divide by.

Returns Variable divided abstract continuous function.

Return type *AbstractContinuousFunction*

__idiv__(*a*)

Implements support for in-place division.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to divide by in-place.

Returns In-place variable divided abstract continuous function.

Return type *AbstractContinuousFunction*

__itruediv__(*a*)

Implements support for in-place division.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to divide by in-place.

Returns In-place variable divided abstract continuous function.

Return type *AbstractContinuousFunction*

__truediv__(*a*)

Implements support for division.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to divide by.

Returns Variable divided abstract continuous function.

Return type *AbstractContinuousFunction*

__pow__(*a*)

Implements support for exponentiation.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to exponentiate by.

Returns Variable exponentiated abstract continuous function.

Return type *AbstractContinuousFunction*

__ipow__(*a*)

Implements support for in-place exponentiation.

Parameters **a** (numeric or array_like or AbstractContinuousFunction) – a variable to exponentiate by in-place.

Returns In-place variable exponentiated abstract continuous function.

Return type *AbstractContinuousFunction*

abstract arithmetical_operation(*a*, *operation*, *in_place=False*)

Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.

Parameters

- **a** (numeric or ndarray or AbstractContinuousFunction) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns Abstract continuous function.

Return type *AbstractContinuousFunction*

abstract fill_nan(*method='Interpolation'*, *default=0*)

Fill NaNs in independent domain x variable and corresponding range y variable using given method, must be reimplemented by sub-classes.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with `default`.
- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns NaNs filled abstract continuous function.

Return type *AbstractContinuousFunction*

__weakref__

list of weak references to the object (if defined)

domain_distance(*a*)

Returns the euclidean distance between given array and independent domain x closest element.

Parameters **a** (numeric or array_like) – a variable to compute the euclidean distance with independent domain x variable.

Returns Euclidean distance between independent domain x variable and given a variable.

Return type numeric or array_like

is_uniform()

Returns if independent domain x variable is uniform.

Returns Is independent domain x variable uniform.

Return type bool

copy()

Returns a copy of the sub-class instance.

Returns Abstract continuous function copy.

Return type *AbstractContinuousFunction*

colour.continuous.Signal

class colour.continuous.**Signal**(*data=None*, *domain=None*, ***kwargs*)

Bases: colour.continuous.abstract.AbstractContinuousFunction

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.

Important: Specific documentation about getting, setting, indexing and slicing the continuous signal values is available in the *Spectral Representation and Continuous Signal* section.

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 with be used to initialise the colour.continuous.Signal.domain attribute.
- **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_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.

- **extrapolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

Attributes

- dtype
- domain
- range
- interpolator
- interpolator_kwargs
- extrapolator
- extrapolator_kwargs
- function

Methods

- __init__()
- __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.]
 [  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(
...         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]
0.0359701...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(range_, domain)[x]
array([  ...,   4.7669395...e-01,   8.4147098...e-01])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> Signal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
0.0555274...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([ 0.        ,  0.4794253...,  0.8414709...])
```

__init__(*data=None*, *domain=None*, ***kwargs*)

property dtype

Getter and setter property for the continuous signal dtype.

Parameters **value** (type) – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type type

property domain

Getter and setter property for the continuous signal independent domain x variable.

Parameters **value** (array_like) – Value to set the continuous signal independent domain x variable with.

Returns Continuous signal independent domain x variable.

Return type ndarray

property range

Getter and setter property for the continuous signal corresponding range y variable.

Parameters **value** (array_like) – Value to set the continuous signal corresponding range y variable with.

Returns Continuous signal corresponding range y variable.

Return type ndarray

property interpolator

Getter and setter property for the continuous signal interpolator type.

Parameters **value** (type) – Value to set the continuous signal interpolator type with.

Returns Continuous signal interpolator type.

Return type type

property interpolator_kwargs

Getter and setter property for the continuous signal interpolator instantiation time arguments.

Parameters **value** (dict) – Value to set the continuous signal interpolator instantiation time arguments to.

Returns Continuous signal interpolator instantiation time arguments.

Return type dict

property extrapolator

Getter and setter property for the continuous signal extrapolator type.

Parameters **value** (type) – Value to set the continuous signal extrapolator type with.

Returns Continuous signal extrapolator type.

Return type type

property extrapolator_kwargs

Getter and setter property for the continuous signal extrapolator instantiation time arguments.

Parameters **value** (dict) – Value to set the continuous signal extrapolator instantiation time arguments to.

Returns Continuous signal extrapolator instantiation time arguments.

Return type dict

property function

Getter property for the continuous signal callable.

Returns Continuous signal callable.

Return type callable

__str__()

Returns a formatted string representation of the continuous signal.

Returns Formatted string representation.

Return type unicode

Examples

```
>>> 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.]]
```

__repr__()

Returns an evaluable string representation of the continuous signal.

Returns Evaluable string representation.

Return type unicode

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> Signal(range_)
Signal([[   0.,   10.],
        [   1.,   20.],
        [   2.,   30.],
        [   3.,   40.],
        [   4.,   50.],
        [   5.,   60.],
        [   6.,   70.],
        [   7.,   80.],
        [   8.,   90.],
        [   9.,  100.]],
       interpolator=KernelInterpolator,
       interpolator_kwargs={},
       extrapolator=Extrapolator,
       extrapolator_kwargs={...})
```

__hash__()

Returns the abstract continuous function hash.

Returns Object hash.

Return type int

__getitem__(x)

Returns the corresponding range y variable for independent domain x variable.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns math:y range value.

Return type numeric or ndarray

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[0]
10.0
>>> signal[np.array([0, 1, 2])]
array([ 10.,  20.,  30.])
>>> signal[0:3]
array([ 10.,  20.,  30.])
>>> signal[np.linspace(0, 5, 5)]
array([ 10.        ,  22.8348902...,  34.8004492...,  47.5535392...,  60.
↪])
```

__setitem__(*x*, *y*)

Sets the corresponding range y variable for independent domain x variable.

Parameters

- **x** (numeric, array_like or slice) – Independent domain x variable.
- **y** (numeric or ndarray) – Corresponding range y variable.

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[0] = 20
>>> signal[0]
20.0
>>> signal[np.array([0, 1, 2])] = 30
>>> signal[np.array([0, 1, 2])]
array([ 30.,  30.,  30.])
>>> signal[0:3] = 40
>>> signal[0:3]
array([ 40.,  40.,  40.])
>>> signal[np.linspace(0, 5, 5)] = 50
>>> print(signal)
[[   0.     50.  ]
 [   1.     40.  ]
 [   1.25   50.  ]
 [   2.     40.  ]
 [   2.5    50.  ]
 [   3.     40.  ]
 [   3.75   50.  ]
 [   4.     50.  ]
 [   5.     50.  ]
 [   6.     70.  ]
 [   7.     80.  ]
 [   8.     90.  ]
 [   9.    100.  ]]
>>> signal[np.array([0, 1, 2])] = np.array([10, 20, 30])
>>> print(signal)
[[   0.     10.  ]
 [   1.     20.  ]
 [   1.25   50.  ]
 [   2.     30.  ]
 [   2.5    50.  ]
 [   3.     40.  ]
 [   3.75   50.  ]
 [   4.     50.  ]
```

(continues on next page)

(continued from previous page)

```
[   5.     50.  ]
[   6.     70.  ]
[   7.     80.  ]
[   8.     90.  ]
[   9.    100.  ]]
```

__contains__(*x*)

Returns whether the continuous signal contains given independent domain x variable.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns Is x domain value contained.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> 0 in signal
True
>>> 0.5 in signal
True
>>> 1000 in signal
False
```

__eq__(*other*)

Returns whether the continuous signal is equal to given other object.

Parameters **other** (object) – Object to test whether it is equal to the continuous signal.

Returns Is given object equal to the continuous signal.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> signal_2 = Signal(range_)
>>> signal_1 == signal_2
True
>>> signal_2[0] = 20
>>> signal_1 == signal_2
False
>>> signal_2[0] = 10
>>> signal_1 == signal_2
True
>>> from colour.algebra import CubicSplineInterpolator
>>> signal_2.interpolator = CubicSplineInterpolator
>>> signal_1 == signal_2
False
```

__ne__(*other*)

Returns whether the continuous signal is not equal to given other object.

Parameters **other** (object) – Object to test whether it is not equal to the continuous signal.

Returns Is given object not equal to the continuous signal.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> signal_2 = Signal(range_)
>>> signal_1 != signal_2
False
>>> signal_2[0] = 20
>>> signal_1 != signal_2
True
>>> signal_2[0] = 10
>>> signal_1 != signal_2
False
>>> from colour.algebra import CubicSplineInterpolator
>>> signal_2.interpolator = CubicSplineInterpolator
>>> signal_1 != signal_2
True
```

arithmetical_operation(*a*, *operation*, *in_place=False*)

Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.

Parameters

- **a** (numeric or ndarray or Signal) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns Continuous signal.

Return type *Signal*

Examples

Adding a single *numeric* variable:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> print(signal_1)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> print(signal_1.arithmetical_operation(10, '+', True))
[[   0.   20.]
```

(continues on next page)

(continued from previous page)

```
[   1.   30.]
[   2.   40.]
[   3.   50.]
[   4.   60.]
[   5.   70.]
[   6.   80.]
[   7.   90.]
[   8.  100.]
[   9.  110.]]
```

Adding an *array_like* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(signal_1.arithmetical_operation(a, '+', True))
[[   0.   30.]
 [   1.   50.]
 [   2.   70.]
 [   3.   90.]
 [   4.  110.]
 [   5.  130.]
 [   6.  150.]
 [   7.  170.]
 [   8.  190.]
 [   9.  210.]]
```

Adding a `colour.continuous.Signal` class:

```
>>> signal_2 = Signal(range_)
>>> print(signal_1.arithmetical_operation(signal_2, '+', True))
[[   0.   40.]
 [   1.   70.]
 [   2.  100.]
 [   3.  130.]
 [   4.  160.]
 [   5.  190.]
 [   6.  220.]
 [   7.  250.]
 [   8.  280.]
 [   9.  310.]]
```

static signal_unpack_data(*data=None*, *domain=None*, *dtype=None*)

Unpack given data for continuous signal instantiation.

Parameters

- **data** (`Series or Signal or array_like or dict_like, optional`) – Data to unpack for continuous signal instantiation.
- **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.
- **dtype** (`type, optional`) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.

Returns Independent domain x variable and corresponding range y variable unpacked for continuous signal instantiation.

Return type tuple

Examples

Unpacking using implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> domain, range_ = Signal.signal_unpack_data(range_)
>>> print(domain)
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> domain, range = Signal.signal_unpack_data(range_, domain)
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *dict*:

```
>>> domain, range_ = Signal.signal_unpack_data(
...     dict(zip(domain, range_)))
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     domain, range = Signal.signal_unpack_data(
...         Series(dict(zip(domain, range_))))
...
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a `colour.continuous.Signal` class:

```
>>> domain, range_ = Signal.signal_unpack_data(
...     Signal(range_, domain))
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

fill_nan(*method='Interpolation'*, *default=0*)

Fill NaNs in independent domain x variable and corresponding range y variable using given method.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with `default`.

- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns NaNs filled continuous signal.

Return type *Signal*

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> signal[3:7] = np.nan
>>> print(signal)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   nan]
 [   4.   nan]
 [   5.   nan]
 [   6.   nan]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> print(signal.fill_nan())
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[3:7] = np.nan
>>> print(signal.fill_nan(method='Constant'))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.    0.]
 [   4.    0.]
 [   5.    0.]
 [   6.    0.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

to_series()

Converts the continuous signal to a *Pandas* `Series` class instance.

Returns Continuous signal as a *Pandas* `Series` class instance.

Return type Series

Examples

```
>>> if is_pandas_installed():
...     range_ = np.linspace(10, 100, 10)
...     signal = Signal(range_)
...     print(signal.to_series())
0.0     10.0
1.0     20.0
2.0     30.0
3.0     40.0
4.0     50.0
5.0     60.0
6.0     70.0
7.0     80.0
8.0     90.0
9.0    100.0
Name: Signal (...), dtype: float64
```

colour.continuous.MultiSignals

class colour.continuous.**MultiSignals**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.continuous.abstract.AbstractContinuousFunction

Defines the base class for multi-continuous signals, a container for multiple colour.continuous.Signal sub-class instances.

Important: Specific documentation about getting, setting, indexing and slicing the multi-continuous signals values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like, optional) – Data to be stored in the multi-continuous signals.
- **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.
- **name** (unicode, optional) – multi-continuous signals 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_kwargs** (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_kwargs** (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.

Attributes

- dtype
- domain
- range
- interpolator
- interpolator_kwargs
- extrapolator
- extrapolator_kwargs
- function
- signals
- labels
- signal_type

Methods

- __init__()
- __str__()
- __repr__()
- __hash__()
- __getitem__()
- __setitem__()
- __contains__()
- __eq__()
- __ne__()
- arithmetical_operation()
- multi_signals_unpack_data()
- fill_nan()
- to_dataframe()

Examples

Instantiation with implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(MultiSignals(range_))
[[  0.   10.]
 [  1.   20.]
 [  2.   30.]
 [  3.   40.]
 [  4.   50.]
```

(continues on next page)

(continued from previous page)

```
 [   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(MultiSignals(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(MultiSignals(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(MultiSignals(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_signals = MultiSignals(range_, domain, signal_type=NotSignal)
>>> print(multi_signals)
[[  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_signals.signals[0])
 <class 'multi_signals.NotSignal'>
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     print(MultiSignals(
...         Series(dict(zip(domain, np.linspace(10, 100, 10))))))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  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(MultiSignals(
...         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])
>>> MultiSignals(range_, domain)[x]
array([ 0.0359701...,  0.2845447...,  0.5331193...])
```

(continues on next page)

(continued from previous page)

```
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignals(range_, domain)[x]
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
>>> MultiSignals(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([ 0.0555274...,  0.3055274...,  0.5555274...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignals(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
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*)

property dtype

Getter and setter property for the continuous signal dtype.

Parameters **value** (type) – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type type

property domain

Getter and setter property for the colour.continuous.Signal sub-class instances independent domain x variable.

Parameters **value** (array_like) – Value to set the colour.continuous.Signal sub-class instances independent domain x variable with.

Returns colour.continuous.Signal sub-class instances independent domain x variable.

Return type ndarray

property range

Getter and setter property for the colour.continuous.Signal sub-class instances corresponding range y variable.

Parameters **value** (array_like) – Value to set the colour.continuous.Signal sub-class instances corresponding range y variable with.

Returns colour.continuous.Signal sub-class instances corresponding range y variable.

Return type ndarray

property interpolator

Getter and setter property for the colour.continuous.Signal sub-class instances interpolator type.

Parameters **value** (type) – Value to set the colour.continuous.Signal sub-class instances interpolator type with.

Returns colour.continuous.Signal sub-class instances interpolator type.

Return type type

property interpolator_kwargs

Getter and setter property for the colour.continuous.Signal sub-class instances interpolator instantiation time arguments.

Parameters **value** (dict) – Value to set the colour.continuous.Signal sub-class instances interpolator instantiation time arguments to.

Returns colour.continuous.Signal sub-class instances interpolator instantiation time arguments.

Return type dict

property extrapolator

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator type.

Parameters **value** (type) – Value to set the colour.continuous.Signal sub-class instances extrapolator type with.

Returns colour.continuous.Signal sub-class instances extrapolator type.

Return type type

property extrapolator_kwargs

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Parameters **value** (dict) – Value to set the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments to.

Returns colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Return type dict

property function

Getter property for the colour.continuous.Signal sub-class instances callable.

Returns colour.continuous.Signal sub-class instances callable.

Return type callable

property signals

Getter and setter property for the colour.continuous.Signal sub-class instances.

Parameters **value** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like) – Attribute value.

Returns colour.continuous.Signal sub-class instances.

Return type OrderedDict

property labels

Getter and setter property for the colour.continuous.Signal sub-class instances name.

Parameters **value** (array_like) – Value to set the colour.continuous.Signal sub-class instances name.

Returns colour.continuous.Signal sub-class instance name.

Return type dict

property signal_type

Getter property for the colour.continuous.Signal sub-class instances type.

Returns colour.continuous.Signal sub-class instances type.

Return type type

__str__()

Returns a formatted string representation of the multi-continuous signals.

Returns Formatted string representation.

Return type unicode

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignals(range_))
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
```

__repr__()

Returns an evaluable string representation of the multi-continuous signals.

Returns Evaluable string representation.

Return type unicode

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> MultiSignals(range_)
MultiSignals([[   0.,   10.,   20.,   30.],
              [   1.,   20.,   30.,   40.],
              [   2.,   30.,   40.,   50.],
              [   3.,   40.,   50.,   60.],
              [   4.,   50.,   60.,   70.],
              [   5.,   60.,   70.,   80.],
              [   6.,   70.,   80.,   90.],
              [   7.,   80.,   90.,  100.],
              [   8.,   90.,  100.,  110.],
              [   9.,  100.,  110.,  120.]],
             labels=[0, 1, 2],
             interpolator=KernelInterpolator,
             interpolator_kwargs={},
             extrapolator=Extrapolator,
             extrapolator_kwargs={...})
```

__hash__()

Returns the abstract continuous function hash.

Returns Object hash.

Return type int

__getitem__(x)

Returns the corresponding range y variable for independent domain x variable.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns math:y range value.

Return type numeric or ndarray

Examples

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals = MultiSignals(range_)
>>> print(multi_signals)
[[  0.   10.   20.   30.]
 [  1.   20.   30.   40.]
 [  2.   30.   40.   50.]
 [  3.   40.   50.   60.]
 [  4.   50.   60.   70.]
 [  5.   60.   70.   80.]
 [  6.   70.   80.   90.]
 [  7.   80.   90.  100.]
 [  8.   90.  100.  110.]
 [  9.  100.  110.  120.]]
>>> multi_signals[0]
array([ 10.,  20.,  30.])
>>> multi_signals[np.array([0, 1, 2])]
array([[ 10.,  20.,  30.],
       [ 20.,  30.,  40.],
       [ 30.,  40.,  50.]])
>>> multi_signals[np.linspace(0, 5, 5)]
array([[ 10.       ...,  20.       ...,  30.       ...],
       [ 22.8348902...,  32.8046056...,  42.774321 ...],
       [ 34.8004492...,  44.7434347...,  54.6864201...],
       [ 47.5535392...,  57.5232546...,  67.4929700...],
       [ 60.       ...,  70.       ...,  80.       ...]])
>>> multi_signals[0:3]
array([[ 10.,  20.,  30.],
       [ 20.,  30.,  40.],
       [ 30.,  40.,  50.]])
>>> multi_signals[:, 0:2]
array([[  10.,   20.],
       [  20.,   30.],
       [  30.,   40.],
       [  40.,   50.],
       [  50.,   60.],
       [  60.,   70.],
       [  70.,   80.],
       [  80.,   90.],
       [  90.,  100.],
       [ 100.,  110.]])
```

__setitem__(*x*, *y*)

Sets the corresponding range y variable for independent domain x variable.

Parameters

- **x** (numeric, array_like or slice) – Independent domain x variable.
- **y** (numeric or ndarray) – Corresponding range y variable.

Examples

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals = MultiSignals(range_)
>>> print(multi_signals)
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
>>> multi_signals[0] = 20
>>> multi_signals[0]
array([ 20.,  20.,  20.])
>>> multi_signals[np.array([0, 1, 2])] = 30
>>> multi_signals[np.array([0, 1, 2])]
array([[ 30.,  30.,  30.],
       [ 30.,  30.,  30.],
       [ 30.,  30.,  30.]])
>>> multi_signals[np.linspace(0, 5, 5)] = 50
>>> print(multi_signals)
[[   0.     50.     50.     50.  ]
 [   1.     30.     30.     30.  ]
 [   1.25   50.     50.     50.  ]
 [   2.     30.     30.     30.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> multi_signals[np.array([0, 1, 2])] = np.array([10, 20, 30])
>>> print(multi_signals)
[[   0.     10.     20.     30.  ]
 [   1.     10.     20.     30.  ]
 [   1.25   50.     50.     50.  ]
 [   2.     10.     20.     30.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
```

(continues on next page)

(continued from previous page)

```
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> y = np.arange(1, 10, 1).reshape(3, 3)
>>> multi_signals[np.array([0, 1, 2])] = y
>>> print(multi_signals)
[[   0.      1.      2.      3.  ]
 [   1.      4.      5.      6.  ]
 [   1.25   50.     50.     50.  ]
 [   2.      7.      8.      9.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     40.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     60.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     70.     80.     90.  ]
 [   7.     80.     90.    100.  ]
 [   8.     90.    100.    110.  ]
 [   9.    100.    110.    120.  ]]
>>> multi_signals[0:3] = 40
>>> multi_signals[0:3]
array([[ 40.,  40.,  40.],
       [ 40.,  40.,  40.],
       [ 40.,  40.,  40.]])
>>> multi_signals[:, 0:2] = 50
>>> print(multi_signals)
[[   0.     50.     50.     40.  ]
 [   1.     50.     50.     40.  ]
 [   1.25   50.     50.     40.  ]
 [   2.     50.     50.      9.  ]
 [   2.5    50.     50.     50.  ]
 [   3.     50.     50.     60.  ]
 [   3.75   50.     50.     50.  ]
 [   4.     50.     50.     70.  ]
 [   5.     50.     50.     50.  ]
 [   6.     50.     50.     90.  ]
 [   7.     50.     50.    100.  ]
 [   8.     50.     50.    110.  ]
 [   9.     50.     50.    120.  ]]
```

__contains__(*x*)

Returns whether the multi-continuous signals contains given independent domain x variable.

Parameters **x** (numeric, array_like or slice) – Independent domain x variable.

Returns Is x domain value contained.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals = MultiSignals(range_)
>>> 0 in multi_signals
True
>>> 0.5 in multi_signals
True
>>> 1000 in multi_signals
False
```

__eq__(*other*)

Returns whether the multi-continuous signals is equal to given other object.

Parameters **other** (object) – Object to test whether it is equal to the multi-continuous signals.

Returns Is given object equal to the multi-continuous signals.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals_1 = MultiSignals(range_)
>>> multi_signals_2 = MultiSignals(range_)
>>> multi_signals_1 == multi_signals_2
True
>>> multi_signals_2[0] = 20
>>> multi_signals_1 == multi_signals_2
False
>>> multi_signals_2[0] = 10
>>> multi_signals_1 == multi_signals_2
True
>>> from colour.algebra import CubicSplineInterpolator
>>> multi_signals_2.interpolator = CubicSplineInterpolator
>>> multi_signals_1 == multi_signals_2
False
```

__ne__(*other*)

Returns whether the multi-continuous signals is not equal to given other object.

Parameters **other** (object) – Object to test whether it is not equal to the multi-continuous signals.

Returns Is given object not equal to the multi-continuous signals.

Return type bool

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> multi_signals_1 = MultiSignals(range_)
>>> multi_signals_2 = MultiSignals(range_)
>>> multi_signals_1 != multi_signals_2
False
>>> multi_signals_2[0] = 20
>>> multi_signals_1 != multi_signals_2
True
>>> multi_signals_2[0] = 10
>>> multi_signals_1 != multi_signals_2
False
>>> from colour.algebra import CubicSplineInterpolator
>>> multi_signals_2.interpolator = CubicSplineInterpolator
>>> multi_signals_1 != multi_signals_2
True
```

arithmetical_operation(*a*, *operation*, *in_place=False*)

Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.

Parameters

- **a** (numeric or ndarray or Signal) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns multi-continuous signals.

Return type *MultiSignals*

Examples

Adding a single *numeric* variable:

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals_1 = MultiSignals(range_)
>>> print(multi_signals_1)
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
>>> print(multi_signals_1.arithmetical_operation(10, '+', True))
[[   0.   20.   30.   40.]
 [   1.   30.   40.   50.]
 [   2.   40.   50.   60.]
 [   3.   50.   60.   70.]
 [   4.   60.   70.   80.]
```

(continues on next page)

(continued from previous page)

```
 [   5.   70.   80.   90.]
 [   6.   80.   90.  100.]
 [   7.   90.  100.  110.]
 [   8.  100.  110.  120.]
 [   9.  110.  120.  130.]]
```

Adding an *array_like* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   30.   40.   50.]
 [   1.   50.   60.   70.]
 [   2.   70.   80.   90.]
 [   3.   90.  100.  110.]
 [   4.  110.  120.  130.]
 [   5.  130.  140.  150.]
 [   6.  150.  160.  170.]
 [   7.  170.  180.  190.]
 [   8.  190.  200.  210.]
 [   9.  210.  220.  230.]]
```

```
>>> a = np.array([[10, 20, 30]])
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   40.   60.   80.]
 [   1.   60.   80.  100.]
 [   2.   80.  100.  120.]
 [   3.  100.  120.  140.]
 [   4.  120.  140.  160.]
 [   5.  140.  160.  180.]
 [   6.  160.  180.  200.]
 [   7.  180.  200.  220.]
 [   8.  200.  220.  240.]
 [   9.  220.  240.  260.]]
```

```
>>> a = np.arange(0, 30, 1).reshape([10, 3])
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   40.   61.   82.]
 [   1.   63.   84.  105.]
 [   2.   86.  107.  128.]
 [   3.  109.  130.  151.]
 [   4.  132.  153.  174.]
 [   5.  155.  176.  197.]
 [   6.  178.  199.  220.]
 [   7.  201.  222.  243.]
 [   8.  224.  245.  266.]
 [   9.  247.  268.  289.]]
```

Adding a `colour.continuous.Signal` sub-class:

```
>>> multi_signals_2 = MultiSignals(range_)
>>> print(multi_signals_1.arithmetical_operation(
...     multi_signals_2, '+', True))
[[   0.   50.   81.  112.]
 [   1.   83.  114.  145.]
 [   2.  116.  147.  178.]
 [   3.  149.  180.  211.]
```

(continues on next page)

(continued from previous page)

```
[   4.  182.  213.  244.]
[   5.  215.  246.  277.]
[   6.  248.  279.  310.]
[   7.  281.  312.  343.]
[   8.  314.  345.  376.]
[   9.  347.  378.  409.]]
```

static multi_signals_unpack_data(*data=None*, *domain=None*, *labels=None*, *dtype=None*, *signal_type=<class 'colour.continuous.signal.Signal'>*, ***kwargs*)

Unpack given data for multi-continuous signals instantiation.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like, optional) – Data to unpack for multi-continuous signals instantiation.
- **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.
- **dtype** (type, optional) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.
- **signal_type** (type, optional) – A colour.continuous.Signal sub-class type.
- **name** (unicode, optional) – multi-continuous signals name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.continuous.Signal sub-class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.continuous.Signal sub-class instances.

Returns Mapping of labeled colour.continuous.Signal sub-class instances.

Return type dict

Examples

Unpacking using implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signals = MultiSignals.multi_signals_unpack_data(range_)
>>> list(signals.keys())
[0]
>>> print(signals[0])
[[  0.   10.]
 [  1.   20.]
 [  2.   30.]
 [  3.   40.]
```

(continues on next page)

(continued from previous page)

```
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Unpacking using explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
[0]
>>> print(signals[0])
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using a *dict*:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_)))
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
```

(continues on next page)

(continued from previous page)

```
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using *MultiSignals.multi_signals_unpack_data* method output:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_)))
>>> signals = MultiSignals.multi_signals_unpack_data(signals)
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     signals = MultiSignals.multi_signals_unpack_data(
...         Series(dict(zip(domain, np.linspace(10, 100, 10)))))
...     print(signals[0])
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using a *Pandas DataFrame*:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     data = dict(zip(['a', 'b', 'c'], tsplit(range_)))
...     signals = MultiSignals.multi_signals_unpack_data(
...         DataFrame(data, domain))
...     print(signals['c'])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
```

(continues on next page)

(continued from previous page)

```
[  800.   100.]
[  900.   110.]
[ 1000.   120.]]
```

fill_nan(*method='Interpolation'*, *default=0*)

Fill NaNs in independent domain x variable and corresponding range y variable using given method.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns

- *Signal* – NaNs filled multi-continuous signals.
- *>>> domain = np.arange(0, 10, 1)*
- *>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)*
- *>>> range_ += np.array([0, 10, 20])*
- *>>> multi_signals = MultiSignals(range_)*
- **>>> multi_signals[3** *(7] = np.nan*)
- *>>> print(multi_signals)*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. nan nan nan] [4. nan nan nan] [5. nan nan nan] [6. nan nan nan] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- *>>> print(multi_signals.fill_nan())*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 40. 50. 60.] [4. 50. 60. 70.] [5. 60. 70. 80.] [6. 70. 80. 90.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- **>>> multi_signals[3** *(7] = np.nan*)
- *>>> print(multi_signals.fill_nan(method='Constant'))*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 0. 0. 0.] [4. 0. 0. 0.] [5. 0. 0. 0.] [6. 0. 0. 0.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]

to_dataframe()

Converts the continuous signal to a *Pandas* DataFrame class instance.

Returns Continuous signal as a *Pandas* DataFrame class instance.

Return type DataFrame

Examples

```
>>> if is_pandas_installed():
...     domain = np.arange(0, 10, 1)
...     range_ = tstack([np.linspace(10, 100, 10)] * 3)
...     range_ += np.array([0, 10, 20])
...     multi_signals = MultiSignals(range_)
...     print(multi_signals.to_dataframe())
          0      1      2
0.0   10.0   20.0   30.0
1.0   20.0   30.0   40.0
2.0   30.0   40.0   50.0
3.0   40.0   50.0   60.0
4.0   50.0   60.0   70.0
5.0   60.0   70.0   80.0
6.0   70.0   80.0   90.0
7.0   80.0   90.0  100.0
8.0   90.0  100.0  110.0
9.0  100.0  110.0  120.0
```

Corresponding Chromaticities

Prediction

colour

corresponding_chromaticities_prediction([...])	Returns the corresponding chromaticities prediction for given chromatic adaptation model.
CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS	Aggregated corresponding chromaticities prediction models.
CorrespondingColourDataset(name, XYZ_r, ...)	Defines a corresponding colour dataset.
CorrespondingChromaticitiesPrediction(name, ...)	Defines a chromatic adaptation model prediction.

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 or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **model** (unicode, optional) – **{'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}**, Chromatic adaptation model.
- **transform** (unicode, optional) – {colour.corresponding.corresponding_chromaticities_prediction_VonKries()}, **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[], [], [], [], [], [], []

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction(2, 'CMCCAT2000')
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[((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': ..., 'Fairchild 1990': ..., 'Von Kries': ..., 'vonkries': ...})**

Aggregated corresponding chromaticities prediction models.

References

[], [], [], [], [], [], []

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

Aliases:

- 'vonkries': 'Von Kries'

colour.CorrespondingColourDataset

class colour.**CorrespondingColourDataset**(*name*, *XYZ_r*, *XYZ_t*, *XYZ_cr*, *XYZ_ct*, *Y_r*, *Y_t*, *B_r*, *B_t*, *metadata*)

Defines a corresponding colour dataset.

Parameters

- **name** (unicode) – Corresponding colour dataset name.
- **XYZ_r** (array_like) – *CIE XYZ* tristimulus values of the reference illuminant.
- **XYZ_t** (array_like) – *CIE XYZ* tristimulus values of the test illuminant.
- **XYZ_cr** (array_like) – Corresponding *CIE XYZ* tristimulus values under the reference illuminant.
- **XYZ_ct** (array_like) – Corresponding *CIE XYZ* tristimulus values under the test illuminant.
- **Y_r** (numeric) – Reference white luminance Y_r in cd/m^2.
- **Y_t** (numeric) – Test white luminance Y_t in cd/m^2.
- **B_r** (numeric) – Luminance factor B_r of reference achromatic background as percentage.
- **B_t** (numeric) – Luminance factor B_t of test achromatic background as percentage.
- **metadata** (dict) – Dataset metadata.

Notes

- This class is compatible with *Luo and Rhodes (1999) Corresponding-Colour Datasets* datasets.

References

[]

Create new instance of CorrespondingColourDataset(name, XYZ_r, XYZ_t, XYZ_cr, XYZ_ct, Y_r, Y_t, B_r, B_t, metadata)

__init__()

Methods

`__init__`()

`count`(value, /)	Return number of occurrences of value.
`index`(value[, start, stop])	Return first index of value.

Attributes

`B_r`	Alias for field number 7
`B_t`	Alias for field number 8
`XYZ_cr`	Alias for field number 3
`XYZ_ct`	Alias for field number 4
`XYZ_r`	Alias for field number 1
`XYZ_t`	Alias for field number 2
`Y_r`	Alias for field number 5
`Y_t`	Alias for field number 6
`metadata`	Alias for field number 9
`name`	Alias for field number 0

colour.CorrespondingChromaticitiesPrediction

class colour.**CorrespondingChromaticitiesPrediction**(*name*, *uv_t*, *uv_m*, *uv_p*)

Defines a chromatic adaptation model prediction.

Parameters

- **name** (unicode) – Test colour name.
- **uv_t** (array_like, (2,)) – Chromaticity coordinates uv_t^p of test colour.
- **uv_m** (array_like, (2,)) – Chromaticity coordinates uv_m^p of matching colour.
- **uv_p** (array_like, (2,)) – Chromaticity coordinates uv_p^p of predicted colour.

Create new instance of CorrespondingChromaticitiesPrediction(name, uv_t, uv_m, uv_p)

__init__()

Methods

`__init__`()

`count`(value, /)	Return number of occurrences of value.
`index`(value[, start, stop])	Return first index of value.

Attributes

name	Alias for field number 0
uv_m	Alias for field number 2
uv_p	Alias for field number 3
uv_t	Alias for field number 1

Dataset

colour

BRENEMAN_EXPERIMENTS	*Breneman (1987)* experiments.
BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES	*Breneman (1987)* experiments primaries chromaticities.

colour.BRENEMAN_EXPERIMENTS

```
colour.BRENEMAN_EXPERIMENTS = {1: (BrenemanExperimentResult(name='Illuminant',
uv_t=array([ 0.259, 0.526]), uv_m=array([ 0.2 , 0.475]), s_uv=array(None, dtype=object),
d_uv_i=array(None, dtype=object), d_uv_g=array(None, dtype=object)),
BrenemanExperimentResult(name='Gray', uv_t=array([ 0.259, 0.524]), uv_m=array([ 0.199,
0.487]), s_uv=array([4, 4]), d_uv_i=array([2, 3]), d_uv_g=array([0, 0])),
BrenemanExperimentResult(name='Red', uv_t=array([ 0.459, 0.522]), uv_m=array([ 0.42 ,
0.509]), s_uv=array([19, 4]), d_uv_i=array([-10, -7]), d_uv_g=array([-19, -3])),
BrenemanExperimentResult(name='Skin', uv_t=array([ 0.307, 0.526]), uv_m=array([ 0.249,
0.497]), s_uv=array([7, 4]), d_uv_i=array([-1, 1]), d_uv_g=array([-6, -1])),
BrenemanExperimentResult(name='Orange', uv_t=array([ 0.36 , 0.544]), uv_m=array([ 0.302,
0.548]), s_uv=array([12, 1]), d_uv_i=array([ 1, -2]), d_uv_g=array([-7, -6])),
BrenemanExperimentResult(name='Brown', uv_t=array([ 0.35 , 0.541]), uv_m=array([ 0.29 ,
0.537]), s_uv=array([11, 4]), d_uv_i=array([3, 0]), d_uv_g=array([-5, -3])),
BrenemanExperimentResult(name='Yellow', uv_t=array([ 0.318, 0.55 ]), uv_m=array([ 0.257,
0.554]), s_uv=array([8, 2]), d_uv_i=array([0, 2]), d_uv_g=array([-5, -5])),
BrenemanExperimentResult(name='Foliage', uv_t=array([ 0.258, 0.542]), uv_m=array([ 0.192,
0.529]), s_uv=array([4, 6]), d_uv_i=array([3, 2]), d_uv_g=array([ 3, -6])),
BrenemanExperimentResult(name='Green', uv_t=array([ 0.193, 0.542]), uv_m=array([ 0.129,
0.521]), s_uv=array([7, 5]), d_uv_i=array([3, 2]), d_uv_g=array([ 9, -7])),
BrenemanExperimentResult(name='Blue-green', uv_t=array([ 0.18 , 0.516]), uv_m=array([
0.133, 0.469]), s_uv=array([4, 6]), d_uv_i=array([-3, -2]), d_uv_g=array([ 2, -5])),
BrenemanExperimentResult(name='Blue', uv_t=array([ 0.186, 0.445]), uv_m=array([ 0.158,
0.34 ]), s_uv=array([13, 33]), d_uv_i=array([2, 7]), d_uv_g=array([ 1, 13])),
BrenemanExperimentResult(name='Sky', uv_t=array([ 0.226, 0.491]), uv_m=array([ 0.178,
0.426]), s_uv=array([ 3, 14]), d_uv_i=array([ 1, -3]), d_uv_g=array([ 0, -1])),
BrenemanExperimentResult(name='Purple', uv_t=array([ 0.278, 0.456]), uv_m=array([ 0.231,
0.365]), s_uv=array([ 4, 25]), d_uv_i=array([0, 2]), d_uv_g=array([-5, 7]))), 2:
(BrenemanExperimentResult(name='Illuminant', uv_t=array([ 0.222, 0.521]), uv_m=array([
0.204, 0.479]), s_uv=array(None, dtype=object), d_uv_i=array(None, dtype=object),
d_uv_g=array(None, dtype=object)), BrenemanExperimentResult(name='Gray', uv_t=array([
0.227, 0.517]), uv_m=array([ 0.207, 0.486]), s_uv=array([2, 5]), d_uv_i=array([-1, 0]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Red', uv_t=array([ 0.464, 0.52 ]),
uv_m=array([ 0.449, 0.511]), s_uv=array([22, 3]), d_uv_i=array([-8, -8]),
d_uv_g=array([-7, -2])), BrenemanExperimentResult(name='Skin', uv_t=array([ 0.286,
0.526]), uv_m=array([ 0.263, 0.505]), s_uv=array([7, 2]), d_uv_i=array([ 0, -1]),
d_uv_g=array([ 0, -1])), BrenemanExperimentResult(name='Orange', uv_t=array([ 0.348,
0.546]), uv_m=array([ 0.322, 0.545]), s_uv=array([13, 3]), d_uv_i=array([ 3, -1]),
d_uv_g=array([ 3, -2])), BrenemanExperimentResult(name='Brown', uv_t=array([ 0.34 ,
0.543]), uv_m=array([ 0.316, 0.537]), s_uv=array([11, 3]), d_uv_i=array([1, 1]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Yellow', uv_t=array([ 0.288,
0.554]), uv_m=array([ 0.265, 0.553]), s_uv=array([5, 2]), d_uv_i=array([-2, 2]),
d_uv_g=array([-1, -2])), BrenemanExperimentResult(name='Foliage', uv_t=array([ 0.244,
0.547]), uv_m=array([ 0.221, 0.538]), s_uv=array([4, 3]), d_uv_i=array([-2, 1]),
d_uv_g=array([ 0, -3])), BrenemanExperimentResult(name='Green', uv_t=array([ 0.156,
0.548]), uv_m=array([ 0.135, 0.532]), s_uv=array([4, 3]), d_uv_i=array([-1, 3]),
d_uv_g=array([ 3, -4])), BrenemanExperimentResult(name='Blue-green', uv_t=array([ 0.159,
0.511]), uv_m=array([ 0.145, 0.472]), s_uv=array([9, 7]), d_uv_i=array([-1, 2]),
d_uv_g=array([2, 1])), BrenemanExperimentResult(name='Blue', uv_t=array([ 0.16 , 0.406]),
uv_m=array([ 0.163, 0.331]), s_uv=array([23, 31]), d_uv_i=array([ 2, -3]),
d_uv_g=array([-1, 3])), BrenemanExperimentResult(name='Sky', uv_t=array([ 0.19 , 0.481]),
uv_m=array([ 0.176, 0.431]), s_uv=array([ 5, 24]), d_uv_i=array([ 2, -2]), d_uv_g=array([2,
0])), BrenemanExperimentResult(name='Purple', uv_t=array([ 0.258, 0.431]), uv_m=array([
0.244, 0.349]), s_uv=array([ 4, 19]), d_uv_i=array([-3, 13]), d_uv_g=array([-4, 19]))), 3:
(BrenemanExperimentResult(name='Illuminant', uv_t=array([ 0.223, 0.521]), uv_m=array([
0.206, 0.478]), s_uv=array(None, dtype=object), d_uv_i=array(None, dtype=object),
d_uv_g=array(None, dtype=object)), BrenemanExperimentResult(name='Gray', uv_t=array([
0.228, 0.517]), uv_m=array([ 0.211, 0.494]), s_uv=array([1, 3]), d_uv_i=array([0, 2]),
d_uv_g=array([0, 0])), BrenemanExperimentResult(name='Red', uv_t=array([ 0.462, 0.519]),
uv_m=array([ 0.448, 0.505]), s_uv=array([11, 4]), d_uv_i=array([-3, 6]), d_uv_g=array([-4,
6])), BrenemanExperimentResult(name='Skin', uv_t=array([ 0.285, 0.524]), uv_m=array([
0.267, 0.507]), s_uv=array([6, 3]), d_uv_i=array([-1, 1]), d_uv_g=array([-5, 0])),
BrenemanExperimentResult(name='Orange', uv_t=array([ 0.346, 0.546]), uv_m=array([ 0.325,
0.541]), s_uv=array([11, 3]), d_uv_i=array([ 1, -2]), d_uv_g=array([2, 3])),
BrenemanExperimentResult(name='Brown', uv_t=array([ 0.338, 0.543]), uv_m=array([ 0.321,
```

Breneman (1987) experiments.

References

[]

BRENEMAN_EXPERIMENTS : dict

colour.BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES

colour.**BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES = {1: PrimariesChromaticityCoordinates(experiment=1, illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(1500), P_uvp=array([0.671, 0.519]), D_uvp=array([-0.586, 0.627]), T_uvp=array([0.253, 0.016])), 2: PrimariesChromaticityCoordinates(experiment=2, illuminants=array(['Projector', 'D55'], dtype='<U9'), Y=array(1500), P_uvp=array([0.675, 0.523]), D_uvp=array([-0.466, 0.617]), T_uvp=array([0.255, 0.018])), 3: PrimariesChromaticityCoordinates(experiment=3, illuminants=array(['Projector', 'D55'], dtype='<U9'), Y=array(75), P_uvp=array([0.664, 0.51]), D_uvp=array([-0.256, 0.729]), T_uvp=array([0.244, 0.003])), 4: PrimariesChromaticityCoordinates(experiment=4, illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(75), P_uvp=array([0.674, 0.524]), D_uvp=array([-0.172, 0.628]), T_uvp=array([0.218, -0.026])), 6: PrimariesChromaticityCoordinates(experiment=6, illuminants=array(['A', 'D55'], dtype='<U3'), Y=array(11100), P_uvp=array([0.659, 0.506]), D_uvp=array([-0.141, 0.615]), T_uvp=array([0.249, 0.009])), 8: PrimariesChromaticityCoordinates(experiment=8, illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(350), P_uvp=array([0.659, 0.505]), D_uvp=array([-0.246, 0.672]), T_uvp=array([0.235, -0.006])), 9: PrimariesChromaticityCoordinates(experiment=9, illuminants=array(['A', 'D65'], dtype='<U3'), Y=array(15), P_uvp=array([0.693, 0.546]), D_uvp=array([-0.446, 0.773]), T_uvp=array([0.221, -0.023])), 11: PrimariesChromaticityCoordinates(experiment=11, illuminants=array(['D55', 'green'], dtype='<U5'), Y=array(1560), P_uvp=array([0.68 , 0.529]), D_uvp=array([0.018, 0.576]), T_uvp=array([0.307, 0.08])), 12: PrimariesChromaticityCoordinates(experiment=12, illuminants=array(['D55', 'green'], dtype='<U5'), Y=array(75), P_uvp=array([0.661, 0.505]), D_uvp=array([0.039, 0.598]), T_uvp=array([0.345, 0.127]))}**

Breneman (1987) experiments primaries chromaticities.

References

[]

BRENEMAN_EXPERIMENT_PRIMARIES_CHROMATICITIES : dict

Fairchild (1990)

colour.corresponding

corresponding_chromaticities_prediction_Fairchild1990([1])	Returns the corresponding chromaticities prediction 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 or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[], [], []

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_Fairchild1990(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2089528...,  0.4724034...])),
 (array([ 0.449,  0.511]), array([ 0.4375652...,  0.5121030...])),
 (array([ 0.263,  0.505]), array([ 0.2621362...,  0.4972538...])),
 (array([ 0.322,  0.545]), array([ 0.3235312...,  0.5475665...])),
 (array([ 0.316,  0.537]), array([ 0.3151391...,  0.5398333...])),
 (array([ 0.265,  0.553]), array([ 0.2634745...,  0.5544335...])),
 (array([ 0.221,  0.538]), array([ 0.2211595...,  0.5324470...])),
 (array([ 0.135,  0.532]), array([ 0.1396949...,  0.5207234...])),
 (array([ 0.145,  0.472]), array([ 0.1512288...,  0.4533041...])),
 (array([ 0.163,  0.331]), array([ 0.1715691...,  0.3026264...])),
 (array([ 0.176,  0.431]), array([ 0.1825792...,  0.4077892...])),
 (array([ 0.244,  0.349]), array([ 0.2418905...,  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 or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance. Returns
- ------- _

- **tuple** – Corresponding chromaticities prediction.

References

[], []

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CIE1994(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2273130...,  0.5267609...])),
 (array([ 0.449,  0.511]), array([ 0.4612181...,  0.5191849...])),
 (array([ 0.263,  0.505]), array([ 0.2872404...,  0.5306938...])),
 (array([ 0.322,  0.545]), array([ 0.3489822...,  0.5454398...])),
 (array([ 0.316,  0.537]), array([ 0.3371612...,  0.5421567...])),
 (array([ 0.265,  0.553]), array([ 0.2889416...,  0.5534074...])),
 (array([ 0.221,  0.538]), array([ 0.2412195...,  0.5464301...])),
 (array([ 0.135,  0.532]), array([ 0.1530344...,  0.5488239...])),
 (array([ 0.145,  0.472]), array([ 0.1568709...,  0.5258835...])),
 (array([ 0.163,  0.331]), array([ 0.1499762...,  0.4401747...])),
 (array([ 0.176,  0.431]), array([ 0.1876711...,  0.5039627...])),
 (array([ 0.244,  0.349]), array([ 0.2560012...,  0.4546263...]))]
```

CMCCAT2000

colour.corresponding

corresponding_chromaticities_prediction_CMCCAT2000(...)	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 or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[], [], []

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CMCCAT2000(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2083210...,  0.4727168...])),
 (array([ 0.449,  0.511]), array([ 0.4459270...,  0.5077735...])),
 (array([ 0.263,  0.505]), array([ 0.2640262...,  0.4955361...])),
 (array([ 0.322,  0.545]), array([ 0.3316884...,  0.5431580...])),
 (array([ 0.316,  0.537]), array([ 0.3222624...,  0.5357624...])),
 (array([ 0.265,  0.553]), array([ 0.2710705...,  0.5501997...])),
 (array([ 0.221,  0.538]), array([ 0.2261826...,  0.5294740...])),
 (array([ 0.135,  0.532]), array([ 0.1439693...,  0.5190984...])),
 (array([ 0.145,  0.472]), array([ 0.1494835...,  0.4556760...])),
 (array([ 0.163,  0.331]), array([ 0.1563172...,  0.3164151...])),
 (array([ 0.176,  0.431]), array([ 0.1763199...,  0.4127589...])),
 (array([ 0.244,  0.349]), array([ 0.2287638...,  0.3499324...]))]
```

Von Kries

colour.corresponding

corresponding_chromaticities_prediction_VonKries([1, ...])	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 or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[], []

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_VonKries(2, 'Bradford')
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2082014...,  0.4722922...])),
 (array([ 0.449,  0.511]), array([ 0.4489102...,  0.5071602...])),
 (array([ 0.263,  0.505]), array([ 0.2643545...,  0.4959631...])),
 (array([ 0.322,  0.545]), array([ 0.3348730...,  0.5471220...])),
 (array([ 0.316,  0.537]), array([ 0.3248758...,  0.5390589...])),
 (array([ 0.265,  0.553]), array([ 0.2733105...,  0.5555028...])),
 (array([ 0.221,  0.538]), array([ 0.227148 ...,  0.5331318...])),
 (array([ 0.135,  0.532]), array([ 0.1442730...,  0.5226804...])),
 (array([ 0.145,  0.472]), array([ 0.1498745...,  0.4550785...])),
 (array([ 0.163,  0.331]), array([ 0.1564975...,  0.3148796...])),
 (array([ 0.176,  0.431]), array([ 0.1760593...,  0.4103772...])),
 (array([ 0.244,  0.349]), array([ 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

delta_E(a, b[, method])	Returns the difference ΔE_{ab} between two given *CIE L*a*b** or $J'a'b'$ colourspace arrays using given method.
DELTA_E_METHODS	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 $J'a'b'$ colourspace arrays using given method.

Parameters

- **a** (array_like) – *CIE L*a*b** or $J'a'b'$ colourspace array a.
- **b** (array_like) – *CIE L*a*b** or $J'a'b'$ colourspace 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.
- **textiles** (bool, optional) – {colour.difference.delta_E_CIE1994(), colour.difference.delta_E_CIE2000(), colour.difference.delta_E_DIN99()}, Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$, $k_1 = 0.048$, $k_2 = 0.014$, $k_E = 2$, $k_CH = 0.5$ weights are used instead of $k_L = k_C = k_H = 1$, $k_1 = 0.045$, $k_2 = 0.015$, $k_E = k_CH = 1.0$.
- **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

[], [], [], [], [], [], [], [], []

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)
94.0356490...
>>> delta_E(a, b, method='CIE 2000')
94.0356490...
>>> delta_E(a, b, method='CIE 1976')
451.7133019...
>>> delta_E(a, b, method='CIE 1994')
83.7792255...
>>> delta_E(a, b, method='CIE 1994', textiles=False)
...
83.7792255...
>>> delta_E(a, b, method='DIN99')
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')
0.0001034...
>>> delta_E(a, b, method='CAM16-LCD')
0.0001034...
```

colour.DELTA_E_METHODS

colour.**DELTA_E_METHODS = CaseInsensitiveMapping({'CIE 1976': ..., 'CIE 1994': ..., 'CIE 2000': ..., 'CMC': ..., 'CAM02-LCD': ..., 'CAM02-SCD': ..., 'CAM02-UCS': ..., 'CAM16-LCD': ..., 'CAM16-SCD': ..., 'CAM16-UCS': ..., 'DIN99': ..., 'cie1976': ..., 'cie1994': ..., 'cie2000': ...})**

Supported ΔE_{ab} computation methods.

References

[], [], [], [], [], [], [], [], []

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

JND_CIE1976	Just Noticeable Difference (JND) according to *CIE 1976* colour difference formula, i.e. Euclidean distance in *CIE L*a*b** colourspace.
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.JND_CIE1976

colour.difference.**JND_CIE1976 = 2.3**

Just Noticeable Difference (JND) according to *CIE 1976* colour difference formula, i.e. Euclidean distance in *CIE L*a*b** colourspace.

Notes

A standard observer sees the difference in colour as follows:

- $0 < \Delta E^*_{ab} < 1$: Observer does not notice the difference.
- $1 < \Delta E^*_{ab} < 2$: Only experienced observer can notice the difference.
- $2 < \Delta E^*_{ab} <$ 3:5 : Unexperienced observer also notices the difference.
- 3:5 $< \Delta E^*_{ab} < 5$: Clear difference in colour is noticed.
- $5 < \Delta E^*_{ab}$: Observer notices two different colours.

References

[]

JND_CIE1976 : numeric

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

Domain	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

[]

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)
451.7133019...
```

CIE 1994

colour.difference

delta_E_CIE1994(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{94} between two given *CIE L*a*b** colourspace arrays using *CIE 1994* 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** colourspace arrays using *CIE 1994* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace 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

Domain	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

[]

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)
83.7792255...
>>> delta_E_CIE1994(Lab_1, Lab_2, textiles=True)
88.3355530...
```

CIE 2000

colour.difference

delta_E_CIE2000(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{00} between two given *CIE L*a*b** colourspace 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** colourspace arrays using *CIE 2000* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace 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

Domain	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]

- 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

[], []

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)
94.0356490...
>>> Lab_2 = np.array([50.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2)
100.8779470...
>>> delta_E_CIE2000(Lab_1, Lab_2, textiles=True)
95.7920535...
```

CMC

colour.difference

delta_E_CMC(Lab_1, Lab_2[, l, c])	Returns the difference ΔE_{CMC} between two given *CIE L*a*b** colourspace arrays using *Colour Measurement Committee* 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

Domain	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

[]

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)
172.7047712...
```

Luo, Cui and Li (2006)

colour.difference

delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ arrays.
delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-UCS* 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 *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Luo et al. (2006) CAM02-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Luo et al. (2006) CAM02-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

[]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)
14.0555464...
```

colour.difference.delta_E_CAM02SCD

colour.difference.**delta_E_CAM02SCD**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Luo et al. (2006) CAM02-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Luo et al. (2006) CAM02-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

[]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)
14.0551718...
```

colour.difference.delta_E_CAM02UCS

colour.difference.**delta_E_CAM02UCS**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Luo et al. (2006) CAM02-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Luo et al. (2006) CAM02-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

[]

Examples

```
>>> Jpapbp_1 = np.array([54.90433134, -0.08450395, -0.06854831])
>>> Jpapbp_2 = np.array([54.80352754, -3.96940084, -13.57591013])
>>> delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)
14.0552982...
```

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

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.
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.
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.

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

[]

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

[]

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

[]

DIN99

colour.difference

delta_E_DIN99(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* 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

Domain	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

[]

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)
1.1772166...
```

Geometry Primitives Generation

- *Primitives*
- *Primitive Vertices*

Primitives

colour

PRIMITIVE_METHODS	Supported geometry primitive generation methods.
primitive([method])	Returns a geometry primitive using given method.

colour.PRIMITIVE_METHODS

colour.**PRIMITIVE_METHODS = CaseInsensitiveMapping({'Grid': ..., 'Cube': ...})**

Supported geometry primitive generation methods.

PRIMITIVE_METHODS [CaseInsensitiveMapping] **{'Grid', 'Cube'}**

colour.primitive

colour.**primitive**(*method='Cube'*, ***kwargs*)

Returns a geometry primitive using given method.

Parameters

- **method** (unicode, optional) – **{'Cube', 'Grid'}**, Generation method.
- **width** (numeric, optional) – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive width.
- **height** (numeric, optional) – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive height.
- **depth** (numeric, optional) – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive depth.
- **width_segments** – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive segments count along the width.

- **height_segments** – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive segments count along the height.
- **depth_segments** – {colour.geometry.primitive_grid_mpl(), colour.geometry.primitive_cube_mpl()}, Primitive segments count along the depth.
- **planes** (array_like, optional) – {colour.geometry.primitive_cube_mpl()}, **{'-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}**, Included grid primitives in the cube construction.

References

[]

Examples

```
>>> vertices, faces, outline = primitive()
>>> print(vertices)
[([-0.5,  0.5, -0.5], [ 0.,  1.], [-0., -0., -1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  1.], [-0., -0., -1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -0., -1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5, -0.5], [ 1.,  0.], [-0., -0., -1.], [ 1.,  0.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  1.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  1.], [-0., -1., -0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  1.], [-0., -1., -0.], [ 1.,  0.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -1., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5, -0.5,  0.5], [ 1.,  0.], [-0., -1., -0.], [ 0.,  0.,  1.,  1.])
 ([ 0.5,  0.5, -0.5], [ 0.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  1.,  1.])
 ([-0.5,  0.5, -0.5], [ 0.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  1.], [-1., -0., -0.], [ 0.,  0.,  1.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  1.], [-1., -0., -0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-1., -0., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5,  0.5, -0.5], [ 1.,  0.], [-1., -0., -0.], [ 0.,  1.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 0.,  1.], [ 1.,  0.,  0.], [ 1.,  0.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 1.,  0.,  0.], [ 1.,  1.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  0.], [ 1.,  0.,  0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  0.], [ 1.,  0.,  0.], [ 1.,  1.,  0.,  1.])]
>>> print(faces)
[[ 1  2  0]
 [ 1  3  2]
 [ 4  6  5]
 [ 6  7  5]
 [ 9 10  8]
 [ 9 11 10]
 [12 14 13]
 [14 15 13]
 [17 18 16]
 [17 19 18]
 [20 22 21]
 [22 23 21]]
>>> print(outline)
[[ 0  2]
```

(continues on next page)

(continued from previous page)

```
 [ 2  3]
 [ 3  1]
 [ 1  0]
 [ 4  6]
 [ 6  7]
 [ 7  5]
 [ 5  4]
 [ 8 10]
 [10 11]
 [11  9]
 [ 9  8]
 [12 14]
 [14 15]
 [15 13]
 [13 12]
 [16 18]
 [18 19]
 [19 17]
 [17 16]
 [20 22]
 [22 23]
 [23 21]
 [21 20]]
>>> vertices, faces, outline = primitive('Grid')
>>> print(vertices)
[([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])]
>>> print(faces)
[[0 2 1]
 [2 3 1]]
>>> print(outline)
[[0 2]
 [2 3]
 [3 1]
 [1 0]]
```

Ancillary Objects

`colour.geometry`

PLANE_TO_AXIS_MAPPING	Plane to axis mapping.
primitive_grid([width, height, ...])	Generates vertices and indices for a filled and outlined grid primitive.
primitive_cube([width, height, depth, ...])	Generates vertices and indices for a filled and outlined cube primitive.

colour.geometry.PLANE_TO_AXIS_MAPPING

colour.geometry.**PLANE_TO_AXIS_MAPPING = CaseInsensitiveMapping({'yz': ..., 'zy': ..., 'xz': ..., 'zx': ..., 'xy': ..., 'yx': ...})**

Plane to axis mapping.

PLANE_TO_AXIS_MAPPING [CaseInsensitiveMapping] **{'-x', '+x', '-y', '+y', '-z', '+z'}**

colour.geometry.primitive_grid

colour.geometry.**primitive_grid**(*width=1*, *height=1*, *width_segments=1*, *height_segments=1*, *axis='+z'*)

Generates vertices and indices for a filled and outlined grid primitive.

Parameters

- **width** (float, optional) – Grid width.
- **height** (float, optional) – Grid height.
- **width_segments** (int, optional) – Grid segments count along the width.
- **height_segments** (float, optional) – Grid segments count along the height.
- **axis** (unicode, optional) – **{'+z', '-x', '+x', '-y', '+y', '-z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}**, Axis the primitive will be normal to, or plane the primitive will be co-planar with.

Returns Tuple of grid vertices, face indices to produce a filled grid and outline indices to produce an outline of the faces of the grid.

Return type tuple

References

[]

Examples

```
>>> vertices, faces, outline = primitive_grid()
>>> print(vertices)
[([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])]
>>> print(faces)
[[0 2 1]
 [2 3 1]]
>>> print(outline)
[[0 2]
 [2 3]
 [3 1]
 [1 0]]
```

colour.geometry.primitive_cube

colour.geometry.**primitive_cube**(*width=1*, *height=1*, *depth=1*, *width_segments=1*, *height_segments=1*, *depth_segments=1*, *planes=None*)

Generates vertices and indices for a filled and outlined cube primitive.

Parameters

- **width** (float, optional) – Cube width.
- **height** (float, optional) – Cube height.
- **depth** (float, optional) – Cube depth.
- **width_segments** (int, optional) – Cube segments count along the width.
- **height_segments** (float, optional) – Cube segments count along the height.
- **depth_segments** (float, optional) – Cube segments count along the depth.
- **planes** (array_like, optional) – **{'-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}**, Grid primitives to include in the cube construction.

Returns Tuple of cube vertices, face indices to produce a filled cube and outline indices to produce an outline of the faces of the cube.

Return type tuple

Examples

```
>>> vertices, faces, outline = primitive_cube()
>>> print(vertices)
[([-0.5,  0.5, -0.5], [ 0.,  1.], [-0., -0., -1.], [ 0.,  1.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  1.], [-0., -0., -1.], [ 1.,  1.,  0.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -0., -1.], [ 0.,  0.,  0.,  1.])
 ([ 0.5, -0.5, -0.5], [ 1.,  0.], [-0., -0., -1.], [ 1.,  0.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  1.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  1.], [-0., -1., -0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 1.,  1.], [-0., -1., -0.], [ 1.,  0.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-0., -1., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5, -0.5,  0.5], [ 1.,  0.], [-0., -1., -0.], [ 0.,  0.,  1.,  1.])
 ([ 0.5,  0.5, -0.5], [ 0.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  0.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 0.,  1.,  0.], [ 1.,  1.,  1.,  1.])
 ([-0.5,  0.5, -0.5], [ 0.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  0.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  0.], [ 0.,  1.,  0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5,  0.5], [ 0.,  1.], [-1., -0., -0.], [ 0.,  0.,  1.,  1.])
 ([-0.5,  0.5,  0.5], [ 1.,  1.], [-1., -0., -0.], [ 0.,  1.,  1.,  1.])
 ([-0.5, -0.5, -0.5], [ 0.,  0.], [-1., -0., -0.], [ 0.,  0.,  0.,  1.])
 ([-0.5,  0.5, -0.5], [ 1.,  0.], [-1., -0., -0.], [ 0.,  1.,  0.,  1.])
 ([ 0.5, -0.5,  0.5], [ 0.,  1.], [ 1.,  0.,  0.], [ 1.,  0.,  1.,  1.])
 ([ 0.5,  0.5,  0.5], [ 1.,  1.], [ 1.,  0.,  0.], [ 1.,  1.,  1.,  1.])
 ([ 0.5, -0.5, -0.5], [ 0.,  0.], [ 1.,  0.,  0.], [ 1.,  0.,  0.,  1.])
 ([ 0.5,  0.5, -0.5], [ 1.,  0.], [ 1.,  0.,  0.], [ 1.,  1.,  0.,  1.])]
>>> print(faces)
[[ 1  2  0]
 [ 1  3  2]
 [ 4  6  5]
 [ 6  7  5]
```

(continues on next page)

(continued from previous page)

```
 [ 9 10  8]
 [ 9 11 10]
 [12 14 13]
 [14 15 13]
 [17 18 16]
 [17 19 18]
 [20 22 21]
 [22 23 21]]
>>> print(outline)
[[ 0  2]
 [ 2  3]
 [ 3  1]
 [ 1  0]
 [ 4  6]
 [ 6  7]
 [ 7  5]
 [ 5  4]
 [ 8 10]
 [10 11]
 [11  9]
 [ 9  8]
 [12 14]
 [14 15]
 [15 13]
 [13 12]
 [16 18]
 [18 19]
 [19 17]
 [17 16]
 [20 22]
 [22 23]
 [23 21]
 [21 20]]
```

Primitive Vertices

colour

PRIMITIVE_VERTICES_METHODS	Supported geometry primitive vertices generation methods.
primitive_vertices([method])	Returns the vertices of a geometry primitive using given method.

colour.PRIMITIVE_VERTICES_METHODS

colour.**PRIMITIVE_VERTICES_METHODS = CaseInsensitiveMapping({'Quad MPL': ..., 'Grid MPL': ..., 'Cube MPL': ..., 'Sphere': ...})**

Supported geometry primitive vertices generation methods.

PRIMITIVE_VERTICES_METHODS [CaseInsensitiveMapping] **{'Cube MPL', 'Quad MPL', 'Grid MPL', 'Sphere'}**

colour.primitive_vertices

colour.**primitive_vertices**(*method='Cube MPL'*, ***kwargs*)

Returns the vertices of a geometry primitive using given method.

Parameters

- **method** (unicode, optional) – **{'Cube MPL', 'Quad MPL', 'Grid MPL', 'Sphere'}**, Vertices generation method.
- **origin** (unicode, optional) – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl(), colour.geometry.primitive_vertices_sphere()}, Primitive origin on the construction plane.
- **axis** (array_like, optional) – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_sphere()}, **{'+z', '+x', '+y', 'yz', 'xz', 'xy'}**, Axis the primitive will be normal to, or plane the primitive will be co-planar with.
- **planes** (array_like, optional) – {colour.geometry.primitive_vertices_cube_mpl()}, **{'-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}**, Included grid primitives in the cube construction.
- **width** (numeric, optional) – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive width.
- **height** (numeric, optional) – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive height.
- **depth** (numeric, optional) – {colour.geometry.primitive_vertices_quad_mpl(), colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive depth.
- **radius** (numeric, optional) – {colour.geometry.primitive_vertices_sphere()}, Sphere radius.
- **segments** (int, optional,) – {colour.geometry.primitive_vertices_sphere()}, Latitude-longitude segments, if the intermediate argument is *True*, then the sphere will have one less segment along its longitude.
- **intermediate** (bool, optional) – {colour.geometry.primitive_vertices_sphere()}, Whether to generate the sphere vertices at the center of the faces outlined by the segments of a regular sphere generated without the intermediate argument set to *True*. The resulting sphere is inscribed on the regular sphere faces but possesses the same poles.
- **width_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive width segments, quad primitive counts along the width.
- **height_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive height segments, quad primitive counts along the height.

- **depth_segments** – {colour.geometry.primitive_vertices_grid_mpl(), colour.geometry.primitive_vertices_cube_mpl()}, Primitive depth segments, quad primitive counts along the depth.

Returns Primitive vertices.

Return type ndarray

Examples

```
>>> primitive_vertices()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.],
        [ 1.,  0.,  1.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  0.,  1.],
        [ 0.,  0.,  1.]],

       [[ 0.,  1.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  1.,  1.],
        [ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 1.,  0.,  1.]]])
>>> primitive_vertices('Quad MPL')
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]])
>>> primitive_vertices('Sphere', segments=4)
array([[[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [ -3.5355339...e-01,  -4.3297802...e-17,   3.5355339...e-01],
        [ -5.0000000...e-01,  -6.1232340...e-17,   3.0616170...e-17],
        [ -3.5355339...e-01,  -4.3297802...e-17,  -3.5355339...e-01],
        [ -6.1232340...e-17,  -7.4987989...e-33,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,  -3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,  -5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,  -3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,  -6.1232340...e-17,  -5.0000000...e-01]],
```

(continues on next page)

(continued from previous page)

```
      [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
       [  3.5355339...e-01,   0.0000000...e+00,   3.5355339...e-01],
       [  5.0000000...e-01,   0.0000000...e+00,   3.0616170...e-17],
       [  3.5355339...e-01,   0.0000000...e+00,  -3.5355339...e-01],
       [  6.1232340...e-17,   0.0000000...e+00,  -5.0000000...e-01]],

      [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
       [  2.1648901...e-17,   3.5355339...e-01,   3.5355339...e-01],
       [  3.0616170...e-17,   5.0000000...e-01,   3.0616170...e-17],
       [  2.1648901...e-17,   3.5355339...e-01,  -3.5355339...e-01],
       [  3.7493994...e-33,   6.1232340...e-17,  -5.0000000...e-01]]])
```

`colour.geometry`

primitive_vertices_quad_mpl([width, height, ...])	Returns the vertices of a quad primitive for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
primitive_vertices_grid_mpl([width, height, ...])	Returns the vertices of a grid primitive made of quad primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
primitive_vertices_cube_mpl([width, height, ...])	Returns the vertices of a cube primitive made of grid primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.
primitive_vertices_sphere([radius, ...])	Returns the vertices of a latitude-longitude sphere primitive.

colour.geometry.primitive_vertices_quad_mpl

`colour.geometry.`**`primitive_vertices_quad_mpl`**(*width=1*, *height=1*, *depth=0*, *origin=array([0, 0])*, *axis='+z'*)

Returns the vertices of a quad primitive for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.

Parameters

- **width** (`numeric, optional`) – Quad width.
- **height** (`numeric, optional`) – Quad height.
- **depth** (`numeric, optional`) – Quad depth.
- **origin** (`array_like, optional`) – Quad origin on the construction plane.
- **axis** (`array_like, optional`) – **{'+z', '+x', '+y', 'yz', 'xz', 'xy'}**, Axis the quad will be normal to, or plane the quad will be co-planar with.

Returns Quad primitive vertices.

Return type ndarray

Examples

```
>>> primitive_vertices_quad_mpl()
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]])
```

colour.geometry.primitive_vertices_grid_mpl

colour.geometry.**primitive_vertices_grid_mpl**(*width=1*, *height=1*, *depth=0*, *width_segments=1*, *height_segments=1*, *origin=array([0, 0])*, *axis='+z'*)

Returns the vertices of a grid primitive made of quad primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.

Parameters

- **width** (numeric, optional) – Grid width.
- **height** (numeric, optional) – Grid height.
- **depth** (numeric, optional) – Grid depth.
- **width_segments** (int, optional) – Grid width segments, quad primitive counts along the width.
- **height_segments** (int, optional) – Grid height segments, quad primitive counts along the height.
- **origin** (array_like, optional) – Grid origin on the construction plane.
- **axis** (array_like, optional) – **{'+z', '+x', '+y', 'yz', 'xz', 'xy'}**, Axis the grid will be normal to, or plane the grid will be co-planar with.

Returns Grid primitive vertices.

Return type ndarray

Examples

```
>>> primitive_vertices_grid_mpl(width_segments=2, height_segments=2)
array([[[ 0. ,  0. ,  0. ],
        [ 0.5,  0. ,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0. ,  0.5,  0. ]],

       [[ 0. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0.5,  1. ,  0. ],
        [ 0. ,  1. ,  0. ]],

       [[ 0.5,  0. ,  0. ],
        [ 1. ,  0. ,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ]],

       [[ 0.5,  0.5,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 1. ,  1. ,  0. ],
        [ 0.5,  1. ,  0. ]]])
```

colour.geometry.primitive_vertices_cube_mpl

colour.geometry.**primitive_vertices_cube_mpl**(*width=1*, *height=1*, *depth=1*, *width_segments=1*, *height_segments=1*, *depth_segments=1*, *origin=array([0, 0, 0])*, *planes=None*)

Returns the vertices of a cube primitive made of grid primitives for use with *Matplotlib* `mpl_toolkits.mplot3d.art3d.Poly3DCollection` class.

Parameters

- **width** (`float, optional`) – Cube width.
- **height** (`float, optional`) – Cube height.
- **depth** (`float, optional`) – Cube depth.
- **width_segments** (`int, optional`) – Cube segments count along the width.
- **height_segments** (`float, optional`) – Cube segments count along the height.
- **depth_segments** (`float, optional`) – Cube segments count along the depth.
- **origin** (`array_like, optional`) – Cube origin.
- **planes** (`array_like, optional`) – **{'-x', '+x', '-y', '+y', '-z', '+z', 'xy', 'xz', 'yz', 'yx', 'zx', 'zy'}**, Grid primitives to include in the cube construction.

Returns Cube primitive vertices.

Return type ndarray

Examples

```
>>> primitive_vertices_cube_mpl()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.],
        [ 1.,  0.,  1.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  0.,  1.],
        [ 0.,  0.,  1.]],

       [[ 0.,  1.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  1.,  1.],
        [ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
```

(continues on next page)

(continued from previous page)

```
       [ 1.,  1.,  1.],
       [ 1.,  0.,  1.]]])
```

colour.geometry.primitive_vertices_sphere

colour.geometry.**primitive_vertices_sphere**(*radius=0.5*, *segments=8*, *intermediate=False*, *origin=array([0, 0, 0])*, *axis='+z'*)

Returns the vertices of a latitude-longitude sphere primitive.

Parameters

- **radius** (numeric, optional) – Sphere radius.
- **segments** (numeric, optional) – Latitude-longitude segments, if the intermediate argument is *True*, then the sphere will have one less segment along its longitude.
- **intermediate** (bool, optional) – Whether to generate the sphere vertices at the center of the faces outlined by the segments of a regular sphere generated without the intermediate argument set to *True*. The resulting sphere is inscribed on the regular sphere faces but possesses the same poles.
- **origin** (array_like, optional) – Sphere origin on the construction plane.
- **axis** (array_like, optional) – **{'+z', '+x', '+y', 'yz', 'xz', 'xy'}**, Axis (or normal of the plane) the poles of the sphere will be aligned with.

Returns Sphere primitive vertices.

Return type ndarray

Notes

- The sphere poles have latitude segments count - 1 co-located vertices.

Examples

```
>>> primitive_vertices_sphere(segments=4)
array([[[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [ -3.5355339...e-01,  -4.3297802...e-17,   3.5355339...e-01],
        [ -5.0000000...e-01,  -6.1232340...e-17,   3.0616170...e-17],
        [ -3.5355339...e-01,  -4.3297802...e-17,  -3.5355339...e-01],
        [ -6.1232340...e-17,  -7.4987989...e-33,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,  -3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,  -5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,  -3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,  -6.1232340...e-17,  -5.0000000...e-01]],

       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  3.5355339...e-01,   0.0000000...e+00,   3.5355339...e-01],
        [  5.0000000...e-01,   0.0000000...e+00,   3.0616170...e-17],
        [  3.5355339...e-01,   0.0000000...e+00,  -3.5355339...e-01],
        [  6.1232340...e-17,   0.0000000...e+00,  -5.0000000...e-01]],
```

(continues on next page)

(continued from previous page)

```
       [[  0.0000000...e+00,   0.0000000...e+00,   5.0000000...e-01],
        [  2.1648901...e-17,   3.5355339...e-01,   3.5355339...e-01],
        [  3.0616170...e-17,   5.0000000...e-01,   3.0616170...e-17],
        [  2.1648901...e-17,   3.5355339...e-01,  -3.5355339...e-01],
        [  3.7493994...e-33,   6.1232340...e-17,  -5.0000000...e-01]]])
```

Automatic Colour Conversion Graph

- *Conversion*

Conversion

`colour`

`convert`(a, source, target, **kwargs)	Converts given object a from source colour representation to target colour representation using the automatic colour conversion graph.
`describe_conversion_path`(source, target[, ...])	Describes the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

colour.convert

`colour.`**`convert`**(*a*, *source*, *target*, ***kwargs*)

Converts given object a from source colour representation to target colour representation using the automatic colour conversion graph.

The conversion is performed by finding the shortest path in a NetworkX `DiGraph` class instance.

The conversion path adopts the **'1'** domain-range scale and the object a is expected to be *soft* normalised accordingly. For example, *CIE XYZ* tristimulus values arguments for use with the *CAM16* colour appearance model should be in domain *[0, 1]* instead of the domain *[0, 100]* used with the **'Reference'** domain-range scale. The arguments are typically converted as follows:

- *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 *360*.
- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.

See the Domain-Range Scales page for more information.

Parameters

- **a** (`array_like or numeric or SpectralDistribution`) – Object a to convert. If a represents a reflectance, transmittance or absorptance value, the expectation is that it is viewed under *CIE Standard Illuminant D Series D65*. The illuminant can be changed on a per definition basis along the conversion path.
- **source** (`unicode`) – Source colour representation, i.e. the source node in the automatic colour conversion graph.
- **target** (`unicode`) – Target colour representation, i.e. the target node in the automatic colour conversion graph.

- ****kwargs** (dict, optional) – {'*'}, Please refer to the documentation of the supported conversion definitions.

  Arguments for the conversion definitions are passed as keyword arguments whose names is those of the conversion definitions and values set as dictionaries. For example, in the conversion from spectral distribution to *sRGB* colourspace, passing arguments to the `colour.sd_to_XYZ()` definition is done as follows:

```
convert(sd, 'Spectral Distribution', 'sRGB', sd_to_XYZ={'illuminant
↪': SDS_ILLUMINANTS['FL2']})
```

  It is also possible to pass keyword arguments directly to the various conversion definitions irrespective of their name. This is dangerous and could cause unexpected behaviour because of unavoidable discrepancies with the underlying `colour.utilities.filter_kwargs()` definition between Python 2.7 and 3.x. Using this direct keyword arguments passing mechanism might also ends up passing incompatible arguments to a given conversion definition. Consider the following conversion:

```
convert(sd, 'Spectral Distribution', 'sRGB', 'illuminant': SDS_
↪ILLUMINANTS['FL2'])
```

  Because both the `colour.sd_to_XYZ()` and `colour.XYZ_to_sRGB()` definitions have an *illuminant* argument, *SDS_ILLUMINANTS['FL2']* will be passed to both of them and will raise an exception in the `colour.XYZ_to_sRGB()` definition. This will be addressed in the future by either catching the exception and trying a new time without the keyword argument or more elegantly via type checking.

  With that in mind, this mechanism offers some good benefits: For example, it allows defining a conversion from *CIE XYZ* colourspace to *n* different colour models while passing an illuminant argument but without having to explicitly define all the explicit conversion definition arguments:

```
a = np.array([0.20654008, 0.12197225, 0.05136952])
illuminant = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer'][
↪'D65']
for model in ('CIE xyY', 'CIE Lab'):
    convert(a, 'CIE XYZ', model, illuminant=illuminant)
```

  Instead of:

```
for model in ('CIE xyY', 'CIE Lab'):
    convert(a, 'CIE XYZ', model, XYZ_to_xyY={'illuminant':␣
↪illuminant}, XYZ_to_Lab={'illuminant': illuminant})
```

  Mixing both approaches is possible for the brevity benefits. It is made possible because the keyword arguments directly passed are filtered first and then the resulting dict is updated with the explicit conversion definition arguments:

```
illuminant = CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer'][
↪'D65']
 convert(sd, 'Spectral Distribution', 'sRGB', 'illuminant': SDS_
↪ILLUMINANTS['FL2'], XYZ_to_sRGB={'illuminant': illuminant})
```

  For inspection purposes, verbose is enabled by passing arguments to the `colour.describe_conversion_path()` definition via the verbose keyword argument as follows:

```
convert(sd, 'Spectral Distribution', 'sRGB', verbose={'mode': 'Long'}
↪)
```

Returns Converted object a.

Return type ndarray or numeric or *SpectralDistribution*

Warning: The domain-range scale is **'1'** and cannot be changed.

Notes

- The **RGB** colour representation is assumed to be linear and representing *scene-referred* imagery, i.e. **Scene-Referred RGB** representation. To encode such *RGB* values as *output-referred* (*display-referred*) imagery, i.e. encode the *RGB* values using an encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF), the **Output-Referred RGB** representation must be used:

```
convert(RGB, 'Scene-Referred RGB', 'Output-Referred RGB')
```

  Likewise, encoded *output-referred RGB* values can be decoded with the **Scene-Referred RGB** representation:

```
convert(RGB, 'Output-Referred RGB', 'Scene-Referred RGB')
```

- Various defaults have been adopted compared to the low-level *Colour* API:
  - The default illuminant for the computation is *CIE Standard Illuminant D Series D65*. It can be changed on a per definition basis along the conversion path.
  - The default *RGB* colourspace primaries and whitepoint are that of the *BT.709/sRGB* colourspace. They can be changed on a per definition basis along the conversion path.
  - When using **sRGB** as a source or target colour representation, the convenient `colour.sRGB_to_XYZ()` and `colour.XYZ_to_sRGB()` definitions are used, respectively. Thus, decoding and encoding using the sRGB electro-optical transfer function (EOTF) and its inverse will be applied by default.
  - Most of the colour appearance models have defaults set according to *IEC 61966-2-1:1999* viewing conditions, i.e. *sRGB* 64 Lux ambiant illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.

Examples

```
>>> from colour import SDS_COLOURCHECKERS
>>> sd = SDS_COLOURCHECKERS['ColorChecker N Ohta']['dark skin']
>>> convert(sd, 'Spectral Distribution', 'sRGB',
...     verbose={'mode': 'Short', 'width': 75})
...
==============================================================================
*                                                                            *
*   [ Conversion Path ]                                                      *
*                                                                            *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                            *
*                                                                            *
==============================================================================
array([ 0.4567579...,  0.3098698...,  0.2486192...])
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> convert(sd, 'Spectral Distribution', 'sRGB',
...     sd_to_XYZ={'illuminant': illuminant})
...
```

(continues on next page)

(continued from previous page)

```
array([ 0.4792457...,  0.3167696...,  0.1736272...])
>>> a = np.array([0.45675795, 0.30986982, 0.24861924])
>>> convert(a, 'Output-Referred RGB', 'CAM16UCS')
...
array([ 0.3999481...,  0.0920655...,  0.0812752...])
>>> a = np.array([0.39994811, 0.09206558, 0.08127526])
>>> convert(a, 'CAM16UCS', 'sRGB', verbose={'mode': 'Short', 'width': 75})
...
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "UCS_Li2017_to_JMh_CAM16" --> "JMh_CAM16_to_CAM16" -->                *
*   "CAM16_to_XYZ" --> "XYZ_to_sRGB"                                      *
*                                                                         *
===========================================================================
array([ 0.4567576...,  0.3098826...,  0.2486222...])
```

colour.describe_conversion_path

colour.**describe_conversion_path**(*source*, *target*, *mode='Short'*, *width=79*, *padding=3*, *print_callable=<built-in function print>*, ***kwargs*)

Describes the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

Parameters

- **source** (unicode) – Source colour representation, i.e. the source node in the automatic colour conversion graph.
- **target** (unicode) – Target colour representation, i.e. the target node in the automatic colour conversion graph.
- **mode** (unicode, optional) – **{'Short', 'Long', 'Extended'}**, Verbose mode: *Short* describes the conversion path, *Long* provides details about the arguments, definitions signatures and output values, *Extended* appends the definitions documentation.
- **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.
- ****kwargs** (dict, optional) – {colour.convert()}, Please refer to the documentation of the previously listed definition.

Examples

```
>>> describe_conversion_path('Spectral Distribution', 'sRGB', width=75)
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                         *
*                                                                         *
===========================================================================
```

Input and Output

Image Data

colour

READ_IMAGE_METHODS	Supported read image methods.
read_image(path[, bit_depth, method])	Reads the image at given path using given method.
WRITE_IMAGE_METHODS	Supported write image methods.
write_image(image, path[, bit_depth, method])	Writes given image at given path using given method.

colour.READ_IMAGE_METHODS

colour.**READ_IMAGE_METHODS = CaseInsensitiveMapping({'Imageio': ..., 'OpenImageIO': ...})**

Supported read image methods.

READ_IMAGE_METHODS [CaseInsensitiveMapping] **{'Imageio', 'OpenImageIO'}**

colour.read_image

colour.**read_image**(*path*, *bit_depth='float32'*, *method='OpenImageIO'*, ***kwargs*)

Reads the image at given path using given method.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, for the *Imageio* method, the image data is converted with colour.io.convert_bit_depth() definition after reading the image, for the *OpenImageIO* method, the bit depth conversion behaviour is driven directly by the library, this definition only converts to the relevant data type after reading.
- **method** (unicode, optional) – **{'OpenImageIO', 'Imageio'}**, Read method, i.e. the image library used for reading images.
- **attributes** (bool, optional) – {colour.io.read_image_OpenImageIO()}, Whether to return the image attributes.

Returns Image as a ndarray.

Return type ndarray

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, `kwargs` is passed directly to the wrapped definition.
- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.WRITE_IMAGE_METHODS

colour.**WRITE_IMAGE_METHODS = CaseInsensitiveMapping({'Imageio': ..., 'OpenImageIO': ...})**

Supported write image methods.

WRITE_IMAGE_METHODS [CaseInsensitiveMapping] **{'Imageio', 'OpenImageIO'}**

colour.write_image

colour.**write_image**(*image*, *path*, *bit_depth='float32'*, *method='OpenImageIO'*, ***kwargs*)

Writes given image at given path using given method.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, for the *Imageio* method, the image data is converted with `colour.io.convert_bit_depth()` definition prior to writing the image.
- **method** (unicode, optional) – **{'OpenImageIO', 'Imageio'}**, Write method, i.e. the image library used for writing images.
- **attributes** (array_like, optional) – {`colour.io.write_image_OpenImageIO()`}, An array of `colour.io.ImageAttribute_Specification` class instances used to set attributes of the image.

Returns Definition success.

Return type bool

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, `kwargs` is passed directly to the wrapped definition.

Examples

Basic image writing:

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Advanced image writing while setting attributes using *OpenImageIO*:

```
>>> compression = ImageAttribute_Specification('Compression', 'none')
>>> write_image(image, path, bit_depth='uint8', attributes=[compression])
...
True
```

Ancillary Objects

`colour.io`

ImageAttribute_Specification(name, value[, ...])	Defines the an image specification attribute.
convert_bit_depth(a[, bit_depth])	Converts given array to given bit depth, the current bit depth of the array is used to determine the appropriate conversion path.
read_image_OpenImageIO(path[, bit_depth, ...])	Reads the image at given path using *OpenImageIO*.
write_image_OpenImageIO(image, path[, ...])	Writes given image at given path using *OpenImageIO*.
read_image_Imageio(path[, bit_depth])	Reads the image at given path using *Imageio*.
write_image_Imageio(image, path[, bit_depth])	Writes given image at given path using *Imageio*.

colour.io.ImageAttribute_Specification

class colour.io.**ImageAttribute_Specification**(*name*, *value*, *type_=None*)

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__()

Methods

__init__()

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

name	Alias for field number 0
type_	Alias for field number 2
value	Alias for field number 1

colour.io.convert_bit_depth

colour.io.**convert_bit_depth**(*a*, *bit_depth='float32'*)

Converts given array to given bit depth, the current bit depth of the array is used to determine the appropriate conversion path.

Parameters

- **a** (array_like) – Array to convert to given bit depth.
- **bit_depth** (unicode) – Bit depth.

Returns Converted array.

Return type ndarray

Examples

```
>>> a = np.array([0.0, 0.5, 1.0])
>>> convert_bit_depth(a, 'uint8')
array([  0, 128, 255], dtype=uint8)
>>> convert_bit_depth(a, 'uint16')
array([    0, 32768, 65535], dtype=uint16)
>>> convert_bit_depth(a, 'float16')
array([ 0. ,  0.5,  1. ], dtype=float16)
>>> a = np.array([0, 128, 255], dtype=np.uint8)
>>> convert_bit_depth(a, 'uint16')
array([    0, 32896, 65535], dtype=uint16)
>>> convert_bit_depth(a, 'float32')
array([ 0.       ,  0.501960...,  1.       ], dtype=float32)
```

colour.io.read_image_OpenImageIO

colour.io.**read_image_OpenImageIO**(*path*, *bit_depth='float32'*, *attributes=False*)

Reads the image at given path using *OpenImageIO*.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, the bit depth conversion behaviour is driven directly by *OpenImageIO*, this definition only converts to the relevant data type after reading.
- **attributes** (bool, optional) – Whether to return the image attributes.

Returns Image as a ndarray or tuple of image as ndarray and list of attributes

Return type ndarray or tuple

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
```

colour.io.write_image_OpenImageIO

colour.io.**write_image_OpenImageIO**(*image*, *path*, *bit_depth='float32'*, *attributes=None*)

Writes given image at given path using *OpenImageIO*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, the bit depth conversion behaviour is ruled directly by *OpenImageIO*.
- **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
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Advanced image writing while setting attributes:

```
>>> compression = ImageAttribute_Specification('Compression', 'none')
>>> write_image(image, path, 'uint8', [compression])
True
```

colour.io.read_image_Imageio

colour.io.**read_image_Imageio**(*path*, *bit_depth='float32'*, ***kwargs*)

Reads the image at given path using *Imageio*.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, the image data is converted with colour.io.convert_bit_depth() definition after reading the image.
- ****kwargs** (dict, optional) – Keywords arguments.

Returns Image as a ndarray.

Return type ndarray

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image_Imageio(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.io.write_image_Imageio

colour.io.**write_image_Imageio**(*image*, *path*, *bit_depth='float32'*, ***kwargs*)

Writes given image at given path using *Imageio*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, the image data is converted with colour.io.convert_bit_depth() definition prior to writing the image.
- ****kwargs** (dict, optional) – Keywords arguments.

Returns Definition success.

Return type bool

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Look Up Table (LUT) Data

colour

LUT1D([table, name, domain, size, comments])	Defines the base class for a 1D *LUT*.
LUT3x1D([table, name, domain, size, comments])	Defines the base class for a 3x1D *LUT*.
LUT3D([table, name, domain, size, comments])	Defines the base class for a 3D *LUT*.
LUTSequence(*args)	Defines the base class for a *LUT* sequence, i.e. a series of *LUTs*.

colour.LUT1D

class colour.**LUT1D**(*table=None*, *name=None*, *domain=None*, *size=10*, *comments=None*)

Bases: colour.io.luts.lut.AbstractLUT

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*.

Methods

- __init__()
- 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
----------------

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)))
LUT1D - ...
--------...

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
--------------

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*)

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[0 1]
```

While an explicit domain defines every single discrete samples:

```
[0.0 0.1 0.2 0.4 0.8 1.0]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT1D().is_domain_explicit()
False
>>> table = domain = np.linspace(0, 1, 10)
>>> LUT1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size=10*, *domain=array([0, 1])*)

Returns a linear table, the number of output samples n is equal to `size`.

Parameters

- **size** (`int, optional`) – Expected table size.
- **domain** (`array_like, optional`) – Domain of the table.

Returns Linear table with `size` samples.

Return type ndarray

Examples

```
>>> LUT1D.linear_table(5, np.array([-0.1, 1.5]))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
>>> LUT1D.linear_table(domain=np.linspace(-0.1, 1.5, 5))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
```

apply(*RGB*, *interpolator=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_kwargs=None*, ***kwargs*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (`array_like`) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (`object, optional`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict_like, optional`) – Arguments to use when instantiating the interpolating function.
- ****kwargs** (`dict, optional`) – Keywords arguments for deprecation management.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT1D(LUT1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *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.
- **interpolator** (`object, optional`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict_like, optional`) – Arguments to use when instantiating the interpolating function.
- **size** (`int, optional`) – Expected table size in case of an upcast to a `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

```
>>> LUT = LUT1D()
>>> print(LUT.as_LUT(LUT1D))
LUT1D - Unity 10 - Converted 1D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D))
LUT3x1D - Unity 10 - Converted 1D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 1D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
```

(continues on next page)

(continued from previous page)

```
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUT3x1D

class colour.**LUT3x1D**(*table=None*, *name=None*, *domain=None*, *size=10*, *comments=None*)

Bases: colour.io.luts.lut.AbstractLUT

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*.

Methods

- __init__()
- 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
------------------

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)))
...
LUT3x1D - ...
----------...

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
----------------

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*)

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 1]
 [0 1]
 [0 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT3x1D().is_domain_explicit()
False
>>> samples = np.linspace(0, 1, 10)
>>> table = domain = tstack([samples, samples, samples])
>>> LUT3x1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size=10*, *domain=array([[0, 0, 0], [1, 1, 1]])*)

Returns a linear table, the number of output samples n is equal to `size * 3` or `size[0] + size[1] + size[2]`.

Parameters

- **size** (int or array_like, optional) – Expected table size.
- **domain** (array_like, optional) – Domain of the table.

Returns Linear table with `size * 3` or `size[0] + size[1] + size[2]` samples.

Return type ndarray

Warning: If `size` is non uniform, the linear table will be padded accordingly.

Examples

```
>>> LUT3x1D.linear_table(
...     5, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[-0.1, -0.2, -0.4],
       [ 0.3,  0.6,  1.2],
       [ 0.7,  1.4,  2.8],
       [ 1.1,  2.2,  4.4],
       [ 1.5,  3. ,  6. ]])
>>> LUT3x1D.linear_table(
...     np.array([5, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[-0.1, -0.2, -0.4],
       [ 0.3,  1.4,  6. ],
       [ 0.7,  3. ,  nan],
       [ 1.1,  nan,  nan],
       [ 1.5,  nan,  nan]])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
...                    [1.5, np.nan, np.nan]])
>>> LUT3x1D.linear_table(domain=domain)
array([[-0.1, -0.2, -0.4],
       [ 0.3,  1.4,  6. ],
       [ 0.7,  3. ,  nan],
       [ 1.1,  nan,  nan],
       [ 1.5,  nan,  nan]])
```

apply(*RGB*, *interpolator=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_kwargs=None*, ***kwargs*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT3x1D(LUT3x1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4423903...,  0.4503801...,  0.3581625...])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
...                    [1.5, np.nan, np.nan]])
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *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.
- **interpolator** (`object`, `optional`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict_like`, `optional`) – Arguments to use when instantiating the interpolating function.
- **size** (`int`, `optional`) – Expected table size in case of an upcast to a `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

```
>>> LUT = LUT3x1D()
>>> print(LUT.as_LUT(LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D))
LUT3x1D - Unity 10 - Converted 3x1D to 3x1D
-------------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 3x1D to 3D
---------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUT3D

class colour.**LUT3D**(*table=None*, *name=None*, *domain=None*, *size=33*, *comments=None*)

Bases: colour.io.luts.lut.AbstractLUT

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*.

Methods

- __init__()
- 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
----------------

Dimensions : 3
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)))
LUT3D - ...
--------...

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
--------------

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*)

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 0 0]
 [1 1 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
```

(continues on next page)

(continued from previous page)

```
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT3D().is_domain_explicit()
False
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.7, 1.4, 6.0],
...                    [1.5, 3.0, np.nan]])
>>> LUT3D(domain=domain).is_domain_explicit()
True
```

static linear_table(*size=33*, *domain=array([[0, 0, 0], [1, 1, 1]])*)

Returns a linear table, the number of output samples n is equal to `size**3 * 3` or `size[0] * size[1] * size[2] * 3`.

Parameters

- **size** (`int or array_like, optional`) – Expected table size.
- **domain** (`array_like, optional`) – Domain of the table.

Returns Linear table with `size**3 * 3` or `size[0] * size[1] * size[2] * 3` samples.

Return type ndarray

Examples

```
>>> LUT3D.linear_table(
...     3, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  2.8],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  2.8],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  2.8],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  2.8],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  2.8],
```

(continues on next page)

(continued from previous page)

```
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  2.8],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  2.8],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  2.8],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  2.8],
         [ 1.5,  3. ,  6. ]]]])
>>> LUT3D.linear_table(
...     np.array([3, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.7, 1.4, 6.0],
...                    [1.5, 3.0, np.nan]])
>>> LUT3D.linear_table(domain=domain)
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],
```

(continues on next page)

(continued from previous page)

```
        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
```

apply(*RGB*, *interpolator=<function table_interpolation_trilinear>*, *interpolator_kwargs=None*, ***kwargs*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (object, optional) – Interpolator object to use as interpolating function.
- **interpolator_kwargs** (dict_like, optional) – Arguments to use when calling the interpolating function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT3D(LUT3D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4583277...,  0.4583277...,  0.4583277...])
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
```

(continues on next page)

(continued from previous page)

```
...                     [1.5, np.nan, np.nan]])
>>> table = spow(LUT3D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *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.
- **interpolator** (`object, optional`) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (`dict_like, optional`) – Arguments to use when instantiating the interpolating function.
- **size** (`int, optional`) – Expected table size in case of a downcast from a `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

```
>>> LUT = LUT3D()
>>> print(LUT.as_LUT(LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D, force_conversion=True))
LUT3x1D - Unity 33 - Converted 3D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D))
LUT3D - Unity 33 - Converted 3D to 3D
-------------------------------------

Dimensions : 3
```

(continues on next page)

(continued from previous page)

```
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

colour.LUTSequence

class colour.**LUTSequence**(**args*)

Bases: collections.abc.MutableSequence

Defines the base class for a *LUT* sequence, i.e. a series of *LUTs*.

The *colour.LUTSequence* class can be used to model series of *LUTs* such as when a shaper *LUT* is combined with a 3D *LUT*.

Parameters ***args** (list, optional) – Sequence of *colour.LUT1D*, *colour.LUT3x1D*, *colour.LUT3D* or *colour.io.lut.l.AbstractLUTSequenceOperator* class instances.

Attributes

- sequence

Methods

- __init__()
- __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))
LUT Sequence
------------

Overview

    LUT1D ---> LUT3D ---> LUT3x1D
```

(continues on next page)

(continued from previous page)

```
Operations

    LUT1D - Unity 10
    ----------------

    Dimensions : 1
    Domain     : [ 0.  1.]
    Size       : (10,)

    LUT3D - Unity 3
    ---------------

    Dimensions : 3
    Domain     : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size       : (3, 3, 3, 3)

    LUT3x1D - Unity 10
    ------------------

    Dimensions : 2
    Domain     : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size       : (10, 3)
```

__init__(**args*)

property sequence

Getter and setter property for the underlying *LUT* sequence.

Parameters **value** (list) – Value to set the the underlying *LUT* sequence with.

Returns Underlying *LUT* sequence.

Return type list

__getitem__(*index*)

Returns the *LUT* sequence item at given index.

Parameters **index** (int) – *LUT* sequence item index.

Returns *LUT* sequence item at given index.

Return type *LUT1D* or *LUT3x1D* or *LUT3D* or *AbstractLUTSequenceOperator*

__setitem__(*index*, *value*)

Sets given the *LUT* sequence item at given index with given value.

Parameters

- **index** (int) – *LUT* sequence item index.
- **value** (LUT1D or LUT3x1D or LUT3D or AbstractLUTSequenceOperator) – Value.

__delitem__(*index*)

Deletes the *LUT* sequence item at given index.

Parameters **index** (int) – *LUT* sequence item index.

__hash__ = None

__weakref__
list of weak references to the object (if defined)

__len__()
Returns the *LUT* sequence items count.

Returns *LUT* sequence items count.

Return type int

__str__()
Returns a formatted string representation of the *LUT* sequence.

Returns Formatted string representation.

Return type unicode

__repr__()
Returns an evaluable string representation of the *LUT* sequence.

Returns Evaluable string representation.

Return type unicode

__eq__(*other*)
Returns whether the *LUT* sequence is equal to given other object.

Parameters **other** (object) – Object to test whether it is equal to the *LUT* sequence.

Returns Is given object equal to the *LUT* sequence.

Return type bool

__ne__(*other*)
Returns whether the *LUT* sequence is not equal to given other object.

Parameters **other** (object) – Object to test whether it is not equal to the *LUT* sequence.

Returns Is given object not equal to the *LUT* sequence.

Return type bool

insert(*index*, *LUT*)
Inserts given *LUT* at given index into the *LUT* sequence.

Parameters

- **index** (index) – Index to insert the *LUT* at into the *LUT* sequence.
- **LUT** (LUT1D or LUT3x1D or LUT3D or AbstractLUTSequenceOperator) – *LUT* to insert into the *LUT* sequence.

apply(*RGB*, *interpolator_1D=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_1D_kwargs=None*, *interpolator_3D=<function table_interpolation_trilinear>*, *interpolator_3D_kwargs=None*, ***kwargs*)
Applies the *LUT* sequence sequentially to given *RGB* colourspace array.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* sequence sequentially onto.
- **interpolator_1D** (object, optional) – Interpolator object to use as interpolating function for colour.LUT1D (and colour.LUT3x1D) class instances.
- **interpolator_1D_kwargs** (dict_like, optional) – Arguments to use when calling the interpolating function for colour.LUT1D (and colour.LUT3x1D) class instances.

- **interpolator_3D** (object, optional) – Interpolator object to use as interpolating function for colour.LUT3D class instances.
- **interpolator_3D_kwargs** (dict_like, optional) – Arguments to use when calling the interpolating function for colour.LUT3D class instances.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Processed *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT_1 = LUT1D(LUT1D.linear_table(16) + 0.125)
>>> LUT_2 = LUT3D(LUT3D.linear_table(16) ** (1 / 2.2))
>>> LUT_3 = LUT3x1D(LUT3x1D.linear_table(16) * 0.750)
>>> LUT_sequence = LUTSequence(LUT_1, LUT_2, LUT_3)
>>> samples = np.linspace(0, 1, 5)
>>> RGB = tstack([samples, samples, samples])
>>> LUT_sequence.apply(RGB)
array([[ 0.2899886...,  0.2899886...,  0.2899886...],
       [ 0.4797662...,  0.4797662...,  0.4797662...],
       [ 0.6055328...,  0.6055328...,  0.6055328...],
       [ 0.7057779...,  0.7057779...,  0.7057779...],
       [ 0.75     ...,  0.75     ...,  0.75     ...]])
```

copy()

Returns a copy of the *LUT* sequence.

Returns *LUT* sequence copy.

Return type *LUTSequence*

read_LUT(path[, method])	Reads given *LUT* file using given method.
write_LUT(LUT, path[, decimals, method])	Writes given *LUT* to given file using given method.

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

[], [], []

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
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'eotf_sRGB_1D.spi1d')
>>> print(read_LUT(path))
LUT1D - eotf sRGB 1D
--------------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3D *Sony .spi3d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'Colour_Correct.spi3d')
>>> print(read_LUT(path))
LUT3D - Colour Correct
----------------------

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

[], [], []

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')
```

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')
```

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')
```

Ancillary Objects

colour.io

AbstractLUTSequenceOperator()	Defines the base class for *LUT* sequence operators.

colour.io.AbstractLUTSequenceOperator

class colour.io.**AbstractLUTSequenceOperator**

Bases: object

Defines the base class for *LUT* sequence operators.

This is an ABCMeta abstract class that must be inherited by sub-classes.

Methods

- apply()

abstract apply(*RGB*, **args*)

Applies the *LUT* sequence operator to given *RGB* colourspace array.

Parameters **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* sequence operator onto.

Returns Processed *RGB* colourspace array.

Return type ndarray

__weakref__

list of weak references to the object (if defined)

LUT_to_LUT(LUT, cls[, force_conversion])	Converts given *LUT* to given cls class instance.
read_LUT_Cinespace(path)	Reads given *Cinespace .csp LUT* file.
write_LUT_Cinespace(LUT, path[, decimals])	Writes given *LUT* to given *Cinespace .csp LUT* file.
read_LUT_IridasCube(path)	Reads given *Iridas .cube LUT* file.
write_LUT_IridasCube(LUT, path[, decimals])	Writes given *LUT* to given *Iridas .cube LUT* file.
read_LUT_SonySPI1D(path)	Reads given *Sony .spi1d LUT* file.
write_LUT_SonySPI1D(LUT, path[, decimals])	Writes given *LUT* to given *Sony .spi1d LUT* file.
read_LUT_SonySPI3D(path)	Reads given *Sony .spi3d LUT* file.
write_LUT_SonySPI3D(LUT, path[, decimals])	Writes given *LUT* to given *Sony .spi3d LUT* file.

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 if destructive.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.

- **interpolator_kwargs** (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
-------------------------------------

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
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT_to_LUT(LUT3D(), LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

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

[]

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
-------------------------------

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',
...     'Colour_Correct.csp')
>>> print(read_LUT_Cinespace(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

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

[]

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')
```

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')
```

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

[]

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
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
```

(continues on next page)

(continued from previous page)

```
             [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 3D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'Colour_Correct.cube')
>>> print(read_LUT_IridasCube(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

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
--------------

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

[]

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')
```

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')
```

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',
...     'eotf_sRGB_1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT1D - eotf sRGB 1D
--------------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3x1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'eotf_sRGB_3x1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT3x1D - eotf sRGB 3x1D
------------------------

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.eotf_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')
```

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,
```

(continues on next page)

(continued from previous page)

```
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI1D(LUT, 'My_LUT.cube')
```

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 an ordered and an unordered 3D *Sony .spi3d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'Colour_Correct.spi3d')
>>> print(read_LUT_SonySPI3D(path))
LUT3D - Colour Correct
----------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'Colour_Correct_Unordered.spi3d')
>>> print(read_LUT_SonySPI3D(path))
LUT3D - Colour Correct Unordered
--------------------------------

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')
```

CSV Tabular Data

colour

read_sds_from_csv_file(path[, delimiter, ...])	Reads the spectral data from given *CSV* file and return its content as an *OrderedDict* of colour.SpectralDistribution classes.
read_spectral_data_from_csv_file(path[, ...])	Reads the spectral data from given *CSV* file in the following form.
write_sds_to_csv_file(sds, path[, ...])	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_ohta.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']
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],
                      [ 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],
```

(continues on next page)

(continued from previous page)

```
                       [ 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_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
```

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_ohta.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',
 '14',
 '15',
 '16',
 '17',
 '18',
 '19',
 '20',
```

(continues on next page)

(continued from previous page)

```
'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

SpectralDistribution_IESTM2714([path, ...])	Defines a *IES TM-27-14* 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*, ***kwargs*)

Bases: colour.colorimetry.spectrum.SpectralDistribution

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** (Header_IESTM2714, 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.
- **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.
- **name** (unicode, optional) – Spectral distribution name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_kwargs** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_kwargs** (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.

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.

Attributes

- `mapping`
- `path`
- `header`
- `spectral_quantity`
- `reflection_geometry`
- `transmission_geometry`
- `bandwidth_FWHM`
- `bandwidth_corrected`

Methods

- `__init__()`
- `read()`
- `write()`

References

[]

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> sd.name
'Unknown - N/A - Rare earth fluorescent lamp'
>>> sd.header.comments
'Ambient temperature 25 degrees C.'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> sd[501.7]
0.0950000...
```

__init__(*path=None*, *header=None*, *spectral_quantity=None*, *reflection_geometry=None*, *transmission_geometry=None*, *bandwidth_FWHM=None*, *bandwidth_corrected=None*, ***kwargs*)

property mapping

Getter property for the mapping structure.

Returns Mapping structure.

Return type *Structure*

property path

Getter and setter property for the path.

Parameters **value** (unicode) – Value to set the path with.

Returns Path.

Return type unicode

property header

Getter and setter property for the header.

Parameters **value** (Header_IESTM2714) – Value to set the header with.

Returns Header.

Return type Header_IESTM2714

property spectral_quantity

Getter and setter property for the spectral quantity.

Parameters **value** (unicode) – Value to set the spectral quantity with.

Returns Spectral quantity.

Return type unicode

property reflection_geometry

Getter and setter property for the reflection geometry.

Parameters **value** (unicode) – Value to set the reflection geometry with.

Returns Reflection geometry.

Return type unicode

property transmission_geometry

Getter and setter property for the transmission geometry.

Parameters **value** (unicode) – Value to set the transmission geometry with.

Returns Transmission geometry.

Return type unicode

property bandwidth_FWHM

Getter and setter property for the full-width half-maximum bandwidth.

Parameters **value** (numeric) – Value to set the full-width half-maximum bandwidth with.

Returns Full-width half-maximum bandwidth.

Return type numeric

property bandwidth_corrected

Getter and setter property for whether bandwidth correction has been applied to the measured data.

Parameters **value** (bool) – Whether bandwidth correction has been applied to the measured data.

Returns Whether bandwidth correction has been applied to the measured data.

Return type bool

read()

Reads and parses the spectral data XML file path.

Returns Definition success.

Return type bool

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> sd.name
'Unknown - N/A - Rare earth fluorescent lamp'
>>> sd.header.comments
'Ambient temperature 25 degrees C.'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> sd[400]
0.0339999...
```

write()

Write the spectral distribution spectral data to XML file path.

Returns Definition success.

Return type bool

Examples

```
>>> from os.path import dirname, join
>>> from shutil import rmtree
>>> from tempfile import mkdtemp
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> temporary_directory = mkdtemp()
>>> sd.path = join(temporary_directory, 'Fluorescent.spdx')
>>> sd.write()
True
>>> rmtree(temporary_directory)
```

X-Rite Data

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.

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',
...                           'X-Rite_Digital_Colour_Checker.txt')
>>> sds_data = read_sds_from_xrite_file(xrite_file)
>>> pprint(list(sds_data.keys()))
['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']
```

Colour Models

- *Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates*
- *Common Models*
- *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)*
- $I_GP_GT_G$ *Colourspace*
- *IPT Colourspace*
- *hdr-CIELAB Colourspace*
- *hdr-IPT Colourspace*
- *OSA UCS Colourspace*
- $JzAzBz$ *Colourspace*

Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates

`colour`

`XYZ_to_xyY`(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE xyY* colourspace and reference *illuminant*.
`xyY_to_XYZ`(xyY)	Converts from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.
`XYZ_to_xy`(XYZ[, illuminant])	Returns the *CIE xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.
`xy_to_XYZ`(xy)	Returns the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.
`xyY_to_xy`(xyY)	Converts from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.
`xy_to_xyY`(xy[, Y])	Converts from *CIE xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given Y *luminance*.

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

[], []

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xyY(XYZ)
array([ 0.5436955...,  0.3210794...,  0.1219722...])
```

colour.xyY_to_XYZ

colour.**xyY_to_XYZ**(*xyY*)

Converts from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.

Parameters **xyY** (array_like) – *CIE xyY* colourspace 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

[], []

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_XYZ(xyY)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.XYZ_to_xy

colour.**XYZ_to_xy**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Returns the *CIE 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 *CIE xy* chromaticity coordinates.

Return type ndarray

Notes

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

References

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xy(XYZ)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_XYZ

colour.**xy_to_XYZ**(*xy*)

Returns the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE 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

[]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_XYZ(xy)
array([ 1.6933366...,  1.        ,  0.4211574...])
```

colour.xyY_to_xy

colour.**xyY_to_xy**(*xyY*)

Converts from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.

xyY argument with last dimension being equal to 2 will be assumed to be a *CIE xy* chromaticity coordinates argument and will be returned directly by the definition.

Parameters **xyY** (array_like) – *CIE xyY* colourspace array or *CIE xy* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_xy(xyY)
array([ 0.54369557...,  0.32107944...])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xyY_to_xy(xy)
array([ 0.54369557...,  0.32107944...])
```

colour.xy_to_xyY

colour.**xy_to_xyY**(*xy*, *Y=1*)

Converts from *CIE 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`) – *CIE 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

[]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944, 1.00000000])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy, 100)
array([   0.5436955...,    0.3210794...,  100.        ])
```

Common Models

`colour.models`

`Jab_to_JCh`(Jab)	Converts from *Jab** colour representation to *JCh* colour representation.
`JCh_to_Jab`(JCh)	Converts from *JCh* colour representation to *Jab** colour representation.

colour.models.Jab_to_JCh

`colour.models.`**`Jab_to_JCh`**(*Jab*)

Converts from *Jab** colour representation to *JCh* colour representation.

This definition is used to perform conversion from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace and for other similar conversions. It implements a generic transformation from *Lightness* J, a and b opponent colour dimensions to the correlates of *Lightness* J, chroma C and hue angle h.

Parameters **Jab** (`array_like`) – *Jab** colour representation array.

Returns *JCh* colour representation array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

References

[]

Examples

```
>>> Jab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Jab_to_JCh(Jab)
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.models.JCh_to_Jab

colour.models.**JCh_to_Jab**(*JCh*)

Converts from *JCh* colour representation to *Jab** colour representation.

This definition is used to perform conversion from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace and for other similar conversions. It implements a generic transformation from the correlates of *Lightness* J, chroma C and hue angle h to *Lightness* J, a and b opponent colour dimensions.

Parameters **JCh** (array_like) – *JCh* colour representation array.

Returns *Jab** colour representation array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

References

[]

Examples

```
>>> JCh = np.array([41.52787529, 59.12425901, 27.08848784])
>>> JCh_to_Jab(JCh)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE L*a*b* Colourspace

colour

XYZ_to_Lab(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.
Lab_to_XYZ(Lab[, illuminant])	Converts from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.
Lab_to_LCHab(Lab)	Converts from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.
LCHab_to_Lab(LCHab)	Converts from *CIE L*C*Hab* colourspace to *CIE L*a*b** 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 CIE 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Lab(XYZ)
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 CIE 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

[]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_XYZ(Lab)
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] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [0, 1]

References

[]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_LCHab(Lab)
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] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [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

[]

Examples

```
>>> import numpy as np
>>> LCHab = np.array([41.52787529, 59.12425901, 27.08848784])
>>> LCHab_to_Lab(LCHab)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE L*u*v* Colourspace

colour

XYZ_to_Luv(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.
Luv_to_XYZ(Luv[, illuminant])	Converts from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.
Luv_to_LCHuv(Luv)	Converts from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.
LCHuv_to_Luv(LCHuv)	Converts from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.
Luv_to_uv(Luv[, illuminant])	Returns the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.
uv_to_Luv(uv[, illuminant, Y])	Returns the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given *L Lightness*.
Luv_uv_to_xy(uv)	Returns the *CIE xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.
xy_to_Luv_uv(xy)	Returns the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given *CIE xy* 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 CIE 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

[], []

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Luv(XYZ)
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 CIE 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

[], []

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_XYZ(Luv)
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] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [0, 1]

References

[]

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_LCHuv(Luv)
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] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [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

[]

Examples

```
>>> import numpy as np
>>> LCHuv = np.array([41.52787529, 98.44997950, 10.38816348])
>>> LCHuv_to_Luv(LCHuv)
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 CIE 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

[]

Examples

```
>>> import numpy as np
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_uv(Luv)
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 CIE 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

[]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.50120264])
>>> uv_to_Luv(uv)
array([ 100.        ,  233.1837603...,   42.7474385...])
```

colour.Luv_uv_to_xy

colour.**Luv_uv_to_xy**(*uv*)

Returns the *CIE 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 *CIE xy* chromaticity coordinates.

Return type ndarray

References

[]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.50120264])
>>> Luv_uv_to_xy(uv)
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 *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns *CIE L*u*v* u"v"* chromaticity coordinates.

Return type ndarray

References

[]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.54369558, 0.32107944])
>>> xy_to_Luv_uv(xy)
array([ 0.3772021...,  0.5012026...])
```

CIE 1960 UCS Colourspace

colour

XYZ_to_UCS(XYZ)	Converts from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.
UCS_to_XYZ(UVW)	Converts from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.
UCS_to_uv(UVW)	Returns the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.
uv_to_UCS(uv[, V])	Returns the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.
UCS_uv_to_xy(uv)	Returns the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.
xy_to_UCS_uv(xy)	Returns the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *CIE 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

[], []

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_UCS(XYZ)
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

[], []

Examples

```
>>> import numpy as np
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_XYZ(UVW)
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

[]

Examples

```
>>> import numpy as np
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_uv(UVW)
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

[]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.33413508])
>>> uv_to_UCS(uv)
array([ 1.1288911...,  1.        ,  0.8639104...])
```

colour.UCS_uv_to_xy

colour.**UCS_uv_to_xy**(*uv*)

Returns the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.

Parameters **uv** (array_like) – *CIE UCS uv* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

References

[]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.37720213, 0.33413508])
>>> UCS_uv_to_xy(uv)
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 *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns *CIE UCS uv* chromaticity coordinates.

Return type ndarray

References

[]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.54369555, 0.32107941])
>>> xy_to_UCS_uv(xy)
array([ 0.3772021...,  0.3341350...])
```

CIE 1964 U*V*W* Colourspace

colour

XYZ_to_UVW(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.
UVW_to_XYZ(UVW[, illuminant])	Converts *CIE 1964 U*V*W** colourspace to *CIE XYZ* 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 CIE 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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_UVW(XYZ)
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 CIE 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

[]

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

XYZ_to_Hunter_Lab(XYZ[, XYZ_n, K_ab])	Converts from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.
Hunter_Lab_to_XYZ(Lab[, XYZ_n, K_ab])	Converts from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.
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.

colour.XYZ_to_Hunter_Lab

colour.**XYZ_to_Hunter_Lab**(*XYZ*, *XYZ_n=array([95.02, 100.0, 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

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Lab(XYZ, D65.XYZ_n, D65.K_ab)
array([ 34.9245257...,  47.0618985...,  14.3861510...])
```

colour.Hunter_Lab_to_XYZ

colour.**Hunter_Lab_to_XYZ**(*Lab*, *XYZ_n=array([95.02, 100.0, 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

[]

Examples

```
>>> Lab = np.array([34.92452577, 47.06189858, 14.38615107])
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB[
...     '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

[]

Examples

```
>>> XYZ = np.array([109.850, 100.000, 35.585])
>>> XYZ_to_K_ab_HunterLab1966(XYZ)
array([ 185.2378721...,   38.4219142...])
```

Hunter Rd,a,b Colour Scale

colour

XYZ_to_Hunter_Rdab(XYZ[, XYZ_n, K_ab])	Converts from *CIE XYZ* tristimulus values to *Hunter Rd,a,b* colour scale.
Hunter_Rdab_to_XYZ(R_d_ab[, XYZ_n, K_ab])	Converts from *Hunter Rd,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, 100.0, 108.82])*, *K_ab=array([172.3, 67.2])*)

Converts from *CIE XYZ* tristimulus values to *Hunter Rd,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 Rd,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

[]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Rdab(XYZ, D65.XYZ_n, D65.K_ab)
...
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, 100.0, 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

Domain	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

[]

Examples

```
>>> import numpy as np
>>> R_d_ab = np.array([12.19722500, 57.12537874, 17.46241341])
>>> D65 = TVS_ILLUMINANTS_HUNTERLAB[
...     '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

Lab_to_DIN99(Lab[, k_E, k_CH])	Converts from *CIE L*a*b** colourspace to *DIN99* colourspace.
DIN99_to_Lab(Lab_99[, k_E, k_CH])	Converts from *DIN99* colourspace to *CIE L*a*b** 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

[]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_DIN99(Lab)
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

Domain	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

[]

Examples

```
>>> import numpy as np
>>> Lab_99 = np.array([53.22821988, 28.41634656, 3.89839552])
>>> DIN99_to_Lab(Lab_99)
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 *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.
CAM02LCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.
JMh_CIECAM02_to_CAM02SCD(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.
CAM02SCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.
JMh_CIECAM02_to_CAM02UCS(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.
CAM02UCS_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh 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

[]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     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 = VIEWING_CONDITIONS_CIECAM02['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)
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

Domain	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

[]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08450395, -0.06854831])
>>> CAM02LCD_to_JMh_CIECAM02(Jpapbp)
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

[]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     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 = VIEWING_CONDITIONS_CIECAM02['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)
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

Domain	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

[]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08436178, -0.06843298])
>>> CAM02SCD_to_JMh_CIECAM02(Jpapbp)
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
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

[]

Examples

```
>>> from colour.appearance import (
...     VIEWING_CONDITIONS_CIECAM02,
...     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 = VIEWING_CONDITIONS_CIECAM02['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02UCS(JMh)
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

Domain	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

[]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08442362, -0.06848314])
>>> CAM02UCS_to_JMh_CIECAM02(Jpapbp)
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, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.
CAM16LCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16SCD(JMh, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.
CAM16SCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.
JMh_CAM16_to_CAM16UCS(JMh, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.
CAM16UCS_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al. (2017) 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

[]

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

Domain	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

[]

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

[]

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

Domain	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

[]

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

[]

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

Domain	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

[]

$I_GP_GT_G$ Colourspace

colour

XYZ_to_IGPGTG(XYZ)	Converts from *CIE XYZ* tristimulus values to $I_GP_GT_G$ colourspace.
IGPGTG_to_XYZ(IGPGTG)	Converts from $I_GP_GT_G$ colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_IGPGTG

colour.**XYZ_to_IGPGTG**(*XYZ*)

Converts from *CIE XYZ* tristimulus values to $I_GP_GT_G$ colourspace.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values.

Returns $I_GP_GT_G$ colourspace array.

Return type ndarray

Notes

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

Range	Scale - Reference	Scale - 1
IGPGTG	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IGPGTG(XYZ)
array([ 0.4242125...,  0.1863249...,  0.1068922...])
```

colour.IGPGTG_to_XYZ

colour.**IGPGTG_to_XYZ**(*IGPGTG*)

Converts from $I_GP_GT_G$ colourspace to *CIE XYZ* tristimulus values.

Parameters **IGPGTG** (array_like) – $I_GP_GT_G$ colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
IGPGTG	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]	IG : [0, 1] PG : [-1, 1] TG : [-1, 1]

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

References

[]

Examples

```
>>> IGPGTG = np.array([0.42421258, 0.18632491, 0.10689223])
>>> IGPGTG_to_XYZ(IGPGTG)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

IPT Colourspace

colour

XYZ_to_IPT(XYZ)	Converts from *CIE XYZ* tristimulus values to *IPT* colourspace.
IPT_to_XYZ(IPT)	Converts from *IPT* colourspace to *CIE XYZ* tristimulus values.
IPT_hue_angle(IPT)	Computes the hue angle in degrees from *IPT* 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] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

- Input *CIE XYZ* tristimulus values must be adapted to *CIE Standard Illuminant D Series D65*.

References

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IPT(XYZ)
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] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

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

References

[]

Examples

```
>>> IPT = np.array([0.38426191, 0.38487306, 0.18886838])
>>> IPT_to_XYZ(IPT)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.IPT_hue_angle

colour.**IPT_hue_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

[]

Examples

```
>>> IPT = np.array([0.96907232, 1, 1.12179215])
>>> IPT_hue_angle(IPT)
48.2852074...
```

hdr-CIELAB Colourspace

colour

XYZ_to_hdr_CIELab(XYZ[, illuminant, Y_s, ...])	Converts from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.
hdr_CIELab_to_XYZ(Lab_hdr[, illuminant, ...])	Converts from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.
HDR_CIELAB_METHODS	Supported *hdr-CIELAB* 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 CIE 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

[], []

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_CIELab(XYZ)
array([ 51.8700206...,  60.4763385...,  32.1455191...])
>>> XYZ_to_hdr_CIELab(XYZ, method='Fairchild 2010')
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 CIE 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

[], []

Examples

```
>>> Lab_hdr = np.array([51.87002062, 60.4763385, 32.14551912])
>>> hdr_CIELab_to_XYZ(Lab_hdr)
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')
...
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

[], []

HDR_CIELAB_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

hdr-IPT Colourspace

colour

XYZ_to_hdr_IPT(XYZ[, Y_s, Y_abs, method])	Converts from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.
hdr_IPT_to_XYZ(IPT_hdr[, Y_s, Y_abs, method])	Converts from *hdr-IPT* colourspace to *CIE XYZ* tristimulus values.
HDR_IPT_METHODS	Supported *hdr-IPT* 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 must be adapted to *CIE Standard Illuminant D Series D65*.

References

[], []

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_IPT(XYZ)
array([ 48.3937634...,  42.4499020...,  22.0195403...])
>>> XYZ_to_hdr_IPT(XYZ, method='Fairchild 2010')
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

Domain	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

[], []

Examples

```
>>> IPT_hdr = np.array([48.39376346, 42.44990202, 22.01954033])
>>> hdr_IPT_to_XYZ(IPT_hdr)
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')
...
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

[], []

HDR_IPT_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

OSA UCS Colourspace

colour

XYZ_to_OSA_UCS(XYZ)	Converts from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.
OSA_UCS_to_XYZ(Ljg[, optimisation_kwargs])	Converts from *OSA UCS* colourspace to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

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

[], []

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_OSA_UCS(XYZ)
array([-3.0049979...,  2.9971369..., -9.6678423...])
```

colour.OSA_UCS_to_XYZ

colour.**OSA_UCS_to_XYZ**(*Ljg*, *optimisation_kwargs=None*, ***kwargs*)

Converts from *OSA UCS* colourspace 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_kwargs** (dict_like, optional) – Parameters for scipy.optimize.fmin() definition.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Return type ndarray

Warning: There is no analytical inverse 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

[], []

Examples

```
>>> import numpy as np
>>> Ljg = np.array([-3.00499790, 2.99713697, -9.66784231])
>>> OSA_UCS_to_XYZ(Ljg)
array([ 20.6540240...,  12.1972369...,   5.1369372...])
```

$JzAzBz$ Colourspace

colour

XYZ_to_JzAzBz(XYZ_D65[, constants])	Converts from *CIE XYZ* tristimulus values to $J_zA_zB_z$ colourspace.
JzAzBz_to_XYZ(JzAzBz[, constants])	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

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function 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. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF / EOCF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
XYZ	UN	UN

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

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_JzAzBz(XYZ)
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 $J_zA_zB_z$ colourspace to *CIE XYZ* tristimulus values.

Parameters

- **JzAzBz** (array_like) – $J_zA_zB_z$ colourspace array where J_z is Lightness, A_z is redness-greenness and B_z is yellowness-blueness.
- **constants** (Structure, optional) – $J_zA_zB_z$ colourspace constants.

Returns *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.

Return type ndarray

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function 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.

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	UN	UN

References

[]

Examples

```
>>> JzAzBz = np.array([0.00535048, 0.00924302, 0.00526007])
>>> JzAzBz_to_XYZ(JzAzBz)
array([ 0.2065402...,  0.1219723...,  0.0513696...])
```

RGB Colourspace and Transformations

colour

XYZ_to_RGB(XYZ, illuminant_XYZ, ...[, ...])	Converts from *CIE XYZ* tristimulus values to *RGB* colourspace array.
RGB_to_XYZ(RGB, illuminant_RGB, ...[, ...])	Converts given *RGB* colourspace array to *CIE XYZ* tristimulus values.
RGB_to_RGB(RGB, input_colourspace, ...[, ...])	Converts given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.
matrix_RGB_to_RGB(input_colourspace, ...[, ...])	Computes the matrix M converting from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

colour.XYZ_to_RGB

colour.**XYZ_to_RGB**(*XYZ*, *illuminant_XYZ*, *illuminant_RGB*, *matrix_XYZ_to_RGB*, *chromatic_adaptation_transform='CAT02'*, *cctf_encoding=None*, ***kwargs*)

Converts from *CIE XYZ* tristimulus values to *RGB* colourspace array.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant_XYZ** (array_like) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the input *CIE XYZ* tristimulus values.

- **illuminant_RGB** (array_like) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the output *RGB* colourspace array.
- **matrix_XYZ_to_RGB** (array_like) – Matrix converting the *CIE XYZ* tristimulus values to *RGB* colourspace array, i.e. the inverse *Normalised Primary Matrix* (NPM).
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **cctf_encoding** (object, optional) – Encoding colour component transfer function (Encoding CCTF) or opto-electronic transfer function (OETF / OECF).
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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'
>>> matrix_XYZ_to_RGB = 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, matrix_XYZ_to_RGB,
...            chromatic_adaptation_transform)
array([ 0.4559557...,  0.0303970...,  0.0408724...])
```

colour.RGB_to_XYZ

colour.**RGB_to_XYZ**(*RGB*, *illuminant_RGB*, *illuminant_XYZ*, *matrix_RGB_to_XYZ*, *chromatic_adaptation_transform='CAT02'*, *cctf_decoding=None*, ***kwargs*)

Converts given *RGB* colourspace array to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **illuminant_RGB** (array_like) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the input *RGB* colourspace array.
- **illuminant_XYZ** (array_like) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array of the *illuminant* for the output *CIE XYZ* tristimulus values.
- **matrix_RGB_to_XYZ** (array_like) – Matrix converting the *RGB* colourspace array to *CIE XYZ* tristimulus values, i.e. the *Normalised Primary Matrix* (NPM).
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **cctf_decoding** (object, optional) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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'
>>> matrix_RGB_to_XYZ = 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, matrix_RGB_to_XYZ,
...            chromatic_adaptation_transform)
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_cctf_decoding=False*, *apply_cctf_encoding=False*, ***kwargs*)

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 2008', 'Bianco 2010', 'Bianco PC 2010', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_cctf_decoding** (bool, optional) – Apply input colourspace decoding colour component transfer function / electro-optical transfer function.
- **apply_cctf_encoding** (bool, optional) – Apply output colourspace encoding colour component transfer function / opto-electronic transfer function.
- ****kwargs** (dict, optional) – Keywords arguments for the colour component transfer functions.

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 (
...     RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> RGB_to_RGB(RGB, RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
...
array([ 0.2568891...,  0.0721446...,  0.0465553...])
```

colour.matrix_RGB_to_RGB

colour.**matrix_RGB_to_RGB**(*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 2008', 'Bianco 2010', 'Bianco PC 2010', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.

Returns Conversion matrix M.

Return type ndarray

Examples

```
>>> from colour.models import (
...     RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
>>> matrix_RGB_to_RGB(RGB_COLOURSPACE_sRGB, RGB_COLOURSPACE_PROPHOTO_RGB)
...
array([[ 0.5288241...,  0.3340609...,  0.1373616...],
       [ 0.0975294...,  0.8790074...,  0.0233981...],
       [ 0.0163599...,  0.1066124...,  0.8772485...]])
```

Ancillary Objects

colour

XYZ_to_sRGB(XYZ[, illuminant, ...])	Converts from *CIE XYZ* tristimulus values to *sRGB* colourspace.
sRGB_to_XYZ(RGB[, illuminant, ...])	Converts from *sRGB* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_sRGB

colour.**XYZ_to_sRGB**(*XYZ*, *illuminant=array([0.3127, 0.329])*, *chromatic_adaptation_transform='CAT02'*, *apply_cctf_encoding=True*, ***kwargs*)

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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *Chromatic adaptation* transform.
- **apply_cctf_encoding** (bool, optional) – Apply *sRGB* encoding colour component transfer function / opto-electronic transfer function.

- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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)
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_cctf_decoding=True*, ***kwargs*)

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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *Chromatic adaptation* method.
- **apply_cctf_decoding** (bool, optional) – Apply *sRGB* decoding colour component transfer function / electro-optical transfer function.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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)
array([ 0.2065429...,  0.1219794...,  0.0513714...])
```

RGB Colourspace Derivation

colour

normalised_primary_matrix(primaries, whitepoint)	Computes the *Normalised Primary Matrix* (NPM) converting a *RGB* colourspace array to *CIE XYZ* tristimulus values using given *primaries* and *whitepoint* xy chromaticity coordinates.
chromatically_adapted_primaries(primaries, ...)	Chromatically adapts given *primaries* xy chromaticity coordinates from test whitepoint_t to reference whitepoint_r.
primaries_whitepoint(npm)	Computes the *primaries* and *whitepoint* xy chromaticity coordinates using given *Normalised Primary Matrix* (NPM).
RGB_luminance(RGB, primaries, whitepoint)	Returns the *luminance* Y of given *RGB* components from given *primaries* and *whitepoint*.
RGB_luminance_equation(primaries, whitepoint)	Returns the *luminance equation* from given *primaries* and *whitepoint*.

colour.normalised_primary_matrix

colour.**normalised_primary_matrix**(*primaries*, *whitepoint*)

Computes the *Normalised Primary Matrix* (NPM) converting a *RGB* colourspace array to *CIE XYZ* tristimulus values 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* (NPM).

Return type ndarray, (3, 3)

References

[]

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)
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 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *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)
...
array([[ 0.6484414...,  0.3308533...],
       [ 0.3211951...,  0.5978443...],
       [ 0.1558932...,  0.0660492...]])
```

colour.primaries_whitepoint

colour.**primaries_whitepoint**(*npm*)

Computes the *primaries* and *whitepoint* xy chromaticity coordinates using given *Normalised Primary Matrix* (NPM).

Parameters **npm** (array_like, (3, 3)) – *Normalised Primary Matrix*.

Returns *Primaries* and *whitepoint* xy chromaticity coordinates.

Return type tuple

References

[]

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 = primaries_whitepoint(npm)
>>> p
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
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)
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)
'Y = 0.3439664...(R) + 0.7281660...(G) + -0.0721325...(B)'
```

RGB Colourspaces

colour

RGB_Colourspace(name, primaries, whitepoint)	Implements support for the *RGB* colourspaces datasets from colour.models.datasets.aces_rgb, etc....

colour.RGB_Colourspace

class colour.**RGB_Colourspace**(*name*, *primaries*, *whitepoint*, *whitepoint_name=None*, *matrix_RGB_to_XYZ=None*, *matrix_XYZ_to_RGB=None*, *cctf_encoding=None*, *cctf_decoding=None*, *use_derived_matrix_RGB_to_XYZ=False*, *use_derived_matrix_XYZ_to_RGB=False*, ***kwargs*)

Bases: object

Implements support for the *RGB* colourspaces datasets from colour.models.datasets.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 `matrix_RGB_to_XYZ` and `matrix_XYZ_to_RGB` 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_matrix_RGB_to_XYZ` and `colour.RGB_Colourspace.use_derived_matrix_XYZ_to_RGB` attribute 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.
- **matrix_RGB_to_XYZ** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **matrix_XYZ_to_RGB** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **cctf_encoding** (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.
- **cctf_decoding** (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_matrix_RGB_to_XYZ** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_matrix_XYZ_to_RGB** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Attributes

- name
- primaries
- whitepoint
- whitepoint_name
- matrix_RGB_to_XYZ
- matrix_XYZ_to_RGB
- cctf_encoding
- cctf_decoding
- use_derived_matrix_RGB_to_XYZ
- use_derived_matrix_XYZ_to_RGB

Methods

- __init__
- __str__
- __repr__
- use_derived_transformation_matrices
- chromatically_adapt
- copy

Notes

- The normalised primary matrix defined by colour.RGB_Colourspace.matrix_RGB_to_XYZ 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

[], []

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                               matrix_RGB_to_XYZ, matrix_XYZ_to_RGB)
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.matrix_XYZ_to_RGB
```

(continues on next page)

(continued from previous page)

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
>>> colourspace.matrix_RGB_to_XYZ
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.matrix_XYZ_to_RGB
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_matrix_RGB_to_XYZ = False
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_matrix_XYZ_to_RGB = False
>>> colourspace.matrix_XYZ_to_RGB
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

__init__(*name*, *primaries*, *whitepoint*, *whitepoint_name=None*, *matrix_RGB_to_XYZ=None*, *matrix_XYZ_to_RGB=None*, *cctf_encoding=None*, *cctf_decoding=None*, *use_derived_matrix_RGB_to_XYZ=False*, *use_derived_matrix_XYZ_to_RGB=False*, ***kwargs*)

property name

Getter and setter property for the name.

Parameters **value** (unicode) – Value to set the name with.

Returns *RGB* colourspace name.

Return type unicode

property primaries

Getter and setter property for the primaries.

Parameters **value** (array_like) – Value to set the primaries with.

Returns *RGB* colourspace primaries.

Return type array_like

property whitepoint

Getter and setter property for the whitepoint.

Parameters **value** (array_like) – Value to set the whitepoint with.

Returns *RGB* colourspace whitepoint.

Return type array_like

property whitepoint_name

Getter and setter property for the whitepoint_name.

Parameters **value** (unicode) – Value to set the whitepoint_name with.

Returns *RGB* colourspace whitepoint name.

Return type unicode

property matrix_RGB_to_XYZ

Getter and setter property for the transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Parameters **value** (array_like) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Returns Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Return type array_like

property matrix_XYZ_to_RGB

Getter and setter property for the transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Parameters **value** (array_like) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Returns Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Return type array_like

property cctf_encoding

Getter and setter property for the encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Parameters **value** (callable) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Returns Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Return type callable

property cctf_decoding

Getter and setter property for the decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Parameters **value** (callable) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Returns Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Return type callable

__weakref__

list of weak references to the object (if defined)

property use_derived_matrix_RGB_to_XYZ

Getter and setter property for whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Parameters **value** (bool) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Returns Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Return type bool

property use_derived_matrix_XYZ_to_RGB

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.

Parameters **value** (bool) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Returns Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Return type bool

__str__()

Returns a formatted string representation of the *RGB* colourspace.

Returns Formatted string representation.

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])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> cctf_encoding = lambda x: x
>>> cctf_decoding = lambda x: x
>>> print(RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                       matrix_RGB_to_XYZ, matrix_XYZ_to_RGB,
...                       cctf_encoding, cctf_decoding))
...
RGB Colourspace
---------------

Primaries          : [[  7.34700000e-01   2.65300000e-01]
                      [  0.00000000e+00   1.00000000e+00]
                      [  1.00000000e-04  -7.70000000e-02]]
Whitepoint         : [ 0.32168  0.33767]
Whitepoint Name    : ACES
Encoding CCTF      : <function <lambda> at 0x...>
Decoding CCTF      : <function <lambda> at 0x...>
NPM                : [[ 1.  0.  0.]
                      [ 0.  1.  0.]
                      [ 0.  0.  1.]]
NPM -1             : [[ 1.  0.  0.]
                      [ 0.  1.  0.]
                      [ 0.  0.  1.]]
Derived NPM        : [[  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]]
Derived NPM -1     : [[  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]]
Use Derived NPM    : False
Use Derived NPM -1 : False
```

__repr__()

Returns an (almost) evaluable string representation of the *RGB* colourspace.

Returns (Almost) evaluable string representation.

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])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> cctf_encoding = lambda x: x
>>> cctf_decoding = lambda x: x
>>> RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                 matrix_RGB_to_XYZ, matrix_XYZ_to_RGB,
...                 cctf_encoding, cctf_decoding)
...
RGB_Colourspace(RGB Colourspace,
                [[  7.34700000e-01,   2.65300000e-01],
                 [  0.00000000e+00,   1.00000000e+00],
                 [  1.00000000e-04,  -7.70000000e-02]],
                [ 0.32168,  0.33767],
                ACES,
                [[ 1.,  0.,  0.],
                 [ 0.,  1.,  0.],
                 [ 0.,  0.,  1.]],
                [[ 1.,  0.,  0.],
                 [ 0.,  1.,  0.],
                 [ 0.,  0.,  1.]],
                <function <lambda> at 0x...>,
                <function <lambda> at 0x...>,
                False,
                False)
```

use_derived_transformation_matrices(*usage=True*)

Enables or disables usage of both derived transformations matrices, the normalised primary matrix and its inverse in subsequent computations.

Parameters **usage** (bool, optional) – Whether to use the derived transformations matrices.

Returns Definition success.

Return type bool

chromatically_adapt(*whitepoint*, *whitepoint_name=None*, *chromatic_adaptation_transform='CAT02'*)

Chromatically adapts the *RGB* colourspace *primaries* xy chromaticity coordinates from *RGB* colourspace whitepoint to reference `whitepoint`.

Parameters

- **whitepoint** (array_like) – Reference illuminant / whitepoint xy chromaticity coordinates.
- **whitepoint_name** (unicode, optional) – Reference illuminant / whitepoint name.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *Chromatic adaptation* transform.

Return type Chromatically adapted *RGB* colourspace.

Examples

```
>>> p = np.array(
...     [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> w_t = np.array([0.32168, 0.33767])
>>> w_r = np.array([0.31270, 0.32900])
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, w_t, 'D65')
>>> print(colourspace.chromatically_adapt(w_r, 'D50', 'Bradford'))
...
RGB Colourspace - Chromatically Adapted to D50
----------------------------------------------

Primaries          : [[ 0.73485524  0.26422533]
                      [-0.00617091  1.01131496]
                      [ 0.01596756 -0.0642355 ]]
Whitepoint         : [ 0.3127  0.329 ]
Whitepoint Name    : D50
Encoding CCTF      : None
Decoding CCTF      : None
NPM                : None
NPM -1             : None
Derived NPM        : [[ 0.93827985 -0.00445145  0.01662752]
                      [ 0.33736889  0.72952157 -0.06689046]
                      [ 0.00117395 -0.00371071  1.09159451]]
Derived NPM -1     : [[ 1.06349549  0.00640891 -0.01580679]
                      [-0.49207413  1.36822341  0.09133709]
                      [-0.00281646  0.00464417  0.91641857]]
Use Derived NPM    : True
Use Derived NPM -1 : True
```

copy()

Returns a copy of the *RGB* colourspace.

Returns *RGB* colourspace copy.

Return type *RGB_Colourspace*

RGB_COLOURSPACES	Aggregated *RGB* colourspaces.

colour.RGB_COLOURSPACES

colour.**RGB_COLOURSPACES = CaseInsensitiveMapping({'ACES2065-1': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ACEScg': ..., 'Adobe RGB (1998)': ..., 'Adobe Wide Gamut RGB': ..., 'Apple RGB': ..., 'ALEXA Wide Gamut': ..., 'Best RGB': ..., 'Beta RGB': ..., 'ITU-R BT.470 - 525': ..., 'ITU-R BT.470 - 625': ..., 'ITU-R BT.709': ..., 'ITU-R BT.2020': ..., 'CIE RGB': ..., 'Cinema Gamut': ..., 'ColorMatch RGB': ..., 'DaVinci Wide Gamut': ..., 'DCDM XYZ': ..., 'DCI-P3': ..., 'DCI-P3+': ..., 'Display P3': ..., 'DJI D-Gamut': ..., 'Don RGB 4': ..., 'ECI RGB v2': ..., 'Ekta Space PS 5': ..., 'FilmLight E-Gamut': ..., 'Protune Native': ..., 'Max RGB': ..., 'P3-D65': ..., 'Pal/Secam': ..., 'REDcolor': ..., 'REDcolor2': ..., 'REDcolor3': ..., 'REDcolor4': ..., 'REDWideGamutRGB': ..., 'DRAGONcolor': ..., 'DRAGONcolor2': ..., 'ROMM RGB': ..., 'RIMM RGB': ..., 'ERIMM RGB': ..., 'F-Gamut': ..., 'ProPhoto RGB': ..., 'Russell RGB': ..., 'Sharp RGB': ..., 'SMPTE 240M': ..., 'SMPTE C': ..., 'NTSC (1953)': ..., 'NTSC (1987)': ..., 'S-Gamut': ..., 'S-Gamut3': ..., 'S-Gamut3.Cine': ..., 'Venice S-Gamut3': ..., 'Venice S-Gamut3.Cine': ..., 'sRGB': ..., 'V-Gamut': ..., 'Xtreme RGB': ..., 'aces': ..., 'adobe1998': ..., 'prophoto': ...})**

Aggregated *RGB* colourspaces.

RGB_COLOURSPACES : CaseInsensitiveMapping

Aliases:

- 'aces': RGB_COLOURSPACE_ACES2065_1.name
- 'adobe1998': RGB_COLOURSPACE_ADOBE_RGB1998.name
- 'prophoto': RGB_COLOURSPACE_PROPHOTO_RGB.name

`colour.models`

RGB_COLOURSPACE_ACES2065_1	*ACES2065-1* colourspace, base encoding, used for exchange of full fidelity images and archiving.
RGB_COLOURSPACE_ACESCC	*ACEScc* colourspace, a working space for color correctors, target for ASC-CDL values created onset.
RGB_COLOURSPACE_ACESCCT	*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.
RGB_COLOURSPACE_ACESPROXY	*ACESproxy* colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management.
RGB_COLOURSPACE_ACESCG	*ACEScg* colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.
RGB_COLOURSPACE_ADOBE_RGB1998	*Adobe RGB (1998)* colourspace.
RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB	*Adobe Wide Gamut RGB* colourspace.
RGB_COLOURSPACE_ALEXA_WIDE_GAMUT	*ARRI ALEXA Wide Gamut* colourspace.
RGB_COLOURSPACE_APPLE_RGB	*Apple RGB* colourspace.
RGB_COLOURSPACE_BEST_RGB	*Best RGB* colourspace.
RGB_COLOURSPACE_BETA_RGB	*Beta RGB* colourspace.
RGB_COLOURSPACE_BT470_525	*ITU-R BT.470 - 525* colourspace.
RGB_COLOURSPACE_BT470_625	*ITU-R BT.470 - 625* colourspace.
RGB_COLOURSPACE_BT709	*ITU-R BT.709* colourspace.
RGB_COLOURSPACE_BT2020	*ITU-R BT.2020* colourspace.
RGB_COLOURSPACE_CIE_RGB	*CIE RGB* colourspace.
RGB_COLOURSPACE_CINEMA_GAMUT	*Canon Cinema Gamut* colourspace.
RGB_COLOURSPACE_COLOR_MATCH_RGB	*ColorMatch RGB* colourspace.
RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT	*DaVinci Wide Gamut* colourspace.
RGB_COLOURSPACE_DCDM_XYZ	*DCDM XYZ* colourspace.
RGB_COLOURSPACE_DCI_P3	*DCI-P3* colourspace.
RGB_COLOURSPACE_DCI_P3_P	*DCI-P3+* colourspace.
RGB_COLOURSPACE_DISPLAY_P3	*Display P3* colourspace.
RGB_COLOURSPACE_DON_RGB_4	*Don RGB 4* colourspace.
RGB_COLOURSPACE_ECI_RGB_V2	*ECI RGB v2* colourspace.
RGB_COLOURSPACE_EKTA_SPACE_PS_5	*Ekta Space PS 5* colourspace.
RGB_COLOURSPACE_F_GAMUT	*Fujifilm F-Gamut* colourspace.
RGB_COLOURSPACE_PROTUNE_NATIVE	*Protune Native* colourspace.
RGB_COLOURSPACE_MAX_RGB	*Max RGB* colourspace.
RGB_COLOURSPACE_NTSC1953	*NTSC (1953)* colourspace.
RGB_COLOURSPACE_NTSC1987	*NTSC (1987)* colourspace.
RGB_COLOURSPACE_P3_D65	*P3-D65* colourspace.
RGB_COLOURSPACE_PAL_SECAM	*Pal/Secam* colourspace.
RGB_COLOURSPACE_RED_COLOR	*REDcolor* colourspace.

continues on next page

Table 1 – continued from previous page

RGB_COLOURSPACE_RED_COLOR_2	*REDcolor2* colourspace.
RGB_COLOURSPACE_RED_COLOR_3	*REDcolor3* colourspace.
RGB_COLOURSPACE_RED_COLOR_4	*REDcolor4* colourspace.
RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB	*REDWideGamutRGB* colourspace.
RGB_COLOURSPACE_DRAGON_COLOR	*DRAGONcolor* colourspace.
RGB_COLOURSPACE_DRAGON_COLOR_2	*DRAGONcolor2* colourspace.
RGB_COLOURSPACE_ROMM_RGB	*ROMM RGB* colourspace.
RGB_COLOURSPACE_RIMM_RGB	*RIMM RGB* colourspace.
RGB_COLOURSPACE_ERIMM_RGB	*ERIMM RGB* colourspace.
RGB_COLOURSPACE_PROPHOTO_RGB	*ProPhoto RGB* colourspace, an alias colourspace for *ROMM RGB*.
RGB_COLOURSPACE_RUSSELL_RGB	*Russell RGB* colourspace.
RGB_COLOURSPACE_SMPTE_240M	*SMPTE 240M* colourspace.
RGB_COLOURSPACE_SMPTE_C	Implements support for the *RGB* colourspaces datasets from `colour.models.datasets.aces_rgb`, etc....
RGB_COLOURSPACE_S_GAMUT	*S-Gamut* colourspace.
RGB_COLOURSPACE_S_GAMUT3	*S-Gamut3* colourspace.
RGB_COLOURSPACE_S_GAMUT3_CINE	*S-Gamut3.Cine* colourspace.
RGB_COLOURSPACE_VENICE_S_GAMUT3	*Venice S-Gamut3* colourspace.
RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE	*Venice S-Gamut3.Cine* colourspace.
RGB_COLOURSPACE_sRGB	*sRGB* colourspace.
RGB_COLOURSPACE_V_GAMUT	*Panasonic V-Gamut* colourspace.
RGB_COLOURSPACE_XTREME_RGB	*Xtreme RGB* colourspace.

colour.models.RGB_COLOURSPACE_ACES2065_1

colour.models.**RGB_COLOURSPACE_ACES2065_1 = RGB_Colourspace(ACES2065-1, [[7.34700000e-01, 2.65300000e-01], [0.00000000e+00, 1.00000000e+00], [1.00000000e-04, -7.70000000e-02]], [0.32168, 0.33767], ACES, [[9.52552396e-01, 0.00000000e+00, 9.36786000e-05], [3.43966450e-01, 7.28166097e-01, -7.21325464e-02], [0.00000000e+00, 0.00000000e+00, 1.00882518e+00]], [[1.04981102e+00, 0.00000000e+00, -9.74845000e-05], [-4.95903023e-01, 1.37331305e+00, 9.82400361e-02], [0.00000000e+00, 0.00000000e+00, 9.91252018e-01]], <function linear_function>, <function linear_function>, False, False)**

ACES2065-1 colourspace, base encoding, used for exchange of full fidelity images and archiving.

References

[], [], []

RGB_COLOURSPACE_ACES2065_1 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ACESCC

colour.models.**RGB_COLOURSPACE_ACESCC = RGB_Colourspace(ACEScc, [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], ACES, [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], <function log_encoding_ACEScc>, <function log_decoding_ACEScc>, False, False)**

ACEScc colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

References

[], [], [], []

RGB_COLOURSPACE_ACESCC : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ACESCCT

colour.models.**RGB_COLOURSPACE_ACESCCT = RGB_Colourspace(ACEScct, [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], ACES, [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], <function log_encoding_ACEScct>, <function log_decoding_ACEScct>, False, False)**

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

[], [], [], []

RGB_COLOURSPACE_ACESCCT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ACESPROXY

colour.models.**RGB_COLOURSPACE_ACESPROXY = RGB_Colourspace(ACESproxy, [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], ACES, [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], <function log_encoding_ACESproxy>, <function log_decoding_ACESproxy>, False, False)**

ACESproxy colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management. Not intended to be stored or used in production imagery or for final colour grading / mastering.

References

[], [], [], []

RGB_COLOURSPACE_ACESPROXY : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ACESCG

colour.models.**RGB_COLOURSPACE_ACESCG = RGB_Colourspace(ACEScg, [[0.713, 0.293], [0.165, 0.83], [0.128, 0.044]], [0.32168, 0.33767], ACES, [[0.66245418, 0.13400421, 0.15618769], [0.27222872, 0.67408177, 0.05368952], [-0.00557465, 0.00406073, 1.0103391]], [[1.64102338, -0.32480329, -0.2364247], [-0.66366286, 1.61533159, 0.01675635], [0.01172189, -0.00828444, 0.98839486]], <function linear_function>, <function linear_function>, False, False)**

ACEScg colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.

References

[], [], [], []

RGB_COLOURSPACE_ACESCG : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ADOBE_RGB1998

colour.models.**RGB_COLOURSPACE_ADOBE_RGB1998 = RGB_Colourspace(Adobe RGB (1998), [[0.64, 0.33], [0.21, 0.71], [0.15, 0.06]], [0.3127, 0.329], D65, [[0.57667, 0.18556, 0.18823], [0.29734, 0.62736, 0.07529], [0.02703, 0.07069, 0.99134]], [[2.04159, -0.56501, -0.34473], [-0.96924, 1.87597, 0.04156], [0.01344, -0.11836, 1.01517]], functools.partial(<function gamma_function>, exponent=0.4547069271758437), functools.partial(<function gamma_function>, exponent=2.19921875), False, False)**

Adobe RGB (1998) colourspace.

References

[]

RGB_COLOURSPACE_ADOBE_RGB1998 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB

colour.models.**RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB = RGB_Colourspace(Adobe Wide Gamut RGB, [[0.7347, 0.2653], [0.1152, 0.8264], [0.1566, 0.0177]], [0.3457, 0.3585], D50, [[0.71650072, 0.10102057, 0.14677439], [0.25872824, 0.72468231, 0.01658944], [0. , 0.05121182, 0.77389278]], [[1.46230418, -0.18452564, -0.27338105], [-0.52286828, 1.4479884 , 0.06812617], [0.03460045, -0.09581963, 1.28766046]], functools.partial(<function gamma_function>, exponent=0.4547069271758437), functools.partial(<function gamma_function>, exponent=2.19921875), False, False)**

Adobe Wide Gamut RGB colourspace.

References

[]

RGB_COLOURSPACE_ADOBE_WIDE_GAMUT_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ALEXA_WIDE_GAMUT

colour.models.**RGB_COLOURSPACE_ALEXA_WIDE_GAMUT = RGB_Colourspace(ALEXA Wide Gamut, [[0.684 , 0.313], [0.221 , 0.848], [0.0861, -0.102]], [0.3127, 0.329], D65, [[0.638008, 0.214704, 0.097744], [0.291954, 0.823841, -0.115795], [0.002798, -0.067034, 1.153294]], [[1.789066, -0.482534, -0.200076], [-0.639849, 1.3964 , 0.194432], [-0.041532, 0.082335, 0.878868]], <function log_encoding_ALEXALogC>, <function log_decoding_ALEXALogC>, False, False)**

ARRI ALEXA Wide Gamut colourspace.

References

[]

RGB_COLOURSPACE_ALEXA_WIDE_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_APPLE_RGB

colour.models.**RGB_COLOURSPACE_APPLE_RGB = RGB_Colourspace(Apple RGB, [[0.625, 0.34], [0.28 , 0.595], [0.155, 0.07]], [0.3127, 0.329], D65, [[0.44966162, 0.31625612, 0.18453819], [0.24461592, 0.67204425, 0.08333983], [0.02518105, 0.14118577, 0.92269093]], [[2.95197848, -1.2896043 , -0.47391531], [-1.08508357, 1.99080934, 0.03720168], [0.08547221, -0.26942971, 1.09102767]], functools.partial(<function gamma_function>, exponent=0.5555555555555556), functools.partial(<function gamma_function>, exponent=1.8), False, False)**

Apple RGB colourspace.

References

[]

RGB_COLOURSPACE_APPLE_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BEST_RGB

colour.models.**RGB_COLOURSPACE_BEST_RGB = RGB_Colourspace(Best RGB, [[0.73519164, 0.26480836], [0.21533613, 0.77415966], [0.13012295, 0.03483607]], [0.3457, 0.3585], D50, [[0.6318944 , 0.20538793, 0.12701335], [0.22760177, 0.73839465, 0.03400357], [0. , 0.01001892, 0.81508568]], [[1.75737181, -0.48538023, -0.25359913], [-0.54199672, 1.50475404, 0.02168337], [0.00666215, -0.01849623, 1.22659836]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Best RGB colourspace.

References

[]

RGB_COLOURSPACE_BEST_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BETA_RGB

colour.models.**RGB_COLOURSPACE_BETA_RGB = RGB_Colourspace(Beta RGB, [[0.6888, 0.3112], [0.1986, 0.7551], [0.1265, 0.0352]], [0.3457, 0.3585], D50, [[6.71355903e-01, 1.74572381e-01, 1.18367393e-01], [3.03318753e-01, 6.63744233e-01, 3.29370136e-02], [5.41053122e-17, 4.06983949e-02, 7.84406208e-01]], [[1.68297071, -0.42817109, -0.23598255], [-0.77107152, 1.70666472, 0.04469277], [0.04000653, -0.08854917, 1.27253082]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Beta RGB colourspace.

References

[]

RGB_COLOURSPACE_BETA_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BT470_525

colour.models.**RGB_COLOURSPACE_BT470_525 = RGB_Colourspace(ITU-R BT.470 - 525, [[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]], [0.31006, 0.31616], C, [[6.06863809e-01, 1.73507281e-01, 2.00334881e-01], [2.98903070e-01, 5.86619855e-01, 1.14477075e-01], [-5.02801622e-17, 6.60980118e-02, 1.11615148e+00]], [[1.91008143, -0.53247794, -0.28822201], [-0.98463135, 1.99910001, -0.02830719], [0.05830945, -0.11838584, 0.89761208]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

ITU-R BT.470 - 525 colourspace.

References

[]

RGB_COLOURSPACE_BT470_525 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BT470_625

colour.models.**RGB_COLOURSPACE_BT470_625 = RGB_Colourspace(ITU-R BT.470 - 625, [[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]], [0.3127, 0.329], D65, [[0.43055381, 0.3415498 , 0.17835231], [0.22200431, 0.70665477, 0.07134092], [0.02018221, 0.12955337, 0.93932217]], [[3.06336109, -1.39339017, -0.47582374], [-0.96924364, 1.8759675 , 0.04155506], [0.06786105, -0.22879927, 1.06908962]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

ITU-R BT.470 - 625 colourspace.

References

[]

RGB_COLOURSPACE_BT470_625 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BT709

colour.models.**RGB_COLOURSPACE_BT709 = RGB_Colourspace(ITU-R BT.709, [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]], [0.3127, 0.329], D65, [[0.4123908 , 0.35758434, 0.18048079], [0.21263901, 0.71516868, 0.07219232], [0.01933082, 0.11919478, 0.95053215]], [[3.24096994, -1.53738318, -0.49861076], [-0.96924364, 1.8759675 , 0.04155506], [0.05563008, -0.20397696, 1.05697151]], <function oetf_BT709>, <function oetf_inverse_BT709>, False, False)**

ITU-R BT.709 colourspace.

References

[]

RGB_COLOURSPACE_BT709 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_BT2020

colour.models.**RGB_COLOURSPACE_BT2020 = RGB_Colourspace(ITU-R BT.2020, [[0.708, 0.292], [0.17 , 0.797], [0.131, 0.046]], [0.3127, 0.329], D65, [[6.36958048e-01, 1.44616904e-01, 1.68880975e-01], [2.62700212e-01, 6.77998072e-01, 5.93017165e-02], [4.99410657e-17, 2.80726930e-02, 1.06098506e+00]], [[1.71665119, -0.35567078, -0.25336628], [-0.66668435, 1.61648124, 0.01576855], [0.01763986, -0.04277061, 0.94210312]], <function eotf_inverse_BT2020>, <function eotf_BT2020>, False, False)**

ITU-R BT.2020 colourspace.

The wavelength of the *ITU-R BT.2020* primary colours are:

- 630nm for the red primary colour
- 532nm for the green primary colour
- 467nm for the blue primary colour.

References

[]

RGB_COLOURSPACE_BT2020 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_CIE_RGB

colour.models.**RGB_COLOURSPACE_CIE_RGB = RGB_Colourspace(CIE RGB, [[0.73474284, 0.26525716], [0.27377903, 0.7174777], [0.16655563, 0.00891073]], [0.33333333, 0.33333333], E, [[0.49 , 0.31 , 0.2], [0.1769, 0.8124, 0.0107], [0. , 0.0099, 0.9901]], [[2.36449012, -0.89655263, -0.46793749], [-0.51493525, 1.42633279, 0.08860245], [0.00514883, -0.01426189, 1.00911305]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

CIE RGB colourspace.

References

[]

RGB_COLOURSPACE_CIE_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_CINEMA_GAMUT

colour.models.**RGB_COLOURSPACE_CINEMA_GAMUT = RGB_Colourspace(Cinema Gamut, [[0.74, 0.27], [0.17, 1.14], [0.08, -0.1]], [0.3127, 0.329], D65, [[0.71604965, 0.12968348, 0.1047228], [0.26126136, 0.86964215, -0.1309035], [-0.00967635, -0.23648164, 1.33521573]], [[1.48981827, -0.2608959 , -0.14242652], [-0.45816657, 1.26162778, 0.15962363], [-0.07034967, 0.22155767, 0.7761816]], <function linear_function>, <function linear_function>, False, False)**

Canon Cinema Gamut colourspace.

References

[]

RGB_COLOURSPACE_CINEMA_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_COLOR_MATCH_RGB

colour.models.**RGB_COLOURSPACE_COLOR_MATCH_RGB = RGB_Colourspace(ColorMatch RGB, [[0.63 , 0.34], [0.295, 0.605], [0.15 , 0.075]], [0.3457, 0.3585], D50, [[0.5094668 , 0.32087954, 0.13394933], [0.27495034, 0.658075 , 0.06697467], [0.02426032, 0.10877273, 0.69207155]], [[2.64164976, -1.22313179, -0.39291946], [-1.11207173, 2.05919502, 0.01596275], [0.08218196, -0.28076676, 1.45620209]], functools.partial(<function gamma_function>, exponent=0.5555555555555556), functools.partial(<function gamma_function>, exponent=1.8), False, False)**

ColorMatch RGB colourspace.

References

[]

RGB_COLOURSPACE_COLOR_MATCH_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT

colour.models.**RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT = RGB_Colourspace(DaVinci Wide Gamut, [[0.8 , 0.313], [0.1682, 0.9877], [0.079 , -0.1155]], [0.3127, 0.329], D65, [[0.70062239, 0.14877482, 0.10105872], [0.27411851, 0.8736319 , -0.14775041], [-0.09896291, -0.13789533, 1.32591599]], [[1.51667204, -0.28147805, -0.14696363], [-0.4649171 , 1.25142378, 0.17488461], [0.06484905, 0.10913934, 0.76141462]], <function linear_function>, <function linear_function>, False, False)**

DaVinci Wide Gamut colourspace.

References

[]

RGB_COLOURSPACE_DAVINCI_WIDE_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DCDM_XYZ

colour.models.**RGB_COLOURSPACE_DCDM_XYZ = RGB_Colourspace(DCDM XYZ, [[1., 0.], [0., 1.], [0., 0.]], [0.33333333, 0.33333333], E, [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]], [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]], <function eotf_inverse_DCDM>, <function eotf_DCDM>, False, False)**

DCDM XYZ colourspace.

References

[]

RGB_COLOURSPACE_DCDM_XYZ : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DCI_P3

colour.models.**RGB_COLOURSPACE_DCI_P3 = RGB_Colourspace(DCI-P3, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]], [0.314, 0.351], DCI-P3, [[4.45169816e-01, 2.77134409e-01, 1.72282670e-01], [2.09491678e-01, 7.21595254e-01, 6.89130679e-02], [-3.63410132e-17, 4.70605601e-02, 9.07355394e-01]], [[2.72539403, -1.01800301, -0.4401632], [-0.79516803, 1.68973205, 0.02264719], [0.04124189, -0.08763902, 1.10092938]], functools.partial(<function gamma_function>, exponent=0.3846153846153846), functools.partial(<function gamma_function>, exponent=2.6), False, False)**

DCI-P3 colourspace.

References

[], []

RGB_COLOURSPACE_DCI_P3 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DCI_P3_P

colour.models.**RGB_COLOURSPACE_DCI_P3_P = RGB_Colourspace(DCI-P3+, [[0.74, 0.27], [0.22, 0.78], [0.09, -0.09]], [0.314, 0.351], DCI-P3, [[0.55907356, 0.24893595, 0.08657739], [0.2039863 , 0.88259109, -0.08657739], [-0.00755505, 0. , 0.961971]], [[1.99040349, -0.56139586, -0.22966194], [-0.45849279, 1.262346 , 0.15487549], [0.01563207, -0.00440904, 1.03772867]], functools.partial(<function gamma_function>, exponent=0.3846153846153846), functools.partial(<function gamma_function>, exponent=2.6), False, False)**

DCI-P3+ colourspace.

References

[]

RGB_COLOURSPACE_DCI_P3_P : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DISPLAY_P3

colour.models.**RGB_COLOURSPACE_DISPLAY_P3 = RGB_Colourspace(Display P3, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]], [0.3127, 0.329], D65, [[4.86570949e-01, 2.65667693e-01, 1.98217285e-01], [2.28974564e-01, 6.91738522e-01, 7.92869141e-02], [-3.97207552e-17, 4.51133819e-02, 1.04394437e+00]], [[2.49349691, -0.93138362, -0.40271078], [-0.82948897, 1.76266406, 0.02362469], [0.03584583, -0.07617239, 0.95688452]], <function eotf_inverse_sRGB>, <function eotf_sRGB>, False, False)**

Display P3 colourspace.

References

[]

RGB_COLOURSPACE_DISPLAY_P3 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DON_RGB_4

colour.models.**RGB_COLOURSPACE_DON_RGB_4 = RGB_Colourspace(Don RGB 4, [[0.69612069, 0.29956897], [0.21468298, 0.76529477], [0.12993763, 0.03534304]], [0.3457, 0.3585], D50, [[0.64631888, 0.19296024, 0.12501655], [0.27813723, 0.68785827, 0.0340045], [0.00400197, 0.01799629, 0.80310634]], [[1.75819127, -0.48659205, -0.25308814], [-0.7112839 , 1.65225302, 0.04076449], [0.00717743, -0.03459953, 1.24551283]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Don RGB 4 colourspace.

References

[]

RGB_COLOURSPACE_DON_RGB_4 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ECI_RGB_V2

colour.models.**RGB_COLOURSPACE_ECI_RGB_V2 = RGB_Colourspace(ECI RGB v2, [[0.67010309, 0.32989691], [0.20990566, 0.70990566], [0.14006179, 0.08032956]], [0.3457, 0.3585], D50, [[0.65032438, 0.177949 , 0.13602229], [0.3201597 , 0.60182752, 0.07801279], [0. , 0.06798052, 0.75712409]], [[1.78215602, -0.49656317, -0.26901095], [-0.95923427, 1.94844461, -0.02843173], [0.08612755, -0.17494658, 1.32334029]], functools.partial(<function _scale_domain_0_100_range_0_1>, callable_=<function lightness_CIE1976>), functools.partial(<function _scale_domain_0_100_range_0_1>, callable_=<function luminance_CIE1976>), False, False)**

ECI RGB v2 colourspace.

References

[]

RGB_COLOURSPACE_ECI_RGB_V2 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_EKTA_SPACE_PS_5

colour.models.**RGB_COLOURSPACE_EKTA_SPACE_PS_5 = RGB_Colourspace(Ekta Space PS 5, [[0.69473684, 0.30526316], [0.26 , 0.7], [0.10972851, 0.00452489]], [0.3457, 0.3585], D50, [[0.59433686, 0.27294481, 0.09701401], [0.26114801, 0.73485141, 0.00400058], [0. , 0.04199151, 0.78311309]], [[2.00336603, -0.73013869, -0.24445204], [-0.71215462, 1.62076569, 0.07994372], [0.03818663, -0.08690749, 1.27266809]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Ekta Space PS 5 colourspace.

References

[]

RGB_COLOURSPACE_EKTA_SPACE_PS_5 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_F_GAMUT

colour.models.**RGB_COLOURSPACE_F_GAMUT = RGB_Colourspace(F-Gamut, [[0.708, 0.292], [0.17 , 0.797], [0.131, 0.046]], [0.3127, 0.329], D65, [[6.36958048e-01, 1.44616904e-01, 1.68880975e-01], [2.62700212e-01, 6.77998072e-01, 5.93017165e-02], [4.99410657e-17, 2.80726930e-02, 1.06098506e+00]], [[1.71665119, -0.35567078, -0.25336628], [-0.66668435, 1.61648124, 0.01576855], [0.01763986, -0.04277061, 0.94210312]], <function log_encoding_FLog>, <function log_decoding_FLog>, False, False)**

Fujifilm F-Gamut colourspace.

References

[]

RGB_COLOURSPACE_F_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_PROTUNE_NATIVE

colour.models.**RGB_COLOURSPACE_PROTUNE_NATIVE = RGB_Colourspace(Protune Native, [[0.69848046, 0.19302645], [0.32955538, 1.02459662], [0.10844263, -0.03467857]], [0.3127, 0.329], D65, [[0.50225719, 0.29296671, 0.15523203], [0.13879976, 0.91084146, -0.04964122], [0.07801426, -0.31483251, 1.325876]], [[2.2668965 , -0.83163359, -0.29654225], [-0.35733783, 1.24337315, 0.08838899], [-0.21823445, 0.34417515, 0.79265501]], <function log_encoding_Protune>, <function log_decoding_Protune>, False, False)**

Protune Native colourspace.

References

[], []

RGB_COLOURSPACE_PROTUNE_NATIVE : RGB_Colourspace

colour.models.RGB_COLOURSPACE_MAX_RGB

colour.models.**RGB_COLOURSPACE_MAX_RGB = RGB_Colourspace(Max RGB, [[0.73413379, 0.26586621], [0.10039113, 0.89960887], [0.03621495, 0.]], [0.3457, 0.3585], D50, [[0.85630404, 0.07698771, 0.03100393], [0.31011011, 0.68988989, 0.], [0. , 0. , 0.8251046]], [[1.2169928 , -0.13580933, -0.04572942], [-0.54704638, 1.51055387, 0.02055568], [0. , 0. , 1.21196755]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Max RGB colourspace.

References

[]

RGB_COLOURSPACE_MAX_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_NTSC1953

colour.models.**RGB_COLOURSPACE_NTSC1953 = RGB_Colourspace(NTSC (1953), [[0.67, 0.33], [0.21, 0.71], [0.14, 0.08]], [0.31006, 0.31616], C, [[6.06863809e-01, 1.73507281e-01, 2.00334881e-01], [2.98903070e-01, 5.86619855e-01, 1.14477075e-01], [-5.02801622e-17, 6.60980118e-02, 1.11615148e+00]], [[1.91008143, -0.53247794, -0.28822201], [-0.98463135, 1.99910001, -0.02830719], [0.05830945, -0.11838584, 0.89761208]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

NTSC (1953) colourspace.

References

[]

RGB_COLOURSPACE_NTSC1953 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_NTSC1987

colour.models.**RGB_COLOURSPACE_NTSC1987 = RGB_Colourspace(NTSC (1987), [[0.63 , 0.34], [0.31 , 0.595], [0.155, 0.07]], [0.3127, 0.329], D65, [[0.3935209 , 0.36525808, 0.19167695], [0.21237636, 0.70105986, 0.08656378], [0.01873909, 0.11193393, 0.95838473]], [[3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [0.05630659, -0.19697565, 1.04995233]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

NTSC (1987) colourspace.

References

[]

RGB_COLOURSPACE_NTSC1987 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_P3_D65

colour.models.**RGB_COLOURSPACE_P3_D65 = RGB_Colourspace(P3-D65, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.06]], [0.3127, 0.329], D65, [[4.86570949e-01, 2.65667693e-01, 1.98217285e-01], [2.28974564e-01, 6.91738522e-01, 7.92869141e-02], [-3.97207552e-17, 4.51133819e-02, 1.04394437e+00]], [[2.49349691, -0.93138362, -0.40271078], [-0.82948897, 1.76266406, 0.02362469], [0.03584583, -0.07617239, 0.95688452]], functools.partial(<function gamma_function>, exponent=0.3846153846153846), functools.partial(<function gamma_function>, exponent=2.6), False, False)**

P3-D65 colourspace.

RGB_COLOURSPACE_P3_D65 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_PAL_SECAM

colour.models.**RGB_COLOURSPACE_PAL_SECAM = RGB_Colourspace(Pal/Secam, [[0.64, 0.33], [0.29, 0.6], [0.15, 0.06]], [0.3127, 0.329], D65, [[0.43055381, 0.3415498 , 0.17835231], [0.22200431, 0.70665477, 0.07134092], [0.02018221, 0.12955337, 0.93932217]], [[3.06336109, -1.39339017, -0.47582374], [-0.96924364, 1.8759675 , 0.04155506], [0.06786105, -0.22879927, 1.06908962]], functools.partial(<function gamma_function>, exponent=0.35714285714285715), functools.partial(<function gamma_function>, exponent=2.8), False, False)**

Pal/Secam colourspace.

References

[]

RGB_COLOURSPACE_PAL_SECAM : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RED_COLOR

colour.models.**RGB_COLOURSPACE_RED_COLOR = RGB_Colourspace(REDcolor, [[0.70105856, 0.33018098], [0.29881132, 0.62516925], [0.13503868, 0.03526178]], [0.3127, 0.329], D65, [[0.42302331, 0.36210731, 0.16532531], [0.19923335, 0.75759632, 0.04317033], [-0.01885014, 0.09212233, 1.01578557]], [[2.99433635, -1.37906534, -0.42873703], [-0.79472663, 1.69283865, 0.0574019], [0.12764085, -0.17911636, 0.97129776]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

REDcolor colourspace.

References

[], []

RGB_COLOURSPACE_RED_COLOR : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RED_COLOR_2

colour.models.**RGB_COLOURSPACE_RED_COLOR_2 = RGB_Colourspace(REDcolor2, [[0.89740722, 0.33077623], [0.29602209, 0.68463555], [0.09979951, -0.02300051]], [0.3127, 0.329], D65, [[0.44957762, 0.3734296 , 0.12744871], [0.16571026, 0.86366248, -0.02937275], [-0.11431396, 0.02440023, 1.17897148]], [[2.55060735, -1.09426927, -0.30298724], [-0.48063394, 0.36324834, 0.0859211], [0.2572561 , -0.13431523, 0.81704083]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

REDcolor2 colourspace.

References

[], []

RGB_COLOURSPACE_RED_COLOR_2 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RED_COLOR_3

colour.models.**RGB_COLOURSPACE_RED_COLOR_3 = RGB_Colourspace(REDcolor3, [[0.70259866, 0.33018559], [0.29578224, 0.68974826], [0.11109053, -0.00433232]], [0.3127, 0.329], D65, [[0.47986312, 0.33439883, 0.13619398], [0.22551123, 0.77980008, -0.00531131], [-0.02239109, 0.01635861, 1.09509023]], [[2.58673915, -1.10240102, -0.32705386], [-0.74762558, 1.6008681 , 0.10074495], [0.06405867, -0.04645456, 0.90497461]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

REDcolor3 colourspace.

References

[], []

RGB_COLOURSPACE_RED_COLOR_3 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RED_COLOR_4

colour.models.**RGB_COLOURSPACE_RED_COLOR_4 = RGB_Colourspace(REDcolor4, [[0.70259815, 0.3301851], [0.29578233, 0.68974825], [0.14445924, 0.05083772]], [0.3127, 0.329], D65, [[0.44431783, 0.30962925, 0.19650885], [0.20880659, 0.72203852, 0.06915489], [-0.02073188, 0.0151468 , 1.09464284]], [[2.78855329, -1.18687705, -0.42561558], [-0.81255797, 1.73265028, 0.03640786], [0.06405707, -0.04645378, 0.9049753]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

REDcolor4 colourspace.

References

[], []

RGB_COLOURSPACE_RED_COLOR_4 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB

colour.models.**RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB = RGB_Colourspace(REDWideGamutRGB, [[0.780308, 0.304253], [0.121595, 1.493994], [0.095612, -0.084589]], [0.3127, 0.329], D65, [[0.735275, 0.068609, 0.146571], [0.286694, 0.842979, -0.129673], [-0.079681, -0.347343, 1.516082]], [[1.41280661, -0.17752237, -0.15177038], [-0.48620319, 1.29069621, 0.15740028], [-0.03713878, 0.28637576, 0.68767961]], <function log_encoding_Log3G10>, <function log_decoding_Log3G10>, False, False)**

REDWideGamutRGB colourspace.

References

[], [], []

RGB_COLOURSPACE_RED_WIDE_GAMUT_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DRAGON_COLOR

colour.models.**RGB_COLOURSPACE_DRAGON_COLOR = RGB_Colourspace(DRAGONcolor, [[0.75865589, 0.33035535], [0.29492362, 0.70805324], [0.0859616 , -0.04587944]], [0.3127, 0.329], D65, [[0.49831915, 0.34905932, 0.10307746], [0.21699218, 0.83802234, -0.05501452], [-0.05846657, -0.00352329, 1.15104761]], [[2.41407671, -1.00664042, -0.26429553], [-0.61715986, 1.45087355, 0.12461203], [0.12073206, -0.04669048, 0.85573054]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

DRAGONcolor colourspace.

References

[], []

RGB_COLOURSPACE_DRAGON_COLOR : RGB_Colourspace

colour.models.RGB_COLOURSPACE_DRAGON_COLOR_2

colour.models.**RGB_COLOURSPACE_DRAGON_COLOR_2 = RGB_Colourspace(DRAGONcolor2, [[0.75865621, 0.33035584], [0.29492389, 0.70805336], [0.14416873, 0.05035738]], [0.3127, 0.329], D65, [[0.43856251, 0.30720212, 0.2046913], [0.19097146, 0.73753094, 0.0714976], [-0.05145591, -0.0031012 , 1.14361486]], [[2.72655873, -1.13744045, -0.41690486], [-0.71770143, 1.654923 , 0.02499461], [0.12073281, -0.04669036, 0.85572978]], <function log_encoding_REDLogFilm>, <function log_decoding_REDLogFilm>, False, False)**

DRAGONcolor2 colourspace.

References

[], []

RGB_COLOURSPACE_DRAGON_COLOR_2 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ROMM_RGB

colour.models.**RGB_COLOURSPACE_ROMM_RGB = RGB_Colourspace(ROMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], D50, [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], <function cctf_encoding_ROMMRGB>, <function cctf_decoding_ROMMRGB>, False, False)**

ROMM RGB colourspace.

References

[], []

RGB_COLOURSPACE_ROMM_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RIMM_RGB

colour.models.**RGB_COLOURSPACE_RIMM_RGB = RGB_Colourspace(RIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], D50, [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], <function cctf_encoding_RIMMRGB>, <function cctf_decoding_RIMMRGB>, False, False)**

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

[]

RGB_COLOURSPACE_RIMM_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_ERIMM_RGB

colour.models.**RGB_COLOURSPACE_ERIMM_RGB = RGB_Colourspace(ERIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], D50, [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], <function log_encoding_ERIMMRGB>, <function log_decoding_ERIMMRGB>, False, False)**

ERIMM RGB colourspace.

References

[]

RGB_COLOURSPACE_ERIMM_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_PROPHOTO_RGB

colour.models.**RGB_COLOURSPACE_PROPHOTO_RGB = RGB_Colourspace(ProPhoto RGB, [[7.34700000e-01, 2.65300000e-01], [1.59600000e-01, 8.40400000e-01], [3.66000000e-02, 1.00000000e-04]], [0.3457, 0.3585], D50, [[7.97700000e-01, 1.35200000e-01, 3.13000000e-02], [2.88000000e-01, 7.11900000e-01, 1.00000000e-04], [0.00000000e+00, 0.00000000e+00, 8.24900000e-01]], [[1.346 , -0.2556, -0.0511], [-0.5446, 1.5082, 0.0205], [0. , 0. , 1.2123]], <function cctf_encoding_ROMMRGB>, <function cctf_decoding_ROMMRGB>, False, False)**

ProPhoto RGB colourspace, an alias colourspace for *ROMM RGB*.

References

[], []

RGB_COLOURSPACE_PROPHOTO_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_RUSSELL_RGB

colour.models.**RGB_COLOURSPACE_RUSSELL_RGB = RGB_Colourspace(Russell RGB, [[0.69, 0.31], [0.18, 0.77], [0.1 , 0.02]], [0.33243, 0.34744], D55, [[7.01583746e-01, 1.55416218e-01, 9.97983328e-02], [3.15204292e-01, 6.64836042e-01, 1.99596666e-02], [5.64430745e-17, 4.31711716e-02, 8.78225329e-01]], [[1.58699918, -0.35980738, -0.17216338], [-0.75352154, 1.67719311, 0.04750942], [0.03704107, -0.08244626, 1.13632451]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Russell RGB colourspace.

References

[]

RGB_COLOURSPACE_RUSSELL_RGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_SMPTE_240M

colour.models.**RGB_COLOURSPACE_SMPTE_240M = RGB_Colourspace(SMPTE 240M, [[0.63 , 0.34], [0.31 , 0.595], [0.155, 0.07]], [0.3127, 0.329], D65, [[0.3935209 , 0.36525808, 0.19167695], [0.21237636, 0.70105986, 0.08656378], [0.01873909, 0.11193393, 0.95838473]], [[3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [0.05630659, -0.19697565, 1.04995233]], <function oetf_SMPTE240M>, <function eotf_SMPTE240M>, False, False)**

SMPTE 240M colourspace.

References

[],

RGB_COLOURSPACE_SMPTE_240M : RGB_Colourspace

colour.models.RGB_COLOURSPACE_SMPTE_C

colour.models.**RGB_COLOURSPACE_SMPTE_C = RGB_Colourspace(SMPTE C, [[0.63 , 0.34], [0.31 , 0.595], [0.155, 0.07]], [0.3127, 0.329], D65, [[0.3935209 , 0.36525808, 0.19167695], [0.21237636, 0.70105986, 0.08656378], [0.01873909, 0.11193393, 0.95838473]], [[3.50600328, -1.73979073, -0.54405827], [-1.06904756, 1.97777888, 0.03517142], [0.05630659, -0.19697565, 1.04995233]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Implements support for the *RGB* colourspaces datasets from `colour.models.datasets.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 `matrix_RGB_to_XYZ` and `matrix_XYZ_to_RGB` 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_matrix_RGB_to_XYZ` and `colour.RGB_Colourspace.use_derived_matrix_XYZ_to_RGB` attribute 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.
- **matrix_RGB_to_XYZ** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **matrix_XYZ_to_RGB** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **cctf_encoding** (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.
- **cctf_decoding** (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_matrix_RGB_to_XYZ** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_matrix_XYZ_to_RGB** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Attributes

- name
- primaries
- whitepoint
- whitepoint_name
- matrix_RGB_to_XYZ
- matrix_XYZ_to_RGB
- cctf_encoding
- cctf_decoding
- use_derived_matrix_RGB_to_XYZ
- use_derived_matrix_XYZ_to_RGB

Methods

- `__init__`
- `__str__`
- `__repr__`
- `use_derived_transformation_matrices`
- `chromatically_adapt`
- `copy`

Notes

- The normalised primary matrix defined by `colour.RGB_Colourspace.matrix_RGB_to_XYZ` 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

[], []

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> matrix_RGB_to_XYZ = np.identity(3)
>>> matrix_XYZ_to_RGB = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                               matrix_RGB_to_XYZ, matrix_XYZ_to_RGB)
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.matrix_XYZ_to_RGB
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
>>> colourspace.matrix_RGB_to_XYZ
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.matrix_XYZ_to_RGB
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_matrix_RGB_to_XYZ = False
>>> colourspace.matrix_RGB_to_XYZ
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

(continues on next page)

(continued from previous page)

```
>>> colourspace.use_derived_matrix_XYZ_to_RGB = False
>>> colourspace.matrix_XYZ_to_RGB
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

colour.models.RGB_COLOURSPACE_S_GAMUT

colour.models.**RGB_COLOURSPACE_S_GAMUT = RGB_Colourspace(S-Gamut, [[0.73 , 0.28], [0.14 , 0.855], [0.1 , -0.05]], [0.3127, 0.329], D65, [[0.70648271, 0.12880105, 0.11517216], [0.27097967, 0.78660641, -0.05758608], [-0.00967785, 0.00460004, 1.09413556]], [[1.5073999 , -0.24582214, -0.17161168], [-0.51815173, 1.35539124, 0.12587867], [0.0155117 , -0.00787277, 0.91191637]], <function log_encoding_SLog2>, <function log_decoding_SLog2>, False, False)**

S-Gamut colourspace.

References

[], []

RGB_COLOURSPACE_S_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_S_GAMUT3

colour.models.**RGB_COLOURSPACE_S_GAMUT3 = RGB_Colourspace(S-Gamut3, [[0.73 , 0.28], [0.14 , 0.855], [0.1 , -0.05]], [0.3127, 0.329], D65, [[0.70648271, 0.12880105, 0.11517216], [0.27097967, 0.78660641, -0.05758608], [-0.00967785, 0.00460004, 1.09413556]], [[1.5073999 , -0.24582214, -0.17161168], [-0.51815173, 1.35539124, 0.12587867], [0.0155117 , -0.00787277, 0.91191637]], <function log_encoding_SLog3>, <function log_decoding_SLog3>, False, False)**

S-Gamut3 colourspace.

References

[]

RGB_COLOURSPACE_S_GAMUT3 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_S_GAMUT3_CINE

colour.models.**RGB_COLOURSPACE_S_GAMUT3_CINE = RGB_Colourspace(S-Gamut3.Cine, [[0.766, 0.275], [0.225, 0.8], [0.089, -0.087]], [0.3127, 0.329], D65, [[0.59908392, 0.24892552, 0.10244649], [0.21507582, 0.8850685 , -0.10014432], [-0.03206585, -0.02765839, 1.14878199]], [[1.84677897, -0.52598612, -0.21054521], [-0.44415326, 1.2594429 , 0.14939997], [0.04085542, 0.01564089, 0.86820725]], <function log_encoding_SLog3>, <function log_decoding_SLog3>, False, False)**

S-Gamut3.Cine colourspace.

References

[]

RGB_COLOURSPACE_S_GAMUT3_CINE : RGB_Colourspace

colour.models.RGB_COLOURSPACE_VENICE_S_GAMUT3

colour.models.**RGB_COLOURSPACE_VENICE_S_GAMUT3 = RGB_Colourspace(Venice S-Gamut3, [[0.74046426, 0.27936437], [0.08924115, 0.89380953], [0.11048824, -0.05257933]], [0.3127, 0.329], D65, [[0.74422299, 0.07790652, 0.12832642], [0.28078248, 0.78028572, -0.0610682], [-0.01992929, 0.01479657, 1.09419047]], [[1.39026398, -0.13557353, -0.17061639], [-0.49777193, 1.32876782, 0.13253885], [0.03205319, -0.02043803, 0.9090178]], <function log_encoding_SLog3>, <function log_decoding_SLog3>, False, False)**

Venice S-Gamut3 colourspace.

References

[]

RGB_COLOURSPACE_VENICE_S_GAMUT3 : RGB_Colourspace

colour.models.RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE

colour.models.**RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE = RGB_Colourspace(Venice S-Gamut3.Cine, [[0.77590187, 0.27450239], [0.1886829 , 0.82868494], [0.10133738, -0.08918752]], [0.3127, 0.329], D65, [[0.63226084, 0.20037001, 0.11782508], [0.22368436, 0.88001406, -0.10369842], [-0.04107303, -0.01844361, 1.14857439]], [[1.70701129, -0.39308248, -0.21060088], [-0.42750858, 1.23694441, 0.1555323], [0.05417788, 0.00580601, 0.86561094]], <function log_encoding_SLog3>, <function log_decoding_SLog3>, False, False)**

Venice S-Gamut3.Cine colourspace.

References

[]

RGB_COLOURSPACE_VENICE_S_GAMUT3_CINE : RGB_Colourspace

colour.models.RGB_COLOURSPACE_sRGB

colour.models.**RGB_COLOURSPACE_sRGB = RGB_Colourspace(sRGB, [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]], [0.3127, 0.329], D65, [[0.4124, 0.3576, 0.1805], [0.2126, 0.7152, 0.0722], [0.0193, 0.1192, 0.9505]], [[3.2406, -1.5372, -0.4986], [-0.9689, 1.8758, 0.0415], [0.0557, -0.204 , 1.057]], <function eotf_inverse_sRGB>, <function eotf_sRGB>, False, False)**

sRGB colourspace.

References

[], []

RGB_COLOURSPACE_sRGB : RGB_Colourspace

colour.models.RGB_COLOURSPACE_V_GAMUT

colour.models.**RGB_COLOURSPACE_V_GAMUT = RGB_Colourspace(V-Gamut, [[0.73 , 0.28], [0.165, 0.84], [0.1 , -0.03]], [0.3127, 0.329], D65, [[0.679644, 0.152211, 0.1186], [0.260686, 0.774894, -0.03558], [-0.00931 , -0.004612, 1.10298]], [[1.589012, -0.313204, -0.180965], [-0.534053, 1.396011, 0.102458], [0.011179, 0.003194, 0.905535]], <function log_encoding_VLog>, <function log_decoding_VLog>, False, False)**

Panasonic V-Gamut colourspace.

References

[]

RGB_COLOURSPACE_V_GAMUT : RGB_Colourspace

colour.models.RGB_COLOURSPACE_XTREME_RGB

colour.models.**RGB_COLOURSPACE_XTREME_RGB = RGB_Colourspace(Xtreme RGB, [[1., 0.], [0., 1.], [0., 0.]], [0.3457, 0.3585], D50, [[0.96429568, 0. , 0.], [0. , 1. , 0.], [0. , 0. , 0.8251046]], [[1.03702632, 0. , 0.], [0. , 1. , 0.], [0. , 0. , 1.21196755]], functools.partial(<function gamma_function>, exponent=0.45454545454545453), functools.partial(<function gamma_function>, exponent=2.2), False, False)**

Xtreme RGB colourspace.

References

[]

RGB_COLOURSPACE_XTREME_RGB : RGB_Colourspace

Colour Component Transfer Functions

colour

cctf_encoding(value[, function])	Encodes linear RGB values to non linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).
CCTF_ENCODINGS	Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES attributes, the colour.models.cctf_encoding_ProPhotoRGB(), colour.models.cctf_encoding_RIMMRGB(), colour.models.cctf_encoding_ROMMRGB() definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).
cctf_decoding(value[, function])	Decodes non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).
CCTF_DECODINGS	Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES attributes, the colour.models.cctf_decoding_ProPhotoRGB(), colour.models.cctf_decoding_RIMMRGB(), colour.models.cctf_decoding_ROMMRGB() definitions and 3 gamma decoding functions (2.2, 2.4, 2.6).
gamma_function(a[, exponent, ...])	Defines a typical gamma encoding / decoding function.
linear_function(a)	Defines a typical linear encoding / decoding function, essentially a pass-through function.

colour.cctf_encoding

colour.**cctf_encoding**(*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.CCTF_ENCODINGS}, Computation function.
- ****kwargs** (dict, optional) – Keywords arguments for the relevant encoding CCTF of the colour.CCTF_ENCODINGS attribute collection.

Warning: For *ITU-R BT.2100*, only the inverse 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

```
>>> cctf_encoding(0.18, function='PLog', log_reference=400)
...
0.3910068...
>>> cctf_encoding(0.18, function='ST 2084', L_p=1000)
...
0.1820115...
>>> cctf_encoding(
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.CCTF_ENCODINGS

colour.**CCTF_ENCODINGS = CaseInsensitiveMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., 'ProPhoto RGB': ..., 'RIMM RGB': ..., 'ROMM RGB': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log C': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'Filmic Pro 6': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'Panalog': ..., 'PLog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ..., 'ARIB STD-B67': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'SMPTE 240M': ..., 'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2020': ..., 'ST 2084': ..., 'sRGB': ...}**)

Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES attributes, the colour.models.cctf_encoding_ProPhotoRGB(), colour.models.cctf_encoding_RIMMRGB(), colour.models.cctf_encoding_ROMMRGB() definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).

Warning: For *ITU-R BT.2100*, only the inverse 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).

CCTF_ENCODINGS [CaseInsensitiveMapping] {colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES}

colour.cctf_decoding

colour.**cctf_decoding**(*value*, *function='sRGB'*, ***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.CCTF_DECODINGS}, Computation function.
- ****kwargs** (dict, optional) – Keywords arguments for the relevant decoding CCTF of the colour.CCTF_DECODINGS 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_inverse()` definition for the inverse opto-electronic transfer functions (OETF / OECF).

Returns Linear RGB values.

Return type numeric or ndarray

Examples

```
>>> cctf_decoding(0.391006842619746, function='PLog', log_reference=400)
...
0.1...
>>> cctf_decoding(0.182011532850008, function='ST 2084', L_p=1000)
...
0.1...
>>> cctf_decoding(
...     0.461356129500442, function='ITU-R BT.1886')
0.1...
```

colour.CCTF_DECODINGS

colour.**CCTF_DECODINGS = CaseInsensitiveMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., 'ProPhoto RGB': ..., 'RIMM RGB': ..., 'ROMM RGB': ..., 'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log C': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'Filmic Pro 6': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'Panalog': ..., 'PLog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ..., 'ARIB STD-B67': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2020': ..., 'SMPTE 240M': ..., 'ST 2084': ..., 'sRGB': ...})**

Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by `colour.LOG_DECODINGS`, `colour.EOTFS`, `colour.OETF_INVERSES` attributes, the `colour.models.cctf_decoding_ProPhotoRGB()`, `colour.models.cctf_decoding_RIMMRGB()`, `colour.models.cctf_decoding_ROMMRGB()` definitions 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.OETF_INVERSES` attribute for the inverse 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.

CCTF_DECODINGS [CaseInsensitiveMapping] {colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES}

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)
0.0229932...
>>> gamma_function(-0.18, 2.0)
0.0323999...
>>> gamma_function(-0.18, 2.2)
nan
>>> gamma_function(-0.18, 2.2, 'Mirror')
-0.0229932...
>>> gamma_function(-0.18, 2.2, 'Preserve')
-0.1...
>>> gamma_function(-0.18, 2.2, 'Clamp')
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
```

colour.models

cctf_encoding_ROMMRGB(X[, bit_depth, out_int])	Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ROMMRGB(X_p[, bit_depth, in_int])	Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).
cctf_encoding_RIMMRGB(X[, bit_depth, ...])	Defines the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_RIMMRGB(X_p[, bit_depth, ...])	Defines the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ROMMRGB

colour.models.**cctf_encoding_ROMMRGB**(*X*, *bit_depth=8*, *out_int=False*)

Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).

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

[], []

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ROMMRGB

colour.models.**cctf_decoding_ROMMRGB**(*X_p*, *bit_depth=8*, *in_int=False*)

Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

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

[], []

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

colour.models.cctf_encoding_RIMMRGB

colour.models.**cctf_encoding_RIMMRGB**(*X*, *bit_depth=8*, *out_int=False*, *E_clip=2.0*)

Defines the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).

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

[]

Examples

```
>>> cctf_encoding_RIMMRGB(0.18)
0.2916737...
>>> cctf_encoding_RIMMRGB(0.18, out_int=True)
74
```

colour.models.cctf_decoding_RIMMRGB

colour.models.**cctf_decoding_RIMMRGB**(*X_p*, *bit_depth=8*, *in_int=False*, *E_clip=2.0*)

Defines the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

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

[]

Examples

```
>>> cctf_decoding_RIMMRGB(0.291673732475746)
0.1...
>>> cctf_decoding_RIMMRGB(74, in_int=True)
0.1...
```

Aliases

colour.models

cctf_encoding_ProPhotoRGB(X[, bit_depth, ...])	Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ProPhotoRGB(X_p[, bit_depth, ...])	Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ProPhotoRGB

colour.models.**cctf_encoding_ProPhotoRGB**(*X*, *bit_depth=8*, *out_int=False*)

Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).

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

[], []

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ProPhotoRGB

colour.models.**cctf_decoding_ProPhotoRGB**(*X_p*, *bit_depth=8*, *in_int=False*)

Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

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

[], []

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

Ancillary Objects

`colour.models`

exponent_function_basic(x[, exponent, style])	Defines the *basic* exponent transfer function.
exponent_function_monitor_curve(x[, ...])	Defines the *Monitor Curve* exponent transfer function.
logarithmic_function_basic(x[, style, base])	Defines the basic logarithmic function.
logarithmic_function_quasilog(x[, style, ...])	Defines the quasilog logarithmic function.
logarithmic_function_camera(x[, style, ...])	Defines the camera logarithmic function.

colour.models.exponent_function_basic

`colour.models.`**`exponent_function_basic`**(*x*, *exponent=1*, *style='basicFwd'*)

Defines the *basic* exponent transfer function.

Parameters

- **x** (`numeric or array_like`) – Data to undergo the basic exponent conversion.
- **exponent** (`numeric or array_like, optional`) – Exponent value used for the conversion.
- **style** (`unicode, optional`) – **{'basicFwd', 'basicRev', 'basicMirrorFwd', 'basicMirrorRev', 'basicPassThruFwd', 'basicPassThruRev'}**, Defines the behaviour for the transfer function to operate:
  - *basicFwd*: *Basic Forward* exponential behaviour where the definition applies a basic power law using the exponent. Values less than zero are clamped.
  - *basicRev*: *Basic Reverse* exponential behaviour where the definition applies a basic power law using the exponent. Values less than zero are clamped.
  - *basicMirrorFwd*: *Basic Mirror Forward* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *basicMirrorRev*: *Basic Mirror Reverse* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *basicPassThruFwd*: *Basic Pass Forward* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and passes values less than zero unchanged.
  - *basicPassThruRev*: *Basic Pass Reverse* exponential behaviour where the definition applies a basic power law using the exponent for values greater than or equal to zero and passes values less than zero unchanged.

Returns Exponentially converted data.

Return type numeric or ndarray

Raises `ValueError` – If the *style* is not defined.

Examples

```
>>> exponent_function_basic(0.18, 2.2)
0.0229932...
>>> exponent_function_basic(-0.18, 2.2)
0.0
>>> exponent_function_basic(0.18, 2.2, 'basicRev')
0.4586564...
>>> exponent_function_basic(-0.18, 2.2, 'basicRev')
0.0
>>> exponent_function_basic(
...     0.18, 2.2, 'basicMirrorFwd')
0.0229932...
>>> exponent_function_basic(
...     -0.18, 2.2, 'basicMirrorFwd')
-0.0229932...
>>> exponent_function_basic(
...     0.18, 2.2, 'basicMirrorRev')
0.4586564...
>>> exponent_function_basic(
...     -0.18, 2.2, 'basicMirrorRev')
-0.4586564...
>>> exponent_function_basic(
...     0.18, 2.2, 'basicPassThruFwd')
0.0229932...
>>> exponent_function_basic(
...     -0.18, 2.2, 'basicPassThruFwd')
-0.1799999...
>>> exponent_function_basic(
...     0.18, 2.2, 'basicPassThruRev')
0.4586564...
>>> exponent_function_basic(
...     -0.18, 2.2, 'basicPassThruRev')
-0.1799999...
```

colour.models.exponent_function_monitor_curve

colour.models.**exponent_function_monitor_curve**(*x*, *exponent=1*, *offset=0*, *style='monCurveFwd'*)

Defines the *Monitor Curve* exponent transfer function.

Parameters

- **x** (numeric or array_like) – Data to undergo the monitor curve exponential conversion.
- **exponent** (numeric or array_like, optional) – Exponent value used for the conversion.
- **offset** (numeric or array_like, optional) – Offset value used for the conversion.

- **style** (unicode, optional) – **{'monCurveFwd', 'monCurveRev', 'monCurveMirrorFwd', 'monCurveMirrorRev'}**, Defines the behaviour for the transfer function to operate:
  - *monCurveFwd*: *Monitor Curve Forward* exponential behaviour where the definition applies a power law function with a linear segment near the origin.
  - *monCurveRev*: *Monitor Curve Reverse* exponential behaviour where the definition applies a power law function with a linear segment near the origin.
  - *monCurveMirrorFwd*: *Monitor Curve Mirror Forward* exponential behaviour where the definition applies a power law function with a linear segment near the origin and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).
  - *monCurveMirrorRev*: *Monitor Curve Mirror Reverse* exponential behaviour where the definition applies a power law function with a linear segment near the origin and mirrors the function for values less than zero (i.e. rotationally symmetric around the origin).

Returns Exponentially converted data.

Return type numeric or ndarray

Raises `ValueError` – If the *style* is not defined.

Examples

```
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 0.001)
0.0232240...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001)
-0.0002054...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 0.001, 'monCurveRev')
0.4581151...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, 'monCurveRev')
-157.7302795...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 2, 'monCurveMirrorFwd')
0.1679399...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, 'monCurveMirrorFwd')
-0.0232240...
>>> exponent_function_monitor_curve(
...     0.18, 2.2, 0.001, 'monCurveMirrorRev')
0.4581151...
>>> exponent_function_monitor_curve(
...     -0.18, 2.2, 0.001, 'monCurveMirrorRev')
-0.4581151...
```

colour.models.logarithmic_function_basic

colour.models.**logarithmic_function_basic**(*x*, *style='log2'*, *base=2*)

Defines the basic logarithmic function.

Parameters

- **x** (numeric) – The data to undergo basic logarithmic conversion.
- **style** (unicode, optional) – **{'log10', 'antiLog10', 'log2', 'antiLog2', 'logB', 'antiLogB'}**, Defines the behaviour for the logarithmic function to operate:
  - *log10*: Applies a base 10 logarithm to the passed value.
  - *antiLog10*: Applies a base 10 anti-logarithm to the passed value.
  - *log2*: Applies a base 2 logarithm to the passed value.
  - *antiLog2*: Applies a base 2 anti-logarithm to the passed value.
  - *logB*: Applies an arbitrary base logarithm to the passed value.
  - *antiLogB*: Applies an arbitrary base anti-logarithm to the passed value.
- **base** (numeric, optional) – Logarithmic base used for the conversion.

Returns Logarithmically converted data.

Return type numeric or ndarray

Raises ValueError – If the *style* is not defined.

Examples

The basic logarithmic function *styles* operate as follows:

```
>>> logarithmic_function_basic(0.18)
-2.4739311...
>>> logarithmic_function_basic(0.18, 'log10')
-0.7447274...
>>> logarithmic_function_basic(
...     0.18, 'logB', 3)
-1.5608767...
>>> logarithmic_function_basic(
...     -2.473931188332412, 'antiLog2')
0.18000000...
>>> logarithmic_function_basic(
...     -0.7447274948966939, 'antiLog10')
0.18000000...
>>> logarithmic_function_basic(
...     -1.5608767950073117, 'antiLogB', 3)
0.18000000...
```

colour.models.logarithmic_function_quasilog

colour.models.**logarithmic_function_quasilog**(*x*, *style='linToLog'*, *base=2*, *log_side_slope=1*, *lin_side_slope=1*, *log_side_offset=0*, *lin_side_offset=0*)

Defines the quasilog logarithmic function.

Parameters

- **x** (numeric) – Linear/non-linear data to undergo encoding/decoding.
- **style** (unicode, optional) – **{'linToLog', 'logToLin'}**, Defines the behaviour for the logarithmic function to operate:
  - *linToLog*: Applies a logarithm to convert linear data to logarithmic data.
  - *logToLin*: Applies an anti-logarithm to convert logarithmic data to linear data.
- **base** (numeric, optional) – Logarithmic base used for the conversion.
- **log_side_slope** (numeric, optional) – Slope (or gain) applied to the log side of the logarithmic function. The default value is 1.
- **lin_side_slope** (numeric, optional) – Slope of the linear side of the logarithmic function. The default value is 1.
- **log_side_offset** (numeric, optional) – Offset applied to the log side of the logarithmic function. The default value is 0.
- **lin_side_offset** (numeric, optional) – Offset applied to the linear side of the logarithmic function. The default value is 0.

Returns Encoded/Decoded data.

Return type numeric or ndarray

Raises ValueError – If the *style* is not defined.

Examples

```
>>> logarithmic_function_quasilog(
...     0.18, 'linToLog')
-2.4739311...
>>> logarithmic_function_quasilog(
...     -2.473931188332412, 'logToLin')
0.18000000...
```

colour.models.logarithmic_function_camera

colour.models.**logarithmic_function_camera**(*x*, *style='cameraLinToLog'*, *base=2*, *log_side_slope=1*, *lin_side_slope=1*, *log_side_offset=0*, *lin_side_offset=0*, *lin_side_break=0.005*, *linear_slope=None*)

Defines the camera logarithmic function.

Parameters

- **x** (numeric) – Linear/non-linear data to undergo encoding/decoding.
- **style** (unicode, optional) – **{'cameraLinToLog', 'cameraLogToLin'}**, Defines the behaviour for the logarithmic function to operate:
  - *cameraLinToLog*: Applies a piece-wise function with logarithmic and linear segments on linear values, converting them to non-linear values.

– *cameraLogToLin*: Applies a piece-wise function with logarithmic and linear segments on non-linear values, converting them to linear values.

- **base** (numeric, optional) – Logarithmic base used for the conversion.
- **log_side_slope** (numeric, optional) – Slope (or gain) applied to the log side of the logarithmic segment. The default value is 1.
- **lin_side_slope** (numeric, optional) – Slope of the linear side of the logarithmic segment. The default value is 1.
- **log_side_offset** (numeric, optional) – Offset applied to the log side of the logarithmic segment. The default value is 0.
- **lin_side_offset** (numeric, optional) – Offset applied to the linear side of the logarithmic segment. The default value is 0.
- **lin_side_break** (numeric) – Break-point, defined in linear space, at which the piece-wise function transitions between the logarithmic and linear segments.
- **linear_slope** (numeric, optional) – Slope of the linear portion of the curve. The default value is *None*.

Returns Encoded/Decoded data.

Return type numeric or ndarray

Raises `ValueError` – If the *style* is not defined.

Examples

```
>>> logarithmic_function_camera(
...     0.18, 'cameraLinToLog')
-2.4739311...
>>> logarithmic_function_camera(
...     -2.4739311883324122, 'cameraLogToLin')
0.1800000...
```

Opto-Electronic Transfer Functions

colour

oetf(value[, function])	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).
OETFS	Supported opto-electrical transfer functions (OETFs / OECFs).
oetf_inverse(value[, function])	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF / OECF).
OETF_INVERSES	Supported inverse opto-electrical transfer functions (OETFs / OECFs).

colour.oetf

colour.**oetf**(*value*, *function='ITU-R BT.709'*, ***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) – **{'ITU-R BT.709', 'ARIB STD-B67', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'SMPTE 240M', 'ST 2084'}**, Opto-electronic transfer function (OETF / OECF).
- **E_clip** (numeric, optional) – {colour.models.cctf_encoding_RIMMRGB()}, Maximum exposure level.
- **I_max** (numeric, optional) – {colour.models.cctf_encoding_ROMMRGB(), colour.models.cctf_encoding_RIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **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)
0.4090077...
>>> oetf(0.18, function='ITU-R BT.601')
0.4090077...
```

colour.OETFS

colour.**OETFS = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'SMPTE 240M': ...})**

Supported opto-electrical transfer functions (OETFs / OECFs).

OETFS [CaseInsensitiveMapping] **{'sRGB', 'ARIB STD-B67', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'SMPTE 240M', 'ST 2084'}**

colour.oetf_inverse

colour.**oetf_inverse**(*value*, *function='ITU-R BT.709'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.709', 'ARIB STD-B67', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', }**, Inverse opto-electronic transfer function (OETF / OECF).
- **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_inverse(0.409007728864150)
0.1...
>>> oetf_inverse(
...     0.409007728864150, function='ITU-R BT.601')
0.1...
```

colour.OETF_INVERSES

colour.**OETF_INVERSES = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ITU-R BT.601': ..., 'ITU-R BT.709': ...})**

Supported inverse opto-electrical transfer functions (OETFs / OECFs).

OETF_INVERSES [CaseInsensitiveMapping] **{'ARIB STD-B67', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}**

colour.models

oetf_ARIBSTDB67(E[, r, constants])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).
oetf_inverse_ARIBSTDB67(E_p[, r, constants])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* inverse opto-electrical transfer function (OETF / OECF).
oetf_HLG_BT2100(E[, constants])	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).
oetf_inverse_HLG_BT2100(E_p[, constants])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF / OECF).
oetf_PQ_BT2100(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).
oetf_inverse_PQ_BT2100(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF / OECF).
oetf_BT601(L)	Defines *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF / OECF).
oetf_inverse_BT601(E)	Defines *Recommendation ITU-R BT.601-7* inverse opto-electronic transfer function (OETF / OECF).
oetf_BT709(L)	Defines *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF / OECF).
oetf_inverse_BT709(V)	Defines *Recommendation ITU-R BT.709-6* inverse opto-electronic transfer function (OETF / OECF).
oetf_SMPTE240M(L_c)	Defines *SMPTE 240M* opto-electrical 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]

- This definition uses the *mirror* negative number handling mode of colour.models.gamma_function() definition to the sign of negative numbers.

References

[]

Examples

```
>>> oetf_ARIBSTDB67(0.18)
0.2121320...
```

colour.models.oetf_inverse_ARIBSTDB67

colour.models.**oetf_inverse_ARIBSTDB67**(*E_p*, *r=0.5*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991073}*)

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* inverse 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]

- This definition uses the *mirror* negative number handling mode of `colour.models.gamma_function()` definition to the sign of negative numbers.

References

[]

Examples

```
>>> oetf_inverse_ARIBSTDB67(0.212132034355964)
0.1799999...
```

colour.models.oetf_HLG_BT2100

`colour.models.`**`oetf_HLG_BT2100`**(*E*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*)

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.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

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

[], []

Examples

```
>>> oetf_HLG_BT2100(0.18 / 12)
0.2121320...
```

colour.models.oetf_inverse_HLG_BT2100

colour.models.**oetf_inverse_HLG_BT2100**(*E_p*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF / OECF).

Parameters

- **E_p** (numeric or array_like) – E' is the resulting non-linear signal R', G', B'.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

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

[], []

Examples

```
>>> oetf_inverse_HLG_BT2100(0.212132034355964)
0.0149999...
```

colour.models.oetf_PQ_BT2100

colour.models.**oetf_PQ_BT2100**(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

[], []

Examples

```
>>> oetf_PQ_BT2100(0.1)
0.7247698...
```

colour.models.oetf_inverse_PQ_BT2100

colour.models.**oetf_inverse_PQ_BT2100**(E_p)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF / OECF).

Parameters **E_p** (numeric or array_like) – E' is the resulting non-linear signal (R', G', B').

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ 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

[], []

Examples

```
>>> oetf_inverse_PQ_BT2100(0.724769816665726)
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

[]

Examples

```
>>> oetf_BT601(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT601

colour.models.**oetf_inverse_BT601**(E)

Defines *Recommendation ITU-R BT.601-7* inverse 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

[]

Examples

```
>>> oetf_inverse_BT601(0.409007728864150)
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

[]

Examples

```
>>> oetf_BT709(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT709

colour.models.**oetf_inverse_BT709**(*V*)

Defines *Recommendation ITU-R BT.709-6* inverse 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
V	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> oetf_inverse_BT709(0.409007728864150)
0.1...
```

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

[]

Examples

```
>>> oetf_SMPTE240M(0.18)
0.4022857...
```

Electro-Optical Transfer Functions

colour

eotf(value[, function])	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).
EOTFS	Supported electro-optical transfer functions (EOTFs / EOCFs).
eotf_inverse(value[, function])	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse electro-optical transfer function (EOTF / EOCF).
EOTF_INVERSES	Supported inverse 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', 'SMPTE 240M', 'ST 2084', 'sRGB'}**, Electro-optical transfer function (EOTF / EOCF).
- **E_clip** (numeric, optional) – {colour.models.cctf_decoding_RIMMRGB()}, Maximum exposure level.
- **I_max** (numeric, optional) – {colour.models.cctf_decoding_ROMMRGB(), colour.models.cctf_decoding_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_HLG_BT2100()}, Screen luminance for black.
- **L_W** (numeric, optional) – {colour.models.eotf_BT1886(), colour.models.eotf_HLG_BT2100()}, Screen luminance for white.
- **L_p** (numeric, optional) – {colour.models.eotf_ST2084()}, Display peak luminance cd/m^2.
- **gamma** (numeric, optional) – {colour.models.eotf_HLG_BT2100()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **is_12_bits_system** (bool, optional) – {colour.models.eotf_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.
- **method** (unicode, optional) – {colour.models.eotf_HLG_BT2100()}, **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**
- **out_int** (bool, optional) – {colour.models.eotf_DCDM(), colour.models.eotf_DICOMGSDF()}, Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> eotf(0.461356129500442)
0.1...
>>> eotf(0.409007728864150, function='ITU-R BT.2020')
...
0.1...
>>> eotf(0.182011532850008, function='ST 2084', L_p=1000)
...
0.1...
```

colour.EOTFS

colour.**EOTFS = CaseInsensitiveMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'SMPTE 240M': ..., 'ST 2084': ..., 'sRGB': ...})**

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', 'SMPTE 240M', 'ST 2084', 'sRGB'}**

colour.eotf_inverse

colour.**eotf_inverse**(*value*, *function='ITU-R BT.1886'*, ***kwargs*)

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse 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', 'ST 2084', 'sRGB'}**, Inverse electro-optical transfer function (EOTF / EOCF).
- **L_B** (numeric, optional) – {colour.models.eotf_inverse_BT1886(), colour.models.eotf_inverse_HLG_BT2100()}, Screen luminance for black.
- **L_W** (numeric, optional) – {colour.models.eotf_inverse_BT1886(), colour.models.eotf_inverse_HLG_BT2100()}, Screen luminance for white.
- **gamma** (numeric, optional) – {colour.models.eotf_HLG_BT2100()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **is_12_bits_system** (bool, optional) – {colour.models.eotf_inverse_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.
- **L_p** (numeric, optional) – {colour.models.eotf_inverse_ST2084()}, Display peak luminance cd/m^2.
- **method** (unicode, optional) – {colour.models.eotf_inverse_HLG_BT2100()}, **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**
- **out_int** (bool, optional) – {colour.models.eotf_inverse_DCDM(), colour.models.eotf_inverse_DICOMGSDF()}, 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_inverse(0.11699185725296059)
0.4090077...
>>> eotf_inverse(
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.EOTF_INVERSES

colour.**EOTF_INVERSES = CaseInsensitiveMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ..., 'ST 2084': ..., 'sRGB': ...})**

Supported inverse electro-optical transfer functions (EOTFs / EOCFs).

EOTF_INVERSES [CaseInsensitiveMapping] **{'DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ST 2084', 'sRGB'}**

colour.models

eotf_DCDM(XYZ_p[, in_int])	Defines the *DCDM* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_DCDM(XYZ[, out_int])	Defines the *DCDM* inverse electro-optical transfer function (EOTF / EOCF).
eotf_DICOMGSDF(J[, in_int])	Defines the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_DICOMGSDF(L[, out_int])	Defines the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF / EOCF).
eotf_BT1886(V[, L_B, L_W])	Defines *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_BT1886(L[, L_B, L_W])	Defines *Recommendation ITU-R BT.1886* inverse electro-optical transfer function (EOTF / EOCF).
eotf_BT2020(E_p[, is_12_bits_system, constants])	Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_BT2020(E[, is_12_bits_system, ...])	Defines *Recommendation ITU-R BT.2020* inverse electro-optical transfer function (EOTF / EOCF).
BT2100_HLG_EOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).
eotf_HLG_BT2100(E_p[, L_B, L_W, gamma, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).
BT2100_HLG_EOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).
eotf_inverse_HLG_BT2100(F_D[, L_B, L_W, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).
eotf_PQ_BT2100(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_PQ_BT2100(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse electro-optical transfer function (EOTF / EOCF).
eotf_SMPTE240M(V_r)	Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).
eotf_ST2084(N[, L_p, constants])	Defines *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF / EOCF).
eotf_inverse_ST2084(C[, L_p, constants])	Defines *SMPTE ST 2084:2014* optimised perceptual inverse electro-optical transfer function (EOTF / EOCF).
eotf_sRGB(V)	Defines the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_sRGB(L)	Defines the *IEC 61966-2-1:1999 sRGB* inverse 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

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* 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.

Domain *	Scale - Reference	Scale - 1
XYZ_p	UN	UN

Range *	Scale - Reference	Scale - 1
XYZ	UN	UN

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[]

Examples

```
>>> eotf_DCDM(0.11281860951766724)
0.18...
>>> eotf_DCDM(462, in_int=True)
0.18...
```

colour.models.eotf_inverse_DCDM

colour.models.**eotf_inverse_DCDM**(*XYZ*, *out_int=False*)

Defines the *DCDM* inverse 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

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* 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.

Domain *	Scale - Reference	Scale - 1
XYZ	UN	UN

Range *	Scale - Reference	Scale - 1
XYZ_p	UN	UN

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[]

Examples

```
>>> eotf_inverse_DCDM(0.18)
0.1128186...
>>> eotf_inverse_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

[]

Examples

```
>>> eotf_DICOMGSDF(0.500486263438448)
130.0628647...
>>> eotf_DICOMGSDF(512, in_int=True)
130.0652840...
```

colour.models.eotf_inverse_DICOMGSDF

colour.models.**eotf_inverse_DICOMGSDF**(*L*, *out_int=False*)

Defines the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF / EOCF).

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

[]

Examples

```
>>> eotf_inverse_DICOMGSDF(130.0662)
0.5004862...
>>> eotf_inverse_DICOMGSDF(130.0662, out_int=True)
512
```

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

[]

Examples

```
>>> eotf_BT1886(0.409007728864150)
0.1169918...
```

colour.models.eotf_inverse_BT1886

colour.models.**eotf_inverse_BT1886**(*L*, *L_B=0*, *L_W=1*)

Defines *Recommendation ITU-R BT.1886* inverse 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

[]

Examples

```
>>> eotf_inverse_BT1886(0.11699185725296059)
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, optional) – *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 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

[]

Examples

```
>>> eotf_BT2020(0.705515089922121)
0.4999999...
```

colour.models.eotf_inverse_BT2020

colour.models.**eotf_inverse_BT2020**(*E*, *is_12_bits_system=False*, *constants={'alpha': <function <lambda>>, 'beta': <function <lambda>>}*)

Defines *Recommendation ITU-R BT.2020* inverse electro-optical transfer function (EOTF / EOCF).

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, optional) – *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

[]

Examples

```
>>> eotf_inverse_BT2020(0.18)
0.4090077...
```

colour.models.BT2100_HLG_EOTF_METHODS

colour.models.**BT2100_HLG_EOTF_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).

References

[], [], []

BT2100_HLG_EOTF_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.eotf_HLG_BT2100

colour.models.**eotf_HLG_BT2100**(*E_p*, *L_B=0*, *L_W=1000*, *gamma=None*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*, *method='ITU-R BT.2100-2'*)

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 $1000 cd/m^2$.

- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

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

[], [], []

Examples

```
>>> eotf_HLG_BT2100(0.212132034355964)
6.4760398...
>>> eotf_HLG_BT2100(0.212132034355964, method='ITU-R BT.2100-1')
...
6.4760398...
>>> eotf_HLG_BT2100(0.212132034355964, 0.01)
...
7.3321975...
```

colour.models.BT2100_HLG_EOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_EOTF_INVERSE_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).

References

[], [], []

BT2100_HLG_EOTF_INVERSE_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.eotf_inverse_HLG_BT2100

colour.models.**eotf_inverse_HLG_BT2100**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991072952956}*, *method='ITU-R BT.2100-2'*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse 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.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

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

[], [], []

Examples

```
>>> eotf_inverse_HLG_BT2100(6.476039825649814)
0.2121320...
>>> eotf_inverse_HLG_BT2100(6.476039825649814, method='ITU-R BT.2100-1')
...
0.2121320...
>>> eotf_inverse_HLG_BT2100(7.332197528353875, 0.01)
0.2121320...
```

colour.models.eotf_PQ_BT2100

colour.models.**eotf_PQ_BT2100**(*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

[], []

Examples

```
>>> eotf_PQ_BT2100(0.724769816665726)
779.9883608...
```

colour.models.eotf_inverse_PQ_BT2100

colour.models.**eotf_inverse_PQ_BT2100**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse 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

[], []

Examples

```
>>> eotf_inverse_PQ_BT2100(779.988360834085370)
0.7247698...
```

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

[]

Examples

```
>>> eotf_SMPTE240M(0.402285796753870)
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

- *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.

Domain	Scale - Reference	Scale - 1
N	UN	UN

Range	Scale - Reference	Scale - 1
C	UN	UN

References

[], []

Examples

```
>>> eotf_ST2084(0.508078421517399)
100.0000000...
```

colour.models.eotf_inverse_ST2084

colour.models.**eotf_inverse_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 inverse electro-optical transfer function (EOTF / EOCF).

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 $10000cd/m^2$ 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 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

- *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. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF / EOCF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
C	UN	UN

Range	Scale - Reference	Scale - 1
N	UN	UN

References

[], []

Examples

```
>>> eotf_inverse_ST2084(100)
0.5080784...
```

colour.models.eotf_sRGB

colour.models.**eotf_sRGB**(*V*)

Defines the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF / EOCF).

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
V	[0, 1]	[0, 1]

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

References

[], []

Examples

```
>>> eotf_sRGB(0.461356129500442)
0.1...
```

colour.models.eotf_inverse_sRGB

colour.models.**eotf_inverse_sRGB**(*L*)

Defines the *IEC 61966-2-1:1999 sRGB* inverse electro-optical transfer function (EOTF / EOCF).

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

[], []

Examples

```
>>> eotf_inverse_sRGB(0.18)
0.4613561...
```

Opto-Optical Transfer Functions

colour

ootf(value[, function])	Maps relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).
OOTFS	Supported opto-optical transfer functions (OOTFs / OOCFs).
ootf_inverse(value[, function])	Maps relative display linear light to scene linear light using given inverse opto-optical transfer function (OOTF / OOCF).
OOTF_INVERSES	Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

colour.ootf

colour.**ootf**(*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)
779.9883608...
>>> ootf(0.1, function='ITU-R BT.2100 HLG')
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_inverse

colour.**ootf_inverse**(*value*, *function='ITU-R BT.2100 PQ'*, ***kwargs*)

Maps relative display linear light to scene linear light using given inverse 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'}** Inverse opto-optical transfer function (OOTF / OOCF).
- **L_B** (numeric, optional) – {colour.models.ootf_inverse_HLG_BT2100()}, L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – {colour.models.ootf_inverse_HLG_BT2100()}, L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – {colour.models.ootf_inverse_HLG_BT2100()}, 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_inverse(779.988360834115840)
0.1000000...
>>> ootf_inverse(
...     63.095734448019336, function='ITU-R BT.2100 HLG')
0.1000000...
```

colour.OOTF_INVERSES

colour.**OOTF_INVERSES = CaseInsensitiveMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})**

Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

OOTF_INVERSES [CaseInsensitiveMapping] **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.models

BT2100_HLG_OOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
ootf_HLG_BT2100(E[, L_B, L_W, gamma, method])	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
BT2100_HLG_OOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_inverse_HLG_BT2100(F_D[, L_B, L_W, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_PQ_BT2100(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).
ootf_inverse_PQ_BT2100(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-optical transfer function (OOTF / OOCF).

colour.models.BT2100_HLG_OOTF_METHODS

colour.models.**BT2100_HLG_OOTF_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

References

[], [], []

BT2100_HLG_OOTF_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.ootf_HLG_BT2100

colour.models.**ootf_HLG_BT2100**(*E*, *L_B=0*, *L_W=1000*, *gamma=None*, *method='ITU-R BT.2100-2'*)

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 $1000cd/m^2$.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

Returns F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, 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

[], []

Examples

```
>>> ootf_HLG_BT2100(0.1)
63.0957344...
>>> ootf_HLG_BT2100(0.1, 0.01, method='ITU-R BT.2100-1')
...
63.1051034...
```

colour.models.BT2100_HLG_OOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_OOTF_INVERSE_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2': ...})**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).

References

[], [], []

BT2100_HLG_OOTF_INVERSE_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.ootf_inverse_HLG_BT2100

colour.models.**ootf_inverse_HLG_BT2100**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*, *method='ITU-R BT.2100-2'*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse 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, orB_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 $1000cd/m^2$.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

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

[], [], []

Examples

```
>>> ootf_inverse_HLG_BT2100(63.095734448019336)
0.1000000...
>>> ootf_inverse_HLG_BT2100(
...     63.105103490674857, 0.01, method='ITU-R BT.2100-1')
...
0.0999999...
```

colour.models.ootf_PQ_BT2100

colour.models.**ootf_PQ_BT2100**(*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) – $E = R_S, G_S, B_S; Y_S; or I_S$ 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; 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

[], []

Examples

```
>>> ootf_PQ_BT2100(0.1)
779.9883608...
```

colour.models.ootf_inverse_PQ_BT2100

colour.models.**ootf_inverse_PQ_BT2100**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse 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; or I_D).

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ 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

[], []

Examples

```
>>> ootf_inverse_PQ_BT2100(779.988360834115840)
0.1000000...
```

Log Encoding and Decoding

colour

log_encoding(value[, function])	Encodes linear-light values to $R'G'B'$ video component signal value using given *log* function.
LOG_ENCODINGS	Supported *log* encoding functions.
log_decoding(value[, function])	Decodes $R'G'B'$ video component signal value to linear-light values using given *log* function.
LOG_DECODINGS	Supported *log* decoding functions.

colour.log_encoding

colour.**log_encoding**(*value*, *function='Cineon'*, ***kwargs*)

Encodes linear-light values to $R'G'B'$ video component signal value using given *log* function.

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log2', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**, Computation function.
- **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.
- **method** (unicode, optional) – {colour.models.log_encoding_Log3G10()}, Whether to use the *Log3G10 v1* or *v2* log encoding curve.
- **out_normalised_code_value** (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 as normalised code values.
- **negative_gamma** (numeric or array_like) – {colour.models.log_encoding_PivotedLog()}, Negative gamma.
- **method** – {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(0.18)
0.4573196...
>>> log_encoding(0.18, function='ACEScc')
0.4135884...
>>> log_encoding(0.18, function='PLog', log_reference=400)
...
0.3910068...
>>> log_encoding(0.18, function='S-Log')
0.3849708...
```

colour.LOG_ENCODINGS

colour.**LOG_ENCODINGS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log C': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'Filmic Pro 6': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'Panalog': ..., 'PLog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ...})**

Supported *log* encoding functions.

LOG_ENCODINGS [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log2', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**

colour.log_decoding

colour.**log_decoding**(*value*, *function='Cineon'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to linear-light values using given *log* function.

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log2', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**, Computation function.
- **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_normalised_code_value** (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 as normalised code values.

- **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.
- **method** (unicode, optional) – {colour.models.log_decoding_Log3G10()}, Whether to use the *Log3G10 v1* or *v2* log encoding curve.
- **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** – {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(0.457319613085418)
0.1...
>>> log_decoding(0.413588402492442, function='ACEScc')
...
0.1...
>>> log_decoding(0.391006842619746, function='PLog', log_reference=400)
...
0.1...
>>> log_decoding(0.376512722254600, function='S-Log')
...
0.1...
```

colour.LOG_DECODINGS

colour.**LOG_DECODINGS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA Log C': ..., 'Canon Log 2': ..., 'Canon Log 3': ..., 'Canon Log': ..., 'Cineon': ..., 'D-Log': ..., 'ERIMM RGB': ..., 'F-Log': ..., 'Filmic Pro 6': ..., 'Log2': ..., 'Log3G10': ..., 'Log3G12': ..., 'Panalog': ..., 'PLog': ..., 'Protune': ..., 'REDLog': ..., 'REDLogFilm': ..., 'S-Log': ..., 'S-Log2': ..., 'S-Log3': ..., 'T-Log': ..., 'V-Log': ..., 'ViperLog': ...})**

Supported *log* decoding functions.

LOG_DECODINGS [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log2', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**

colour.models

log_encoding_ACEScc(lin_AP1)	Defines the *ACEScc* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScc(ACEScc)	Defines the *ACEScc* colourspace log decoding / electro-optical transfer function.

continues on next page

Table 2 – continued from previous page

log_encoding_ACEScct(lin_AP1[, constants])	Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScct(ACEScct[, constants])	Defines the *ACEScct* colourspace log decoding / electro-optical transfer function.
log_encoding_ACESproxy(lin_AP1[, bit_depth, ...])	Defines the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.
log_decoding_ACESproxy(ACESproxy[, ...])	Defines the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.
log_encoding_ALEXALogC(x[, firmware, method, EI])	Defines the *ARRI ALEXA Log C* log encoding curve / opto-electronic transfer function.
log_decoding_ALEXALogC(t[, firmware, method, EI])	Defines the *ARRI ALEXA Log C* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog2(x[, bit_depth, ...])	Defines the *Canon Log 2* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog2(clog2[, bit_depth, ...])	Defines the *Canon Log 2* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog3(x[, bit_depth, ...])	Defines the *Canon Log 3* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog3(clog3[, bit_depth, ...])	Defines the *Canon Log 3* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog(x[, bit_depth, ...])	Defines the *Canon Log* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog(clog[, bit_depth, ...])	Defines the *Canon Log* log decoding curve / electro-optical transfer function.
log_encoding_Cineon(x[, black_offset])	Defines the *Cineon* log encoding curve / opto-electronic transfer function.
log_decoding_Cineon(y[, black_offset])	Defines the *Cineon* log decoding curve / electro-optical transfer function.
log_encoding_ERIMMRGB(X[, bit_depth, ...])	Defines the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF / OECF).
log_decoding_ERIMMRGB(X_p[, bit_depth, ...])	Defines the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF / EOCF).
log_encoding_FLog(in_r[, bit_depth, ...])	Defines the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.
log_decoding_FLog(out_r[, bit_depth, ...])	Defines the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.
log_encoding_Log2(lin[, middle_grey, ...])	Defines the common *Log2* encoding function.
log_decoding_Log2(log_norm[, middle_grey, ...])	Defines the common *Log2* decoding function.
LOG3G10_ENCODING_METHODS	Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.
log_encoding_Log3G10(x[, method])	Defines the *Log3G10* log encoding curve / opto-electronic transfer function.
LOG3G10_DECODING_METHODS	Supported *Log3G10* log decoding curve / electro-optical transfer function methods.
log_decoding_Log3G10(y[, method])	Defines the *Log3G10* log decoding curve / electro-optical transfer function.
log_encoding_Log3G12(x)	Defines the *Log3G12* log encoding curve / opto-electronic transfer function.
log_decoding_Log3G12(y)	Defines the *Log3G12* log decoding curve / electro-optical transfer function.
log_encoding_Panalog(x[, black_offset])	Defines the *Panalog* log encoding curve / opto-electronic transfer function.
log_decoding_Panalog(y[, black_offset])	Defines the *Panalog* log decoding curve / electro-optical transfer function.

continues on next page

Table 2 – continued from previous page

log_encoding_PivotedLog(x[, log_reference, ...])	Defines the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.
log_decoding_PivotedLog(y[, log_reference, ...])	Defines the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.
log_encoding_Protune(x)	Defines the *Protune* log encoding curve / opto-electronic transfer function.
log_decoding_Protune(y)	Defines the *Protune* log decoding curve / electro-optical transfer function.
log_encoding_REDLog(x[, black_offset])	Defines the *REDLog* log encoding curve / opto-electronic transfer function.
log_decoding_REDLog(y[, black_offset])	Defines the *REDLog* log decoding curve / electro-optical transfer function.
log_encoding_REDLogFilm(x[, black_offset])	Defines the *REDLogFilm* log encoding curve / opto-electronic transfer function.
log_decoding_REDLogFilm(y[, black_offset])	Defines the *REDLogFilm* log decoding curve / electro-optical transfer function.
log_encoding_SLog(x[, bit_depth, ...])	Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.
log_decoding_SLog(y[, bit_depth, ...])	Defines the *Sony S-Log* log decoding curve / electro-optical transfer function.
log_encoding_SLog2(x[, bit_depth, ...])	Defines the *Sony S-Log2* log encoding curve / opto-electronic transfer function.
log_decoding_SLog2(y[, bit_depth, ...])	Defines the *Sony S-Log2* log decoding curve / electro-optical transfer function.
log_encoding_SLog3(x[, bit_depth, ...])	Defines the *Sony S-Log3* log encoding curve / opto-electronic transfer function.
log_decoding_SLog3(y[, bit_depth, ...])	Defines the *Sony S-Log3* log decoding curve / electro-optical transfer function.
log_encoding_VLog(L_in[, bit_depth, ...])	Defines the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.
log_decoding_VLog(V_out[, bit_depth, ...])	Defines the *Panasonic V-Log* log decoding curve / electro-optical transfer function.
log_encoding_ViperLog(x)	Defines the *Viper Log* log encoding curve / opto-electronic transfer function.
log_decoding_ViperLog(y)	Defines the *Viper Log* 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
lin_AP1	[0, 1]	[0, 1]

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

References

[], [], [], []

Examples

```
>>> log_encoding_ACEScc(0.18)
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

[], [], [], []

Examples

```
>>> log_decoding_ACEScc(0.413588402492442)
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

[], [], [], []

Examples

```
>>> log_encoding_ACEScct(0.18)
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

[], [], [], []

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)
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

[], [], [], []

Examples

```
>>> log_encoding_ACESproxy(0.18)
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

[], [], [], []

Examples

```
>>> log_decoding_ACESproxy(0.416422287390029)
0.1...
>>> log_decoding_ACESproxy(426, in_int=True)
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 *ARRI 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) – Ei.

Returns *ARRI ALEXA Log C* encoded data t.

Return type numeric or ndarray

References

[]

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)
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 *ARRI ALEXA Log C* log decoding curve / electro-optical transfer function.

Parameters

- **t** (numeric or array_like) – *ARRI 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) – Ei.

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

[]

Examples

```
>>> log_decoding_ALEXALogC(0.391006832034084)
0.18...
```

colour.models.log_encoding_CanonLog2

colour.models.**log_encoding_CanonLog2**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_encoding_CanonLog2(0.18) * 100
39.8254694...
```

colour.models.log_decoding_CanonLog2

colour.models.**log_decoding_CanonLog2**(*clog2*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_CanonLog2(39.825469498316735 / 100)
0.1799999...
```

colour.models.log_encoding_CanonLog3

colour.models.**log_encoding_CanonLog3**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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
```

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog3	[0, 1]	[0, 1]

References

[]

Examples

```
>>> log_encoding_CanonLog3(0.18) * 100
34.3389369...
```

colour.models.log_decoding_CanonLog3

colour.models.**log_decoding_CanonLog3**(*clog3*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_CanonLog3(34.338936938868677 / 100)
0.1800000...
```

colour.models.log_encoding_CanonLog

colour.models.**log_encoding_CanonLog**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log* non-linear data is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns *Canon Log* non-linear data.

Return type numeric or ndarray

References

[]

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
34.3389651...
```

The values of *Table 2 Canon-Log Code Values* table in [] are obtained as follows:

```
>>> x = np.array([0, 2, 18, 90, 720]) / 100
>>> np.around(log_encoding_CanonLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 128,  169,  351,  614, 1016])
>>> np.around(log_encoding_CanonLog(x, 10, False) * 100, 1)
array([   7.3,   12. ,   32.8,   62.7,  108.7])
```

colour.models.log_decoding_CanonLog

colour.models.**log_decoding_CanonLog**(*clog*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the *Canon Log* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_CanonLog(34.338965172606912 / 100)
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

[]

Examples

```
>>> log_encoding_Cineon(0.18)
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

[]

Examples

```
>>> log_decoding_Cineon(0.457319613085418)
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

[]

Examples

```
>>> log_encoding_ERIMMRGB(0.18)
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

[]

Examples

```
>>> log_decoding_ERIMMRGB(0.410052389492129)
0.1...
>>> log_decoding_ERIMMRGB(105, in_int=True)
0.1...
```

colour.models.log_encoding_FLog

colour.models.**log_encoding_FLog**(*in_r*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, *constants={'a': 0.555556, 'b': 0.009468, 'c': 0.344676, 'cut1': 0.00089, 'cut2': 0.100537775223865, 'd': 0.790453, 'e': 8.735631, 'f': 0.092864}*)

Defines the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **in_r** (numeric or array_like) – Linear reflection data :math`in`.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Fujifilm F-Log* data out is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level :math`in` to a camera is reflection.
- **constants** (Structure, optional) – *Fujifilm F-Log* constants.

Returns Non-linear data out.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> log_encoding_FLog(0.18)
0.4593184...
```

The values of *2-2. F-Log Code Value* table in [] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_FLog(x, 10, False) * 100, 1)
array([  3.5,  46.3,  73.2])
>>> np.around(log_encoding_FLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 95, 470, 705])
```

colour.models.log_decoding_FLog

colour.models.**log_decoding_FLog**(*out_r*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, *constants={'a': 0.555556, 'b': 0.009468, 'c': 0.344676, 'cut1': 0.00089, 'cut2': 0.100537775223865, 'd': 0.790453, 'e': 8.735631, 'f': 0.092864}*)

Defines the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.

Parameters

- **out_r** (numeric or array_like) – Non-linear data out.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Fujifilm F-Log* data out is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level :math`in` to a camera is reflection.
- **constants** (Structure, optional) – *Fujifilm F-Log* constants.

Returns Linear reflection data :math`in`.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> log_decoding_FLog(0.45931845866162124)
0.1800000...
```

colour.models.log_encoding_Log2

colour.models.**log_encoding_Log2**(*lin*, *middle_grey=0.18*, *min_exposure=- 6.5*, *max_exposure=6.5*)

Defines the common *Log2* encoding function.

Parameters

- **lin** (numeric or array_like) – Linear data to undergo encoding.
- **middle_grey** (numeric, optional) – *Middle Grey* exposure value.
- **min_exposure** (numeric, optional) – Minimum exposure level.
- **max_exposure** (numeric, optional) – Maximum exposure level.

Returns Non-linear *Log2* encoded data.

Return type numeric or ndarray

Notes

- The common *Log2* encoding function can be used to build linear to logarithmic shapers in the *ACES OCIO configuration*.
- A (48-nits OCIO) shaper having values in a linear domain, can be encoded to a logarithmic domain:

Shaper Domain	Shaper Range
[0.002, 16.291]	[0, 1]

References

[]

Examples

```
>>> log_encoding_Log2(0.18)
0.5
```

colour.models.log_decoding_Log2

colour.models.**log_decoding_Log2**(*log_norm*, *middle_grey=0.18*, *min_exposure=- 6.5*, *max_exposure=6.5*)

Defines the common *Log2* decoding function.

Parameters

- **log_norm** (numeric or array_like) – Logarithmic data to undergo decoding.
- **middle_grey** (numeric, optional) – *Middle Grey* exposure value.
- **min_exposure** (numeric, optional) – Minimum exposure level.
- **max_exposure** (numeric, optional) – Maximum exposure level.

Returns Linear *Log2* decoded data.

Return type numeric or ndarray

Notes

- The common *Log2* decoding function can be used to build logarithmic to linear shapers in the *ACES OCIO configuration*.
- The shaper with logarithmic encoded values can be decoded back to linear domain:

Shaper Range	Shaper Domain
[0, 1]	[0.002, 16.291]

References

[]

Examples

```
>>> log_decoding_Log2(0.5)
0.1799999...
```

colour.models.LOG3G10_ENCODING_METHODS

colour.models.**LOG3G10_ENCODING_METHODS = CaseInsensitiveMapping({'v1': ..., 'v2': ...})**

Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.

References

[]

LOG3G10_ENCODING_METHODS [CaseInsensitiveMapping] **{'v1', 'v2'}**

colour.models.log_encoding_Log3G10

colour.models.**log_encoding_Log3G10**(*x*, *method='v2'*, ***kwargs*)

Defines the *Log3G10* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **method** (unicode, optional) – **{'v1', 'v2'}**, Computation method.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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 *Log3G10 v1* 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 the *Log3G10 v2* log encoding curve in the release version of the *RED SDK*.
- The intent of the *Log3G10 v1* 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

[]

Examples

```
>>> log_encoding_Log3G10(0.0)
0.0915514...
>>> log_encoding_Log3G10(0.18, method='v1')
0.3333336...
```

colour.models.LOG3G10_DECODING_METHODS

colour.models.**LOG3G10_DECODING_METHODS = CaseInsensitiveMapping({'v1': ..., 'v2': ...})**

Supported *Log3G10* log decoding curve / electro-optical transfer function methods.

References

[]

LOG3G10_DECODING_METHODS [CaseInsensitiveMapping] **{'v1', 'v2'}**

colour.models.log_decoding_Log3G10

colour.models.**log_decoding_Log3G10**(*y*, *method='v2'*, ***kwargs*)

Defines the *Log3G10* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **method** (unicode, optional) – **{'v1', 'v2'}**, Computation method.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_Log3G10(1.0)
184.3223476...
>>> log_decoding_Log3G10(1.0 / 3, method='v1')
0.1799994...
```

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
x	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> log_encoding_Log3G12(0.18)
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

[]

Examples

```
>>> log_decoding_Log3G12(1.0 / 3)
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

[]

Examples

```
>>> log_encoding_Panalog(0.18)
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

[]

Examples

```
>>> log_decoding_Panalog(0.374576791382298)
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

[]

Examples

```
>>> log_encoding_PivotedLog(0.18)
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

[]

Examples

```
>>> log_decoding_PivotedLog(0.434995112414467)
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

[]

Examples

```
>>> log_encoding_Protune(0.18)
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
y	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> log_decoding_Protune(0.645623486803636)
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
x	[0, 1]	[0, 1]

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

References

[]

Examples

```
>>> log_encoding_REDLog(0.18)
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

[]

Examples

```
>>> log_decoding_REDLog(0.637621845988175)
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

[]

Examples

```
>>> log_encoding_REDLogFilm(0.18)
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

[]

Examples

```
>>> log_decoding_REDLogFilm(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_SLog

colour.models.**log_encoding_SLog**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_encoding_SLog(0.18)
0.3849708...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog(x, 10, False) * 100).astype(np.int)
array([ 3, 38, 65])
>>> np.around(log_encoding_SLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 90, 394, 636])
```

colour.models.log_decoding_SLog

colour.models.**log_decoding_SLog**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_SLog(0.384970815928670)
0.1...
```

colour.models.log_encoding_SLog2

colour.models.**log_encoding_SLog2**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.

- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_encoding_SLog2(0.18)
0.3395325...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog2(x, 10, False) * 100).astype(np.int)
array([ 3, 32, 59])
>>> np.around(log_encoding_SLog2(x) * (2 ** 10 - 1)).astype(np.int)
array([ 90, 347, 582])
```

colour.models.log_decoding_SLog2

colour.models.**log_decoding_SLog2**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_SLog2(0.339532524633774)
0.1...
```

colour.models.log_encoding_SLog3

colour.models.**log_encoding_SLog3**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_encoding_SLog3(0.18)
0.4105571...
```

The values of *S-Log3 10bit code values (18%, 90%)* table in [] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog3(x, 10, False) * 100).astype(np.int)
array([ 4, 41, 61])
>>> np.around(log_encoding_SLog3(x) * (2 ** 10 - 1)).astype(np.int)
array([ 95, 420, 598])
```

colour.models.log_decoding_SLog3

colour.models.**log_decoding_SLog3**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_SLog3(0.410557184750733)
0.1...
```

colour.models.log_encoding_VLog

colour.models.**log_encoding_VLog**(*L_in*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, *constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206}*, ***kwargs*)

Defines the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **L_in** (numeric or array_like) – Linear reflection data :math`L_{in}`.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_encoding_VLog(0.18)
0.4233114...
```

The values of *Fig.2.2 V-Log Code Value* table in [] are obtained as follows:

```
>>> L_in = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_VLog(L_in, 10, False) * 100).astype(np.int)
array([ 7, 42, 61])
>>> np.around(log_encoding_VLog(L_in) * (2 ** 10 - 1)).astype(np.int)
array([128, 433, 602])
>>> np.around(log_encoding_VLog(L_in) * (2 ** 12 - 1)).astype(np.int)
array([ 512, 1733, 2409])
```

Note that some values in the last column values of *Fig.2.2 V-Log Code Value* table in [] are different by a code: [512, 1732, 2408].

colour.models.log_decoding_VLog

colour.models.**log_decoding_VLog**(*V_out*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, *constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206}*, ***kwargs*)

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_normalised_code_value** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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

[]

Examples

```
>>> log_decoding_VLog(0.423311448760136)
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

[]

Examples

```
>>> log_encoding_ViperLog(0.18)
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

[]

Examples

```
>>> log_decoding_ViperLog(0.636008067010413)
0.1799999...
```

Colour Encodings

Y'CbCr Colour Encoding

colour

RGB_to_YCbCr(RGB[, K, in_bits, in_legal, ...])	Converts an array of *R'G'B'* values to the corresponding *Y'CbCr* colour encoding values array.
YCbCr_to_RGB(YCbCr[, K, in_bits, in_legal, ...])	Converts an array of *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.
WEIGHTS_YCBCR	Implements a case-insensitive mutable mapping / *dict* object.
RGB_to_YcCbcCrc(RGB[, out_bits, out_legal, ...])	Converts an array of *RGB* linear values to the corresponding *Yc'Cbc'Crc'* colour encoding values array.
YcCbcCrc_to_RGB(YcCbcCrc[, in_bits, ...])	Converts an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* 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 *Y'CbCr* 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.WEIGHTS_YCBCR 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*.
- **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 *Y'CbCr* 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_YcCbcCrc() definition should be used for the constant luminance case as per [].

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

[], [], [], []

Examples

```
>>> RGB = np.array([1.0, 1.0, 1.0])
>>> RGB_to_YCbCr(RGB)
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))
...
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)
...
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 []:

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(RGB, K=WEIGHTS_YCBCR['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=WEIGHTS_YCBCR['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 *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.

Parameters

- **YCbCr** (array_like) – Input *Y'CbCr* colour encoding array of integer or float values.
- **K** (array_like, optional) – Luma weighting coefficients of red and blue. See colour.WEIGHTS_YCBCR 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 *8*.
- **in_legal** (bool, optional) – Whether to treat the input values as legal range. Default is *True*.
- **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 *10*.
- **out_legal** (bool, optional) – Whether to return legal range values. Default is *False*.
- **out_int** (bool, optional) – Whether to return values as out_bits integer code values. Default is *False*.
- **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.YcCbcCrc_to_RGB()` definition should be used for the constant luminance case as per [].

References

[], [], [], []

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.WEIGHTS_YCBCR

colour.**WEIGHTS_YCBCR = CaseInsensitiveMapping({'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-R BT.2020': ..., 'SMPTE-240M': ...})**

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.
- ****kwargs** (`dict, optional`) – Key / Value pairs to store into the mapping at initialisation.

Attributes

- `data`

Methods

- `__init__()`
- `__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

[]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

colour.RGB_to_YcCbcCrc

colour.**RGB_to_YcCbcCrc**(*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 *Yc'Cbc'Crc'* 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*.
- **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'Crc'* 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
YcCbcCrc	[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

[], []

Examples

```
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> RGB_to_YcCbcCrc(RGB, out_legal=True, out_bits=10, out_int=True,
...                 is_12_bits_system=False)
...
array([422, 512, 512]...)
```

colour.YcCbcCrc_to_RGB

colour.**YcCbcCrc_to_RGB**(*YcCbcCrc*, *in_bits=10*, *in_legal=True*, *in_int=False*, *is_12_bits_system=False*, ***kwargs*)

Converts an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* array of linear values.

Parameters

- **YcCbcCrc** (array_like) – Input *Yc'Cbc'Crc'* 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*.
- **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
YcCbcCrc	[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

[], []

Examples

```
>>> YcCbcCrc = np.array([1689, 2048, 2048])
>>> YcCbcCrc_to_RGB(YcCbcCrc, in_legal=True, in_bits=12, in_int=True,
...                 is_12_bits_system=True)
...
array([ 0.1800903...,  0.1800903...,  0.1800903...])
```

Ancillary Objects

colour

full_to_legal(CV[, bit_depth, in_int, out_int])	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).
legal_to_full(CV[, bit_depth, in_int, out_int])	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).
CV_range([bit_depth, is_legal, is_int])	Returns the code value CV 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)
0.0625610...
>>> full_to_legal(1.0)
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)
0.0625610...
>>> full_to_legal(1023, in_int=True)
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)
array([ 0.0627451...,  0.9215686...])
>>> CV_range(10, False, False)
array([ 0.,  1.])
```

YCoCg Colour Encoding

colour

RGB_to_YCoCg(RGB)	Converts an array of *R'G'B'* values to the corresponding *YCoCg* colour encoding values array.
YCoCg_to_RGB(YCoCg)	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

[]

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

[]

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

RGB_to_ICTCP(RGB[, L_p])	Converts from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.
ICTCP_to_RGB(ICTCP[, L_p])	Converts from IC_TC_P colour encoding to *ITU-R BT.2020* 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. 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.

Returns IC_TC_P colour encoding array.

Return type ndarray

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function 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. The effective domain of *SMPTE ST 2084:2014* inverse electro-optical transfer function (EOTF / EOCF) is [0.0001, 10000].

Domain	Scale - Reference	Scale - 1
RGB	UN	UN

Range	Scale - Reference	Scale - 1
ICTCP	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

References

[], []

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_ICTCP(RGB)
array([ 0.0735136...,  0.0047525...,  0.0935159...])
```

colour.ICTCP_to_RGB

colour.**ICTCP_to_RGB**(*ICTCP*, *L_p=10000*)

Converts from IC_TC_P colour encoding to *ITU-R BT.2020* colourspace.

Parameters

- **ICTCP** (array_like) – IC_TC_P colour encoding array.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding. 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.

Returns *ITU-R BT.2020* colourspace array.

Return type ndarray

Warning: The underlying *SMPTE ST 2084:2014* transfer function is an absolute transfer function.

Notes

- The underlying *SMPTE ST 2084:2014* transfer function 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.

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	UN	UN

References

[], []

Examples

```
>>> ICTCP = np.array([0.07351364, 0.00475253, 0.09351596])
>>> ICTCP_to_RGB(ICTCP)
array([ 0.4562052...,  0.0308107...,  0.0409195...])
```

RGB Representations

Prismatic Colourspace

colour

RGB_to_Prismatic(RGB)	Converts from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.
Prismatic_to_RGB(Lrgb)	Converts from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* 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

[]

Examples

```
>>> RGB = np.array([0.25, 0.50, 0.75])
>>> RGB_to_Prismatic(RGB)
array([ 0.75...   ,  0.1666666...,  0.3333333...,  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

[]

Examples

```
>>> Lrgb = np.array([0.75000000, 0.16666667, 0.33333333, 0.50000000])
>>> Prismatic_to_RGB(Lrgb)
array([ 0.25...   ,  0.4999999...,  0.75...  ])
```

HSV Colourspace

colour

RGB_to_HSV(RGB)	Converts from *RGB* colourspace to *HSV* colourspace.
HSV_to_RGB(HSV)	Converts from *HSV* colourspace to *RGB* 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

[], [], []

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSV(RGB)
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

[], [], []

Examples

```
>>> HSV = np.array([0.99603944, 0.93246304, 0.45620519])
>>> HSV_to_RGB(HSV)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

HSL Colourspace

colour

RGB_to_HSL(RGB)	Converts from *RGB* colourspace to *HSL* colourspace.
HSL_to_RGB(HSL)	Converts from *HSL* colourspace to *RGB* 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

[], [], []

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSL(RGB)
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

[], [], []

Examples

```
>>> HSL = np.array([0.99603944, 0.87347144, 0.24350795])
>>> HSL_to_RGB(HSL)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

CMY Colourspace

colour

RGB_to_CMY(RGB)	Converts from *RGB* colourspace to *CMY* colourspace.
CMY_to_RGB(CMY)	Converts from *CMY* colourspace to *CMY* colourspace.
CMY_to_CMYK(CMY)	Converts from *CMY* colourspace to *CMYK* colourspace.
CMYK_to_CMY(CMYK)	Converts from *CMYK* colourspace to *CMY* 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

[]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_CMY(RGB)
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

[]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_RGB(CMY)
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

[]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_CMYK(CMY)
array([ 0.        ,  0.9324630...,  0.9103045...,  0.5437948...])
```

colour.CMYK_to_CMY

colour.**CMYK_to_CMY**(*CMYK*)

Converts from *CMYK* colourspace to *CMY* colourspace.

Parameters **CMYK** (array_like) – *CMYK* colourspace 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

[]

Examples

```
>>> CMYK = np.array([0.50000000, 0.00000000, 0.74400000, 0.01960784])
>>> CMYK_to_CMY(CMYK)
array([ 0.5098039...,  0.0196078...,  0.7490196...])
```

Pointer's Gamut

colour

CCS_ILLUMINANT_POINTER_GAMUT	ndarray(shape, dtype=float, buffer=None, offset=0,
DATA_POINTER_GAMUT_VOLUME	ndarray(shape, dtype=float, buffer=None, offset=0,
CCS_POINTER_GAMUT_BOUNDARY	ndarray(shape, dtype=float, buffer=None, offset=0,

colour.models.CCS_ILLUMINANT_POINTER_GAMUT

colour.models.**CCS_ILLUMINANT_POINTER_GAMUT = array([0.31005673, 0.3161457])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.

- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time `(2 * 4)`.

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

zeros Create an array, each element of which is zero.

empty Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

dtype Create a data-type.

numpy.typing.NDArray A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.models.DATA_POINTER_GAMUT_VOLUME

colour.models.**DATA_POINTER_GAMUT_VOLUME = array([[15, 10, 0], [15, 15, 10], [15, 14, 20], ..., [90, 9, 330], [90, 4, 340], [90, 6, 350]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the `__new__` method; see Notes) –
- **shape** (`tuple of ints`) – Shape of created array.
- **dtype** (`data-type, optional`) – Any object that can be interpreted as a numpy data type.
- **buffer** (`object exposing buffer interface, optional`) – Used to fill the array with data.
- **offset** (`int, optional`) – Offset of array data in buffer.
- **strides** (`tuple of ints, optional`) – Strides of data in memory.
- **order** (`{'C', 'F'}, optional`) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

colour.models.CCS_POINTER_GAMUT_BOUNDARY

colour.models.**CCS_POINTER_GAMUT_BOUNDARY = array([[0.659, 0.316], [0.634, 0.351], [0.594, 0.391], [0.557, 0.427], [0.523, 0.462], [0.482, 0.491], [0.444, 0.515], [0.409, 0.546], [0.371, 0.558], [0.332, 0.573], [0.288, 0.584], [0.242, 0.576], [0.202, 0.53], [0.177, 0.454], [0.151, 0.389], [0.151, 0.33], [0.162, 0.295], [0.157, 0.266], [0.159, 0.245], [0.142, 0.214], [0.141, 0.195], [0.129, 0.168], [0.138, 0.141], [0.145, 0.129], [0.145, 0.106], [0.161, 0.094], [0.188, 0.084], [0.252, 0.104], [0.324, 0.127], [0.393, 0.165], [0.451, 0.199], [0.508, 0.226]])**

ndarray(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)

An array object represents a multidimensional, homogeneous array of fixed-size items. An associated data-type object describes the format of each element in the array (its byte-order, how many bytes it occupies in memory, whether it is an integer, a floating point number, or something else, etc.)

Arrays should be constructed using *array*, *zeros* or *empty* (refer to the See Also section below). The parameters given here refer to a low-level method (*ndarray(...)*) for instantiating an array.

For more information, refer to the *numpy* module and examine the methods and attributes of an array.

Parameters

- **below)** ((for the __new__ method; see Notes) –
- **shape** (tuple of ints) – Shape of created array.
- **dtype** (data-type, optional) – Any object that can be interpreted as a numpy data type.
- **buffer** (object exposing buffer interface, optional) – Used to fill the array with data.
- **offset** (int, optional) – Offset of array data in buffer.
- **strides** (tuple of ints, optional) – Strides of data in memory.
- **order** ({'C', 'F'}, optional) – Row-major (C-style) or column-major (Fortran-style) order.

Attributes

T [ndarray] Transpose of the array.

data [buffer] The array's elements, in memory.

dtype [dtype object] Describes the format of the elements in the array.

flags [dict] Dictionary containing information related to memory use, e.g., 'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.

flat [numpy.flatiter object] Flattened version of the array as an iterator. The iterator allows assignments, e.g., `x.flat = 3` (See *ndarray.flat* for assignment examples; TODO).

imag [ndarray] Imaginary part of the array.

real [ndarray] Real part of the array.

size [int] Number of elements in the array.

itemsize [int] The memory use of each array element in bytes.

nbytes [int] The total number of bytes required to store the array data, i.e., `itemsize * size`.

ndim [int] The array's number of dimensions.

shape [tuple of ints] Shape of the array.

strides [tuple of ints] The step-size required to move from one element to the next in memory. For example, a contiguous `(3, 4)` array of type `int16` in C-order has strides `(8, 2)`. This implies that to move from element to element in memory requires jumps of 2 bytes. To move from row-to-row, one needs to jump 8 bytes at a time (`2 * 4`).

ctypes [ctypes object] Class containing properties of the array needed for interaction with ctypes.

base [ndarray] If the array is a view into another array, that array is its *base* (unless that array is also a view). The *base* array is where the array data is actually stored.

See also:

`array` Construct an array.

`zeros` Create an array, each element of which is zero.

`empty` Create an array, but leave its allocated memory unchanged (i.e., it contains "garbage").

`dtype` Create a data-type.

`numpy.typing.NDArray` A generic version of ndarray.

Notes

There are two modes of creating an array using `__new__`:

1. If *buffer* is None, then only *shape*, *dtype*, and *order* are used.
2. If *buffer* is an object exposing the buffer interface, then all keywords are interpreted.

No `__init__` method is needed because the array is fully initialized after the `__new__` method.

Examples

These examples illustrate the low-level *ndarray* constructor. Refer to the *See Also* section above for easier ways of constructing an ndarray.

First mode, *buffer* is None:

```
>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])
```

Second mode:

```
>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
```

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 Representation*

Munsell Renotation System

`colour`

`munsell_colour_to_xyY`(munsell_colour)	Converts given *Munsell* colour to *CIE xyY* colourspace.
`xyY_to_munsell_colour`(xyY[, hue_decimals, ...])	Converts from *CIE xyY* colourspace to *Munsell* 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

[], []

Examples

```
>>> munsell_colour_to_xyY('4.2YR 8.1/5.3')
array([ 0.3873694...,  0.3575165...,  0.59362   ])
>>> munsell_colour_to_xyY('N8.9')
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

[], []

Examples

```
>>> xyY = np.array([0.38736945, 0.35751656, 0.59362000])
>>> # Doctests skip for Python 2.x compatibility.
>>> xyY_to_munsell_colour(xyY)
'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': ..., 'all': ..., '1929': ..., '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'*)

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', '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

[], []

Examples

```
>>> munsell_value(12.23634268)
4.0824437...
>>> munsell_value(12.23634268, method='Priest 1920')
3.4980484...
>>> munsell_value(12.23634268, method='Munsell 1933')
4.1627702...
>>> munsell_value(12.23634268, method='Moon 1943')
4.0688120...
>>> munsell_value(12.23634268, method='Saunderson 1944')
...
4.0444736...
>>> munsell_value(12.23634268, method='Ladd 1955')
4.0511633...
>>> munsell_value(12.23634268, method='McCamy 1987')
array(4.0814348...)
```

colour.MUNSELL_VALUE_METHODS

colour.**MUNSELL_VALUE_METHODS = CaseInsensitiveMapping({'Priest 1920': ..., 'Munsell 1933': ..., 'Moon 1943': ..., 'Saunderson 1944': ..., 'Ladd 1955': ..., 'McCamy 1987': ..., 'ASTM D1535': ..., 'astm2008': ...})**

Supported *Munsell* value computation methods.

References

[], []

MUNSELL_VALUE_METHODS [CaseInsensitiveMapping] **{'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987', 'ASTM D1535'}**

Aliases:

- 'astm2008': 'ASTM D1535'

Priest, Gibson and MacNicholas (1920)

colour.notation

munsell_value_Priest1920(Y)	Returns the *Munsell* value V of given *luminance* Y using *Priest et al. (1920)* 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

[]

Examples

```
>>> munsell_value_Priest1920(12.23634268)
3.4980484...
```

Munsell, Sloan and Godlove (1933)

colour.notation

munsell_value_Munsell1933(Y)	Returns the *Munsell* value V of given *luminance* Y using *Munsell et al. (1933)* 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
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[]

Examples

```
>>> munsell_value_Munsell1933(12.23634268)
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

[]

Examples

```
>>> munsell_value_Moon1943(12.23634268)
4.0688120...
```

Saunderson and Milner (1944)

colour.notation

munsell_value_Saunderson1944(Y)	Returns the *Munsell* value V of given *luminance* Y using *Saunderson and Milner (1944)* 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

[]

Examples

```
>>> munsell_value_Saunderson1944(12.23634268)
4.0444736...
```

Ladd and Pinney (1955)

colour.notation

munsell_value_Ladd1955(Y)	Returns the *Munsell* value V of given *luminance* Y using *Ladd and Pinney (1955)* 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

[]

Examples

```
>>> munsell_value_Ladd1955(12.23634268)
4.0511633...
```

McCamy (1987)

colour.notation

munsell_value_McCamy1987(Y)	Returns the *Munsell* value V of given *luminance* Y using *McCamy (1987)* 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

[]

Examples

```
>>> munsell_value_McCamy1987(12.23634268)
array(4.0814348...)
```

ASTM D1535-08e1

colour.notation

munsell_value_ASTMD1535(Y)	Returns the *Munsell* value V of given *luminance* Y using an inverse lookup table from *ASTM D1535-08e1* method.

colour.notation.munsell_value_ASTMD1535

colour.notation.**munsell_value_ASTMD1535**(*Y*)

Returns the *Munsell* value V of given *luminance* Y using an inverse 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

[]

Examples

```
>>> munsell_value_ASTMD1535(12.23634268)
4.0824437...
```

Hexadecimal Representation

colour.notation

RGB_to_HEX(RGB)	Converts from *RGB* colourspace to hexadecimal representation.
HEX_to_RGB(HEX)	Converts from hexadecimal representation to *RGB* colourspace.

colour.notation.RGB_to_HEX

colour.notation.**RGB_to_HEX**(*RGB*)

Converts from *RGB* colourspace to hexadecimal representation.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns Hexadecimal representation.

Return type unicode

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.66666667, 0.86666667, 1.00000000])
>>> # Doctests skip for Python 2.x compatibility.
>>> RGB_to_HEX(RGB)
'#aaddff'
```

colour.notation.HEX_to_RGB

colour.notation.**HEX_to_RGB**(*HEX*)

Converts from hexadecimal representation to *RGB* colourspace.

Parameters **HEX** (unicode or array_like) – Hexadecimal representation.

Returns *RGB* colourspace array.

Return type ndarray

Notes

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

Examples

```
>>> HEX = '#aaddff'
>>> HEX_to_RGB(HEX)
array([ 0.6666666...,  0.8666666...,  1.        ])
```

Optical Phenomena

- *Rayleigh Scattering*

Rayleigh Scattering

colour

rayleigh_scattering(wavelength[, ...])	Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
sd_rayleigh_scattering([shape, ...])	Returns the *Rayleigh* spectral distribution for given spectral shape.
scattering_cross_section(wavelength[, ...])	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.

colour.rayleigh_scattering

colour.**rayleigh_scattering**(*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 $F(air)$ 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

[], []

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
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*, *altitude=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

[], []

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_rayleigh_scattering()
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 ...],
```

(continues on next page)

(continued from previous page)

```
                       [ 412.        ,    0.3413716...],
                       [ 413.        ,    0.3379587...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 468.        ,    0.2017487...],
                       [ 469.        ,    0.1999865...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 524.        ,    0.1269560...],
                       [ 525.        ,    0.1259707...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 580.        ,    0.0839104...],
                       [ 581.        ,    0.0833242...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 636.        ,    0.0576953...],
                       [ 637.        ,    0.0573286...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 692.        ,    0.0409820...],
                       [ 693.        ,    0.0407431...],
                       [ 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...],
```

(continues on next page)

(continued from previous page)

```
                       [ 748.        ,    0.0299149...],
                       [ 749.        ,    0.0297538...],
                       [ 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_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
```

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 $F(air)$ 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

[], []

Examples

```
>>> scattering_cross_section(555 * 10e-8)
4.6613309...e-27
```

colour.phenomena

rayleigh_optical_depth(wavelength[, ...])	Returns the *Rayleigh* 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 $F(air)$ 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

[], []

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
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*
- *ANSI/IES TM-30-18 Colour Rendition Report*
- *Colour Models Volume*
- *Automatic Colour Conversion Graph*

Common

colour.plotting

colour_style([use_style])	Returns *Colour* plotting style.
colour_cycle(**kwargs)	Returns a colour cycle iterator using given colour map.
artist(**kwargs)	Returns the current figure and its axes or creates a new one.
camera(**kwargs)	Sets the camera settings.
render(**kwargs)	Renders the current figure while adjusting various settings such as the bounding box, the title or background transparency.
label_rectangles(labels, rectangles[, ...])	Add labels above given rectangles.
uniform_axes3d(**kwargs)	Sets equal aspect ratio to given 3d axes.
plot_single_colour_swatch(colour_swatch, ...)	Plots given colour swatch.
plot_multi_colour_swatches(colour_swatches)	Plots given colours swatches.
plot_single_function(function[, samples, ...])	Plots given function.
plot_multi_functions(functions[, samples, ...])	Plots given functions.
plot_image(image[, imshow_kwargs, text_kwargs])	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.

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.

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 Current figure and axes.

Return type tuple

colour.plotting.camera

colour.plotting.**camera**(***kwargs*)

Sets the camera settings.

Parameters

- **figure** (Figure, optional) – Figure to apply the render elements onto.
- **axes** (Axes, optional) – Axes to apply the render elements onto.
- **azimuth** (numeric, optional) – Camera azimuth.
- **camera_aspect** (unicode, optional) – Matplotlib axes aspect. Default is *equal*.
- **elevation** (numeric, optional) – Camera elevation.

Returns Current figure and axes.

Return type tuple

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.

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 *True*.
- **title** (unicode, optional) – Figure title.
- **wrap_title** (unicode, optional) – Whether to wrap the figure title. Default is *True*.
- **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.
- **figure** (Figure, optional) – Figure to apply the render elements onto.
- **axes** (Axes, optional) – Axes to apply the render elements onto.

Returns Current figure and axes.

Return type tuple

colour.plotting.uniform_axes3d

colour.plotting.**uniform_axes3d**(***kwargs*)

Sets equal aspect ratio to given 3d axes.

Parameters

- **figure** (Figure, optional) – Figure to apply the render elements onto.
- **axes** (Axes, optional) – Axes to apply the render elements onto.

Returns Current figure and axes.

Return type tuple

colour.plotting.plot_single_colour_swatch

colour.plotting.**plot_single_colour_swatch**(*colour_swatch*, ***kwargs*)

Plots given colour swatch.

Parameters

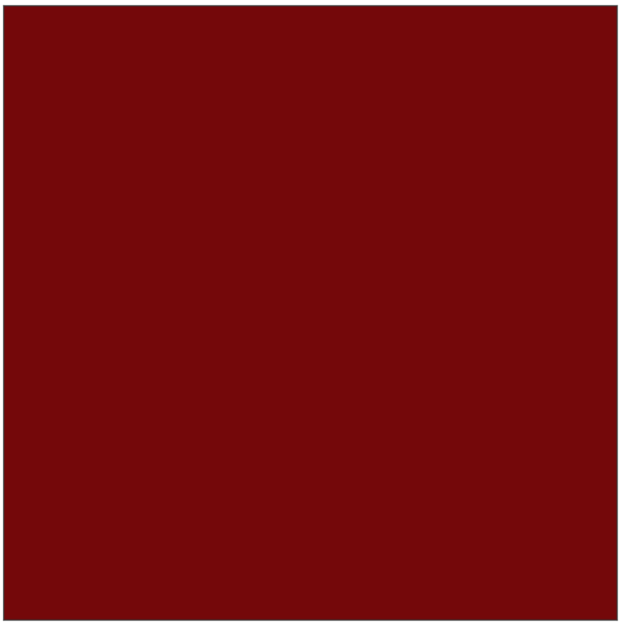
- **colour_swatch** (array_like or ColourSwatch) – Colour swatch, either a regular *array_like* or a colour.plotting.ColourSwatch class instance.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, Please refer to the documentation 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_kwargs** (dict, optional) – {colour.plotting.plot_multi_colour_swatches()}, Keyword arguments for the plt.text() definition. The following special keywords can also be used:
  - *offset*: Sets 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)
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_colour_swatches

colour.plotting.**plot_multi_colour_swatches**(*colour_swatches*, *width=1*, *height=1*, *spacing=0*, *columns=None*, *direction='+y'*, *text_kwargs=None*, *background_colour=(1.0, 1.0, 1.0)*, *compare_swatches=None*, ***kwargs*)

Plots given colours swatches.

Parameters

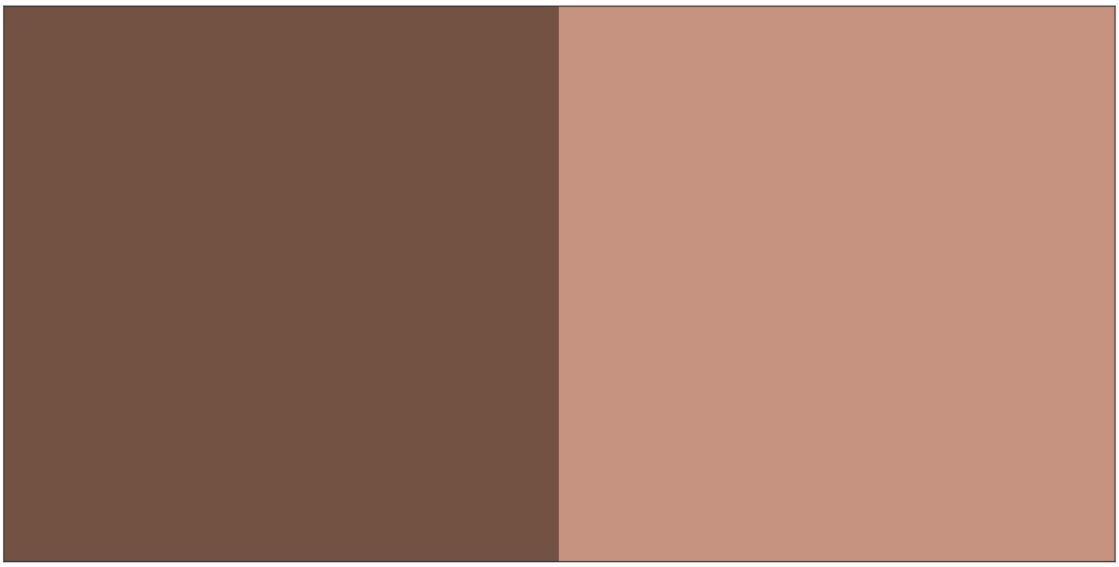
- **colour_swatches** (array_like) – Colour swatch sequence, either a regular *array_like* or a sequence of colour.plotting.ColourSwatch class instances.
- **width** (numeric, optional) – Colour swatch width.
- **height** (numeric, optional) – Colour swatch height.
- **spacing** (numeric, optional) – Colour swatches spacing.
- **columns** (int, optional) – Colour swatches columns count, defaults to the colour swatch count or half of it if comparing.
- **direction** (unicode, optional) – {'+y', '-y'} Row stacking direction.
- **text_kwargs** (dict, optional) – Keyword arguments for the plt.text() definition. The following special keywords can also be used:
  - *offset*: Sets the text offset.
  - *visible*: Makes the text visible.
- **background_colour** (array_like or unicode, optional) – Background colour.
- **compare_swatches** (unicode, optional) – **{None, 'Diagonal', 'Stacked'}**, 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.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

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])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_function

colour.plotting.**plot_single_function**(*function*, *samples=None*, *log_x=None*, *log_y=None*, *plot_kwargs=None*, ***kwargs*)

Plots given function.

Parameters

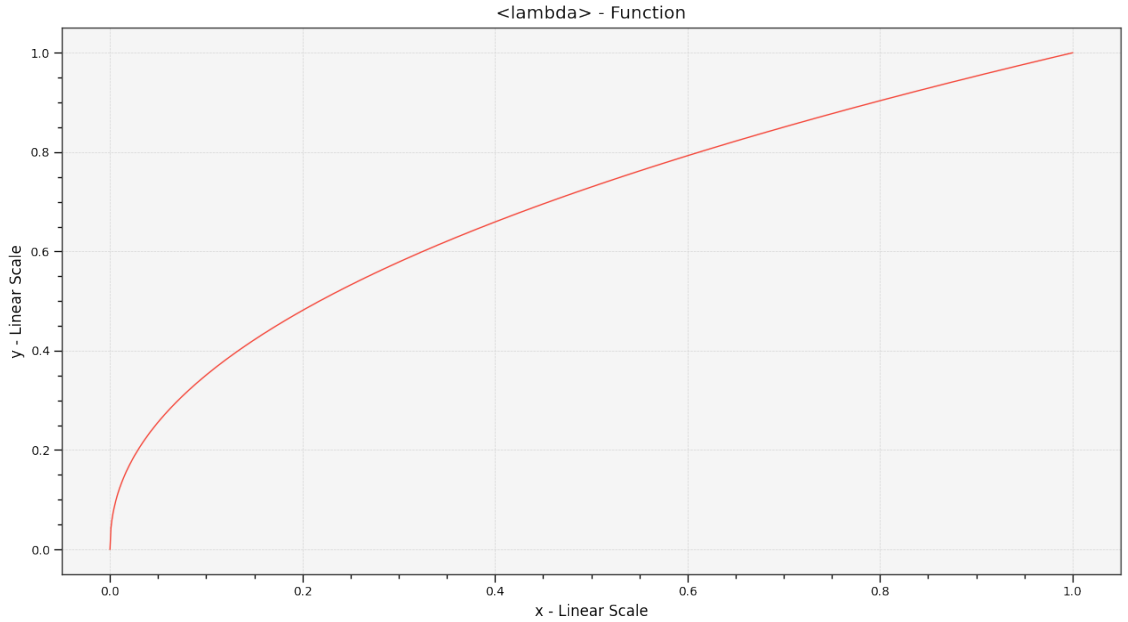
- **function** (callable) – Function to plot.
- **samples** (array_like, optional,) – Samples to evaluate the functions with.
- **log_x** (int, optional) – Log base to use for the *x* axis scale, if *None*, the *x* axis scale will be linear.
- **log_y** (int, optional) – Log base to use for the *y* axis scale, if *None*, the *y* axis scale will be linear.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted function.
- ****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

```
>>> from colour.models import gamma_function
>>> plot_single_function(partial(gamma_function, exponent=1 / 2.2))
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_functions

colour.plotting.**plot_multi_functions**(*functions*, *samples=None*, *log_x=None*, *log_y=None*, *plot_kwargs=None*, ***kwargs*)

Plots given functions.

Parameters

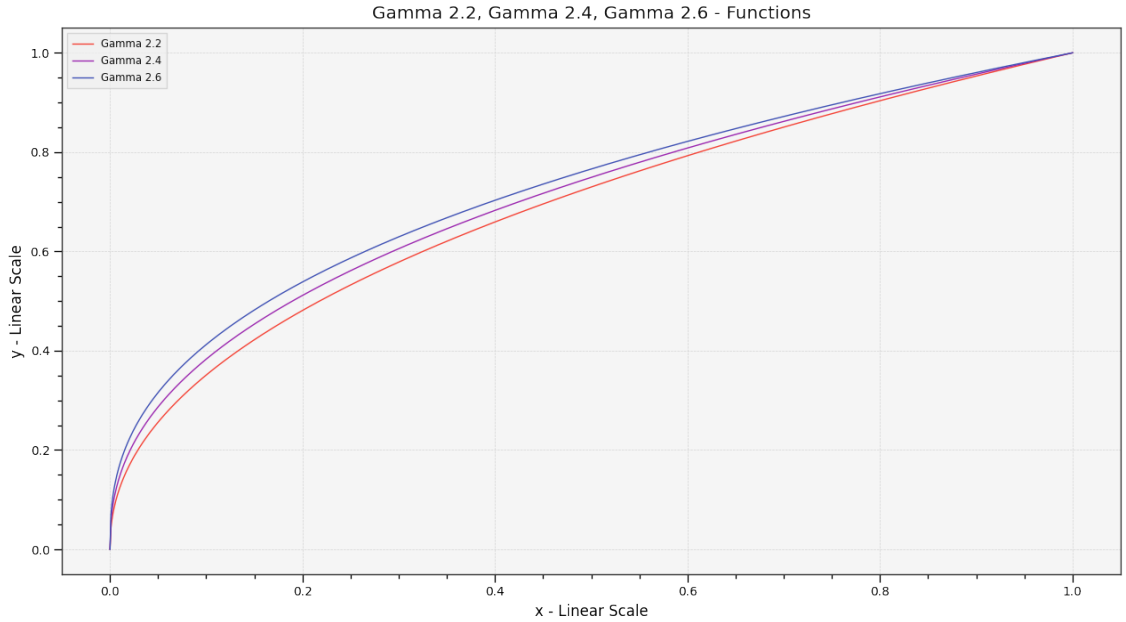
- **functions** (dict) – Functions to plot.
- **samples** (array_like, optional,) – Samples to evaluate the functions with.
- **log_x** (int, optional) – Log base to use for the *x* axis scale, if *None*, the *x* axis scale will be linear.
- **log_y** (int, optional) – Log base to use for the *y* axis scale, if *None*, the *y* axis scale will be linear.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the `plt.plot()` definition, used to control the style of the plotted functions. `plot_kwargs` can be either a single dictionary applied to all the plotted functions with same settings or a sequence of dictionaries with different settings for each plotted function.
- ****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),
... }
>>> plot_multi_functions(functions)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_image

colour.plotting.**plot_image**(*image*, *imshow_kwargs=None*, *text_kwargs=None*, ***kwargs*)

Plots given image.

Parameters

- **image** (array_like) – Image to plot.
- **imshow_kwargs** (dict, optional) – Keyword arguments for the plt.imshow() definition.
- **text_kwargs** (dict, optional) – Keyword arguments for the plt.text() definition. The following special keyword arguments can also be used:
  - *offset* : array_like, sets the text offset.
- ****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))
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

Colorimetry

colour.plotting

plot_single_sd(sd[, cmfs, ...])	Plots given spectral distribution.
plot_multi_sds(sds[, plot_kwargs])	Plots given spectral distributions.
plot_single_cmfs([cmfs])	Plots given colour matching functions.
plot_multi_cmfs(cmfs, **kwargs)	Plots given colour matching functions.
plot_single_illuminant_sd(illuminant[, cmfs])	Plots given single illuminant spectral distribution.
plot_multi_illuminant_sds(illuminants, **kwargs)	Plots given illuminants spectral distributions.
plot_visible_spectrum([cmfs, ...])	Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.
plot_single_lightness_function(function, ...)	Plots given *Lightness* function.
plot_multi_lightness_functions(functions, ...)	Plots given *Lightness* functions.
plot_single_luminance_function(function, ...)	Plots given *Luminance* function.
plot_multi_luminance_functions(functions, ...)	Plots given *Luminance* functions.
plot_blackbody_spectral_radiance([...])	Plots given blackbody spectral radiance.
plot_blackbody_colours([shape, cmfs])	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.
- **cmfs** (unicode or LMS_ConeFundamentals or RGB_ColourMatchingFunctions or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **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.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

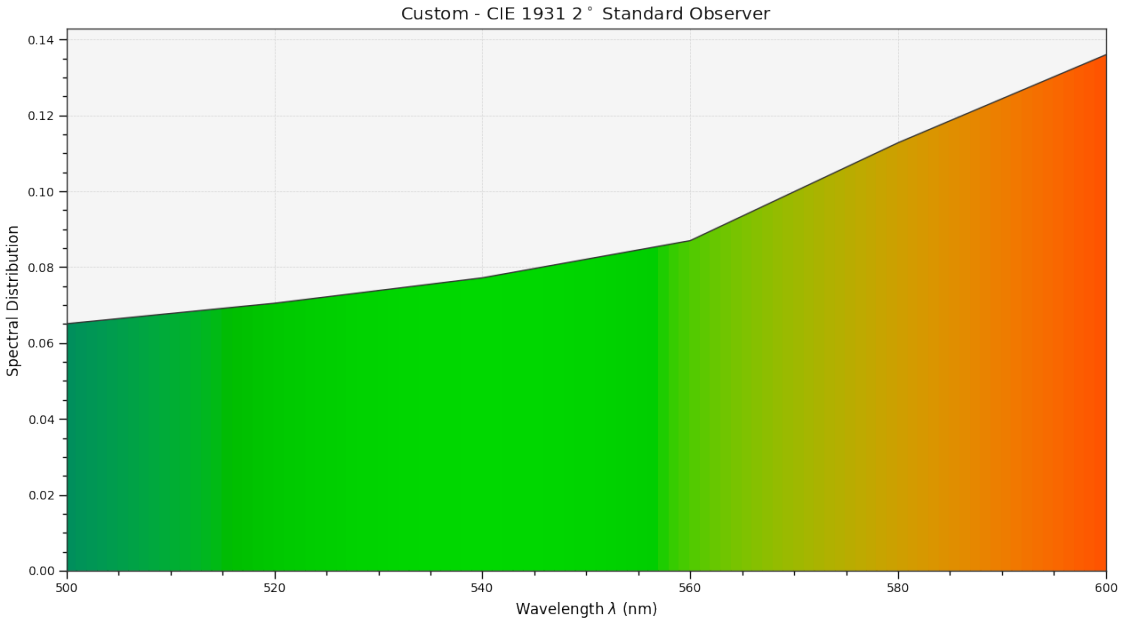
Return type tuple

References

[]

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)
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_sds

`colour.plotting.`**`plot_multi_sds`**(*sds*, *plot_kwargs=None*, ***kwargs*)

Plots given spectral distributions.

Parameters

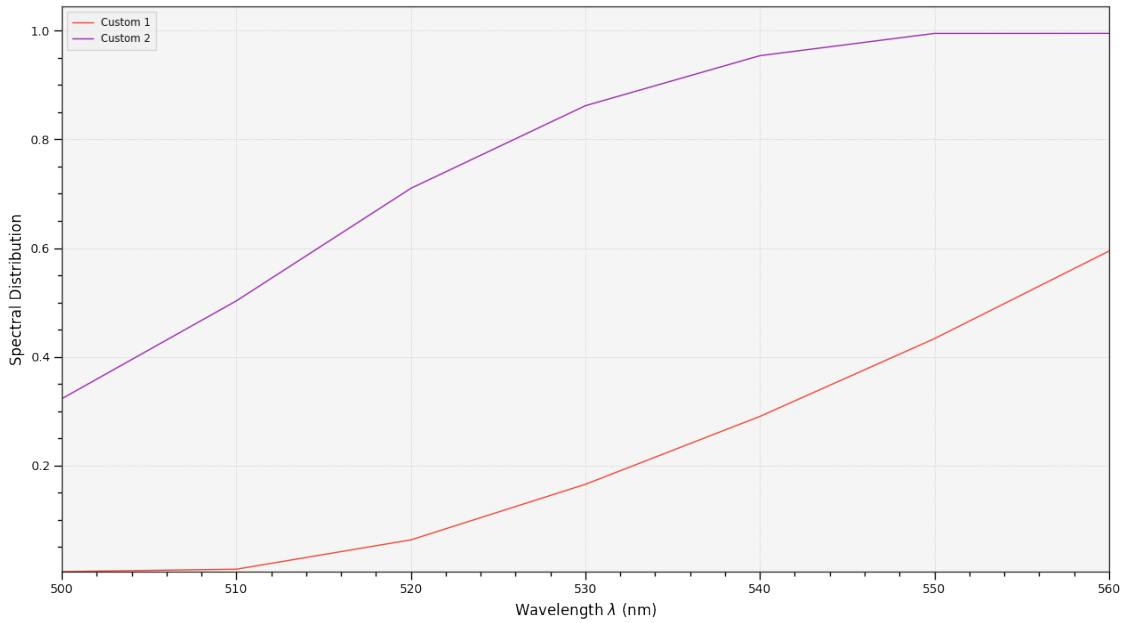
- **sds** (`array_like` or `MultiSpectralDistributions`) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single `colour.MultiSpectralDistributions` class instance, a list of `colour.MultiSpectralDistributions` class instances or a list of `colour.SpectralDistribution` class instances.
- **plot_kwargs** (`dict` or `array_like`, `optional`) – Keyword arguments for the `plt.plot()` definition, used to control the style of the plotted spectral distributions. `plot_kwargs` can be either a single dictionary applied to all the plotted spectral distributions with same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - *illuminant* : unicode or `colour.SpectralDistribution`, the illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. `illuminant` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.
  - *cmfs* : unicode, the standard observer colour matching functions used for computing the spectral distributions colours. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.
  - *normalise_sd_colours* : bool, whether to normalise the computed spectral distributions colours. The default is *True*.
  - *use_sd_colours* : bool, whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the `plt.plot()` definition `color` argument with pre-computed values. The default is *True*.
- ****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
... }
>>> sd_1 = SpectralDistribution(data_1, name='Custom 1')
>>> sd_2 = SpectralDistribution(data_2, name='Custom 2')
>>> plot_kwargs = [
...     {'use_sd_colours': True},
...     {'use_sd_colours': True, 'linestyle': 'dashed'},
... ]
>>> plot_multi_sds([sd_1, sd_2], plot_kwargs=plot_kwargs)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_cmfs

colour.plotting.**plot_single_cmfs**(*cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots given colour matching functions.

Parameters

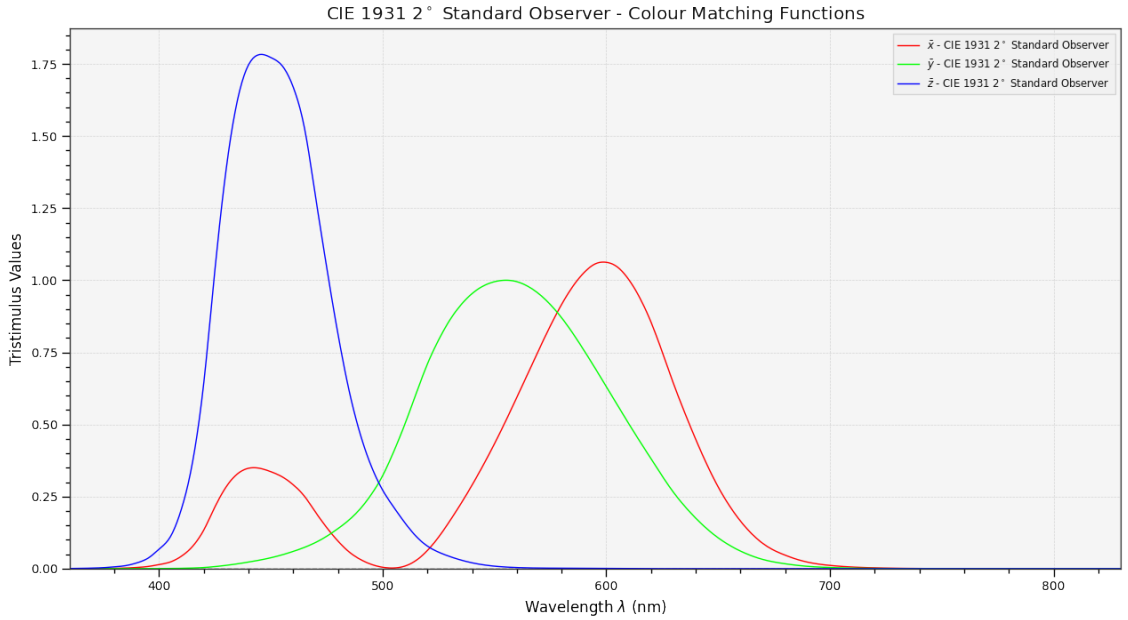
- **cmfs** (unicode or LMS_ConeFundamentals or RGB_ColourMatchingFunctions or XYZ_ColourMatchingFunctions, optional) – Colour matching functions to plot. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), 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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_cmfs

colour.plotting.**plot_multi_cmfs**(*cmfs*, ***kwargs*)

Plots given colour matching functions.

Parameters

- **cmfs** (unicode or LMS_ConeFundamentals or RGB_ColourMatchingFunctions or XYZ_ColourMatchingFunctions or array_like) – Colour matching functions to plot. cmfs elements can be of any type or form supported by the colour.plotting.filter_cmfs() definition.

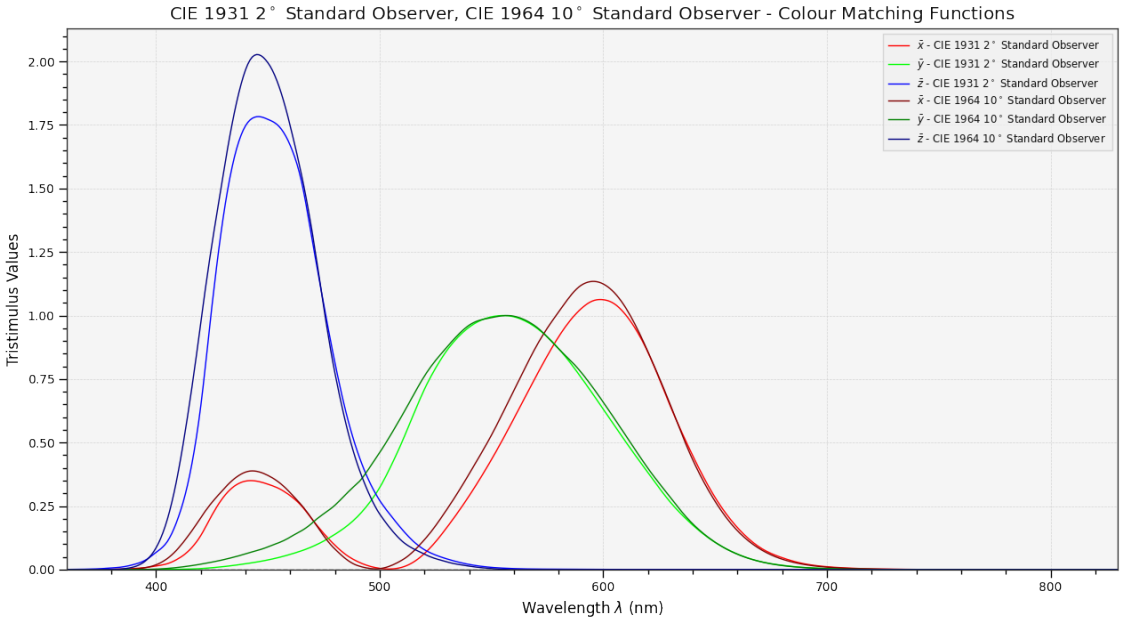
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

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)
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_illuminant_sd

colour.plotting.**plot_single_illuminant_sd**(*illuminant*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots given single illuminant spectral distribution.

Parameters

- **illuminant** (unicode or LMS_ConeFundamentals or RGB_ColourMatchingFunctions or XYZ_ColourMatchingFunctions, optional) – Illuminant to plot. illuminant can be of any type or form supported by the colour.plotting.filter_illuminants() definition.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- ****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.

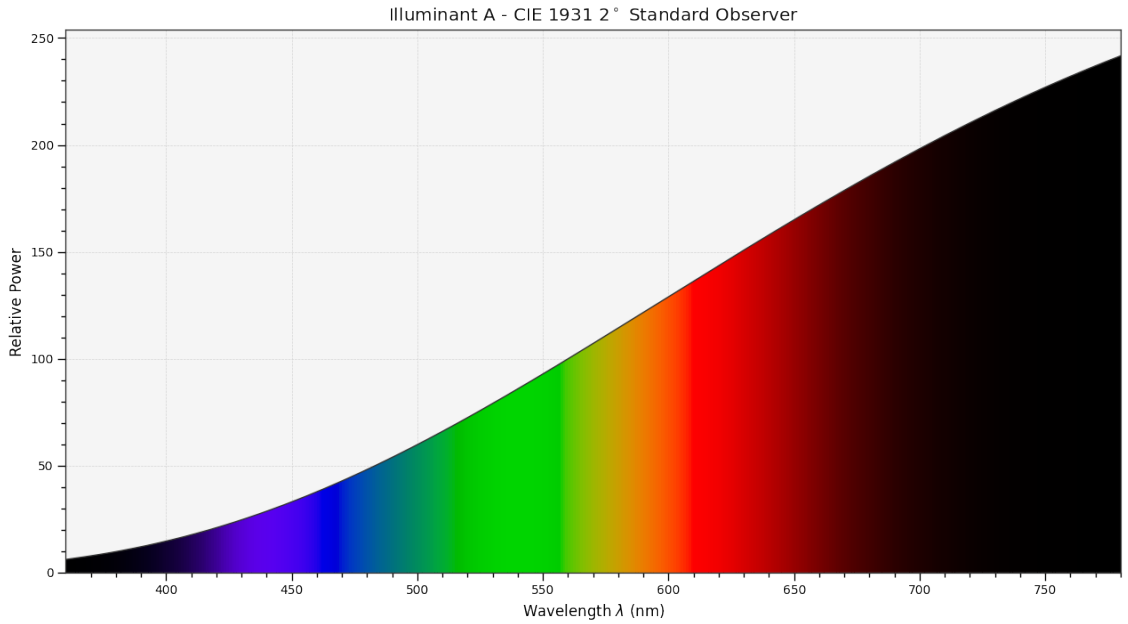
Return type tuple

References

[]

Examples

```
>>> plot_single_illuminant_sd('A')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_illuminant_sds

colour.plotting.**plot_multi_illuminant_sds**(*illuminants*, ***kwargs*)

Plots given illuminants spectral distributions.

Parameters

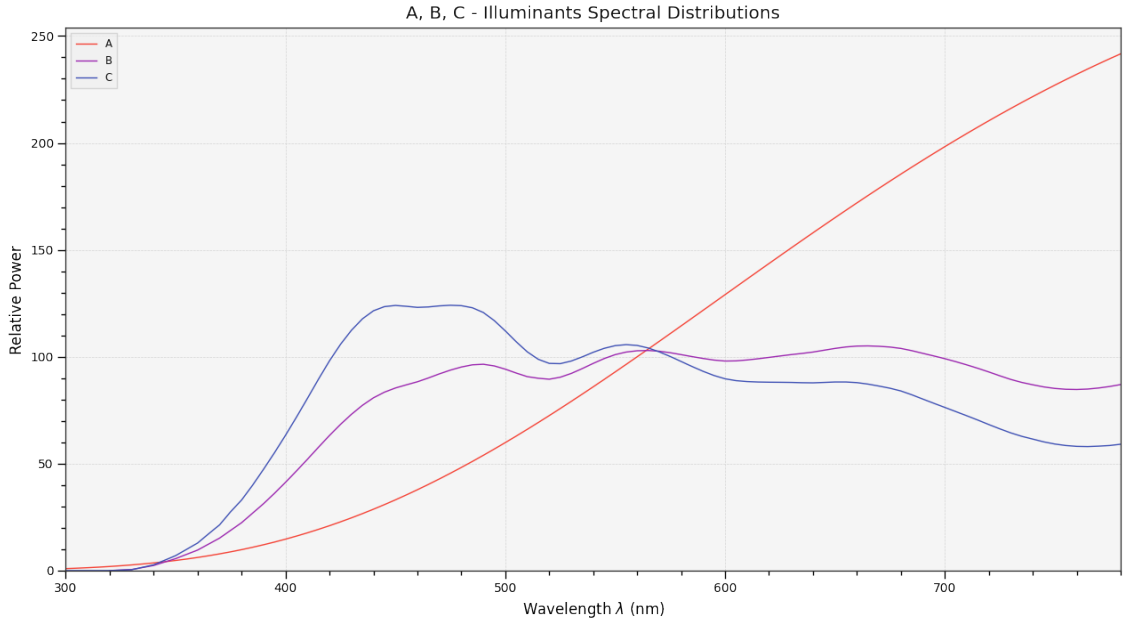
- **illuminants** (unicode or SpectralDistribution or array_like) – Illuminants to plot. illuminants elements can be of any type or form supported by the colour.plotting.filter_illuminants() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_sds(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_illuminant_sds(['A', 'B', 'C'])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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 or LMS_ConeFundamentals or RGB_ColourMatchingFunctions or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **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.
- ****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.

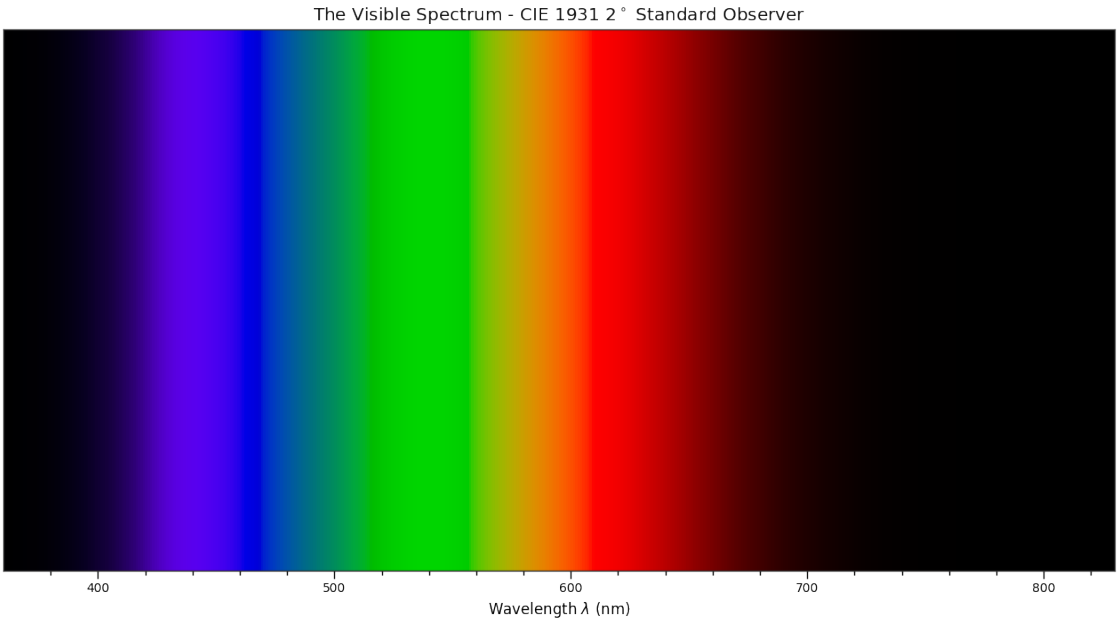
Return type tuple

References

[]

Examples

```
>>> plot_visible_spectrum()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_lightness_function

colour.plotting.**plot_single_lightness_function**(*function*, ***kwargs*)

Plots given *Lightness* function.

Parameters

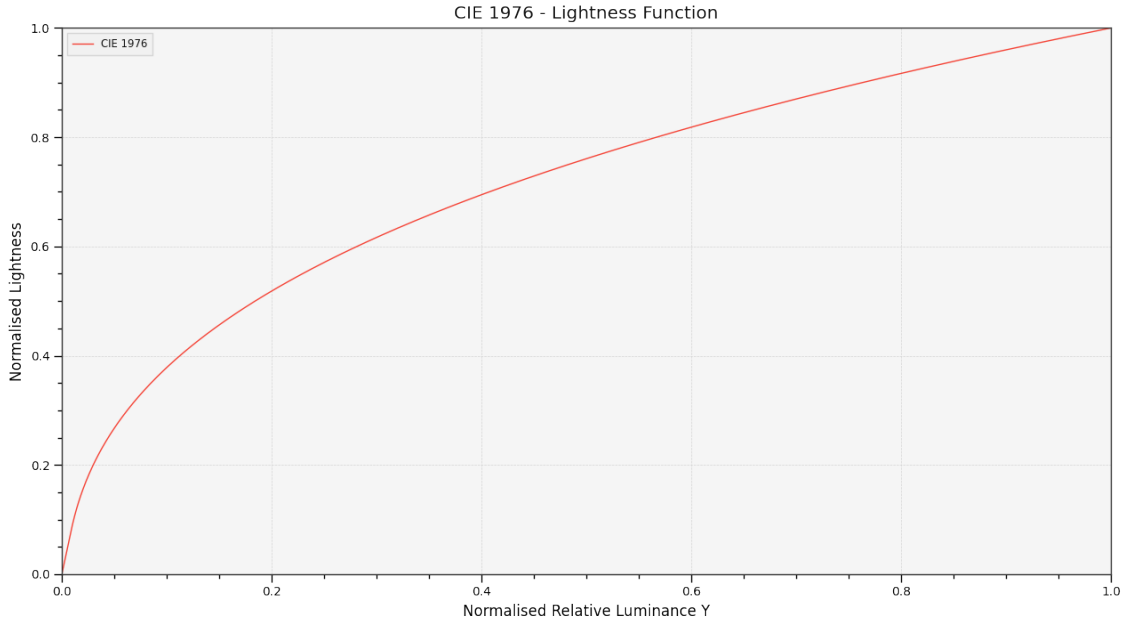
- **function** (unicode or object) – *Lightness* function to plot. function can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_lightness_function('CIE 1976')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_lightness_functions

colour.plotting.**plot_multi_lightness_functions**(*functions*, ***kwargs*)

Plots given *Lightness* functions.

Parameters

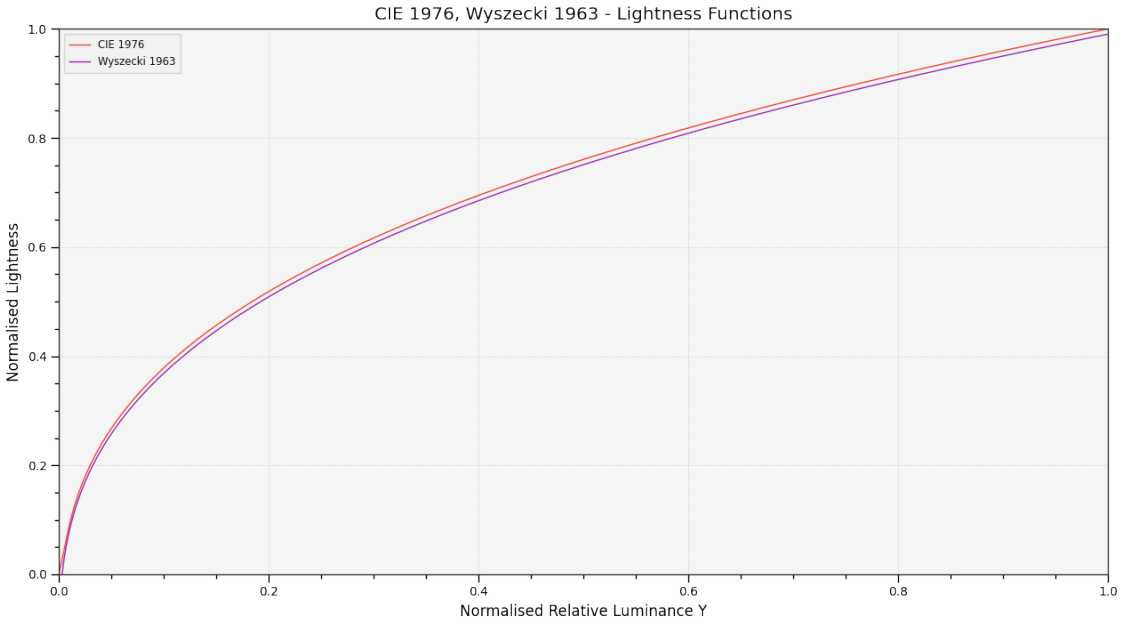
- **functions** (unicode or object or array_like) – *Lightness* functions to plot. functions elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, 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'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_luminance_function

colour.plotting.**plot_single_luminance_function**(*function*, ***kwargs*)

Plots given *Luminance* function.

Parameters

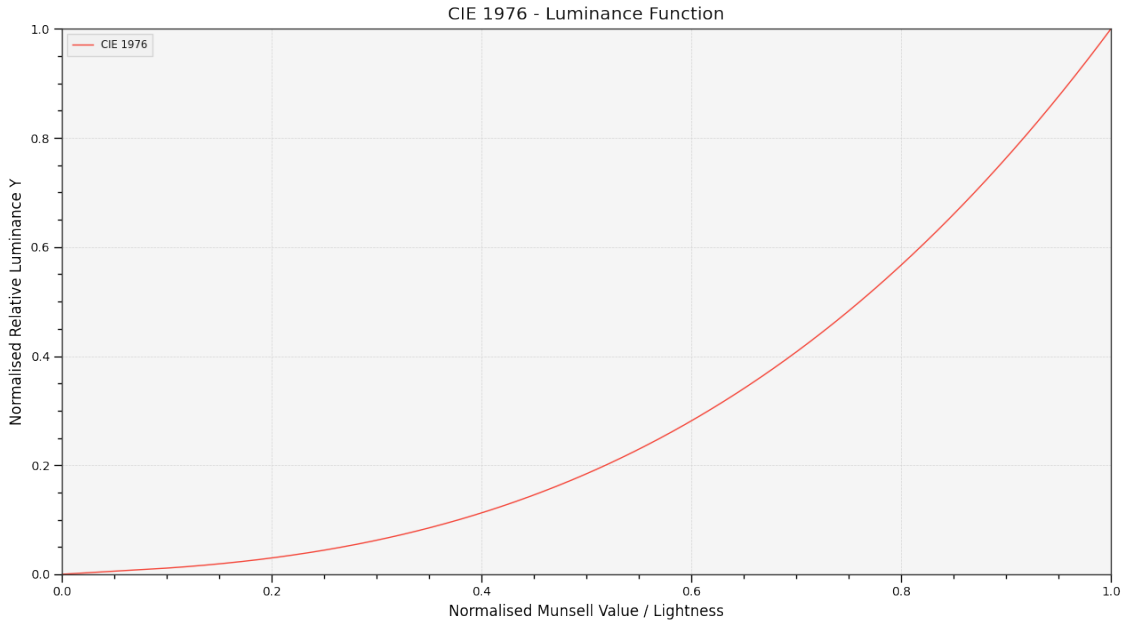
- **function** (unicode or object, optional) – *Luminance* function to plot.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_luminance_function('CIE 1976')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_luminance_functions

colour.plotting.**plot_multi_luminance_functions**(*functions*, ***kwargs*)

Plots given *Luminance* functions.

Parameters

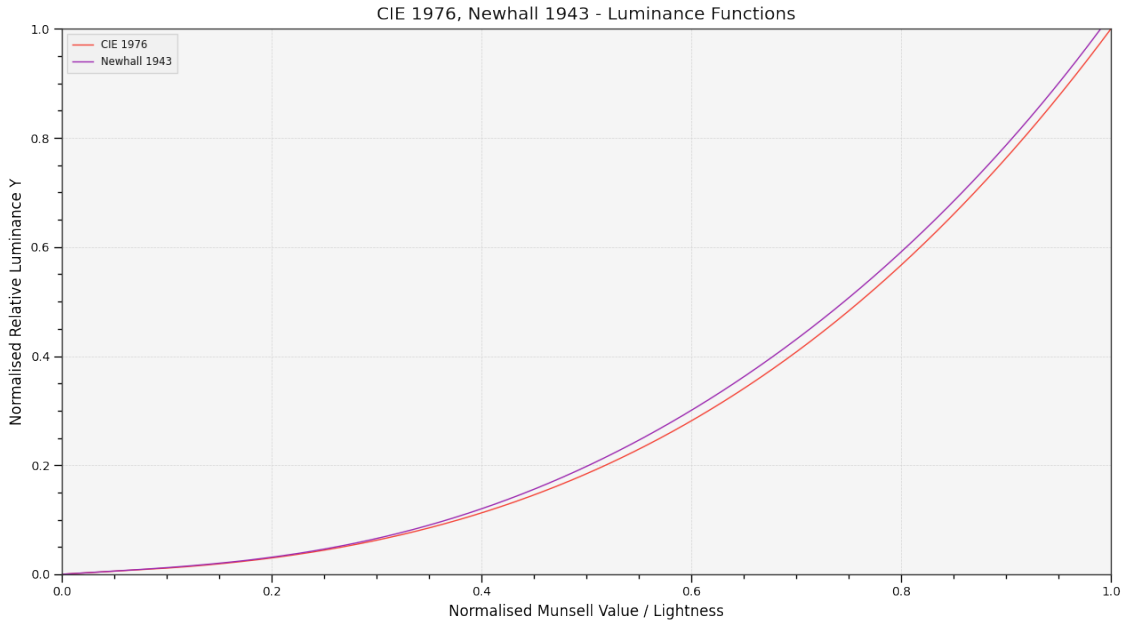
- **functions** (unicode or object or array_like) – *Luminance* functions to plot. functions elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, 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'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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

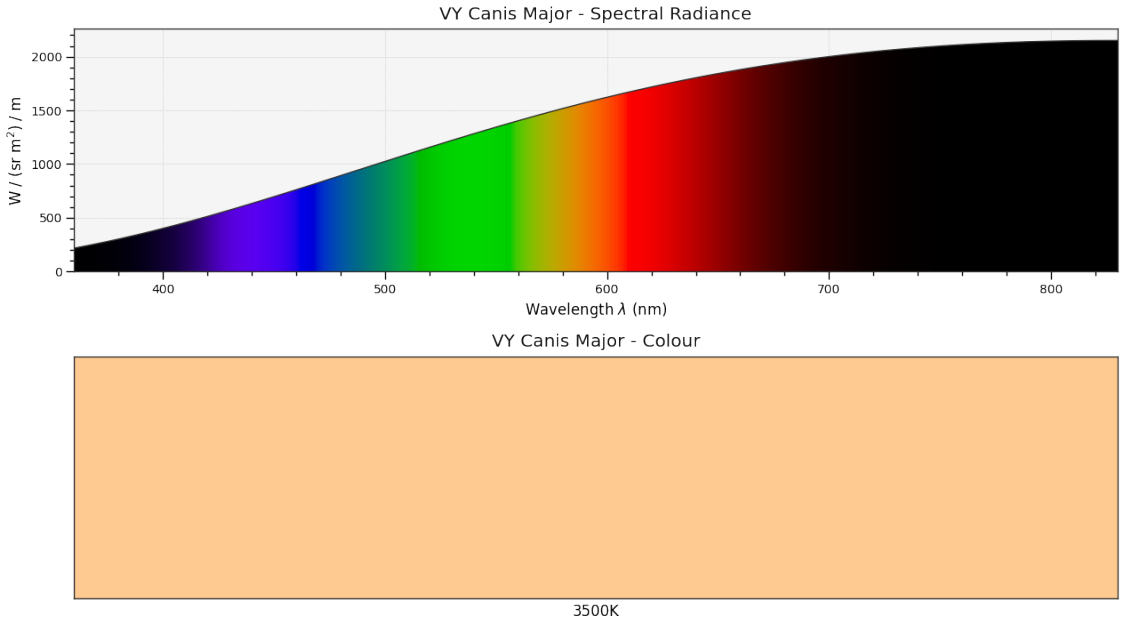
- **temperature** (numeric, optional) – Blackbody temperature.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **blackbody** (unicode, optional) – Blackbody name.
- ****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')
...
(<Figure size ... with 2 Axes>, <...AxesSubplot...>)
```



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

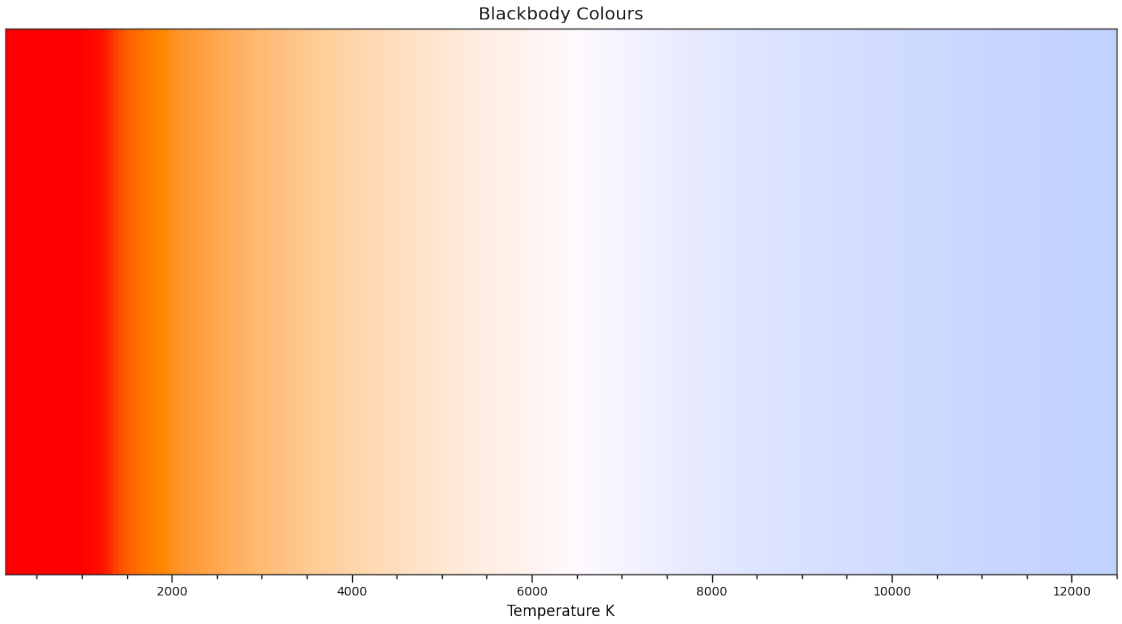
- **shape** (SpectralShape, optional) – Spectral shape to use as plot boundaries.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for computing the blackbody colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- ****kwargs** (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))
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Colour Vision Deficiency

colour.plotting

plot_cvd_simulation_Machado2009(RGB[, ...])	Performs colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al. (2009)* 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.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_image(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Notes

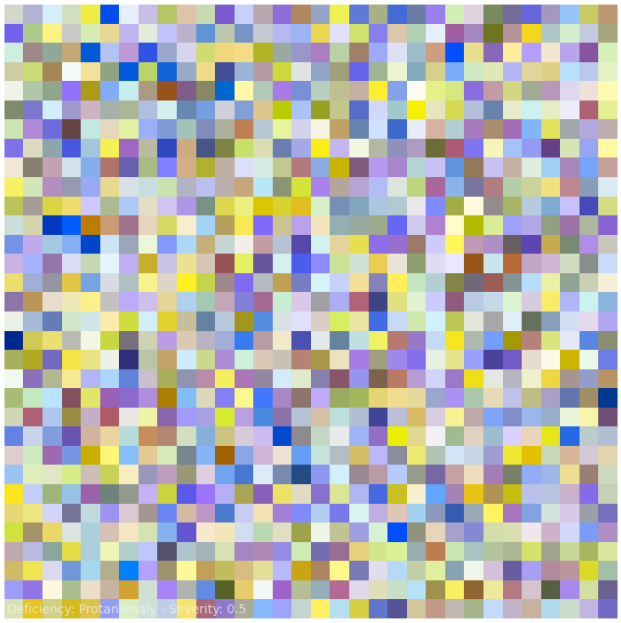
- Input *RGB* array is expected to be linearly encoded.

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)
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Colour Characterisation

colour.plotting

plot_single_colour_checker([colour_checker])	Plots given colour checker.
plot_multi_colour_checkers(colour_checkers, ...)	Plots and compares given colour checkers.

colour.plotting.plot_single_colour_checker

colour.plotting.**plot_single_colour_checker**(*colour_checker='ColorChecker24 - After November 2014'*, ***kwargs*)

Plots given colour checker.

Parameters

- **colour_checker** (unicode or ColourChecker, optional) – Color checker to plot. colour_checker can be of any type or form supported by the colour.plotting.filter_colour_checkers() definition.

- ****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')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

colour.plotting.plot_multi_colour_checkers

colour.plotting.**plot_multi_colour_checkers**(*colour_checkers*, ***kwargs*)

Plots and compares given colour checkers.

Parameters

- **colour_checkers** (unicode or ColourChecker or array_like) – Color checker to plot, count must be less than or equal to 2. colour_checkers elements can be of any type or form supported by the colour.plotting.filter_colour_checkers() definition.
- ****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'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

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

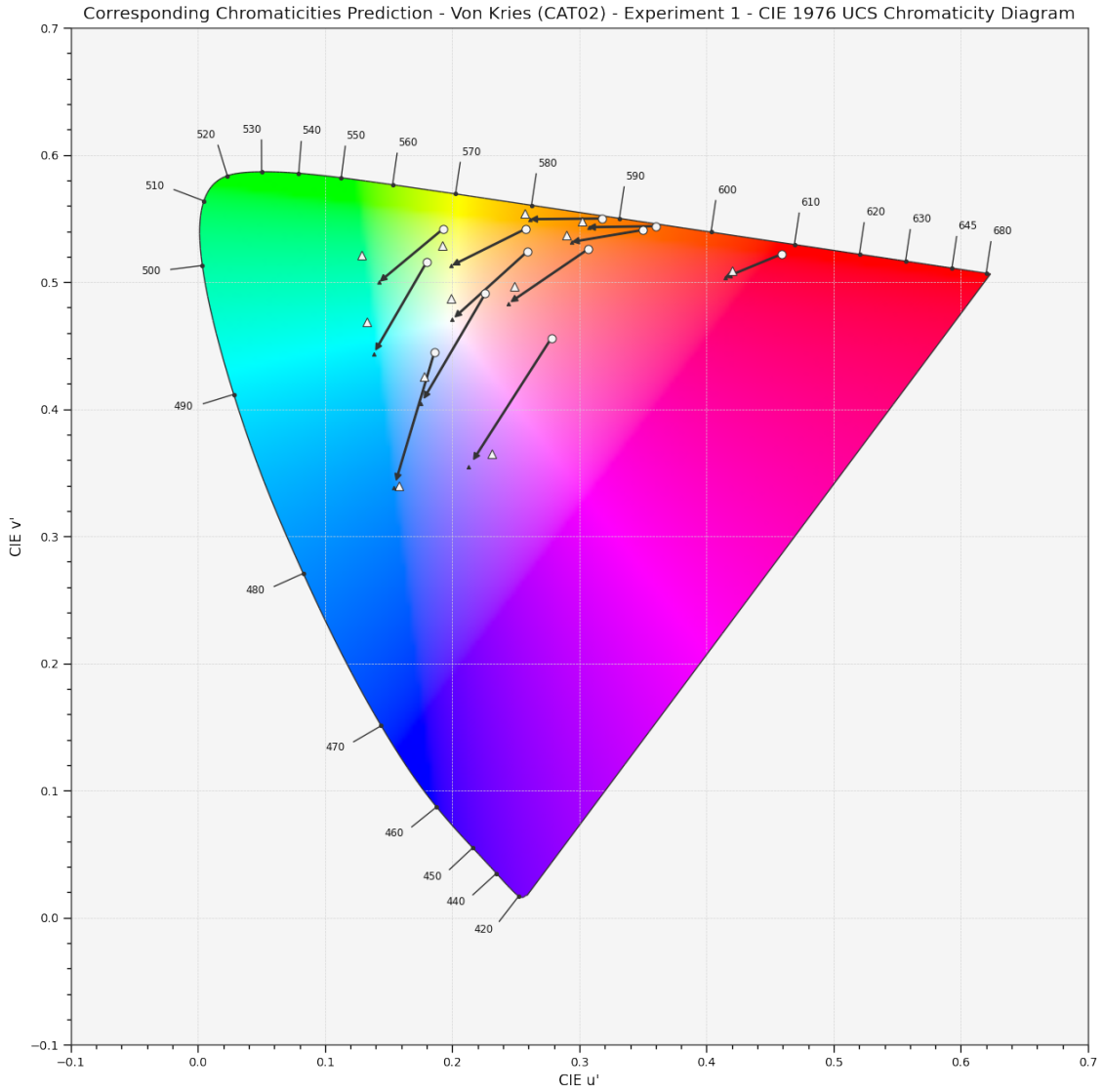
- **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **model** (unicode, optional) – Corresponding chromaticities prediction model name.
- **transform** (unicode, optional) – Transformation to use with *Von Kries* chromatic adaptation model.
- ****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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



CIE Chromaticity Diagrams

colour.plotting

plot_chromaticity_diagram_CIE1931([cmfs, ...])	Plots the *CIE 1931 Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1960UCS([cmfs, ...])	Plots the *CIE 1960 UCS Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1976UCS([cmfs, ...])	Plots the *CIE 1976 UCS Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1931(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1960UCS(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1976UCS(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

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

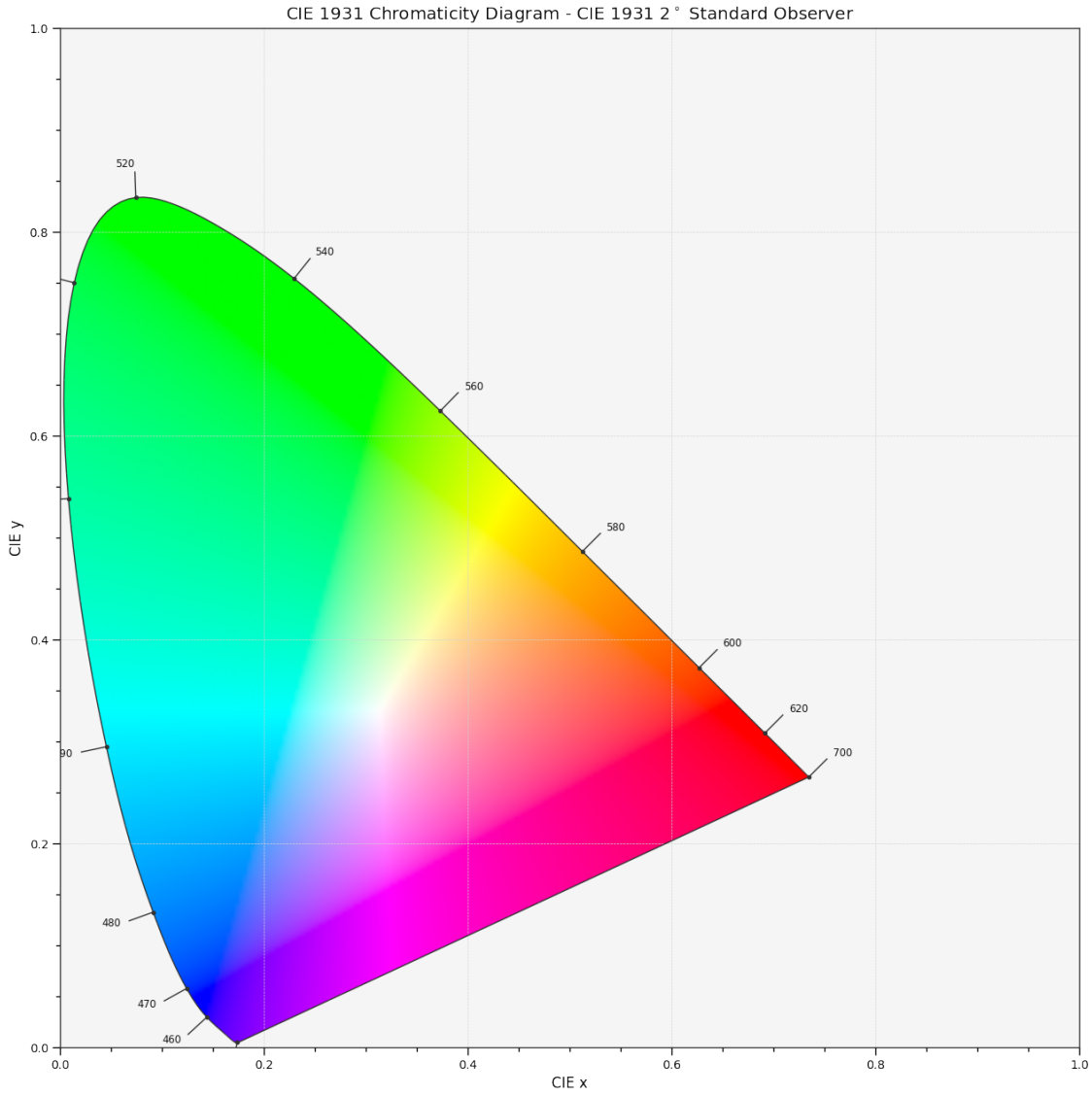
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.
- ****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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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

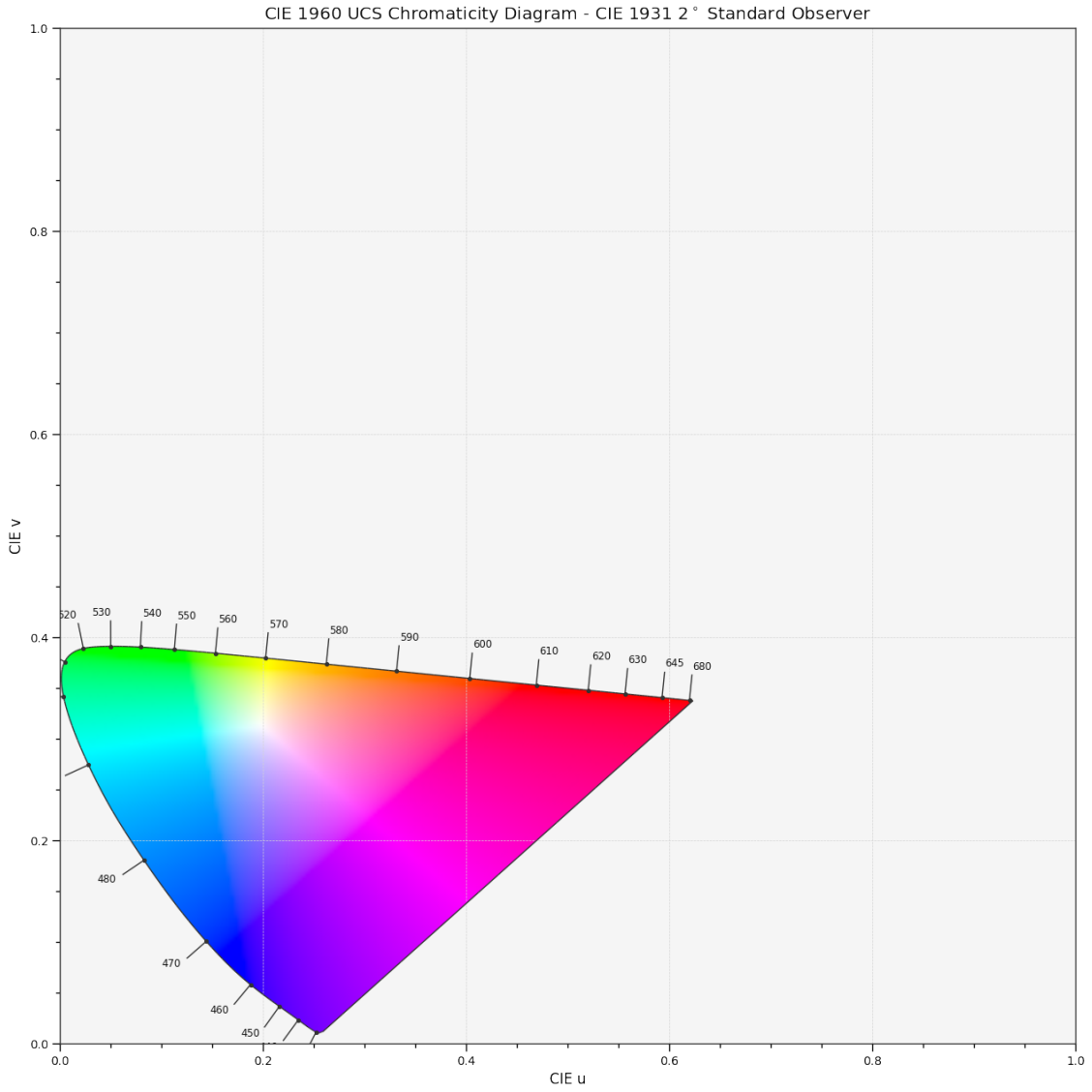
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.
- ****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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_chromaticity_diagram_CIE1976UCS**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, show_spectral_locus=True, **kwargs*)

Plots the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus bound-

aries. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.

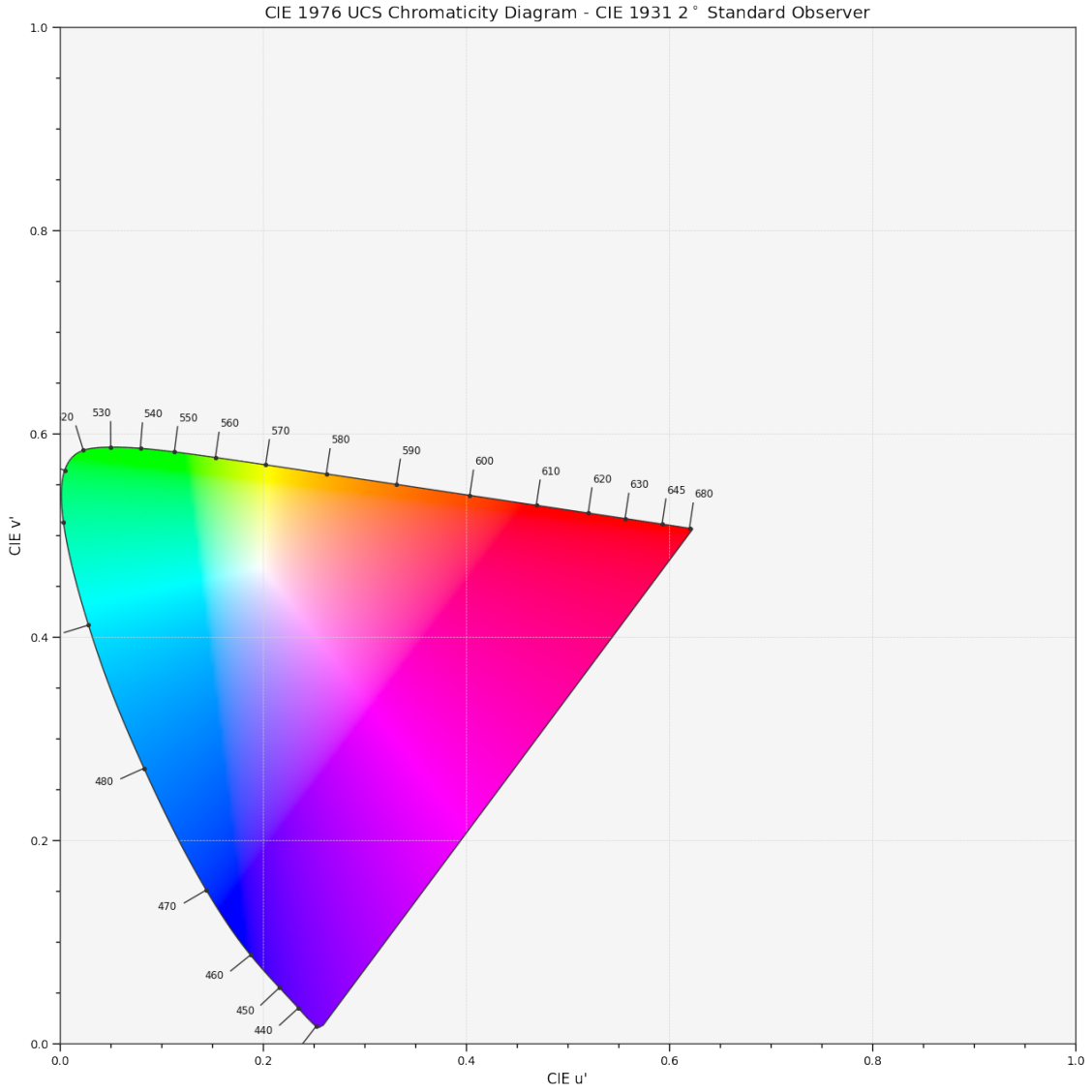
- **show_diagram_colours** (`bool`, `optional`) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (`bool`, `optional`) – Whether to display the *Spectral Locus*.
- ****kwargs** (`dict`, `optional`) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1931**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1931=<function plot_chromaticity_diagram_CIE1931>*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs 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. The following special keyword arguments can also be used:
  - *annotate* : bool, whether to annotate the spectral distributions.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted spectral distributions. plot_kwargs can be either a single dictionary applied to all the plotted spectral distributions with same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - *illuminant* : unicode or colour.SpectralDistribution, the illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *cmfs* : unicode, the standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *normalise_sd_colours* : bool, whether to normalise the computed spectral distributions colours. The default is *True*.
  - *use_sd_colours* : bool, whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the plt.plot() definition color argument with pre-computed values. The default is *True*.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please

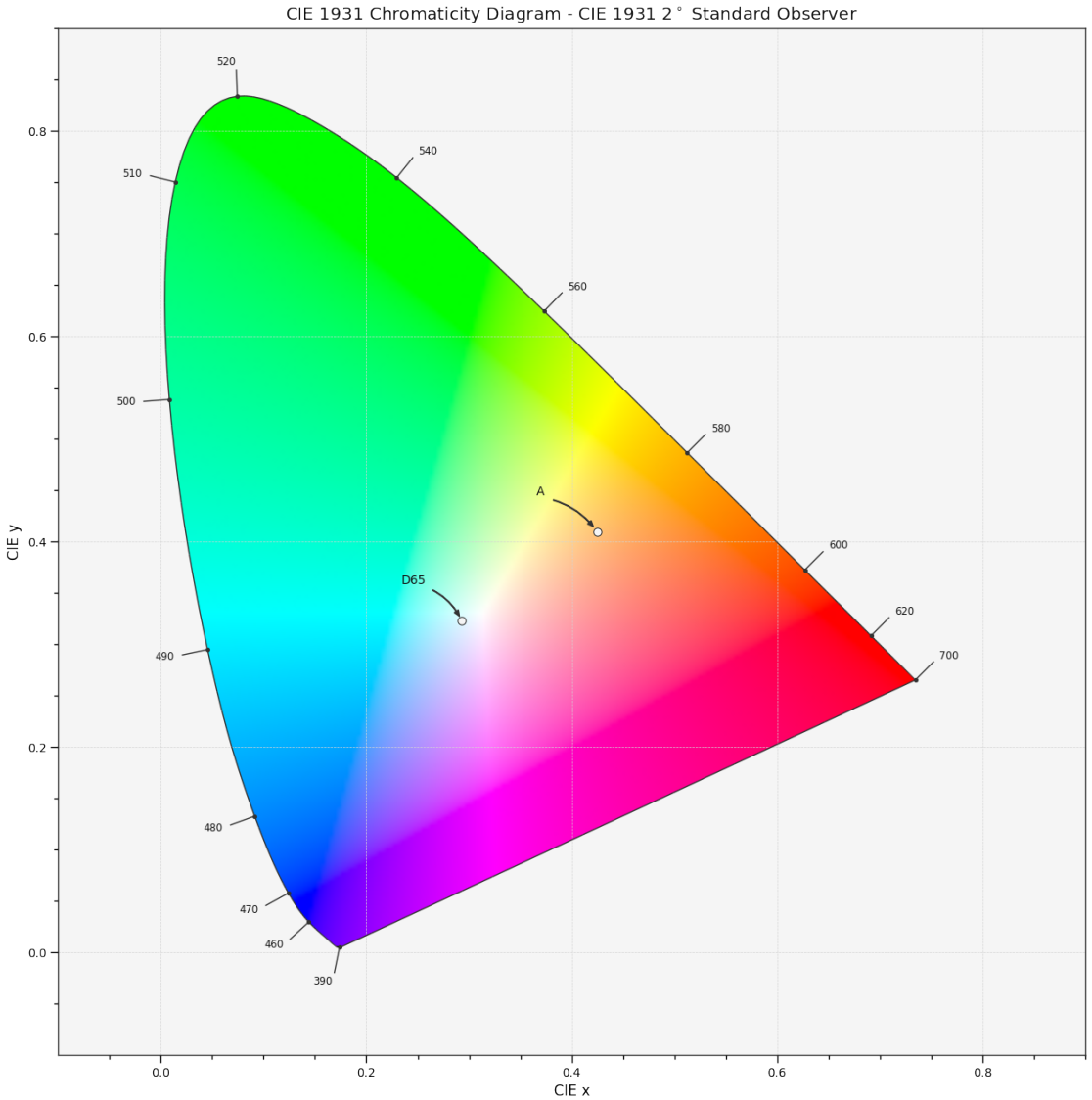
refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> A = SDS_ILLUMINANTS['A']
>>> D65 = SDS_ILLUMINANTS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1931([A, D65])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1960UCS**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1960UCS=<function plot_chromaticity_diagram_CIE1960UCS>*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs 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. The following special keyword arguments can also be used:
  - *annotate* : bool, whether to annotate the spectral distributions.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted spectral distributions. plot_kwargs can be either a single dictionary applied to all the plotted spectral distributions with same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - *illuminant* : unicode or colour.SpectralDistribution, the illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *cmfs* : unicode, the standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *normalise_sd_colours* : bool, whether to normalise the computed spectral distributions colours. The default is *True*.
  - *use_sd_colours* : bool, whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the plt.plot() definition color argument with pre-computed values. The default is *True*.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please

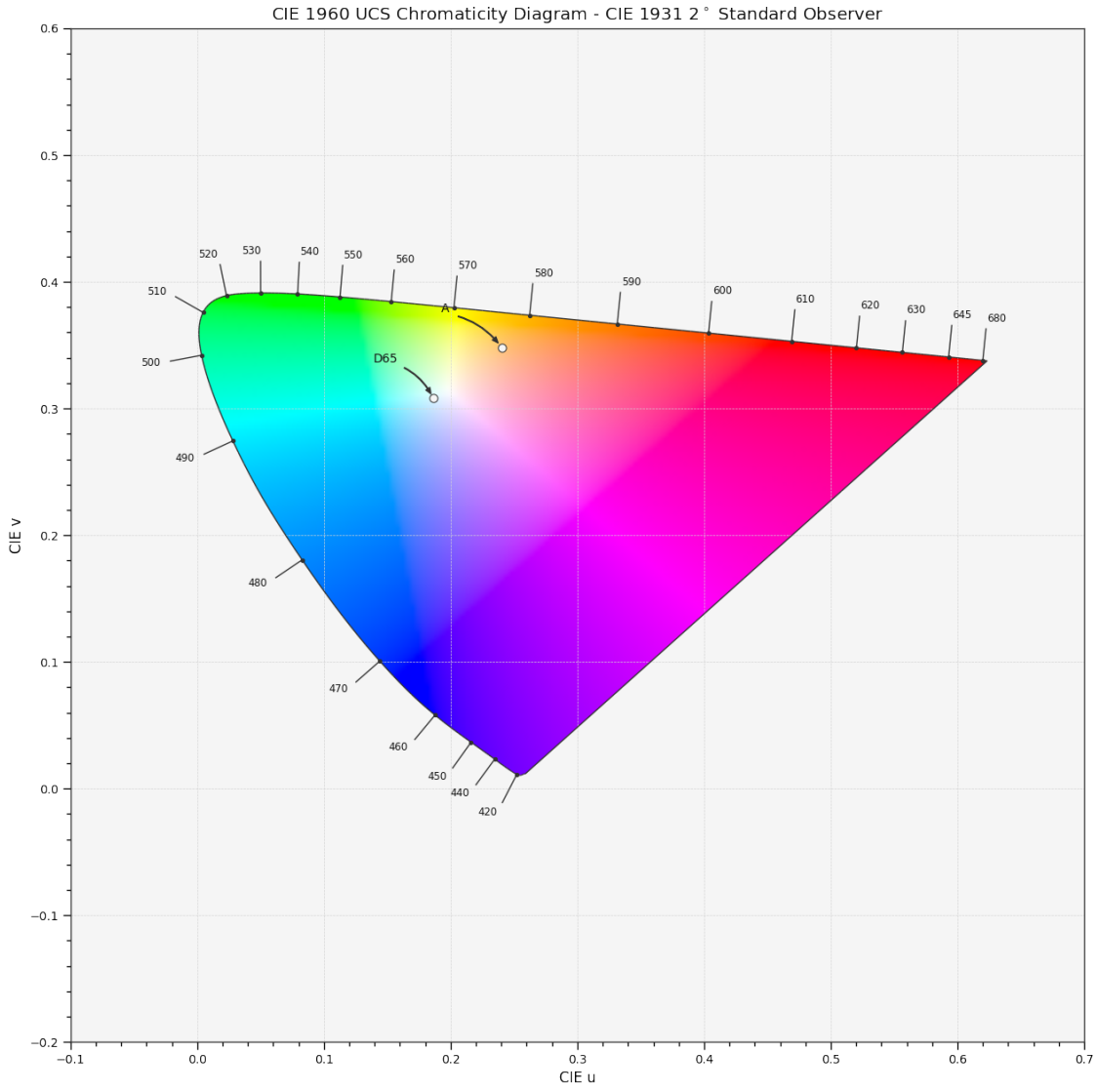
refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> A = SDS_ILLUMINANTS['A']
>>> D65 = SDS_ILLUMINANTS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1960UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1976UCS**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1976UCS=<function plot_chromaticity_diagram_CIE1976UCS>*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. annotate_kwargs 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. The following special keyword arguments can also be used:
  - *annotate* : bool, whether to annotate the spectral distributions.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted spectral distributions. plot_kwargs can be either a single dictionary applied to all the plotted spectral distributions with same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - *illuminant* : unicode or colour.SpectralDistribution, the illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. illuminant can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *cmfs* : unicode, the standard observer colour matching functions used for computing the spectral distributions colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
  - *normalise_sd_colours* : bool, whether to normalise the computed spectral distributions colours. The default is *True*.
  - *use_sd_colours* : bool, whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the plt.plot() definition color argument with pre-computed values. The default is *True*.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please

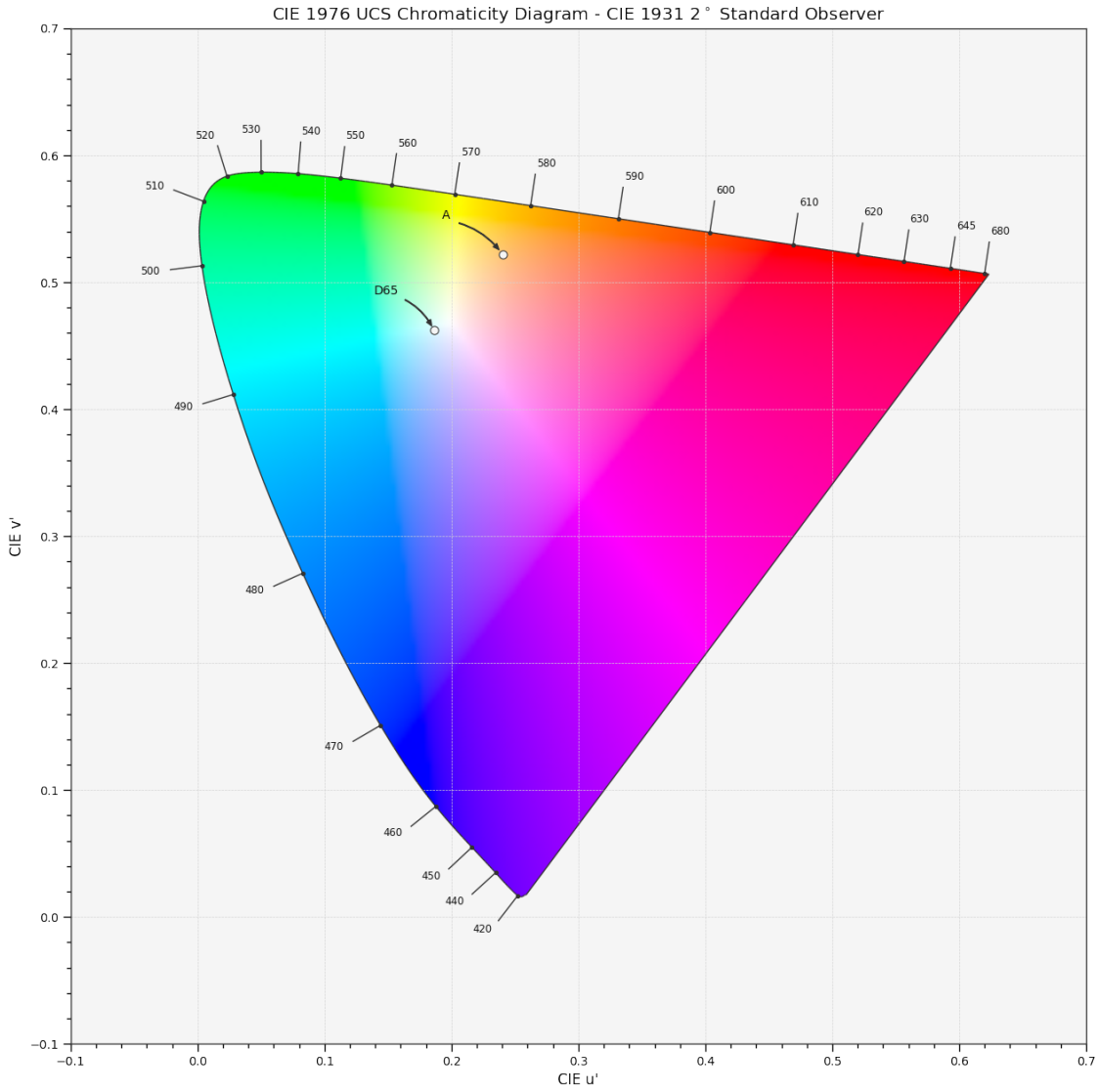
refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> A = SDS_ILLUMINANTS['A']
>>> D65 = SDS_ILLUMINANTS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1976UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Ancillary Objects

colour.plotting.diagrams

plot_spectral_locus([cmfs, ...])	Plots the *Spectral Locus* according to given method.
plot_chromaticity_diagram_colours([samples, ...])	Plots the *Chromaticity Diagram* colours according to given method.
plot_chromaticity_diagram([cmfs, ...])	Plots the *Chromaticity Diagram* according to given method.
plot_sds_in_chromaticity_diagram(sds[, ...])	Plots given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

colour.plotting.diagrams.plot_spectral_locus

colour.plotting.diagrams.**plot_spectral_locus**(*cmfs='CIE 1931 2 Degree Standard Observer'*, *spectral_locus_colours=None*, *spectral_locus_labels=None*, *method='CIE 1931'*, ***kwargs*)

Plots the *Spectral Locus* according to given method.

Parameters

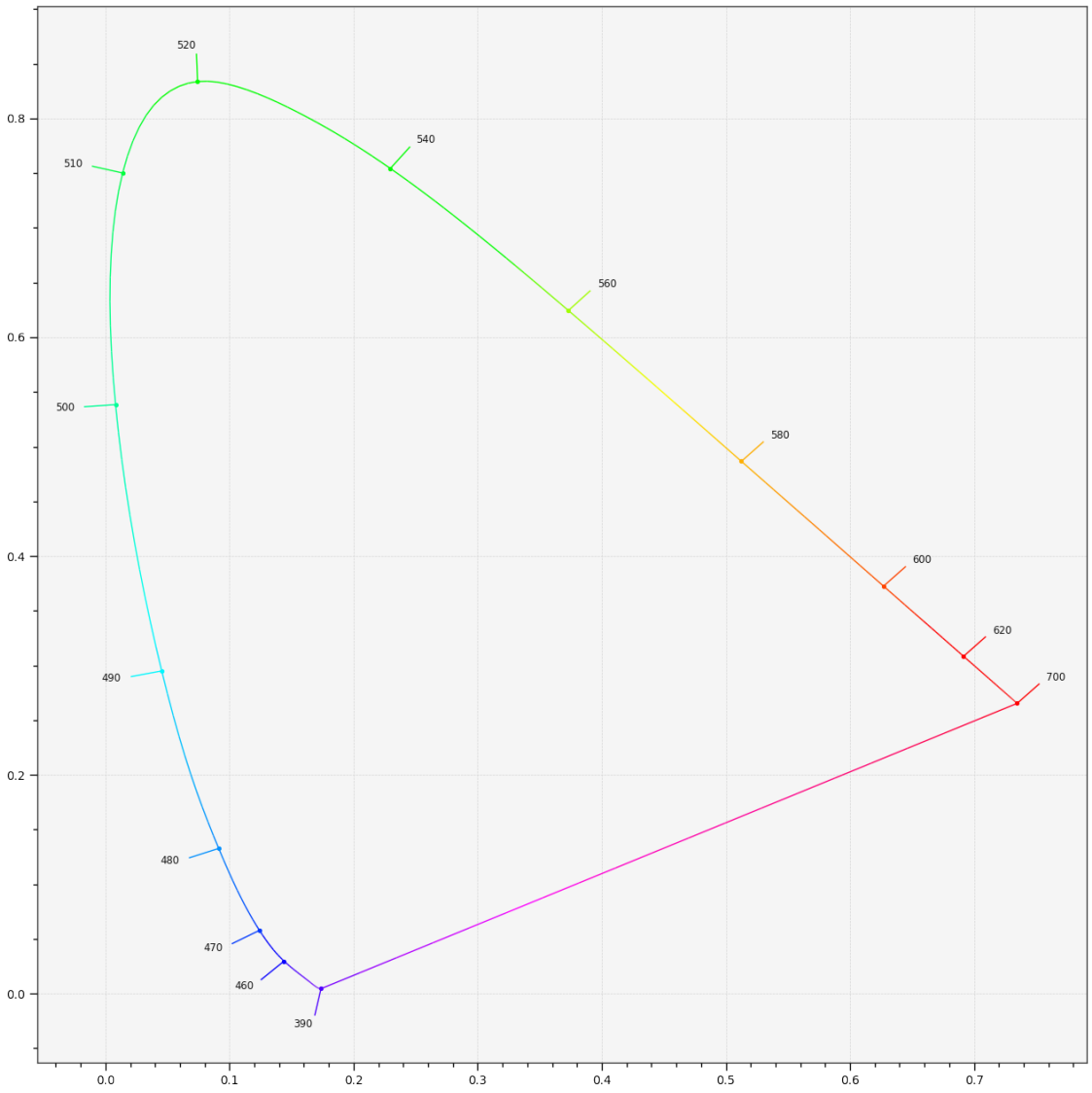
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **spectral_locus_colours** (array_like or unicode, 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.
- ****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')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.diagrams.plot_chromaticity_diagram_colours

colour.plotting.diagrams.**plot_chromaticity_diagram_colours**(*samples=256*, *diagram_opacity=1.0*, *diagram_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 or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus bound-

aries. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.

- **`method`** (`unicode`, `optional`) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- **`**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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

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

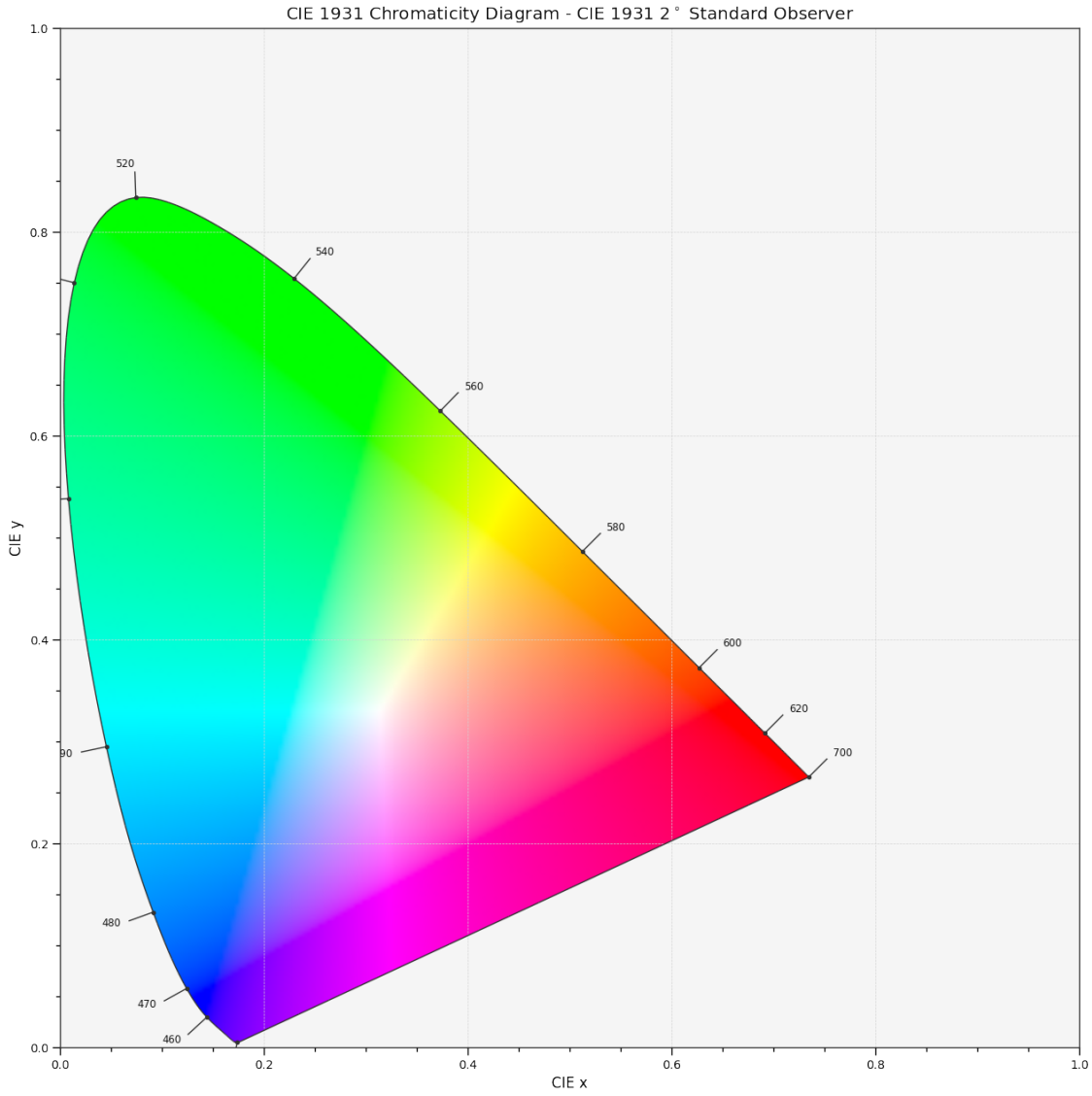
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **show_diagram_colours** (bool, optional) – Whether 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.
- ****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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.diagrams.plot_sds_in_chromaticity_diagram

colour.plotting.diagrams.**plot_sds_in_chromaticity_diagram**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *method='CIE 1931'*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus bound-

aries. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.

- **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.
- **annotate_kwargs** (`dict or array_like, optional`) – Keyword arguments for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names. `annotate_kwargs` 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. The following special keyword arguments can also be used:
  - *annotate* : bool, whether to annotate the spectral distributions.
- **plot_kwargs** (`dict or array_like, optional`) – Keyword arguments for the `plt.plot()` definition, used to control the style of the plotted spectral distributions. `plot_kwargs` can be either a single dictionary applied to all the plotted spectral distributions with same settings or a sequence of dictionaries with different settings for each plotted spectral distributions. The following special keyword arguments can also be used:
  - *illuminant* : unicode or `colour.SpectralDistribution`, the illuminant used to compute the spectral distributions colours. The default is the illuminant associated with the whitepoint of the default plotting colourspace. `illuminant` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.
  - *cmfs* : unicode, the standard observer colour matching functions used for computing the spectral distributions colours. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.
  - *normalise_sd_colours* : bool, whether to normalise the computed spectral distributions colours. The default is *True*.
  - *use_sd_colours* : bool, whether to use the computed spectral distributions colours under the plotting colourspace illuminant. Alternatively, it is possible to use the `plt.plot()` definition `color` argument with pre-computed values. The default is *True*.
- ****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. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

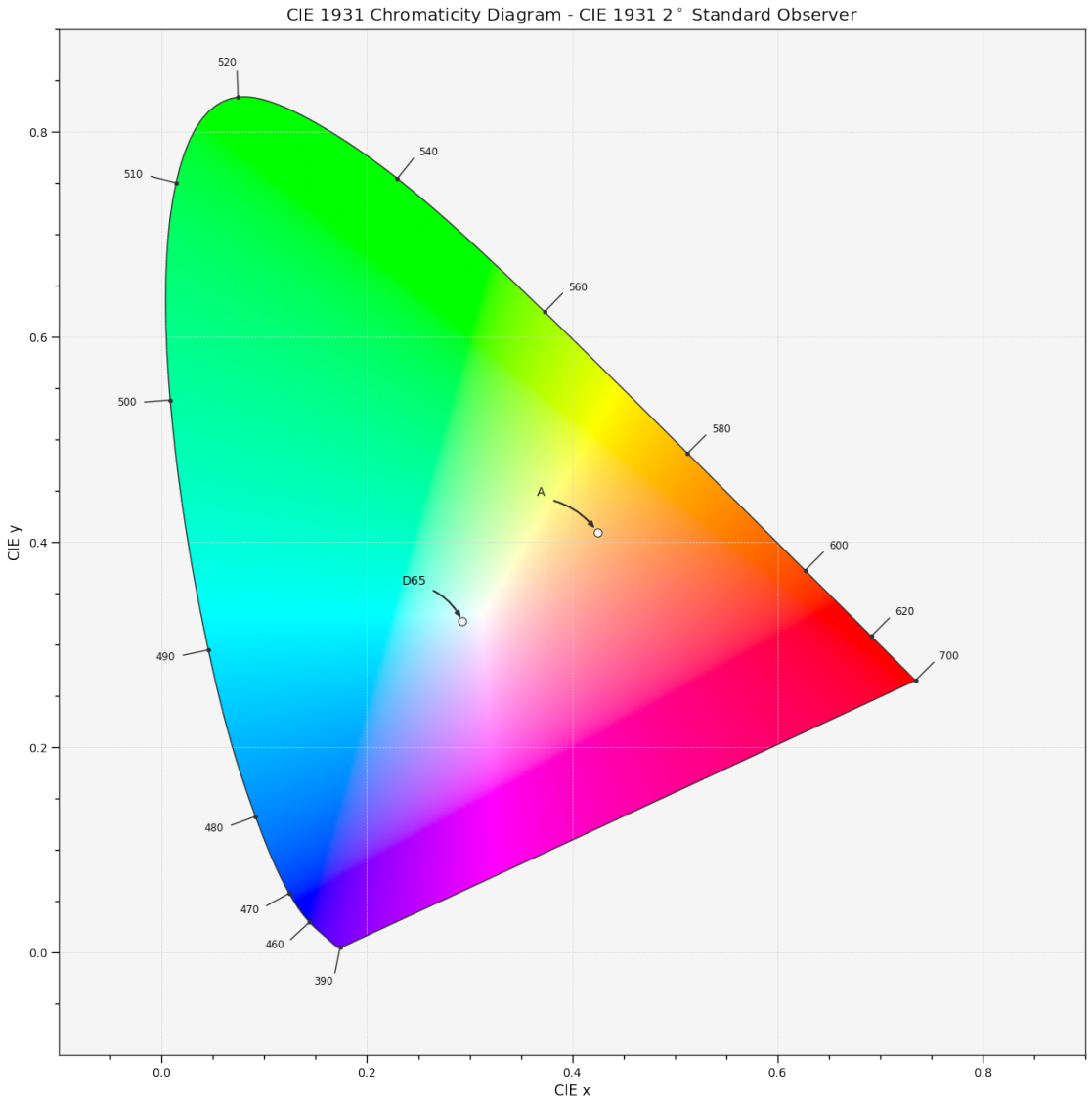
Examples

```
>>> A = SDS_ILLUMINANTS['A']
>>> D65 = SDS_ILLUMINANTS['D65']
>>> annotate_kwargs = [
...     {'xytext': (-25, 15), 'arrowprops':{'arrowstyle':'-'}},
...     {}
... ]
>>> plot_kwargs = [
...     {
```

(continues on next page)

(continued from previous page)

```
...         'illuminant': SDS_ILLUMINANTS['E'],
...         'markersize' : 15,
...         'normalise_sd_colours': True,
...         'use_sd_colours': True
...     },
...     {'illuminant': SDS_ILLUMINANTS['E']},
... ]
>>> plot_sds_in_chromaticity_diagram(
...     [A, D65], annotate_kwargs=annotate_kwargs, plot_kwargs=plot_kwargs)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Colour Models

colour.plotting

plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931(...)	Plots given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS(...)	Plots given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS(...)	Plots given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(RGB)	Plots given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(RGB)	Plots given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(RGB)	Plots given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1931 Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1976 UCS Chromaticity Diagram*.
plot_single_cctf(cctf[, cctf_decoding])	Plots given colourspace colour component transfer function.
plot_multi_cctfs(cctfs[, cctf_decoding])	Plots given colour component transfer functions.
plot_constant_hue_loci(data, model[, ...])	Plots given constant hue loci colour matches data such as that from [] or [] that are easily loaded with Colour - Datasets.

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931**(*colourspaces*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1931=<functio plot_chromaticity_diagram_CIE1931>*, *show_whitepoints=True*, *show_pointer_gamut=False*, *chromatically_adapt=False*, *plot_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

- **show_whitepoints** (bool, optional) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool, optional) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (bool, optional) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- ****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'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS**(*colourspaces*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1960UCS=<*
plot_chromaticity_diagram_CIE1960U
show_whitepoints=True, *show_pointer_gamut=False*, *chromatically_adapt=False*, *plot_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form sup-

ported by the colour.plotting.filter_RGB_colourspaces() definition.

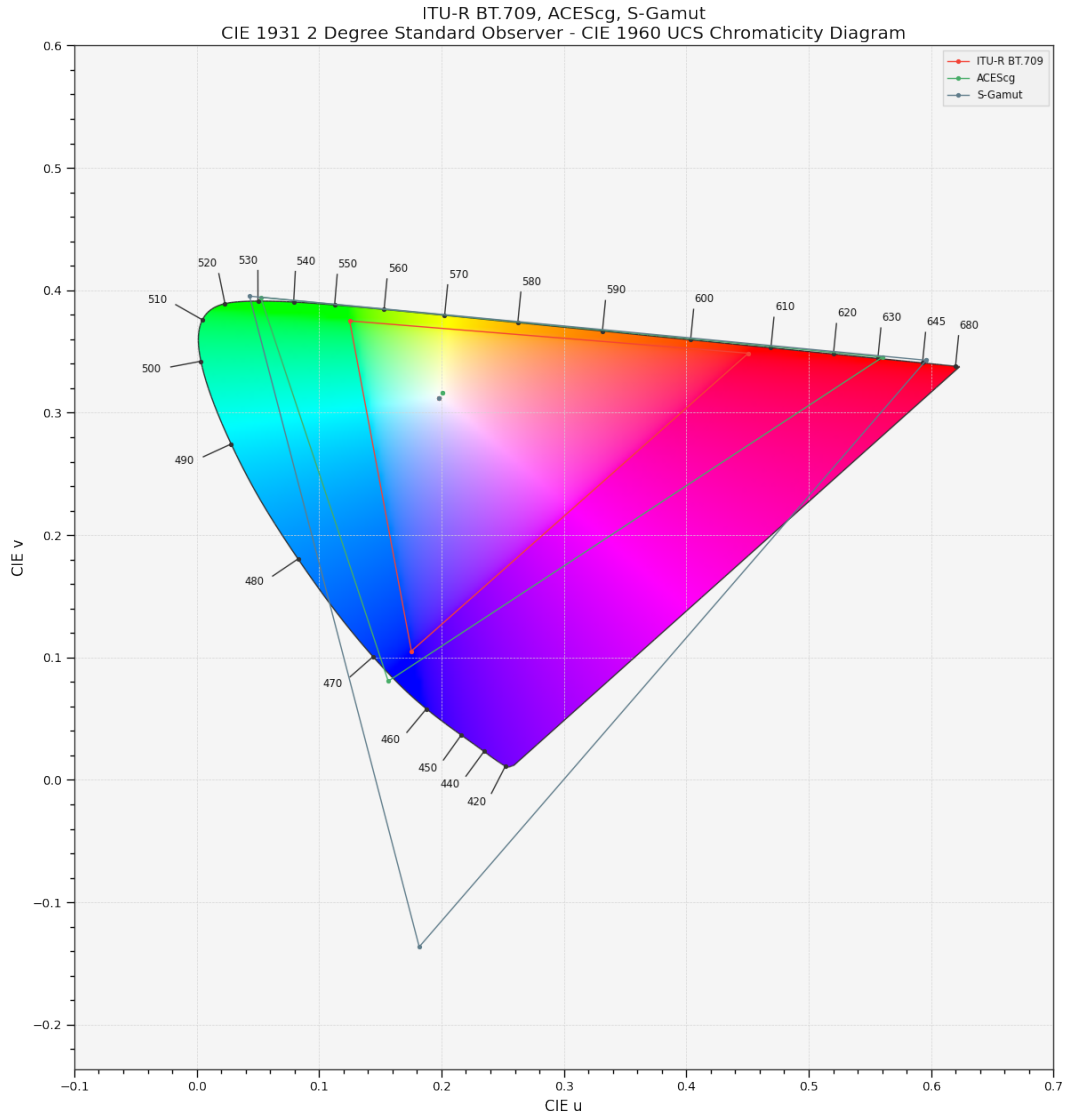
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **show_whitepoints** (bool, optional) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool, optional) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (bool, optional) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- ****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_CIE1960UCS(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS**(*colourspaces*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1976UCS=<*
plot_chromaticity_diagram_CIE1976U
show_whitepoints=True, *show_pointer_gamut=False*, *chromatically_adapt=False*, *plot_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form sup-

ported by the `colour.plotting.filter_RGB_colourspaces()` definition.

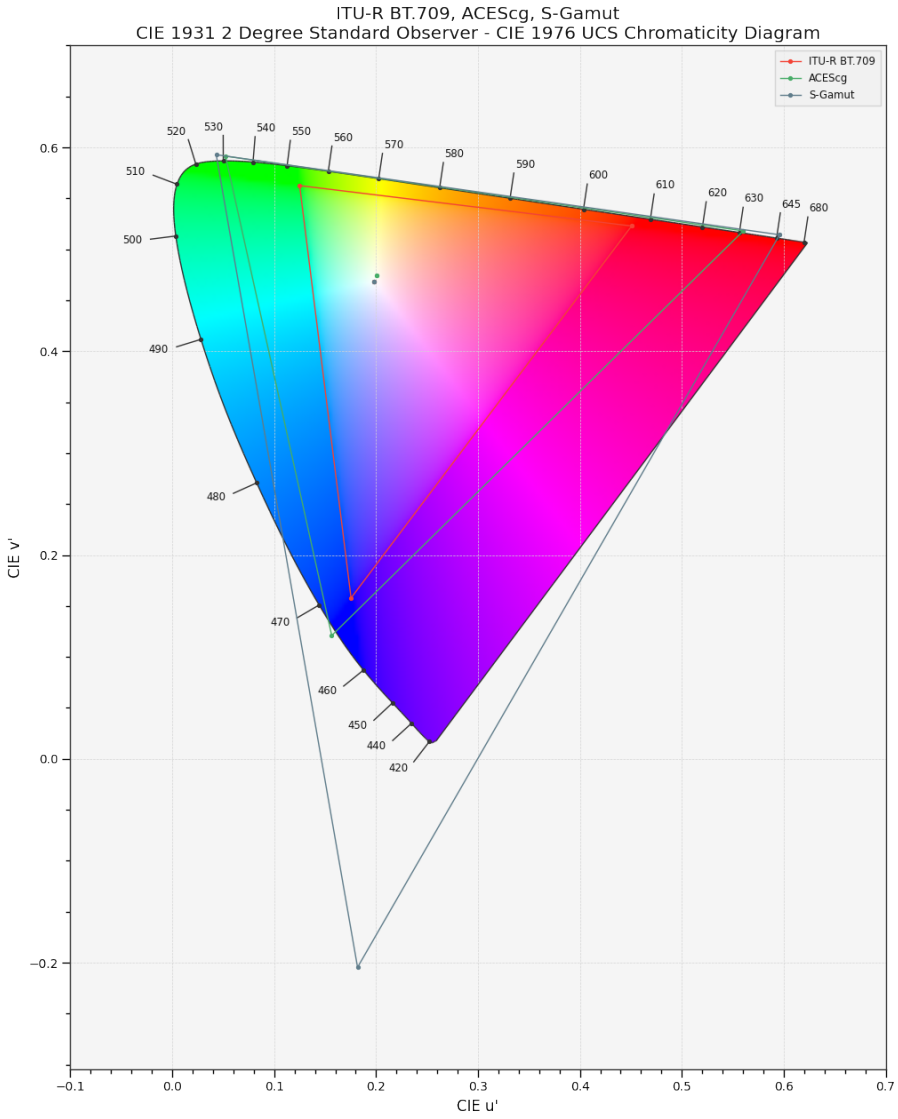
- **cmfs** (`unicode or XYZ_ColourMatchingFunctions, optional`) – Standard observer colour matching functions used for computing the spectral locus boundaries. `cmfs` can be of any type or form supported by the `colour.plotting.filter_cmfs()` definition.
- **chromaticity_diagram_callable_CIE1976UCS** (`callable, optional`) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **show_whitepoints** (`bool, optional`) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (`bool, optional`) – Whether to display the *Pointer's Gamut*.
- **chromatically_adapt** (`bool, optional`) – Whether to chromatically adapt the *RGB* colourspaces given in `colourspaces` to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (`dict or array_like, optional`) – Keyword arguments for the `plt.plot()` definition, used to control the style of the plotted *RGB* colourspaces. `plot_kwargs` can be either a single dictionary applied to all the plotted *RGB* colourspaces with same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- ****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_CIE1976UCS(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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=<func*
plot_RGB_colourspaces_in_chromaticity
scatter_kwargs=None, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (unicode or RGB_Colourspace, optional) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

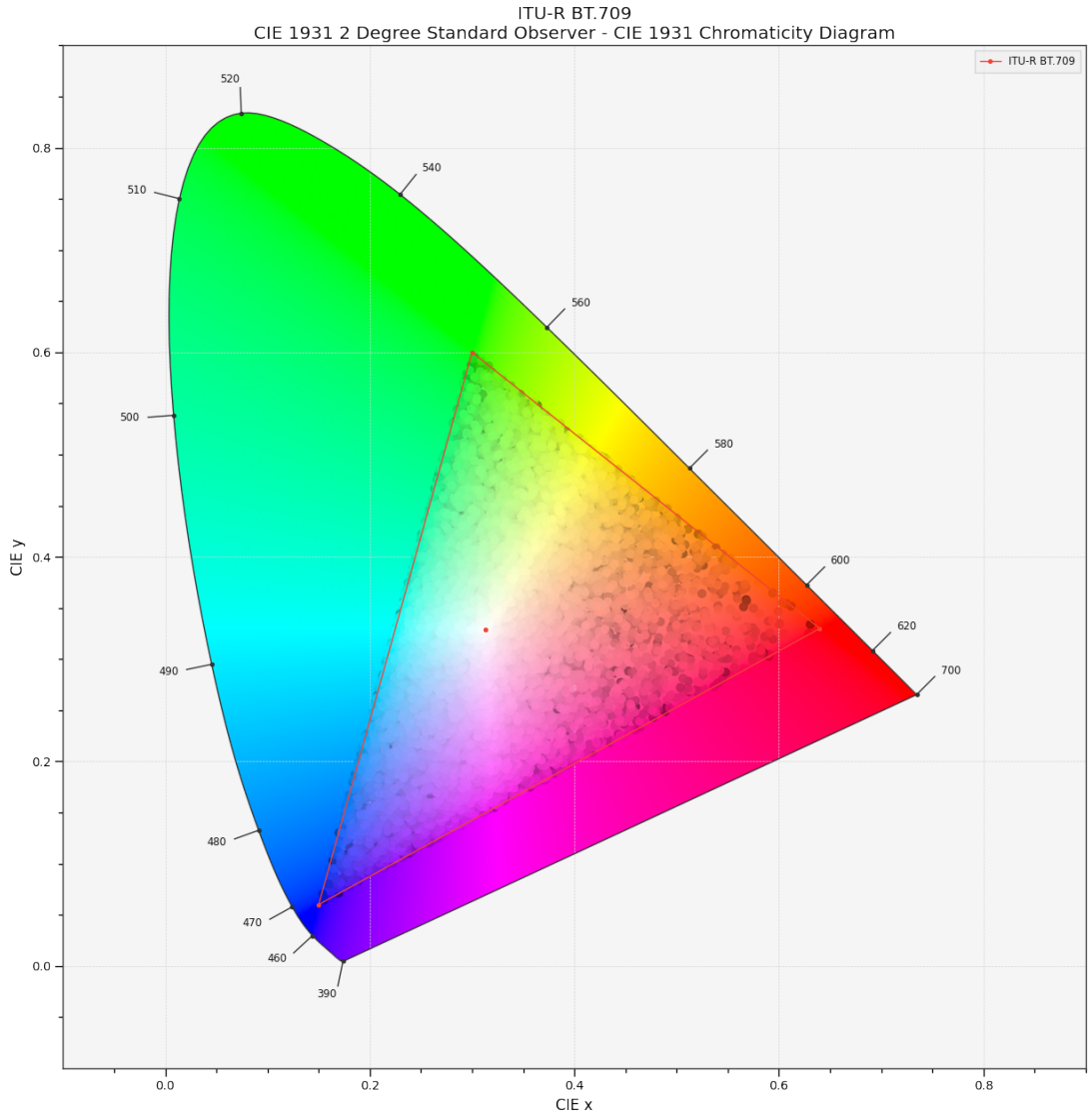
- **scatter_kwargs** (dict, optional) – Keyword arguments for the `plt.scatter()` definition. The following special keyword arguments can also be used:
  - *c* : unicode or array_like, if `c` is set to *RGB*, the scatter will use the colours as given by the `RGB` argument.
- ****kwargs** (dict, optional) – {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. Also handles keywords arguments for deprecation management.

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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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=plot_RGB_colourspaces_in_chromat*, *scatter_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (unicode or RGB_Colourspace, optional) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

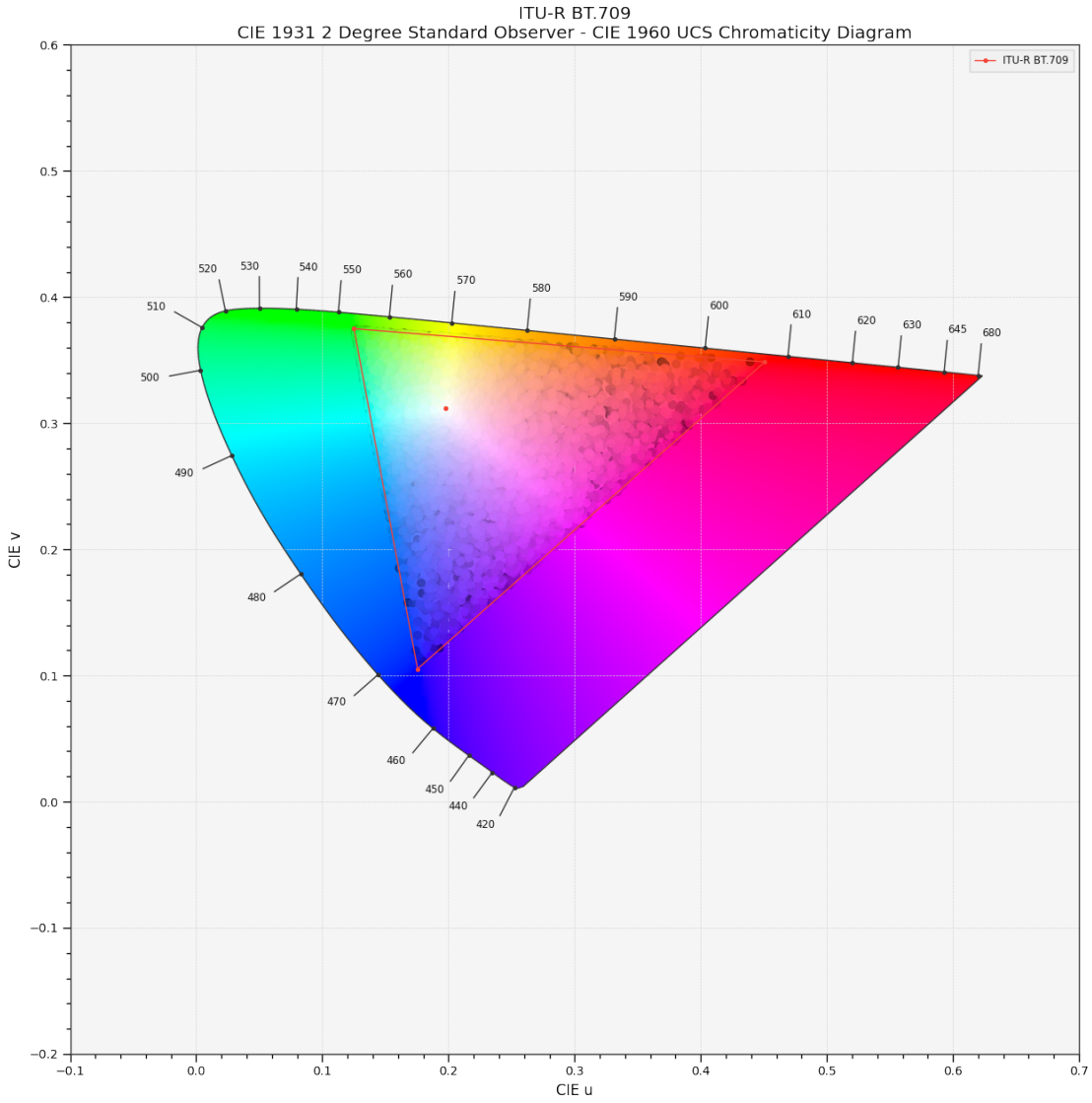
- **scatter_kwargs** (dict, optional) – Keyword arguments for the plt.scatter() definition. The following special keyword arguments can also be used:
  - *c* : unicode or array_like, if c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
- ****kwargs** (dict, optional) – {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. Also handles keywords arguments for deprecation management.

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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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=plot_RGB_colourspaces_in_chromat*, *scatter_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (unicode or RGB_Colourspace, optional) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

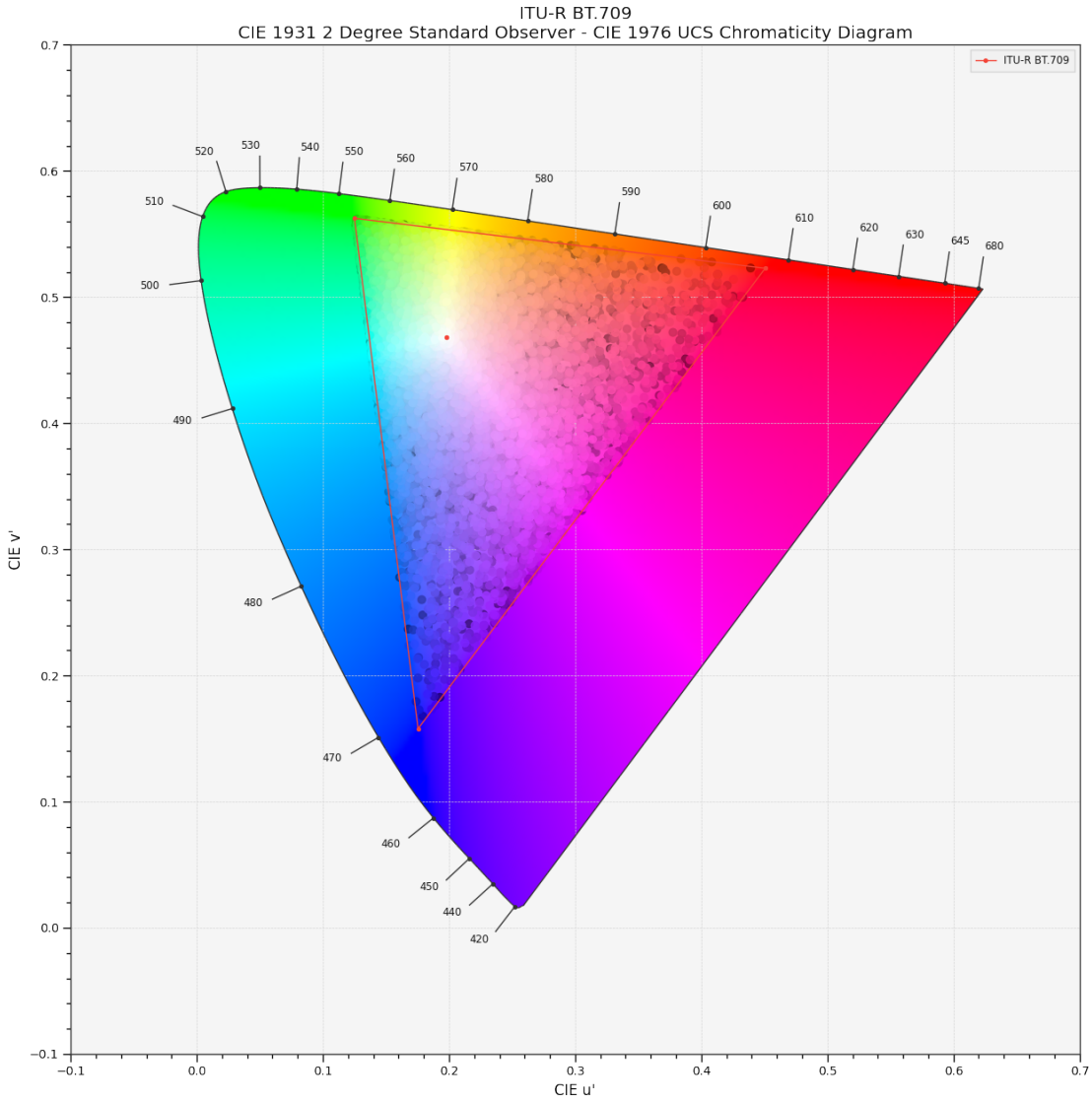
- **scatter_kwargs** (dict, optional) – Keyword arguments for the `plt.scatter()` definition. The following special keyword arguments can also be used:
  - *c* : unicode or array_like, if `c` is set to *RGB*, the scatter will use the colours as given by the `RGB` argument.
- ****kwargs** (dict, optional) – {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. Also handles keywords arguments for deprecation management.

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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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_kwargs=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_kwargs** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_kwargs 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.

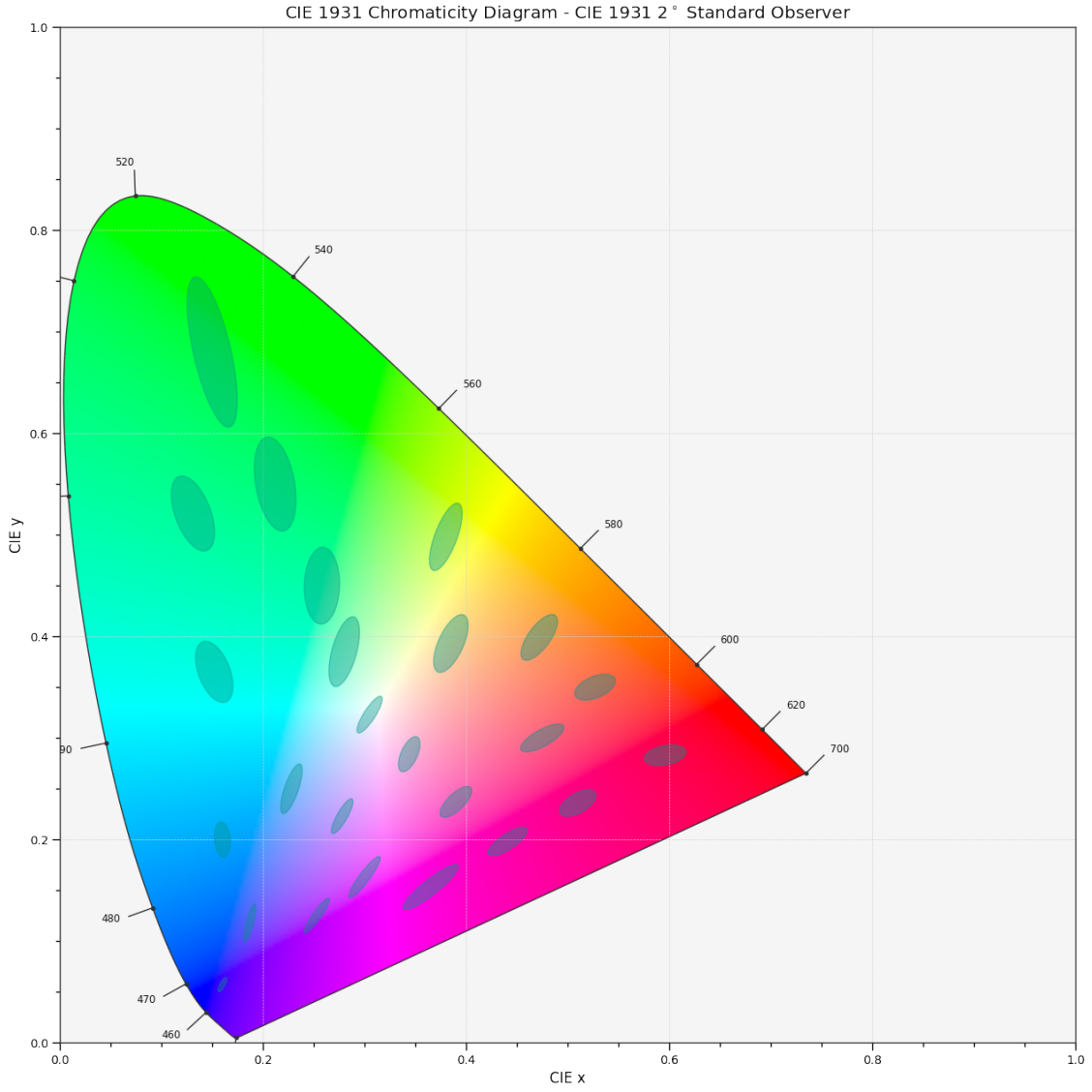
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, Please refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931()
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS**(*chromaticity_diagram_callable_C*
plot_chromaticity_diagram_CIE1
chromatic-
ity_diagram_clipping=False,
el-
lipse_kwargs=None,
***kwargs*)

Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

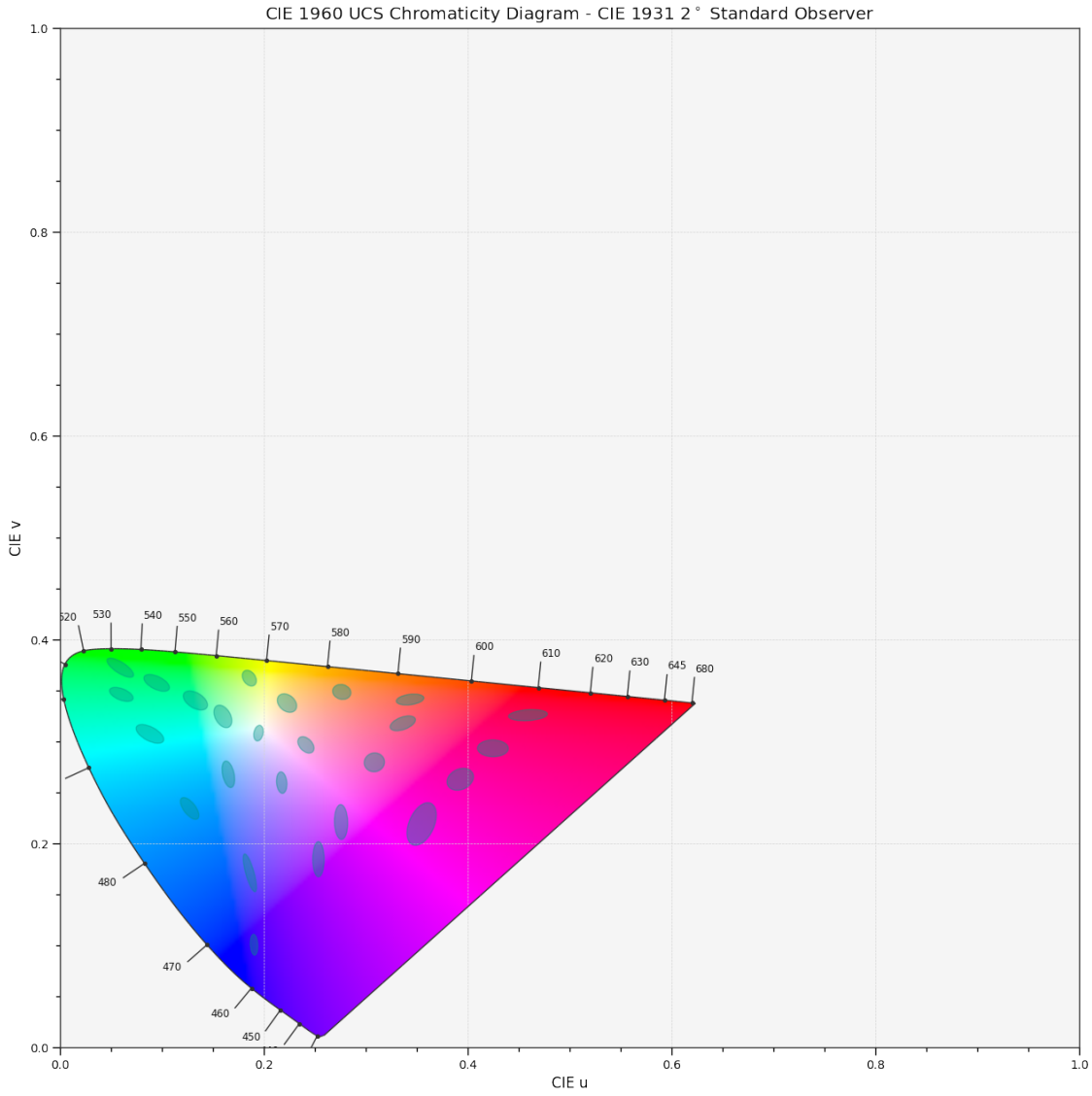
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool, optional,) – Whether to clip the *CIE 1960 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_kwargs** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_kwargs 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.
- ****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. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS()
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS**(*chromaticity_diagram_callable_C*
plot_chromaticity_diagram_CIE1
chromatic-
ity_diagram_clipping=False,
el-
lipse_kwargs=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_kwargs** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_kwargs 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.

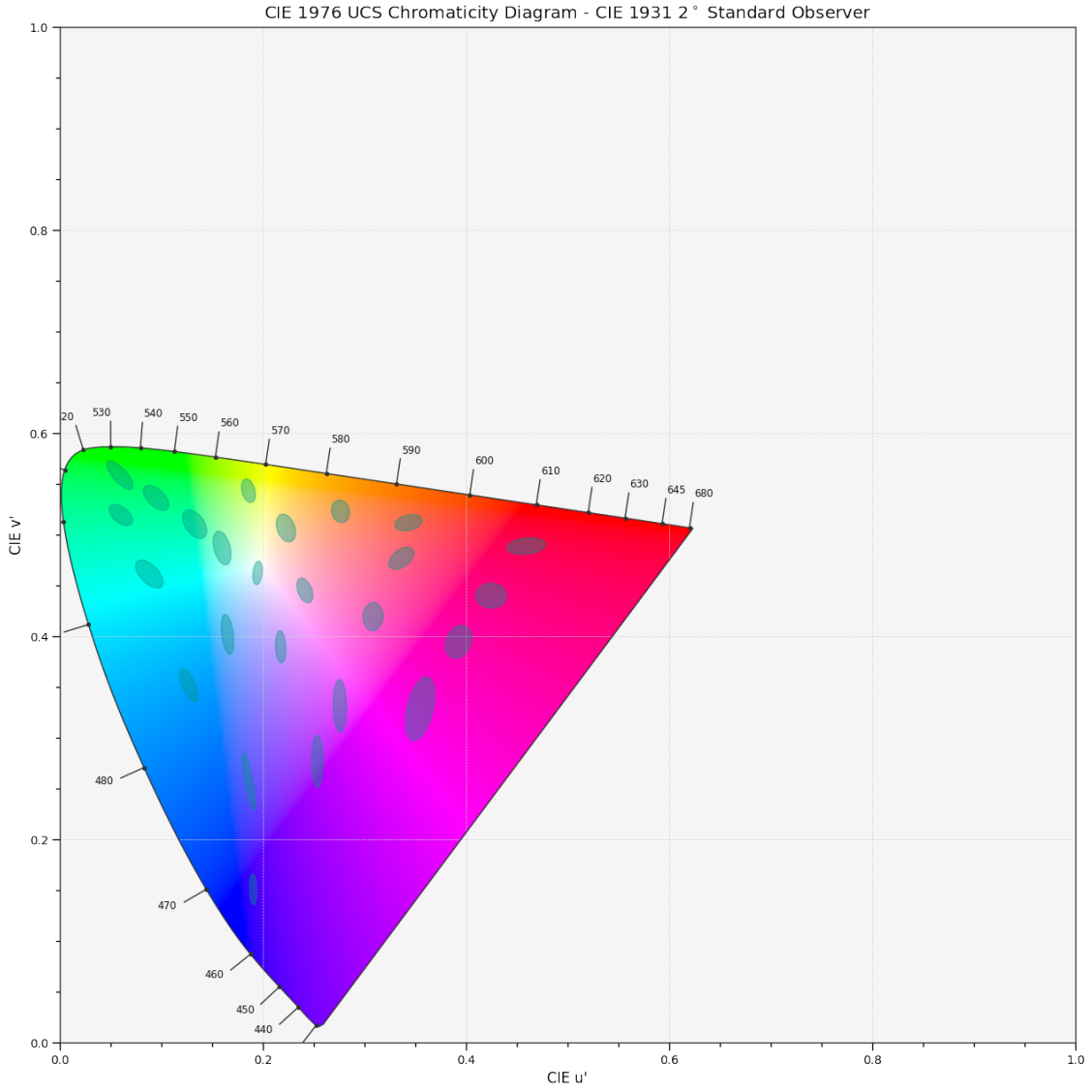
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, Please refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS()
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_cctf

colour.plotting.**plot_single_cctf**(*cctf*, *cctf_decoding=False*, ***kwargs*)

Plots given colourspace colour component transfer function.

Parameters

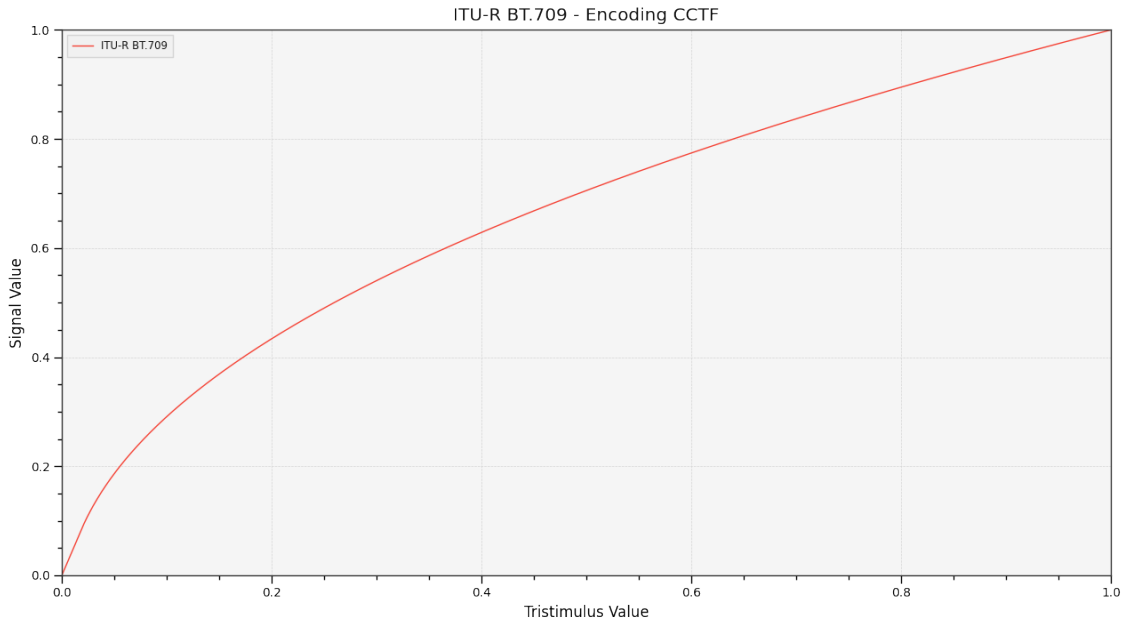
- **cctf** (unicode or object) – Colour component transfer function to plot. function can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- **cctf_decoding** (bool) – Plot the decoding colour component transfer function instead.
- ****kwargs** (dict, optional) – {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')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_cctfs

colour.plotting.**plot_multi_cctfs**(*cctfs*, *cctf_decoding=False*, ***kwargs*)

Plots given colour component transfer functions.

Parameters

- **cctfs** (unicode or object or array_like, optional) – Colour component transfer function to plot. cctfs elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.

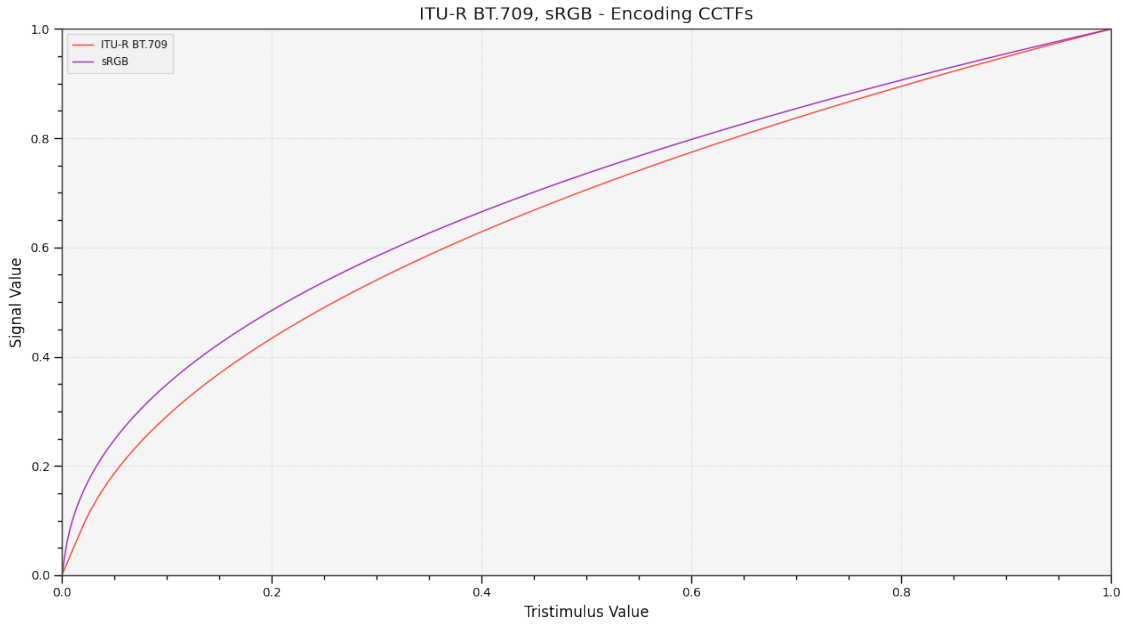
- **cctf_decoding** (bool) – Plot the decoding colour component transfer function instead.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_cctfs(['ITU-R BT.709', 'sRGB'])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_constant_hue_loci

colour.plotting.**plot_constant_hue_loci**(*data*, *model*, *scatter_kwargs=None*, ***kwargs*)

Plots given constant hue loci colour matches data such as that from [] or [] that are easily loaded with Colour - Datasets.

Parameters

- **data** (array_like) – Constant hue loci colour matches data expected to be an *array_like* as follows:

```
[
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}
↪),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}
↪),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}
↪),
    ...
]
```

where `name` is the hue angle or name, `XYZ_r` the *CIE XYZ* tristimulus values of the reference illuminant, `XYZ_cr` the *CIE XYZ* tristimulus values of the reference colour under the reference illuminant, `XYZ_ct` the *CIE XYZ* tristimulus values of the colour matches under the reference illuminant and `metadata` the dataset metadata.

- **model** (`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'}**, Colourspace model.
- **scatter_kwargs** (`dict, optional`) – Keyword arguments for the `plt.scatter()` definition. The following special keyword arguments can also be used:
  - *c* : unicode or array_like, if `c` is set to *RGB*, the scatter will use the colours as given by the `RGB` argument.
- ****kwargs** (`dict, optional`) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

References

[], [], []

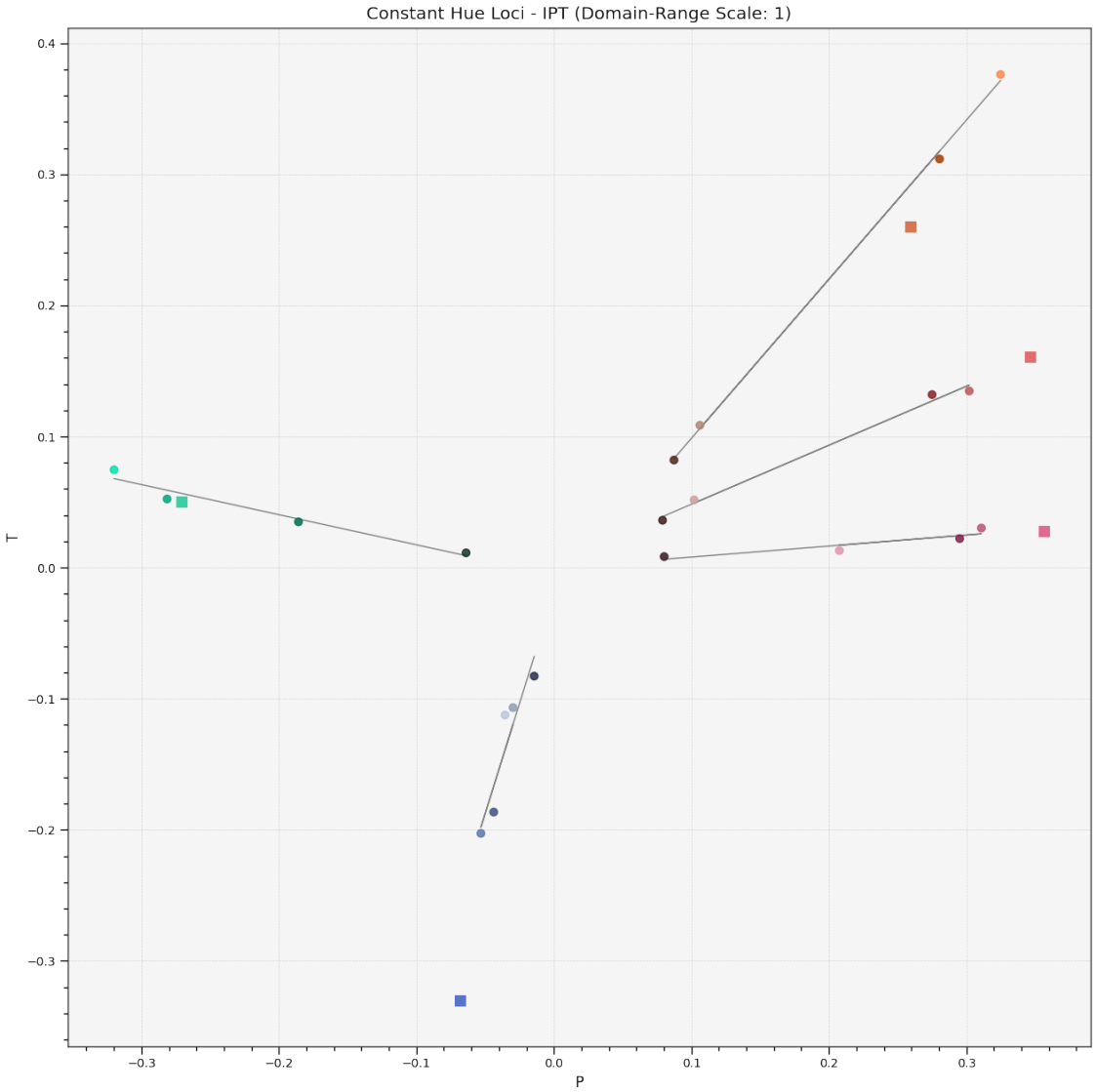
Examples

```
>>> data = np.array([
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.40920000, 0.28120000, 0.30600000]),
...         np.array([
...             [0.02495100, 0.01908600, 0.02032900],
...             [0.10944300, 0.06235900, 0.06788100],
...             [0.27186500, 0.18418700, 0.19565300],
...             [0.48898900, 0.40749400, 0.44854600],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.30760000, 0.48280000, 0.42770000]),
...         np.array([
...             [0.02108000, 0.02989100, 0.02790400],
...             [0.06194700, 0.11251000, 0.09334400],
...             [0.15255800, 0.28123300, 0.23234900],
...             [0.34157700, 0.56681300, 0.47035300],
...         ]),
...         None,
...     ],
...     [
...         None,
```

(continues on next page)

(continued from previous page)

```
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.39530000, 0.28120000, 0.18450000]),
...         np.array([
...             [0.02436400, 0.01908600, 0.01468800],
...             [0.10331200, 0.06235900, 0.02854600],
...             [0.26311900, 0.18418700, 0.12109700],
...             [0.43158700, 0.40749400, 0.39008600],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.20510000, 0.18420000, 0.57130000]),
...         np.array([
...             [0.03039800, 0.02989100, 0.06123300],
...             [0.08870000, 0.08498400, 0.21843500],
...             [0.18405800, 0.18418700, 0.40111400],
...             [0.32550100, 0.34047200, 0.50296900],
...             [0.53826100, 0.56681300, 0.80010400],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.35770000, 0.28120000, 0.11250000]),
...         np.array([
...             [0.03678100, 0.02989100, 0.01481100],
...             [0.17127700, 0.11251000, 0.01229900],
...             [0.30080900, 0.28123300, 0.21229800],
...             [0.52976000, 0.40749400, 0.11720000],
...         ]),
...         None,
...     ],
... ])
>>> plot_constant_hue_loci(data, 'IPT')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Ancillary Objects

`colour.plotting.models`

`common_colourspace_model_axis_reorder`(a[, model])	Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.
`plot_pointer_gamut`([method])	Plots *Pointer's Gamut* according to given method.
`plot_RGB_colourspaces_in_chromaticity_diagram`()	Plots given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.
`plot_RGB_chromaticities_in_chromaticity_diagram`(RGB)	Plots given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

colour.plotting.models.common_colourspace_model_axis_reorder

colour.plotting.models.**common_colourspace_model_axis_reorder**(*a*, *model=None*)

Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.

Parameters

- **a** (array_like) – Colourspace model a array.
- **model** (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'}**, Colourspace model.

Returns Reordered colourspace model a array.

Return type ndarray

Examples

```
>>> a = np.array([0, 1, 2])
>>> common_colourspace_model_axis_reorder(a)
array([0, 1, 2])
>>> common_colourspace_model_axis_reorder(a, 'CIE Lab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE LCHab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE Luv')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE LCHab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'DIN 99')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'Hunter Lab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'Hunter Rdab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'IPT')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'JzAzBz')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'OSA UCS')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'hdr-CIELAB')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'hdr-IPT')
array([ 1.,  2.,  0.])
```

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

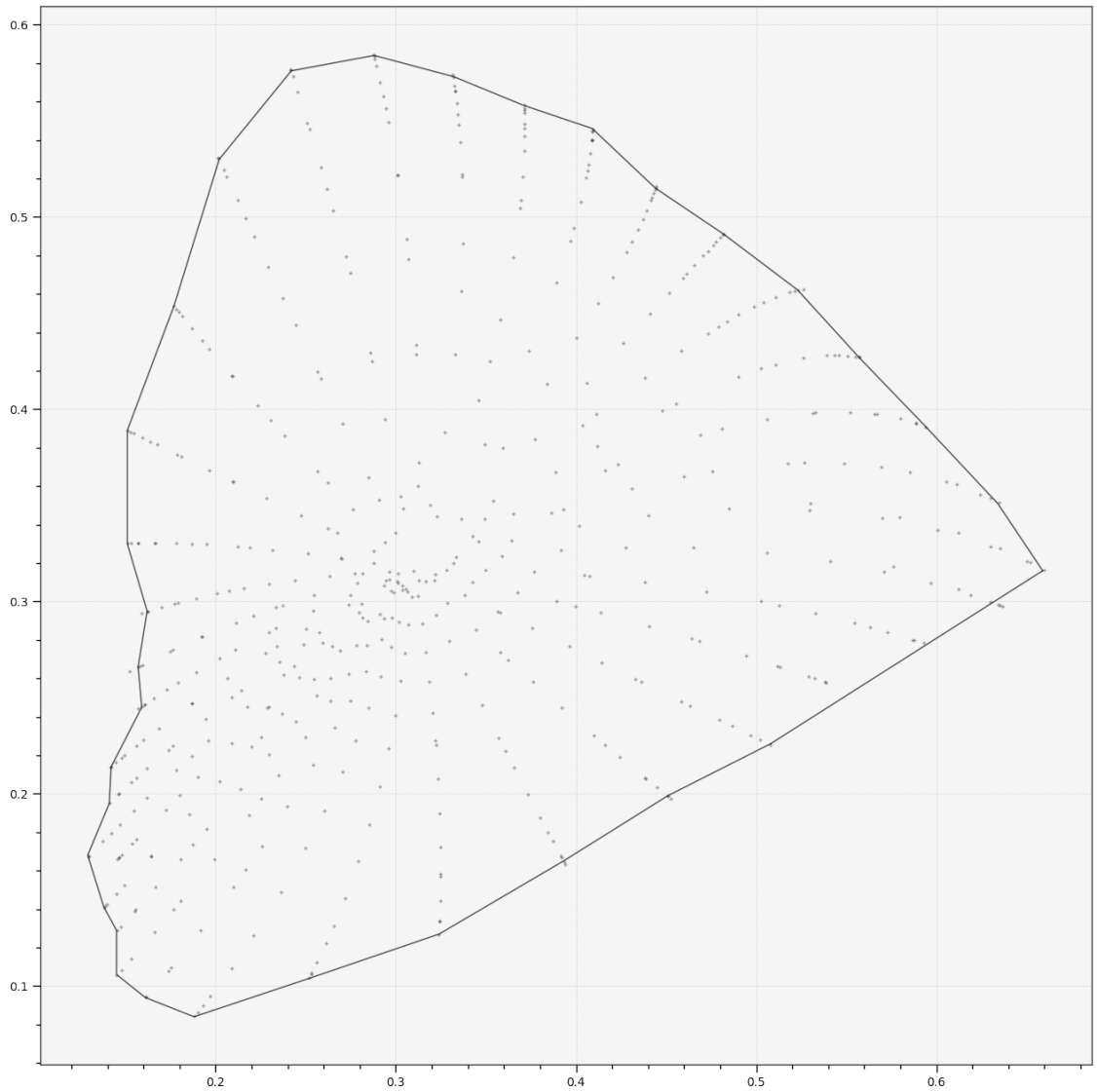
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, Plotting method.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_pointer_gamut()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram

colour.plotting.models.**plot_RGB_colourspaces_in_chromaticity_diagram**(*colourspaces*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *method='CIE 1931'*, *show_whitepoints=True*, *show_pointer_gamut=False*, *chromatically_adapt=False*, *plot_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.

Parameters

- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot. colourspaces elements can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **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*.
- **chromatically_adapt** (bool, optional) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted *RGB* colourspaces. plot_kwargs can be either a single dictionary applied to all the plotted *RGB* colourspaces with same settings or a sequence of dictionaries with different settings for each plotted *RGB* colourspace.
- ****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_kwargs = [
...     {'color': 'r'},
...     {'linestyle': 'dashed'},
...     {'marker': None}
... ]
>>> plot_RGB_colourspaces_in_chromaticity_diagram(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'], plot_kwargs=plot_kwargs)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```

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_colourspaces_in_chromaticity_*, *method='CIE 1931'*, *scatter_kwargs=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

Parameters

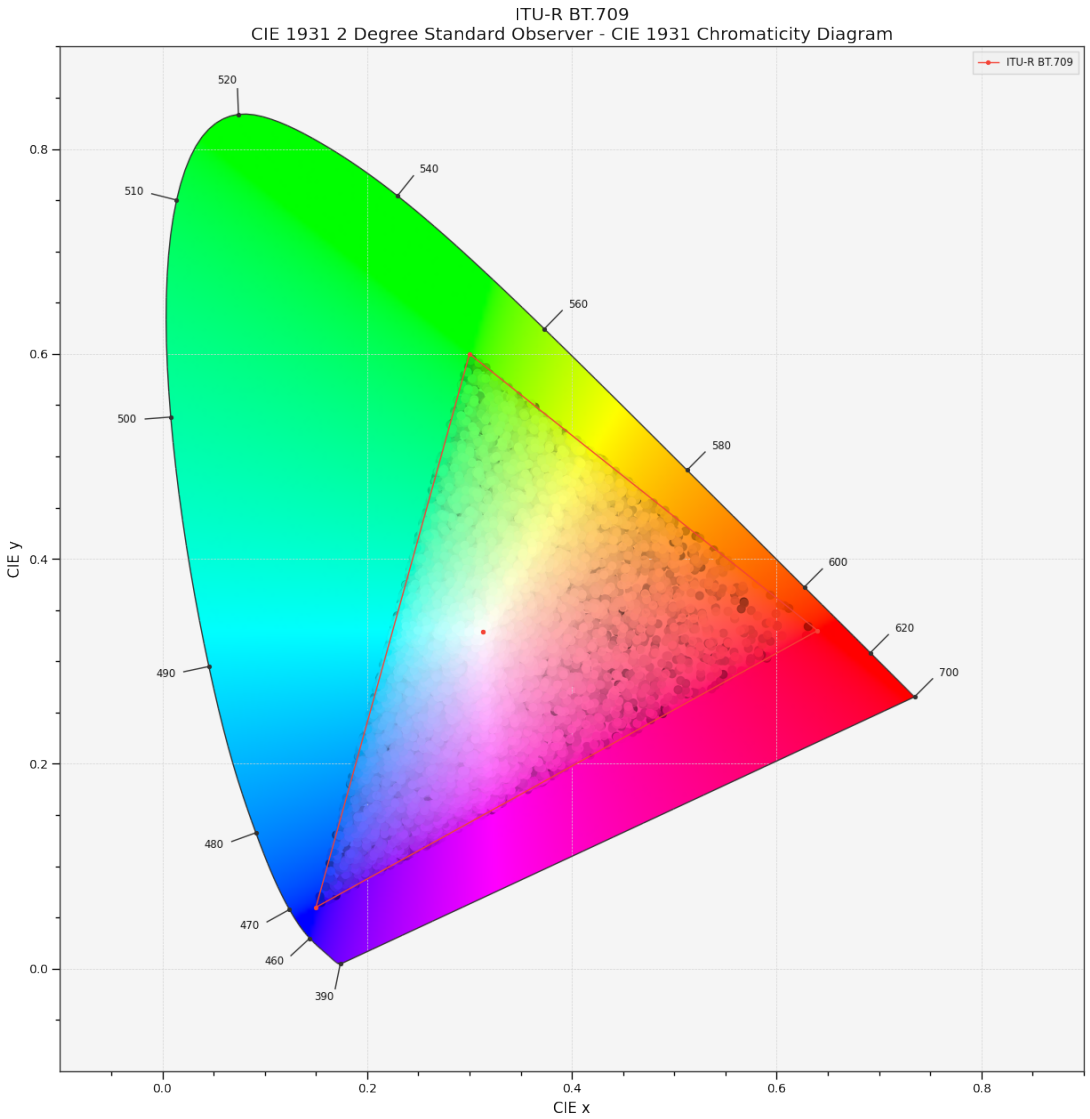
- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (unicode or RGB_Colourspace, optional) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- **scatter_kwargs** (dict, optional) – Keyword arguments for the plt.scatter() definition. The following special keyword arguments can also be used:
  - *c* : unicode or array_like, if c is set to *RGB*, the scatter will use the colours as given by the RGB argument.
- ****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. Also handles keywords arguments for deprecation management.

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')
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Colour Notation Systems

`colour.plotting`

`plot_single_munsell_value_function`(function, ...)	Plots given *Lightness* function.
`plot_multi_munsell_value_functions`(...)	Plots given *Munsell* value functions.

colour.plotting.plot_single_munsell_value_function

colour.plotting.**plot_single_munsell_value_function**(*function*, ***kwargs*)

Plots given *Lightness* function.

Parameters

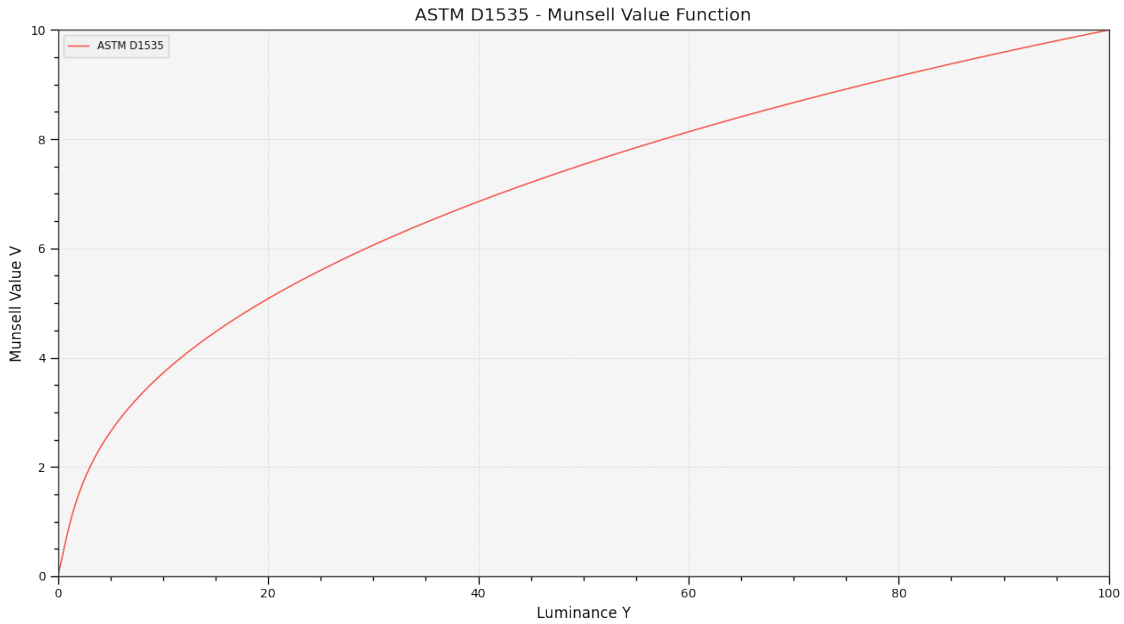
- **function** (unicode or object) – *Munsell* value function to plot. function can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, 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')
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_munsell_value_functions

colour.plotting.**plot_multi_munsell_value_functions**(*functions*, ***kwargs*)

Plots given *Munsell* value functions.

Parameters

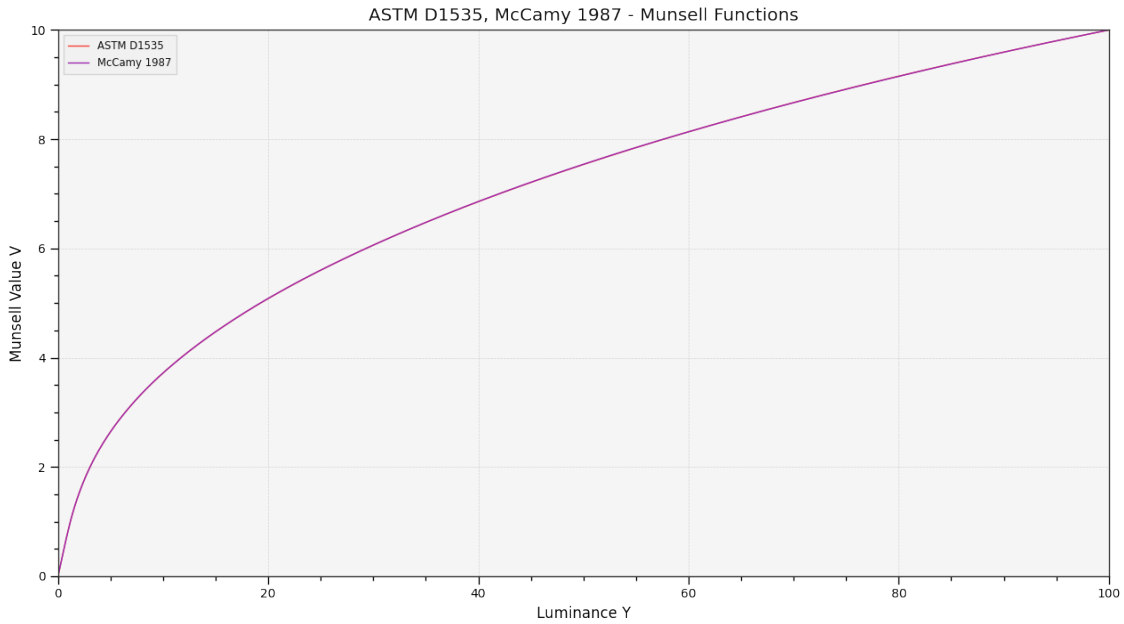
- **functions** (unicode or object or array_like) – *Munsell* value functions to plot. functions elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, 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', 'McCamy 1987'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Optical Phenomena

colour.plotting

plot_single_sd_rayleigh_scattering([...])	Plots a single *Rayleigh* scattering spectral distribution.
plot_the_blue_sky([cmfs])	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.

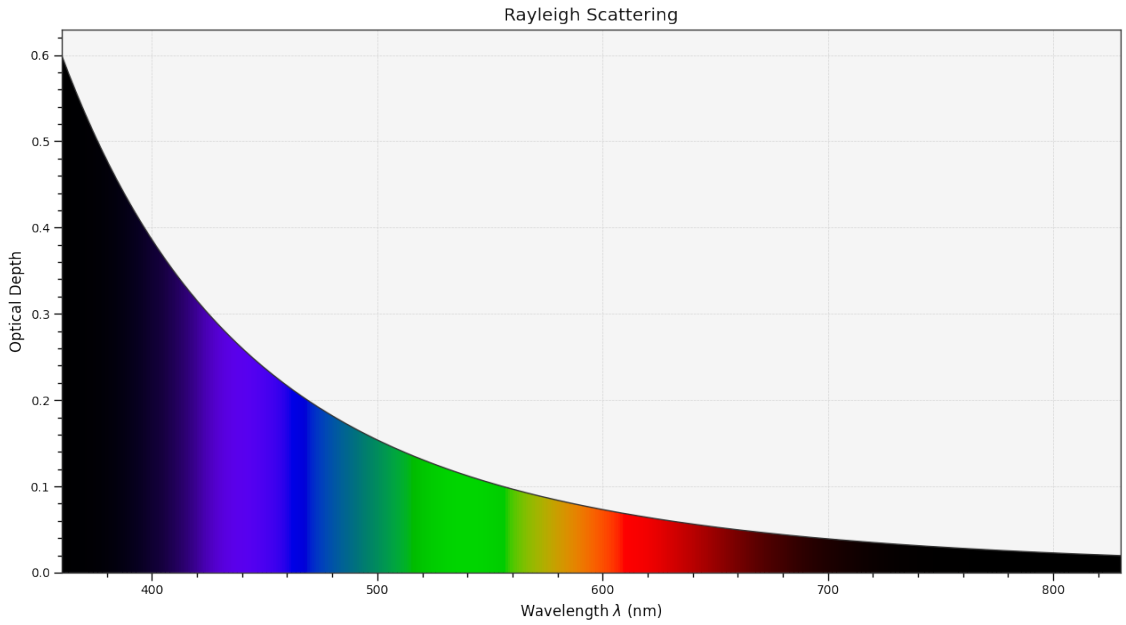
- **altitude** (numeric, optional) – Altitude of the site in meters.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- ****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()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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 or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectrum domain and colours. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.

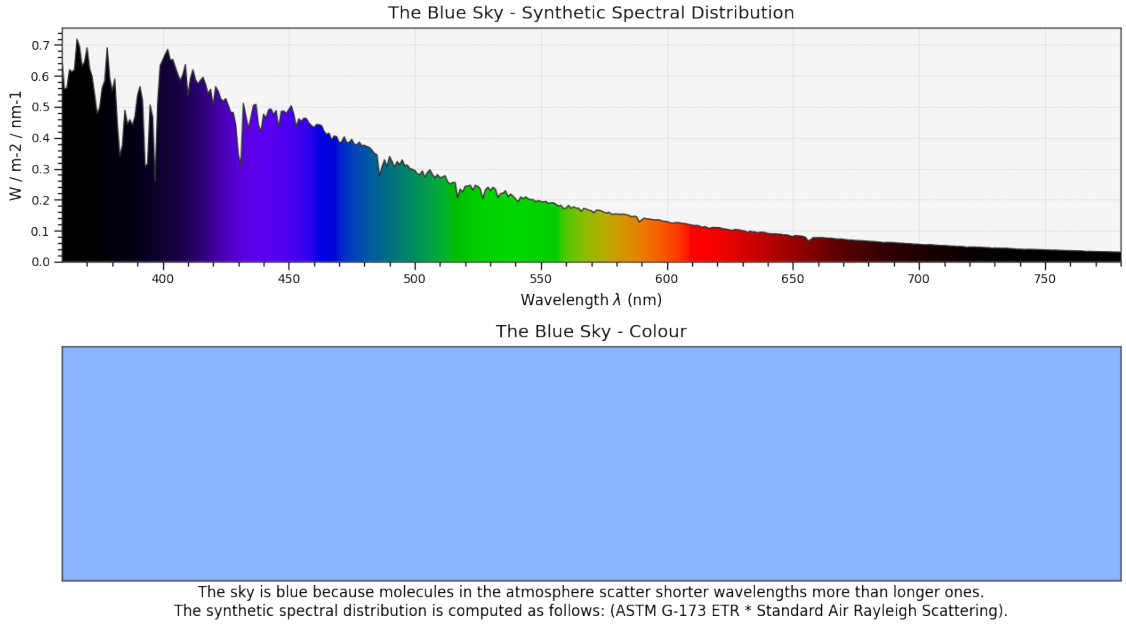
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_the_blue_sky()
(<Figure size ... with 2 Axes>, <...AxesSubplot...>)
```



Colour Quality

colour.plotting

plot_single_sd_colour_rendering_index_bars(sd, ...)	Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_rendering_indexes_bars(...)	Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.
plot_single_sd_colour_quality_scale_bars(sd)	Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_quality_scales_bars(sds)	Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

colour.plotting.plot_single_sd_colour_rendering_index_bars

colour.plotting.**plot_single_sd_colour_rendering_index_bars**(*sd*, ***kwargs*)

Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.

Parameters

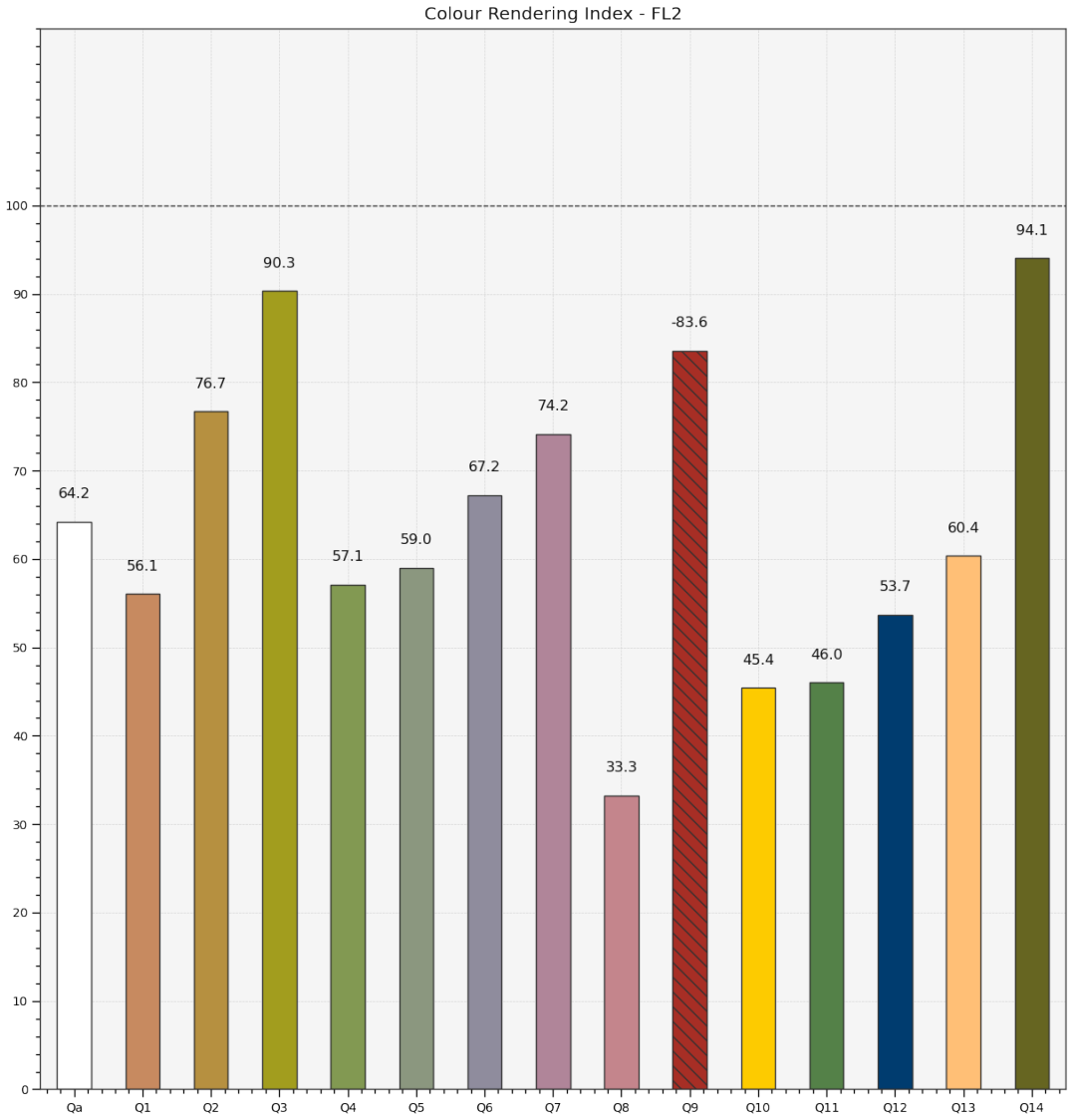
- **sd** (SpectralDistribution) – Illuminant or light source spectral distribution to plot the *Colour Rendering Index* (CRI).
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation 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 SDS_ILLUMINANTS
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_rendering_index_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



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 or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- ****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.

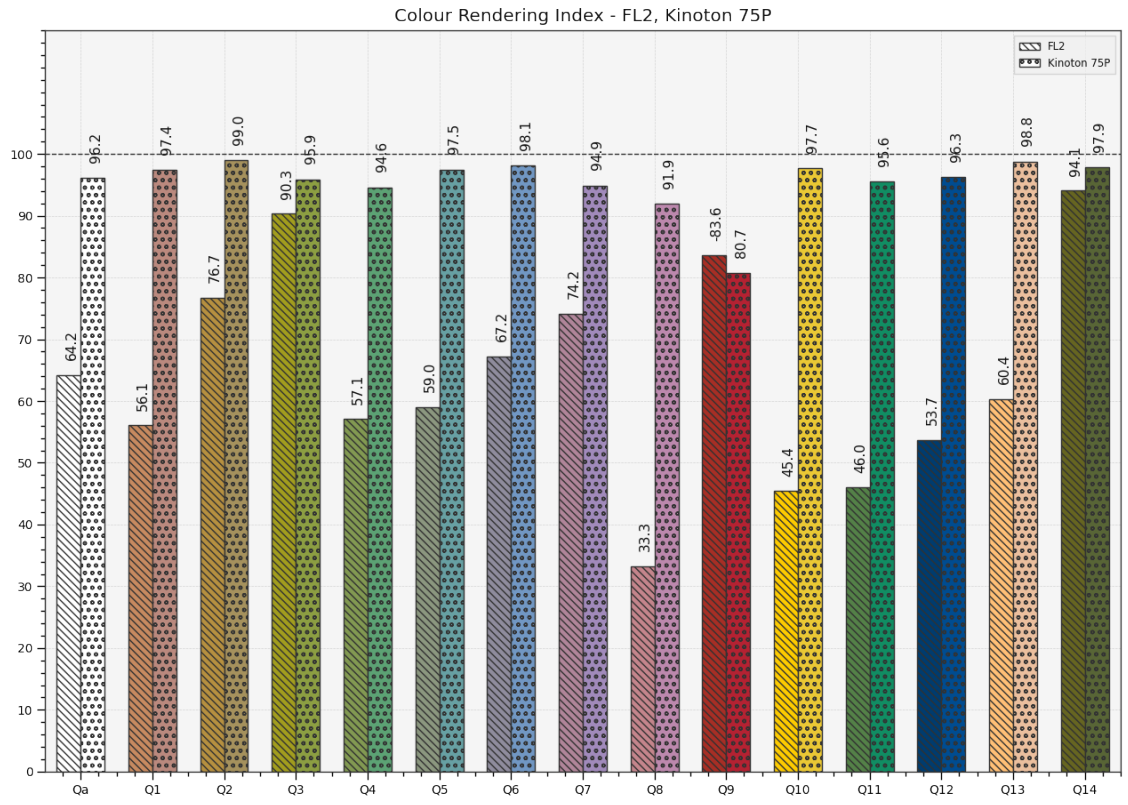
- **hatching_repeat** (int, optional) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import (SDS_ILLUMINANTS,
...                     SDS_LIGHT_SOURCES)
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> light_source = SDS_LIGHT_SOURCES['Kinoton 75P']
>>> plot_multi_sds_colour_rendering_indexes_bars(
...     [illuminant, light_source])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_single_sd_colour_quality_scale_bars

colour.plotting.**plot_single_sd_colour_quality_scale_bars**(*sd*, *method='NIST CQS 9.0'*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.

Parameters

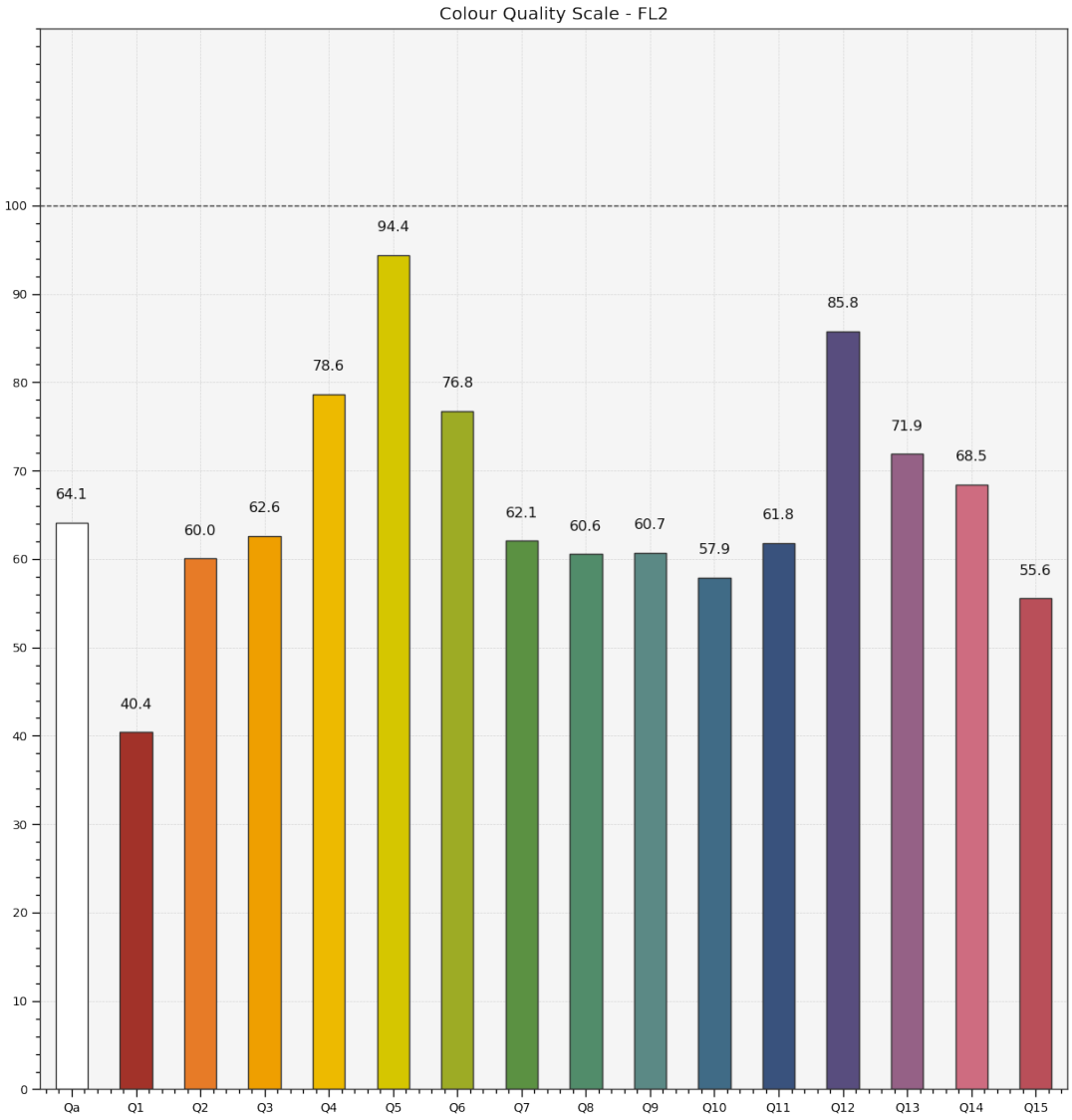
- **sd** (SpectralDistribution) – Illuminant or light source spectral distribution to plot the *Colour Quality Scale* (CQS).
- **method** (unicode, optional) – **{'NIST CQS 7.4', 'NIST CQS 9.0'}**, *Colour Quality Scale* (CQS) computation method.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), 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 SDS_ILLUMINANTS
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_quality_scale_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_multi_sds_colour_quality_scales_bars

colour.plotting.**plot_multi_sds_colour_quality_scales_bars**(*sds*, *method='NIST CQS 9.0'*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **method** (unicode, optional) – **{'NIST CQS 7.4'}**, *Colour Quality Scale* (CQS) computation method.
- ****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.

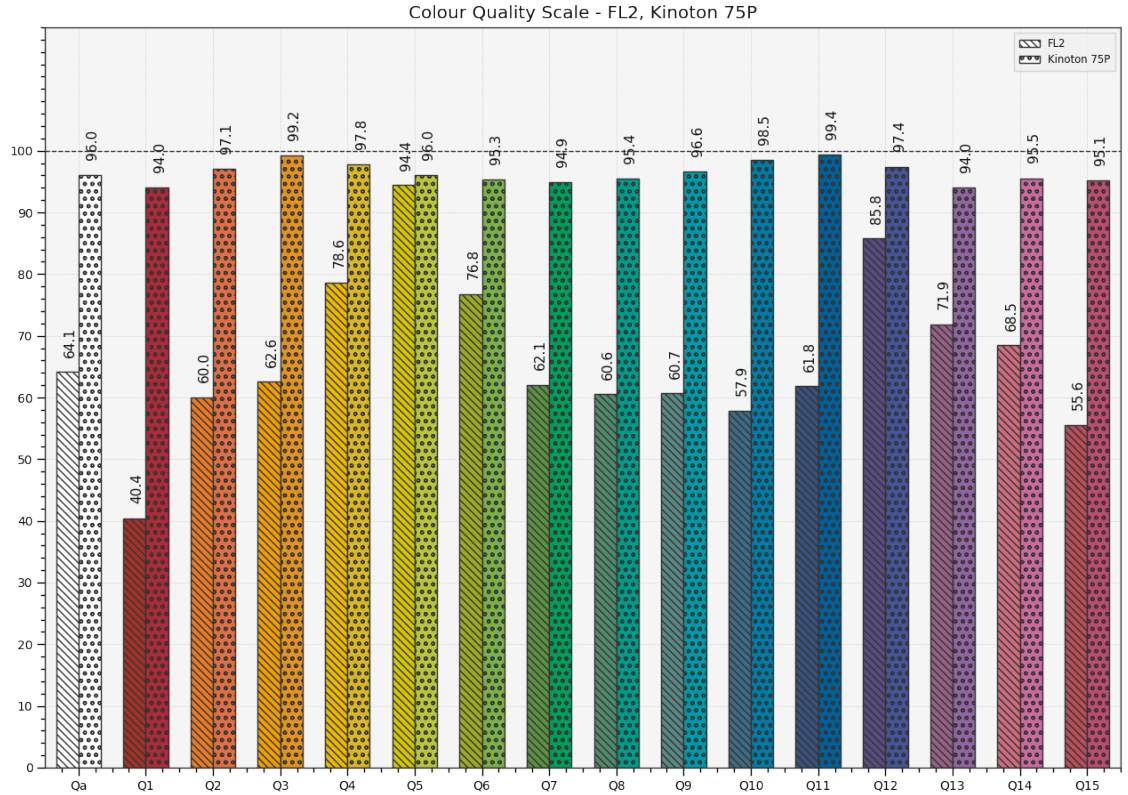
- **hatching** (bool or None, optional) – {colour.plotting.quality.plot_colour_quality_bars()}, Use hatching for the bars.
- **hatching_repeat** (int, optional) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import (SDS_ILLUMINANTS,
...                     SDS_LIGHT_SOURCES)
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> light_source = SDS_LIGHT_SOURCES['Kinoton 75P']
>>> plot_multi_sds_colour_quality_scales_bars([illuminant, light_source])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Ancillary Objects

colour.plotting.quality

plot_colour_quality_bars(specifications[, ...])	Plots the colour quality data of given illuminants or light sources colour quality specifications.

colour.plotting.quality.plot_colour_quality_bars

colour.plotting.quality.**plot_colour_quality_bars**(*specifications*, *labels=True*, *hatching=None*, *hatching_repeat=2*, ***kwargs*)

Plots the colour quality data of given illuminants or light sources colour quality specifications.

Parameters

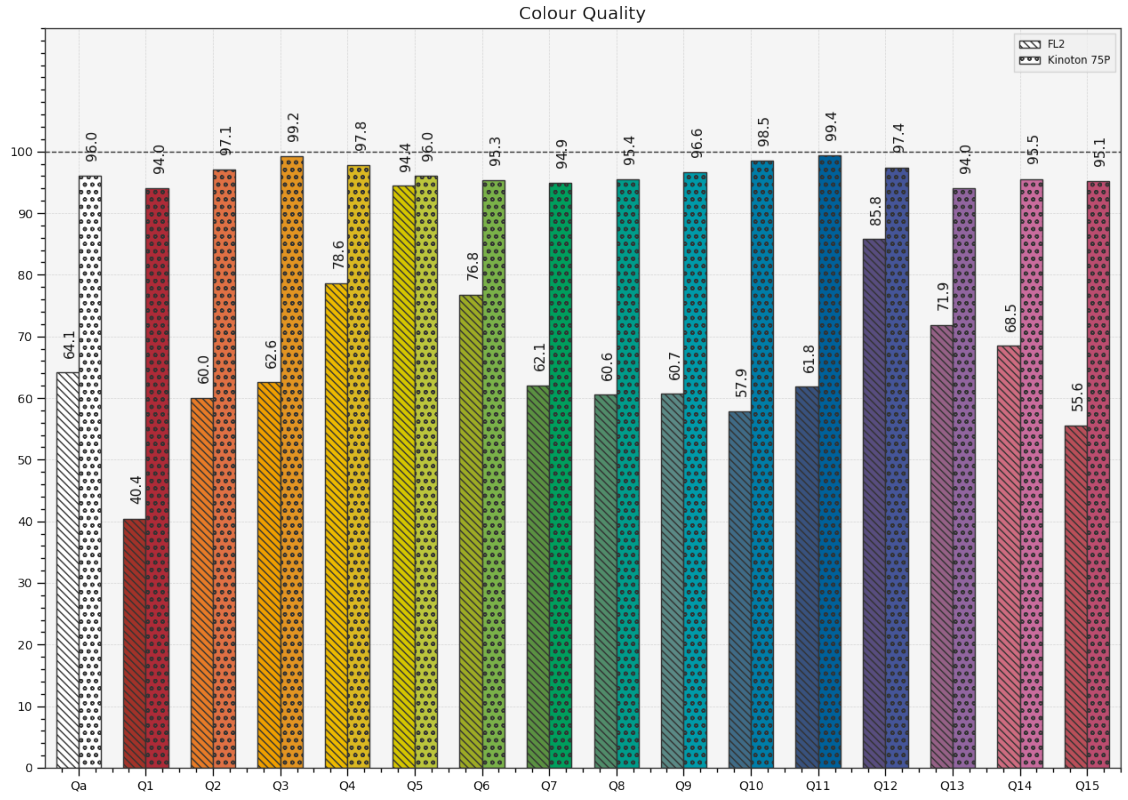
- **specifications** (array_like) – Array of illuminants or light sources colour quality specifications.
- **labels** (bool, optional) – Add labels above bars.
- **hatching** (bool or None, optional) – Use hatching for the bars.
- **hatching_repeat** (int, optional) – Hatching pattern repeat.
- ****kwargs** (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 (SDS_ILLUMINANTS,
...                     SDS_LIGHT_SOURCES, SpectralShape)
>>> illuminant = SDS_ILLUMINANTS['FL2']
>>> light_source = SDS_LIGHT_SOURCES['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])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Colour Temperature & Correlated Colour Temperature

`colour.plotting`

`plot_planckian_locus_in_chromaticity_diagram_CIE1931(...)`	Plots the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.
`plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(...)`	Plots 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*, *chromaticity_diagram_callable_CIE1931=<function plot_chromaticity_diagram_CIE1931>*, *planckian_locus_callable_CIE1931=<function plot_planckian_locus_CIE1931>*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.

Parameters

- **illuminants** (unicode or object or array_like) – Illuminants to plot. illuminants elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **planckian_locus_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *Planckian Locus* according to *CIE 1931* method.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names. annotate_kwargs 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. The following special keyword arguments can also be used:

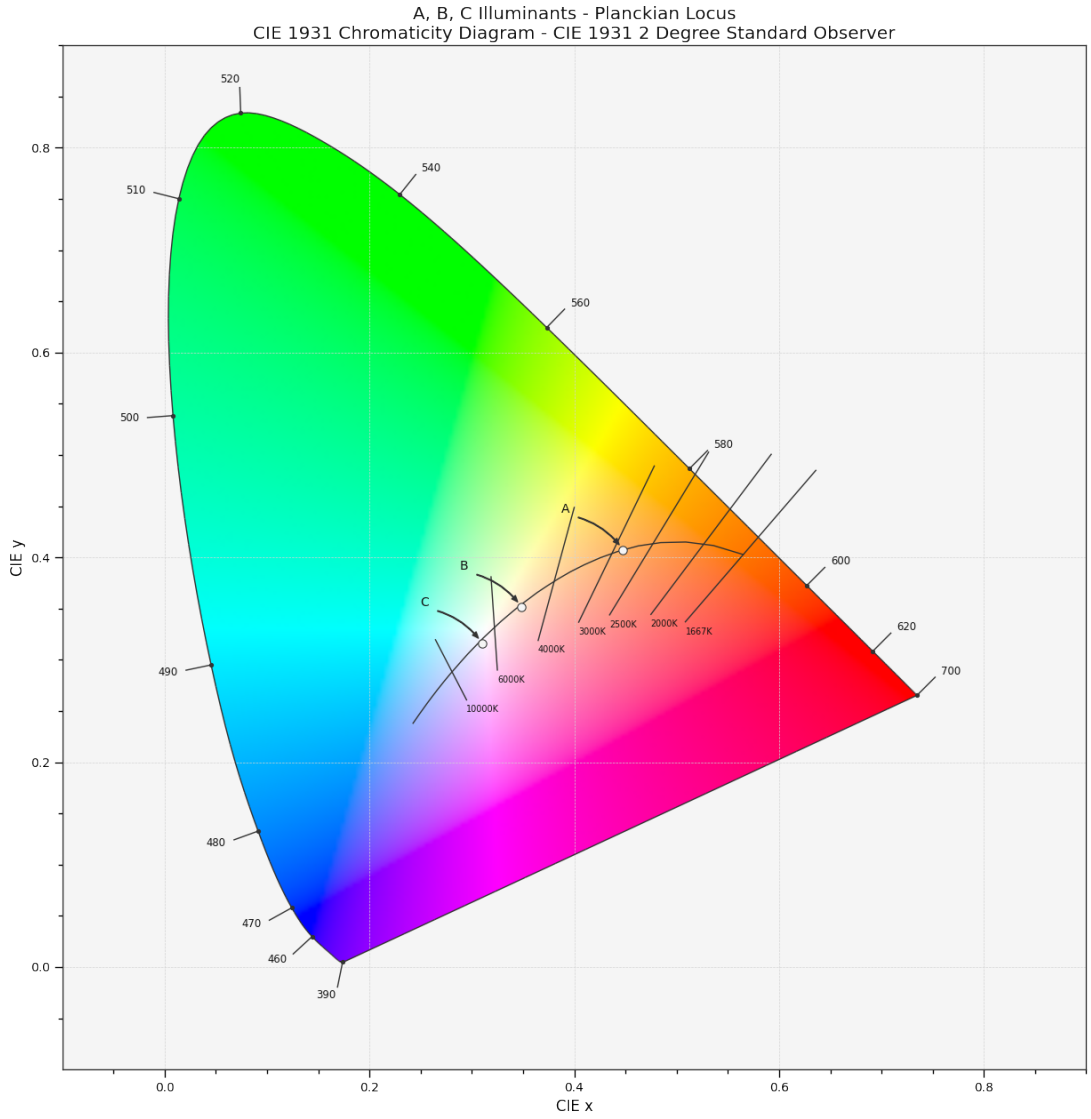
  – *annotate* : bool, whether to annotate the illuminants.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted illuminants. plot_kwargs can be either a single dictionary applied to all the plotted illuminants with same settings or a sequence of dictionaries with different settings for each plotted illuminant.
- ****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. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1931(['A', 'B', 'C'])
...
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS**(*illuminants*, *chromaticity_diagram_callable_CIE1960UCS=<*, *plot_chromaticity_diagram_CIE1960UC*, *planckian_locus_callable_CIE1960UCS=<fun*, *plot_planckian_locus_CIE1960UCS>*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **illuminants** (unicode or object or array_like) – Illuminants to plot. `illuminants` elements can be of any type or form supported by the `colour.plotting.filter_passthrough()` definition.

- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **planckian_locus_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *Planckian Locus* according to *CIE 1960 UCS* method.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names. `annotate_kwargs` 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. The following special keyword arguments can also be used:

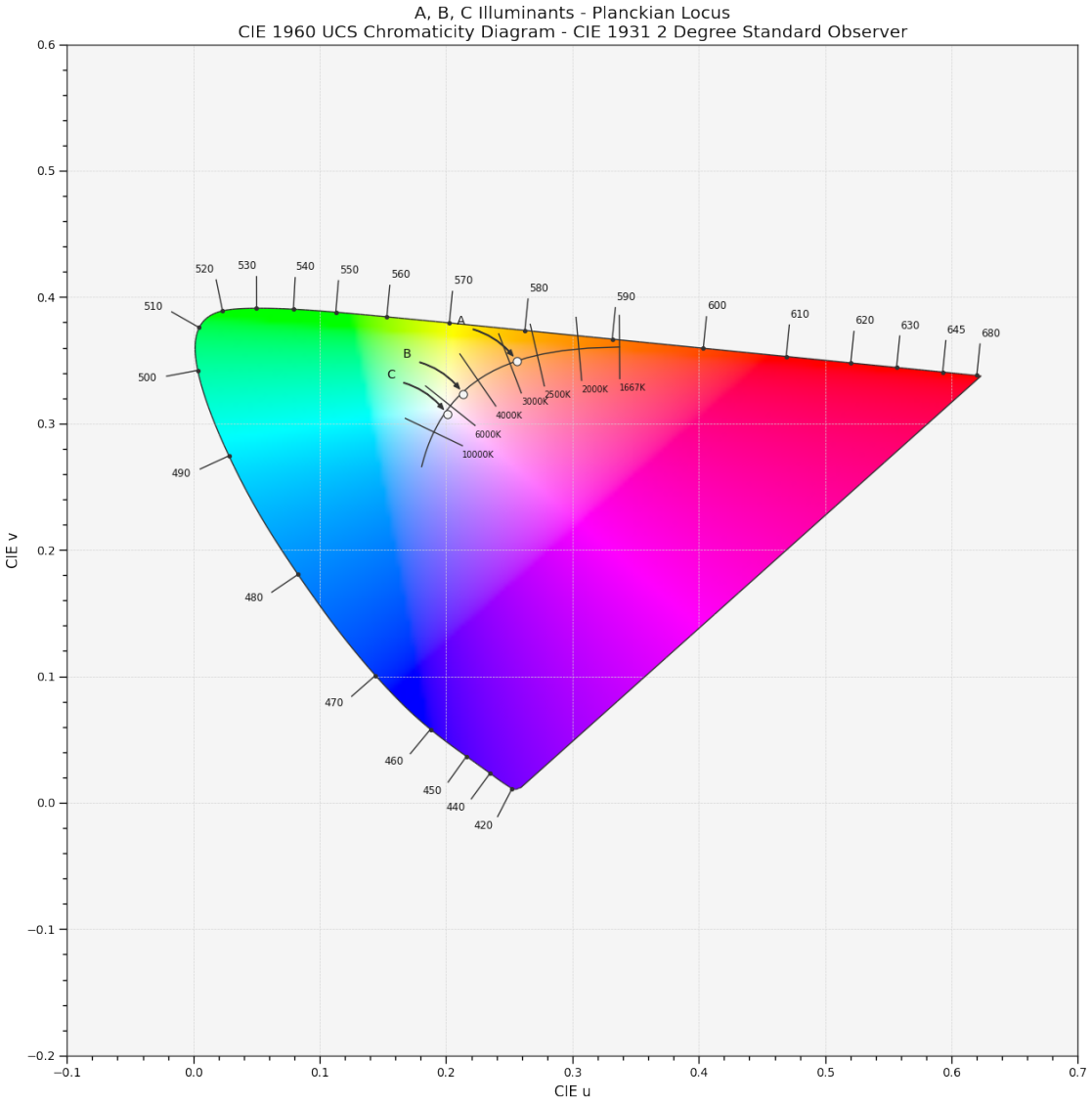
  – *annotate* : bool, whether to annotate the illuminants.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the `plt.plot()` definition, used to control the style of the plotted illuminants. `plot_kwargs` can be either a single dictionary applied to all the plotted illuminants with same settings or a sequence of dictionaries with different settings for each plotted illuminant.
- ****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. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(
...     ['A', 'C', 'E'])
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



Ancillary Objects

`colour.plotting.temperature`

`plot_planckian_locus`([...])	Plots the *Planckian Locus* according to given method.
`plot_planckian_locus_CIE1931`([...])	Plots the *Planckian Locus* according to *CIE 1931* method.
`plot_planckian_locus_CIE1960UCS`([...])	Plots the *Planckian Locus* according to *CIE 1960 UCS* 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

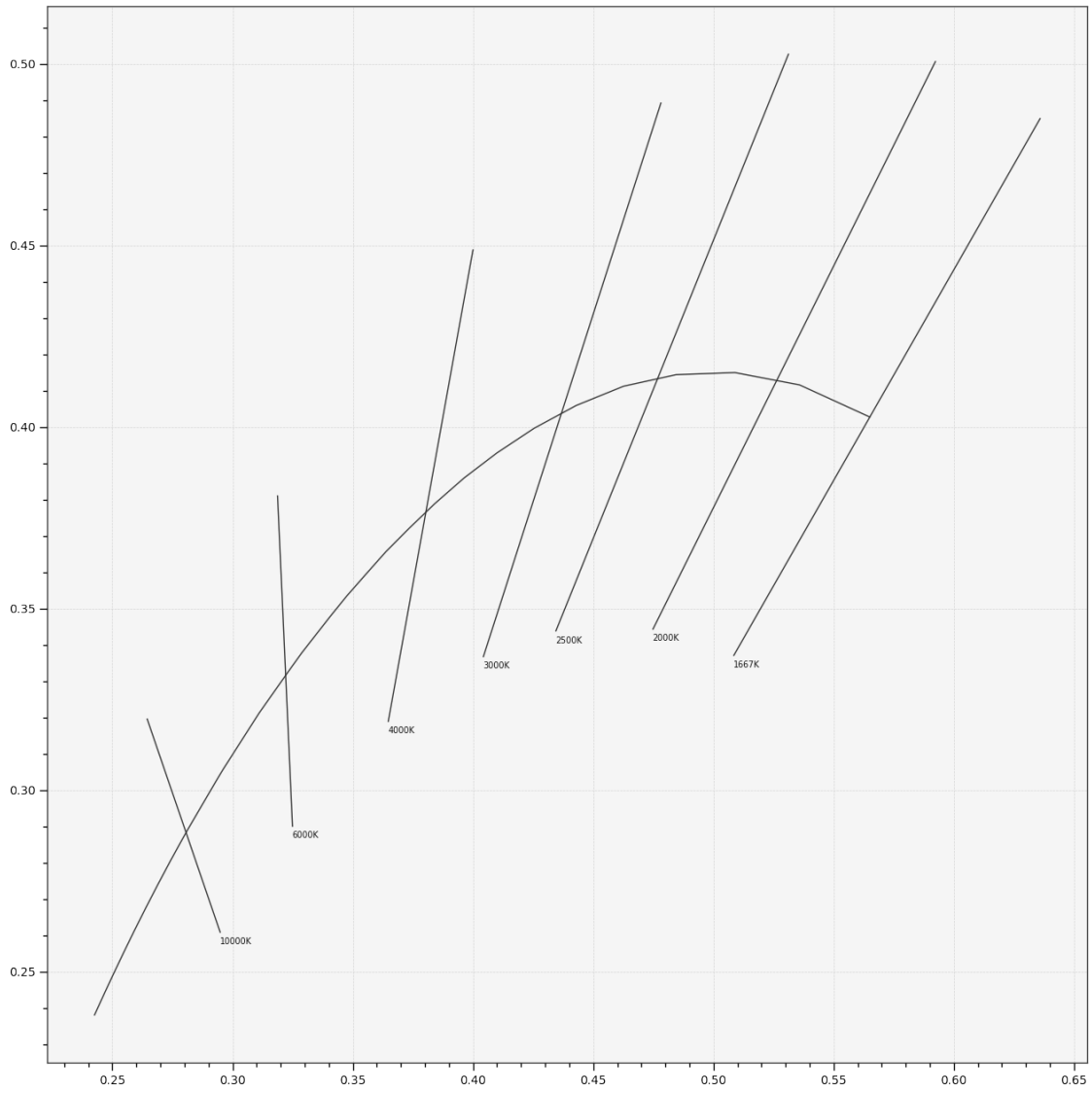
- **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.temperature.plot_planckian_locus_CIE1931

colour.plotting.temperature.**plot_planckian_locus_CIE1931**(*planckian_locus_colours=None*, ***kwargs*)

Plots the *Planckian Locus* according to *CIE 1931* method.

Parameters

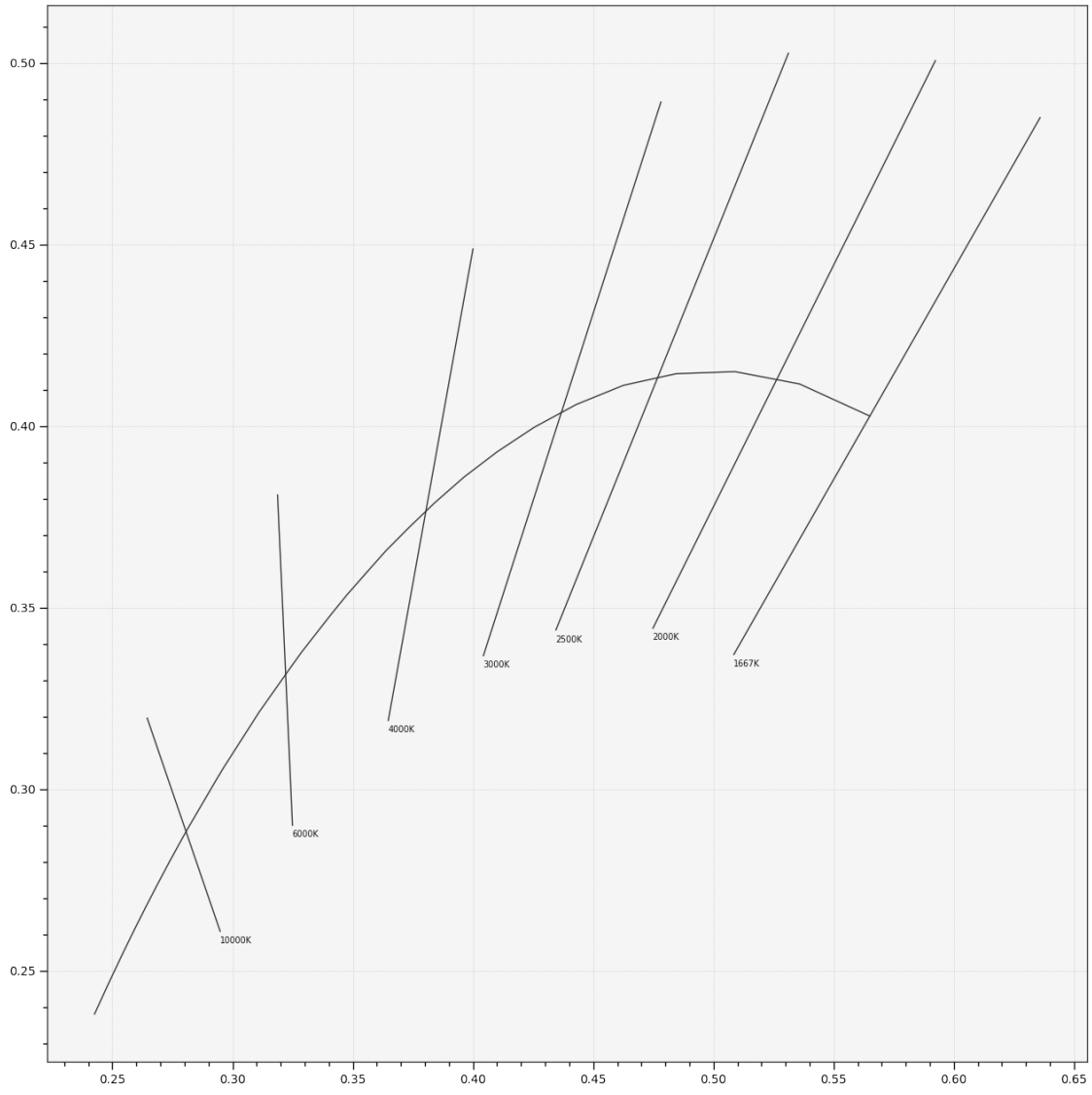
- **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_CIE1931()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.temperature.plot_planckian_locus_CIE1960UCS

colour.plotting.temperature.**plot_planckian_locus_CIE1960UCS**(*planckian_locus_colours=None*, ***kwargs*)

Plots the *Planckian Locus* according to *CIE 1960 UCS* method.

Parameters

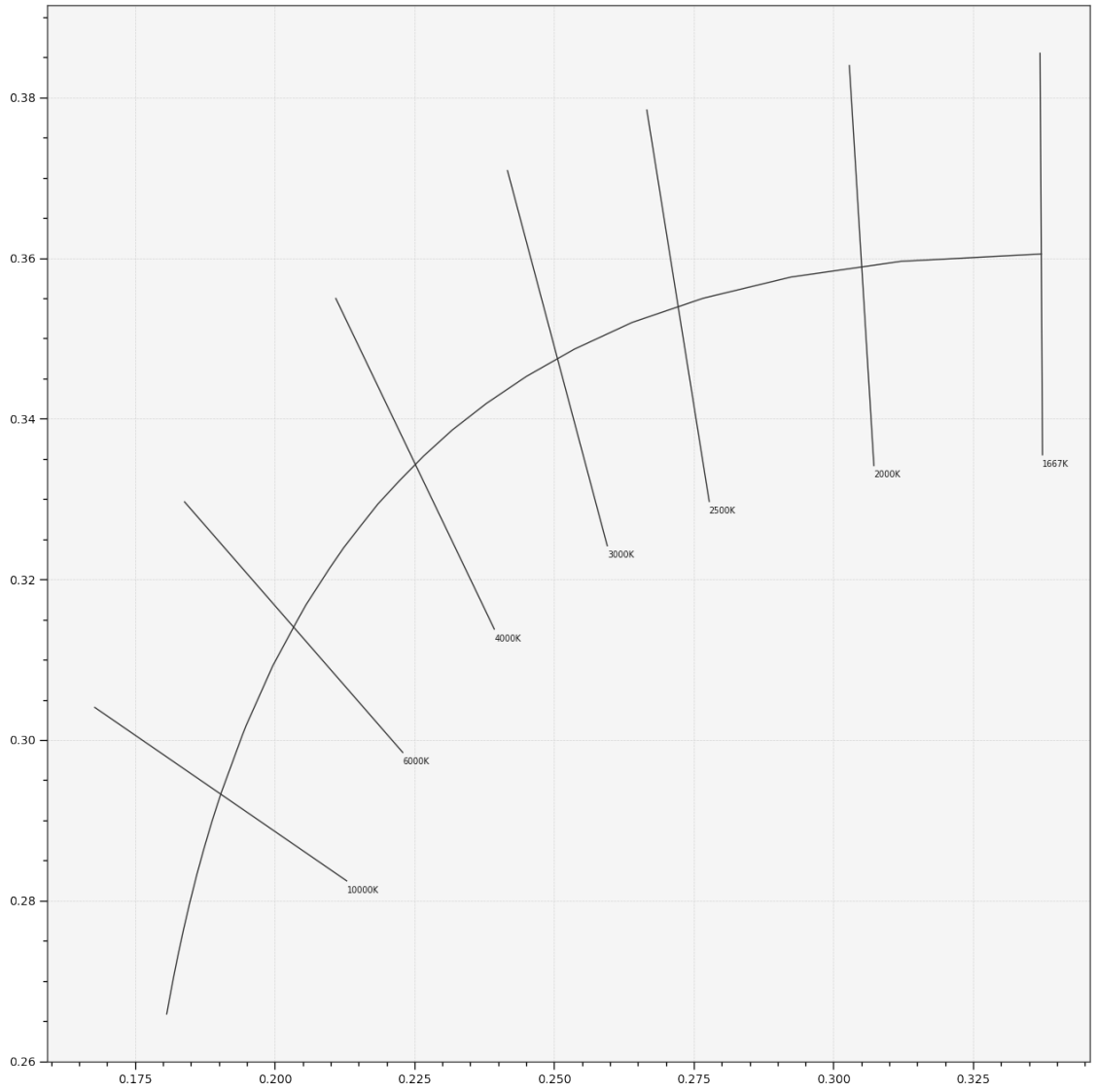
- **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_CIE1960UCS()
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram

colour.plotting.temperature.**plot_planckian_locus_in_chromaticity_diagram**(*illuminants*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *planckian_locus_callable=<function plot_planckian_locus>*, *method='CIE 1931'*, *annotate_kwargs=None*, *plot_kwargs=None*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given

method.

Parameters

- **illuminants** (unicode or object or array_like) – Illuminants to plot. illuminants elements can be of any type or form supported by the colour.plotting.filter_passthrough() definition.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **planckian_locus_callable** (callable, optional) – Callable responsible for drawing the *Planckian Locus*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- **annotate_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names. annotate_kwargs 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. The following special keyword arguments can also be used:

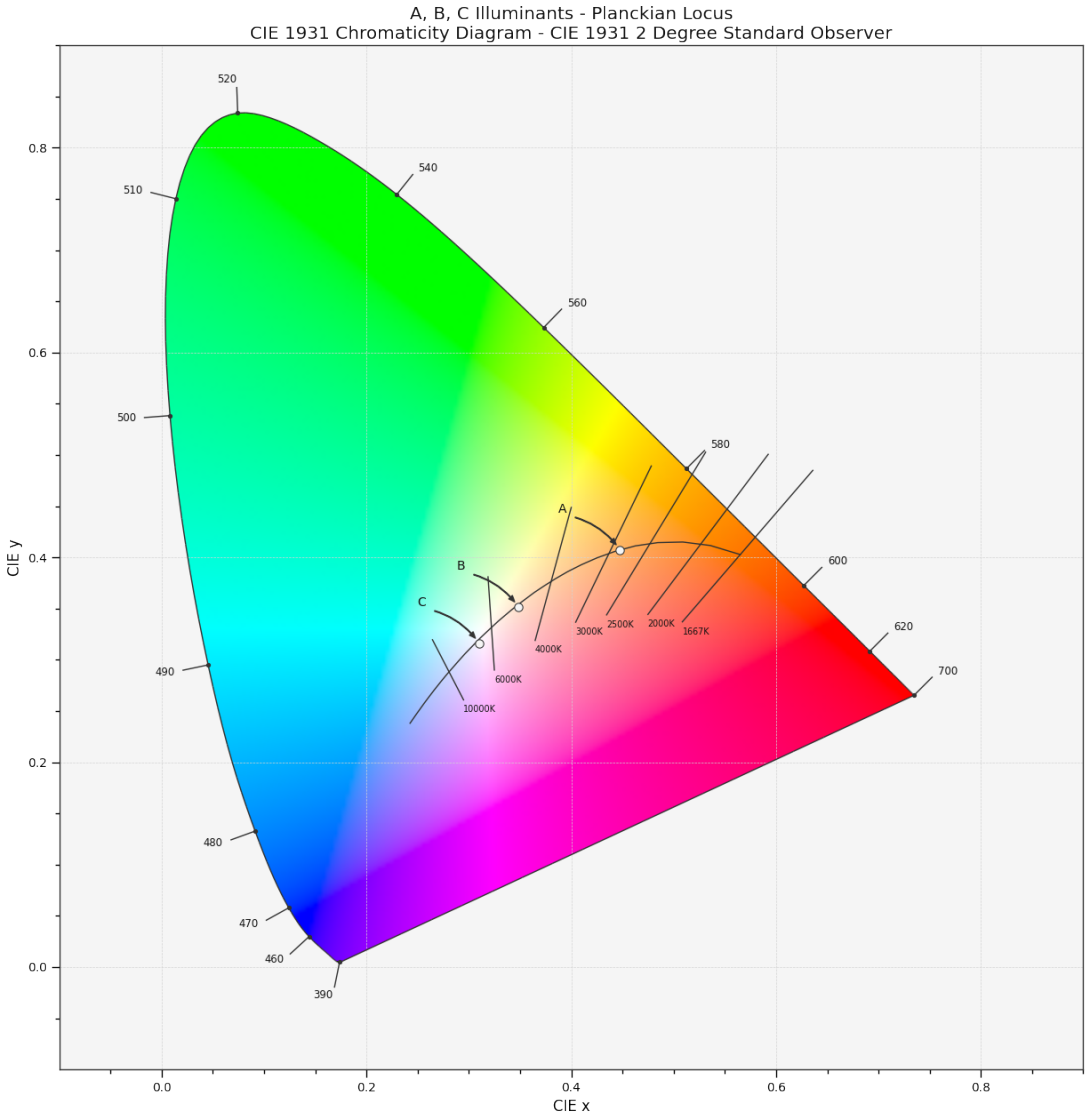
  – *annotate* : bool, whether to annotate the illuminants.
- **plot_kwargs** (dict or array_like, optional) – Keyword arguments for the plt.plot() definition, used to control the style of the plotted illuminants. plot_kwargs can be either a single dictionary applied to all the plotted illuminants with same settings or a sequence of dictionaries with different settings for each plotted illuminant.
- ****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. Also handles keywords arguments for deprecation management.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> annotate_kwargs = [
...     {'xytext': (-25, 15), 'arrowprops':{'arrowstyle':'-'}},
...     {'arrowprops':{'arrowstyle':'-['}},
...     {},
... ]
>>> plot_kwargs = [
...     {
...         'markersize' : 15,
...     },
...     {   'color': 'r'},
...     {},
... ]
>>> plot_planckian_locus_in_chromaticity_diagram(
...     ['A', 'B', 'C'],
...     annotate_kwargs=annotate_kwargs,
...     plot_kwargs=plot_kwargs
... )
(<Figure size ... with 1 Axes>, <...AxesSubplot...>)
```



ANSI/IES TM-30-18 Colour Rendition Report

`colour.plotting`

`plot_single_sd_colour_rendition_report`(sd[, ...])	Generates the *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution according to given method.

colour.plotting.plot_single_sd_colour_rendition_report

`colour.plotting.`**`plot_single_sd_colour_rendition_report`**(*sd*, *method='Full'*, ***kwargs*)

Generates the *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution according to given method.

Parameters

- **sd** (`SpectralDistribution` or `SpectralDistribution_IESTM2714`) – Spectral distribution of the emission source to generate the report for.

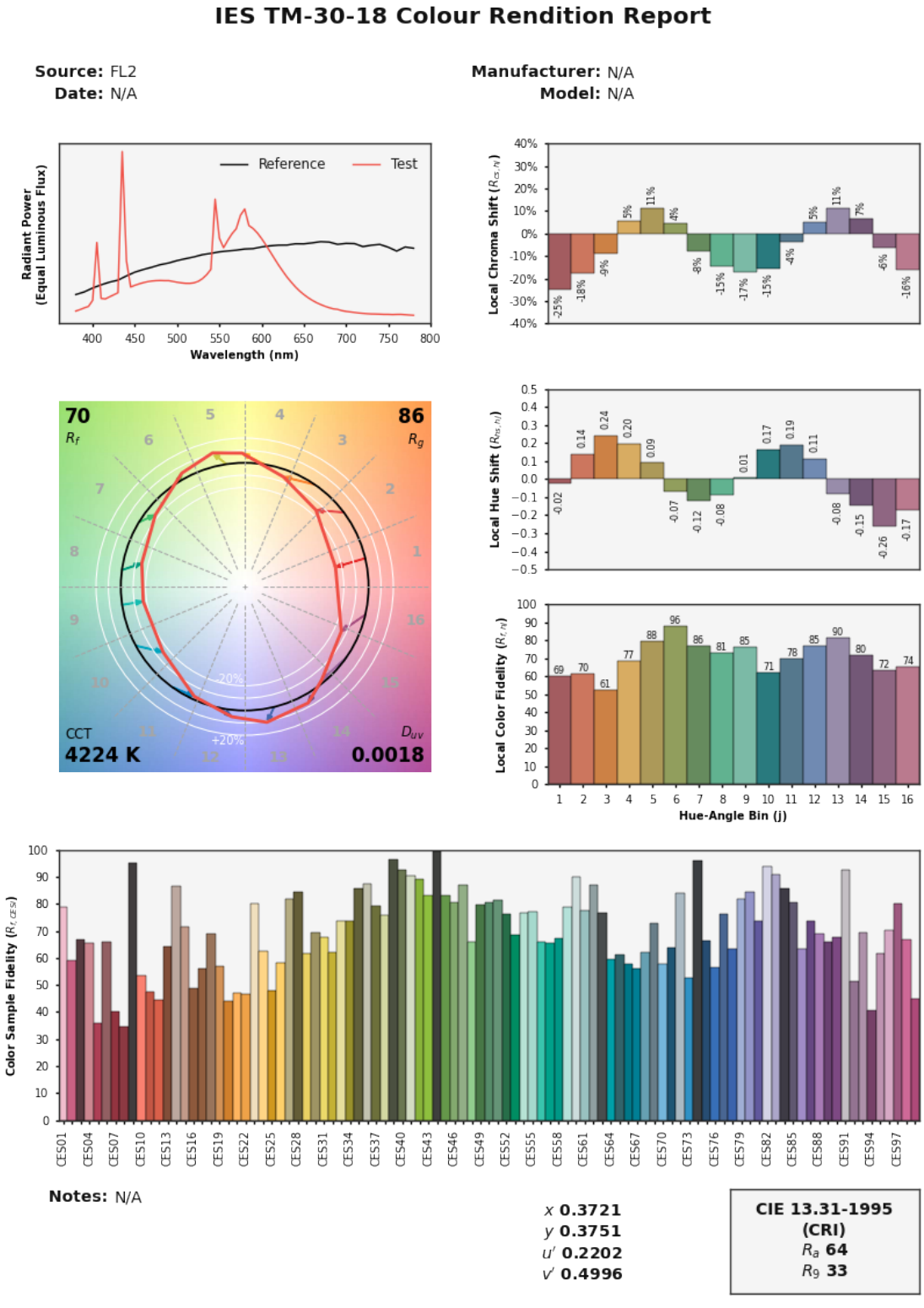
- **method** (unicode, optional) – **{'Full', 'Intermediate', 'Simple'}**, Report plotting method.
- ****kwargs** (dict, optional) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **source** (unicode, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full()}, Emission source name, defaults to *colour.SpectralDistribution_IESTM2714.header.description* or *colour.SpectralDistribution_IESTM2714.name* properties value.
- **date** (unicode, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full()}, Emission source measurement date, defaults to *colour.SpectralDistribution_IESTM2714.header.report_date* property value.
- **manufacturer** (unicode, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full()}, Emission source manufacturer, defaults to *colour.SpectralDistribution_IESTM2714.header.manufacturer* property value.
- **model** (unicode, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full()}, Emission source model, defaults to *colour.SpectralDistribution_IESTM2714.header.catalog_number* property value.
- **notes** (unicode, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full()}, Notes pertaining to the emission source, defaults to *colour.SpectralDistribution_IESTM2714.header.comments* property value.
- **report_size** (array_like, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate(), :func:`colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple}, Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (array_like, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate(), :func:`colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple}, Report size row height ratios.
- **report_box_padding** (array_like, optional) – {colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full(), colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate(), :func:`colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple}, Report box padding, tries to define the padding around the figure and in-between the axes.

Returns Current figure and axes.

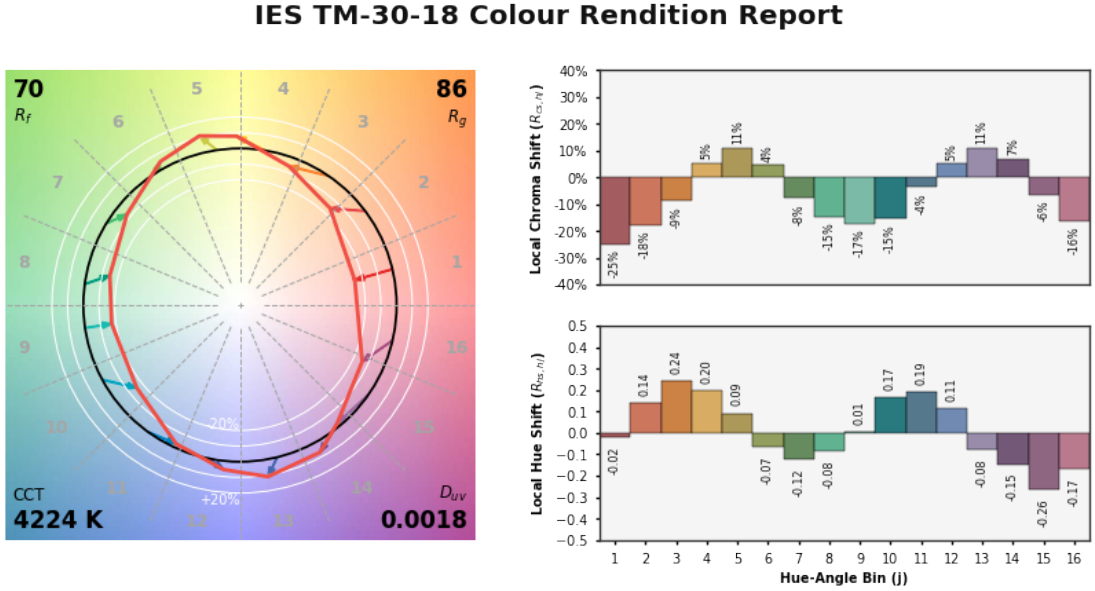
Return type tuple

Examples

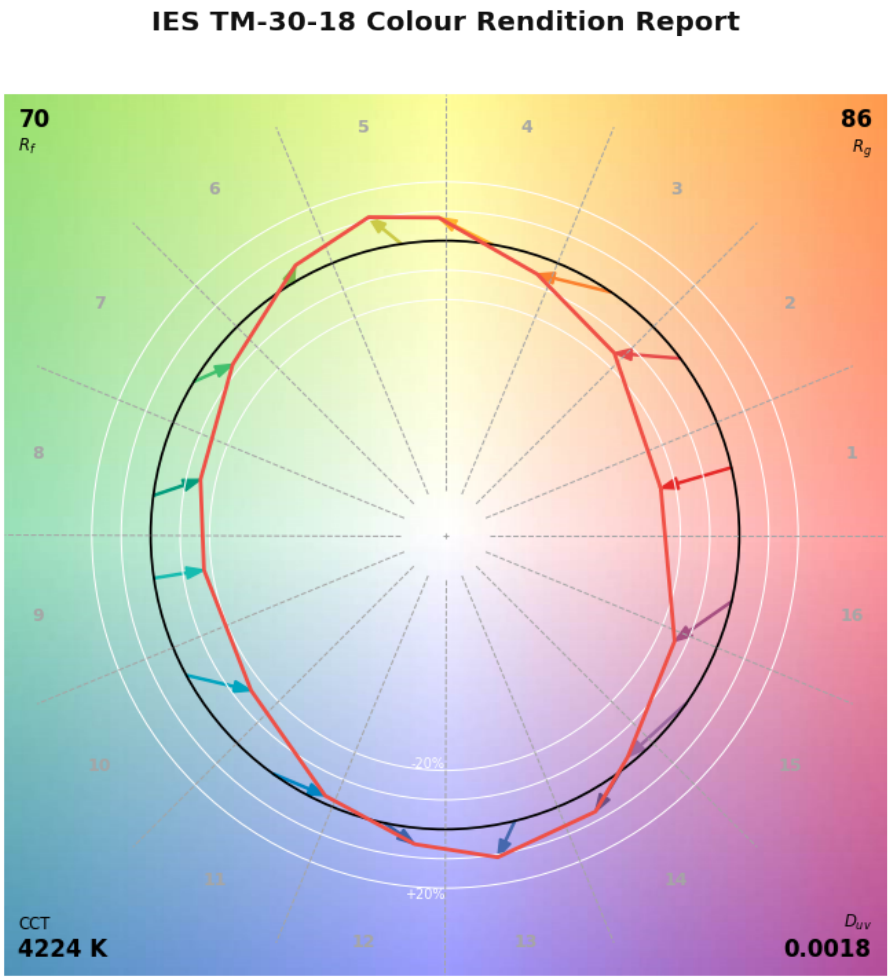
```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_rendition_report(sd)
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```




```
>>> plot_single_sd_colour_rendition_report(sd, 'Intermediate')
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```




```
>>> plot_single_sd_colour_rendition_report(sd, 'Simple')
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```



Ancillary Objects

`colour.plotting.tm3018`

`plot_single_sd_colour_rendition_report_full(sd)`	Generates the full *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.
`plot_single_sd_colour_rendition_report_intermediate(sd)`	Generates the intermediate *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.
`plot_single_sd_colour_rendition_report_simple(sd)`	Generates the simple *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

colour.plotting.tm3018.plot_single_sd_colour_rendition_report_full

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_full**(*sd*, *source=None*, *date=None*, *manufacturer=None*, *model=None*, *notes=None*, *report_size=(8.27, 11.69)*, *report_row_height_ratios=(1, 2, 24, 3, 1)*, *report_box_padding=None*, ***kwargs*)

Generates the full *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

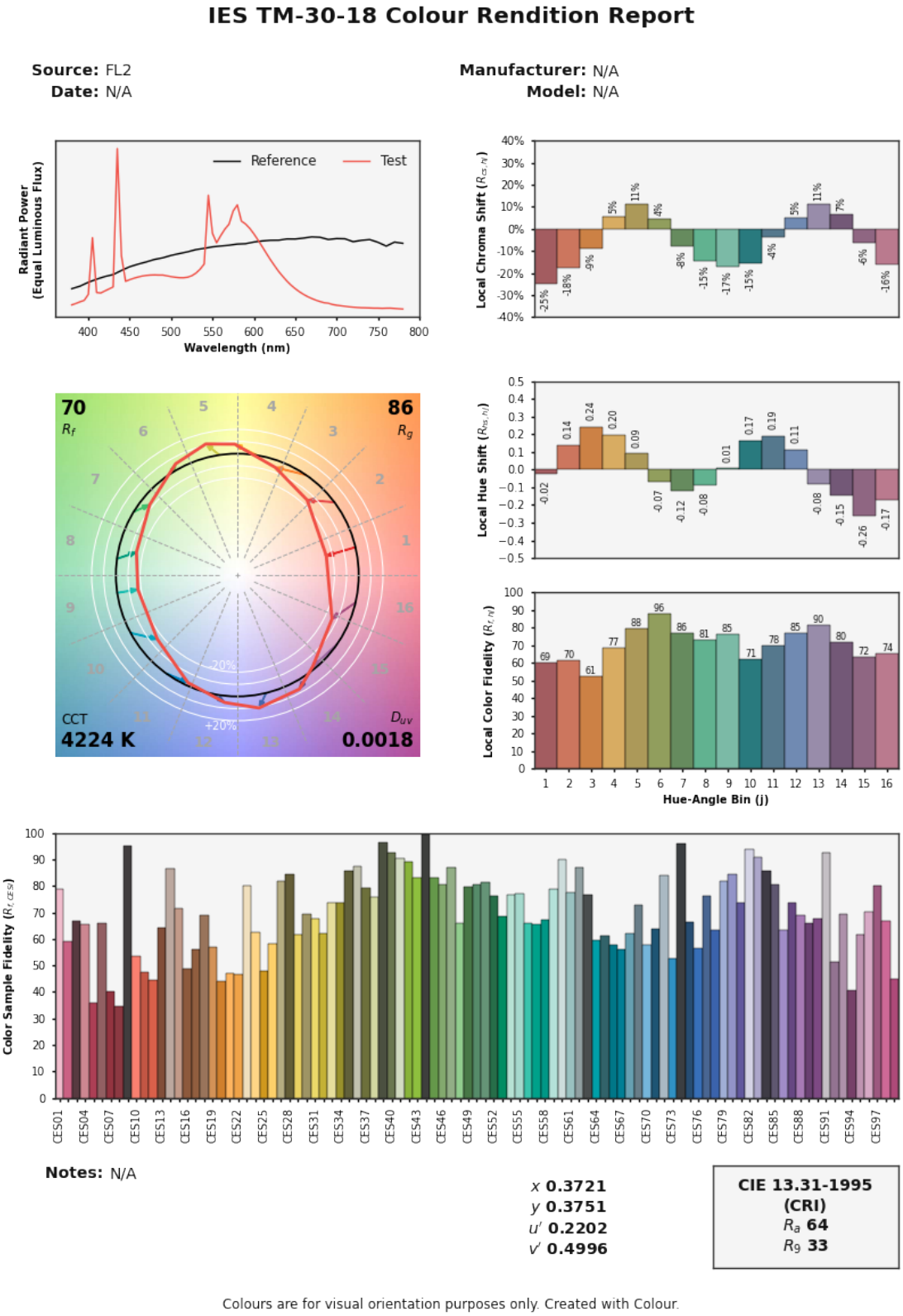
- **sd** (SpectralDistribution or SpectralDistribution_IESTM2714) – Spectral distribution of the emission source to generate the report for.
- **source** (unicode, optional) – Emission source name, defaults to *colour.SpectralDistribution_IESTM2714.header.description* or *colour.SpectralDistribution_IESTM2714.name* properties value.
- **date** (unicode, optional) – Emission source measurement date, defaults to *colour.SpectralDistribution_IESTM2714.header.report_date* property value.
- **manufacturer** (unicode, optional) – Emission source manufacturer, defaults to *colour.SpectralDistribution_IESTM2714.header.manufacturer* property value.
- **model** (unicode, optional) – Emission source model, defaults to *colour.SpectralDistribution_IESTM2714.header.catalog_number* property value.
- **notes** (unicode, optional) – Notes pertaining to the emission source, defaults to *colour.SpectralDistribution_IESTM2714.header.comments* property value.
- **report_size** (array_like, optional) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (array_like, optional) – Report size row height ratios.
- **report_box_padding** (array_like, optional) – Report box padding, tries to define the padding around the figure and in-between the axes.
- ****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 SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_rendition_report_full(sd)
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```


IES TM-30-18 Colour Rendition Report
Source: FL2
Date: N/A
Manufacturer: N/A
Model: N/A
Radiant Power (Equal Luminous Flux)
Reference
Test
Wavelength (nm)
Local Chroma Shift ($R_{cs,hj}$)
Local Hue Shift ($R_{hs,hj}$)
Local Color Fidelity ($R_{f,hj}$)
Hue-Angle Bin (j)
70
R_f
86
R_g
CCT
4224 K
D_{uv}
0.0018
Color Sample Fidelity ($R_{f,cesi}$)
Notes: N/A
x 0.3721
y 0.3751
u′ 0.2202
v′ 0.4996
CIE 13.31-1995
(CRI)
R_a 64
R_9 33
Colours are for visual orientation purposes only. Created with Colour.

colour.plotting.tm3018.plot_single_sd_colour_rendition_report_intermediate

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_intermediate**(*sd*, *report_size=(8.27, 4.974468085106382)*, *report_row_height_ratios=(1, 8, 1)*, *report_box_padding=None*, ***kwargs*)

Generates the intermediate *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

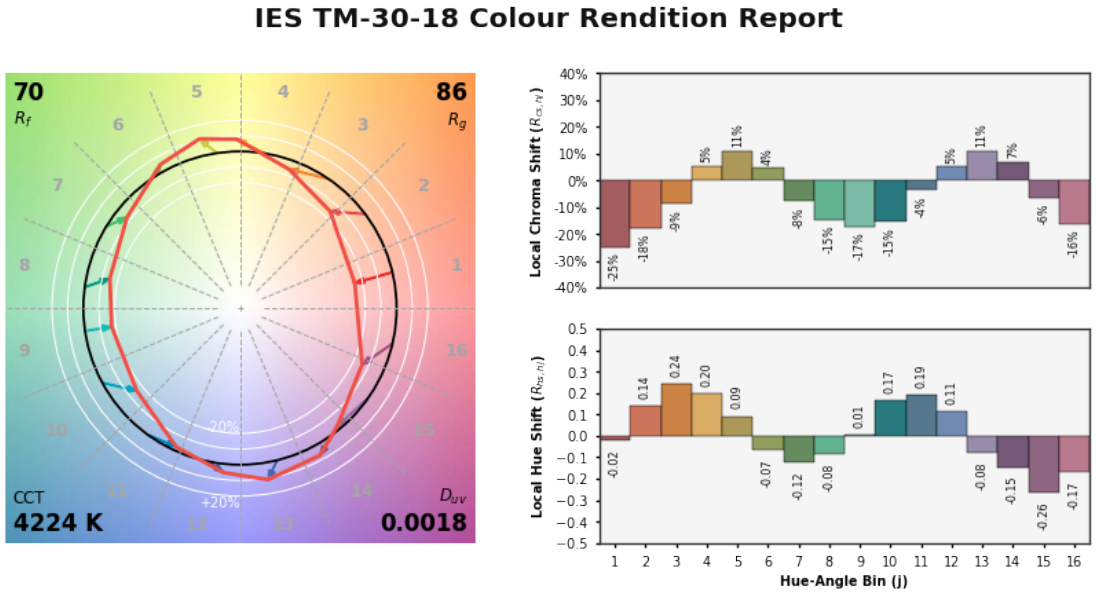
- **sd** (SpectralDistribution or SpectralDistribution_IESTM2714) – Spectral distribution of the emission source to generate the report for.
- **report_size** (array_like, optional) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (array_like, optional) – Report size row height ratios.
- **report_box_padding** (array_like, optional) – Report box padding, tries to define the padding around the figure and in-between the axes.
- ****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 SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_rendition_report_intermediate(sd)
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```



colour.plotting.tm3018.plot_single_sd_colour_rendition_report_simple

colour.plotting.tm3018.**plot_single_sd_colour_rendition_report_simple**(*sd*, *report_size=(8.27, 8.27)*, *report_row_height_ratios=(1, 8, 1)*, *report_box_padding=None*, ***kwargs*)

Generates the simple *ANSI/IES TM-30-18 Colour Rendition Report* for given spectral distribution.

Parameters

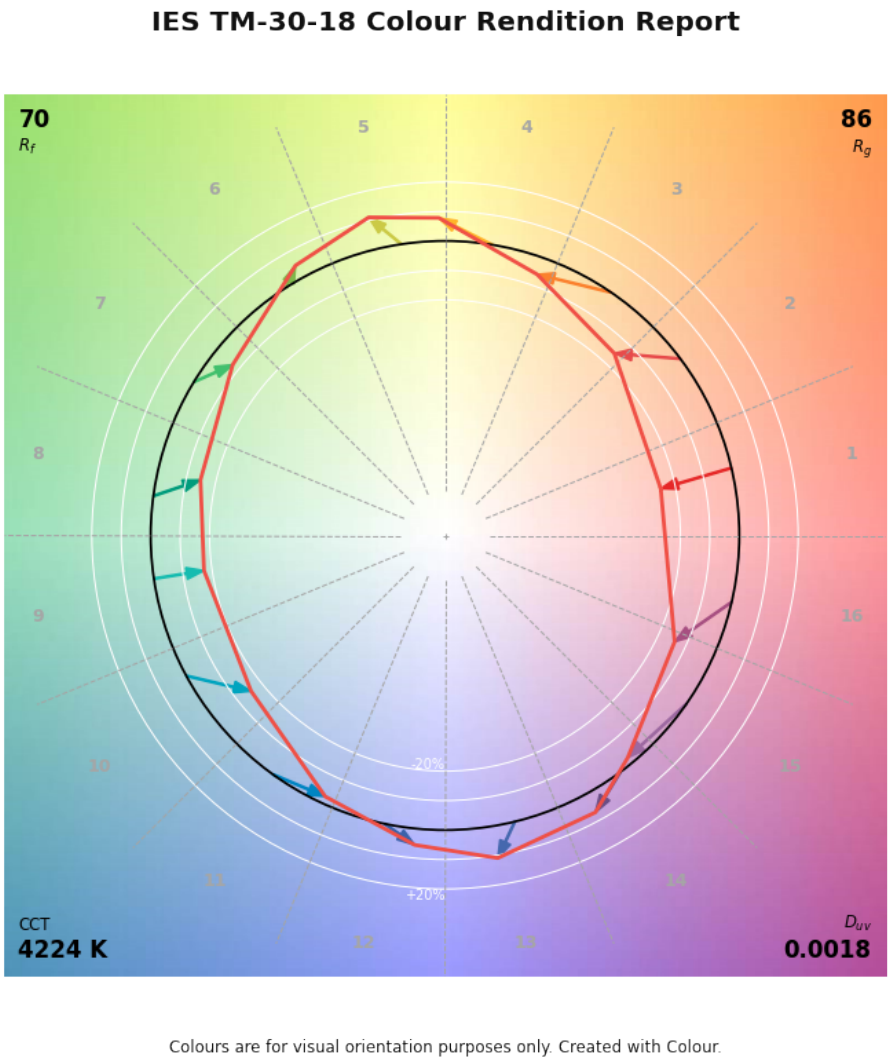
- **sd** (SpectralDistribution or SpectralDistribution_IESTM2714) – Spectral distribution of the emission source to generate the report for.
- **report_size** (array_like, optional) – Report size, default to A4 paper size in inches.
- **report_row_height_ratios** (array_like, optional) – Report size row height ratios.
- **report_box_padding** (array_like, optional) – Report box padding, tries to define the padding around the figure and in-between the axes.
- ****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 SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> plot_single_sd_colour_rendition_report_simple(sd)
...
(<Figure size ... with ... Axes>, <...AxesSubplot...>)
```



Colour Models Volume

`colour.plotting`

`plot_RGB_colourspaces_gamuts`(colourspaces[, ...])	Plots given *RGB* colourspaces gamuts in given reference colourspace.
`plot_RGB_scatter`(RGB, colourspace[, ...])	Plots given *RGB* colourspace array in a scatter plot.

colour.plotting.plot_RGB_colourspaces_gamuts

colour.plotting.**plot_RGB_colourspaces_gamuts**(*colourspaces*, *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'*, *chromatically_adapt=False*, ***kwargs*)

Plots given *RGB* colourspaces gamuts in given reference colourspace.

Parameters

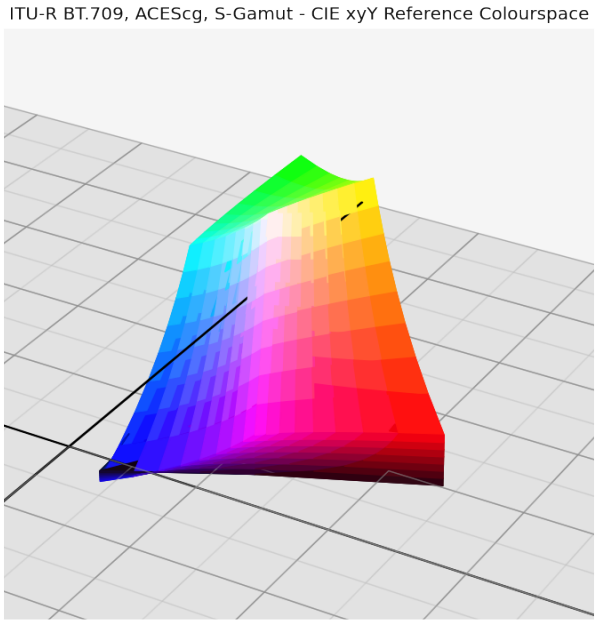
- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot the gamuts. colourspaces elements can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **reference_colourspace** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE 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 or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromatically_adapt** (bool, optional) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- ****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'])
...
(<Figure size ... with 1 Axes>, <...Axes3DSubplot...>)
```



colour.plotting.plot_RGB_scatter

colour.plotting.**plot_RGB_scatter**(*RGB*, *colourspace*, *reference_colourspace='CIE xyY'*, *colourspaces=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'*, *chromatically_adapt=False*, ***kwargs*)

Plots given *RGB* colourspace array in a scatter plot.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (unicode or RGB_Colourspace) – *RGB* colourspace of the *RGB* array. colourspace can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **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.
- **colourspaces** (unicode or RGB_Colourspace or array_like) – *RGB* colourspaces to plot the gamuts. colourspaces elements can be of any type or form supported by the colour.plotting.filter_RGB_colourspaces() definition.
- **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.

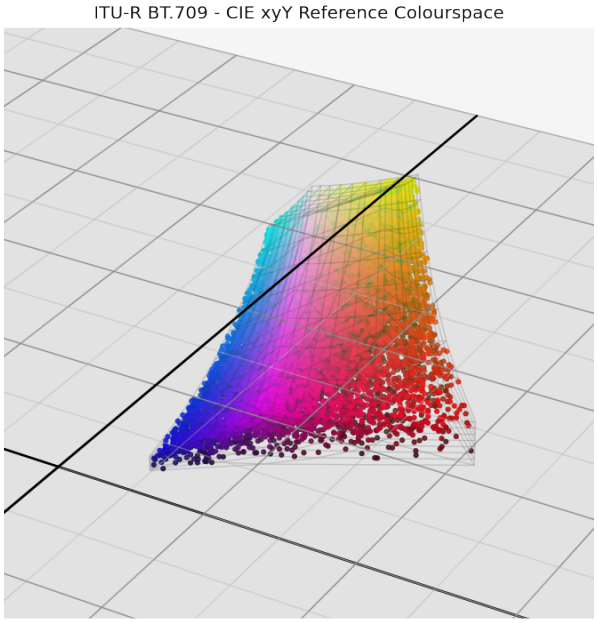
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **show_spectral_locus** (bool, optional) – Whether to show the spectral locus.
- **spectral_locus_colour** (array_like, optional) – Spectral locus colour.
- **points_size** (numeric, optional) – Scatter points size.
- **cmfs** (unicode or XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions used for computing the spectral locus boundaries. cmfs can be of any type or form supported by the colour.plotting.filter_cmfs() definition.
- **chromatically_adapt** (bool, optional) – Whether to chromatically adapt the *RGB* colourspaces given in colourspaces to the whitepoint of the default plotting colourspace.
- ****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')
(<Figure size ... with 1 Axes>, <...Axes3DSubplot...>)
```



Automatic Colour Conversion Graph

colour.plotting

plot_automatic_colour_conversion_graph(filename)	Plots *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

colour.plotting.plot_automatic_colour_conversion_graph

colour.plotting.**plot_automatic_colour_conversion_graph**(*filename*, *prog='fdp'*, *args=''*)

Plots *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

Parameters

- **filename** (unicode) – Filename to use to save the image.
- **prog** (unicode, optional) – {'neato', 'dot', 'twopi', 'circo', 'fdp', 'nop'}, *Graphviz* layout method.
- **args** (unicode, optional) – Additional arguments for *Graphviz*.

Returns *Pyraphviz* graph.

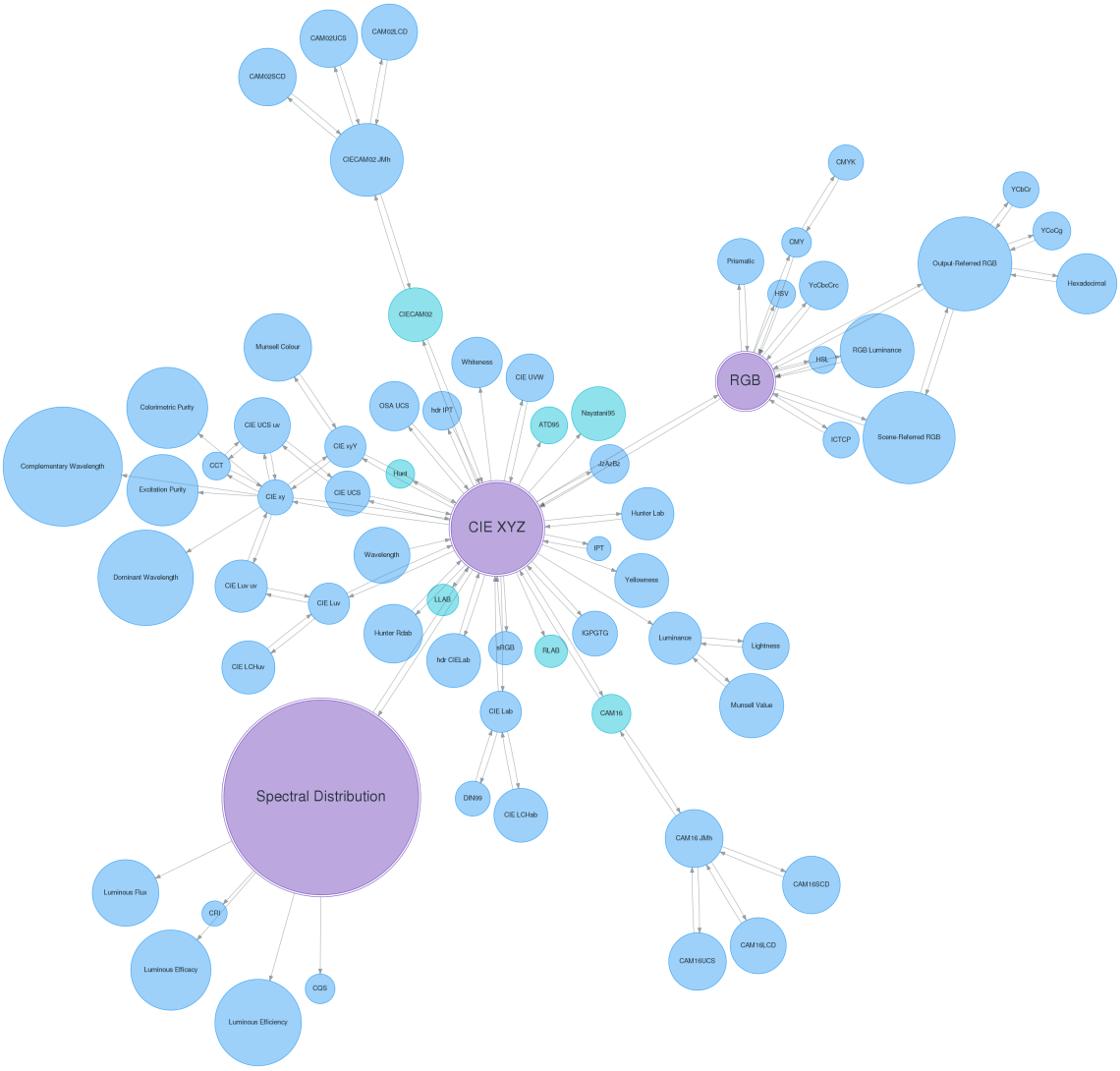
Return type AGraph

Notes

- This definition does not directly plot the *Colour* automatic colour conversion graph but instead write it to an image.

Examples

```
>>> import tempfile
>>> import colour
>>> from colour import read_image
>>> from colour.plotting import plot_image
>>> filename = '{0}.png'.format(tempfile.mkstemp()[-1])
>>> _ = plot_automatic_colour_conversion_graph(filename, 'dot')
...
>>> plot_image(read_image(filename))
```



Colour Quality

- *Colour Fidelity Index*
- *Colour Rendering Index*
- *Colour Quality Scale*
- *Academy Spectral Similarity Index (SSI)*

Colour Fidelity Index

`colour`

`COLOUR_FIDELITY_INDEX_METHODS`	Supported *Colour Fidelity Index* (CFI) computation methods.
`colour_fidelity_index`(sd_test[, ...])	Returns the *Colour Fidelity Index* (CFI) R_f of given spectral distribution using given method.

colour.COLOUR_FIDELITY_INDEX_METHODS

colour.**COLOUR_FIDELITY_INDEX_METHODS = CaseInsensitiveMapping({'CIE 2017': ..., 'ANSI/IES TM-30-18': ...})**

Supported *Colour Fidelity Index* (CFI) computation methods.

References

[], []

COLOUR_FIDELITY_INDEX_METHODS [tuple] **{'CIE 2017', 'ANSI/IES TM-30-18'}**

colour.colour_fidelity_index

colour.**colour_fidelity_index**(*sd_test*, *additional_data=False*, *method='CIE 2017'*)

Returns the *Colour Fidelity Index* (CFI) R_f 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) – **{'CIE 2017', 'ANSI/IES TM-30-18'}**, Computation method.

Returns *Colour Fidelity Index* (CFI) R_f.

Return type numeric or *ColourRendering_Specification_CIE2017* or *ColourQuality_Specification_ANSIIESTM3018*

References

[], []

Examples

```
>>> from colour.colorimetry import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> colour_fidelity_index(sd)
70.1208254...
```

colour.quality

ColourRendering_Specification_CIE2017(name, ...)	Defines the *CIE 2017 Colour Fidelity Index* (CFI) colour quality specification.
colour_fidelity_index_CIE2017(sd_test[, ...])	Returns the *CIE 2017 Colour Fidelity Index* (CFI) R_f of given spectral distribution.
ColourQuality_Specification_ANSIIESTM3018(...)	Defines the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) colour quality specification.
colour_fidelity_index_ANSIIESTM3018(sd_test)	Returns the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

colour.quality.ColourRendering_Specification_CIE2017

class colour.quality.**ColourRendering_Specification_CIE2017**(*name*, *sd_reference*, *R_f*, *R_s*, *CCT*, *D_uv*, *colorimetry_data*, *delta_E_s*)

Defines the *CIE 2017 Colour Fidelity Index* (CFI) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **sd_reference** (SpectralDistribution) – Spectral distribution of the reference illuminant.
- **R_f** (numeric) – *CIE 2017 Colour Fidelity Index* (CFI) R_f.
- **R_s** (array_like) – Individual *colour fidelity indexes* data for each sample.
- **CCT** (numeric) – Correlated colour temperature T_{cp}.
- **D_uv** (numeric) – Distance from the Planckian locus Δ_{uv}.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.
- **delta_E_s** (ndarray, (16,)) – Colour shifts of samples.

Create new instance of ColourRendering_Specification_CIE2017(name, sd_reference, R_f, R_s, CCT, D_uv, colorimetry_data, delta_E_s)

__init__()

Methods

__init__()

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

CCT	Alias for field number 4
D_uv	Alias for field number 5
R_f	Alias for field number 2
R_s	Alias for field number 3
colorimetry_data	Alias for field number 6
delta_E_s	Alias for field number 7
name	Alias for field number 0
sd_reference	Alias for field number 1

colour.quality.colour_fidelity_index_CIE2017

colour.quality.**colour_fidelity_index_CIE2017**(*sd_test*, *additional_data=False*)

Returns the *CIE 2017 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool, optional) – Whether to output additional data.

Returns *CIE 2017 Colour Fidelity Index* (CFI) R_f.

Return type numeric or *ColourRendering_Specification_CIE2017*

References

[]

Examples

```
>>> from colour.colorimetry import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> colour_fidelity_index_CIE2017(sd)
70.1208254...
```

colour.quality.ColourQuality_Specification_ANSIIESTM3018

class colour.quality.**ColourQuality_Specification_ANSIIESTM3018**(*name*, *sd_test*, *sd_reference*, *R_f*, *R_s*, *CCT*, *D_uv*, *colorimetry_data*, *R_g*, *bins*, *averages_test*, *averages_reference*, *average_norms*, *R_fs*, *R_cs*, *R_hs*)

Defines the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **sd_test** (SpectralDistribution) – Spectral distribution of the tested illuminant.
- **sd_reference** (SpectralDistribution) – Spectral distribution of the reference illuminant.
- **R_f** (numeric) – *Colour Fidelity Index* (CFI) R_f.
- **R_s** (list) – Individual *colour fidelity indexes* data for each sample.
- **CCT** (numeric) – Correlated colour temperature T_{cp}.
- **D_uv** (numeric) – Distance from the Planckian locus Δ_{uv}.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.
- **bins** (list of list of int) – List of 16 lists, each containing the indexes of colour samples that lie in the respective hue bin.
- **averages_test** (ndarray, (16, 2)) – Averages of *CAM02-UCS* a', b' coordinates for each hue bin for test samples.

- **averages_reference** (ndarray, (16, 2)) – Averages for reference samples.
- **average_norms** (ndarray, (16,)) – Distance of averages for reference samples from the origin.
- **R_fs** (ndarray, (16,)) – Local colour fidelities for each hue bin.
- **R_cs** (ndarray, (16,)) – Local chromaticity shifts for each hue bin, in percents.
- **R_hs** (ndarray, (16,)) – Local hue shifts for each hue bin.

Create new instance of ColourQuality_Specification_ANSIIESTM3018(name, sd_test, sd_reference, R_f, R_s, CCT, D_uv, colorimetry_data, R_g, bins, averages_test, averages_reference, average_norms, R_fs, R_cs, R_hs)

__init__()

Methods

__init__()	
count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

CCT	Alias for field number 5
D_uv	Alias for field number 6
R_cs	Alias for field number 14
R_f	Alias for field number 3
R_fs	Alias for field number 13
R_g	Alias for field number 8
R_hs	Alias for field number 15
R_s	Alias for field number 4
average_norms	Alias for field number 12
averages_reference	Alias for field number 11
averages_test	Alias for field number 10
bins	Alias for field number 9
colorimetry_data	Alias for field number 7
name	Alias for field number 0
sd_reference	Alias for field number 2
sd_test	Alias for field number 1

colour.quality.colour_fidelity_index_ANSIIESTM3018

colour.quality.**colour_fidelity_index_ANSIIESTM3018**(*sd_test*, *additional_data=False*)

Returns the *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI) R_f of given spectral distribution.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool, optional) – Whether to output additional data.

Returns *ANSI/IES TM-30-18 Colour Fidelity Index* (CFI).

Return type numeric or *ColourQuality_Specification_ANSIIESTM3018*

References

[]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> colour_fidelity_index_ANSIIESTM3018(sd)
70.1208254...
```

Colour Rendering Index

colour

colour_rendering_index(sd_test[, ...])	Returns the *Colour Rendering Index* (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 *ColourRendering_Specification_CRI*

References

[]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> colour_rendering_index(sd)
64.2337241...
```

colour.quality

ColourRendering_Specification_CRI(name, Q_a, ...)	Defines the *Colour Rendering Index* (CRI) colour quality specification.

colour.quality.ColourRendering_Specification_CRI

class colour.quality.**ColourRendering_Specification_CRI**(*name*, *Q_a*, *Q_as*, *colorimetry_data*)

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

[]

Create new instance of ColourRendering_Specification_CRI(name, Q_a, Q_as, colorimetry_data)

__init__()

Methods

__init__()	

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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

COLOUR_QUALITY_SCALE_METHODS	Supported *Colour Quality Scale* (CQS) computation methods.
colour_quality_scale(sd_test[, ...])	Returns the *Colour Quality Scale* (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

[], [], []

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 *ColourRendering_Specification_CQS*

References

[], [], []

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd = SDS_ILLUMINANTS['FL2']
>>> colour_quality_scale(sd)
64.1117031...
```

colour.quality

ColourRendering_Specification_CQS(name, Q_a, ...)	Defines the *Colour Quality Scale* (CQS) colour rendering (quality) specification.

colour.quality.ColourRendering_Specification_CQS

class colour.quality.**ColourRendering_Specification_CQS**(*name*, *Q_a*, *Q_f*, *Q_p*, *Q_g*, *Q_d*, *Q_as*, *colorimetry_data*)

Defines the *Colour Quality Scale* (CQS) colour rendering (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

[], [], []

Create new instance of ColourRendering_Specification_CQS(name, Q_a, Q_f, Q_p, Q_g, Q_d, Q_as, colorimetry_data)

__init__()

Methods

__init__()

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	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

Academy Spectral Similarity Index (SSI)

colour

spectral_similarity_index(sd_test, sd_reference)	Returns the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

colour.spectral_similarity_index

colour.**spectral_similarity_index**(*sd_test*, *sd_reference*)

Returns the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **sd_reference** (SpectralDistribution) – Reference spectral distribution.

Returns *Academy Spectral Similarity Index* (SSI).

Return type numeric

References

[]

Examples

```
>>> from colour import SDS_ILLUMINANTS
>>> sd_test = SDS_ILLUMINANTS['C']
>>> sd_reference = SDS_ILLUMINANTS['D65']
>>> spectral_similarity_index(sd_test, sd_reference)
94.0
```

Reflectance Recovery

- *CIE XYZ Colourspace to Spectral*
- *Jakob and Hanika (2019)*
- *Mallett and Yuksel (2019)*
- *Meng, Simon and Hanika (2015)*
- *Otsu, Yamamoto and Hachisuka (2018)*
- *Smits (1999)*

CIE XYZ Colourspace to Spectral

colour

XYZ_to_sd(XYZ[, method])	Recovers the spectral distribution of given *CIE XYZ* tristimulus values using given method.
XYZ_TO_SD_METHODS	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', 'Jakob 2019', 'Mallett 2019', 'Otsu 2018', 'Smits 1999'}** Computation method.
- **additional_data** (bool, optional) – {colour.recovery.XYZ_to_sd_Jakob2019()}, If *True*, error will be returned alongside sd.
- **basis_functions** (MultiSpectralDistributions) – {colour.recovery.RGB_to_sd_Mallett2019()}, Basis functions for the method. The default is to use the built-in *sRGB* basis functions, i.e. colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019.
- **clip** (bool, optional) – {colour.recovery.XYZ_to_sd_Otsu2018()}, If *True*, the default, values below zero and above unity in the recovered spectral distributions will be clipped. This ensures that the returned reflectance is physical and conserves energy, but will cause noticeable colour differences in case of very saturated colours.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – {colour.recovery.XYZ_to_sd_Meng2015()}, Standard observer colour matching functions.
- **colourspace** (RGB_Colourspace, optional) – {colour.recovery.XYZ_to_sd_Jakob2019()}, *RGB* colourspace of the target colour. Note that no chromatic adaptation is performed between illuminant and the colourspace whitepoint.
- **dataset** (Dataset_Otsu2018, optional) – {colour.recovery.XYZ_to_sd_Otsu2018()}, Dataset to use for reconstruction. The default is to use the published data.
- **illuminant** (SpectralDistribution, optional) – {colour.recovery.XYZ_to_sd_Jakob2019(), colour.recovery.XYZ_to_sd_Meng2015()}, Illuminant spectral distribution.
- **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_kwargs** (dict_like, optional) – {colour.recovery.XYZ_to_sd_Jakob2019(), colour.recovery.XYZ_to_sd_Meng2015()}, Parameters for scipy.optimize.minimize() and colour.recovery.find_coefficients_Jakob2019() definitions.

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 *sRGB* colourspace array assuming equal energy illuminant *E*.

References

[], [], [], [], []

Examples

Jakob and Hanika (2009) reflectance recovery:

```
>>> import numpy as np
>>> from colour.colorimetry import (
...     MSDS_CMFS_STANDARD_OBSERVER, SDS_ILLUMINANTS, SpectralShape,
...     sd_to_XYZ_integration
... )
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method='Jakob 2019', cmfs=cmfs, illuminant=illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.4853113...],
                      [ 370.        ,    0.3229161...],
                      [ 380.        ,    0.2131016...],
                      [ 390.        ,    0.1473214...],
                      [ 400.        ,    0.1081038...],
                      [ 410.        ,    0.0838303...],
                      [ 420.        ,    0.0681378...],
                      [ 430.        ,    0.0576120...],
                      [ 440.        ,    0.0503683...],
                      [ 450.        ,    0.0453253...],
                      [ 460.        ,    0.0418415...],
                      [ 470.        ,    0.0395292...],
                      [ 480.        ,    0.0381568...],
                      [ 490.        ,    0.0375964...],
                      [ 500.        ,    0.0377979...],
                      [ 510.        ,    0.0387793...],
                      [ 520.        ,    0.0406297...],
                      [ 530.        ,    0.0435277...],
                      [ 540.        ,    0.0477792...],
```

(continues on next page)

(continued from previous page)

```
                      [ 550.        ,    0.0538903...],
                      [ 560.        ,    0.0627048...],
                      [ 570.        ,    0.0756689...],
                      [ 580.        ,    0.0953554...],
                      [ 590.        ,    0.1265094...],
                      [ 600.        ,    0.1779719...],
                      [ 610.        ,    0.2648611...],
                      [ 620.        ,    0.4036392...],
                      [ 630.        ,    0.5826025...],
                      [ 640.        ,    0.7439192...],
                      [ 650.        ,    0.8495306...],
                      [ 660.        ,    0.9091674...],
                      [ 670.        ,    0.9423310...],
                      [ 680.        ,    0.9614969...],
                      [ 690.        ,    0.9731491...],
                      [ 700.        ,    0.9805843...],
                      [ 710.        ,    0.9855336...],
                      [ 720.        ,    0.9889492...],
                      [ 730.        ,    0.9913796...],
                      [ 740.        ,    0.9931546...],
                      [ 750.        ,    0.9944803...],
                      [ 760.        ,    0.9954898...],
                      [ 770.        ,    0.9962715...],
                      [ 780.        ,    0.9968859...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065462...,  0.1220220...,  0.0513514...])
```

Mallett and Yuksel (2019) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_sRGB_MALLETT2019)
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(XYZ, method='Mallett 2019')
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 380.        ,    0.1735531...],
                      [ 385.        ,    0.1720357...],
                      [ 390.        ,    0.1677721...],
                      [ 395.        ,    0.1576605...],
                      [ 400.        ,    0.1372829...],
                      [ 405.        ,    0.1170849...],
                      [ 410.        ,    0.0895694...],
                      [ 415.        ,    0.0706232...],
                      [ 420.        ,    0.0585765...],
                      [ 425.        ,    0.0523959...],
                      [ 430.        ,    0.0497598...],
                      [ 435.        ,    0.0476057...],
                      [ 440.        ,    0.0465079...],
                      [ 445.        ,    0.0460337...],
```

(continues on next page)

(continued from previous page)

```
                       [ 450.        ,    0.0455839...],
                       [ 455.        ,    0.0452872...],
                       [ 460.        ,    0.0450981...],
                       [ 465.        ,    0.0448895...],
                       [ 470.        ,    0.0449257...],
                       [ 475.        ,    0.0448987...],
                       [ 480.        ,    0.0446834...],
                       [ 485.        ,    0.0441372...],
                       [ 490.        ,    0.0417137...],
                       [ 495.        ,    0.0373832...],
                       [ 500.        ,    0.0357657...],
                       [ 505.        ,    0.0348263...],
                       [ 510.        ,    0.0341953...],
                       [ 515.        ,    0.0337683...],
                       [ 520.        ,    0.0334979...],
                       [ 525.        ,    0.0332991...],
                       [ 530.        ,    0.0331909...],
                       [ 535.        ,    0.0332181...],
                       [ 540.        ,    0.0333387...],
                       [ 545.        ,    0.0334970...],
                       [ 550.        ,    0.0337381...],
                       [ 555.        ,    0.0341847...],
                       [ 560.        ,    0.0346447...],
                       [ 565.        ,    0.0353993...],
                       [ 570.        ,    0.0367367...],
                       [ 575.        ,    0.0392007...],
                       [ 580.        ,    0.0445902...],
                       [ 585.        ,    0.0625633...],
                       [ 590.        ,    0.2965381...],
                       [ 595.        ,    0.4215576...],
                       [ 600.        ,    0.4347139...],
                       [ 605.        ,    0.4385134...],
                       [ 610.        ,    0.4385184...],
                       [ 615.        ,    0.4385249...],
                       [ 620.        ,    0.4374694...],
                       [ 625.        ,    0.4384672...],
                       [ 630.        ,    0.4368251...],
                       [ 635.        ,    0.4340867...],
                       [ 640.        ,    0.4303219...],
                       [ 645.        ,    0.4243257...],
                       [ 650.        ,    0.4159482...],
                       [ 655.        ,    0.4057443...],
                       [ 660.        ,    0.3919874...],
                       [ 665.        ,    0.3742784...],
                       [ 670.        ,    0.3518421...],
                       [ 675.        ,    0.3240127...],
                       [ 680.        ,    0.2955145...],
                       [ 685.        ,    0.2625658...],
                       [ 690.        ,    0.2343423...],
                       [ 695.        ,    0.2174830...],
                       [ 700.        ,    0.2060461...],
                       [ 705.        ,    0.1977437...],
                       [ 710.        ,    0.1916846...],
                       [ 715.        ,    0.1861020...],
                       [ 720.        ,    0.1823908...],
                       [ 725.        ,    0.1807923...],
```

(continues on next page)

(continued from previous page)

```
                      [ 730.        ,    0.1795571...],
                      [ 735.        ,    0.1785623...],
                      [ 740.        ,    0.1775758...],
                      [ 745.        ,    0.1771614...],
                      [ 750.        ,    0.1767431...],
                      [ 755.        ,    0.1764319...],
                      [ 760.        ,    0.1762597...],
                      [ 765.        ,    0.1762209...],
                      [ 770.        ,    0.1761803...],
                      [ 775.        ,    0.1761195...],
                      [ 780.        ,    0.1760763...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
...
array([ 0.2065436...,  0.1219996...,  0.0513764...])
```

Meng (2015) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method='Meng 2015', cmfs=cmfs, illuminant=illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.0765153...],
                      [ 370.        ,    0.0764771...],
                      [ 380.        ,    0.0764286...],
                      [ 390.        ,    0.0764329...],
                      [ 400.        ,    0.0765863...],
                      [ 410.        ,    0.0764339...],
                      [ 420.        ,    0.0757213...],
                      [ 430.        ,    0.0733091...],
                      [ 440.        ,    0.0676493...],
                      [ 450.        ,    0.0577616...],
                      [ 460.        ,    0.0440805...],
                      [ 470.        ,    0.0284802...],
                      [ 480.        ,    0.0138019...],
                      [ 490.        ,    0.0033557...],
                      [ 500.        ,    0.       ...],
                      [ 510.        ,    0.       ...],
                      [ 520.        ,    0.       ...],
                      [ 530.        ,    0.       ...],
                      [ 540.        ,    0.0055360...],
                      [ 550.        ,    0.0317335...],
                      [ 560.        ,    0.075457 ...],
                      [ 570.        ,    0.1314930...],
                      [ 580.        ,    0.1938219...],
                      [ 590.        ,    0.2559747...],
                      [ 600.        ,    0.3122869...],
```

(continues on next page)

(continued from previous page)

```
                      [ 610.        ,    0.3584363...],
                      [ 620.        ,    0.3927112...],
                      [ 630.        ,    0.4158866...],
                      [ 640.        ,    0.4305832...],
                      [ 650.        ,    0.4391142...],
                      [ 660.        ,    0.4439484...],
                      [ 670.        ,    0.4464121...],
                      [ 680.        ,    0.4475718...],
                      [ 690.        ,    0.4481182...],
                      [ 700.        ,    0.4483734...],
                      [ 710.        ,    0.4484743...],
                      [ 720.        ,    0.4485753...],
                      [ 730.        ,    0.4486474...],
                      [ 740.        ,    0.4486629...],
                      [ 750.        ,    0.4486995...],
                      [ 760.        ,    0.4486925...],
                      [ 770.        ,    0.4486794...],
                      [ 780.        ,    0.4486982...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

Otsu, Yamamoto and Hachisuka (2018) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_OTSU2018)
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(
...     XYZ, method='Otsu 2018', cmfs=cmfs, illuminant=illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 380.        ,    0.0601939...],
                      [ 390.        ,    0.0568063...],
                      [ 400.        ,    0.0517429...],
                      [ 410.        ,    0.0495841...],
                      [ 420.        ,    0.0502007...],
                      [ 430.        ,    0.0506489...],
                      [ 440.        ,    0.0510020...],
                      [ 450.        ,    0.0493782...],
                      [ 460.        ,    0.0468046...],
                      [ 470.        ,    0.0437132...],
                      [ 480.        ,    0.0416957...],
                      [ 490.        ,    0.0403783...],
                      [ 500.        ,    0.0405197...],
                      [ 510.        ,    0.0406031...],
                      [ 520.        ,    0.0416912...],
                      [ 530.        ,    0.0430956...],
                      [ 540.        ,    0.0444474...],
                      [ 550.        ,    0.0459336...],
                      [ 560.        ,    0.0507631...],
```

(continues on next page)

(continued from previous page)

```
                      [ 570.        ,    0.0628967...],
                      [ 580.        ,    0.0844661...],
                      [ 590.        ,    0.1334277...],
                      [ 600.        ,    0.2262428...],
                      [ 610.        ,    0.3599330...],
                      [ 620.        ,    0.4885571...],
                      [ 630.        ,    0.5752546...],
                      [ 640.        ,    0.6193023...],
                      [ 650.        ,    0.6450744...],
                      [ 660.        ,    0.6610548...],
                      [ 670.        ,    0.6688673...],
                      [ 680.        ,    0.6795426...],
                      [ 690.        ,    0.6887933...],
                      [ 700.        ,    0.7003469...],
                      [ 710.        ,    0.7084128...],
                      [ 720.        ,    0.7154674...],
                      [ 730.        ,    0.7234334...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065494...,  0.1219712...,  0.0514002...])
```

Smits (1999) reflectance recovery:

```
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['E'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd(XYZ, method='Smits 1999')
>>> with numpy_print_options(suppress=True):
...     sd
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_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.1894770...,  0.1126470...,  0.0474420...])
```

colour.XYZ_TO_SD_METHODS

colour.**XYZ_TO_SD_METHODS = CaseInsensitiveMapping({'Jakob 2019': ..., 'Mallett 2019': ..., 'Meng 2015': ..., 'Otsu 2018': ..., 'Smits 1999': ...})**

Supported spectral distribution recovery methods.

References

[], [], [], []

XYZ_TO_SD_METHODS [CaseInsensitiveMapping] **{'Jakob 2019', 'Mallett 2019', 'Meng 2015', 'Otsu 2018', 'Smits 1999'}**

Jakob and Hanika (2019)

colour.recovery

XYZ_to_sd_Jakob2019(XYZ[, cmfs, illuminant, ...])	Recovers the spectral distribution of given RGB colourspace array using *Jakob and Hanika (2019)* method.
LUT3D_Jakob2019()	Class for working with pre-computed lookup tables for the *Jakob and Hanika (2019)* spectral upsampling method.

colour.recovery.XYZ_to_sd_Jakob2019

colour.recovery.**XYZ_to_sd_Jakob2019**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*, *optimisation_kwargs=None*, *additional_data=False*)

Recovers the spectral distribution of given RGB colourspace array using *Jakob and Hanika (2019)* method.

Parameters

- **XYZ** (array_like, (3,)) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **optimisation_kwargs** (dict_like, optional) – Parameters for colour.recovery.find_coefficients_Jakob2019() definition.
- **additional_data** (bool, optional) – If *True*, error will be returned alongside sd.

Returns

- **sd** (*SpectralDistribution*) – Recovered spectral distribution.
- **error** (*float*) – ΔE_{76} between the target colour and the colour corresponding to the computed coefficients.

References

[]

Examples

```
>>> from colour.colorimetry import CCS_ILLUMINANTS, sd_to_XYZ_integration
>>> from colour.models import XYZ_to_sRGB
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Jakob2019(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.4884502...],
                      [ 370.        ,    0.3251871...],
                      [ 380.        ,    0.2144337...],
                      [ 390.        ,    0.1480663...],
                      [ 400.        ,    0.1085298...],
                      [ 410.        ,    0.0840835...],
                      [ 420.        ,    0.0682934...],
                      [ 430.        ,    0.0577098...],
                      [ 440.        ,    0.0504300...],
                      [ 450.        ,    0.0453634...],
                      [ 460.        ,    0.0418635...],
                      [ 470.        ,    0.0395397...],
                      [ 480.        ,    0.0381585...],
                      [ 490.        ,    0.0375912...],
                      [ 500.        ,    0.0377870...],
                      [ 510.        ,    0.0387631...],
                      [ 520.        ,    0.0406086...],
                      [ 530.        ,    0.0435015...],
                      [ 540.        ,    0.0477476...],
                      [ 550.        ,    0.0538528...],
                      [ 560.        ,    0.0626607...],
                      [ 570.        ,    0.0756177...],
                      [ 580.        ,    0.0952978...],
                      [ 590.        ,    0.1264501...],
                      [ 600.        ,    0.1779277...],
                      [ 610.        ,    0.2648782...],
                      [ 620.        ,    0.4037993...],
                      [ 630.        ,    0.5829234...],
                      [ 640.        ,    0.7442651...],
                      [ 650.        ,    0.8497961...],
                      [ 660.        ,    0.9093483...],
                      [ 670.        ,    0.9424527...],
                      [ 680.        ,    0.9615805...],
                      [ 690.        ,    0.9732085...],
                      [ 700.        ,    0.9806277...],
                      [ 710.        ,    0.9855663...],
                      [ 720.        ,    0.9889743...],
```

(continues on next page)

(continued from previous page)

```
                       [ 730.        ,    0.9913993...],
                       [ 740.        ,    0.9931703...],
                       [ 750.        ,    0.9944931...],
                       [ 760.        ,    0.9955002...],
                       [ 770.        ,    0.9962802...],
                       [ 780.        ,    0.9968932...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065841...,  0.1220125...,  0.0514023...])
```

colour.recovery.LUT3D_Jakob2019

class colour.recovery.**LUT3D_Jakob2019**

Class for working with pre-computed lookup tables for the *Jakob and Hanika (2019)* spectral upsampling method. It allows significant time savings by performing the expensive numerical optimization ahead of time and storing the results in a file.

The file format is compatible with the code and **.coeff* files in the supplemental material published alongside the article. They are directly available from Colour - Datasets under the record *4050598*.

Attributes

- size
- lightness_scale
- coefficients
- interpolator

Methods

- __init__()
- generate()
- RGB_to_coefficients()
- RGB_to_sd()
- read()
- write()

References

[]

Examples

```
>>> import os
>>> import colour
>>> from colour.models import RGB_COLOURSPACE_sRGB
>>> from colour.utilities import numpy_print_options
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> LUT = LUT3D_Jakob2019()
>>> LUT.generate(RGB_COLOURSPACE_sRGB, cmfs, illuminant, 3, lambda x: x)
>>> path = os.path.join(colour.__path__[0], 'recovery', 'tests',
...                     'resources', 'sRGB_Jakob2019.coeff')
>>> LUT.write(path)
>>> LUT.read(path)
>>> RGB = np.array([0.70573936, 0.19248266, 0.22354169])
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     LUT.RGB_to_sd(RGB, cmfs.shape)
SpectralDistribution([[ 360.        ,    0.7663248...],
                      [ 370.        ,    0.6248040...],
                      [ 380.        ,    0.4582328...],
                      [ 390.        ,    0.3161403...],
                      [ 400.        ,    0.2196885...],
                      [ 410.        ,    0.1597642...],
                      [ 420.        ,    0.1226653...],
                      [ 430.        ,    0.0990878...],
                      [ 440.        ,    0.0836822...],
                      [ 450.        ,    0.0734525...],
                      [ 460.        ,    0.0667002...],
                      [ 470.        ,    0.0624502...],
                      [ 480.        ,    0.0601529...],
                      [ 490.        ,    0.0595328...],
                      [ 500.        ,    0.0605182...],
                      [ 510.        ,    0.0632235...],
                      [ 520.        ,    0.0679778...],
                      [ 530.        ,    0.0754093...],
                      [ 540.        ,    0.0866232...],
                      [ 550.        ,    0.1035471...],
                      [ 560.        ,    0.1295933...],
                      [ 570.        ,    0.1708525...],
                      [ 580.        ,    0.2377171...],
                      [ 590.        ,    0.3442627...],
                      [ 600.        ,    0.4952907...],
                      [ 610.        ,    0.6605014...],
                      [ 620.        ,    0.7914286...],
                      [ 630.        ,    0.8738002...],
                      [ 640.        ,    0.9212534...],
                      [ 650.        ,    0.9486329...],
                      [ 660.        ,    0.9650124...],
```

(continues on next page)

(continued from previous page)

```
                       [ 670.        ,    0.9752510...],
                       [ 680.        ,    0.9819246...],
                       [ 690.        ,    0.9864387...],
                       [ 700.        ,    0.9895916...],
                       [ 710.        ,    0.9918554...],
                       [ 720.        ,    0.9935199...],
                       [ 730.        ,    0.9947694...],
                       [ 740.        ,    0.9957242...],
                       [ 750.        ,    0.9964656...],
                       [ 760.        ,    0.9970494...],
                       [ 770.        ,    0.9975148...],
                       [ 780.        ,    0.9978900...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
```

__init__()

Methods

RGB_to_coefficients(RGB)	Look up a given *RGB* colourspace array and return corresponding coefficients.
RGB_to_sd(RGB[, shape])	Looks up a given *RGB* colourspace array and return the corresponding spectral distribution.
__init__()	
generate(colourspace[, cmfs, illuminant, ...])	Generates the lookup table data for given *RGB* colourspace, colour matching functions, illuminant and given size.
read(path)	Loads a lookup table from a **.coeff* file.
write(path)	Writes the lookup table to a **.coeff* file.

Attributes

coefficients	Getter property for the *Jakob and Hanika (2019)* interpolator coefficients.
interpolator	Getter property for the *Jakob and Hanika (2019)* interpolator.
lightness_scale	Getter property for the *Jakob and Hanika (2019)* interpolator lightness scale.
size	Getter property for the *Jakob and Hanika (2019)* interpolator size, i.e. the samples count on one side of the 3D table.

Ancillary Objects

colour.recovery

sd_Jakob2019(coefficients[, shape])	Returns a spectral distribution following the spectral model given by *Jakob and Hanika (2019)*.
find_coefficients_Jakob2019(XYZ[, cmfs, ...])	Computes the coefficients for *Jakob and Hanika (2019)* reflectance spectral model.

colour.recovery.sd_Jakob2019

colour.recovery.**sd_Jakob2019**(*coefficients*, *shape=SpectralShape(360, 780, 5)*)

Returns a spectral distribution following the spectral model given by *Jakob and Hanika (2019)*.

Parameters

- **coefficients** (array_like) – Dimensionless coefficients for *Jakob and Hanika (2019)* reflectance spectral model.
- **shape** (SpectralShape, optional) – Shape used by the spectral distribution.

Returns *Jakob and Hanika (2019)* spectral distribution.

Return type *SpectralDistribution*

References

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_Jakob2019([-9e-05, 8.5e-02, -20], SpectralShape(400, 700, 20))
...
SpectralDistribution([[ 400.        ,    0.3143046...],
                      [ 420.        ,    0.4133320...],
                      [ 440.        ,    0.4880034...],
                      [ 460.        ,    0.5279562...],
                      [ 480.        ,    0.5319346...],
                      [ 500.        ,    0.5      ...],
                      [ 520.        ,    0.4326202...],
                      [ 540.        ,    0.3373544...],
                      [ 560.        ,    0.2353056...],
                      [ 580.        ,    0.1507665...],
                      [ 600.        ,    0.0931332...],
                      [ 620.        ,    0.0577434...],
                      [ 640.        ,    0.0367011...],
                      [ 660.        ,    0.0240879...],
                      [ 680.        ,    0.0163316...],
                      [ 700.        ,    0.0114118...]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={...})
```

colour.recovery.find_coefficients_Jakob2019

colour.recovery.**find_coefficients_Jakob2019**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*, *coefficients_0=array([0., 0., 0.])*, *max_error=0.023*, *dimensionalise=True*)

Computes the coefficients for *Jakob and Hanika (2019)* reflectance spectral model.

Parameters

- **XYZ** (array_like, (3,)) – *CIE XYZ* tristimulus values to find the coefficients for.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **coefficients_0** (array_like, (3,), optional) – Starting coefficients for the solver.
- **max_error** (float, optional) – Maximal acceptable error. Set higher to save computational time. If *None*, the solver will keep going until it is very close to the minimum. The default is ACCEPTABLE_DELTA_E.
- **dimensionalise** (bool, optional) – If *True*, returned coefficients are dimensionful and will not work correctly if fed back as coefficients_0. The default is *True*.

Returns

- **coefficients** (*ndarray, (3,)*) – Computed coefficients that best fit the given colour.
- **error** (*float*) – ΔE_{76} between the target colour and the colour corresponding to the computed coefficients.

References

[]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> find_coefficients_Jakob2019(XYZ)
(array([  1.3723791...e-04,  -1.3514399...e-01,   3.0838973...e+01]), 0.0141941...)
```

Mallett and Yuksel (2019)

colour.recovery

RGB_to_sd_Mallett2019(RGB[, basis_functions])	Recovers the spectral distribution of given *RGB* colourspace array using *Mallett and Yuksel (2019)* method.

colour.recovery.RGB_to_sd_Mallett2019

colour.recovery.**RGB_to_sd_Mallett2019**(*RGB*, *basis_functions=MultiSpectralDistributions(name='Basis Functions - sRGB - Mallett 2019', ...)*)

Recovers the spectral distribution of given *RGB* colourspace array using *Mallett and Yuksel (2019)* method.

Parameters

- **RGB** (array_like, (3,)) – *RGB* colourspace array.
- **basis_functions** (MultiSpectralDistributions) – Basis functions for the method. The default is to use the built-in *sRGB* basis functions, i.e. colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019.

Returns Recovered reflectance.

Return type *SpectralDistribution*

References

[]

Notes

- In-addition to the *BT.709* primaries used by the *sRGB* colourspace, [] tried *BT.2020*, *P3 D65*, *Adobe RGB 1998*, *NTSC (1987)*, *Pal/Secam*, *ProPhoto RGB*, and *Adobe Wide Gamut RGB* primaries, every one of which encompasses a larger (albeit not-always-enveloping) set of *CIE L*a*b** colours than BT.709. Of these, only *Pal/Secam* produces a feasible basis, which is relatively unsurprising since it is very similar to *BT.709*, whereas the others are significantly larger.

Examples

```
>>> from colour.colorimetry import SDS_ILLUMINANTS, sd_to_XYZ_integration
>>> from colour.models import XYZ_to_sRGB
>>> from colour.recovery import SPECTRAL_SHAPE_sRGB_MALLETT2019
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> RGB = XYZ_to_sRGB(XYZ, apply_cctf_encoding=False)
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_sRGB_MALLETT2019)
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = RGB_to_sd_Mallett2019(RGB)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 380.        ,    0.1735531...],
                      [ 385.        ,    0.1720357...],
                      [ 390.        ,    0.1677721...],
                      [ 395.        ,    0.1576605...],
                      [ 400.        ,    0.1372829...],
                      [ 405.        ,    0.1170849...],
                      [ 410.        ,    0.0895694...],
```

(continues on next page)

(continued from previous page)

```
                       [ 415.        ,    0.0706232...],
                       [ 420.        ,    0.0585765...],
                       [ 425.        ,    0.0523959...],
                       [ 430.        ,    0.0497598...],
                       [ 435.        ,    0.0476057...],
                       [ 440.        ,    0.0465079...],
                       [ 445.        ,    0.0460337...],
                       [ 450.        ,    0.0455839...],
                       [ 455.        ,    0.0452872...],
                       [ 460.        ,    0.0450981...],
                       [ 465.        ,    0.0448895...],
                       [ 470.        ,    0.0449257...],
                       [ 475.        ,    0.0448987...],
                       [ 480.        ,    0.0446834...],
                       [ 485.        ,    0.0441372...],
                       [ 490.        ,    0.0417137...],
                       [ 495.        ,    0.0373832...],
                       [ 500.        ,    0.0357657...],
                       [ 505.        ,    0.0348263...],
                       [ 510.        ,    0.0341953...],
                       [ 515.        ,    0.0337683...],
                       [ 520.        ,    0.0334979...],
                       [ 525.        ,    0.0332991...],
                       [ 530.        ,    0.0331909...],
                       [ 535.        ,    0.0332181...],
                       [ 540.        ,    0.0333387...],
                       [ 545.        ,    0.0334970...],
                       [ 550.        ,    0.0337381...],
                       [ 555.        ,    0.0341847...],
                       [ 560.        ,    0.0346447...],
                       [ 565.        ,    0.0353993...],
                       [ 570.        ,    0.0367367...],
                       [ 575.        ,    0.0392007...],
                       [ 580.        ,    0.0445902...],
                       [ 585.        ,    0.0625633...],
                       [ 590.        ,    0.2965381...],
                       [ 595.        ,    0.4215576...],
                       [ 600.        ,    0.4347139...],
                       [ 605.        ,    0.4385134...],
                       [ 610.        ,    0.4385184...],
                       [ 615.        ,    0.4385249...],
                       [ 620.        ,    0.4374694...],
                       [ 625.        ,    0.4384672...],
                       [ 630.        ,    0.4368251...],
                       [ 635.        ,    0.4340867...],
                       [ 640.        ,    0.4303219...],
                       [ 645.        ,    0.4243257...],
                       [ 650.        ,    0.4159482...],
                       [ 655.        ,    0.4057443...],
                       [ 660.        ,    0.3919874...],
                       [ 665.        ,    0.3742784...],
                       [ 670.        ,    0.3518421...],
                       [ 675.        ,    0.3240127...],
                       [ 680.        ,    0.2955145...],
                       [ 685.        ,    0.2625658...],
                       [ 690.        ,    0.2343423...],
```

(continues on next page)

(continued from previous page)

```
                       [ 695.        ,    0.2174830...],
                       [ 700.        ,    0.2060461...],
                       [ 705.        ,    0.1977437...],
                       [ 710.        ,    0.1916846...],
                       [ 715.        ,    0.1861020...],
                       [ 720.        ,    0.1823908...],
                       [ 725.        ,    0.1807923...],
                       [ 730.        ,    0.1795571...],
                       [ 735.        ,    0.1785623...],
                       [ 740.        ,    0.1775758...],
                       [ 745.        ,    0.1771614...],
                       [ 750.        ,    0.1767431...],
                       [ 755.        ,    0.1764319...],
                       [ 760.        ,    0.1762597...],
                       [ 765.        ,    0.1762209...],
                       [ 770.        ,    0.1761803...],
                       [ 775.        ,    0.1761195...],
                       [ 780.        ,    0.1760763...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
...
array([ 0.2065436...,  0.1219996...,  0.0513764...])
```

Ancillary Objects

colour.recovery

MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019	the base object for multi spectral computations.
SPECTRAL_SHAPE_sRGB_MALLETT2019	Shape for *Mallett and Yuksel (2019) sRGB* colourspace basis functions: (380, 780, 5).
spectral_primary_decomposition_Mallett2019(...)	Performs the spectral primary decomposition as described in *Mallett and Yuksel (2019)* for given *RGB* colourspace.

colour.recovery.MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019

colour.recovery.**MSDS_BASIS_FUNCTIONS_sRGB_MALLETT2019 = MultiSpectralDistributions(name='Basis Functions - sRGB - Mallett 2019', ...)**

the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc...

The multi-spectral distributions 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.

Important: Specific documentation about getting, setting, indexing and slicing the multi-spectral power distributions values is available in the *Spectral Representation and Continuous Signal* section.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **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.
- **name** (unicode, optional) – Multi-spectral distributions name.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_kwargs** (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_kwargs** (dict_like, optional) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (array_like, optional) – Multi-spectral distributions labels for figures, default to colour.MultiSpectralDistributions.labels attribute value.

Attributes

- strict_name
- strict_labels
- wavelengths
- values
- shape

Methods

- __init__()
- interpolate()
- extrapolate()
- align()
- trim()
- normalise()
- to_sds()

References

[], [], []

Examples

Instantiating the multi-spectral distributions 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):
...     MultiSpectralDistributions(data, labels=labels)
...
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_kwargs={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_kwargs={...})
```

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):
...     MultiSpectralDistributions(data, labels=labels)
...
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_kwargs={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_kwargs={...})
```

Instantiation with a *Pandas DataFrame*:

```
>>> from colour.utilities import is_pandas_installed
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     x_bar = [data[key][0] for key in sorted(data.keys())]
...     y_bar = [data[key][1] for key in sorted(data.keys())]
...     z_bar = [data[key][2] for key in sorted(data.keys())]
...     print(MultiSignals(
...         DataFrame(
...             dict(zip(labels, [x_bar, y_bar, z_bar])), data.keys())))
[[  5.0000000...e+02   4.9000000...e-03   3.2300000...e-01   2.7200000...e-01]
 [  5.1000000...e+02   9.3000000...e-03   5.0300000...e-01   1.5820000...e-01]
 [  5.2000000...e+02   3.1400000...e-03   3.1416000...e-01   3.1420000...e-02]
 [  5.3000000...e+02   6.3270000...e-02   7.1000000...e-01   7.8250000...e-02]
 [  5.4000000...e+02   1.6550000...e-01   8.6200000...e-01   4.2160000...e-02]
 [  5.5000000...e+02   2.9040000...e-01   9.5400000...e-01   2.0300000...e-02]
 [  5.6000000...e+02   4.3345000...e-01   9.9495000...e-01   8.7500000...e-03]
 [  5.1100000...e+02   5.9450000...e-01   9.9500000...e-01   3.9000000...e-03]]
```

Type Defines the multi-spectral distributions

colour.recovery.SPECTRAL_SHAPE_sRGB_MALLETT2019

colour.recovery.**SPECTRAL_SHAPE_sRGB_MALLETT2019 = SpectralShape(380, 780, 5)**

Shape for *Mallett and Yuksel (2019) sRGB* colourspace basis functions: (380, 780, 5).

References

[]

SPECTRAL_SHAPE_sRGB_MALLETT2019 : SpectralShape

colour.recovery.spectral_primary_decomposition_Mallett2019

colour.recovery.**spectral_primary_decomposition_Mallett2019**(*colourspace*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*, *metric=<function norm>*, *metric_args=()*, *optimisation_kwargs=None*)

Performs the spectral primary decomposition as described in *Mallett and Yuksel (2019)* for given *RGB* colourspace.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **metric** (unicode, optional) – Function to be minimised, i.e. the objective function.

```
metric(basis, *metric_args) -> float
```

where `basis` is three reflectances concatenated together, each with a shape matching `shape`.

- **`metric_args`** (`tuple, optional`) – Additional arguments passed to `metric`.
- **`optimisation_kwargs`** (`dict_like, optional`) – Parameters for `scipy.optimize.minimize()` definition.

Returns Basis functions for given *RGB* colourspace.

Return type *MultiSpectralDistributions*

References

[]

Notes

- In-addition to the *BT.709* primaries used by the *sRGB* colourspace, [] tried *BT.2020*, *P3 D65*, *Adobe RGB 1998*, *NTSC (1987)*, *Pal/Secam*, *ProPhoto RGB*, and *Adobe Wide Gamut RGB* primaries, every one of which encompasses a larger (albeit not-always-enveloping) set of *CIE L*a*b** colours than BT.709. Of these, only *Pal/Secam* produces a feasible basis, which is relatively unsurprising since it is very similar to *BT.709*, whereas the others are significantly larger.

Examples

```
>>> from colour.colorimetry import SpectralShape
>>> from colour.models import RGB_COLOURSPACE_PAL_SECAM
>>> from colour.utilities import numpy_print_options
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> msds = spectral_primary_decomposition_Mallett2019(
...     RGB_COLOURSPACE_PAL_SECAM, cmfs, illuminant, optimisation_kwargs={
...         'options': {'ftol': 1e-5}
...     }
... )
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     print(msds)
[[ 360.            0.3324728...    0.3332663...    0.3342608...]
 [ 370.            0.3323307...    0.3327746...    0.3348946...]
 [ 380.            0.3341115...    0.3323995...    0.3334889...]
 [ 390.            0.3337570...    0.3298092...    0.3364336...]
 [ 400.            0.3209352...    0.3218213...    0.3572433...]
 [ 410.            0.2881025...    0.2837628...    0.4281346...]
 [ 420.            0.1836749...    0.1838893...    0.6324357...]
 [ 430.            0.0187212...    0.0529655...    0.9283132...]
 [ 440.            0.       ...    0.       ...    1.       ...]
 [ 450.            0.       ...    0.       ...    1.       ...]
 [ 460.            0.       ...    0.       ...    1.       ...]
 [ 470.            0.       ...    0.0509556...    0.9490443...]
```

(continues on next page)

(continued from previous page)

```
[ 480.            0.        ...    0.2933996...    0.7066003...]
[ 490.            0.        ...    0.5001396...    0.4998603...]
[ 500.            0.        ...    0.6734805...    0.3265195...]
[ 510.            0.        ...    0.8555213...    0.1444786...]
[ 520.            0.        ...    0.9999985...    0.0000014...]
[ 530.            0.        ...    1.        ...   0.        ...]
[ 540.            0.        ...    1.        ...   0.        ...]
[ 550.            0.        ...    1.        ...   0.        ...]
[ 560.            0.        ...    0.9924229...    0.        ...]
[ 570.            0.        ...    0.9913344...    0.0083032...]
[ 580.            0.0370289...     0.9145168...    0.0484542...]
[ 590.            0.7100075...     0.2898477...    0.0001446...]
[ 600.            1.        ...    0.        ...   0.        ...]
[ 610.            1.        ...    0.        ...   0.        ...]
[ 620.            1.        ...    0.        ...   0.        ...]
[ 630.            1.        ...    0.        ...   0.        ...]
[ 640.            0.9711347...     0.0137659...    0.0150993...]
[ 650.            0.7996619...     0.1119379...    0.0884001...]
[ 660.            0.6064640...     0.202815 ...    0.1907209...]
[ 670.            0.4662959...     0.2675037...    0.2662005...]
[ 680.            0.4010958...     0.2998989...    0.2990052...]
[ 690.            0.3617485...     0.3208921...    0.3173592...]
[ 700.            0.3496691...     0.3247855...    0.3255453...]
[ 710.            0.3433979...     0.3273540...    0.329248 ...]
[ 720.            0.3358860...     0.3345583...    0.3295556...]
[ 730.            0.3349498...     0.3314232...    0.3336269...]
[ 740.            0.3359954...     0.3340147...    0.3299897...]
[ 750.            0.3310392...     0.3327595...    0.3362012...]
[ 760.            0.3346883...     0.3314158...    0.3338957...]
[ 770.            0.3332167...     0.333371 ...    0.3334122...]
[ 780.            0.3319670...     0.3325476...    0.3354852...]]
```

Meng, Simon and Hanika (2015)

`colour.recovery`

`XYZ_to_sd_Meng2015`(XYZ[, cmfs, illuminant, ...])	Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al. (2015)* method.

colour.recovery.XYZ_to_sd_Meng2015

`colour.recovery.`**`XYZ_to_sd_Meng2015`**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*, *optimisation_kwargs=None*, ***kwargs*)

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. The wavelength λ_i range interval of the colour matching functions affects directly the time the computations take. The current default interval of 5 is a good compromise between precision and time spent.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **optimisation_kwargs** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

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_ASTME308() definition that handles only measurement interval of 1, 5, 10 or 20nm.

References

[]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Meng2015(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.0765153...],
                      [ 370.        ,    0.0764771...],
                      [ 380.        ,    0.0764286...],
                      [ 390.        ,    0.0764329...],
                      [ 400.        ,    0.0765863...],
                      [ 410.        ,    0.0764339...],
                      [ 420.        ,    0.0757213...],
                      [ 430.        ,    0.0733091...],
                      [ 440.        ,    0.0676493...],
                      [ 450.        ,    0.0577616...],
                      [ 460.        ,    0.0440805...],
```

(continues on next page)

(continued from previous page)

```
                       [ 470.        ,    0.0284802...],
                       [ 480.        ,    0.0138019...],
                       [ 490.        ,    0.0033557...],
                       [ 500.        ,    0.       ...],
                       [ 510.        ,    0.       ...],
                       [ 520.        ,    0.       ...],
                       [ 530.        ,    0.       ...],
                       [ 540.        ,    0.0055360...],
                       [ 550.        ,    0.0317335...],
                       [ 560.        ,    0.075457 ...],
                       [ 570.        ,    0.1314930...],
                       [ 580.        ,    0.1938219...],
                       [ 590.        ,    0.2559747...],
                       [ 600.        ,    0.3122869...],
                       [ 610.        ,    0.3584363...],
                       [ 620.        ,    0.3927112...],
                       [ 630.        ,    0.4158866...],
                       [ 640.        ,    0.4305832...],
                       [ 650.        ,    0.4391142...],
                       [ 660.        ,    0.4439484...],
                       [ 670.        ,    0.4464121...],
                       [ 680.        ,    0.4475718...],
                       [ 690.        ,    0.4481182...],
                       [ 700.        ,    0.4483734...],
                       [ 710.        ,    0.4484743...],
                       [ 720.        ,    0.4485753...],
                       [ 730.        ,    0.4486474...],
                       [ 740.        ,    0.4486629...],
                       [ 750.        ,    0.4486995...],
                       [ 760.        ,    0.4486925...],
                       [ 770.        ,    0.4486794...],
                       [ 780.        ,    0.4486982...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

Otsu, Yamamoto and Hachisuka (2018)

`colour.recovery`

XYZ_to_sd_Otsu2018(XYZ[, cmfs, illuminant, ...])	Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Otsu et al. (2018)* method.

colour.recovery.XYZ_to_sd_Otsu2018

colour.recovery.**XYZ_to_sd_Otsu2018**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*, *dataset=<colour.recovery.otsu2018.Dataset_Otsu2018 object>*, *clip=True*)

Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Otsu et al. (2018)* method.

Parameters

- **XYZ** (array_like, (3,)) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **dataset** (Dataset_Otsu2018, optional) – Dataset to use for reconstruction. The default is to use the published data.
- **clip** (bool, optional) – If *True*, the default, values below zero and above unity in the recovered spectral distributions will be clipped. This ensures that the returned reflectance is physical and conserves energy, but will cause noticeable colour differences in case of very saturated colours.

Returns Recovered spectral distribution. Its shape is always that of the colour.recovery.SPECTRAL_SHAPE_OTSU2018 class instance.

Return type *SpectralDistribution*

References

[]

Examples

```
>>> from colour.colorimetry import CCS_ILLUMINANTS, sd_to_XYZ_integration
>>> from colour.models import XYZ_to_sRGB
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_OTSU2018)
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> sd = XYZ_to_sd_Otsu2018(XYZ, cmfs, illuminant)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 380.        ,    0.0601939...],
                      [ 390.        ,    0.0568063...],
                      [ 400.        ,    0.0517429...],
                      [ 410.        ,    0.0495841...],
                      [ 420.        ,    0.0502007...],
                      [ 430.        ,    0.0506489...],
```

(continues on next page)

(continued from previous page)

```
                       [ 440.        ,    0.0510020...],
                       [ 450.        ,    0.0493782...],
                       [ 460.        ,    0.0468046...],
                       [ 470.        ,    0.0437132...],
                       [ 480.        ,    0.0416957...],
                       [ 490.        ,    0.0403783...],
                       [ 500.        ,    0.0405197...],
                       [ 510.        ,    0.0406031...],
                       [ 520.        ,    0.0416912...],
                       [ 530.        ,    0.0430956...],
                       [ 540.        ,    0.0444474...],
                       [ 550.        ,    0.0459336...],
                       [ 560.        ,    0.0507631...],
                       [ 570.        ,    0.0628967...],
                       [ 580.        ,    0.0844661...],
                       [ 590.        ,    0.1334277...],
                       [ 600.        ,    0.2262428...],
                       [ 610.        ,    0.3599330...],
                       [ 620.        ,    0.4885571...],
                       [ 630.        ,    0.5752546...],
                       [ 640.        ,    0.6193023...],
                       [ 650.        ,    0.6450744...],
                       [ 660.        ,    0.6610548...],
                       [ 670.        ,    0.6688673...],
                       [ 680.        ,    0.6795426...],
                       [ 690.        ,    0.6887933...],
                       [ 700.        ,    0.7003469...],
                       [ 710.        ,    0.7084128...],
                       [ 720.        ,    0.7154674...],
                       [ 730.        ,    0.7234334...]],
                      interpolator=SpragueInterpolator,
                      interpolator_kwargs={},
                      extrapolator=Extrapolator,
                      extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.2065494...,  0.1219712...,  0.0514002...])
```

Ancillary Objects

`colour.recovery`

Dataset_Otsu2018([shape, basis_functions, ...])	Stores all the information needed for the *Otsu et al. (2018)* spectral upsampling method.
NodeTree_Otsu2018(reflectances[, cmfs, ...])	A sub-class of `colour.recovery.otsu2018.Node` class representing the root node of a tree containing information shared with all the nodes, such as the standard observer colour matching functions and the illuminant, if any is used.

colour.recovery.Dataset_Otsu2018

class colour.recovery.**Dataset_Otsu2018**(*shape=None*, *basis_functions=None*, *means=None*, *selector_array=None*)

Stores all the information needed for the *Otsu et al. (2018)* spectral upsampling method.

Datasets can be either generated and converted as a `colour.recovery.Dataset_Otsu2018` class instance using the `colour.recovery.NodeTree_Otsu2018.to_dataset()` method or alternatively, loaded from disk with the `colour.recovery.Dataset_Otsu2018.read()` method.

Parameters

- **shape** (SpectralShape) – Shape of the spectral data.
- **basis_functions** (array_like, (n, 3, m)) – Three basis functions for every cluster.
- **means** (array_like, (n, m)) – Mean for every cluster.
- **selector_array** (array_like, (k, 4)) – Array describing how to select the appropriate cluster. See `colour.recovery.Dataset_Otsu2018.select()` method for details.

Attributes

- shape
- basis_functions
- means
- selector_array

Methods

- __init__()
- select()
- cluster()
- read()
- write()

References

[]

Examples

```
>>> import os
>>> import colour
>>> from colour.characterisation import SDS_COLOURCHECKERS
>>> reflectances = [
...     sd.copy().align(SPECTRAL_SHAPE_OTSU2018).values
...     for sd in SDS_COLOURCHECKERS['ColorChecker N Ohta'].values()
... ]
>>> node_tree = NodeTree_Otsu2018(reflectances)
>>> node_tree.optimise(iterations=2, print_callable=lambda x: x)
```

(continues on next page)

(continued from previous page)

```
>>> dataset = node_tree.to_dataset()
>>> path = os.path.join(colour.__path__[0], 'recovery', 'tests',
...                     'resources', 'ColorChecker_Otsu2018.npz')
>>> dataset.write(path)
>>> dataset = Dataset_Otsu2018()
>>> dataset.read(path)
```

__init__(*shape=None*, *basis_functions=None*, *means=None*, *selector_array=None*)

Methods

__init__([shape, basis_functions, means, ...])	
cluster(xy)	Returns the basis functions and dataset mean for the given *CIE xy* coordinates.
read(path)	Reads and loads a dataset from an *.npz* file.
select(xy)	Returns the cluster index appropriate for the given *CIE xy* coordinates.
write(path)	Writes the dataset to an *.npz* file at given path.

Attributes

basis_functions	Getter property for the basis functions of the *Otsu et al. (2018)* dataset.
means	Getter property for means of the *Otsu et al. (2018)* dataset.
selector_array	Getter property for the selector array of the *Otsu et al. (2018)* dataset.
shape	Getter property for the shape used by the *Otsu et al. (2018)* dataset.

colour.recovery.NodeTree_Otsu2018

class colour.recovery.**NodeTree_Otsu2018**(*reflectances*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*)

A sub-class of colour.recovery.otsu2018.Node class representing the root node of a tree containing information shared with all the nodes, such as the standard observer colour matching functions and the illuminant, if any is used.

Global operations involving the entire tree, such as optimisation and reconstruction, are implemented in this sub-class.

Parameters

- **reflectances** (ndarray, (n, m)) – Reflectances of the *n* reference colours to use for optimisation.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.

Attributes

- reflectances
- cmfs
- illuminant
- minimum_cluster_size

Methods

- __init__()
- __str__()
- msds_to_XYZ()
- optimise()
- to_dataset()

References

[]

Examples

```
>>> import os
>>> import colour
>>> from colour.characterisation import SDS_COLOURCHECKERS
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['D65'].copy().align(cmfs.shape)
>>> reflectances = [
...     sd.copy().align(cmfs.shape).values
...     for sd in SDS_COLOURCHECKERS['ColorChecker N Ohta'].values()
... ]
>>> node_tree = NodeTree_Otsu2018(reflectances, cmfs, illuminant)
>>> node_tree.optimise(iterations=2, print_callable=lambda x: x)
>>> dataset = node_tree.to_dataset()
>>> path = os.path.join(colour.__path__[0], 'recovery', 'tests',
...                     'resources', 'ColorChecker_Otsu2018.npz')
>>> dataset.write(path)
>>> dataset = Dataset_Otsu2018()
>>> dataset.read(path)
>>> sd = XYZ_to_sd_Otsu2018(XYZ, cmfs, illuminant, dataset)
...
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.0651341...],
                      [ 370.        ,    0.0651341...],
                      [ 380.        ,    0.0651341...],
```

(continues on next page)

(continued from previous page)

```
                    [ 390.        ,    0.0749684...],
                    [ 400.        ,    0.0815578...],
                    [ 410.        ,    0.0776439...],
                    [ 420.        ,    0.0721897...],
                    [ 430.        ,    0.0649064...],
                    [ 440.        ,    0.0567185...],
                    [ 450.        ,    0.0484685...],
                    [ 460.        ,    0.0409768...],
                    [ 470.        ,    0.0358964...],
                    [ 480.        ,    0.0307857...],
                    [ 490.        ,    0.0270148...],
                    [ 500.        ,    0.0273773...],
                    [ 510.        ,    0.0303157...],
                    [ 520.        ,    0.0331285...],
                    [ 530.        ,    0.0363027...],
                    [ 540.        ,    0.0425987...],
                    [ 550.        ,    0.0513442...],
                    [ 560.        ,    0.0579256...],
                    [ 570.        ,    0.0653850...],
                    [ 580.        ,    0.0929522...],
                    [ 590.        ,    0.1600326...],
                    [ 600.        ,    0.2586159...],
                    [ 610.        ,    0.3701242...],
                    [ 620.        ,    0.4702243...],
                    [ 630.        ,    0.5396261...],
                    [ 640.        ,    0.5737561...],
                    [ 650.        ,    0.590848 ...],
                    [ 660.        ,    0.5935371...],
                    [ 670.        ,    0.5923295...],
                    [ 680.        ,    0.5956326...],
                    [ 690.        ,    0.5982513...],
                    [ 700.        ,    0.6017904...],
                    [ 710.        ,    0.6016419...],
                    [ 720.        ,    0.5996892...],
                    [ 730.        ,    0.6000018...],
                    [ 740.        ,    0.5964443...],
                    [ 750.        ,    0.5868181...],
                    [ 760.        ,    0.5860973...],
                    [ 770.        ,    0.5614878...],
                    [ 780.        ,    0.5289331...]],
                   interpolator=SpragueInterpolator,
                   interpolator_kwargs={},
                   extrapolator=Extrapolator,
                   extrapolator_kwargs={...})
```

__init__(*reflectances*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='D65', ...)*)

Methods

PCA()	Performs the *Principal Component Analysis* (PCA) on the colours data of the node and sets the relevant private attributes accordingly.
__init__(reflectances[, cmfs, illuminant])	
branch_reconstruction_error()	Computes the reconstruction error for an entire branch of the tree, starting from the node, i.e. the reconstruction errors summation for all the leaves in the branch.
find_best_partition()	Finds the best partition for the node.
is_leaf()	Returns whether the node is a leaf.
leaf_reconstruction_error()	Reconstructs the reflectance of the *CIE XYZ* tristimulus values in the colour data of this node using PCA and compares the reconstructed spectrum against the measured spectrum.
msds_to_XYZ(reflectances)	Computes the XYZ tristimulus values of a given reflectance.
optimise([iterations, minimum_cluster_size, ...])	Optimises the tree by repeatedly performing optimal partitioning of the nodes, creating a tree that minimizes the total reconstruction error.
partition_reconstruction_error(axis)	Computes the reconstruction errors summation of the two nodes created by splitting the node with a given partition.
reconstruct(XYZ)	Reconstructs the reflectance for the given *CIE XYZ* tristimulus values.
split(children, partition_axis)	Converts the leaf node into a non-leaf node using given children and partition axis.
to_dataset()	Creates a colour.recovery.Dataset_Otsu2018 class instance based on data stored in the tree.

Attributes

basis_functions	Getter property for the node basis functions.
children	Getter property for the node children.
cmfs	Getter property for the standard observer colour matching functions.
colour_data	Getter property for the node colour data.
id	Getter property for the node id.
illuminant	Getter property for the illuminant.
leaves	Getter property for the node leaves.
mean	Getter property for the node mean distribution.
minimum_cluster_size	Getter property for the minimum cluster size.
partition_axis	Getter property for the node partition axis.
reflectances	Getter property for the reflectances.
tree	Getter property for the node tree.

Smits (1999)

colour.recovery

RGB_to_sd_Smits1999(RGB)	Recovers the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.
SDS_SMITS1999	*Smits (1999)* 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

[]

Examples

```
>>> from colour.colorimetry import (
...     MSDS_CMFS_STANDARD_OBSERVER, SDS_ILLUMINANTS,
...     SpectralShape, sd_to_XYZ_integration
... )
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> RGB = XYZ_to_RGB_Smits1999(XYZ)
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> illuminant = SDS_ILLUMINANTS['E'].copy().align(cmfs.shape)
>>> sd = RGB_to_sd_Smits1999(RGB)
>>> with numpy_print_options(suppress=True):
...     sd
SpectralDistribution([[ 380.        ,    0.0787830...],
                      [ 417.7778    ,    0.0622018...],
                      [ 455.5556    ,    0.0446206...],
                      [ 493.3333    ,    0.0352220...],
                      [ 531.1111    ,    0.0324149...],
                      [ 568.8889    ,    0.0330105...],
                      [ 606.6667    ,    0.3207115...],
```

(continues on next page)

(continued from previous page)

```
                     [ 644.4444    ,    0.3836164...],
                     [ 682.2222    ,    0.3836164...],
                     [ 720.        ,    0.3835649...]],
                    interpolator=LinearInterpolator,
                    interpolator_kwargs={},
                    extrapolator=Extrapolator,
                    extrapolator_kwargs={...})
>>> sd_to_XYZ_integration(sd, cmfs, illuminant) / 100
array([ 0.1894770...,  0.1126470...,  0.0474420...])
```

colour.recovery.SDS_SMITS1999

colour.recovery.**SDS_SMITS1999 = CaseInsensitiveMapping({'white': ..., 'cyan': ..., 'magenta': ..., 'yellow': ..., 'red': ..., 'green': ..., 'blue': ...})**

Smits (1999) spectral distributions.

References

[]

SDS_SMITS1999 : CaseInsensitiveMapping

Colour Temperature

Correlated Colour Temperature

colour

uv_to_CCT(uv[, method])	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.
UV_TO_CCT_METHODS	Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_uv(CCT_D_uv[, method])	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_UV_METHODS	Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.
xy_to_CCT(xy[, method])	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.
XY_TO_CCT_METHODS	Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_xy(CCT[, method])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_XY_METHODS	Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

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', 'Krystek 1985, 'Robertson 1968'}**, Computation method.
- **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.
- **optimisation_kwargs** (dict_like, optional) – {colour.temperature.uv_to_CCT_Krystek1985()}, Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

[], [], [], [], []

Examples

```
>>> import numpy as np
>>> uv = np.array([0.1978, 0.3122])
>>> # Doctests skipping for Python 2.x compatibility.
>>> uv_to_CCT(uv)
array([  6.5074738...e+03,   3.2233460...e-03])
```

colour.UV_TO_CCT_METHODS

colour.**UV_TO_CCT_METHODS = CaseInsensitiveMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Robertson 1968': ..., 'ohno2013': ..., 'robertson1968': ...})**

Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[], [], [], []

UV_TO_CCT_METHODS [CaseInsensitiveMapping] **{'Ohno 2013', 'Krystek 1985, 'Robertson 1968'}**

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

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.
- **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

[], [], [], [], []

Examples

```
>>> import numpy as np
>>> CCT_D_uv = np.array([6507.47380460, 0.00322335])
>>> CCT_to_uv(CCT_D_uv)
array([ 0.1977999...,  0.3121999...])
```

colour.CCT_TO_UV_METHODS

colour.**CCT_TO_UV_METHODS = CaseInsensitiveMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Robertson 1968': ..., 'ohno2013': ..., 'robertson1968': ...})**

Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.

References

- [] : Adobe Systems. (2013). 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. https://www.adobe.com/support/downloads/dng/dng_sdk.html
- [] : Adobe Systems. (2013). Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::xy_coord. https://www.adobe.com/support/downloads/dng/dng_sdk.html
- [] : Herna?ndez-Andre?s, J., Lee, R. L., & Romero, J. (1999). Calculating correlated color temperatures across the entire gamut of daylight and skylight chromaticities. Applied Optics, 38(27), 5703. doi:10.1364/AO.38.005703
- [] : Kang, B., Moon, O., Hong, C., Lee, H., Cho, B., & Kim, Y. (2002). Design of advanced color: Temperature control system for HDTV applications. Journal of the Korean Physical Society, 41(6), 865-871.
- [] : Krystek, M. (1985). An algorithm to calculate correlated colour temperature. Color Research & Application, 10(1), 38-40. doi:10.1002/col.5080100109
- [] : Ohno, Yoshiro. (2014). Practical Use and Calculation of CCT and Duv. LEUKOS, 10(1), 47-55. doi:10.1080/15502724.2014.839020
- [] : Wikipedia. (2001). Approximation. Retrieved June 28, 2014, from http://en.wikipedia.org/wiki/Color_temperature#Approximation
- [] : Wikipedia. (2001). Color temperature. Retrieved June 28, 2014, from http://en.wikipedia.org/wiki/Color_temperature
- [] : Wyszecki, Gu?nther, & Stiles, W. S. (2000). DISTRIBUTION TEMPERATURE, COLOR TEMPERATURE, AND CORRELATED COLOR TEMPERATURE. In Color Science: Concepts and Methods, Quantitative Data and Formulae (pp. 224-229). Wiley. ISBN:978-0-471-39918-6
- [] : Wyszecki, Gu?nther, & Stiles, W. S. (2000). CIE Method of Calculating D-Illuminants. In Color Science: Concepts and Methods, Quantitative Data and Formulae (pp. 145-146). Wiley. ISBN:978-0-471-39918-6

colour.xy_to_CCT

colour.**xy_to_CCT**(*xy*, *method='CIE Illuminant D Series'*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **method** (unicode, optional) – **{'CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'}**, Computation method.
- **optimisation_kwargs** (dict_like, optional) – {colour.temperature.xy_to_CCT_CIE_D(), colour.temperature.xy_to_CCT_Kang2002()}, Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numeric or ndarray

References

[], [], [], [], []

Examples

```
>>> import numpy as np
>>> xy_to_CCT(np.array([0.31270, 0.32900]))
6508.1175148...
>>> xy_to_CCT(np.array([0.31270, 0.32900]), 'Hernandez 1999')
...
6500.7420431...
```

colour.XY_TO_CCT_METHODS

colour.**XY_TO_CCT_METHODS = CaseInsensitiveMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ..., 'Kang 2002': ..., 'McCamy 1992': ..., 'daylight': ..., 'kang2002': ..., 'mccamy1992': ..., 'hernandez1999': ...})**

Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[], [], [], [], []

XY_TO_CCT_METHODS [CaseInsensitiveMapping] **{'McCamy 1992', 'CIE Illuminant D Series, 'Kang 2002', 'Hernandez 1999'}**

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

colour.CCT_to_xy

colour.**CCT_to_xy**(*CCT*, *method='CIE Illuminant D Series'*)

Returns the *CIE 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) – **{'CIE Illuminant D Series', 'Hernandez 1999', 'Kang 2002', 'McCamy 1992'}**, Computation method.
- **optimisation_kwargs** (dict_like, optional) – {colour.temperature.CCT_to_xy_Hernandez1999(), colour.temperature.CCT_to_xy_McCamy1992()}, Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

References

[], [], [], [], []

Examples

```
>>> CCT_to_xy(6504.38938305)
array([ 0.3127077...,  0.3291128...])
>>> CCT_to_xy(6504.38938305, 'Kang 2002')
...
array([ 0.313426 ...,  0.3235959...])
```

colour.CCT_TO_XY_METHODS

colour.**CCT_TO_XY_METHODS = CaseInsensitiveMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ..., 'Kang 2002': ..., 'McCamy 1992': ..., 'daylight': ..., 'kang2002': ..., 'mccamy1992': ..., 'hernandez1999': ...})**

Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

References

[], [], [], [], []

CCT_TO_XY_METHODS [CaseInsensitiveMapping] **{'Kang 2002', 'CIE Illuminant D Series', 'Hernandez 1999', 'McCamy 1992'}**

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

Robertson (1968)

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.
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.

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

[], []

Examples

```
>>> uv = np.array([0.193741375998230, 0.315221043940594])
>>> uv_to_CCT_Robertson1968(uv)
array([  6.5000162...e+03,   8.3333289...e-03])
```

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

[], []

Examples

```
>>> CCT_D_uv = np.array([6500.0081378199056, 0.008333331244225])
>>> CCT_to_uv_Robertson1968(CCT_D_uv)
array([ 0.1937413...,  0.3152210...])
```

Krystek (1985)

`colour.temperature`

uv_to_CCT_Krystek1985(uv[, optimisation_kwargs])	Returns the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.
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.uv_to_CCT_Krystek1985

`colour.temperature.`**`uv_to_CCT_Krystek1985`**(*uv*, *optimisation_kwargs=None*, ***kwargs*)

Returns the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.

Parameters

- **uv** (`array_like`) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **optimisation_kwargs** (`dict_like, optional`) – Parameters for `scipy.optimize.minimize()` definition.
- ****kwargs** (`dict, optional`) – Keywords arguments for deprecation management.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: *Krystek (1985)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates, the current implementation relies on optimization using `scipy.optimize.minimize()` definition and thus has reduced precision and poor performance.

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[]

Examples

```
>>> uv_to_CCT_Krystek1985(np.array([0.20047203, 0.31029290]))
...
6504.3894290...
```

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** (array_like) – 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

[]

Examples

```
>>> CCT_to_uv_Krystek1985(6504.38938305)
array([ 0.2004720...,  0.3102929...])
```

Ohno (2013)

colour.temperature

uv_to_CCT_Ohno2013(uv[, cmfs, start, end, ...])	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates, colour matching functions and temperature range using *Ohno (2013)* method.
CCT_to_uv_Ohno2013(CCT_D_uv[, cmfs])	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.

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* colourspace *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* colourspace *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

[]

Examples

```
>>> from colour.colorimetry import (
...     SPECTRAL_SHAPE_DEFAULT, MSDS_CMFS_STANDARD_OBSERVER)
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> uv = np.array([0.1978, 0.3122])
>>> # Doctests skipping for Python 2.x compatibility.
>>> uv_to_CCT_Ohno2013(uv, cmfs)
array([  6.5074738...e+03,   3.2233460...e-03])
```

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

[]

Examples

```
>>> from colour.colorimetry import (
...     SPECTRAL_SHAPE_DEFAULT, MSDS_CMFS_STANDARD_OBSERVER)
>>> cmfs = (
...     MSDS_CMFS_STANDARD_OBSERVER['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SPECTRAL_SHAPE_DEFAULT)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs)
array([ 0.1977999...,  0.3122004...])
```

McCamy (1992)

colour.temperature

xy_to_CCT_McCamy1992(xy)	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.
CCT_to_xy_McCamy1992(CCT[, optimisation_kwargs])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

colour.temperature.xy_to_CCT_McCamy1992

colour.temperature.**xy_to_CCT_McCamy1992**(*xy*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numeric or ndarray

References

[]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_McCamy1992(xy)
6505.0805913...
```

colour.temperature.CCT_to_xy_McCamy1992

colour.temperature.**CCT_to_xy_McCamy1992**(*CCT*, *optimisation_kwargs=None*, ***kwargs*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.
- **optimisation_kwargs** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Warning: *McCamy (1992)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is not a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications. The current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[]

Examples

```
>>> CCT_to_xy_McCamy1992(6505.0805913074782)
array([ 0.3127...,  0.329...])
```

Hernandez-Andres, Lee and Romero (1999)

colour.temperature

xy_to_CCT_Hernandez1999(xy)	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.
CCT_to_xy_Hernandez1999(CCT[, ...])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al. (1999)* method.

colour.temperature.xy_to_CCT_Hernandez1999

colour.temperature.**xy_to_CCT_Hernandez1999**(*xy*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numeric

References

[]

Examples

```
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_Hernandez1999(xy)
6500.7420431...
```

colour.temperature.CCT_to_xy_Hernandez1999

colour.temperature.**CCT_to_xy_Hernandez1999**(*CCT*, *optimisation_kwargs=None*, ***kwargs*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al. (1999)* method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.
- **optimisation_kwargs** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Warning: *Hernandez-Andres et al. (1999)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is not a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications. The current implementation relies on optimization using `scipy.optimize.minimize()` definition and thus has reduced precision and poor performance.

References

[]

Examples

```
>>> CCT_to_xy_Hernandez1999(6500.7420431786531)
array([ 0.3127...,  0.329...])
```

Kang, Moon, Hong, Lee, Cho and Kim (2002)

`colour.temperature`

`xy_to_CCT_Kang2002`(xy[, optimisation_kwargs])	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al. (2002)* method.
`CCT_to_xy_Kang2002`(CCT)	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al. (2002)* method.

colour.temperature.xy_to_CCT_Kang2002

`colour.temperature.`**`xy_to_CCT_Kang2002`**(*xy*, *optimisation_kwargs=None*, ***kwargs*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al. (2002)* method.

Parameters

- **xy** (`array_like`) – *CIE xy* chromaticity coordinates.
- **optimisation_kwargs** (`dict_like, optional`) – Parameters for `scipy.optimize.minimize()` definition.
- ****kwargs** (`dict, optional`) – Keywords arguments for deprecation management.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: *Kang et al. (2002)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimization using `scipy.optimize.minimize()` definition and thus has reduced precision and poor performance.

References

[]

Examples

```
>>> xy_to_CCT_Kang2002(np.array([0.31342600, 0.32359597]))
...
6504.3893128...
```

colour.temperature.CCT_to_xy_Kang2002

colour.temperature.**CCT_to_xy_Kang2002**(*CCT*)

Returns the *CIE 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 *CIE xy* chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

[]

Examples

```
>>> CCT_to_xy_Kang2002(6504.38938305)
array([ 0.313426 ...,  0.3235959...])
```

CIE Illuminant D Series

colour.temperature

xy_to_CCT_CIE_D(xy[, optimisation_kwargs])	Returns the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.
CCT_to_xy_CIE_D(CCT)	Returns the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

colour.temperature.xy_to_CCT_CIE_D

colour.temperature.**xy_to_CCT_CIE_D**(*xy*, *optimisation_kwargs=None*, ***kwargs*)

Returns the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **optimisation_kwargs** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.
- ****kwargs** (dict, optional) – Keywords arguments for deprecation management.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: The *CIE Illuminant D Series* method does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[]

Examples

```
>>> xy_to_CCT_CIE_D(np.array([0.31270775, 0.32911283]))
...
6504.3895840...
```

colour.temperature.CCT_to_xy_CIE_D

colour.temperature.**CCT_to_xy_CIE_D**(*CCT*)

Returns the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

Parameters **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

[]

Examples

```
>>> CCT_to_xy_CIE_D(6504.38938305)
array([ 0.3127077...,  0.3291128...])
```

Utilities

Common

colour

domain_range_scale(scale)	A context manager and decorator temporarily setting *Colour* domain-range scale.
get_domain_range_scale()	Returns the current *Colour* domain-range scale.
set_domain_range_scale([scale])	Sets the current *Colour* 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.

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('1'):
...     to_domain_1(1)
array(1.0)
>>> with domain_range_scale('Reference'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_1(1)
array(1.0)
>>> with domain_range_scale('1'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_1(1)
array(0.01)
>>> with domain_range_scale('100'):
...     from_range_1(1)
100
```

__init__(*scale*)

Methods

__init__(scale)

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

handle_numpy_errors(**kwargs)	Decorator for handling *Numpy* errors.
ignore_numpy_errors(function)	Wrapper for given function.
raise_numpy_errors(function)	Wrapper for given function.
print_numpy_errors(function)	Wrapper for given function.
warn_numpy_errors(function)	Wrapper for given function.
ignore_python_warnings(function)	Decorator for ignoring *Python* warnings.
batch(iterable[, k])	Returns a batch generator from given iterable.
disable_multiprocessing()	A context manager and decorator temporarily disabling *Colour* multiprocessing.
multiprocessing_pool(*args, **kwargs)	A context manager providing a multiprocessing pool.
is_matplotlib_installed([raise_exception])	Returns if *Matplotlib* is installed and available.
is_networkx_installed([raise_exception])	Returns if *NetworkX* is installed and available.
is_openimageio_installed([raise_exception])	Returns if *OpenImageIO* is installed and available.
is_pandas_installed([raise_exception])	Returns if *Pandas* is installed and available.
is_tqdm_installed([raise_exception])	Returns if *tqdm* is installed and available.
required(*requirements)	A decorator checking if various requirements are satisfied.
is_iterable(a)	Returns if given a variable is iterable.
is_string(a)	Returns if given a variable is a *string* like variable.
is_numeric(a)	Returns if given a variable is a number.
is_integer(a)	Returns if given a variable is an integer under given threshold.
is_sibling(element, mapping)	Returns whether given element type is present in given mapping types.
filter_kwargs(function, **kwargs)	Filters keyword arguments incompatible with the given function signature.
filter_mapping(mapping, filterers[, ...])	Filters given mapping with given filterers.
first_item(a)	Return the first item of an iterable.
to_domain_1(a[, scale_factor, dtype])	Scales given array a to domain '**1**'.
to_domain_10(a[, scale_factor, dtype])	Scales given array a to domain '**10**', used by *Munsell Renotation System*.
to_domain_100(a[, scale_factor, dtype])	Scales given array a to domain '**100**'.
to_domain_degrees(a[, scale_factor, dtype])	Scales given array a to degrees domain.
to_domain_int(a[, bit_depth, dtype])	Scales given array a to int domain.
from_range_1(a[, scale_factor])	Scales given array a from range '**1**'.
from_range_10(a[, scale_factor])	Scales given array a from range '**10**', used by *Munsell Renotation System*.
from_range_100(a[, scale_factor])	Scales given array a from range '**100**'.

continues on next page

Table 3 – continued from previous page

from_range_degrees(a[, scale_factor])	Scales given array a from degrees range.
from_range_int(a[, bit_depth, dtype])	Scales given array a from int range.
copy_definition(definition[, name])	Copies a definition with same code, globals, defaults, closure, and name.

colour.utilities.handle_numpy_errors

colour.utilities.**handle_numpy_errors**(***kwargs*)

Decorator for handling *Numpy* errors.

Parameters ****kwargs** (dict, optional) – Keywords arguments.

Return type object

References

[]

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.

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)))
<generator object batch at 0x...>
```

colour.utilities.disable_multiprocessing

class colour.utilities.**disable_multiprocessing**

A context manager and decorator temporarily disabling *Colour* multiprocessing.

__init__()

Methods

`__init__`()

colour.utilities.multiprocessing_pool

colour.utilities.**multiprocessing_pool**(**args*, ***kwargs*)

A context manager providing a multiprocessing pool.

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))
...
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

colour.utilities.is_matplotlib_installed

colour.utilities.**is_matplotlib_installed**(*raise_exception=False*)

Returns if *Matplotlib* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *Matplotlib* is unavailable.

Returns Is *Matplotlib* installed.

Return type bool

Raises ImportError – If *Matplotlib* is not installed.

colour.utilities.is_networkx_installed

colour.utilities.**is_networkx_installed**(*raise_exception=False*)

Returns if *NetworkX* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *NetworkX* is unavailable.

Returns Is *NetworkX* installed.

Return type bool

Raises ImportError – If *NetworkX* is not installed.

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_tqdm_installed

colour.utilities.**is_tqdm_installed**(*raise_exception=False*)

Returns if *tqdm* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *tqdm* is unavailable.

Returns Is *tqdm* installed.

Return type bool

Raises ImportError – If *tqdm* is not installed.

colour.utilities.required

colour.utilities.**required**(**requirements*)

A decorator checking if various requirements are satisfied.

Parameters ***requirements** (list, optional) – **{'Matplotlib', 'NetworkX', 'OpenImageIO', 'Pandas', 'tqdm'}**, Requirements to check whether they are satisfied.

Return type object

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.
- **`**kwargs`** (`dict, optional`) – Keywords arguments.

Returns Filtered keyword arguments.

Return type dict

Warning: Python 2.7 does not support inspecting the signature of *partial* functions, this could cause unexpected behaviour when using this definition.

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))
```

(continues on next page)

(continued from previous page)

```
(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*)

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')
{u'Element A': <colour.utilities.common.Element object at 0x...>}
>>> # Doctests skip for Python 2.x compatibility.
>>> sorted(filter_mapping(mapping, 'Element.*'))
[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.

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=None*)

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=None*)

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=None*)

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=None*)

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=None*)

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

Warning: The scale conversion of a variable happens in-place, i.e. a will be mutated!

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

Warning: The scale conversion of a variable happens in-place, i.e. a will be mutated!

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

Warning: The scale conversion of a variable happens in-place, i.e. a will be mutated!

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.

Warning: The scale conversion of a variable happens in-place, i.e. a will be mutated!

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)
0.0027777...
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_degrees(1)
0.2777777...
```

colour.utilities.from_range_int

colour.utilities.**from_range_int**(*a*, *bit_depth=8*, *dtype=None*)

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

Warning: The scale conversion of a variable happens in-place, i.e. a will be mutated!

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)
array(0.0039215...)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_int(1)
array(0.3921568...)
```

colour.utilities.copy_definition

colour.utilities.**copy_definition**(*definition*, *name=None*)

Copies a definition with same code, globals, defaults, closure, and name.

Parameters

- **definition** (callable) – Definition to be copied.
- **name** (unicode, optional) – Optional definition copy name.

Returns Definition copy.

Return type callable

Array

colour.utilities

as_array(a[, dtype])	Converts given a variable to *ndarray* using given type.
as_int_array(a[, dtype])	Converts given a variable to *ndarray* using given type.
as_float_array(a[, dtype])	Converts given a variable to *ndarray* using given type.
as_numeric(a[, dtype])	Converts given a variable to *numeric*.
as_int(a[, dtype])	Attempts to converts given a variable to *int* using given type.
as_float(a[, dtype])	Converts given a variable to *numeric* using given type.
set_float_precision([dtype])	Sets *Colour* float precision by setting `colour.constant.DEFAULT_FLOAT_DTYPE` attribute with given type wherever the attribute is imported.
set_int_precision([dtype])	Sets *Colour* integer precision by setting `colour.constant.DEFAULT_INT_DTYPE` attribute with given type wherever the attribute is imported.
as_namedtuple(a, named_tuple)	Converts given a variable to given *namedtuple* class instance.
closest_indexes(a, b)	Returns the a variable closest element indexes to reference b variable elements.
closest(a, b)	Returns the a variable closest elements to reference b variable elements.
normalise_maximum(a[, axis, factor, clip])	Normalises given *array_like* a variable values by a variable maximum value and optionally clip them between.
interval(distribution[, unique])	Returns the interval size of given distribution.
is_uniform(distribution)	Returns if given distribution is uniform.
in_array(a, b[, tolerance])	Tests whether each element of an array is also present in a second array within given tolerance.
tstack(a[, dtype])	Stacks arrays in sequence along the last axis (tail).
tsplit(a[, dtype])	Splits arrays in sequence along the last axis (tail).
row_as_diagonal(a)	Returns the per row diagonal matrices of the given array.
vector_dot(m, v)	Convenient wrapper around `np.einsum()` with the following subscripts: '*...ij,...j->...i*'.
matrix_dot(a, b)	Convenient wrapper around `np.einsum()` with the following subscripts: '*...ij,...jk->...ik*'.
orient(a, orientation)	Orient given array according to given `orientation` value.
centroid(a)	Computes the centroid indexes of given a array.
linear_conversion(a, old_range, new_range)	Performs a simple linear conversion of given array between the old and new ranges.
lerp(a, b, c)	Performs a simple linear interpolation between given array a and array b using c value.
fill_nan(a[, method, default])	Fills given array NaNs according to given method.
ndarray_write(a)	A context manager setting given array writeable to perform an operation and then read-only.
zeros(shape[, dtype, order])	Simple wrapper around `np.zeros()` definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.
ones(shape[, dtype, order])	Simple wrapper around `np.ones()` definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.
full(shape, fill_value[, dtype, order])	Simple wrapper around `np.full()` definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.

colour.utilities.as_array

colour.utilities.**as_array**(*a*, *dtype=None*)

Converts given a variable to *ndarray* using given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion, default to the type defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

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*, *dtype=None*)

Converts given a variable to *ndarray* using given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion, default to the type defined by the colour.constant.DEFAULT_INT_DTYPE attribute.

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*, *dtype=None*)

Converts given a variable to *ndarray* using given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion, default to the type defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

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=None*)

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, default to the type defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.

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*, *dtype=None*)

Attempts to converts given a variable to *int* using given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion, default to the type defined by the 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.

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*, *dtype=None*)

Converts given a variable to *numeric* using given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion, default to the type defined by the 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_FLOAT_DTYPE attribute.

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.set_float_precision

colour.utilities.**set_float_precision**(*dtype=<class 'numpy.float64'>*)

Sets *Colour* float precision by setting colour.constant.DEFAULT_FLOAT_DTYPE attribute with given type wherever the attribute is imported.

Parameters **dtype** (object) – Type to set colour.constant.DEFAULT_FLOAT_DTYPE with.

Warning: Changing float precision might result in various *Colour* functionality breaking entirely: https://github.com/numpy/numpy/issues/6860. With great power comes great responsibility.

Notes

- It is possible to define the float precision at import time by setting the *COLOUR_SCIENCE__FLOAT_PRECISION* environment variable, for example *set COLOUR_SCIENCE__FLOAT_PRECISION=float32*.
- Some definition returning a single-scalar ndarray might not honour the given float precision: https://github.com/numpy/numpy/issues/16353

Examples

```
>>> as_float_array(np.ones(3)).dtype
dtype('float64')
>>> set_float_precision(np.float16)
>>> as_float_array(np.ones(3)).dtype
dtype('float16')
>>> set_float_precision(np.float64)
>>> as_float_array(np.ones(3)).dtype
dtype('float64')
```

colour.utilities.set_int_precision

colour.utilities.**set_int_precision**(*dtype=<class 'numpy.int64'>*)

Sets *Colour* integer precision by setting `colour.constant.DEFAULT_INT_DTYPE` attribute with given type wherever the attribute is imported.

Parameters **dtype** (`object`) – Type to set `colour.constant.DEFAULT_INT_DTYPE` with.

Notes

- It is possible to define the int precision at import time by setting the *COLOUR_SCIENCE__INT_PRECISION* environment variable, for example *set COLOUR_SCIENCE__INT_PRECISION=int32*.

Warning: This definition is mostly given for consistency purposes with `colour.utilities.set_float_precision()` definition but contrary to the latter, changing integer precision will almost certainly completely break *Colour*. With great power comes great responsibility.

Examples

```
>>> as_int_array(np.ones(3)).dtype
dtype('int64')
>>> set_int_precision(np.int32)
>>> as_int_array(np.ones(3)).dtype
dtype('int32')
>>> set_int_precision(np.int64)
>>> as_int_array(np.ones(3)).dtype
dtype('int64')
```

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 math:*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)
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)
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=2.220446049250313e-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

[]

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=None*)

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*, default to the type defined by colour.constant.DEFAULT_FLOAT_DTYPE attribute.

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.],
```

(continues on next page)

(continued from previous page)

```
       [ 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=None*)

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*, default to the type defined by colour.constant.DEFAULT_FLOAT_DTYPE attribute.

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],
```

(continues on next page)

(continued from previous page)

```
...       [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.]]])
```

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.

Return type ndarray

References

[]

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]],

       [[ 0.30851087,  0.        ,  0.        ],
        [ 0.        ,  0.37131459,  0.        ],
        [ 0.        ,  0.        ,  0.16274825]],

       [[ 0.71061831,  0.        ,  0.        ],
        [ 0.        ,  0.67718718,  0.        ],
        [ 0.        ,  0.        ,  0.09562581]],

       [[ 0.71588836,  0.        ,  0.        ],
        [ 0.        ,  0.76772047,  0.        ],
        [ 0.        ,  0.        ,  0.15476079]],

       [[ 0.92985142,  0.        ,  0.        ],
        [ 0.        ,  0.22263399,  0.        ],
        [ 0.        ,  0.        ,  0.88027331]]])
```

colour.utilities.vector_dot

colour.utilities.**vector_dot**(*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.

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))
>>> vector_dot(m, v)
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.matrix_dot

colour.utilities.**matrix_dot**(*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.
- **dtype** (object) – Type to use for conversion, default to the type defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

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
>>> matrix_dot(a, b)
array([[[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 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
```

colour.utilities.zeros

colour.utilities.**zeros**(*shape*, *dtype=None*, *order='C'*)

Simple wrapper around np.zeros() definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.

Parameters

- **shape** (int or array_like) – Shape of the new array, e.g., (2, 3) or 2.
- **dtype** (object) – Type to use for conversion, default to the type defined by the colour.constant.DEFAULT_FLOAT_DTYPE attribute.
- **order** (unicode, optional) – {'C', 'F'}, Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and type, filled with zeros.

Return type ndarray

Examples

```
>>> zeros(3)
array([ 0.,  0.,  0.])
```

colour.utilities.ones

colour.utilities.**ones**(*shape*, *dtype=None*, *order='C'*)

Simple wrapper around `np.ones()` definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.

Parameters

- **shape** (`int or array_like`) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **dtype** (`object`) – Type to use for conversion, default to the type defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.
- **order** (`unicode, optional`) – {'C', 'F'}, Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and type, filled with ones.

Return type ndarray

Examples

```
>>> ones(3)
array([ 1.,  1.,  1.])
```

colour.utilities.full

colour.utilities.**full**(*shape*, *fill_value*, *dtype=None*, *order='C'*)

Simple wrapper around `np.full()` definition to create arrays with the active type defined by the:attr:*colour.constant.DEFAULT_FLOAT_DTYPE* attribute.

Parameters

- **shape** (`int or array_like`) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **fill_value** (`numeric`) – Fill value.
- **dtype** (`object`) – Type to use for conversion, default to the type defined by the `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.
- **order** (`unicode, optional`) – {'C', 'F'}, Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns Array of given shape and type, filled with given value.

Return type ndarray

Examples

```
>>> ones(3)
array([ 1.,  1.,  1.])
```

Metrics

colour.utilities

metric_mse(a, b[, axis])	Computes the mean squared error (MSE) or mean squared deviation (MSD) between given *array_like* a and b variables.
metric_psnr(a, b[, max_a, axis])	Computes the peak signal-to-noise ratio (PSNR) between given *array_like* a and b variables.

colour.utilities.metric_mse

colour.utilities.**metric_mse**(*a*, *b*, *axis=None*)

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.
- **axis** (None or int or tuple of ints, optional) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.

Returns Mean squared error (MSE).

Return type float

References

[]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_mse(a, b)
0.0012714...
```

colour.utilities.metric_psnr

colour.utilities.**metric_psnr**(*a*, *b*, *max_a=1*, *axis=None*)

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.
- **axis** (None or int or tuple of ints, optional) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.

Returns Peak signal-to-noise ratio (PSNR).

Return type float

References

[]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_psnr(a, b)
28.9568515...
```

Data Structures

colour.utilities

CaseInsensitiveMapping([data])	Implements a case-insensitive mutable mapping / *dict* object.
LazyCaseInsensitiveMapping([data])	Implements a lazy case-insensitive mutable mapping / *dict* object by inheriting from colour.utilities.CaseInsensitiveMapping class.
Lookup	Extends *dict* type to provide a lookup by value(s).
Structure(*args, **kwargs)	Defines a dict-like object allowing to access key values using dot syntax.

colour.utilities.CaseInsensitiveMapping

class colour.utilities.**CaseInsensitiveMapping**(*data=None*, ***kwargs*)

Bases: collections.abc.MutableMapping

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.
- ****kwargs** (dict, optional) – Key / Value pairs to store into the mapping at initialisation.

Attributes

- data

Methods

- __init__()
- __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

[]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

__init__(*data=None*, ***kwargs*)

property data

Getter property for the data.

Returns Data.

Return type dict

__setitem__(*item*, *value*)

Sets given item with given value.

The item is stored as lower in the mapping while the original name and its value are stored together as the value in a *tuple*:

{"item.lower()": ("item", value)}

Parameters

- **item** (object) – Attribute.
- **value** (object) – Value.

__getitem__(*item*)

Returns the value of given item.

The item value is retrieved using its lower name in the mapping.

Parameters **item** (unicode) – Item name.

Returns Item value.

Return type object

__delitem__(*item*)

Deletes the item with given name.

The item is deleted from the mapping using its lower name.

Parameters **item** (unicode) – Item name.

__contains__(*item*)

Returns if the mapping contains given item.

Parameters **item** (unicode) – Item name.

Returns Is item in mapping.

Return type bool

__iter__()

Iterates over the items names in the mapping.

The item names returned are the original input ones.

Returns Item names.

Return type generator

__len__()

Returns the items count.

Returns Items count.

Return type int

__eq__(*item*)

Returns the equality with given object.

Parameters **item** – Object item.

Returns Equality.

Return type bool

__ne__(*item*)

Returns the inequality with given object.

Parameters **item** – Object item.

Returns Inequality.

Return type bool

__repr__()

Returns an ellipsis string representation of the case-insensitive mutable mapping for documentation purposes.

Returns Ellipsis string representation.

Return type unicode

copy()

Returns a copy of the mapping.

Returns Mapping copy.

Return type *CaseInsensitiveMapping*

Notes

- The colour.utilities.CaseInsensitiveMapping class copy returned is a simple *copy* not a *deepcopy*.

lower_items()

Iterates over the lower items names.

Returns Lower item names.

Return type generator

__hash__ = None

__weakref__

list of weak references to the object (if defined)

colour.utilities.LazyCaseInsensitiveMapping

class colour.utilities.**LazyCaseInsensitiveMapping**(*data=None*, ***kwargs*)

Bases: colour.utilities.data_structures.CaseInsensitiveMapping

Implements a lazy case-insensitive mutable mapping / *dict* object by inheriting from colour.utilities.CaseInsensitiveMapping class.

Allows lazy 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. The lazy retrieval is performed as follows: If the value is a callable, then it is evaluated and its return value is stored in place of the current value.

Parameters

- **data** (dict) – *dict* of data to store into the mapping at initialisation.
- ****kwargs** (dict, optional) – Key / Value pairs to store into the mapping at initialisation.

Methods

- `__getitem__()`

Warning: The keys are expected to be unicode or string-like objects.

Examples

```
>>> def callable_a():
...     print(2)
...     return 2
>>> methods = LazyCaseInsensitiveMapping(
...     {'McCamy': 1, 'Hernandez': callable_a})
>>> methods['mccamy']
1
>>> methods['hernandez']
2
2
```

__getitem__(*item*)

Returns the value of given item.

The item value is retrieved using its lower name in the mapping. If the value is a callable, then it is evaluated and its return value is stored in place of the current value.

Parameters **item** (unicode) – Item name.

Returns Item value.

Return type object

colour.utilities.Lookup

class colour.utilities.**Lookup**

Bases: dict

Extends *dict* type to provide a lookup by value(s).

Methods

- `keys_from_value()`
- `first_key_from_value()`

References

[]

Examples

```
>>> person = Lookup(first_name='John', last_name='Doe', gender='male')
>>> person.first_key_from_value('John')
'first_name'
>>> persons = Lookup(John='Doe', Jane='Doe', Luke='Skywalker')
>>> sorted(persons.keys_from_value('Doe'))
['Jane', 'John']
```

keys_from_value(*value*)

Gets the keys with given value.

Parameters **value** (object) – Value.

Returns Keys.

Return type object

first_key_from_value(*value*)

Gets the first key with given value.

Parameters **value** (object) – Value.

Returns Key.

Return type object

__weakref__

list of weak references to the object (if defined)

colour.utilities.Structure

class colour.utilities.**Structure**(**args*, ***kwargs*)

Bases: dict

Defines a dict-like object allowing to access key values using dot syntax.

Parameters

- ***args** (list, optional) – Arguments.
- ****kwargs** (dict, optional) – Key / Value pairs.

Methods

- __init__()

References

[]

Examples

```
>>> person = Structure(first_name='John', last_name='Doe', gender='male')
>>> # Doctests skip for Python 2.x compatibility.
>>> person.first_name
'John'
>>> sorted(person.keys())
['first_name', 'gender', 'last_name']
>>> # Doctests skip for Python 2.x compatibility.
>>> person['gender']
'male'
```

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

__weakref__
list of weak references to the object (if defined)

Verbose

colour.utilities

message_box(message[, width, padding, ...])	Prints a message inside a box.
warning(*args, **kwargs)	Issues a warning.
filter_warnings([colour_runtime_warnings, ...])	Filters *Colour* and also optionally overall Python warnings.
suppress_warnings([colour_runtime_warnings, ...])	A context manager filtering *Colour* and also optionally overall Python warnings.
numpy_print_options(*args, **kwargs)	A context manager implementing context changes to *Numpy* print behaviour.
describe_environment([runtime_packages, ...])	Describes *Colour* running environment, i.e. interpreter, runtime and development packages.

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.                                                  *
*                                                                         *
===========================================================================
True
```

colour.utilities.warning

colour.utilities.**warning**(**args*, ***kwargs*)

Issues a warning.

Parameters

- ***args** (list, optional) – Arguments.
- ****kwargs** (dict, optional) – Keywords arguments.

Returns Definition success.

Return type bool

Examples

```
>>> warning('This is a warning!')
```

colour.utilities.filter_warnings

colour.utilities.**filter_warnings**(*colour_runtime_warnings=None*, *colour_usage_warnings=None*, *colour_warnings=None*, *python_warnings=None*)

Filters *Colour* and also optionally overall Python warnings.

The possible values for all the actions, i.e. each argument, are as follows:

- *None* (No action is taken)
- *True* (*ignore*)
- *False* (*default*)
- *error*
- *ignore*
- *always*
- *default*
- *module*
- *once*

Parameters

- **colour_runtime_warnings** (bool or unicode, optional) – Whether to filter *Colour* runtime warnings according to the action value.
- **colour_usage_warnings** (bool or unicode, optional) – Whether to filter *Colour* usage warnings according to the action value.
- **colour_warnings** (bool or unicode, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings according to the action value.
- **python_warnings** (bool or unicode, optional) – Whether to filter *Python* warnings according to the action value.

Examples

Filtering *Colour* runtime warnings:

```
>>> filter_warnings(colour_runtime_warnings=True)
```

Filtering *Colour* usage warnings:

```
>>> filter_warnings(colour_usage_warnings=True)
```

Filtering *Colour* warnings:

```
>>> filter_warnings(colour_warnings=True)
```

Filtering all the *Colour* and also Python warnings:

```
>>> filter_warnings(python_warnings=True)
```

Enabling all the *Colour* and Python warnings:

```
>>> filter_warnings(*[False] * 4)
```

Enabling all the *Colour* and Python warnings using the *default* action:

```
>>> filter_warnings(*['default'] * 4)
```

Setting back the default state:

```
>>> filter_warnings(colour_runtime_warnings=True)
```

colour.utilities.suppress_warnings

colour.utilities.**suppress_warnings**(*colour_runtime_warnings=None*, *colour_usage_warnings=None*, *colour_warnings=None*, *python_warnings=None*)

A context manager filtering *Colour* and also optionally overall Python warnings.

The possible values for all the actions, i.e. each argument, are as follows:

- *None* (No action is taken)
- *True* (*ignore*)
- *False* (*default*)
- *error*
- *ignore*
- *always*
- *default*
- *module*
- *once*

Parameters

- **colour_runtime_warnings** (bool or unicode, optional) – Whether to filter *Colour* runtime warnings according to the action value.
- **colour_usage_warnings** (bool or unicode, optional) – Whether to filter *Colour* usage warnings according to the action value.
- **colour_warnings** (bool or unicode, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings according to the action value.
- **python_warnings** (bool or unicode, optional) – Whether to filter *Python* warnings according to the action value.

colour.utilities.numpy_print_options

colour.utilities.**numpy_print_options**(**args*, ***kwargs*)

A context manager implementing context changes to *Numpy* print behaviour.

Parameters

- ***args** (list, optional) – Arguments.
- ****kwargs** (dict, optional) – Keywords arguments.

Examples

```
>>> np.array([np.pi])
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*, *extras_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.
- **extras_packages** (bool, optional) – Whether to return the extras packages versions.
- **print_environment** (bool, optional) – Whether to print the environment.
- **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)
================================================================================
*                                                                              *
*   Interpreter :                                                              *
*       python : 3.7.4 (default, Sep  7 2019, 18:27:02)                        *
*                [Clang 10.0.1 (clang-1001.0.46.4)]                            *
*                                                                              *
*   colour-science.org :                                                       *
*       colour : v0.3.13-293-gecf1dc8a                                         *
*                                                                              *
*   Runtime :                                                                  *
*       imageio : 2.6.1                                                        *
*       numpy : 1.17.2                                                         *
*       scipy : 1.3.1                                                          *
*       six : 1.12.0                                                           *
*       pandas : 0.24.2                                                        *
*       matplotlib : 3.0.3                                                     *
*       networkx : 2.3                                                         *
*       pygraphviz : 1.5                                                       *
*                                                                              *
================================================================================
>>> environment = describe_environment(True, True, True, width=75)
...
================================================================================
*                                                                              *
*   Interpreter :                                                              *
*       python : 3.7.4 (default, Sep  7 2019, 18:27:02)                        *
*                [Clang 10.0.1 (clang-1001.0.46.4)]                            *
*                                                                              *
*   colour-science.org :                                                       *
*       colour : v0.3.13-293-gecf1dc8a                                         *
*                                                                              *
*   Runtime :                                                                  *
*       imageio : 2.6.1                                                        *
*       numpy : 1.17.2                                                         *
*       scipy : 1.3.1                                                          *
*       six : 1.12.0                                                           *
*       pandas : 0.24.2                                                        *
*       matplotlib : 3.0.3                                                     *
*       networkx : 2.3                                                         *
*       pygraphviz : 1.5                                                       *
*                                                                              *
*   Development :                                                              *
*       biblib-simple : 0.1.1                                                  *
*       coverage : 4.5.4                                                       *
*       coveralls : 1.8.2                                                      *
*       flake8 : 3.7.8                                                         *
*       invoke : 1.3.0                                                         *
*       jupyter : 1.0.0                                                        *
*       mock : 3.0.5                                                           *
*       nose : 1.3.7                                                           *
*       pre-commit : 1.18.3                                                    *
*       pytest : 5.2.1                                                         *
*       restructuredtext-lint : 1.3.0                                          *
*       sphinx : 2.2.0                                                         *
```

(continues on next page)

(continued from previous page)

```
*       sphinx_rtd_theme : 0.4.3                                              *
*       sphinxcontrib-bibtex : 1.0.0                                          *
*       toml : 0.10.0                                                         *
*       twine : 1.15.0                                                        *
*       yapf : 0.23.0                                                         *
*                                                                             *
*   Extras :                                                                  *
*       ipywidgets : 7.5.1                                                    *
*       notebook : 6.0.1                                                      *
*                                                                             *
===============================================================================
```

Ancillary Objects

`colour.utilities`

ColourWarning	This is the base class of *Colour* warnings.
ColourUsageWarning	This is the base class of *Colour* usage warnings.
ColourRuntimeWarning	This is the base class of *Colour* runtime warnings.

colour.utilities.ColourWarning

class colour.utilities.**ColourWarning**

Bases: Warning

This is the base class of *Colour* warnings. It is a subclass of Warning class.

__weakref__

list of weak references to the object (if defined)

colour.utilities.ColourUsageWarning

class colour.utilities.**ColourUsageWarning**

Bases: Warning

This is the base class of *Colour* usage warnings. It is a subclass of colour.utilities.ColourWarning class.

__weakref__

list of weak references to the object (if defined)

colour.utilities.ColourRuntimeWarning

class colour.utilities.**ColourRuntimeWarning**

Bases: Warning

This is the base class of *Colour* runtime warnings. It is a subclass of colour.utilities.ColourWarning class.

__weakref__

list of weak references to the object (if defined)

Colour Volume

- *Optimal Colour Stimuli - MacAdam Limits*
- *Mesh Volume*
- *Pointer's Gamut*
- *RGB Volume*
- *Visible Spectrum*

Optimal Colour Stimuli - MacAdam Limits

`colour`

`is_within_macadam_limits`(xyY, illuminant[, ...])	Returns if given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.
`OPTIMAL_COLOUR_STIMULI_ILLUMINANTS`	Illuminants *Optimal Colour Stimuli*.

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.OPTIMAL_COLOUR_STIMULI_ILLUMINANTS

colour.**OPTIMAL_COLOUR_STIMULI_ILLUMINANTS = CaseInsensitiveMapping({'A': ..., 'C': ..., 'D65': ...})**

Illuminants *Optimal Colour Stimuli*.

References

[]

OPTIMAL_COLOUR_STIMULI_ILLUMINANTS [CaseInsensitiveMapping] **{'A', 'C', 'D65'}**

Mesh Volume

colour

is_within_mesh_volume(points, mesh[, tolerance])	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

is_within_pointer_gamut(XYZ[, tolerance])	Returns if given *CIE XYZ* 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

RGB_colourspace_limits(colourspace[, illuminant])	Computes given *RGB* colourspace volume limits in *CIE L*a*b** colourspace.
RGB_colourspace_pointer_gamut_coverage_MonteCarlo(...)	Returns given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.
RGB_colourspace_visible_spectrum_coverage_MonteCarlo(...)	Returns given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.
RGB_colourspace_volume_MonteCarlo(colourspace)	Performs given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.
RGB_colourspace_volume_coverage_MonteCarlo(...)	Returns given *RGB* 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 *CIE L*a*b** colourspace.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the volume of.
- **illuminant** (array_like, optional) – *CIE L*a*b** colourspace *illuminant* chromaticity coordinates.

Returns *RGB* colourspace volume limits.

Return type ndarray

Examples

```
>>> from colour.models import RGB_COLOURSPACE_sRGB as sRGB
>>> RGB_colourspace_limits(sRGB)
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 RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_pointer_gamut_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
81...
```

colour.RGB_colourspace_visible_spectrum_coverage_MonteCarlo

colour.**RGB_colourspace_visible_spectrum_coverage_MonteCarlo**(*colourspace*, *samples=10000000.0*, *random_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 RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_visible_spectrum_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
46...
```

colour.RGB_colourspace_volume_MonteCarlo

colour.**RGB_colourspace_volume_MonteCarlo**(*colourspace, samples=10000000.0, limits=array([[0., 100.], [-150., 150.], [-150., 150.]]), illuminant_Lab=array([0.3127, 0.329]), chromatic_adaptation_method='CAT02', random_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) – *CIE L*a*b** colourspace volume.
- **illuminant_Lab** (array_like, optional) – *CIE L*a*b** colourspace *illuminant* chromaticity coordinates.
- **chromatic_adaptation_method** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02 Brill 2008', 'Bianco 2010', 'Bianco PC 2010'}**, *Chromatic adaptation* method.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples within the *CIE L*a*b** 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 RGB_COLOURSPACE_sRGB 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)
...
8...
```

colour.RGB_colourspace_volume_coverage_MonteCarlo

colour.**RGB_colourspace_volume_coverage_MonteCarlo**(*colourspace*, *coverage_sampler*, *samples=10000000.0*, *random_generator=<function random_triplet_generator>*, *random_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 RGB_COLOURSPACE_sRGB as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_volume_coverage_MonteCarlo(
...     sRGB, is_within_pointer_gamut, 10e3, random_state=prng)
...
81...
```

Visible Spectrum

colour

is_within_visible_spectrum(XYZ[, cmfs, ...])	Returns if given *CIE XYZ* tristimulus values are within visible spectrum volume / given colour matching functions volume.

colour.is_within_visible_spectrum

colour.**is_within_visible_spectrum**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *tolerance=None*, ***kwargs*)

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.
- **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.
- ****kwargs** (dict, optional) – {colour.msds_to_XYZ()}, Please refer to the documentation of the previously listed definition.

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

generate_pulse_waves(bins)	Generates the pulse waves of given number of bins necessary to totally stimulate the colour matching functions.
XYZ_outer_surface([cmfs, illuminant])	Generates the *CIE XYZ* colourspace outer surface for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* 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 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

[], []

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**(*cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, ***kwargs*)

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

- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- ****kwargs** (dict, optional) – {colour.msds_to_XYZ()}, Please refer to the documentation of the previously listed definition.

Returns Outer surface *CIE XYZ* tristimulus values.

Return type ndarray

References

[], []

Examples

```
>>> from colour.colorimetry import SPECTRAL_SHAPE_DEFAULT
>>> shape = SpectralShape(
...     SPECTRAL_SHAPE_DEFAULT.start, SPECTRAL_SHAPE_DEFAULT.end, 84)
>>> cmfs = MSDS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> XYZ_outer_surface(cmfs.copy().align(shape))
array([[  0.0000000...e+00,   0.0000000...e+00,   0.0000000...e+00],
       [  9.6361381...e-05,   2.9056776...e-06,   4.4961226...e-04],
       [  2.5910529...e-01,   2.1031298...e-02,   1.3207468...e+00],
       [  1.0561021...e-01,   6.2038243...e-01,   3.5423571...e-02],
       [  7.2647980...e-01,   3.5460869...e-01,   2.1005149...e-04],
       [  1.0971874...e-02,   3.9635453...e-03,   0.0000000...e+00],
       [  3.0792572...e-05,   1.1119762...e-05,   0.0000000...e+00],
       [  2.5920165...e-01,   2.1034203...e-02,   1.3211965...e+00],
       [  3.6471551...e-01,   6.4141373...e-01,   1.3561704...e+00],
       [  8.3209002...e-01,   9.7499113...e-01,   3.5633622...e-02],
       [  7.3745167...e-01,   3.5857224...e-01,   2.1005149...e-04],
       [  1.1002667...e-02,   3.9746651...e-03,   0.0000000...e+00],
       [  1.2715395...e-04,   1.4025439...e-05,   4.4961226...e-04],
       [  3.6481187...e-01,   6.4141663...e-01,   1.3566200...e+00],
       [  1.0911953...e+00,   9.9602242...e-01,   1.3563805...e+00],
       [  8.4306189...e-01,   9.7895467...e-01,   3.5633622...e-02],
       [  7.3748247...e-01,   3.5858336...e-01,   2.1005149...e-04],
       [  1.1099028...e-02,   3.9775708...e-03,   4.4961226...e-04],
       [  2.5923244...e-01,   2.1045323...e-02,   1.3211965...e+00],
       [  1.0912916...e+00,   9.9602533...e-01,   1.3568301...e+00],
       [  1.1021671...e+00,   9.9998597...e-01,   1.3563805...e+00],
       [  8.4309268...e-01,   9.7896579...e-01,   3.5633622...e-02],
       [  7.3757883...e-01,   3.5858626...e-01,   6.5966375...e-04],
       [  2.7020432...e-01,   2.5008868...e-02,   1.3211965...e+00],
       [  3.6484266...e-01,   6.4142775...e-01,   1.3566200...e+00],
       [  1.1022635...e+00,   9.9998888...e-01,   1.3568301...e+00],
       [  1.1021979...e+00,   9.9999709...e-01,   1.3563805...e+00],
       [  8.4318905...e-01,   9.7896870...e-01,   3.6083235...e-02],
       [  9.9668412...e-01,   3.7961756...e-01,   1.3214065...e+00],
       [  3.7581454...e-01,   6.4539130...e-01,   1.3566200...e+00],
       [  1.0913224...e+00,   9.9603645...e-01,   1.3568301...e+00],
       [  1.1022943...e+00,   1.0000000...e+00,   1.3568301...e+00]])
```

Indices and tables

- genindex
- search

4.3.1.5 Bibliography

Indirect References

Some extra references used in the codebase but not directly part of the public api:

- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []
- []

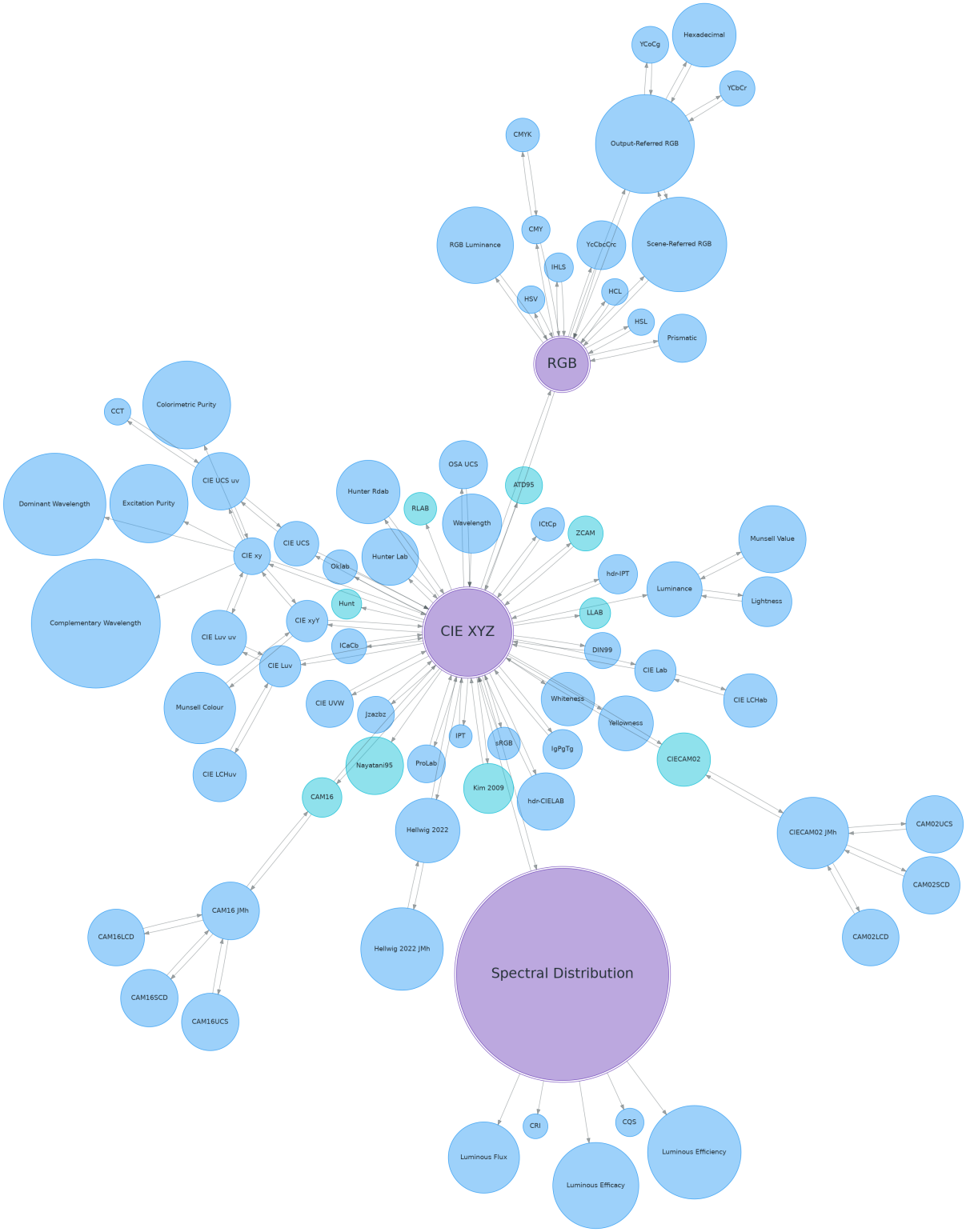
4.4 5.4 Examples

Most of the objects are available from the `colour` namespace:

```
>>> import colour
```

4.4.1 5.4.1 Automatic Colour Conversion Graph - `colour.graph`

Starting with version *0.3.14*, **Colour** implements an automatic colour conversion graph enabling easier colour conversions.




```
>>> sd = colour.SDS_COLOURCHECKERS['ColorChecker N Ohta']['dark skin']
>>> colour.convert(sd, 'Spectral Distribution', 'sRGB', verbose={'mode': 'Short'})
```

```
================================================================================
*                                                                              *
*    [ Conversion Path ]                                                       *
*                                                                              *
*    "sd_to_XYZ" --> "XYZ_to_sRGB"                                             *
```

(continues on next page)

(continued from previous page)

```
*                                                                               *
================================================================================
array([ 0.45675795,  0.30986982,  0.24861924])
```

```
>>> illuminant = colour.SDS_ILLUMINANTS['FL2']
>>> colour.convert(sd, 'Spectral Distribution', 'sRGB', sd_to_XYZ={'illuminant':␣
↪illuminant})
array([ 0.47924575,  0.31676968,  0.17362725])
```

4.4.2 5.4.2 Chromatic Adaptation - colour.adaptation

```
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> D65 = colour.CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
>>> A = colour.CCS_ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['A']
>>> colour.chromatic_adaptation(
...     XYZ, colour.xy_to_XYZ(D65), colour.xy_to_XYZ(A))
array([ 0.2533053 ,  0.13765138,  0.01543307])
>>> sorted(colour.CHROMATIC_ADAPTATION_METHODS)
['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries']
```

4.4.3 5.4.3 Algebra - colour.algebra

4.4.3.1 5.4.3.1 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])
```

4.4.3.2 5.4.3.2 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])
```

4.4.4 5.4.4 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)
CAM_Specification_CIECAM02(J=34.434525727858997, C=67.365010921125915, h=22.
↪279164147957076, s=62.814855853327131, Q=177.47124941102123, M=70.024939419291385, H=2.
↪689608534423904, HC=None)
```

4.4.5 5.4.5 Colour Blindness - colour.blindness

```
>>> import numpy as np
>>> cmfs = colour.LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> colour.msds_cmfs_anomalous_trichromacy_Machado2009(cmfs, np.array([15, 0, 0]))[450]
array([ 0.08912884,  0.0870524 ,  0.955393  ])
>>> primaries = colour.MSDS_DISPLAY_PRIMARIES['Apple Studio Display']
>>> d_LMS = (15, 0, 0)
>>> colour.matrix_anomalous_trichromacy_Machado2009(cmfs, primaries, d_LMS)
array([[-0.27774652,  2.65150084, -1.37375432],
       [ 0.27189369,  0.20047862,  0.52762768],
       [ 0.00644047,  0.25921579,  0.73434374]])
```

4.4.6 5.4.6 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([ 0.1806237 ,  0.07234791,  0.07848845])
>>> sorted(colour.COLOUR_CORRECTION_METHODS)
['Cheung 2004', 'Finlayson 2015', 'Vandermonde']
```

4.4.7 5.4.7 ACES Input Transform - colour characterisation

```
>>> sensitivities = colour.MSDS_CAMERA_SENSITIVITIES['Nikon 5100 (NPL)']
>>> illuminant = colour.SDS_ILLUMINANTS['D55']
>>> colour.matrix_idt(sensitivities, illuminant)
array([[ 0.46579991,  0.13409239,  0.01935141],
       [ 0.01786094,  0.77557292, -0.16775555],
       [ 0.03458652, -0.16152926,  0.74270359]])
```

4.4.8 5.4.8 Colorimetry - colour.colorimetry

4.4.8.1 5.4.8.1 Spectral Computations

```
>>> colour.sd_to_XYZ(colour.SDS_LIGHT_SOURCES['Neodimium Incandescent'])
array([ 36.94726204,  32.62076174,  13.0143849 ])
>>> sorted(colour.SPECTRAL_TO_XYZ_METHODS)
['ASTM E308', 'Integration', 'astm2015']
```

4.4.8.2 5.4.8.2 Multi-Spectral Computations

```
>>> msds = 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.msds_to_XYZ(msds, method='Integration',
...                    shape=colour.SpectralShape(400, 700, 60))
array([[[  7.68544647,   4.09414317,   8.49324254],
        [ 17.12567298,  27.77681821,  25.52573685],
        [ 19.10280411,  34.45851476,  29.76319628]],
       [[ 18.03375827,   8.62340812,   9.71702574],
        [ 15.03110867,   6.54001068,  24.53208465],
        [ 37.68269495,  26.4411103 ,  10.66361816]],
       [[  8.09532373,  12.75333339,  25.79613956],
        [  7.09620297,   2.79257389,  11.15039854],
        [  8.933163  ,  19.39985815,  17.14915636]],
       [[ 80.00969553,  80.39810464,  76.08184429],
        [ 33.27611427,  24.38947838,  39.34919287],
        [  8.89425686,  11.05185138,  10.86767594]]])
>>> sorted(colour.MSDS_TO_XYZ_METHODS)
['ASTM E308', 'Integration', 'astm2015']
```

4.4.8.3 5.4.8.3 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}
↪)
```

4.4.8.4 5.4.8.4 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]))
```

4.4.8.5 5.4.8.5 Lightness Computation

```
>>> colour.lightness(12.19722535)
41.527875844653451
>>> sorted(colour.LIGHTNESS_METHODS)
['CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Glasser 1958',
 'Lstar1976',
 'Wyszecki 1963']
```

4.4.8.6 5.4.8.6 Luminance Computation

```
>>> colour.luminance(41.52787585)
12.197225353400775
>>> sorted(colour.LUMINANCE_METHODS)
['ASTM D1535',
 'CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Newhall 1943',
 'astm2008',
 'cie1976']
```

4.4.8.7 5.4.8.7 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)
['ASTM E313',
 'Berger 1959',
 'CIE 2004',
 'Ganz 1979',
 'Stensby 1968',
 'Taube 1960',
 'cie2004']
```

4.4.8.8 5.4.8.8 Yellowness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> colour.yellowness(XYZ)
11.065000000000003
>>> sorted(colour.YELLOWNESS_METHODS)
['ASTM D1925', 'ASTM E313']
```

4.4.8.9 5.4.8.9 Luminous Flux, Efficiency & Efficacy Computation

```
>>> sd = colour.SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> colour.luminous_flux(sd)
23807.655527367202
>>> sd = colour.SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> colour.luminous_efficiency(sd)
0.19943935624521045
>>> sd = colour.SDS_LIGHT_SOURCES['Neodimium Incandescent']
>>> colour.luminous_efficacy(sd)
136.21708031547874
```

4.4.9 5.4.9 Contrast Sensitivity Function - `colour.contrast`

```
>>> colour.contrast_sensitivity_function(u=4, X_0=60, E=65)
358.51180789884984
>>> sorted(colour.CONTRAST_SENSITIVITY_METHODS)
['Barten 1999']
```

4.4.10 5.4.10 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)
['CAM02-LCD',
 'CAM02-SCD',
 'CAM02-UCS',
 'CAM16-LCD',
 'CAM16-SCD',
 'CAM16-UCS',
 'CIE 1976',
 'CIE 1994',
 'CIE 2000',
 'CMC',
 'DIN99',
 'cie1976',
 'cie1994',
 'cie2000']
```

4.4.11 5.4.11 IO - `colour.io`

4.4.11.1 5.4.11.1 Images

```
>>> RGB = colour.read_image('Ishihara_Colour_Blindness_Test_Plate_3.png')
>>> RGB.shape
(276, 281, 3)
```

4.4.11.2 5.4.11.2 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])
```

4.4.12 5.4.12 Colour Models - `colour.models`

4.4.12.1 5.4.12.1 CIE xyY Colourspace

```
>>> colour.XYZ_to_xyY([0.20654008, 0.12197225, 0.05136952])
array([ 0.54369557,  0.32107944,  0.12197225])
```

4.4.12.2 5.4.12.2 CIE L*a*b* Colourspace

```
>>> colour.XYZ_to_Lab([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  52.63858304,  26.92317922])
```

4.4.12.3 5.4.12.3 CIE L*u*v* Colourspace

```
>>> colour.XYZ_to_Luv([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  96.83626054,  17.75210149])
```

4.4.12.4 5.4.12.4 CIE 1960 UCS Colourspace

```
>>> colour.XYZ_to_UCS([0.20654008, 0.12197225, 0.05136952])
array([ 0.13769339,  0.12197225,  0.1053731 ])
```

4.4.12.5 5.4.12.5 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])
```

4.4.12.6 5.4.12.6 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])
```

4.4.12.7 5.4.12.7 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])
```

4.4.12.8 5.4.12.8 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.VIEWING_CONDITIONS_CIECAM02['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 ])
```

4.4.12.9 5.4.12.9 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.VIEWING_CONDITIONS_CAM16['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])
```

4.4.12.10 5.4.12.10 IGPGTG Colourspace

```
>>> colour.XYZ_to_IGPGTG([0.20654008, 0.12197225, 0.05136952])
array([ 0.42421258,  0.18632491,  0.10689223])
```

4.4.12.11 5.4.12.11 IPT Colourspace

```
>>> colour.XYZ_to_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 0.38426191,  0.38487306,  0.18886838])
```

4.4.12.12 5.4.12.12 DIN99 Colourspace

```
>>> Lab = [41.52787529, 52.63858304, 26.92317922]
>>> colour.Lab_to_DIN99(Lab)
array([ 53.22821988,  28.41634656,   3.89839552])
```

4.4.12.13 5.4.12.13 hdr-CIELAB Colourspace

```
>>> colour.XYZ_to_hdr_CIELab([0.20654008, 0.12197225, 0.05136952])
array([ 51.87002062,  60.4763385 ,  32.14551912])
```

4.4.12.14 5.4.12.14 hdr-IPT Colourspace

```
>>> colour.XYZ_to_hdr_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 25.18261761, -22.62111297,   3.18511729])
```

4.4.12.15 5.4.12.15 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])
```

4.4.12.16 5.4.12.16 JzAzBz Colourspace

```
>>> colour.XYZ_to_JzAzBz([0.20654008, 0.12197225, 0.05136952])
array([ 0.00535048,  0.00924302,  0.00526007])
```

4.4.12.17 5.4.12.17 Y'CbCr Colour Encoding

```
>>> colour.RGB_to_YCbCr([1.0, 1.0, 1.0])
array([ 0.92156863,  0.50196078,  0.50196078])
```

4.4.12.18 5.4.12.18 YCoCg Colour Encoding

```
>>> colour.RGB_to_YCoCg([0.75, 0.75, 0.0])
array([ 0.5625,  0.375 ,  0.1875])
```

4.4.12.19 5.4.12.19 ICTCP Colour Encoding

```
>>> colour.RGB_to_ICTCP([0.45620519, 0.03081071, 0.04091952])
array([ 0.07351364,  0.00475253,  0.09351596])
```

4.4.12.20 5.4.12.20 HSV Colourspace

```
>>> colour.RGB_to_HSV([0.45620519, 0.03081071, 0.04091952])
array([ 0.99603944,  0.93246304,  0.45620519])
```

4.4.12.21 5.4.12.21 Prismatic Colourspace

```
>>> colour.RGB_to_Prismatic([0.25, 0.50, 0.75])
array([ 0.75      ,  0.16666667,  0.33333333,  0.5       ])
```

4.4.12.22 5.4.12.22 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'
>>> matrix_XYZ_to_RGB = [
        [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,
        matrix_XYZ_to_RGB,
        chromatic_adaptation_transform)
array([ 0.45595571,  0.03039702,  0.04087245])
```

4.4.12.23 5.4.12.23 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]])
```

4.4.12.24 5.4.12.24 RGB Colourspaces

```
>>> sorted(colour.RGB_COLOURSPACES)
['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',
 'DaVinci Wide Gamut',
 'DCDM XYZ',
 'DCI-P3',
 'DCI-P3+',
 'DJI D-Gamut',
 'DRAGONcolor',
 'DRAGONcolor2',
 'Display P3',
 'Don RGB 4',
 'ECI RGB v2',
 'ERIMM RGB',
 'Ekta Space PS 5',
 'F-Gamut',
 'FilmLight E-Gamut',
 'ITU-R BT.2020',
 'ITU-R BT.470 - 525',
 'ITU-R BT.470 - 625',
 'ITU-R BT.709',
 'Max RGB',
 'NTSC (1953)',
 'NTSC (1987)',
 'P3-D65',
 'Pal/Secam',
 'ProPhoto RGB',
 'Protune Native',
 'REDWideGamutRGB',
 'REDcolor',
 'REDcolor2',
 'REDcolor3',
 'REDcolor4',
 'RIMM RGB',
 'ROMM RGB',
 'Russell RGB',
 'S-Gamut',
 'S-Gamut3',
 'S-Gamut3.Cine',
 'SMPTE 240M',
 'SMPTE C',
 'Sharp RGB',
 'V-Gamut',
```

(continues on next page)

(continued from previous page)

```
 'Venice S-Gamut3',
 'Venice S-Gamut3.Cine',
 'Xtreme RGB',
 'aces',
 'adobe1998',
 'prophoto',
```

4.4.12.25 5.4.12.25 OETFs

```
>>> sorted(colour.OETFS)
['ARIB STD-B67',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'SMPTE 240M']
```

4.4.12.26 5.4.12.26 OETFs Inverse

```
>>> sorted(colour.OETF_INVERSES)
['ARIB STD-B67',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709']
```

4.4.12.27 5.4.12.27 EOTFs

```
>>> sorted(colour.EOTFS)
['DCDM',
 'DICOM GSDF',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'SMPTE 240M',
 'ST 2084',
 'sRGB']
```

4.4.12.28 5.4.12.28 EOTFs Inverse

```
>>> sorted(colour.EOTF_INVERSES)
['DCDM',
 'DICOM GSDF',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ST 2084',
 'sRGB']
```

4.4.12.29 5.4.12.29 OOTFs

```
>>> sorted(colour.OOTFS)
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

4.4.12.30 5.4.12.30 OOTFs Inverse

```
>>> sorted(colour.OOTF_INVERSES)
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

4.4.12.31 5.4.12.31 Log Encoding / Decoding

```
>>> sorted(colour.LOG_ENCODINGS)
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'ERIMM RGB',
 'F-Log',
 'Filmic Pro 6',
 'Log2',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'T-Log',
 'V-Log',
 'ViperLog']
```

4.4.12.32 5.4.12.32 CCTFs Encoding / Decoding

```
>>> sorted(colour.CCTF_ENCODINGS)
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'ARIB STD-B67',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
```

(continues on next page)

(continued from previous page)

```
 'D-Log',
 'DCDM',
 'DICOM GSDF',
 'ERIMM RGB',
 'F-Log',
 'Filmic Pro 6',
 'Gamma 2.2',
 'Gamma 2.4',
 'Gamma 2.6',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'Log2',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'ProPhoto RGB',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'RIMM RGB',
 'ROMM RGB',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'SMPTE 240M',
 'ST 2084',
 'T-Log',
 'V-Log',
 'ViperLog',
 'sRGB']
```

4.4.13 5.4.13 Colour Notation Systems - `colour.notation`

4.4.13.1 5.4.13.1 Munsell Value

```
>>> colour.munsell_value(12.23634268)
4.0824437076525664
>>> sorted(colour.MUNSELL_VALUE_METHODS)
['ASTM D1535',
 'Ladd 1955',
 'McCamy 1987',
 'Moon 1943',
 'Munsell 1933',
 'Priest 1920',
 'Saunderson 1944',
 'astm2008']
```

4.4.13.2 5.4.13.2 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   ])
```

4.4.14 5.4.14 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}
↪)
```

4.4.15 5.4.15 Light Quality - `colour.quality`

4.4.15.1 5.4.15.1 Colour Fidelity Index

```
>>> colour.colour_fidelity_index(colour.SDS_ILLUMINANTS['FL2'])
70.120825477833037
>>> sorted(colour.COLOUR_FIDELITY_INDEX_METHODS)
['ANSI/IES TM-30-18', 'CIE 2017']
```

4.4.15.2 5.4.15.2 Colour Rendering Index

```
>>> colour.colour_quality_scale(colour.SDS_ILLUMINANTS['FL2'])
64.111703163816699
>>> sorted(colour.COLOUR_QUALITY_SCALE_METHODS)
['NIST CQS 7.4', 'NIST CQS 9.0']
```

4.4.15.3 5.4.15.3 Colour Quality Scale

```
>>> colour.colour_rendering_index(colour.SDS_ILLUMINANTS['FL2'])
64.233724121664807
```

4.4.15.4 5.4.15.4 Academy Spectral Similarity Index (SSI)

```
>>> colour.spectral_similarity_index(colour.SDS_ILLUMINANTS['C'], colour.SDS_ILLUMINANTS[
↪'D65'])
94.0
```

4.4.16 5.4.16 Spectral Up-Sampling & Reflectance Recovery - colour.recovery

```
>>> colour.XYZ_to_sd([0.20654008, 0.12197225, 0.05136952])
SpectralDistribution([[  3.60000000e+02,   8.37868873e-02],
                      [  3.65000000e+02,   8.39337988e-02],
                      ...
                      [  7.70000000e+02,   4.46793405e-01],
                      [  7.75000000e+02,   4.46872853e-01],
                      [  7.80000000e+02,   4.46914431e-01]],
                     interpolator=SpragueInterpolator,
                     interpolator_kwargs={},
                     extrapolator=Extrapolator,
                     extrapolator_kwargs={'method': 'Constant', 'left': None, 'right':␣
↪None})

>>> sorted(colour.REFLECTANCE_RECOVERY_METHODS)
['Jakob 2019', 'Mallett 2019', 'Meng 2015', 'Otsu 2018', 'Smits 1999']
```

4.4.17 5.4.17 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)
['Krystek 1985', 'Ohno 2013', 'Robertson 1968', 'ohno2013', 'robertson1968']
>>> sorted(colour.XY_TO_CCT_METHODS)
['CIE Illuminant D Series',
 'Hernandez 1999',
 'Kang 2002',
 'McCamy 1992',
 'daylight',
 'hernandez1999',
 'kang2002',
 'mccamy1992']
```

4.4.18 5.4.18 Colour Volume - colour.volume

```
>>> colour.RGB_colourspace_volume_MonteCarlo(colour.RGB_COLOURSPACE_RGB['sRGB'])
821958.30000000005
```

4.4.19 5.4.19 Geometry Primitives Generation - `colour.geometry`

```
>>> colour.primitive('Grid')
(array([ ([-0.5,  0.5,  0. ], [ 0.,  1.], [ 0.,  0.,  1.], [ 0.,  1.,  0.,  1.]),
       ([ 0.5,  0.5,  0. ], [ 1.,  1.], [ 0.,  0.,  1.], [ 1.,  1.,  0.,  1.]),
       ([-0.5, -0.5,  0. ], [ 0.,  0.], [ 0.,  0.,  1.], [ 0.,  0.,  0.,  1.]),
       ([ 0.5, -0.5,  0. ], [ 1.,  0.], [ 0.,  0.,  1.], [ 1.,  0.,  0.,  1.])],
      dtype=[('position', '<f4', (3,)), ('uv', '<f4', (2,)), ('normal', '<f4', (3,)), (
↪'colour', '<f4', (4,))]), array([[0, 2, 1],
       [2, 3, 1]], dtype=uint32), array([[0, 2],
       [2, 3],
       [3, 1],
       [1, 0]], dtype=uint32))
>>> sorted(colour.PRIMITIVE_METHODS)
['Cube', 'Grid']
>>> colour.primitive_vertices('Quad MPL')
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  0.],
       [ 0.,  1.,  0.]])
>>> sorted(colour.PRIMITIVE_VERTICES_METHODS)
['Cube MPL', 'Grid MPL', 'Quad MPL', 'Sphere']
```

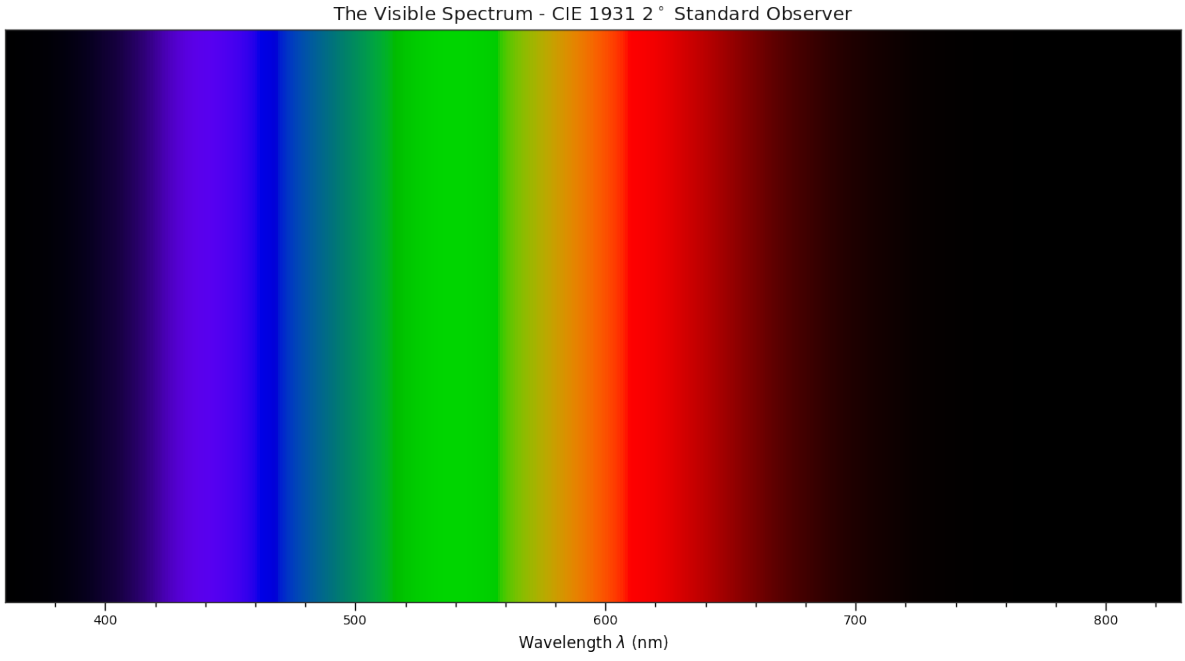
4.4.20 5.4.20 Plotting - `colour.plotting`

Most of the objects are available from the `colour.plotting` namespace:

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

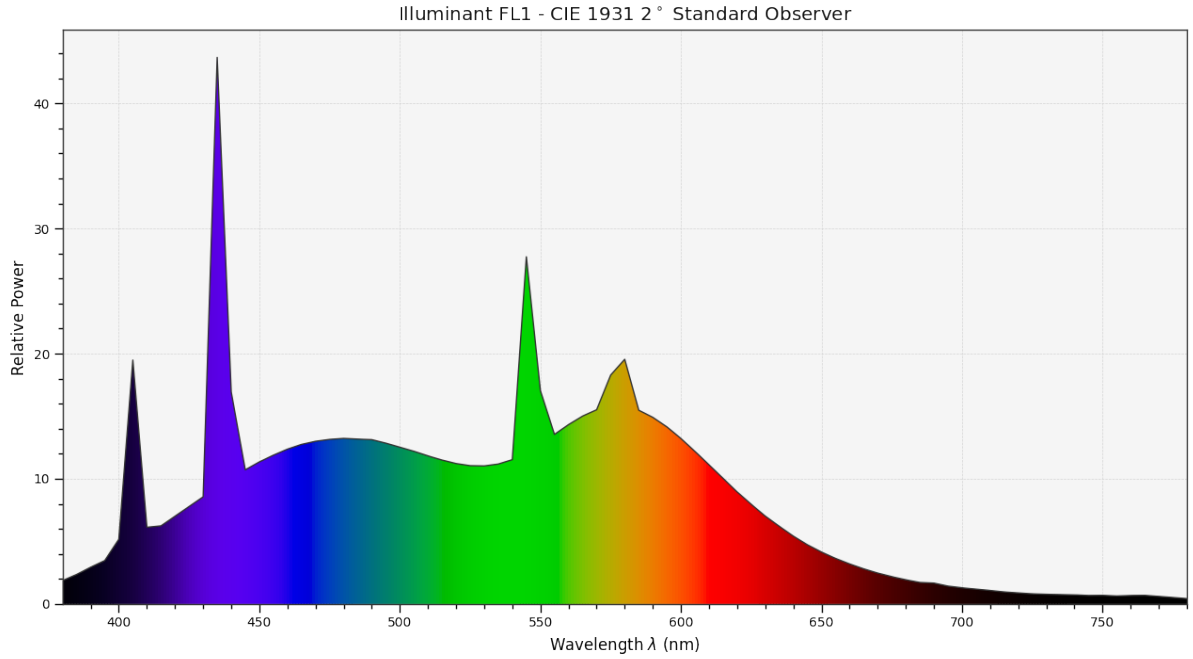
4.4.20.1 5.4.20.1 Visible Spectrum

```
>>> plot_visible_spectrum('CIE 1931 2 Degree Standard Observer')
```



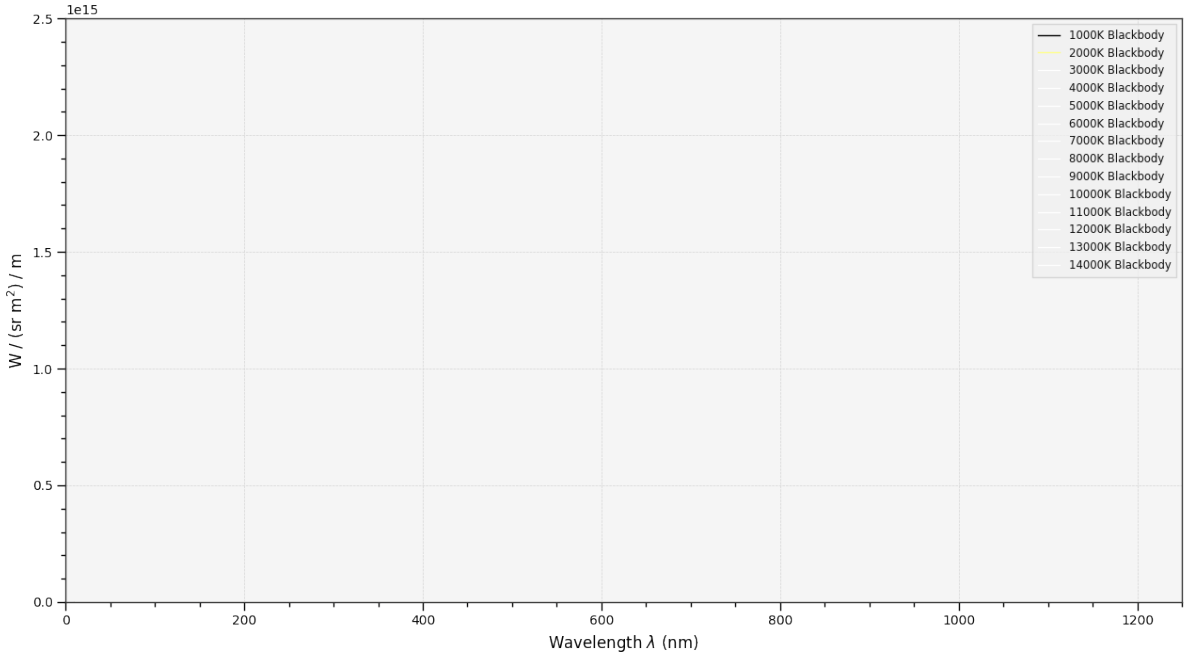
4.4.20.2 5.4.20.2 Spectral Distribution

```
>>> plot_single_illuminant_sd('FL1')
```



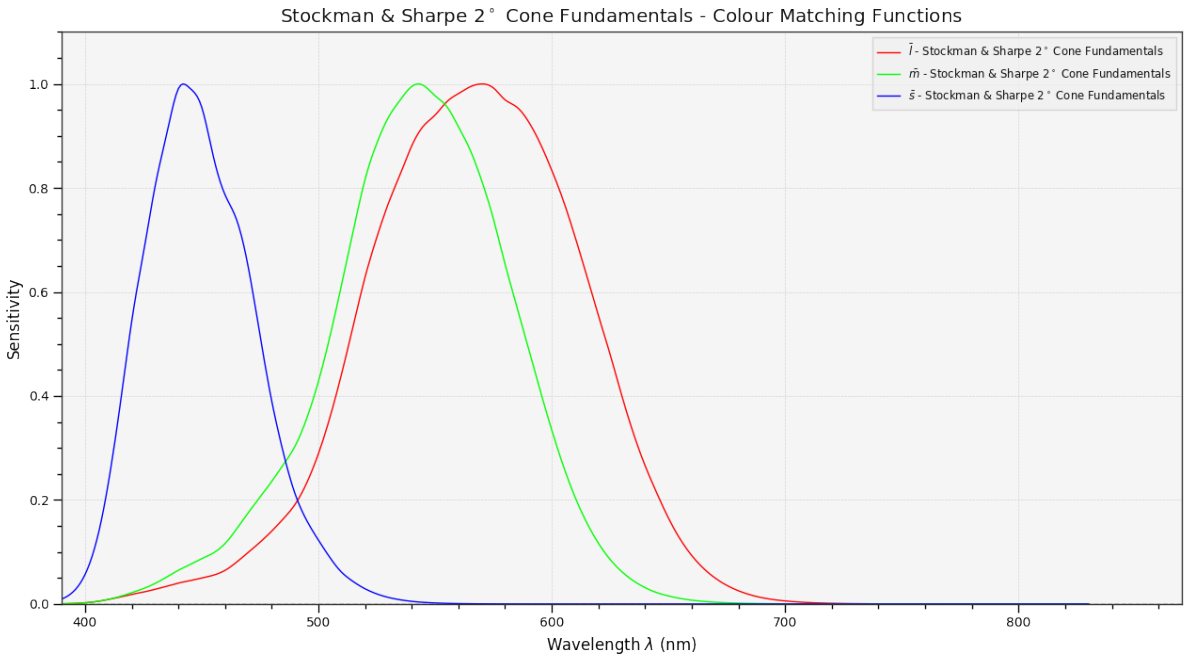
4.4.20.3 5.4.20.3 Blackbody

```
>>> 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',
...     plot_kwargs={
...         use_sd_colours=True,
...         normalise_sd_colours=True,
...     },
...     legend_location='upper right',
...     bounding_box=(0, 1250, 0, 2.5e15))
```



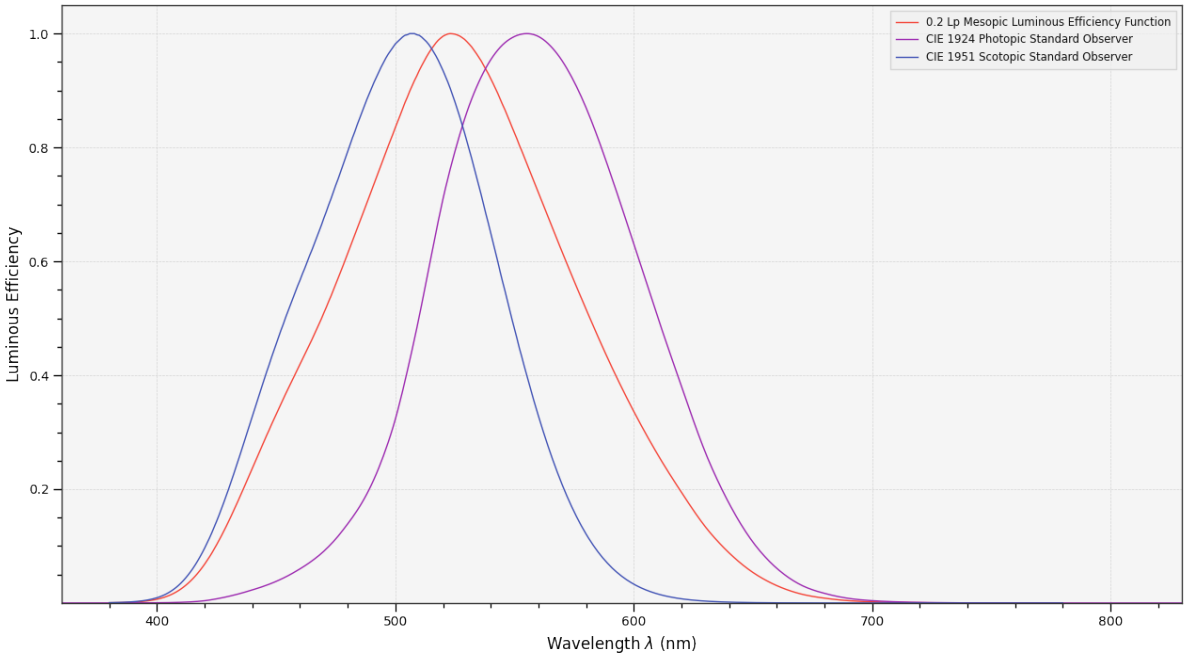
4.4.20.4 5.4.20.4 Colour Matching Functions

```
>>> plot_single_cmfs(
...     'Stockman & Sharpe 2 Degree Cone Fundamentals',
...     y_label='Sensitivity',
...     bounding_box=(390, 870, 0, 1.1))
```



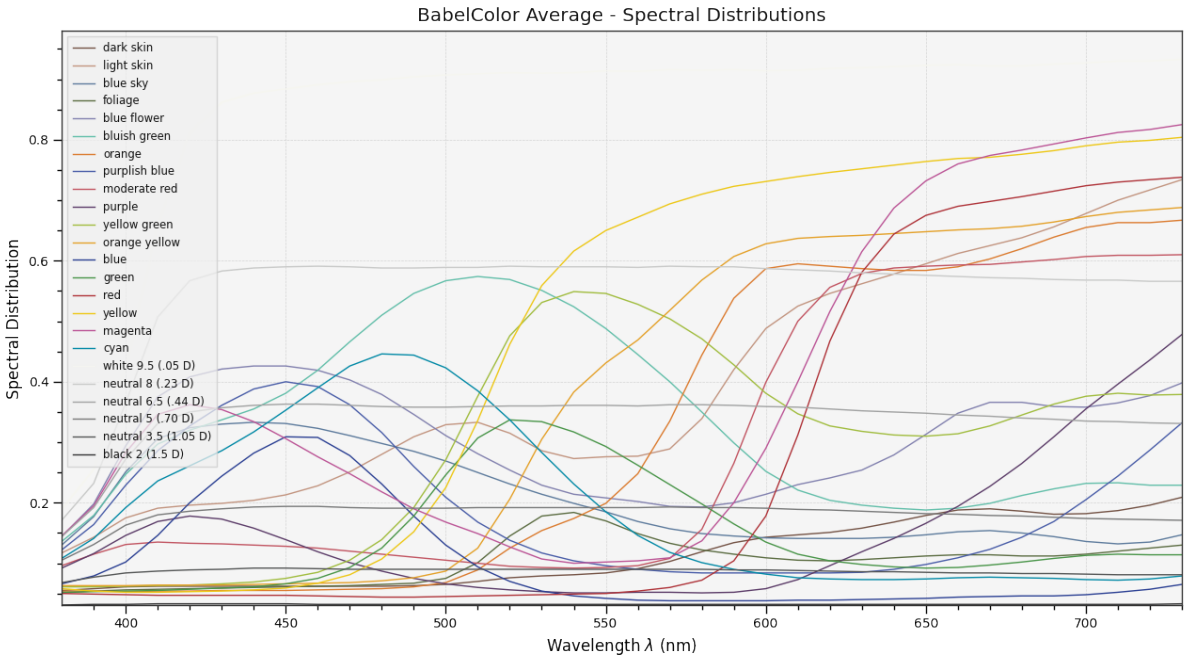
4.4.20.5 5.4.20.5 Luminous Efficiency

```
>>> 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))
```



4.4.20.6 5.4.20.6 Colour Checker

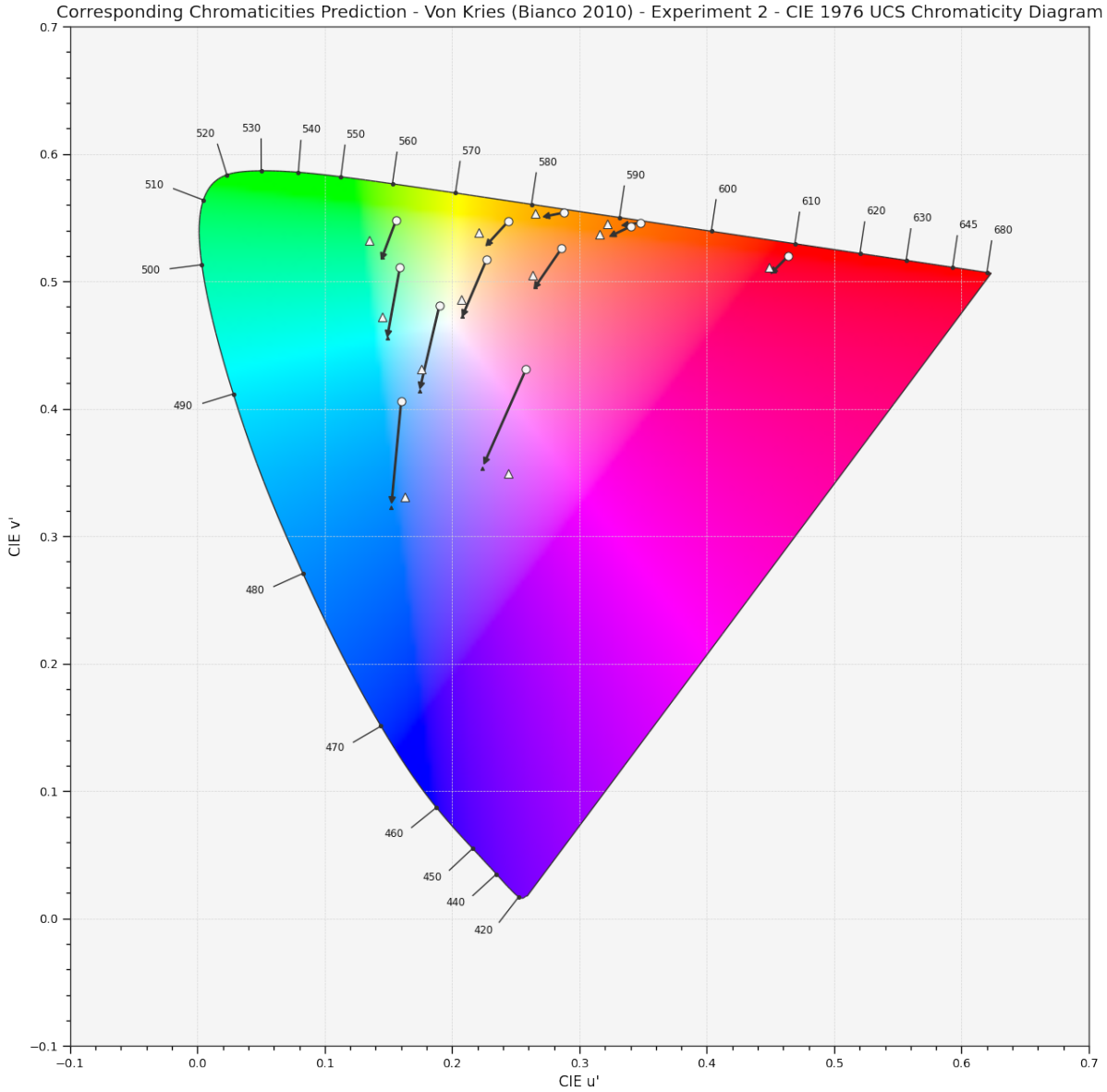
```
>>> from colour.characterisation.dataset.colour_checkers.sds import (
...     COLOURCHECKER_INDEXES_TO_NAMES_MAPPING)
>>> plot_multi_sds(
...     [
...         colour.SDS_COLOURCHECKERS['BabelColor Average'][value]
...         for key, value in sorted(
...             COLOURCHECKER_INDEXES_TO_NAMES_MAPPING.items())
...     ],
...     plot_kwargs={
...         use_sd_colours=True,
...     },
...     title=('BabelColor Average - '
...            'Spectral Distributions'))
```




```
>>> plot_single_colour_checker(
...     'ColorChecker 2005', text_kwargs={'visible': False})
```

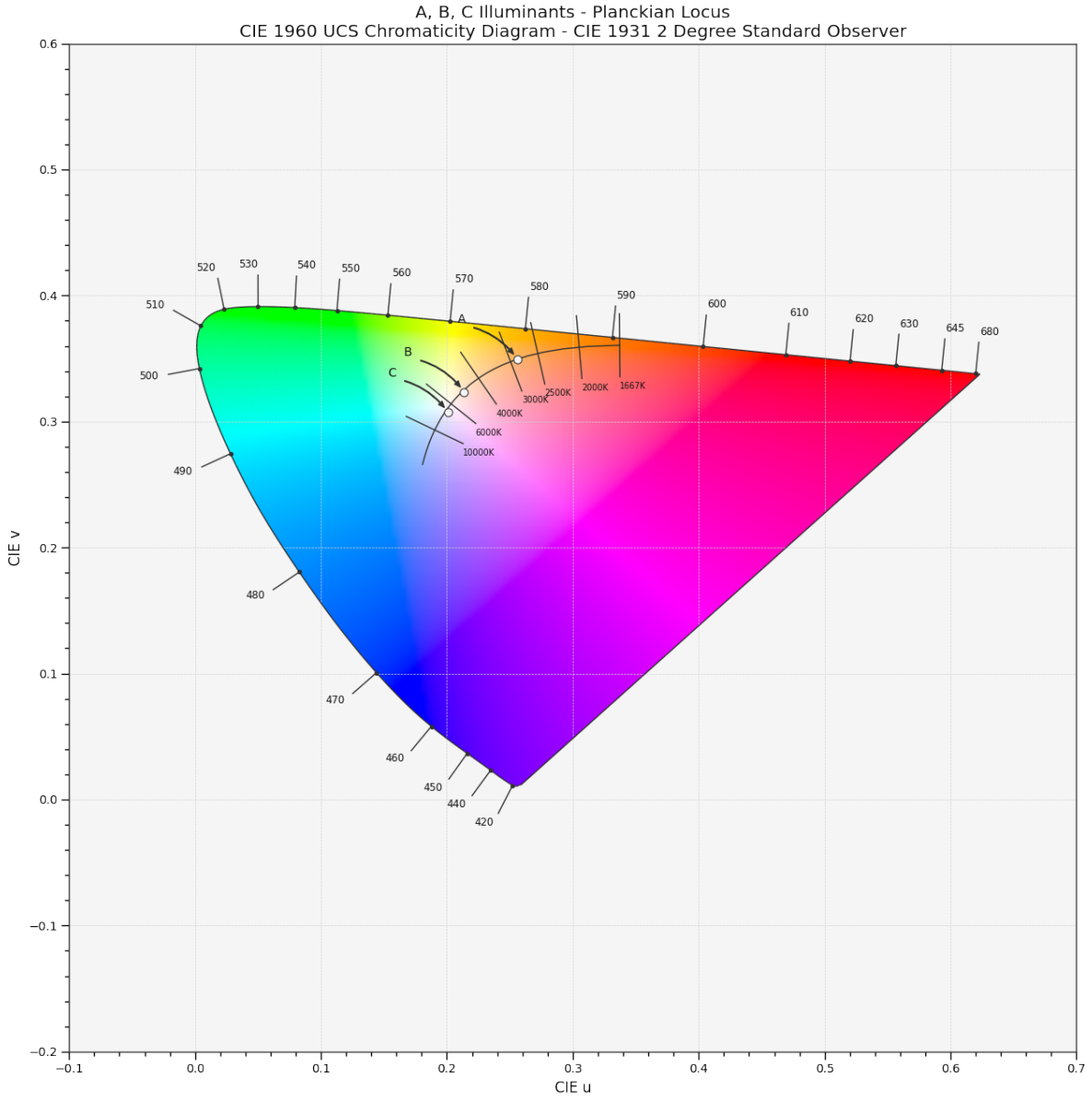
4.4.20.7 5.4.20.7 Chromaticities Prediction

```
>>> plot_corresponding_chromaticities_prediction(
...     2, 'Von Kries', 'Bianco 2010')
```



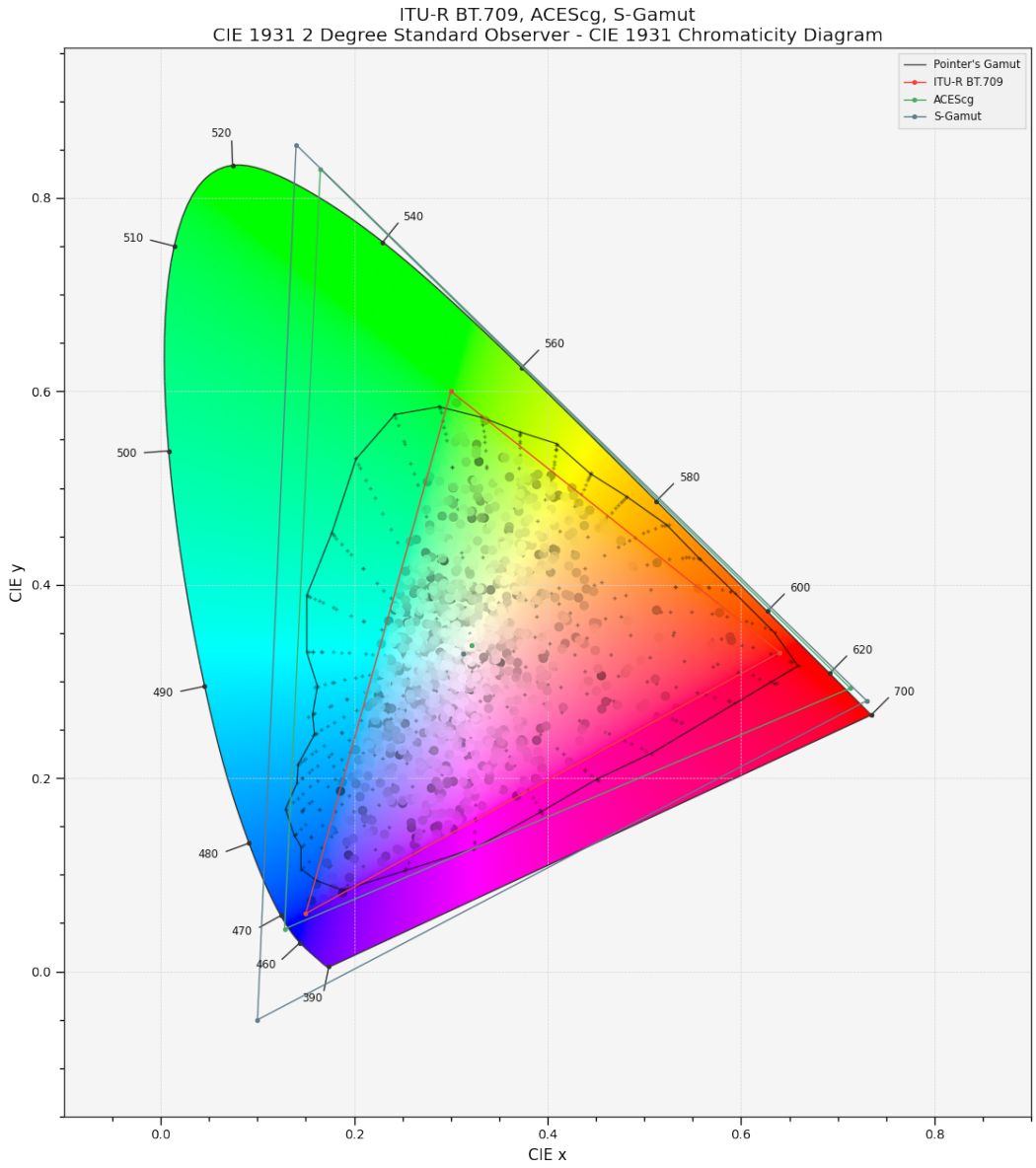
4.4.20.8 5.4.20.8 Colour Temperature

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(['A', 'B', 'C'])
```



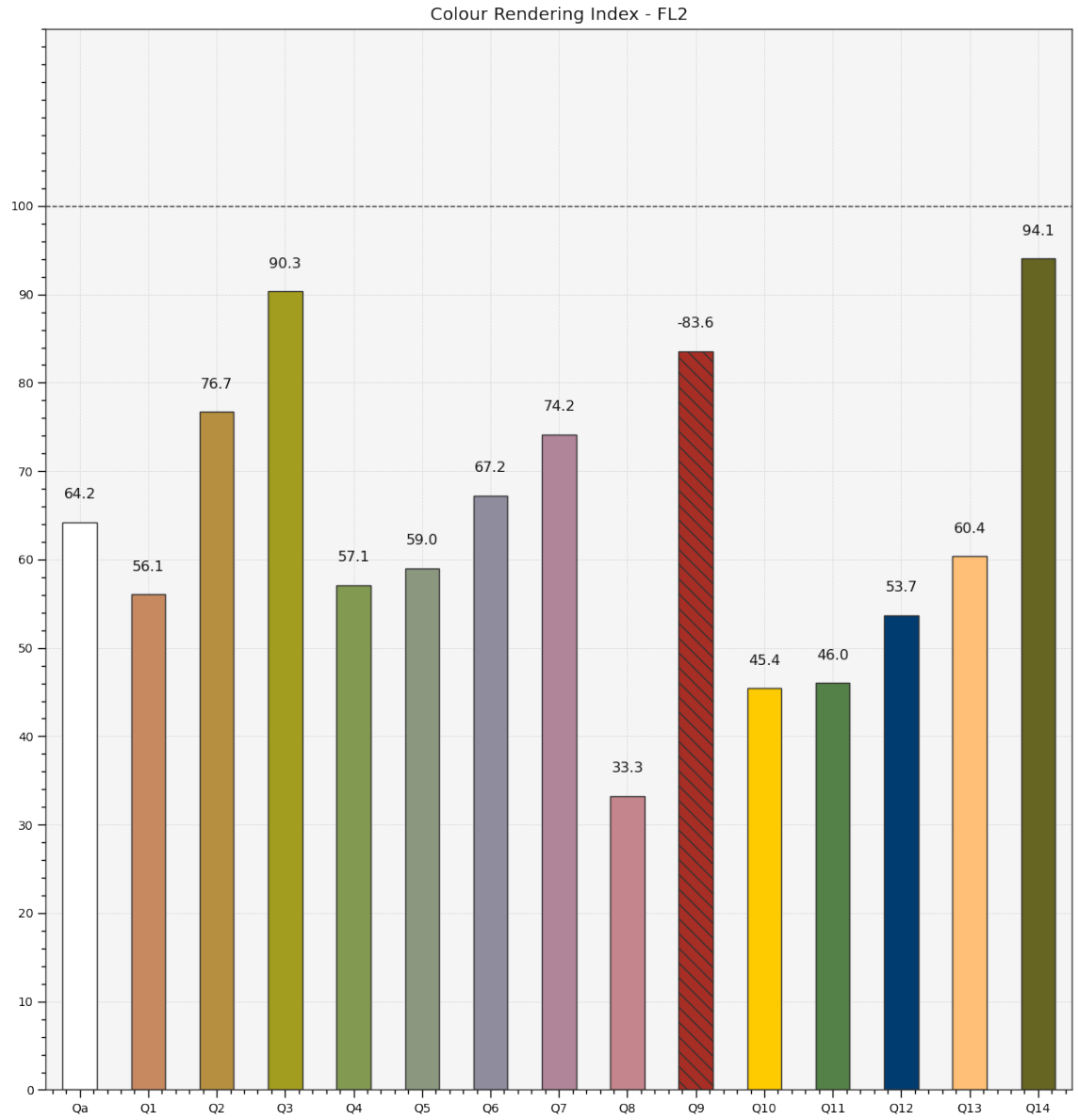
4.4.20.9 5.4.20.9 Chromaticities

```
>>> 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)
```



4.4.20.10 5.4.20.10 Colour Rendering Index

```
>>> plot_single_sd_colour_rendering_index_bars(
...     colour.SDS_ILLUMINANTS['FL2'])
```



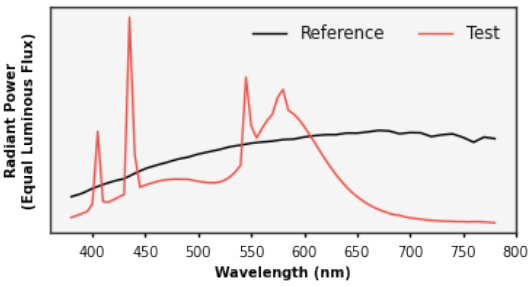
4.4.20.11 5.4.20.11 ANSI/IES TM-30-18 Colour Rendition Report

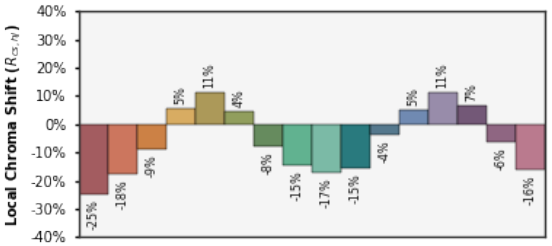
```
>>> plot_single_sd_colour_rendition_report(
...     colour.SDS_ILLUMINANTS['FL2'])
```

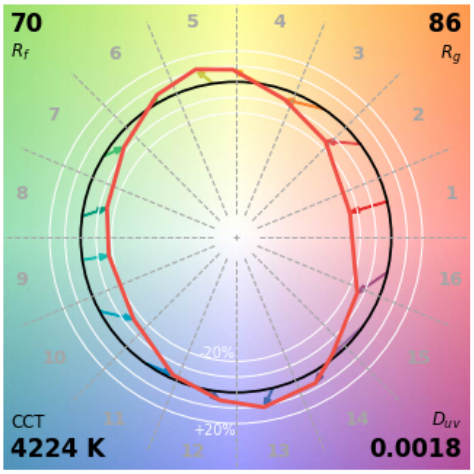
IES TM-30-18 Colour Rendition Report

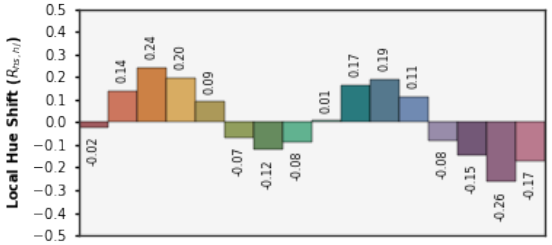
Source: FL2
Date: N/A

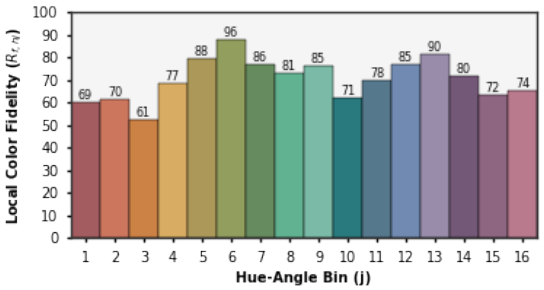
Manufacturer: N/A
Model: N/A

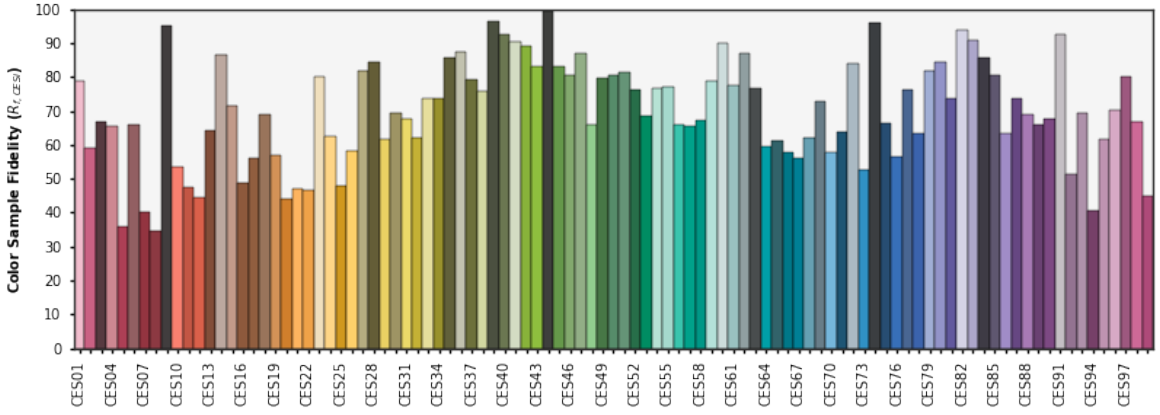












Notes: N/A

x **0.3721**
y **0.3751**
u' **0.2202**
v' **0.4996**

CIE 13.31-1995
(CRI)
R_a **64**
R_9 **33**

Colours are for visual orientation purposes only. Created with Colour.

CHAPTER

FIVE

6 CONTRIBUTING

If you would like to contribute to **Colour**, please refer to the following Contributing guide.

CHAPTER

SIX

7 CHANGES

The changes are viewable on the Releases page.

CHAPTER

SEVEN

8 BIBLIOGRAPHY

The bibliography is available on the Bibliography page.

It is also viewable directly from the repository in BibTeX format.

CHAPTER

EIGHT

9 SEE ALSO

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.

Go

- go-colorful by Beyer, L., 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

NINE

10 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

TEN

11 ABOUT

Colour by Colour Developers

https://github.com/colour-science/colour

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z